TECHNISCHE UNIVERSITÄT DRESDEN

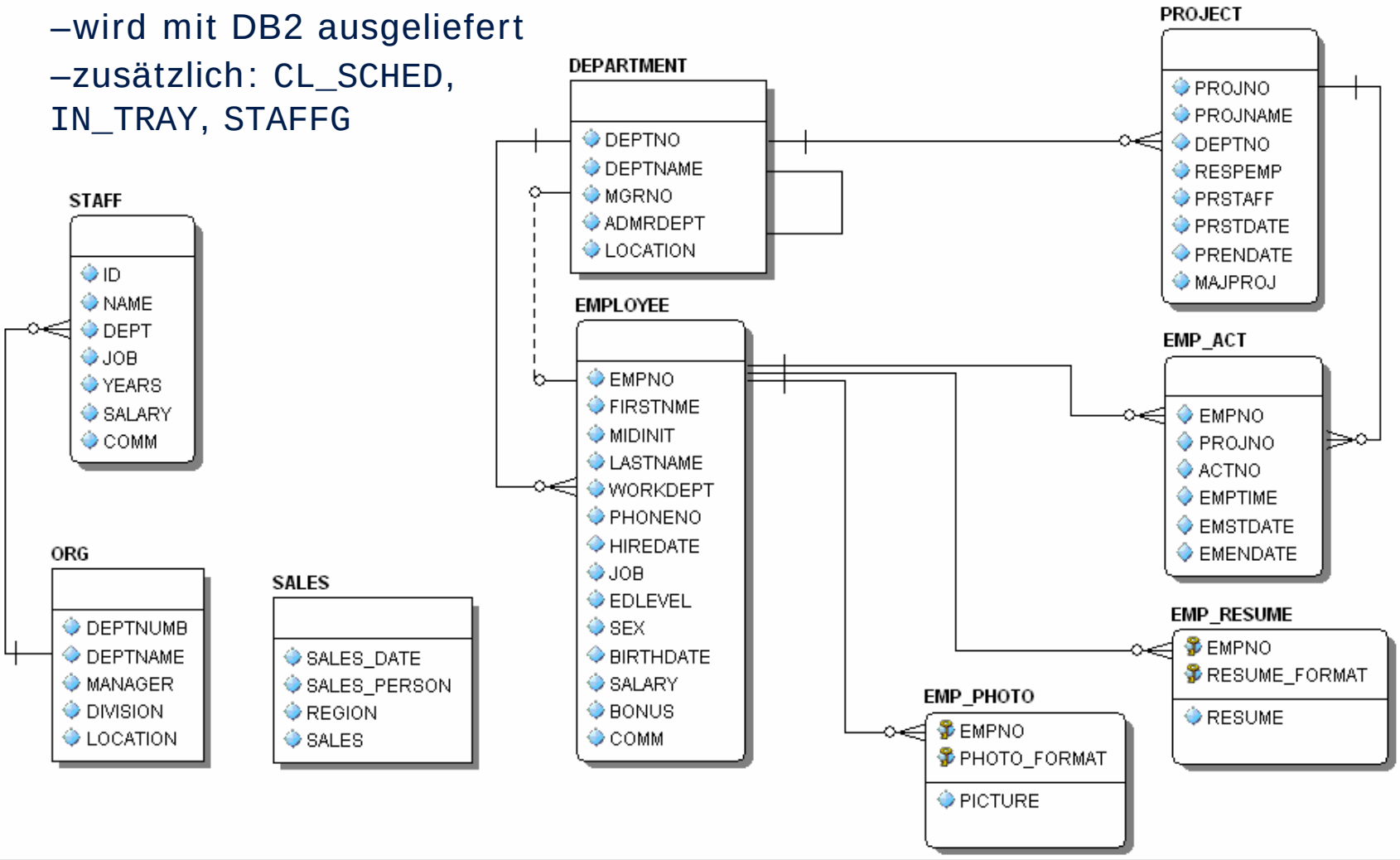
Fakultät Informatik, Institut Systemarchitektur, Professur Datenbanken

Datenbankprogrammierung
1. SQL Grundlagen

Beispieldatenbanken

DB2-Beispieldatenbank

- **DB2-Beispieldatenbank (SAMPLE)**
  - wird mit DB2 ausgeliefert
  - zusätzlich: CL_SCHED, IN_TRAY, STAFFG

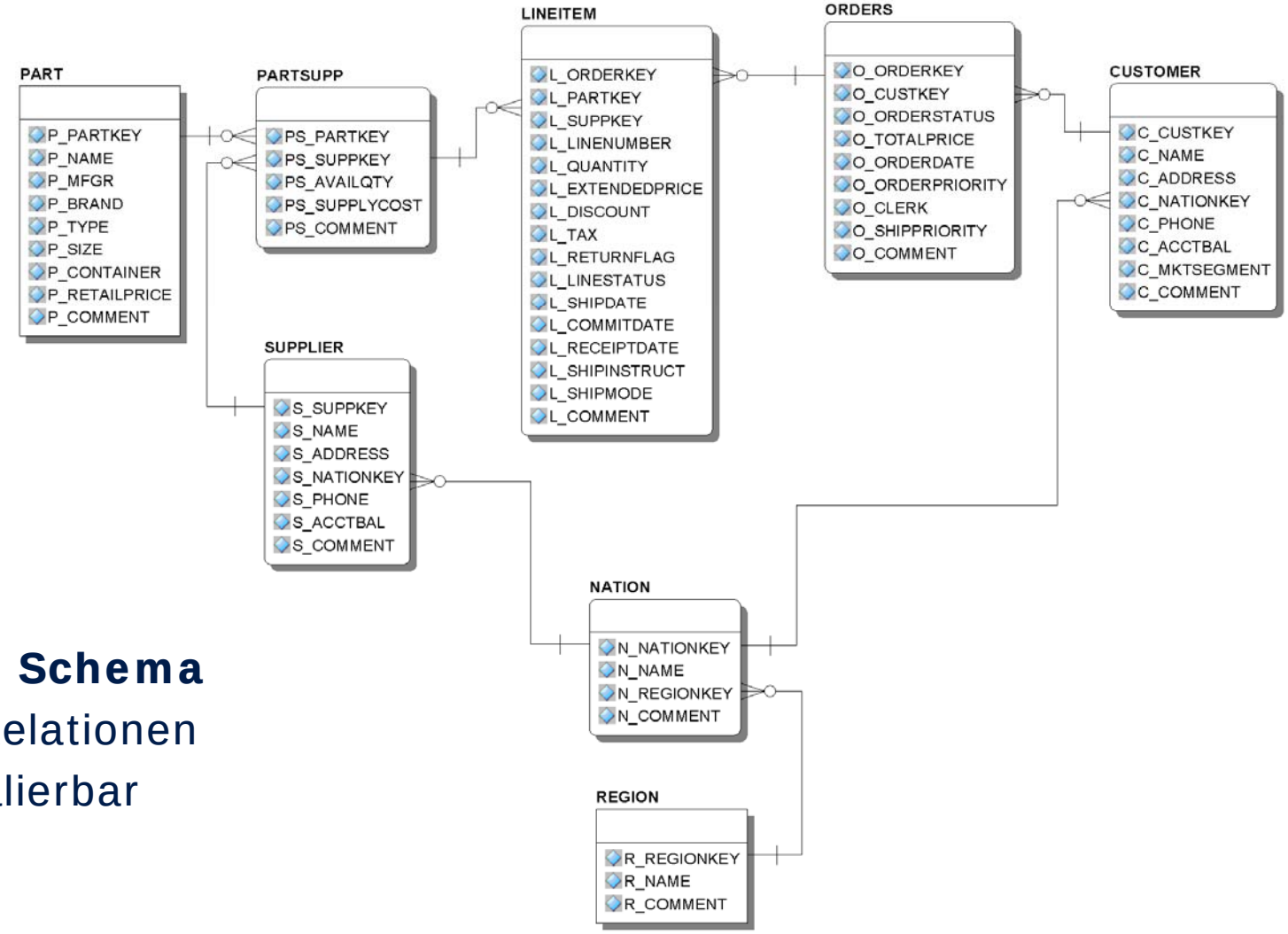


- **Transaction Processing Performance Council (TPC)**
  - Web: http://www.tpc.org
  - Geschwindigkeitsvergleich mit standardisierten Daten und Anfragen
  - TPC-H: Decision Support Benchmark

100 GB Results

Rank	Company	System	QphH	Price/ QphH	System Availability	Database	Operating System	Date Submitted	Cluster
1	hp invent	HP ProLiant DL585G2 4P	19,323	10.67 US $	01/16/07	Microsoft SQL Server 2005 x64 Enterprise Edt. SP1	Microsoft Windows Server 2003 Enterprise x64 Edition SP1	09/25/06	N
2	DELL	PowerEdge 6950/2.8GHz/ 2MB	17,179	10.02 US $	12/04/06	Microsoft SQL Server 2005 Enterprise x64 Edition	Microsoft Windows Server 2003 Enterprise x64 Edition	10/23/06	N
3	hp invent	HP ProLiant DL580G4 4P	17,120	7.91 US $	11/22/06	Microsoft SQL Server 2005 x64 Enterprise Edt. SP1	Microsoft Windows Server 2003 Enterprise x64 Edition SP1	09/05/06	N
4	DELL	PowerEdge 6800/3.4GHz/ 16MB	16,320	13.40 US $	08/28/06	Microsoft SQL Server 2005 Enterprise Edt (x64)	Microsoft Windows Server 2003 Enterprise x64 Edition	08/01/06	N
5	DELL	PowerEdge 2900/2.66GHz/ 8MB	15,723	9.61 US $	12/31/06	Microsoft SQL Server 2005 x64 Enterprise Edt. SP1	Microsoft Windows Server 2003 Enterprise x64 Edition SP1	11/27/06	N
6	DELL	PowerEdge 6950/2.8GHz/ 2MB	14,923	6.04 US $	12/04/06	Microsoft SQL Server 2005 Enterprise x64 Edition	Microsoft Windows Server 2003 Enterprise x64 Edition	10/23/06	N
7	hp invent	HP ProLiant ML570G4 4P	14,242	10.88 US $	11/22/06	Microsoft SQL Server 2005 Enterprise x64 Edition	Microsoft Windows Server 2003 Enterprise x64 Edition SP1	05/22/06	N
8	hp invent	HP ProLiant DL585 G1 4P	12,600	7.67 US $	11/07/05	Microsoft SQL Server 2005 Enterprise x64 Edition	Microsoft Windows Server 2003 Enterprise x64 Edition	11/04/05	N
9	IBM	IBM eServer 325	12,216	70.68 US $	11/08/03	IBM DB2 UDB 8.1	Suse Linux Enterprise Server 8	07/29/03	Y
10	DELL	PowerEdge PE6800/3.0/ 64GB	11,529	11.18 US $	05/01/06	Microsoft SQL Server 2005 Enterprise x64 Edition	Microsoft Windows Server 2003 Enterprise x64 Edition	05/01/06	N

TPC-H



- **TPC-H Schema**
  - 8 Relationen
  - skalierbar

- **GeoDB-Schema**
  - Ortsinformationen (geographische Lage, Zugehörigkeit zu Bundesland usw.)

SQL Grundlagen

- **SQL (Structured Query Language)**
  - deskriptive Anfragesprache
  - angelehnt an die englische Umgangssprache

- **Enthält**
  - DQL (*data query language*): Datenbankabfrage
  - DML (*data manipulation language*): Datenmanipulation
  - DDL (*data definition language*): Datendefinition
  - DCL (*data control language*): Rechteverwaltung

- **SQL-Standard**
  - SQL/86, SQL/89, SQL/92 (=SQL2), SQL/99, SQL:2003 (=SQL3)
  - Part 1: SQL/Framework (Aufbau des Standards)
  - ***Part 2: SQL/Foundation***
  - Part 3: SQL/CLI (*call level interface*)
  - Part 4: SQL/PSM (*persistent storage modules*)
  - ...

SELECT-Anweisung

- **SELECT-Anweisung**

```
SELECT [DISTINCT] <attribute-list>      → Projektion, Duplikatelim.
FROM <table-name> [AS <alias>], …       → Kartesisches Produkt
WHERE <predicate-list>                  → Selektion auf Tupelebene
GROUP BY <attribute-list>               → Gruppierung
HAVING <predicate-list>                 → Selektion auf Gruppenebene
ORDER BY <attribute> [ASC|DESC],…       → Sortierung
FETCH FIRST <k> ROWS ONLY               → Top-k (DB2-spezifisch!)
```

- **Beispiele für Selektionsbedingungen**
  - Mustersuche in Zeichenketten

```
WHERE R_NAME [NOT] LIKE 'A%'
```

  - Bereichssuche

```
WHERE R_REGIONKEY [NOT] BETWEEN 1 AND 3
```

  - **NULL**-Werte

```
WHERE R_COMMENT IS [NOT] NULL
```

SELECT-Anweisung

- **Beispiel (SAMPLE)**
  - Name und Gehalt der Mitarbeiter der Abteilung A00

```
SELECT FIRSTNME, LASTNAME, WORKDEPT, SALARY
FROM EMPLOYEE
WHERE WORKDEPT = 'A00'
```

  - Ergebnis

```
FIRSTNME     LASTNAME     WORKDEPT     SALARY
-----------  -----------  -----------  ---------
CHRISTINE    HAAS         A00          52750
VINCENZO     LUCCHESSI    A00          46500
SEAN         O'CONNELL    A00          29250
```

SELECT-Anweisung

- **Beispiel (SAMPLE)**
  - Durchschnittsgehalt pro Abteilung

```
SELECT DEPTNAME, AVG(SALARY) AS AVG_SALARY
FROM DEPARTMENT D, EMPLOYEE E
WHERE E.WORKDEPT = D.DEPTNO
GROUP BY DEPTNAME
```

Aggregationsfunktion
(z.B. **COUNT**,**SUM**,**MIN**,**MAX**)

  - Ergebnis

```
DEPTNAME                       AVG_SALARY
------------------------------ ----------
ADMINISTRATION SYSTEMS         25153,33
INFORMATION CENTER             30156,66
MANUFACTURING SYSTEMS          24677,77
OPERATIONS                     20998
PLANNING                       41250
SOFTWARE SUPPORT               23827,5
SPIFFY COMPUTER SERVICE DIV.   42833,33
SUPPORT SERVICES               40175
```

SELECT-Anweisung

- **Beispiel (TPCH)**
  - Gesamtumsatz nach Ländern, falls über 50 Millionen; sortiert

```
SELECT N_NAME, SUM(O_TOTALPRICE) AS TURNOVER
FROM ORDERS, CUSTOMER, NATION
WHERE O_CUSTKEY = C_CUSTKEY
  AND C_NATIONKEY = N_NATIONKEY
GROUP BY N_NAME
HAVING SUM(O_TOTALPRICE)>50000000
ORDER BY N_NAME
```

  - Ergebnis

```
N_NAME                    TURNOVER
------------------------- ------------
ARGENTINA                 129997977,11
EGYPT                     66482178,24
JORDAN                    273941626,19
MOROCCO                   1093739712,6
```

Verbundoperationen

- **Innerer Verbund (*inner join*)**
  - standardmäßige Verbundart
  - nur Tupel mit Verbundpartner werden in Ergebnis aufgenommen
  - Beispiel
    - ```
      SELECT MA.Name, ABT.Name, ABT.Chef
      FROM MITARBEITER MA, ABTEILUNG ABT
      WHERE MA.Abteilung = ABT.Name
      ```
    - ```
      MA.Name  ABT.Name      ABT.Chef
      -------  ------------  --------
      Paul     Verwaltung    Paul
      Dirk     Produktion    Dirk
      ```

MITARBEITER	
Name	**Abteilung**
Paul	Verwaltung
Fritz	**NULL**
Dirk	Produktion

ABTEILUNG	
Name	**Chef**
Verwaltung	Paul
Produktion	Dirk
IT	**NULL**

Verbundoperationen

- **Äußerer Verbund (*outer join*)**
  - auch Tupel ohne Verbundpartner werden aufgenommen
  - Attribute der anderen Relation werden mit **NULL**-Werten aufgefüllt

- **Linker äußerer Verbund (*left outer join*)**
  - alle Tupel der linken Relation erscheinen im Verbundergebnis
  - Beispiel
    - ```
      SELECT MA.Name, ABT.Name, ABT.Chef
      FROM MITARBEITER MA LEFT OUTER JOIN ABTEILUNG ABT
           ON MA.Abteilung = ABT.Name
      ```
    - ```
      MA.Name  ABT.Name      ABT.Chef
      -------  ------------  --------
      Paul     Verwaltung    Paul
      Dirk     Produktion    Dirk
      Fritz    -             -
      ```

Verbundoperationen

- **Rechter äußerer Verbund (*right outer join*)**
  - alle Tupel der rechten Relation erscheinen im Verbundergebnis
  - Beispiel
    - ```
      SELECT MA.Name, ABT.Name, ABT.Chef
      FROM MITARBEITER MA RIGHT OUTER JOIN ABTEILUNG ABT
           ON MA.Abteilung = ABT.Name
      ```
    - ```
      MA.Name  ABT.Name      ABT.Chef
      -------  ------------  --------
      Paul     Verwaltung    Paul
      Dirk     Produktion    Dirk
      -        IT            -
      ```

Verbundoperationen

- **Vollständiger äußerer Verbund (*full outer join*)**
  - alle Tupel erscheinen im Verbundergebnis

  - ```
    SELECT MA.Name, ABT.Name, ABT.Chef
    FROM MITARBEITER MA FULL OUTER JOIN ABTEILUNG ABT
         ON MA.Abteilung = ABT.Name
    ```

  - ```
    MA.Name  ABT.Name      ABT.Chef
    -------  ------------  --------
    Paul     Verwaltung    Paul
    Dirk     Produktion    Dirk
    Fritz    -             -
    -        IT            -
    ```

Mengenoperationen

- **Mengenoperationen**
  - auf Relationen mit gleichem Schema
  - Vereinigung (**UNION**), Subtraktion (**EXCEPT**), Schnittmenge (**INTERSECT**)
  - Mengensemantik (z.B. **UNION**) vs. Multimenge (**UNION ALL**)
  - Beispiel
    - ```
      SELECT R_NAME FROM REGION
      UNION
      SELECT 'ANTARTICA' FROM SYSIBM.SYSDUMMY1
      ```
    - ```
      1
      -----------
      AFRICA
      AMERICA
      ANTARTICA
      ASIA
      EUROPE
      MIDDLE EAST
      ```

Fallunterscheidung

- **CASE-Ausdruck**
  - Fallunterscheidung
  - in **SELECT**- oder **GROUP BY**-Klausel verwendet
  - Beispiel

```
SELECT P_NAME,
       CASE WHEN P_SIZE < 10 THEN 'SMALL'
            WHEN P_SIZE < 20 THEN 'NORMAL'
            ELSE 'BIG'
       END AS SIZE
FROM PART
```

```
P_NAME                                     SIZE
-----------------------------------------  -------
goldenrod                                  SMALL
frosted orange turquoise dim chocolate     NORMAL
royal lace plum spring coral               BIG
…
```

  - Achtung: Reihenfolge der Bedingungen beachten!

Übungen

Übungen

- **Übungen**
  - immer im Anschluss an Vorlesung
  - Abgabe bis jeweils folgenden Sonntag (6 Tage)
  - bewertet mit 0-10 Punkten
  - Tutor: Martin Sommerlandt
    - erster Ansprechpartner bei Problemen
    - E-Mail: martin.sommerlandt@inf.tu-dresden.de

- **Server**
  - Linux-Server, Zugangsdaten auf Handout
  - bis zum Ende des Semesters zur Verfügung gestellt
  - Verzeichnis: `/home/<login>`
    - eigenes Verzeichnis
    - DB2-Instanz (namens `<login>`)
    - beliebig viele Datenbanken
    - eigene Skripte
  - Verzeichnis `/home/dbprog`
    - Daten
    - Skripte (`*.sql`)
    - Lösungen der Übungsaufgaben

DB2-Befehlszeilenprozessor

- **DB2-Befehlszeilenprozessor**
  - verarbeitet SQL-Anweisungen und DB2-Kommandos
  - 2 Modi
    - Skriptmodus
    - interaktiver Modus

- **Skriptmodus**
  - führt SQL-Skripte aus
  - starten: `db2 –tf <name>.sql`
  - Anweisungstrennzeichen ist Semikolon (;)
  - Beispiel: `test.sql`
    - ```
      CONNECT TO SAMPLE;
      SELECT COUNT(*) FROM EMPLOYEE;
      CONNECT RESET;
      ```
      ```

DB2-Befehlszeilenprozessor

- **Interaktiver Modus**
  - starten mit: db2
  - DB2 => **CONNECT TO** SAMPLE

```
   Datenbankverbindungsinformationen
   Datenbank-Server                 = DB2/LINUX 8.2.1
   SQL-Berechtigungs-ID             = RG
   Aliasname der lokalen Datenbank = SAMPLE

DB2 => SELECT COUNT(*) FROM EMPLOYEE
1
-----------
         32
  1 Satz/Sätze ausgewählt.

DB2 => QUIT
DB20000I  Der Befehl QUIT wurde erfolgreich ausgeführt.
```

Prompt

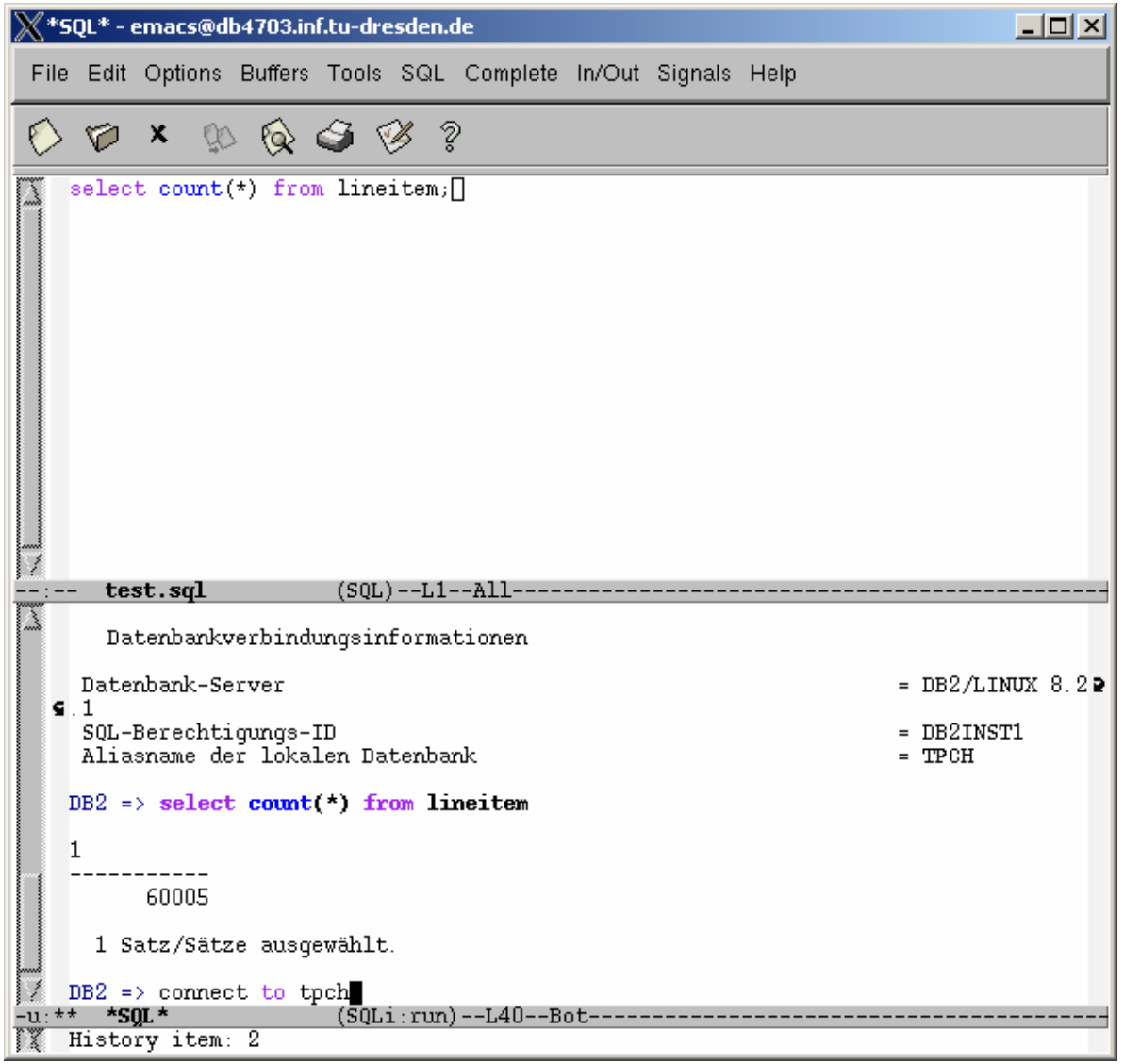
Befehl

- **SSH-Verbindung**
  - unter Linux: Umleitung von Grafikausgabe möglich
  - `ssh –X login@server` oder `ssh –Y login@server`
- **Emacs**
  - starten mit: `emacs <dateiname> &`
  - speichern: Strg+X S
  - verlassen: Strg+X Strg+C
  - Emacs-DB2-Mode: Alt+X sql-db2
  - letzer/nächster Befehl: Strg+↑ / Strg+↓
- **Alternative Editoren**
  - kwrite
  - kate
  - vi
- **Windows**
  - 2-mal verbinden (PuTTY)
  - Editor in Konsole 1, Shell in Konsole 2

Aufbau des Verzeichnis DBPROG

- **Aufbau des Verzeichnis DBPROG**
  - /build
    - Skripte zur Erstellung von Datenbankanwendung, Stored Procedures und UDFs
    - Aufruf: `build-app.sh <datei>`
  - /data
    - DDL- und Load-Skripte für TPC-H und GeoDB
  - /examples
    - Quellcode ausgewählter Beispiele aus den Vorlesungen
  - /jars
    - Bibliotheken zur Entwicklungen von Webservice-Anwendungen
  - /solutions
    - Übungslösungen, Veröffentlichung nach dem jeweiligen Abgabetermin

Screenshot


SQL - emacs@db4703.inf.tu-dresden.de
File Edit Options Buffers Tools SQL Complete In/Out Signals Help
select count(*) from lineitem;
--:-- test.sql (SQL)--L1--All
Datenbankverbindungsinformationen
Datenbank-Server = DB2/LINUX 8.2
SQL-Berechtigungs-ID = DB2INST1
Aliasname der lokalen Datenbank = TPCH
DB2 => select count(*) from lineitem
1
60005
1 Satz/Sätze ausgewählt.
DB2 => connect to tpch
-u:** *SQL* (SQLi:run)--L40--Bot
History item: 2

Zusammenfassung

- **Beispieldatenbanken**
  - DB2-Beispieldatenbank
  - TPC-H
  - GeoDB
- **SQL Grundlagen**
  - **SELECT**-Anweisung
  - innerer und äußerer Verbund
  - Mengenoperationen
  - Fallunterscheidung
- **Arbeiten mit DB2**
  - DB2-Befehlszeilenprozessor
  - siehe Handout

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut Systemarchitektur, Professur Datenbanken

Datenbankprogrammierung
2. (Noch mehr) SQL

Modularisierung von Unterabfragen

- **Modularisierung von Anfragen**
  - Unterabfrage (**WHERE**)
  - Nested Table Expression (**FROM**)
  - Scalar Full Select (**SELECT**, **WHERE**, **HAVING**)
  - Common Table Expression (**WITH**)

- **Unterabfrage** (*subquery*)
  - liefert Relation als Ergebnis
  - in **WHERE**-Klausel zur Existenzüberprüfung verwendet
  - Syntax

```
WHERE EXISTS
      | <expr> [NOT] IN
      | <expr> <|>|… [ALL|SOME|ANY]
(<subquery>)
```

- **Unkorrelierte Unterabfrage**
  - **SELECT** N_NAME **FROM** NATION **WHERE** N_REGIONKEY **NOT IN** (
    SELECT R_REGIONKEY **FROM** REGION **WHERE** R_NAME <> 'EUROPE')

```
N_NAME
--------------
FRANCE
GERMANY
ROMANIA
RUSSIA
UNITED KINGDOM
```

  - **SELECT** LASTNAME, JOB, SALARY **FROM** EMPLOYEE **WHERE** SALARY > **SOME**
    (**SELECT** SALARY **FROM** EMPLOYEE **WHERE** JOB='MANAGER')

```
LASTNAME     JOB       SALARY
-----------  --------  ---------
HAAS         PRES      152750
THOMPSON     MANAGER   94250
KWAN         MANAGER   98250
GEYER        MANAGER   80175
…
```

- **SELECT** LASTNAME, JOB, SALARY **FROM** EMPLOYEE
  WHERE SALARY > **ALL** (**SELECT** SALARY **FROM** EMPLOYEE
  WHERE JOB='MANAGER')

```
LASTNAME     JOB       SALARY
-----------  --------  ---------
HAAS         PRES      152750
```

- **Korrelierte Unterabfrage**
  - **SELECT** R_NAME **FROM** REGION **WHERE NOT EXISTS** (
    SELECT * **FROM** NATION
    WHERE N_REGIONKEY = R_REGIONKEY **AND** N_NAME **LIKE** 'A%')

```
R_NAME
-----------
ASIA
EUROPE
MIDDLE EAST
```

Modularisierung von Anfragen

- **Nested Table Expression**
  - liefert Relation als Ergebnis
  - in **FROM**-Klausel als Eingaberelation verwendet
  - muss benannt werden (**AS**)

- **Beispiel**
  - ```
    SELECT N_NAME FROM NATION AS T1,
           (SELECT R_REGIONKEY FROM REGION
            WHERE R_NAME = 'EUROPE') AS T2
    WHERE T1.N_REGIONKEY = T2.R_REGIONKEY

    N_NAME
    --------------
    FRANCE
    GERMANY
    ROMANIA
    RUSSIA
    UNITED KINGDOM
    ```
    ```

Modularisierung von Anfragen

- **Scalar Full Select**
  - liefert einen einzigen Rückgabewert als Ergebnis
  - im Projektions-/Selektionsteil verwendet

- **Beispiel**
  -
```
SELECT C_NATIONKEY,
       COUNT(*) AS PER_NATION,
       (SELECT COUNT(*) FROM CUSTOMER) AS TOTAL,
FROM CUSTOMER
GROUP BY C_NATIONKEY

C_NATIONKEY     PER_NATION     TOTAL
--------------  -------------  --------
0               3              1500
1               107            1500
2               6              1500
3               46             1500
…
```

Modularisierung von Anfragen

- **Gemeinsame Tabellenausdrücke** *(common table expressions)*
  - DB2-spezifisch
  - Definition von (virtuellen) Tabellen für **SELECT**-Anweisung
  - ermöglicht Rekursion
  - **WITH** <tab-name> [(<col-name>,…)] **AS** <select-stmt>, …
    SELECT …

- **Beispiel**
  -
```
WITH EUROPEAN_NATIONS(N_NATIONKEY, N_NAME) AS (
    SELECT N_NATIONKEY, N_NAME FROM NATION, REGION
    WHERE N_REGIONKEY = R_REGIONKEY AND R_NAME = 'EUROPE'
)
SELECT N_NAME FROM EUROPEAN_NATIONS;

N_NAME
--------------
FRANCE
GERMANY
ROMANIA
RUSSIA
UNITED KINGDOM
```

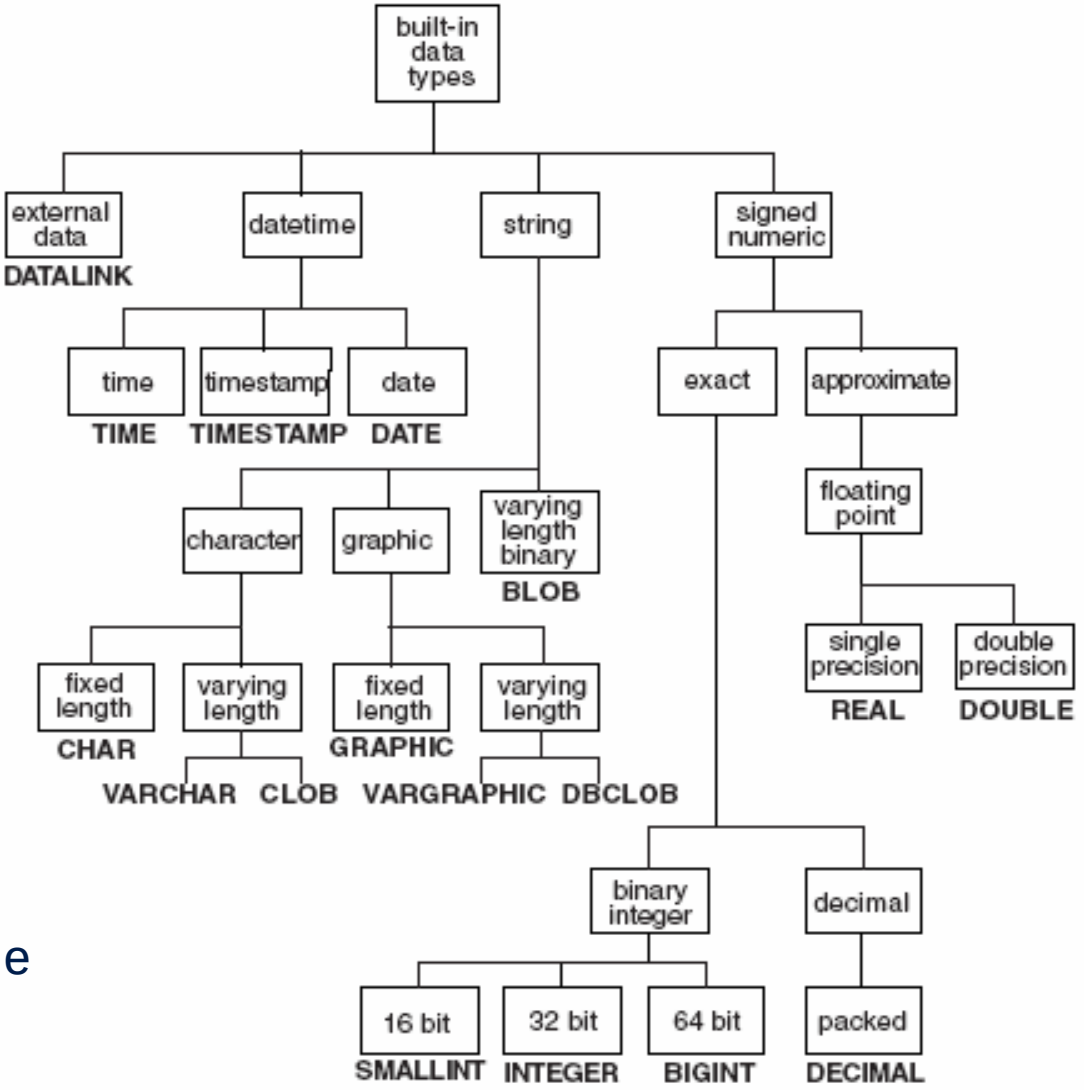
Datentypen

Datentypen

- **Standarddatentypen**
  - z.B. **FLOAT**, **DOUBLE**, **INTEGER**, **CHAR**, **VARCHAR**, **BOOLEAN**
  - atomar / keine Struktur
  - vom Datenbanksystem auswertbar
- **Datentypen für große Objekte (*LOB*, *large object*)**
  - z.B. **CLOB** für Text-, **BLOB** für Binärdaten
  - Struktur dem Datenbanksystem unbekannt, z.B. XML-Dateien, Bilder, …
  - vom Datenbanksystem nicht auswertbar, es existieren bspw. keine Vergleichsoperationen (<, =, >)
  - oft in getrennten Tabellenbereichen gespeichert
  - Verarbeitung mit Hilfe von DB2-Extendern möglich
- **Strukturierte Datentypen**
  - siehe Datenbankadministration

Eingebaute Datentypen (SQL Reference Vol. 1)

- **Eingebaute Datentypen**
  - Zeichenketten
    - feste oder variable Länge
    - 1-Byte-Zeichen (**CHARACTER**)
    - 2-Byte-Zeichen (**GRAPHIC**)
    - Binärdaten (**BLOB**)
  - Zahlen
    - exakt
    - approximiert
  - Zeit
  - Datalinks
    - verweisen auf externe Daten



Typkonvertierung

- **Implizite Typkonvertierung (*type promotion*)**
  - bei kompatiblen Datentypen
  - nach Vorrangliste

Quelldatentyp	Zieldatentyp (absteigend)
SMALLINT	SMALLINT, INTEGER, DECIMAL, REAL, DOUBLE
INTEGER	INTEGER, DECIMAL, REAL, DOUBLE
DECIMAL	DECIMAL, REAL, DOUBLE
REAL	REAL, DOUBLE
DOUBLE	DOUBLE
CHAR / GRAPHIC	CHAR / GRAPHIC, VARCHAR / VARGRAPHIC, CLOB / DBCLOB
VARCHAR / VARGRAPHIC	VARCHAR / VARGRAPHIC, CLOB / DBCLOB
CLOB / DBCLOB	CLOB / DBCLOB
BLOB	BLOB
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP

Typkonvertierung

- **Explizite Typkonvertierung** (*type cast*)
  - Typkonvertierung kann explizit gefordert werden
  - **CAST**-Operator
    - Syntax: **CAST**(<value> **AS** <data-type>)
  - auch Umwandlung zwischen numerischen Datentypen und Zeichenketten
  - Beispiel
    - **SELECT** 1/2, **CAST**(1/2 **AS DOUBLE**),
      CAST(1 **AS DOUBLE**)/2, 1.0/2
      FROM SYSIBM.SYSDUMMY1

```
1      2      3      4
----   ----   ----   ----
0      0      0.5    0.5
```

implizite Typkonvertierung
(**INTEGER → DOUBLE**)

INTEGER **DOUBLE**

Typkonvertierung

- **Explizite Typkonvertierung**
  - unterstützte Konvertierungen

Cast from data type –	To data type[1]															
	SMALLINT	INTEGER	DECIMAL	REAL	DOUBLE	CHAR	VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	DBCLOB	BLOB	DATE	TIME	TIMESTAMP	ROWID
SMALLINT	Y	Y	Y	Y	Y	Y	Y									
INTEGER	Y	Y	Y	Y	Y	Y	Y									
DECIMAL	Y	Y	Y	Y	Y	Y	Y									
REAL	Y	Y	Y	Y	Y	Y	Y									
DOUBLE	Y	Y	Y	Y	Y	Y	Y									
CHAR	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
VARCHAR	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
CLOB						Y	Y	Y	Y	Y	Y	Y				
GRAPHIC	Y	Y	Y	Y	Y	Y[2]	Y[2]	Y[2]	Y	Y	Y	Y	Y[3]	Y[3]	Y[3]	
VARGRAPHIC	Y	Y	Y	Y	Y	Y[2]	Y[2]	Y[2]	Y	Y	Y	Y	Y	Y	Y	
DBCLOB						Y[2]	Y[2]	Y[2]	Y	Y	Y	Y				
BLOB												Y				
DATE						Y	Y						Y			
TIME						Y	Y							Y		
TIMESTAMP						Y	Y						Y	Y	Y	
ROWID						Y	Y					Y				Y

Note:
1. Other synonyms for the listed data types are considered to be the same as the synonym listed. Some exceptions exist when the cast involves character string data if the subtype is FOR BIT DATA.
2. The result length for these casts is 3 * LENGTH(graphic string).
3. These data types are castable between each other only if the data is Unicode.

Distinct Types

- **User-Defined Distinct Type (*UDDT*)**
  - nutzerdefiniert
  - aber: nicht kompatibel mit Quelldatentyp (*strong typing*)
  - machen die Arbeit mit dem System sicherer (z.B. **BLOB**)
  - **CREATE DISTINCT TYPE** <name> **AS** <source-data-type> [**WITH COMPARISONS**]
    - Quelldatentyp muss eingebauten Datentyp entsprechen
    - **WITH COMPARISONS** erlaubt Vergleiche
      - muss für Standarddatentypen angegeben werden
      - darf nicht für **LOB**/**DATALINK** angegeben werden
  - automatische Erstellung von Typumwandlungsfunktionen
    - <source-type-name> (<type-name>)
    - <type-name> (<source-type-name>)
  - Löschen mittels: **DROP** <name>
    - nur falls nicht verwendet (z.B. in Tabelle/Sicht)

Distinct Types

- **Beispiel**
  - ```
    CREATE DISTINCT TYPE IMAGE_ID_T AS INTEGER
    WITH COMPARISONS;
    CREATE DISTINCT TYPE IMAGE_T AS BLOB;

    CREATE TABLE IMAGES (
        ID IMAGE_ID_T,
        IMAGE IMAGE_T
    );
    ```
  - `SELECT ID+1 FROM IMAGES` → Fehler
  - `SELECT INTEGER(ID)+1 FROM IMAGES` → OK
  - `SELECT IMAGE FROM IMAGES WHERE ID=1` → Fehler
  - `SELECT IMAGE FROM IMAGES WHERE ID=IMAGE_ID_T(1)` → OK
    ```

Datenmanipulation (DML)
Datendefinition (DDL)

- **Einfügen von Tupeln**
  - `INSERT INTO REGION VALUES (6,'Antarctica','')`
  - `INSERT INTO NATION (N_NATIONKEY, N_NAME, N_REGIONKEY)`
    `SELECT NATIONKEY, NAME, 6 FROM ANTARCTIC_NATIONS`
- **Verändern von Tupeln**
  - `UPDATE LINEITEM`
    `SET L_DISCOUNT = L_DISCOUNT + 0.01`
    `WHERE L_SUPPKEY = 12`
- **Löschen von Tupeln**
  - `DELETE FROM REGION WHERE R_REGIONKEY > 5`

- **Zusammenführen von Daten (`MERGE`)**
  - DB2-spezifische Kombination aus **`INSERT`**, **`DELETE`** und **`UPDATE`**
  - vereinigt zwei Relationen auf vorgegebene Art und Weise
  - **`MERGE INTO`** `<table1>`
    `USING` `<table2>|<fullselect>` **`ON`** `<condition>`
    `WHEN` `[`**`NOT`**`]` **`MATCHED THEN`** `<operation>`
  - Parameter
    - `<table1>`: zu aktualisierende Tabelle
    - `<table2>`: neue Daten
    - `<condition>`: Tupelvergleich (wann sind 2 Tupel gleich?)
    - **`MATCHED`**: Tupel in beiden Tabellen vorhanden
    - **`NOT MATCHED`**: Tupel nur in neuer Tabelle
    - `<operation>`: Aktion (**`UPDATE`**, **`INSERT`**, **`DELETE`**)

- **Beispiel (MERGE)**
  - 2 Relationen: PERSONS (links) und NEW_PERSONS (rechts)

```
NAME     SALARY              NAME     SALARY
-------  ---------           -------  ---------
Paul     2000
Fritz    1000                Fritz    1500
                             Dirk     3000
```

  - Vereinigung beider Relationen

```
MERGE INTO PERSONS P
USING NEW_PERSONS N ON P.NAME = N.NAME
WHEN MATCHED THEN UPDATE SET P.SALARY = N.SALARY
WHEN NOT MATCHED THEN INSERT (NAME,SALARY)
                      VALUES (N.NAME, N.SALARY)
```

  - Ergebnis (PERSONS)

```
NAME     SALARY
-------  ---------
Paul     2000
Fritz    1500
Dirk     3000
```

DB2-Datenbankobjekte

- **Datentyp** (**TYPE**)
  - Definitionsbereich eines Attributes
  - fest eingebaut (*built-in*) oder nutzerdefiniert (*user-defined*)
- **Relation** (**TABLE**)
  - ungeordnete Menge von Tupeln
  - permanent oder temporär
- **Sicht** (**VIEW**)
  - virtuelle Relation
  - z.B. Vereinfachung von Anfragen, nutzerspezifische Datendarstellung
- **Index** (**INDEX**)
  - primär oder sekundär
  - Datenstruktur zum Ablegen / Auffinden von Daten
  - auch: Sicherstellung der Eindeutigkeit, Vorsortierung
- **u.a.** Datenbank (**DATABASE**), Tabellenbereich (**TABLESPACE**), Schema (**SCHEMA**), Pufferspeicher (**BUFFERPOOL**) …

- **Definition von Datenbankobjekten (DDL)**
  - hier am Beispiel einer Tabelle
  - Erzeugen mittels **CREATE**

```
CREATE TABLE REGION (
    R_REGIONKEY INTEGER NOT NULL PRIMARY KEY,
    R_NAME CHAR(25) NOT NULL,
    R_COMMENT VARCHAR(152)
)
```

  - Löschen mittels **DROP**

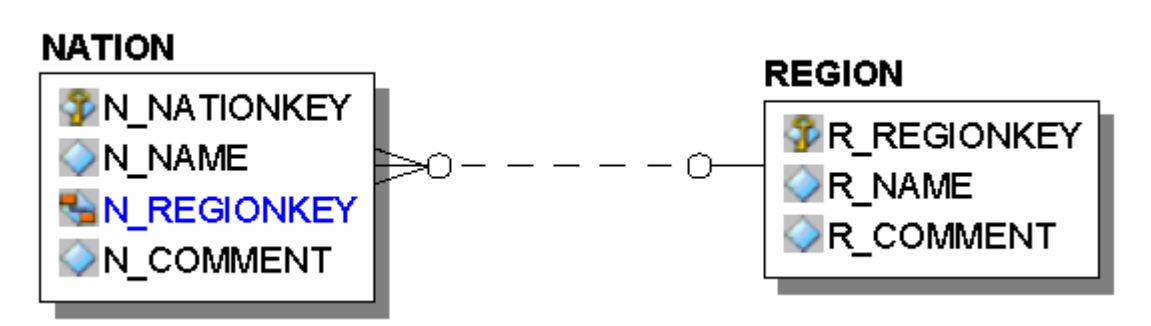
```
DROP TABLE REGION
```

  - Verändern mittels **ALTER**

```
ALTER TABLE REGION ADD COLUMN AREA INT
```

- **Integritätsbedingungen (*constraints*)**
  - Primärschlüssel, Fremdschlüssel, Disjunktheit (**UNIQUE**), wertebasiert (**CHECK**)
  - **ALTER TABLE** <table> **ADD CONSTRAINT** <constraint-name>
    - **PRIMARY KEY** (<attribute-list>)
    - **FOREIGN KEY** (<attr-list>) **REFERENCES** <table>(<attr-list>)
    - **UNIQUE** (<attribute-list>)
    - **CHECK** (<predicate>)
  - Beispiel

```
ALTER TABLE REGION
ADD CONSTRAINT MAX5 CHECK (R_REGIONKEY BETWEEN 1 AND 5)
```



- **Sicht** (**VIEW**)
  - virtuelle Relation zur Vereinfachung von Anfragen bzw. nutzerspezifische Datendarstellung
  - auch Datenschutz durch Ausblenden von Tupeln / Attributen
  - Spezifikation des Inhaltes durch SQL-Anfrage
  - Beispiel
    - ```
      CREATE VIEW NAT_REG AS
      SELECT N_NAME, R_NAME
      FROM NATION, REGION
      WHERE N_REGIONKEY = R_REGIONKEY
      ```
    - ```
      SELECT * FROM NAT_REG
      ```

      ```
      N_NAME          R_NAME
      -------------   --------
      ALGERIA         AFRICA
      MOZAMBIQUE      AFRICA
      MOROCCO         AFRICA
      …
      ```

Modifikation von Sichten

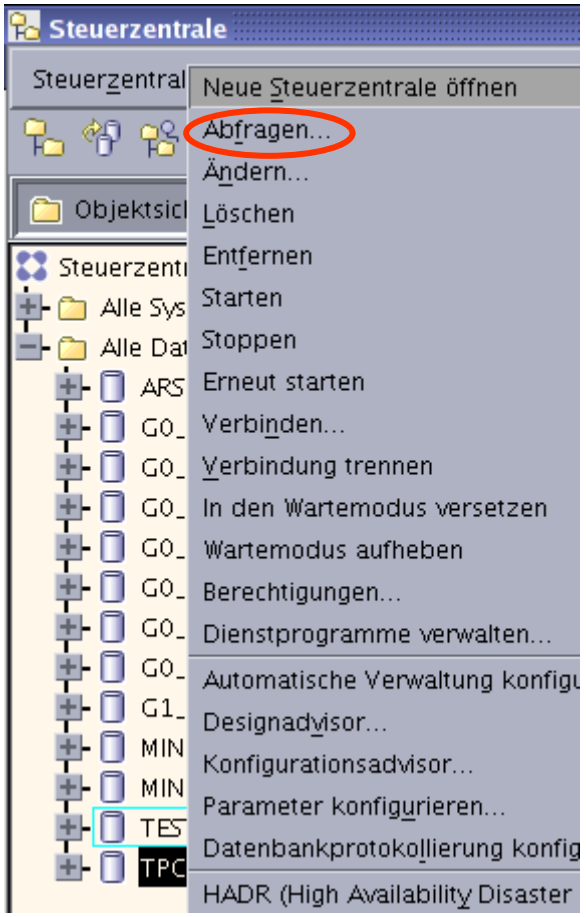
- **Modifikationen von Sichten**
  - **INSERT**, **DELETE** und **UPDATE** möglich, falls
    - Variante 1: **INSTEAD OF**-Trigger definiert (später)
    - Variante 2: bestimmte Bedingungen erfüllt, siehe http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0000935.htm
  - **WITH CHECK OPTION**
    - eingefügte/aktualisierte Tupel müssen zur Sicht gehören

- **Beispiel**
  - ```
    CREATE VIEW R AS
    SELECT R_REGIONKEY, R_NAME FROM REGION
    WHERE R_REGIONKEY<10
    WITH CHECK OPTION
    ```
  - `INSERT INTO R VALUES (8, 'A')` → OK
  - `INSERT INTO R VALUES (11, 'B')` → Fehler
    ```

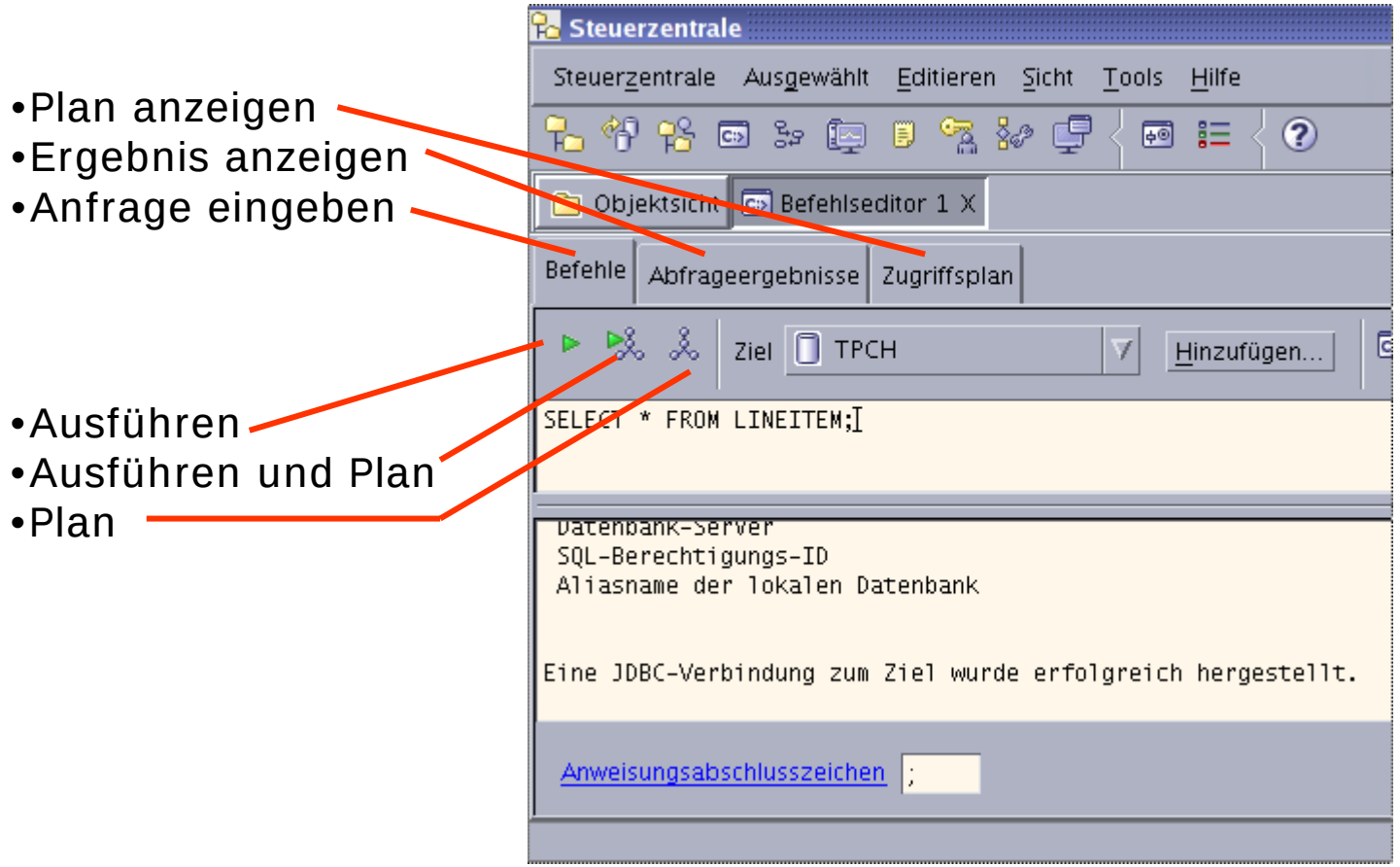
DB2-Steuerzentrale

DB2-Steuerzentrale

- **DB2-Steuerzentrale (*command center*)**
  - zentrale Administrationsplattform
  - starten mit: db2cc
  - ermöglicht
    - Anlegen und Löschen von Datenbanken
    - Konfiguration von Datenbanken
    - Ausführung von SQL-Befehlen
    - Starten anderer DB2-Tools
    - ...



- **DB2-Steuerzentrale**
  - Ausführen von SQL-Kommandos



Zusammenfassung

- **Modularisierung von Anfragen**
  - Unterabfrage
  - Nested Table Expressions
  - Scalar Full Select
  - Gemeinsame Tabellenausdrücke
- **Datentypen**
  - eingebaute Datentypen & Typkonvertierung
  - UDDTs
- **Datendefinition und Datenmanipulation**
  - **`INSERT`**, **`UPDATE`**, **`DELETE`**, **`MERGE`**
  - Tabelle
  - Integritätsbedingungen
  - Sichten

Die Lösung der ersten Übung befindet sich auf dem Server:
`/home/dbprog/solutions/01/`

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut Systemarchitektur, Professur Datenbanken

Datenbankprogrammierung
3. DB2 Architektur

DB2-Produktpalette

DB2-Produktpalette

- **DB2 Universal Database**
  - Everyplace
    - Fingerprint-Datenbank für mobile Geräte (Handhelds)
  - Personal Edition
    - Einzelnutzer-RDBMS
  - Workgroup Server Edition
    - Client/Server-RDBMS mit voller Funktionalität
    - Lizenzierung pro Nutzer
  - Workgroup Server Unlimited Edition
    - Lizenzierung pro Prozessor
  - Enterprise Server Edition
    - Erweiterte Funktionalität, z.B. Database Partitioning Feature (DPF)

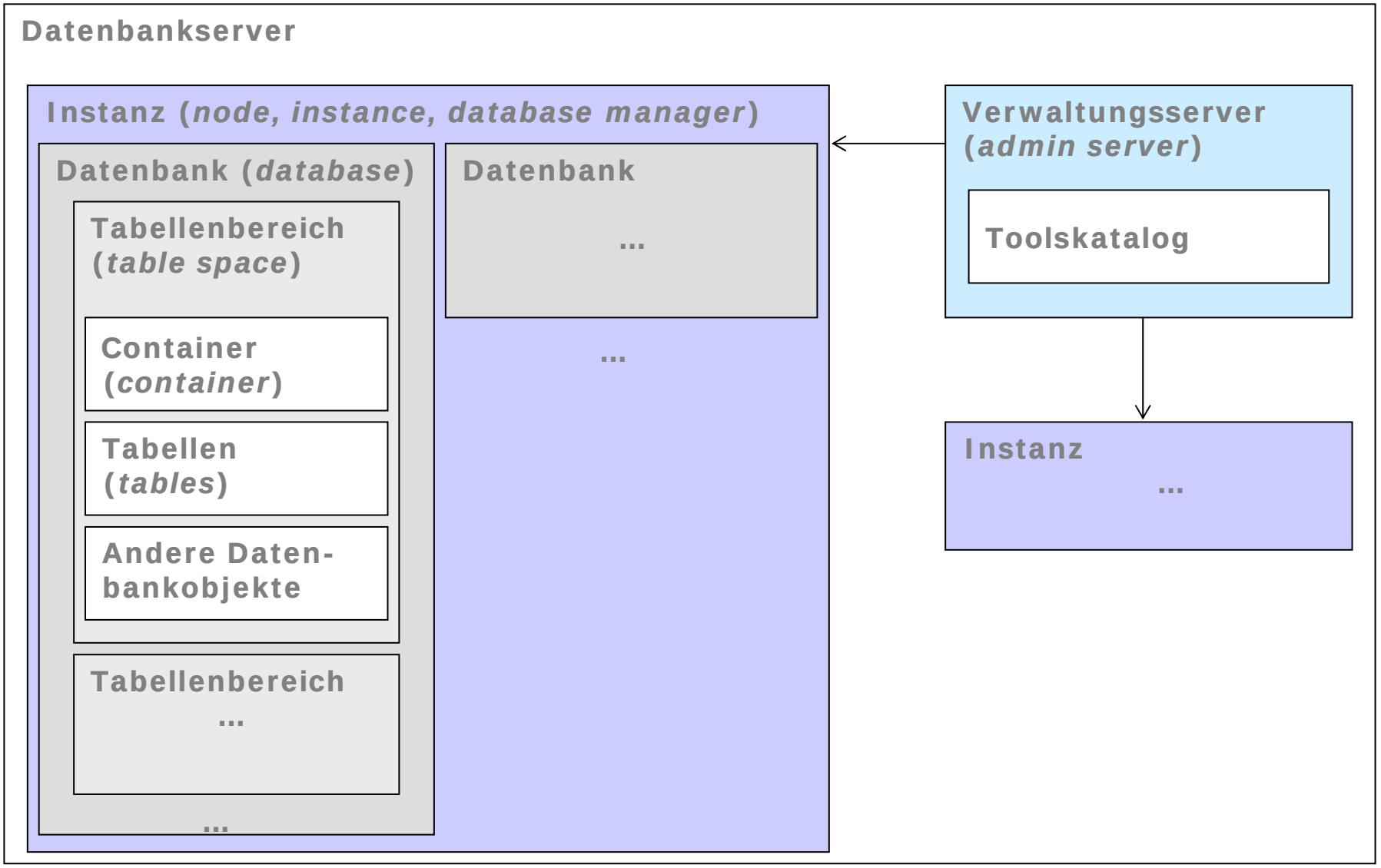
DB2-Produktpalette

- **DB2 Express-C**
  - Community-Edition
  - Windows, Linux
  - erhältlich unter: http://www-306.ibm.com/software/data/db2/udb/db2express/
  - einige Einschränkungen, z.B.
    - maximal 2 Prozessoren
    - maximal 4 GB Hauptspeicher

- **Weitere Informationen**
  - Tutorial 700/1
  - Achtung: Zertifizierung!

DB2-Architektur

DB2-Komponenten



Verwaltungsserver

- **DB2 Verwaltungsserver (*admin server*)**
  - max. 1 Verwaltungsserver pro DB2-Server
  - ermöglicht
    - Fernverwaltung
    - Jobverwaltung (z.B. Abbrechen von Stored Procedures)
    - Durchführen von Verwaltungsaufgaben (Tasks)
    - DB2-Discovery zum Aufspüren von Datenbanken
    - Verwendung der DB2 Kommandozentrale
  - Toolskatalog
    - nimmt Informationen über Tasks und deren Ergebnisse auf
  - Starten / Stoppen
    - nur vom DB2-Verwaltungsnutzer
    - **`db2admin start|stop`**
  - Konfiguration
    - **`GET ADMIN CFG`**
    - **`UPDATE ADMIN CFG USING`** `<parameter> <value>`

- **Konfiguration des Verwaltungsservers**

```
DB2 => GET ADMIN CFG

Konfiguration des Verwaltungs-Servers

 DAS für Authentifizierungstyp                  (AUTHENTICATION)  = SERVER_ENCRYPT

 DASADM-Gruppenname                             (DASADM_GROUP)    = dasadm1

 DAS-Discovery-Modus                            (DISCOVER)        = SEARCH
 Name des DB2-Serversystems                     (DB2SYSTEM)       = DB4708

 Installationspfad für Java Development Kit     (JDK_PATH)        = /opt/…
 Installationspfad für Java Development Kit     (JDK_64_PATH)     = /opt/…

 DAS-Codepage                                   (DAS_CODEPAGE)    = 0
 DAS-Gebiet                                     (DAS_TERRITORY)   = 0

 Speicherposition für Ansprechpartnerliste      (CONTACT_HOST)    =
 Abgelaufene Tasks ausführen                    (EXEC_EXP_TASK)   = NO
 Schedulermodus                                 (SCHED_ENABLE)    = OFF
 SMTP-Server                                    (SMTP_SERVER)     =
 Toolskatalogdatenbank                          (TOOLSCAT_DB)     =
 Exemplar der Toolskatalogdatenbank             (TOOLSCAT_INST)   =
 Schema für Toolskatalogdatenbank               (TOOLSCAT_SCHEMA) =
 Benutzer-ID für Scheduler                                        =
```

Instanz

- **Instanz (*instance, node, database manager*)**
  - Was?
    - Logischer Kontext für Prozeduren und DB2 Kommandos
    - mehrere Instanzen pro DB2-Server möglich
    - enthalten Datenbanken
    - sind untereinander unabhängig (!)
  - Instanz für Instanzkommandos festlegen
    - **ATTACH TO** <instance>
    - **DETACH**
  - Starten / Stoppen
    - nur vom Instanzadministrator
    - **db2start** | **db2stop**
  - Parameter:
    - **GET DBM CFG** [**SHOW DETAIL**]
    - **UPDATE DBM CFG USING** <parameter> <value>

- **Instanzkonfiguration**

```
DB2 => GET DBM CFG

Datenbankmanagerkonfiguration

Knotentyp                 = Enterprise Server Edition mit lokalen und fernen Clients

Releasestand der Datenbankmanagerkonfiguration                    = 0x0a00

Max. Anzahl offener Dateien                        (MAXTOTFILOP)    = 16000
CPU-Geschwindigkeit (ms/Anweisung)                 (CPUSPEED)       = 6,140476e-007
Kommunikationsbandbreite (MB/s)                    (COMM_BANDWIDTH) = 1,000000e+002

Max. Anzahl gleichzeitig aktiver Datenbanken       (NUMDB)          = 8
Data Links-Unterstützung                           (DATALINKS)      = NO
Unterst. f. System zusammengeschloss. Datenbanken  (FEDERATED)      = NO
Name des TP-Monitors                               (TP_MON_NAME)    =

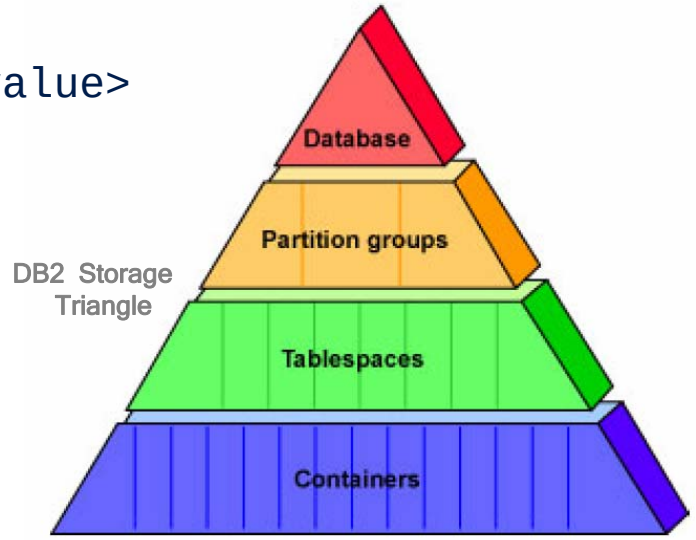
Standardkonto für Rückerstattung                   (DFT_ACCOUNT_STR)=

Java Development Kit: Installationspfad            (JDK_PATH)       = /opt/…

Aufzeichnungsebene bei Fehlerdiagnose              (DIAGLEVEL)      = 3
Aufzeichnungsebene                                 (NOTIFYLEVEL)    = 3
Verzeichnispfad für Diagnosedaten                  (DIAGPATH)       =

Standardschalter für den Datenbankmonitor
  Pufferpool                                       (DFT_MON_BUFPOOL)= OFF
  Sperre                                           (DFT_MON_LOCK)   = OFF
  Sortierung                                       (DFT_MON_SORT)   = OFF
  Anweisung                                        (DFT_MON_STMT)   = OFF
  Tabelle                                          (DFT_MON_TABLE)  = OFF
                                        …
```

- **Datenbank (*database*)**
  - Sammlung von logischen und physischen Datenbankobjekten
    - Tabellen, Sichten, Indizes, Schemata, Sperren, Trigger, Prozeduren, Pakete, Puffer, Logdateien, Tabellenbereiche, …
  - Verbindung
    - **`CONNECT TO`** `<database>` [**`USER`** `<username>` **`USING`** `<password>`]
    - **`CONNECT RESET`**
  - Parameter
    - **`GET DB CFG`** [**`SHOW DETAIL`**]
    - **`UPDATE DB CFG USING`** `<parameter> <value>`



- **Datenbankkonfiguration**

```
DB2 => GET DB CFG

Konfiguration für Datenbank

 Releasestand der Datenbankkonfiguration                    = 0x0a00
 Releasestand der Datenbank                                 = 0x0a00

 Datenbankgebiet                                            = DE
 Codepage für Datenbank                                     = 1208
 Codierter Zeichensatz der Datenbank                        = UTF-8
 Landescode der Datenbank                                   = 49
 Sortierfolge der Datenbank                                 = BINARY
 Alternative Sortierfolge                  (ALT_COLLATE)    =

 Dynamische SQL-Abfrageverwaltung          (DYN_QUERY_MGMT) = DISABLE

 Discovery-Unterstützung für diese Datenbank (DISCOVER_DB)  = ENABLE

 Standardabfrageoptimierungsklasse         (DFT_QUERYOPT)   = 5
 Grad der Parallelität                     (DFT_DEGREE)     = 1
 Bei arithm. Ausnahmebedingungen fortsetzen (DFT_SQLMATHWARN) = NO
 Standardaktualisierungsalter              (DFT_REFRESH_AGE) = 0
 Standardmäßig für Optimierung verwendete
   verwaltete Tabellentypen                (DFT_MTTB_TYPES) = SYSTEM
 Anzahl der häufigsten Werte               (NUM_FREQVALUES) = 10
 Anzahl der Quantile für Spalten           (NUM_QUANTILES)  = 20

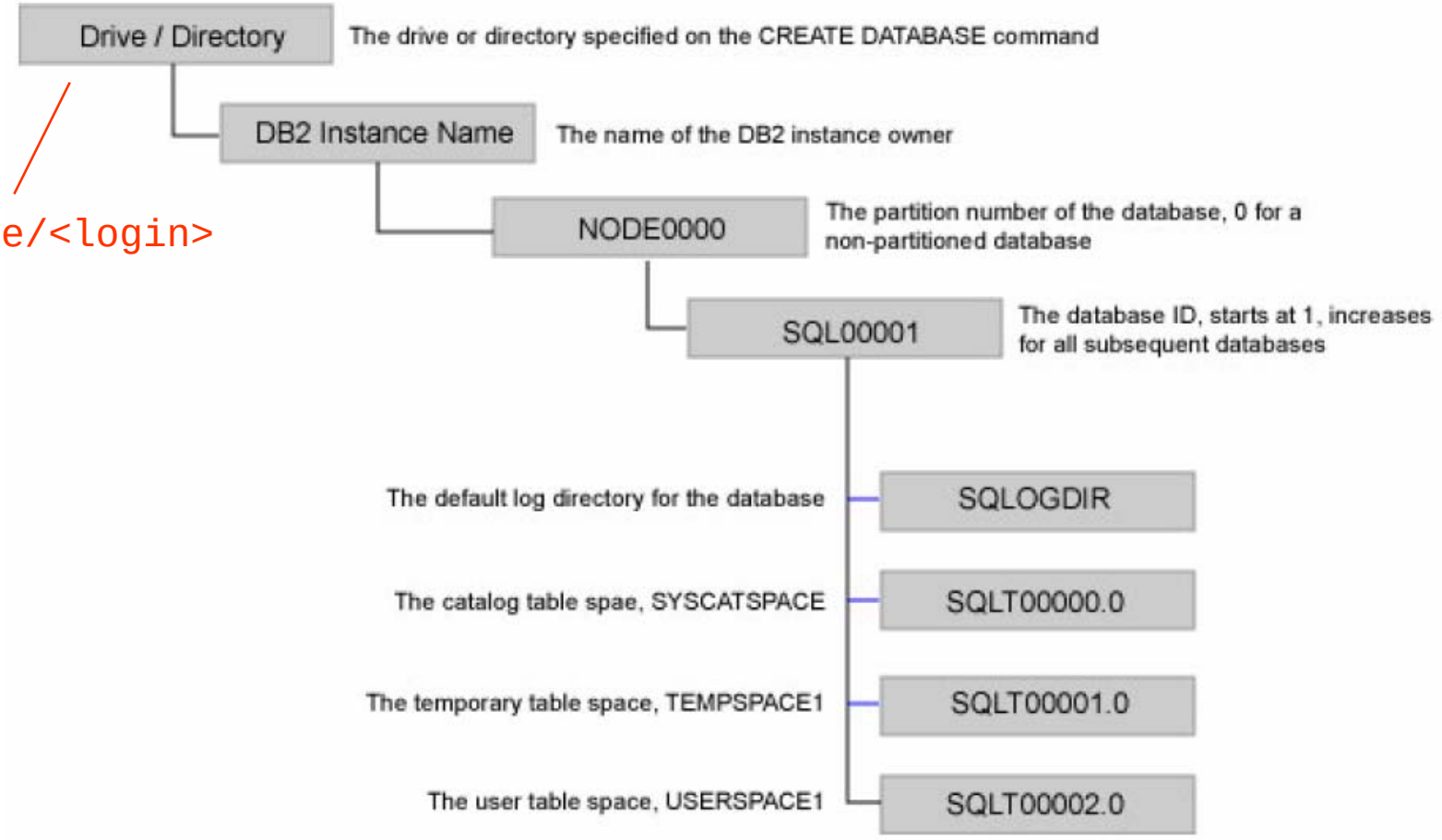
 Sicherung anstehend                                        = NO

 Datenbank ist konsistent                                   = YES
 Aktualisierende Wiederherstellung anstehend                = NO
 Wiederherstellung anstehend                                = NO
                                    ...
```

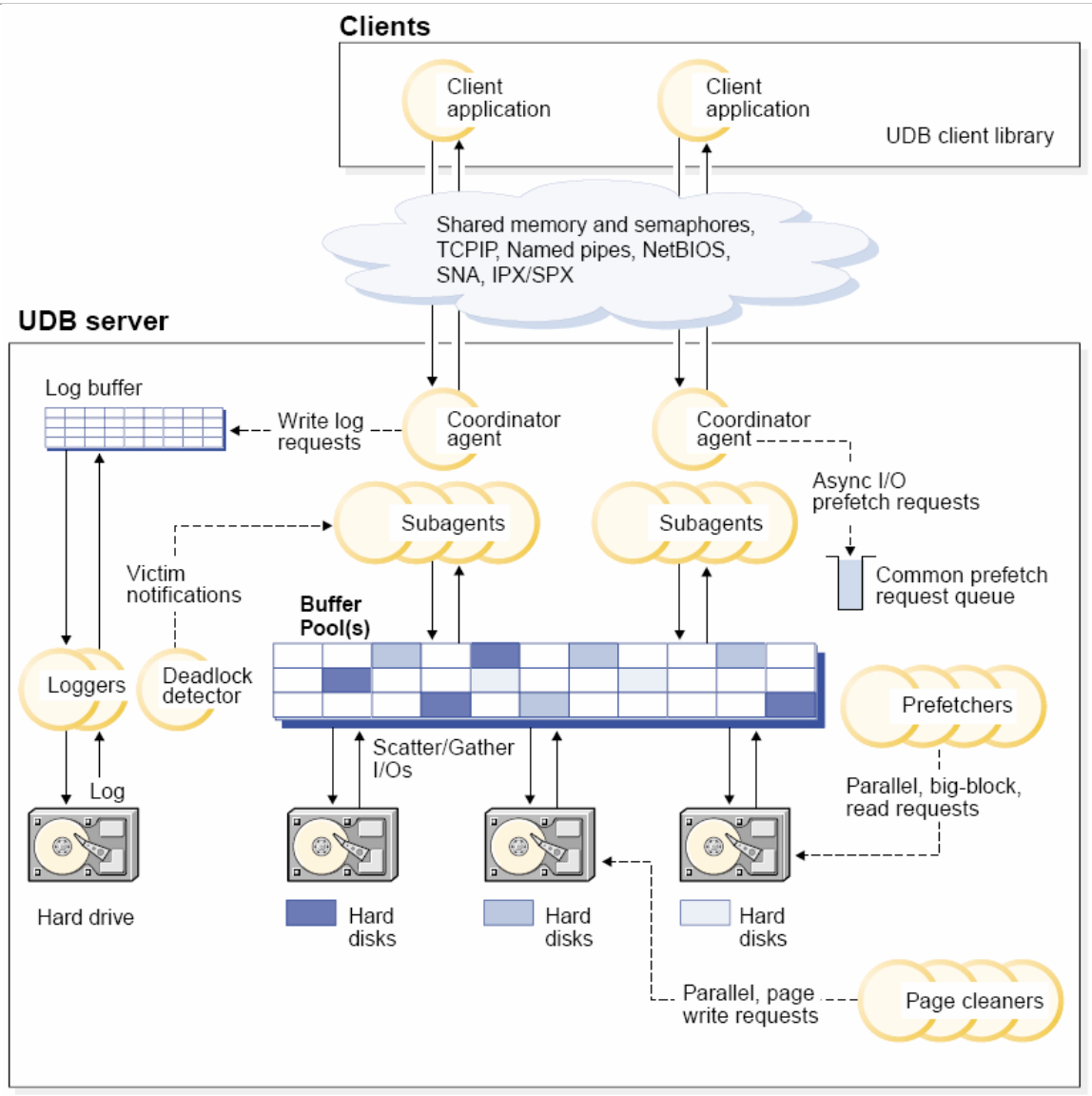
Tabellenbereich

- **Tabellenbereich (*table space*)**
  - bestimmt physische Datenablage
  - besteht aus Containern (physischer Speicherort)
    - Speichernutzung gleichmäßig auf alle Container verteilt
    - Tabellenbereich voll, sobald einer der Container voll
  - Relationen/Indizes sind immer an einen Tabellenbereich gebunden
  - Standardtabellenbereiche
    - SYSCATSPACE: Datenbankkatalog
    - USERSPACE1: Nutzerdefinierte Datenbankobjekte
    - TEMPSPACE1: Temporäre Datenbankobjekte
  - vom System oder der Datenbank verwaltet
  - können explizit bei Erstellung von DB-Objekten angegeben werden
    - `CREATE TABLE <name> (…) [IN <tablespace> [INDEX IN <tablespace>] [LONG IN <tablespace>]]`
  - siehe Datenbankadministration

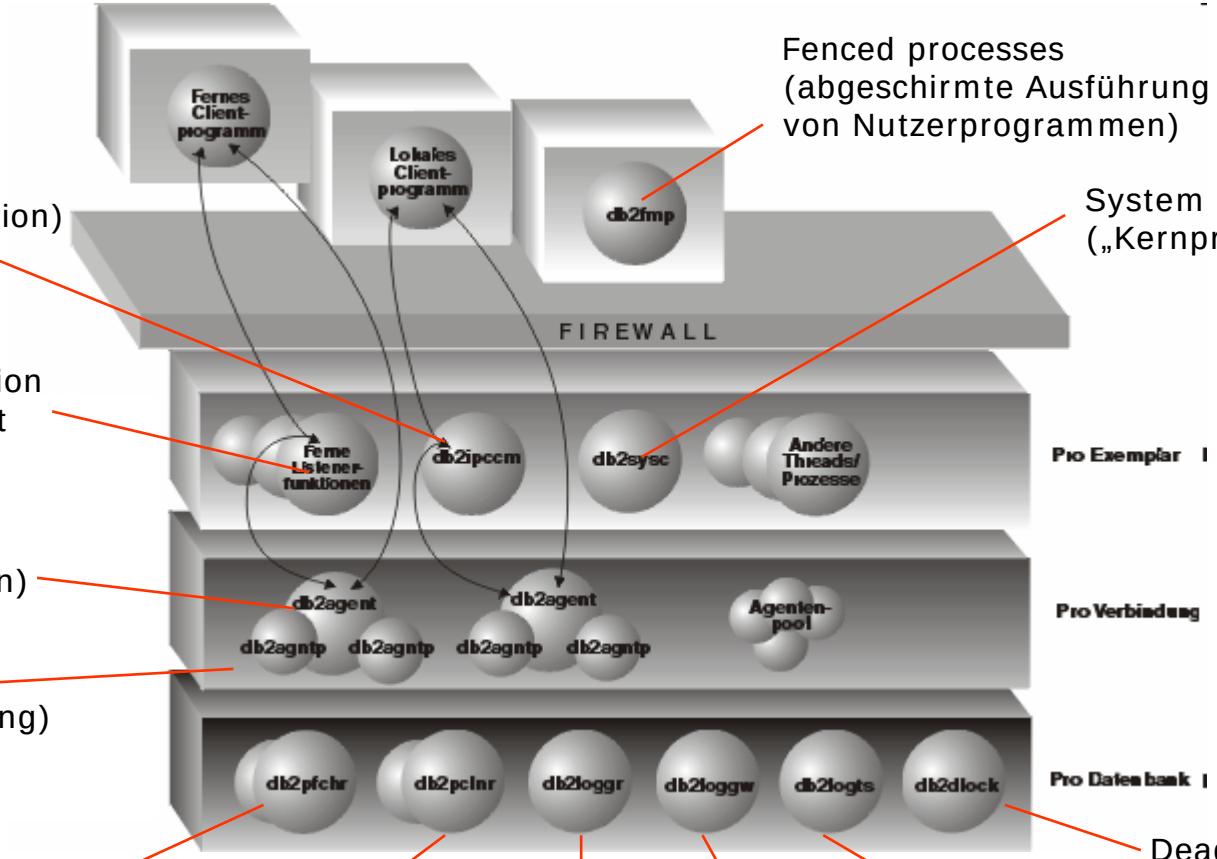
Standardtabellenbereiche



DB2-Architektur



DB2-Prozessmodel



IPC communication manager (Inter-prozesskommunikation)

Fenced processes (abgeschirmte Ausführung von Nutzerprogrammen)

System controller / engine („Kernprozess“)

TPC/IP communication Manager (bearbeitet entfernte Anfragen)

Coordinator agent (bearbeitet Anfragen)

Subagent (parallele Abarbeitung)

Deadlock-Detector (Erkennen und Auflösen von Deadlocks)

Prefetcher (Vorablesen von Daten)

Page cleaner (Schreiben von modi-fizierten Seiten)

Log reader (Lesen von Log-Dateien)

Log writer (Ausschreiben der Log-Dateien)

Log history (Geschwindigkeitsgewinn durch Statistiken)

Systemkatalog

Systemkatalog

- **Systemkatalog**
  - enthält Beschreibungen aller Datenbankobjekte
  - besteht aus Relationen und Sichten
  - Zusammensetzung
    - Tabellenbereich: SYSCATSPACE
    - Schemata: SYSIBM, SYSCAT, SYSFUN, SYSSTAT, SYSPROC
  - SYSCAT-Schema enthält nützliche Sichten
    - BUFFERPOOLS
    - TABLES
    - VIEWS
    - INDEXES
    - TABLESPACES
    - SCHEMATA
    - REFERENCES
    - CHECKS
  - Abfragen per SQL
    - **SELECT** * **FROM** SYSCAT.TABLES

Beispiel: Integritätsbedingungen

- **Beispiel: Integritätsbedingungen**
  - abgelegt in **SYSCAT.TABCONST**
    - NAME: Name der Integritätsbedingung
    - TYPE: Primärschlüssel (P), **UNIQUE**-Constraint (U) , Fremdschlüssel (F), **CHECK**-Constraint (K)
  - weitere Details
    - SYSCAT.INDEXES (Primärschlüssel, **UNIQUE**-Constraint)
    - SYSCAT.REFERENCES (Fremdschlüssel)
    - SYSCAT.CHECKS (**CHECK**-Constraint)

```
DB2=> SELECT * FROM SYSCAT.TABCONST
CONSTNAME   TABSCHEMA TABNAME DEFINER TYPE ENFORCED CHECKEXISTINGDATA ENABLEQUERYOPT
----------- --------- ------- ------- ---- -------- ---------------- --------------
ORDERS_PK   RG        ORDERS  RG      P    Y        I                Y
ORDERS_FK_1 RG        ORDERS  RG      F    Y        I                Y
```

Katalogisieren entfernter Datenbanken

- **Verbindungsaufbau erfordert Katalogisieren**
  - Verwaltungsserver
    - **CATALOG ADMIN TCPIP NODE** <nodename>
      REMOTE <host-name>|<ip-address>
  - Instanz
    - **CATALOG TCPIP NODE** <nodename>
      REMOTE <host-name>|<ip-address>
      SERVER <service-name>|<port>
      [**REMOTE_INSTANCE** <instance-name>]
  - Datenbank
    - **CATALOG DATABASE** <dbname> **AS** <alias>
      AT NODE <nodename>
- **Objekte aus dem Katalog entfernen**
  - **UNCATALOG NODE** <nodename>
  - **UNCATALOG DATABASE** <dbname>
- **Aktualisieren des Verzeichniscache**
  - **TERMINATE**

Ausblick

Inhalt

- **Teil 1: Grundlagen** ✓
  - DB2, SQL, Datenbankobjekte, Systemkatalog, ...
- **Teil 2: Datenbankinterne Programmierung**
  - Nutzerdefinierte Funktionen
  - Stored Procedures
  - Trigger
- **Teil 3: Datenbankexterne Programmierung**
  - JDBC
  - CLI/ODBC
  - (OLEDB/ADO)
- **Teil 4: Mischformen**
  - externe UDFs/SPs/Trigger
  - embedded SQL, SQLJ

Zusammenfassung

- **DB2 Produktpalette**
  - von der mobilen Datenbank bis zum partitionierten DBMS
  - DB2 Express-C
- **DB2 Architektur**
  - Verwaltungsserver
  - Instanz
  - Datenbank
  - Tabellenbereiche
  - Prozessmodell
- **DB2 Systemkatalog**
  - Beschreibung aller bekannten Datenbankobjekte
  - Katalogisieren entfernter Datenbanken

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut Systemarchitektur, Professur Datenbanken

Datenbankprogrammierung
4. Nutzerdefinierte Funktionen

Anwendungslogik in der Datenbank

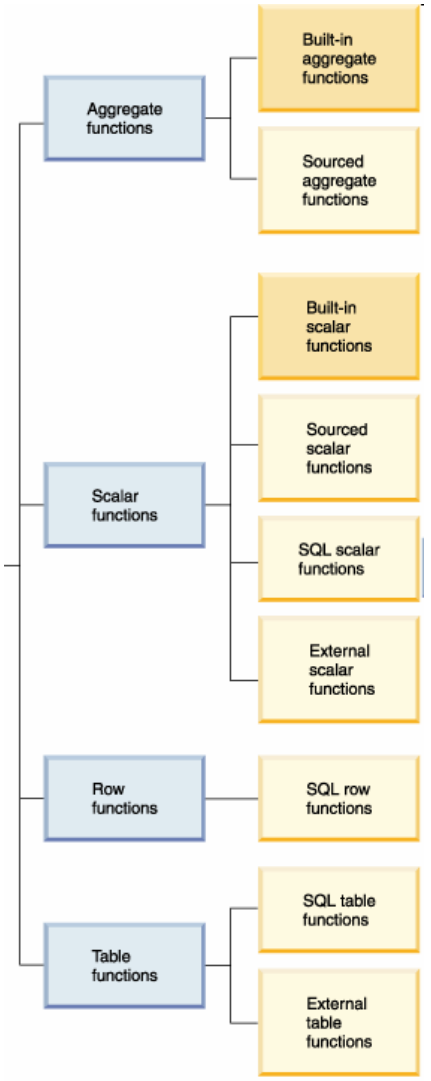
- **Vorteile**
  - räumliche Nähe von Daten und Anwendungslogik
  - reduzierter Netzwerkverkehr
  - Kapselung und Wiederverwendung
  - höheres Optimierungspotential für SQL-Anweisungen
  - Transaktionsschutz
- **Umsetzung**
  - nutzerdefinierte Funktionen (*user-defined functions*)
  - gespeicherte Prozeduren (*stored procedures*)
  - Trigger
- **Unterstützte Programmiersprachen**
  - SQL PL (*SQL procedural language*)
  - C
  - Java
  - Cobol
  - Visual Basic

Funktionen in DB2

Funktionen in DB2

- **Funktion**
  - Abbildung zw. Eingabewerten und Ausgabewerten
  - eingebaut oder nutzerdefiniert
  - können in SQL-Anweisungen verwendet werden

- **Arten**
  - Skalarfunktion (*scalar function*)
    - einzelne Werte → ein Wert (=Skalar)
  - Aggregationsfunktion (*aggregate function*)
    - auch Spaltenfunktion (*column function*)
    - Menge von Werten → ein Wert
  - Zeilenfunktion (*row function*)
    - … → Tupel
  - Tabellenfunktion (*table function*)
    - … → Tabelle



Skalarfunktionen

- **Skalarfunktionen**
  - Eingabe: feste Anzahl von Werten (Skalare)
  - Ausgabe: ein einziger Wert (Skalar)
  - **SELECT**-, **WHERE**-, **GROUP BY**-, **HAVING**-, **ORDER BY**-Klausel
- **Eingebaute Skalarfunktionen**
  - alle Typumwandlungsfunktionen, z.B. **INTEGER**(20.5)
  - Datumsfunktionen, z.B. **DAY**, **MONTH**, **YEAR**
  - Numerische Funktionen, z.B. **ABS**, **SQRT**, **SIN**
  - Zeichenkettenfunktionen, z.B. **LOWER**, **UPPER**, **CONCAT**, **SUBSTR**
  - Sonstige Funktionen, z.B. **COALESCE**
- **Beispiel**
  - **VALUES**(**CONCAT**('Hallo ','Welt'))
  - liefert `Hallo Welt`

Aggregationsfunktionen

- **Aggregationsfunktionen**
  - Eingabe: eine Menge von Werten (Spalte)
  - Ausgabe: ein einziger Wert (Skalar)
  - **SELECT**-, **HAVING**-, **ORDER BY**-Klausel

- **Eingebaute Aggregationsfunktionen**
  - Minimum (**MIN**), Maximum (**MAX**), Anzahl (**COUNT**), Durchschnitt (**AVG**), Summe (**SUM**)

- **Beispiel**
  - `SELECT AVG(SALARY) FROM EMPLOYEE`

```
1
-----------
27303,59
```

Zeilen- und Tabellenfunktionen

- **Zeilenfunktionen**
  - Eingabe: feste Anzahl von strukturierten Werten
  - Ausgabe: ein einzelnes Tupel (festes Schema)
  - nur im Kontext abstrakter Datentypen anwendbar
  - Zerlegung strukturierter Datentypen in ihre Bestandteile (*transform function*)

- **Tabellenfunktionen**
  - Eingabe: beliebig
  - Ausgabe: eine Tabelle (festes Schema)
  - **FROM**-Klausel
  - Anwendung: **FROM TABLE**(<fun-name>(<fun-arg>)) **AS** <name>

Nutzerdefinierte Funktionen

Nutzerdefinierte Funktionen

- **Nutzerdefinierte Funktionen** (*user-defined function, UDF*)
  - Erweiterung der Funktionalität („SQL-Erweiterung“)
  - Wiederverwendung, Verfügbarkeit, Performanz
  - automatisch generierte UDFs
    - einige bereits in SYSFUN und SYSPROC-Schema
    - bei der Erstellung von Distinct Types (2 Typumwandlung, 6 Vergleich)

- **Arten**
  - Sourced (oder Template)
    - Wrapper für andere UDFs
  - SQL Scalar/Table/Row
    - in SQL geschrieben (DB2-Dialekt)
  - External Scalar/Table
    - in höherer Programmiersprache geschrieben (C, Java, Cobol)
    - externe Bibliothek (registriert in DB2)
  - OLE DB External Table
    - Zugriff auf OLE DB-Provider

Nutzerdefinierte Funktionen

		Skalar	Zeile	Aggregation	Tabelle
Eingebaut		ja	-	ja	-
Nutzerdefiniert	Sourced	ja	-	ja	ja
	SQL	ja	ja	-	ja
	External	ja	-	-	ja
	OLE DB	-	-	-	ja

Nachteil gegenüber anderen DBMS

Sourced Functions

- **Sourced Functions**
  - basieren auf bestehenden Funktionen (= *source function*)
  - automatische Typkonvertierung durch DB2
  - Anwendung
    - vorhandende Funktionen mit nutzerdefinierten Datentypen
    - meist mit selben Namen (Polymorphie)

- **Syntax**

```
CREATE FUNCTION <name> ([<par-name>] <par-type>, …)
RETURNS <type>
[SPECIFIC <name>]
SOURCE <source-function>(<par-type>, …)
[AS TEMPLATE]
```

- **Optionen**
  - **SPECIFIC:** eindeutiger Name (z.B. für Löschen, nicht Aufruf)
  - **AS TEMPLATE:** Erzeugung eines Templates (föderierte Datenbanken)

Sourced Functions

- **Beispiel**
  - Definition

```
CREATE DISTINCT TYPE T_KM AS INTEGER WITH COMPARISONS

CREATE FUNCTION MAX(T_KM)
RETURNS T_KM
SPECIFIC KM_MAX
SOURCE SYSIBM.MAX(INTEGER)
```

  - Verwendung

```
SELECT MAX(MILEAGE)
FROM CARS
```

  - Entfernen

```
DROP FUNCTION MAX(T_KM)
```

    oder **DROP SPECIFIC FUNCTION** KM_MAX

UDFs in SQL

- **UDFs in SQL**
  - Skalar-, Zeilen- oder Tabellenfunktion
  - parametrisiert (nur Eingabe)

- **Syntax**

```
CREATE FUNCTION <name> ([<par-name>] <par-type>, …)
 RETURNS <type> |
         ROW(<name> <type>, …) |
         TABLE(<name> <type>, …)
[SPECIFIC <unique-name>]
[LANGUAGE SQL]
[[NOT] DETERMINISTIC]
[[NO] EXTERNAL ACTION]
[CONTAINS SQL | READS SQL DATA | MODIFIES SQL DATA]
[CALLED ON NULL INPUT]
 RETURN <sql-stmt> | <dynamic-compound-sql-stmt>
```

UDFs in SQL

- **Optionen der `CREATE FUNCTION`-Anweisung**
  - **`SPECIFIC`**: eindeutiger Name (z.B. für **`DROP FUNCTION`**)
  - **`DETERMINISTIC`**: gleiche Eingabewerte, gleiches Ergebnis (→ Cache)
  - **`EXTERNAL ACTION`**: Änderungsverhalten der Funktion (→ Cache)
  - **`CONTAINS SQL`**: weder lesen noch schreiben
  - **`READS SQL DATA`**: nur lesen
  - **`MODIFIES SQL DATA`**: lesen und schreiben
  - **`CALLED ON NULL INPUT`**: Funktion verarbeitet **`NULL`**-Argumente

UDFs in SQL

- **Beispiel** (Skalarfunktion)
  - Berechnung von π

```
CREATE FUNCTION PI()
RETURNS DOUBLE
DETERMINISTIC
CONTAINS SQL
RETURN ASIN(1)*2;

VALUES(PI());

 1
 -----------------
 3,141592653589793
```

UDFs in SQL

- **Beispiel** (Tabellenfunktion)
  - Durchschnittliches Gehalt nach Beruf & Geschlecht

```
CREATE FUNCTION AVGSAL(SEX VARCHAR(1))
RETURNS TABLE(JOB VARCHAR(8), AVGSAL DECIMAL(9,2))
DETERMINISTIC READS SQL DATA
RETURN SELECT JOB, AVG(SALARY) FROM EMPLOYEE
       WHERE SEX = AVGSAL.SEX
       GROUP BY JOB;
```

Bezug auf Argument

Bezug auf EMPLOYEE

```
SELECT * FROM TABLE(AVGSAL('F')) X;
 JOB       AVGSAL
 --------  ---------
 ANALYST   26110
 CLERK     22315
 DESIGNER  24476,66
 MANAGER   34723,33
 OPERATOR  21075
 PRES      52750
```

Compound SQL (dynamisch)

Compound SQL

- **Compound SQL**
  - Zusammenfassen mehrerer SQL-Anweisungen
  - 3 Arten
    - dynamisch (UDFs, JDBC, ...)
    - eingebettet (Embedded SQL, SQLJ)
    - prozedural (Stored Procedures)

- **Dynamisches Compound SQL**
  - dynamisch → dem DBMS vorher „unbekannt“
  - atomar (Alles-oder-Nichts-Prinzip)
  - als eine Anweisung kompiliert → gut falls wenig Logik, aber viele Daten
  - Syntax: **`BEGIN ATOMIC`** `[<stmt>;]*` **`END`**
  - unterstützte Funktionalität
    `SELECT`, `CALL`, `FOR`, `IF`, `GET DIAGNOSTICS`, `IF`, `INSERT`, `ITERATE`, `LEAVE`, `MERGE`, `DELETE` (searched), **`UPDATE`** (searched), **`SET`, `SIGNAL`, `WHILE`**

Besonderheiten für UDFs

- **Besonderheiten für UDFs**
  - Rückgabewert erforderlich: **`RETURN`**
  - keine Rekursion
  - kein Aufruf einer Funktion, die zu erstellende UDF verwendet
  - mit **`READ SQL DATA`**
    - Funktion kann nicht in DML-Operationen verwendet werden
    - Beispiel (nicht erlaubt!)
      `UPDATE EMPLOYEE SET SALARY = SALARY + BONUS(EMPNO)`
  - mit **`MODIFIES SQL DATA`**
    - aufgerufene Funktionen/Stored Procedures dürfen modifizierte Tabellen nicht lesen

- **Privilegien**
  - **`EXECUTE`**-Privileg (automatisch an Ersteller)
  - keine Privilegien für einzelne Operationen notwendig

Compound SQL: Variablen

- **Variablendeklaration**
  - immer am Anfang einer Compound SQL-Anweisung
  - Gültigkeitsbereich: aktueller Block und Unterblöcke
  - **DECLARE** <var-name> <data-type> [**DEFAULT** <value>]
- **Wertzuweisung**
  - **SET** <var-name> = <value> | (<select-stmt>)
  - **SELECT** <attribute>, … **INTO** <host-var>, … **FROM** …
    (nicht in dynamischem Compound SQL)

- **Abfrage**
  - innerhalb von Kontrollstrukturen
  - innerhalb von SQL-Anweisungen

- **Beispiel**
  - ```
    DECLARE NONATIONS INTEGER;
    SET NONATIONS = (SELECT COUNT(*) FROM NATION);
    ```
    ```

Compound SQL: Kontrollstrukturen

- **Verzweigung und Schleifen**
  - **IF** <expression> **THEN** …
    [**ELSIF** <expression> **THEN** …]*
    [**ELSE** …]
    END IF
  - **CASE**
    [**WHEN** <expression> **THEN** …]+
    [**ELSE** …]
    END CASE
  - **WHILE** <expression> **DO**
    …
    END WHILE
  - **REPEAT**
    …
    UNTIL <expression> **END REPEAT**
  - **FOR** <loop-name> **AS** <select-statement> **DO**
    …
    END FOR

- **Beispiel**

```
UPDATE COMMAND OPTIONS USING C OFF@

CREATE FUNCTION DEPTOF(EMPNO CHAR(6))
RETURNS VARCHAR(29)
BEGIN ATOMIC
    DECLARE WORKDEPT CHAR(3);

    SET WORKDEPT = (SELECT WORKDEPT FROM EMPLOYEE
                    WHERE EMPNO=DEPTOF.EMPNO);

    RETURN SELECT DEPTNAME FROM DEPARTMENT
           WHERE DEPTNO=WORKDEPT;
END@

VALUES(DEPTOF(CHAR('000050')))@

ROLLBACK@
```

liefert:

```
1
----------------
SUPPORT SERVICES
```

Entwickeln von UDFs

- **Arbeiten mit dem Befehlszeilenprozessor**
  - UDFs verwenden `;` als Anweisungstrennzeichen
  - Änderung des Trennzeichens des Kommandozeileninterpreters notwendig (hier @)
    - `db2 –td@ [-f <filename>]`
  - Tipp: Automatisches Commit des Interpreters abschalten
    - **`UPDATE COMMAND OPTIONS USING C OFF`**
    - **`COMMIT`**-Anweisung führt das Commit durch
    - **`ROLLBACK`**-Anweisung setzt alle Änderungen zurück
  - siehe Compound SQL-Beispiel

Zusammenfassung

- **Funktionen**
  - Skalarfunktionen
  - Aggregationsfunktionen
  - Zeilenfunktionen
  - Tabellenfunktionen

- **Nutzerdefinierte Funktionen**
  - Sourced Functions
  - UDFs in SQL
  - dynamisches Compound SQL

Alle Beispiele liegen unter: `/home/dbprog/examples/04`

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut Systemarchitektur, Professur Datenbanken

Datenbankprogrammierung
5. Stored Procedures

Stored Procedures

Stored Procedure

- **Stored Procedure (SP)**
  - in der Datenbank abgelegte Prozeduren
  - parametrisiert (Ein- und Ausgabe)
  - optional: **`INTEGER`**-Rückgabewert als Statuscode
  - SQL oder extern (Java, C, …)
  - Aufruf mittels **`CALL`**`(<arg>, …)`

- **Stored Procedures im Vergleich zu UDFs**
  - können nicht in SQL-Anweisungen/Triggern/Funktionen verwendet werden
  - Rückgabewert ist **`INTEGER`**, aber beliebige Ausgabewerte
    - DB2-Standarddatentypen
    - (mehrere!) Relationen
  - Länge limitiert auf 64kB Text
  - mächtiger, z.B. bessere Fehlerverarbeitung

Syntax

- **Syntax für SQL-basierte Stored Procedures**

```
CREATE PROCEDURE <name>
     (IN|OUT|INOUT <par-name> <par-type>, …)
[SPECIFIC <unique-name>]
[DYNAMIC RESULT SETS <number>]
[CONTAINS SQL | READS SQL DATA | MODIFIES SQL DATA]
[[NOT] DETERMINISTIC]
[CALLED ON NULL INPUT]
[LANGUAGE SQL]
<procedural-compound-sql-stmt>
```

- **Optionen**
  - **IN**: Eingabeparameter
  - **OUT**: Ausgabeparameter
  - **INOUT**: Ein- und Ausgabeparameter
  - **DYNAMIC RESULT SETS**: Anzahl der max. zurückgelieferten Ergebnismengen („Tabellen“)
  - restliche Parameter siehe UDFs

Beispiel

```
CREATE PROCEDURE GET_EMP_NAME (IN EMPNO CHAR(6),
                               OUT NAME CHAR(25))
BEGIN ATOMIC
    DECLARE FIRSTNME VARCHAR(12);
    DECLARE MIDINIT CHAR(1);
    DECLARE LASTNAME VARCHAR(15);

    SELECT FIRSTNME, MIDINIT, LASTNAME
           INTO FIRSTNME, MIDINIT, LASTNAME
    FROM EMPLOYEE
    WHERE EMPNO=GET_EMP_NAME.EMPNO;

    IF MIDINIT IS NULL OR MIDINIT = '' THEN
        SET NAME = FIRSTNME || ' ' || LASTNAME;
    ELSE
        SET NAME = FIRSTNME || ' ' || MIDINIT || '. '
                   || LASTNAME;
    END IF;
END
```

Beispiel

- **Beispielaufruf**
  - **CALL** GET_EMP_NAME('000010', ?)

```
Wert der Ausgabeparameter
-------------------------
Parametername: NAME
Parameterwert: CHRISTINE I. HAAS

Rückgabestatus                  = 0
```

Platzhalter für Argument

  - **CALL** GET_EMP_NAME('000120', ?)

```
Wert der Ausgabeparameter
-------------------------
Parametername: NAME
Parameterwert: SEAN O'CONNELL

Rückgabestatus                  = 0
```

Prozedurales Compound SQL

Prozedurales Compound SQL

- **Prozedurales Compound SQL**
  - (ausschließlich im) Rumpf einer Stored Procedure
  - vorkompiliert
  - mächtiger als dynamisches Compound SQL
    - `SQLCODE` und `SQLSTATE`: Abfragen von Warnungen/Fehlern
    - Cursor: tupelweiser Zugriff auf Relationen
    - Condition Handler: semi-automatische Fehlerbehandlung
  - Syntax: `BEGIN [ATOMIC] … END`
    - `ATOMIC`: atomare Ausführung (Alles-oder-Nichts-Prinzip)
    - sonst bei Fehlerfall kein Rücksetzen der Änderungen (!)

- **Reihenfolge von Deklarationen**
  - Variablen
  - Cursor
  - temporäre Tabellen

Cursor

- **Cursor**
  - zeilenweiser Zugriff auf SQL-Ergebnismenge
  - muss geöffnet und kann geschlossen werden
  - Definition
    - **DECLARE** <cur-name> **CURSOR**
      [**WITH HOLD**] [**WITH RETURN** [**TO CALLER**|**CLIENT**]]
      FOR <select-statement> | <stmt-name>
  - Optionen
    - **WITH HOLD:** nicht schließen durch **COMMIT** (explizites Schließen vor Verbindungsabbau notwendig!)
    - **WITH RETURN**: Cursor als Rückgabewert nutzen
  - Beispiel
    - **DECLARE** C1 **CURSOR FOR**
      SELECT LASTNAME, SALARY **FROM** EMPLOYEE

Cursor

- **Verwendung**
  - Öffnen eines Cursors
    - **OPEN** <cur-name>
  - Schließen eines Cursors
    - **CLOSE** <cur-name>
  - Daten aus Cursor lesen
    - muss geöffnet sein
    - setzt **SQLCODE**-Variable (100 wenn keine Tupel mehr)
      → ermöglicht Iteration
    - **FETCH** <cur-name> **INTO** <var-name> [, <var-name>]*
  - Beispiel
    - ```
      OPEN C1;
      FETCH C1 INTO name, salary;
      CLOSE C1;
      ```
      ```

Rückgabe von Ergebnismenge

- **Rückgabe von Ergebnismengen**
  - geöffneter Cursor mit der **`WITH RETURN`**-Option beim Beenden
  - Optionen
    - **`TO CALLER`**: Rückgabe an Aufrufer (Standardwert)
    - **`TO CLIENT`**: Rückgabe an Client (zwischengelagerte SP übersprungen)

- **Abfrage von Ergebnismengen**
  - erfordert Definition von Locator-Variablen
  - werden den zurückgelieferten Ergebnismengen assoziiert
  - anschließend Allokation als Cursor allokiert (kein **`OPEN`** mehr notwendig)
  - Beispiel

```
DECLARE RSL RESULT_SET_LOCATOR VARYING
…
CALL TEST(…)
ASSOCIATE RESULT SET LOCATORS (RSL) WITH PROCEDURE test
ALLOCATE C1 CURSOR FOR RESULT SET RSL
FETCH FROM C1 INTO …
```

Fehlernachrichten

- **Fehlernachrichten**
  - 2 Variablen: **`SQLCODE`** und **`SQLSTATE`**
    - werden nach jeder SQL-Anweisung gesetzt
    - müssen deklariert werden, um sie abzufragen
      - `DECLARE SQLCODE INTEGER DEFAULT 0`
      - `DECLARE SQLSTATE CHAR(5) DEFAULT '00000'`
  - **`SQLCODE`**
    - nach ISO/ANSI SQL-Standard
    - =0: erfolgreiche Ausführung
    - =100: keine Daten gefunden (vgl. **`FETCH`**)
    - >0 & ≠100: erfolgreiche Ausführung mit Warnungen
    - <0: Fehler
  - **`SQLSTATE`**
    - genauere Fehlerbeschreibung
    - siehe DB2 Message Reference (erhältlich unter http://www-306.ibm.com/software/data/db2/udb/support/manualsv9.html oder unter http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.msg.doc/doc/r0sttmsg.htm)

Beispiel

```
CREATE PROCEDURE SALARY_OF_DEP(IN DEPNO CHAR(3),
                               OUT SALARY DECIMAL(9,2))
BEGIN
     DECLARE SQLCODE INTEGER DEFAULT 0;
     DECLARE result DECIMAL(9,2) DEFAULT 0.00;
     DECLARE x DECIMAL(9,2);
     DECLARE cursor1 CURSOR FOR
             SELECT SALARY FROM EMPLOYEE WHERE WORKDEPT = DEPNO;

     OPEN cursor1;
     FETCH cursor1 INTO x;
     WHILE SQLCODE <> 100 DO
             SET result = result + x;
             FETCH cursor1 INTO x;
     END WHILE;
     CLOSE cursor1;

     SET SALARY = result;
END
```

```
db2 => CALL salary_of_dep('A00', ?)

  Value of output parameters

  --------------------------

  Parameter Name  : SALARY

  Parameter Value : 354250,00


  Return Status = 0
```

Fehlerbehandlung

- **Condition Handler**
  - Definition des Verhaltens einer Stored Procedure beim Auftreten eines Fehlers
  - **DECLARE** <handler-type> **HANDLER FOR** <condition-list> **BEGIN** … **END**
  - Typ des Handlers
    - **CONTINUE**: nach Abarbeitung des Handlers fortsetzen
    - **EXIT**: nach Abarbeitung des Handlers beenden
    - **UNDO**: vor Ausführung des Handlers alle Aktionen rückgängig machen und anschließend beenden. Nur bei Verwendung von **ATOMIC**.
  - Bedingungen
    - **NOT FOUND**: alle Ereignisse mit **SQLCODE = 100**
    - **SQLEXCEPTION**: alle Ereignisse mit **SQLCODE < 0**
    - **SQLWARNING**: alle anderen Ereignisse
    - **SQLSTATE** '<state>': bestimmtes Ereignis

Fehlerbehandlung

- **SIGNAL und RESIGNAL**
  - Erzeugen/Weiterleiten von SQLSTATE-Fehlersignalen
  - `[RE]SIGNAL SQLSTATE '<state>'`
    `[SET MESSAGE_TEXT = '<text>'];`

- **Beispiel**
  - SQL-Ausnahmen auf anwendungsspezifische Fehlermeldung umleiten
  - ```
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
       RESIGNAL SQLSTATE '80000'
           SET MESSAGE_TEXT = 'Got an SQL exception.';
    END
    ```
    ```

Dynamisches SQL

Statisch vs. dynamisch

- **Statisches SQL**
  - feste SQL-Anweisungen, vorher bekannt
  - Parametrierung durch Variablen zulässig
  - Performanz
    - vorkompiliert und optimiert durch das DBMS → kein Overhead
    - aber:
      - sensitiv zu Änderungen der Datenverteilung (erfordert erneutes Binden)
      - Probleme mit nicht gleichverteilten Attributen (*skew*): Index oder Tablescan?

- **Dynamisches SQL**
  - dynamisch generierte SQL-Anweisungen
  - typischerweise langsamer (Kompilierung!)
  - DB2 interner Anweisungscache

Dynamisches SQL

- **Kompilierung**
  - kostet Zeit, aber komplilierte Anweisung kann mehrfach verwendet werden
  - Syntax
    - **`PREPARE`** `<stmt-name>` **`FROM`** `<var-name>`
    - `<stmt-name>` vergibt Namen für SQL-Anweisung
    - `<var-name>` ist eine vollständige SQL-Anweisung
  - keine Referenzen auf Hostvariablen, aber Parametermarker

- **Parametermarker**
  - können an allen Stellen auftauchen, an denen auch Hostvariablen auftauchen können (falls typisiert)
  - 2 Arten
    - typisiert: **`CAST`**`(?` **`AS`** `<type>)`
    - untypisiert: `?`
  - auch hier: Probleme mit nicht-gleichverteilten Attributen

Dynamisches SQL

- **Ausführung**
  - Cursor für die angegebene Anweisung erstellen (**SELECT**)
    - max. ein Cursor pro Anweisung
    - Anweisung muss vor **OPEN** kompiliert worden sein
    - Syntax
      - **DECLARE** <cur-name> **CURSOR FOR** <stmt-name>
      - **OPEN** <cur-name> [**USING** <argument>, …]
    - **USING**: ersetzt Parametermarker (Reihenfolge beachten!)
  - Anweisung ausführen (≠**SELECT**)
    - Nach Kompilierung
      - **EXECUTE** <stmt-name> [**INTO** <arg>,…] [**USING** <arg>, …]
      - **INTO**: Ausgabeparameter
      - **USING**: Belegung von Parametermarkern / Eingabeparameter
    - Vor Kompilierung
      - **EXECUTE IMMEDIATE** <var-name>
      - entspricht **PREPARE** und **EXECUTE**
      - keine Parametermarker

Komplexes Beispiel

```
CREATE PROCEDURE DROP_IF_LARGER (IN TABLENAME VARCHAR(20),
                                 IN MAXSIZE INT,
                                OUT ACTION_PERFORMED VARCHAR(100))
BEGIN ATOMIC
   DECLARE SQLCODE INT DEFAULT 0;
   DECLARE SIZE INT DEFAULT -1;
   DECLARE SQL VARCHAR(200);
   DECLARE C1 CURSOR FOR STMT;
   DECLARE UNDO HANDLER FOR SQLEXCEPTION BEGIN
      SET ACTION_PERFORMED = 'Ein Fehler ist aufgetreten.';
   END;

   SET SQL = 'SELECT COUNT(*) FROM ' || TABLENAME;
   PREPARE STMT FROM SQL;
   OPEN C1;
   FETCH C1 INTO SIZE;
   CLOSE C1;

   IF (SIZE <= MAXSIZE) THEN
      SET ACTION_PERFORMED = 'Tabelle wird nicht gelöscht.';
   ELSE
      SET SQL = 'DROP TABLE ' || TABLENAME;
      EXECUTE IMMEDIATE SQL;
      SET ACTION_PERFORMED = 'Tabelle gelöscht.';
   END IF;
END
```

Temporäre Tabellen

Temporäre Tabellen

- **Temporäre Tabellen** (*temporary tables*)
  - Vorteile
    - nehmen Zwischenergebnisse auf
    - verhindern mehrfache Berechnungen der gleichen Anfragen
  - Eigenschaften
    - Tabelle mit optionalen Indizes und Constraints
    - nur von aktueller Sitzung aus zugreifbar
    - automatisch bei Verbindungsabbau gelöscht
    - sofern möglich im Hauptspeicher gehalten
  - oft innerhalb von Prozeduren verwendet

- **Voraussetzung**
  - temporärer Nutzer-Tabellenbereich (*user temporary tablespace*)
  - `CREATE USER TEMPORARY TABLESPACE usertmp1 MANAGED BY AUTOMATIC STORAGE`

Temporäre Tabellen

- **Erstellung**
  - automatisch in sitzungsbezogenem Schema `SESSION`
  - **`DECLARE GLOBAL TEMPORARY TABLE`** `<name>` (
    `<column-definition>`
) [**`WITH REPLACE`**]
    [**`ON COMMIT`** [**`DELETE ROWS`** | **`PRESERVE ROWS`**]]
    [**`IN`** `<tablespace>`]

- **Optionen**
  - **`WITH REPLACE`**: Tabelle ersetzen, falls schon vorhanden
  - Verhalten nach Commit: **`ON COMMIT`**
    - **`DELETE ROWS`**: Löschen aller Tupel (Standardwert)
    - **`PRESERVE ROWS`**: Behalten aller Tupel
  - **`IN`** `<tablespace>`: in angegebenen Tabellenbereich erzeugen

Temporäre Tabellen

- **Beispiel** (ohne automatisches Commit)
  - `DECLARE GLOBAL TEMPORARY TABLE MYTEMP (ID INT);`
  - `INSERT INTO SESSION.MYTEMP VALUES (1);`
  - `SELECT * FROM SESSION.MYTEMP;`

```
ID
------
     1
```

  - `DROP TABLE SESSION.MYTEMP;`
  - `COMMIT;`

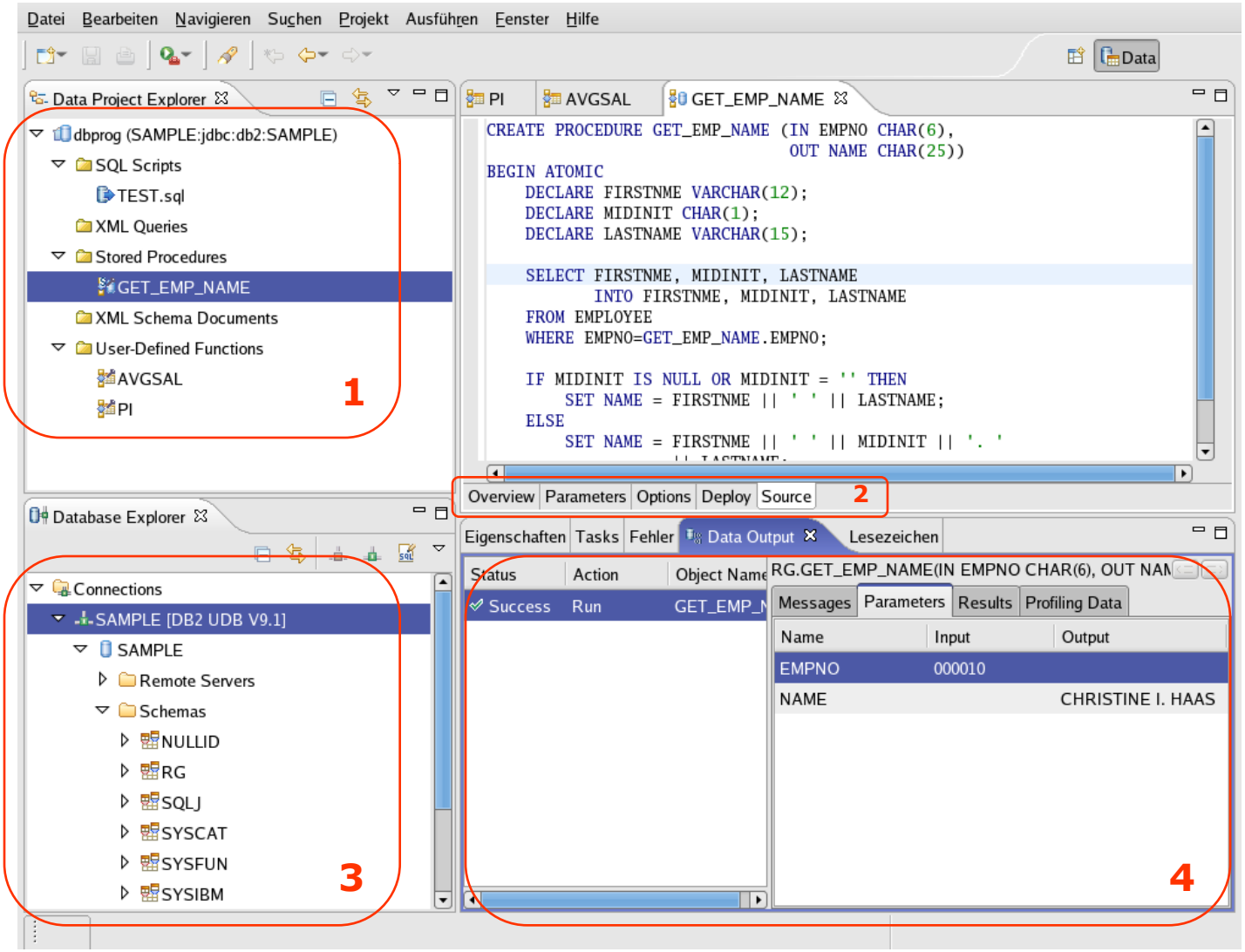
Entwicklungswerkzeuge

DB2 Developer Workbench

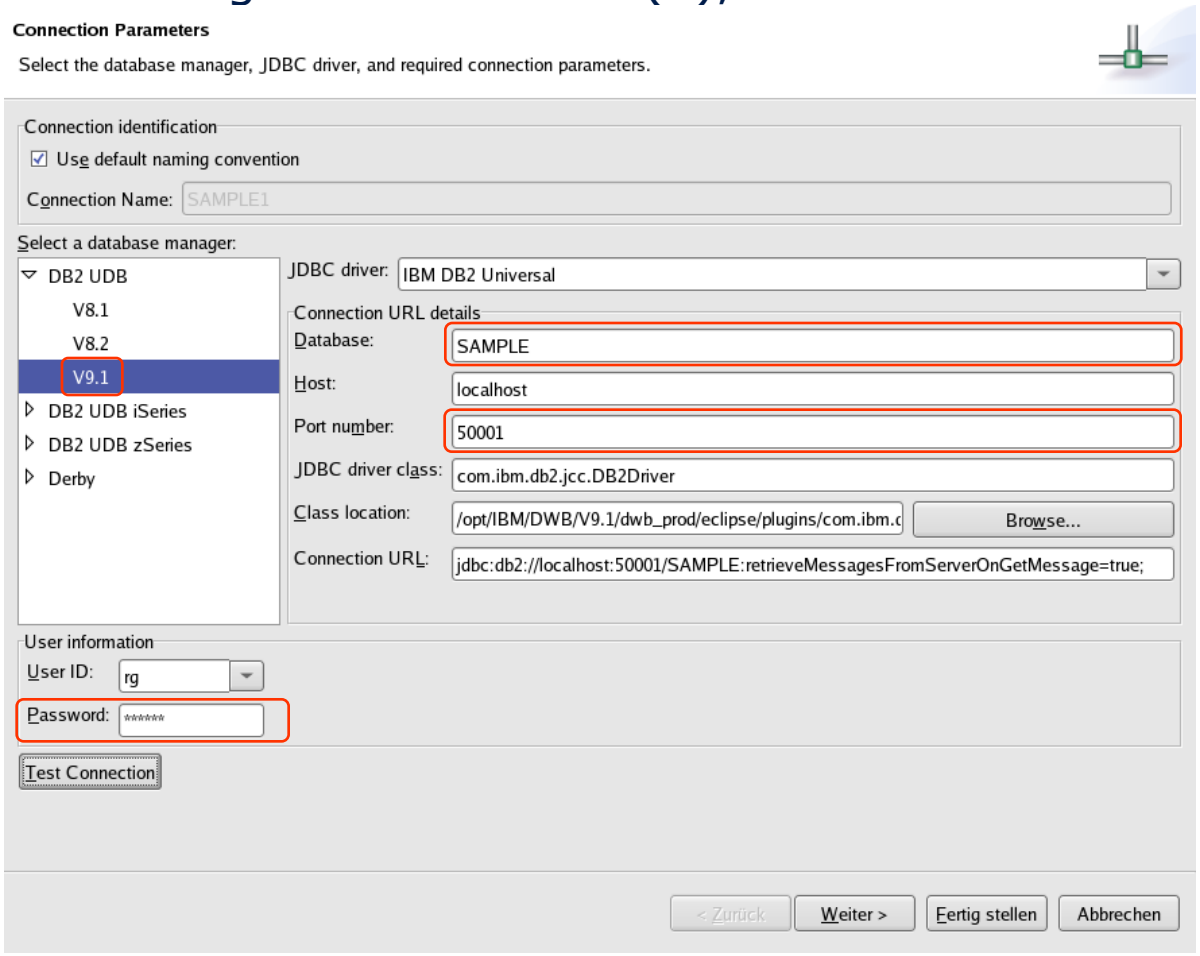
- **Entwicklungswerkzeuge in DB2**
  - DB2 V7: Stored Procedure Builder
  - DB2 V8: Entwicklungszentrale (*development center*)
  - DB2 V9: Developer Workbench

- **DB2 Developer Workbench**
  - Eclipse-basierte IDE
    - Datenbankobjekte anlegen/bearbeiten
    - Visueller Editor für SQL/XQuery-Anfragen
    - Entwicklung/Debugging von UDFs und Stored Procedures
    - Java-Unterstützung (Stored Procedures, SQLJ)
    - Arbeiten mit Massendaten (**`LOAD`**, **`EXPORT`**)
  - starten mittels: `db2dwb`
  - Arbeitsbereiche (*workspace*) und Projekte

DB2 Developer Workbench

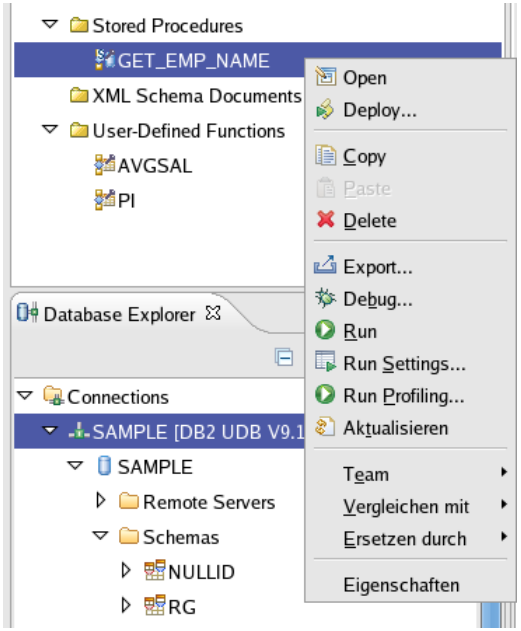


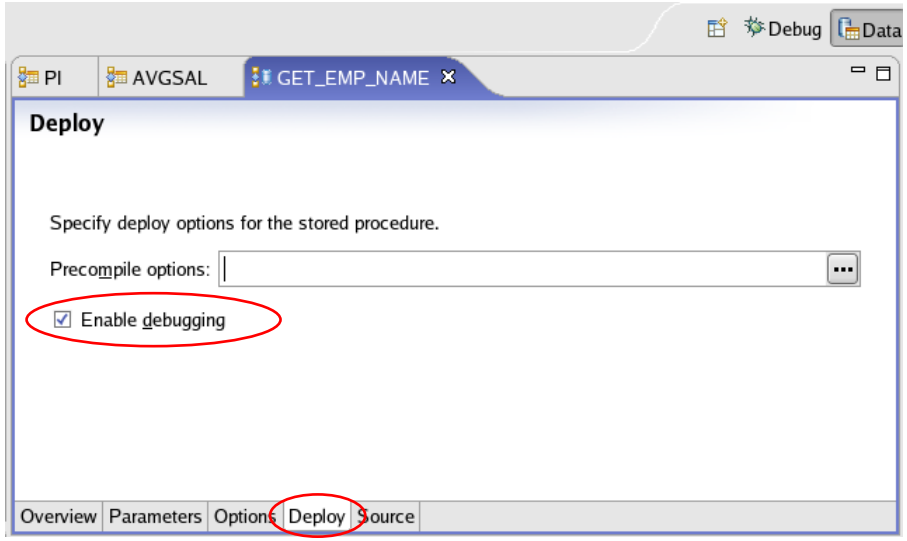
- **Verbindungen anlegen**
  - Neue Verbindung: Rechtsklick in (3), dann “New Connection”



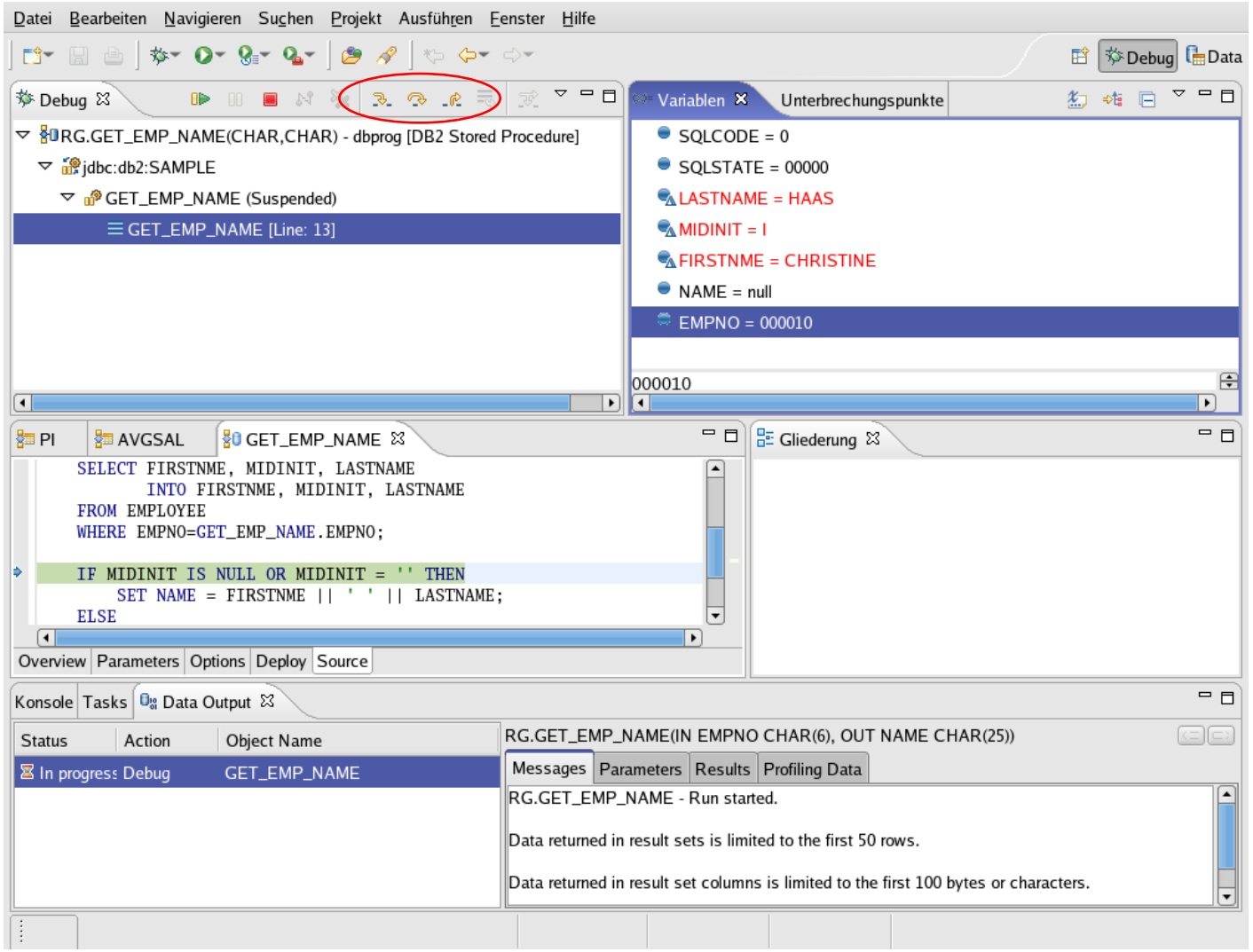
DB2 Developer Workbench

- **Bearbeiten/Ausführen/Debugging**
  - Rechtsklick in (1)
- **Debugging aktivieren**
  - Klick auf Deploy in (2)
  - "Enable debugging" aktivieren





DB2 Developer Workbench



Zusammenfassung

- **Stored Procedures**
  - Ein- und Ausgabeparameter
  - Rückgabe von (mehreren) Relationen möglich
  - prozedurales Compound SQL (Cursor, Fehlerbehandlung)
- **Dynamisches SQL**
  - Vor- und Nachteile (gegenüber statischem SQL)
- **Temporäre Tabellen**
  - Zwischenspeichern von Teilergebnissen
- **DB2 Developer Workbench**
  - Eclipse-basierte IDE

Alle Beispiele liegen unter: `/home/dbprog/examples/05`

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut Systemarchitektur, Professur Datenbanken

Datenbankprogrammierung
6. Trigger

Trigger

- **Trigger**
  - durch DML-Operationen ausgelöste Aktionsfolge (**UPDATE, DELETE, INSERT**)
  - in dynamischen Compound SQL geschrieben
  - Verwendung
    - Einhalten von Geschäftsregeln
    - Sicherstellen von Datenintegrität (Erweiterungen der **CHECK**-Constraints)
    - Propagierung von Updates
    - Automatisches Generieren abgeleiteter Werte
    - Auslösen von beliebigen (externen) Aktionen
  - mehrere Trigger pro Tabelle/Sicht möglich
  - sofort nach Definition aktiv
  - Definition in Katalogtabellen (Sichten **SYSCAT.TRIGGERS**, **SYSCAT.TRIGDEPT**)

ECA-Konzept

ECA-Konzept

- **ECA-Konzept**
  - Ereignis (e*vent*)
    - auslösende DML-Anweisung (**`INSERT, UPDATE, DELETE`**)
  - Bedingung (*condition)*
    - Trigger wird nur ausgelöst, wenn Bedingung erfüllt
    - optional
  - Aktion (*action*)
    - durchzuführende Aktion in Compound SQL
    - Einbinden komplexer Logik durch UDFs und Stored Procedures
  - Triggerkaskadierung
    - Ausführung eines Triggers kann andere Trigger auslösen
    - ACHTUNG: Kettenreaktionen möglich

Anlegen von Triggern

- **Anlegen eines Triggers**

```
CREATE TRIGGER <trigger-name>
BEFORE | AFTER | INSTEAD OF
INSERT | DELETE | UPDATE [OF (<column>, …)]
ON <table-name>
REFERENCING [OLD AS <name>] [NEW AS <name>]
            [OLD_TABLE AS <name>] [NEW_TABLE AS <name>]]
FOR EACH ROW | FOR EACH STATEMENT
MODE DB2SQL
[WHEN (<condition>)]
BEGIN [ATOMIC]
 <sql-stmt> | <dynamic-compound-sql-stmt>
END
```

Auslösezeitpunkt

Auslöser

Übergangs-variablen

Granularität

Bedingung

Aktion

- **Entfernen von Triggern**
  - **DROP TRIGGER** <trigger-name>

Auslösezeitpunkt

- **Auslösezeitpunkt (`BEFORE`)**
  - vor der Ausführung des DML-Befehls
  - keine Modifikation (außer aktuelles Tupel) erlaubt
  - keine kaskadierenden Trigger möglich
  - Einsatz
    - Überprüfung von Integritätsbedingungen
    - Automatische Generierung von Werten (im aktuellen Tupel)
  - **`NO CASCADE`** keine anderen Trigger aktiviert

Auslösezeitpunkt

- **Beispiel** (**BEFORE**-Trigger)
  - bringt Anfang und Ende einer Unterrichtsstunde in chronologische Reihenfolge

```
CREATE TRIGGER CLASS_END
BEFORE INSERT ON CL_SCHED
REFERENCING NEW AS new_row
FOR EACH ROW
MODE DB2SQL
BEGIN ATOMIC
 DECLARE t TIME;
   IF new_row.STARTING > new_row.ENDING THEN
     SET t = new_row.ENDING;
     SET new_row.ENDING = new_row.STARTING;
     SET new_row.STARTING = t;
   END IF;
END@
```

Auslösezeitpunkt

- **Auslösezeitpunkt (AFTER)**
  - nach der Ausführung des DML-Befehls
  - Ausführung nach Überprüfung bereits existierender Constraints
  - Automatische Generierung von Werten (in anderen Tupeln/ Relationen)

- **Beispiel** (**AFTER**-Trigger)
  - Pflegen der Firmenstatistik nach Einfügen eines neuer Mitarbeiters

```
CREATE TRIGGER NEW_HIRE
AFTER INSERT ON EMPLOYEE
FOR EACH ROW
MODE DB2SQL
UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
```

Auslösezeitpunkt

- **Auslösezeitpunkt (`INSTEAD OF`)**
  - anstelle der Ausführung des DML-Befehls
  - nur für Sichten anwendbar
  - keine Bedingungen erlaubt (**`WHEN`**)
  - nur ein **`INSTEAD OF`**-Trigger pro Ereignis und Sicht
  - Definition der Aktualisierungssemantik für Sichten

Auslösezeitpunkt

- **Beispiel** (**INSTEAD OF**-Trigger)
  - Löschen in Sichten

```
DELETE FROM v1
WHERE A=1 AND B=2   → FEHLER

CREATE TRIGGER V1_DELETE
INSTEAD OF DELETE ON v1
REFERENCING OLD AS o
FOR EACH ROW
MODE DB2SQL
DELETE FROM t1 WHERE o.A = A
    AND o.B = B

DELETE FROM v1
WHERE A=1 AND B=2 → OKAY
```

View v1

A	B	C
1	2	8
2	3	9

Tabelle t1

A	B
1	2
2	3

Tabelle t2

B	C
2	8
3	9

A	B
2	3

Granularität

- **Granularität**
  - **FOR EACH ROW**
    - pro bearbeitetes Tupel
    - n-mal ausgeführt, bevor/nachdem n Tupel abgearbeitet
  - **FOR EACH STATEMENT**
    - pro DML-Anweisung
    - einmal ausgeführt nachdem Anweisung abgearbeitet
    - nur für **AFTER**-Trigger
    - bei **UPDATE** und **DELETE**: auch aktiviert falls kein Tupel verändert/gelöscht
- **Beispiel**
  - Pflege der Firmenstatistik nach jedem Einfügen eines neuen Mitarbeiters

```
CREATE TRIGGER NEW_HIRED
AFTER INSERT ON EMPLOYEE
FOR EACH ROW
UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
```

Granularität

- **Beispiel**
  - Pflegen der Firmenstatistik nach Einfügen einer Menge neuer Mitarbeiter

```
CREATE TRIGGER NEW_HIRED
AFTER INSERT ON EMPLOYEE
REFERENCING NEW_TABLE AS NEWEMPS
FOR EACH STATEMENT
UPDATE COMPANY_STATS
SET NBEMP = (SELECT COUNT(*) FROM NEWEMPS) ;
```

Übergangsvariablen/-tabellen, Bedingungen

- **Übergangsvariable** (*transition variable*)
  - **`OLD AS`** `<alias>` bzw. **`NEW AS`** `<alias>`
  - betroffenes Tupel vor bzw. nach Ausführung der Operation
- **Übergangstabelle** (*transition table*)
  - **`OLD_TABLE AS`** `<alias>` bzw. **`NEW_TABLE AS`** `<alias>`
  - betroffene Tabelle vor bzw. nach Ausführung der Operation
- **Bedingungen**
  - optionale Bedingung für Ausführung, ähnlich **`WHERE`**-Klausel
  - Nutzung von Übergangsvariablen

Übergangsvariablen und -tabellen

Zeitpunkt	Auslösendes Ereignis	FOR EACH ROW	FOR EACH STATEMENT
BEFORE	INSERT	• NEW • -	-
	UPDATE	• OLD,NEW • -	-
	DELETE	• OLD • -	-
AFTER	INSERT	• NEW • NEW_TABLE	• - • NEW_TABLE
	UPDATE	• OLD,NEW • OLD_TABLE,NEW_TABLE	• - • OLD_TABLE,NEW_TABLE
	DELETE	• OLD • OLD_TABLE	• - • OLD_TABLE

Übergangsvariablen und -tabellen

- **Beispiel**
  - Verhinderung von Gehaltserhöhungen um mehr als 20%

```
CREATE TRIGGER SAL_CHECK
AFTER UPDATE OF SALARY ON EMPLOYEE
REFERENCING OLD AS OLD_EMP
            NEW AS NEW_EMP
FOR EACH ROW
MODE DB2SQL
WHEN (NEW_EMP.SALARY > (OLD_EMP.SALARY * 1.20))
SIGNAL SQLSTATE '80000' ('Keine Gehaltserhöhung über
                                        20% erlaubt.');
```

Aufrufreihenfolge

- **Aufrufreihenfolge**
  - mehrere Trigger gleichen Typs: Ausführung in Reihenfolge des Erstellens
  - Aufrufreihenfolge kann Ergebnis beeinflussen
  - Zeit der Erstellung **CREATE_TIME** in **SYSIBM.SYSTRIGGERS** hinterlegt

- **Beispiel**
  - Trigger A addiert 100 Euro zum Gehalt
  - Trigger B multipliziert das Gehalt mit 1.1

Reihenfolge: A → B (1000 + 100) * 1.1 = 1210
Reihenfolge: B → A (1000 * 1.1) + 100 = 1200 ≠

Zusammenfassung

- **Trigger**
  - ereignisgesteuerte Anwendungslogik (ECA-Konzept)
- **Aktivierungsarten**
  - **BEFORE, AFTER** oder **INSTEAD OF**
- **Granularität**
  - **ROW** oder **STATEMENT**
- **Übergangsvariablen**
  - vor oder nach Änderungen
  - Tupel oder temporäre Tabelle

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut Systemarchitektur, Professur Datenbanken

Datenbankprogrammierung
7. Java Database Connectivity

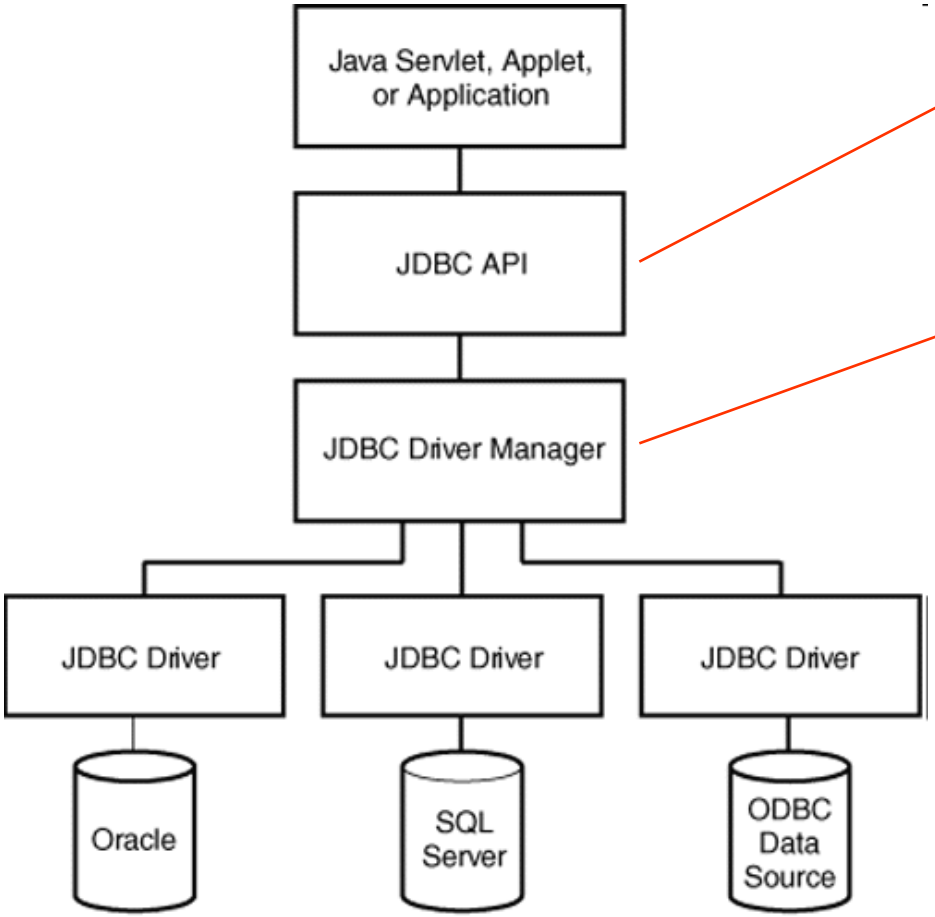
Java Database Connectivity

- **Java Database Connectivity (JDBC)**
  - Industriestandard von Sun Microsystems
  - aktuelle Version: JDBC 3.0 (2002)
  - ermöglicht Kommunikation zwischen Java-Anwendungen und Datenbanken
  - unabhängig vom verwendeten Datenbanksystem (*write once, run anywhere*)
  - tabellenbasiert (hauptsächlich für relationale DBMS)

- **Grundlegender Funktionsumfang**
  - Datenbankverbindung aufbauen
  - SQL-Anweisungen absetzen
  - Ergebnisse bearbeiten
  - Metadaten ermitteln

JDBC-Architektur

JDBC-Architektur



Pakete:
`java.sql`
`javax.sql`

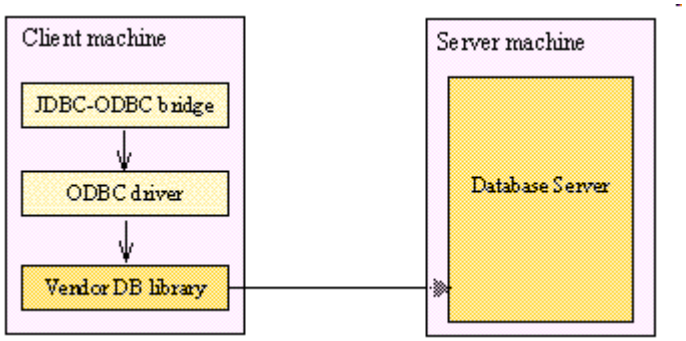
Verwaltung von JDBC-Treibern,
Auswahl des korrekten Treibers
→ Datenbankunabhängigkeit

Übersetzung von JDBC-Aufrufen
in spez. Datenbankprotokoll
→ 4 Typen

Quelle: http://www.awprofessional.com

Typ 1-Treiber

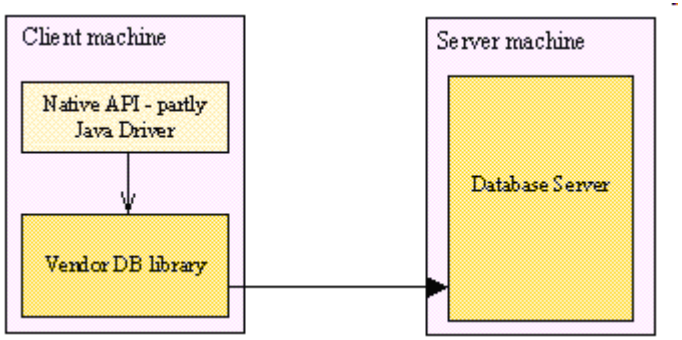
- **Typ 1: JDBC-ODBC Bridge**
  - übersetzt JDBC-Aufrufe in ODBC
  - mit fast allen Datenbanksystemen verwendbar
  - Datenbankbibliothek auf Client erforderlich
  - langsam



Quelle: http://www.javaworld.com

Typ 2-Treiber

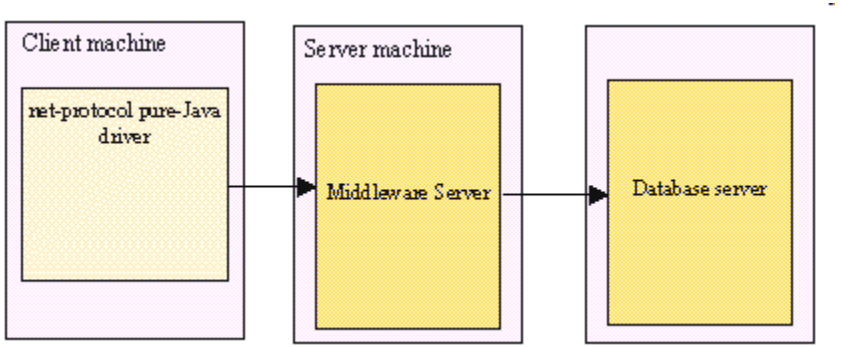
- **Typ 2: Native-API/partly Java driver**
  - JDBC-Treiber verwendet datenbankspezifische API
  - Datenbankbibliothek auf Client erforderlich
  - schneller als Typ 1



Quelle: http://www.javaworld.com

Typ 3-Treiber

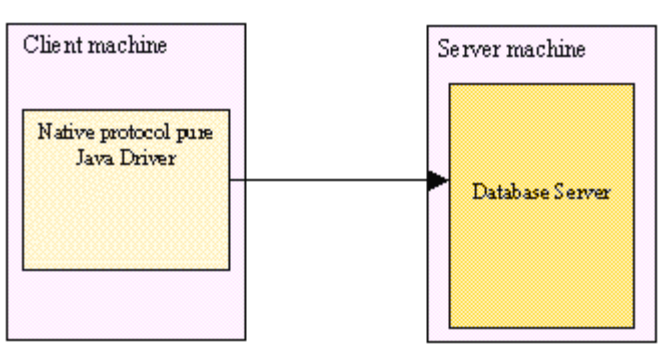
- **Typ 3: Net-protocol/all-Java driver**
  - 3-Schichten
    - JDBC-Aufrufe werden an Middleware-Server weitergeleitet
    - Middleware verwendet natives Datenbankprotokoll
  - kein Binärcode auf Client, aber datenbankspezifische Middleware
  - hohes Optimierungspotential (Caching, Lastbalancierung)



Quelle: http://www.javaworld.com

Typ 4-Treiber

- **Typ 4: Native-protocol/all-Java driver**
  - direkte Kommunikation mit der Datenbank
  - javabasierte Treiber (→ plattformunabhängig)
  - schnell



Quelle: http://www.javaworld.com

DB2-JDBC-Treiber

- **DB2-JDBC-Treiber**
  - CLI-basierter JDBC-Treiber
    - Typ 2
      - Klasse: `COM.ibm.db2.jdbc.app.DB2Driver`
      - bezeichnet als *app driver*
    - *Typ 3*
      - Klasse: `COM.ibm.db2.jdbc.net.DB2Driver`
      - bezeichnet als *net driver*
      - veraltet seit DB2 V8
  - DB2 Universal JDBC-Treiber
    - auch als JCC-Treiber bezeichnet (*Java Common Connectivity*)
    - Typ 2 und Typ 4
    - Klasse: `com.ibm.db2.jcc.DB2Driver`
    - verwendet DRDA (*Distributed Relational Database Architecture*)
    - Bibliotheken: `db2jcc.jar` und `db2jcc_license_cu.jar` in `sqllib/java`

JDBC-API

JDBC-API

- **Pakete**
  - `java.sql`: Zugriff auf Datenquellen
  - `javax.sql`: serverseitiger Datenzugriff

- **Wichtige Klassen/Interfaces**
  - `Connection`: Verbindung zu einer Datenquelle
  - `Statement`: Abarbeiten von SQL-Anweisungen
  - `PreparedStatement`: vorkompilierte SQL-Anweisungen
  - `CallableStatement`: vorkompilierte Stored Procedure-Aufrufe
  - `ResultSet`: Abfrage von Ergebnissen
  - `SQLWarning`: Informationen über (unkritische) Warnungen
  - `SQLException`: Informationen im Fehlerfall

Verbindungsaufbau

- **Verbindung**
  - über `java.sql.DriverManager` (hier) oder `javax.sql.DataSource`
  - Aufbau

```
// Treiber laden
Class.forName(<driver>);

// Verbindung aufbauen
Connection con = DriverManager.getConnection(
                 <url> [, <username>, <password>]);
…
// Verbindung schließen
con.close();
```

- **JDBC-URL**
  - beschreibt Datenbank
  - grundlegender Aufbau
    - `jdbc:<subprotocol>:<subname>`
    - mehrere Treiber für eine URL prinzipiell möglich
  - DB2
    - Typ 2 : `jdbc:db2:<db-name>`
    - Typ 3/4: `jdbc:db2://<host>:<port>/<db-name>`

Connection-Objekt

- **Wichtige Methoden des `Connection`-Interfaces**
  - `void setAutoCommit(boolean autoCommit)`
    - Ab-/Anschalten des automatischen Commits
  - `void commit() / void rollback()`
    - Festschreiben/Zurücksetzen aller vorgenommenen Änderungen (falls kein Auto-Commit)
  - `Statement createStatement()`
    - Erzeugen eines `Statement`-Objekts zur Ausführung von SQL-Anweisungen
  - `PreparedStatement prepareStatement(String sql)`
    - Erzeugen einer vorkompilierten SQL-Anweisung
  - `CallableStatement prepareCall(String sql)`
    - Erzeugen eines vorkompilierten Stored Procedure-Aufrufs
  - `DatabaseMetaData getMetaData()`
    - liefert Metadaten zur aktuellen Verbindung

Beispiel

```
try {
    Class.forName("com.ibm.db2.jcc.DB2Driver");
    Connection con =
            DriverManager.getConnection("jdbc:db2:sample");
    DatabaseMetaData dbmd = con.getMetaData();
    System.out.println("Datenbanksystem: "
            + dbmd.getDatabaseProductName());
    System.out.println("Version          : "
            + dbmd.getDatabaseMajorVersion() + "."
            + dbmd.getDatabaseMinorVersion());
    con.close();
} catch (ClassNotFoundException cnfe) {
    System.err.println("Error loading JDBC driver");
} catch (SQLException se) {
    System.err.println("Error connecting to database");
}
```

- **Ausgabe**

```
Datenbanksystem: DB2/LINUX
Version          : 9.1
```

Statement-Objekt

- **`Statement`-Objekt**
  - Ausführen eines Statements & Abfragen des Ergebnis
  - erzeugt mittels `Connection.createStatement()`
  - explizites Schließen notwendig (mit `close()`)
  - Achtung: Ausführung einer SQL-Anweisung verwirft bisherige Ergebnisse → ggf. mehrere Instanzen notwendig
  - Ergebnis(se): Änderungszähler oder `ResultSet`

- **Wichtige Methoden**
  - `ResultSet executeQuery(String sql)`
    - Ausführen einer DQL-Anweisung
  - `int executeUpdate(String sql)`
    - Ausführung einer DDL/DML/DCL-Anweisung
  - `boolean execute(String sql)`
    - Ausführen einer beliebigen SQL-Anweisung
  - `int getUpdateCount()`
    - Liefert aktuelles Ergebnis, falls Änderungszähler (sonst -1)
  - `ResultSet getResultSet()`
    - Liefert aktuelles Ergebnis, falls `ResultSet` (sonst `null`)
  - `boolean getMoreResults()`
    - aktuelles Ergebnis verwerfen, auf nächstes Ergebnis weiterrücken

ResultSet-Objekt

- **`ResultSet`-Objekt**
  - repräsentiert Anfrageergebnis (Tabelle)
  - Grundprinzip: Cursor (auch: Iterator)
    - Zugriff nur auf aktuelle Zeile
    - `boolean next()`
      - positioniert Cursor auf nächste Zeile
      - ermittelt ob weitere Tupel verfügbar
  - explizites Schließen notwendig (mit `close()`)
  - Arten
    - *forward only*: sequentielle Abarbeitung
    - *scrollable*: freie Positionierung des Cursors (aufwendiger!)

- **Wichtige Methoden**
  - `get*(int) / get*(String)`
    - liefert Wert eines Attributes (1-basierter Index oder Name)
  - `update*(int, *) / update*(String, *)`
    - ändert Wert eines Attributes (1-basierter Index oder Name)
  - `ResultSetMetaData getMetaData()`
    - Metadaten über das Anfrageergebnis

Beispiel

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(
            "SELECT LASTNAME FROM EMPLOYEE");
while (rs.next()) {
    System.out.print(rs.getString("LASTNAME") + "; ");
}
rs.close();
stmt.close();
```

- **Ausgabe**

```
HAAS; THOMPSON; KWAN; GEYER; STERN; PULASKI; HENDERSON;
SPENSER; LUCCHESSI; O'CONNELL; QUINTANA; NICHOLLS; ADAMSON;
PIANKA; YOSHIMURA; SCOUTTEN; WALKER; BROWN; JONES; LUTZ;
JEFFERSON; MARINO; SMITH; JOHNSON; PEREZ; SCHNEIDER;
PARKER; SMITH; SETRIGHT; MEHTA; LEE; GOUNOT;
```

Vorkompilierte SQL-Anweisungen

- **Vorkompilierte SQL-Anweisungen**
  - Kompilierung kostet Zeit
  - Idee: oft verwendete Anweisungen nur einmal kompilieren
    → `PreparedStatement`
  - erzeugen mittels: `Connection.prepareStatement(String sql)`
  - Unterklasse von `Statement`
  - parametrierbar mittels `?`
    (Effizienzprobleme bei hohem Skew, siehe 5. Stored Procedures)

- **Wichtige Methoden**
  - `executeQuery() / executeUpdate() / execute()`
    - siehe `Statement`
  - `set*(int i, *)`
    - Setzen des i-ten Parameters (1-basiert)

Beispiel

- ```
  PreparedStatement pstmt = con.prepareStatement(
              "SELECT SALARY FROM EMPLOYEE WHERE LASTNAME=?");
  pstmt.setString(1, "HAAS");
  ResultSet rs = pstmt.executeQuery();
  rs.next();
  System.out.println(rs.getDouble(1));
  rs.close();
  pstmt.close();
  ```

- **Ausgabe**
  ```
  52750.0
  ```
  ```

Aufruf von Stored Procedures

- **`CallableStatement`-Objekt**
  - Ausführen von Stored Procedures
  - erzeugen mittels: `Connection.prepareCall(String sql)`
    - Parametersyntax: `"{CALL <name>(?,…)}"`
    - automatisches Umschreiben auf Syntax des DBMS
  - Unterklasse von `PreparedStatement` → vorkompiliert
  - Ausgabeparameter müssen registriert werden

- **Wichtige Methoden**
  - siehe `PreparedStatement`
  - `registerOutParameter(int i, int type);`
    - Registrierung der Ausgabeparameter
    - Argumente
      - Parameterindex (1-basiert)
      - Datentyp (siehe `java.sql.Types`)

Beispiel

- `GET_TABLE(IN TABLENAME VARCHAR(32), OUT SQL VARCHAR(128));`
  - Stored Procedure: liefert Inhalt einer Relation sowie **SELECT**-Anweisung

-
```
CallableStatement cs = con.prepareCall("{CALL GET_TABLE(?,?)}");
cs.setString(1, "EMPLOYEE");
cs.registerOutParameter(2, Types.VARCHAR);
cs.execute();
System.out.println("PARAMETER 2 (SQL): " + cs.getString(2));
System.out.print("RESULTSET (LASTNAME COLUMN): ");
ResultSet rs = cs.getResultSet();
while (rs.next()) {
  System.out.print(rs.getString("LASTNAME") + " ");
}
System.out.println();
rs.close();
cs.close();
```
- **Ausgabe**
  - `PARAMETER 2 (SQL): SELECT * FROM EMPLOYEE`
  - `RESULTSET (LASTNAME COLUMN): HAAS THOMPSON KWAN GEYER STERN …`

JDBC-Protokollierung

- **JDBC-Protokollierung**
  - in DB2 mittels JDBC trace facility
  - Aufzeichnung aller JDBC-Aufrufe
  - nur für CLI-basierten Typ-2-Treiber (*app driver*)
  - Aktivieren: in Datei `sqllib/db2cli.ini`:

```
[COMMON]
JDBCTrace=1
JDBCTraceFlush=1
JDBCTracePathName="/home/<login>/traces"
```

  - erheblicher Performance-Einfluss
    → Deaktivieren falls nicht notwendig

JDBC trace

- **Beispielausgabe**

```
jdbc.app.DB2Statement -> executeQuery( SELECT LASTNAME FROM EMPLOYEE ) (2006-10-24 11:14:30.625)
| 10: Statement Handle = 1:1
| jdbc.app.DB2Statement -> getStatementType( SELECT LASTNAME FROM EMPLOYEE ) (2006-10-24 11:14:30.625)
| jdbc.app.DB2Statement <- getStatementType() returns STMT_TYPE_QUERY (24) [Time Elapsed = 0.0] (2006-10-24…
| jdbc.app.DB2Statement -> execute2( SELECT LASTNAME FROM EMPLOYEE ) (2006-10-24 11:14:30.625)
| | 10: StatementHandle  = 1:1
| | 10: SQLExecDirect - returnCode  = 0
| | 10: rowCount = 0
| jdbc.app.DB2Statement <- execute2() [Time Elapsed = 0.0] (2006-10-24 11:14:30.625)
| jdbc.app.DB2Statement -> getResultSet() (2006-10-24 11:14:30.625)
| | 10: Statement Handle = 1:1
| | jdbc.app.DB2ResultSetTrace -> DB2ResultSet( stmt, nCols=0) (2006-10-24 11:14:30.656)
| | | 10: numCols = 1
| | jdbc.app.DB2ResultSetTrace <- DB2ResultSet() [Time Elapsed = 0.0] (2006-10-24 11:14:30.656)
| | jdbc.app.DB2ResultSetTrace -> DB2ResultSetTrace( stmt,0 ) (2006-10-24 11:14:30.656)
| | | Statement handle = 1:1
| | jdbc.app.DB2ResultSetTrace <- DB2ResultSetTrace (2006-10-24 11:14:30.656)
| jdbc.app.DB2Statement <- getResultSet() [Time Elapsed = 0.062] (2006-10-24 11:14:30.656)
jdbc.app.DB2Statement <- executeQuery() [Time Elapsed = 0.062] (2006-10-24 11:14:30.656)

jdbc.app.DB2ResultSetTrace -> next (2006-10-24 11:14:30.656)
| Statement handle = 1:1
| jdbc.app.DB2ResultSetTrace -> clearIsList (2006-10-24 11:14:30.656)
| | Statement handle = 1:1
| jdbc.app.DB2ResultSetTrace <- clearIsList - Time Elapsed = 0.0 (2006-10-24 11:14:30.656)
| jdbc.app.DB2ResultSetTrace -> clearWarnings (2006-10-24 11:14:30.671)
| | Statement handle = 1:1
| jdbc.app.DB2ResultSetTrace <- clearWarnings - Time Elapsed = 0.0 (2006-10-24 11:14:30.671)
jdbc.app.DB2ResultSetTrace <- next - Time Elapsed = 0.015 (2006-10-24 11:14:30.671)
```

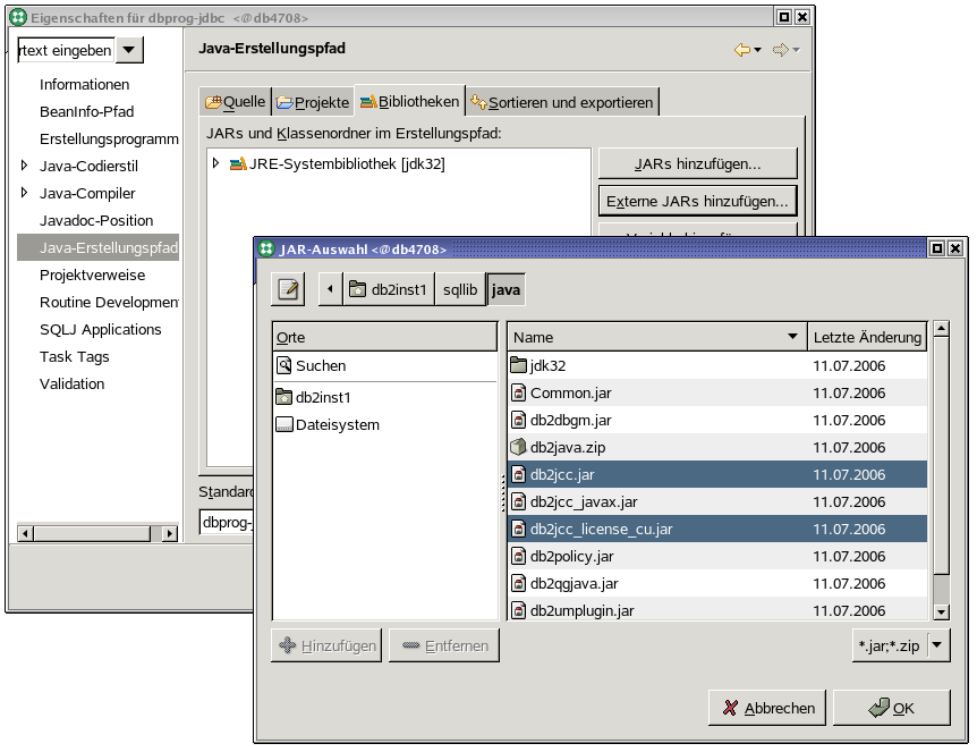
Java-Entwicklung mit der DB2 Developer Workbench

Java-Entwicklung mit DB2DWB

- **Per Konsole**
  - Skript zum Kompilieren und Ausführen:
    `/home/dbprog/build/build-app.sh <datei>`

- **DB2 Developer Workbench**
  - unterstützt Entwicklung von Java-Programmen
  - verwendet das JDT-Plugin (*Java Development Tools*) von Eclipse
  - wichtige Shortcuts
    - Shift+F1: Anzeigen von JavaDoc-Dokumentation
    - Strg+1: Automatische Fehlerbehebung
      - fehlende Importanweisungen
      - `throws`-Klauseln
    - Strg+Leertaste: Automatisches Vervollständigen, Methoden einer Klasse anzeigen
    - Strg+F11: Ausführen des zuletzt gestarteten Programms

- **Schritte zum Erstellen von JDBC-Anwendungen**
  - neues Java-Projekt anlegen
  - Bibliotheken einbinden
    - Rechtsklick auf das Projekt
    - Eigenschaften
    - Java-Erstellungspfad
    - Bibliotheken
    - Externe JARs hinzufügen
    - unter `~/sqllib/java` die Dateien `db2jcc.jar` und `db2jcc_license_cu.jar` auswählen



Zusammenfassung

- **JDBC**
  - Industriestandard zum Zugriff auf Datenbanken aus Java-Anwendungen
  - datenbankenunabhängig
  - 4 verschiedene Treibertypen
  - wichtige Klassen: `Connection`, `Statement`, `ResultSet`
- **DB2 und Java**
  - Typ 2/3/4-Treiber
  - neuere DB2-Versionen besitzen einheitlichen JDBC-Treiber (JCC)
  - JDBC trace facility
- **DB2 Developer Workbench**
  - enthält JDT
  - Einbinden der DB2-Bibliotheken erforderlich

Alle Beispiele liegen unter: `/home/dbprog/examples/07/`

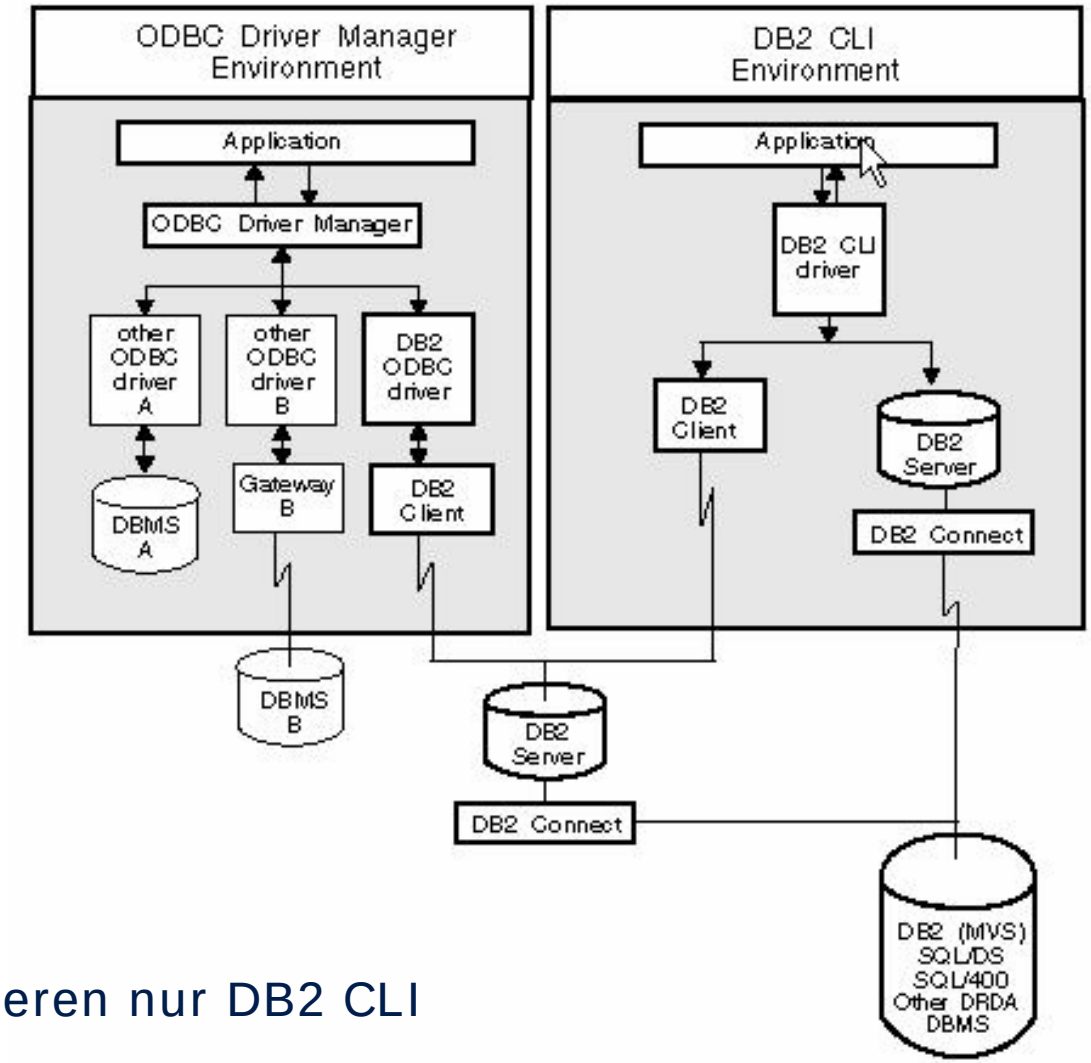
TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik; Institut Systemarchitektur; Professur Datenbanken

Datenbankprogrammierung
8. CLI und ODBC

- **CLI / ODBC**
  - Beide Standards basieren auf der X/Open Call-Level Interface Spezifikation
  - **CLI** (*call level interface*)
    - Prozeduren/Funktionen zur
      - Kommunikation mit dem DBMS
      - Definition und Ausführung von Anfragen
      - Verarbeitung von Ergebnissen
  - **ODBC** (*Open Data Base Connectivity*)
    - CLI-konforme Implementierung für Microsoft Windows
    - Erweitert X/Open CLI um zusätzliche Funktionalität
    - Zugriff auf verschiedene Datenbanksysteme über systemspezifische Treiber möglich
    - dynamisches Laden von Treibern (implementiert als DLL)

Architektur



➡ im Weiteren nur DB2 CLI

Unterstützte DB2-Features

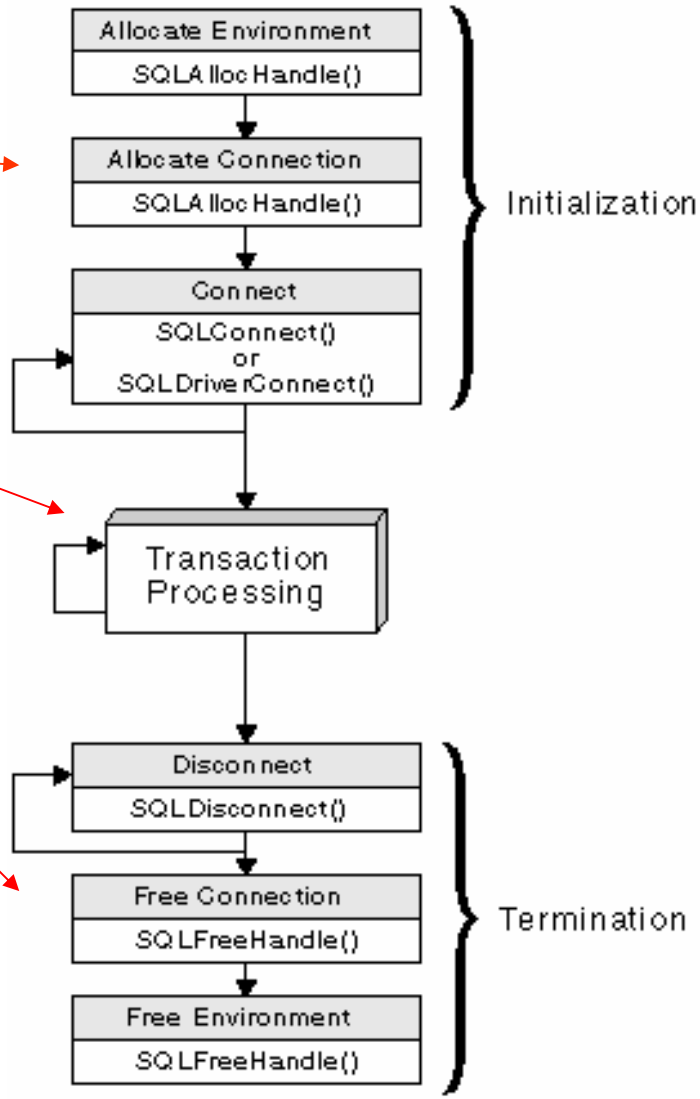
- **Unterstützte DB2-Features**
  - Compound SQL
  - UDTs
  - UDFs und Stored Procedures
  - Distributed Unit of Work (DUOW)
  - 2-Phasen-Commit-Protokoll

- **CLI-Funktionsübersicht**
  http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.apdv.cli.doc/doc/r0000553.htm

- **CLI-Include-Dateien:**
  - unter \sqllib\include\sqlcli.h und sqlcli1.h
  - Enthalten CLI-Konstanten und Funktionsprototypen

CLI Funktionsaufrufe

- **Aufrufsequenz**
  - Initialisierung
    - Allokation von Systemressourcen
    - Referenzierung über Handler
  - Anfrageverarbeitung
  - Ressourcenfreigabe
    - Freigabe aller allokierter Ressourcen
- **Handle**
  - Verweis auf globale Datenstruktur zur Kommunikation mit DBMS
  - C-Datentyp: SQLHANDLE
  - erfordern explizite Allokation



Handle-Arten

- **Handle-Arten**
  - Environment-Handle: verwaltet Informationen über globalen Zustand einer Anwendung
  - Connection-Handle: Verwalten von Verbindungsdaten
  - Descriptor-Handle: Daten zu Ergebnisspalten bzw. Parametern
  - Statement-Handle: Information zu einer SQL-Anweisung
    - Binden von SQL-Platzhalter
    - Kompilieren und Ausführen einer Anweisung
    - Metadaten zu Ergebnismengen
    - Binden von Spalten der Ergebnismenge
    - Abrufen von Daten aus der Ergebnismenge

Initialisierung

- **Initialisierung**
  - `SQLAllocHandle (SQLSMALLINT HandleType, SQLHANDLE InputHandle, SQLHANDLE *OutputHandle) ;`
  - Ab ODBC 3.0, ersetzt `SQLAllocConnect()`, `SQLAllocEnv()` und `SQLAllocStmt()`
  - `InputHandle` bildet Kontext für neu anzulegenden `OutputHandle`

InputHandle (Kontext)	**OutputHandle** (zu erstellen)	**Beschreibung**
SQL_NULL_HANDLE	SQL_HANDLE_ENV	Umgebungs-Handle
SQL_HANDLE_ENV	SQL_HANDLE_DBC	Verbindungs-Handle
SQL_HANDLE_DBC	SQL_HANDLE_STMT	Anweisungs-Handle
SQL_HANDLE_DBC	SQL_HANDLE_DESC	Descriptor-Handle

Verbindungsaufbau

- **Verbindungsaufbau**
  - SQLConnect (SQLHDBC ConnectionHandle,
    SQLCHAR *ServerName,
    SQLSMALLINT ServerNameLength,
    SQLCHAR *UserName,
    SQLSMALLINT UserNameLength,
    SQLCHAR *Authentication,
    SQLSMALLINT AuthenticationLength)
  - stellt Verbindung zur Datenbank her
  - Längenargument für Zeichenketten
    - exakt
    - Null-terminierter String (SQL_NTS)
    - NULL-Werte (SQL_NULL_DATA)

Initialisierung

- **Beispiel** (mycliapp.c)

```
SQLHANDLE henv, hdbc, hstmt ;

SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv) ;

if (henv != 0)
    SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc) ;

if (hdbc != 0)
    cliRC = SQLConnect(hdbc, (SQLCHAR *) "SAMPLE",
                       SQL_NTS, (SQLCHAR *) "db2inst1",
                       SQL_NTS, (SQLCHAR *) "foo",
                       SQL_NTS) ;

if (cliRC == SQL_SUCCESS)
    SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
```

Terminierung

- **Terminierung**
  - SQLDisconnect(SQLHDBC ConnectionHandle)
    - Verbindungslösung von der Datenquelle
    - Treiber freigeben
  - SQLFreeHandle(SQLSMALLINT HandleType,
    SQLHANDLE Handle)
    - Verbindungsspeicher zur Datenquelle freigeben
    - verwendet für SQL_HANDLE_STMT, SQL_HANDLE_DBC, SQL_HANDLE_DESC und SQL_HANDLE_ENV

- **Beispiel**

```
SQLFreeHandle(SQL_HANDLE_STMT, hstmt) ;
SQLDisconnect(hdbc) ;
SQLFreeHandle(SQL_HANDLE_DBC, hdbc) ;
SQLFreeHandle(SQL_HANDLE_ENV, henv) ;
```

- **Anfrageverarbeitung**
  - SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt)
    - Allokiert Speicher für eine SQL-Anweisung
  - Kompilieren und Ausführen
    1. SQLPrepare(SQLHSTMT hstmt,
       SQLCHAR *StatementText,
       SQLINTEGER TextLength)
       - Anweisung/SP-Aufruf am DB-Server kompilieren
       - SP-Syntax: "{CALL <name>(?,…)}"
       - Rückgabe einer Referenz auf das kompilierte Statement
    2. SQLBindParameter(SQLHSTMT StatementHandle,
       SQLUSMALLINT ParameterNumber,
       SQLSMALLINT InputOutputType,
       SQLSMALLINT ValueType,
       SQLSMALLINT ParameterType,
       SQLUINTEGER ColumnSize,
       SQLSMALLINT DecimalDigits,
       SQLPOINTER ParameterValuePtr,
       SQLINTEGER BufferLength,
       SQLINTEGER *StrLen_or_IndPtr)
       - wenn die Anweisung Parametermarker enthält

Anfrageverarbeitung

- **Anfrageverarbeitung**
  - Kompilieren und Ausführen
    3. SQLExecute(SQLHSTMT hstmt)
       - Ausführung der kompilierten Anweisung
       - Referenz und optionale Parameter als Eingabe übergeben

  - Direkte Ausführung
    1. SQLExecDirect(SQLHSTMT hstmt,
       SQLCHAR *StatementText,
       SQLINTEGER TextLength)
       - Ausführen der SQL-Anweisung

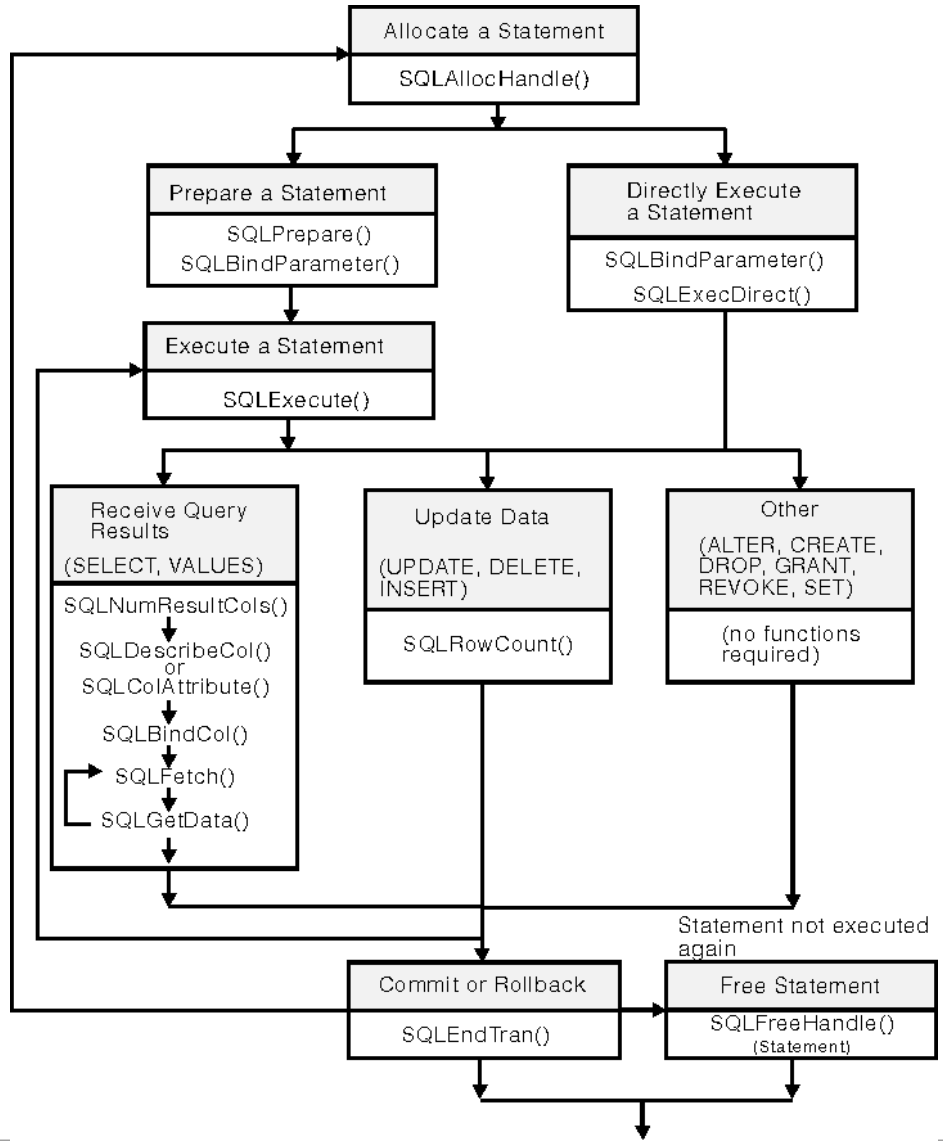
- **Verarbeitung der Ergebnismenge**
  - Nur bei **SELECT**- und **VALUE**-Anweisungen
  - Ermitteln der Struktur der Ergebnismenge
    - `SQLNumResultCols(SQLHSTMT hstmt, SQLSMALLINT *ColumnCountPtr)`
      - Ermittelt Anzahl der Ergebnisspalten
    - `SQLDescribeCol(SQLHSTMT hstmt, SQLSMALLINT ColumnNumber, SQLCHAR *ColumnName, SQLSMALLINT BufferLength, SQLSMALLINT *NameLengthPtr, SQLSMALLINT *DataTypePtr, SQLUINTEGER *ColumnSizePtr, SQLSMALLINT *DecimalDigitsPtr, SQLSMALLINT *NullablePtr)`
      - Ermittelt allgemeine Spalteninformationen (Name, Typ, Genauigkeit, …)

- **Verarbeitung der Ergebnismenge**
  - Ermitteln der Struktur der Ergebnismenge
    - SQLColAttribute(SQLHSTMT hstmt,
      SQLSMALLINT ColumnNumber,
      SQLSMALLINT FieldIdentifier,
      SQLPOINTER CharacterAttributePtr,
      SQLSMALLINT BufferLength,
      SQLSMALLINT *StringLengthPtr,
      SQLPOINTER NumericAttributePtr)
      - Ermittelt speziellere Spalteninformationen (Kardinalität, Nullwerte vorhanden?, UDDT-Name)

Verarbeitung der Ergebnismenge

- **Verarbeitung der Ergebnismenge**
  - Ausgabe der Ergebnismenge
    - `SQLBindCol(SQLHSTMT hstmt, SQLUSMALLINT ColumnNumber, SQLSMALLINT TargetType, SQLPOINTER TargetValuePtr, SQLINTEGER BufferLength, SQLINTEGER *StrLen_or_IndPtr)`
      - Binden von Anwendungsvariablen an Spalten der Ergebnismenge (optional)
    - `SQLFetch(SQLHSTMT hstmt)`
      - Zeilenweises Einlesen der Daten in die gebundenen Variablen
    - `SQLGetData(SQLHSTMT hstmt, SQLUSMALLINT ColumnNumber, SQLSMALLINT TargetType, SQLPOINTER TargetValuePtr, SQLINTEGER BufferLength, SQLINTEGER *StrLen_or_IndPtr)`
      - Alternativ zu `SQLBindCol ()` für ungebundene Spalten
      - Aufruf nach `SQLFetch()`

Anfrageverarbeitung



- **Beispiel**

```
SQLHANDLE hstmt = 0;
SQLCHAR SQLStmt[255];
SQLCHAR JobType[10];
…
strcpy(SQLStmt, "SELECT LASTNAME FROM EMPLOYEE WHERE JOB = ?") ;

SQLPrepare(hstmt, SQLStmt, SQL_NTS) ;
SQLBindParameter(hstmt, 1,
                 SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
                 sizeof(JobType), 0, JobType,
                 sizeof(JobType), NULL) ;

strcpy(JobType, "DESIGNER") ;
SQLExecute(hstmt) ;
```

- **…Beispiel Fortsetzung**

```
SQLBindCol(hstmt, 1, SQL_C_CHAR,
          (SQLPOINTER) LastName, sizeof(LastName), NULL) ;
cliRC = SQLFetch(hstmt) ;
while (cliRC != SQL_NO_DATA)
  {
    printf("%s; ", LastName) ;
    cliRC = SQLFetch(hstmt) ;
  }
```

- **Ausgabe**

```
ADAMSON; PIANKA; YOSHIMURA; SCOUTTEN; WALKER; BROWN; JONES;
LUTZ; YAMAMOTO; JOHN
```

Datenquellen- und Treiberattribute

- **Statisch**
  - SQLGetFunctions(SQLHDBC hdbc,
    SQLUSMALLINT fFunction,
    SQLUSMALLINT *SupportedPtr)
    - Ermittelt ob eine spezifische Funktion unterstützt wird
  - SQLDataSources(SQLHENV henv,
    SQLUSMALLINT Direction,
    SQLCHAR *ServerName,
    SQLSMALLINT BufferLength1,
    SQLSMALLINT *NameLength1Ptr,
    SQLCHAR *Description,
    SQLSMALLINT BufferLength2,
    SQLSMALLINT *NameLength2Ptr)
    - Zeigt alle katalogisierten Datenbanken an

- **Statisch**
  - SQLGetInfo(SQLHDBC hdbc,
    SQLUSMALLINT InfoType,
    SQLPOINTER InfoValuePtr,
    SQLSMALLINT BufferLength,
    SQLSMALLINT *StringLengthPtr)
    - Allgemeine Informationen zum verbundenen DBMS
    - z.B. unterstützte Aggregationsfunktionen (insgesamt 165 Einzelinformationen)
  - SQLGetTypeInfo(SQLHSTMT hstmt,
    SQLSMALLINT DataType)
    - Ermittelt die unterstützten Datentypen
    - Erzeugt SQL-Ergebnismenge

Datenquellen- und Treiberattribute

- **Beispiel**

```
SQLUSMALLINT supported ;
SQLCHAR dbInfoBuf[255] ;
SQLSMALLINT outlen ;

SQLGetFunctions(hdbc, SQL_API_SQLGETINFO, &supported) ;

if (supported == SQL_TRUE)
{
 SQLGetInfo(hdbc, SQL_DATABASE_NAME, dbInfoBuf,
            255, &outlen) ;
 printf("Server Database Name : %s\n", dbInfoBuf) ;
}
else
 printf("SQLGetInfo is not supported!\n") ;
```

Datenquellen- und Treiberattribute

- **Dynamisch**
  - SQLGetEnvAttr(SQLHENV henv,
    SQLINTEGER Attribute,
    SQLPOINTER ValuePtr,
    SQLINTEGER BufferLength,
    SQLINTEGER *StringLengthPtr)
    - Umgebungsattribute, ODBC 2.0 oder 3.0
  - SQLGetConnectAttr(SQLHDBC hdbc,
    SQLINTEGER Attribute,
    SQLPOINTER ValuePtr,
    SQLINTEGER BufferLength,
    SQLINTEGER *StringLengthPtr)
    - Verbindungsattribute, z.B. Auto-Commit, Anzahl der gleichzeitigen Verbindungen, Isolation-Level der aktuellen Verbindung

Datenquellen- und Treiberattribute

- **Dynamisch**
  - SQLGetStmtAttr(SQLHSTMT hstmt,
    SQLINTEGER Attribute,
    SQLPOINTER ValuePtr,
    SQLINTEGER BufferLength,
    SQLINTEGER *StringLengthPtr)
    - Anweisungattribute, z.B. Concurrency Level, Cursor-Typen
  - SQLSetStmtAttr(SQLHSTMT hstmt,
    SQLINTEGER Attribute,
    SQLPOINTER ValuePtr,
    SQLINTEGER StringLength)
    - Setzen der Anweisungsattribute

Anweisungsattribute

- **Beispiel**

```
SQLINTEGER cursor_hold;
SQLGetStmtAttr( hstmt,
                SQL_ATTR_CURSOR_HOLD,
                &cursor_hold,
                0,
                NULL ) ;

printf( "\nCursor With Hold is: " ) ;
if ( cursor_hold == SQL_CURSOR_HOLD_ON )
  printf( "ON\n" ) ;
else
  printf( "OFF\n" ) ;
```

Fehlerbehandlung

- **Fehlerbehandlung**
  - Rückgabe-Code (Integerwert) bei Funktionsaufruf
    - `SQL_SUCCESS (0)`
      - Funktion erfolgreich ausgeführt
    - `SQL_SUCCESS_WITH_INFO (1)`
      - Funktion erfolgreich ausgeführt (mit Warnungen)
    - `SQL_NO_DATA (100)`
      - Funktion erfolgreich ausgeführt
      - keine Daten zurück gegeben
    - `SQL_INVALID_HANDLE (-2)`
      - Funktionsausführung fehlgeschlagen
      - Programmierfehler, nicht allokierter oder falscher Handle
    - `SQL_NEED_DATA (99)`
      - Funktionsausführung fehlgeschlagen aufgrund fehlender Daten
    - `SQL_STILL_EXECUTING (2)`
      - Funktion noch in Ausführung
    - `SQL_ERROR (-1)`
      - Funktionsausführung fehlgeschlagen

Diagnoseinformationen

- **Diagnoseinformationen**
  - Erlaubt detaillierte Fehlerdiagnose
  - SQLSTATE
    - Rückgabewert jedes DB2 CLI-Funktionsaufrufs
    - Bestehend aus 5 Hexadezimalzeichen
    - Fehlerklasse (cc):
      - 01 = Warnung
      - HY = Fehler hervorgerufen durch DB2 CLI
      - IM = Fehler hervorgerufen durch ODBC Driver Manager
    - Fehlerunterklasse (sss)

  - `SQLGetDiagField(HandleType, Handle, RecNumber, DiagIdentifier, DiagInfoPtr, BufferLength, *StringLengthPtr)`
    - Liefert Headereinträge der Diagnoseinformation
  - `SQLGetDiagRec(HandleType, Handle, RecNumber, *SQLState, *NativeErrorPtr, *MessageText, BufferLength, *TextLengthPtr)`
    - Liefert einen Eintrag der Diagnoseinformation

Fehlerbehandlung

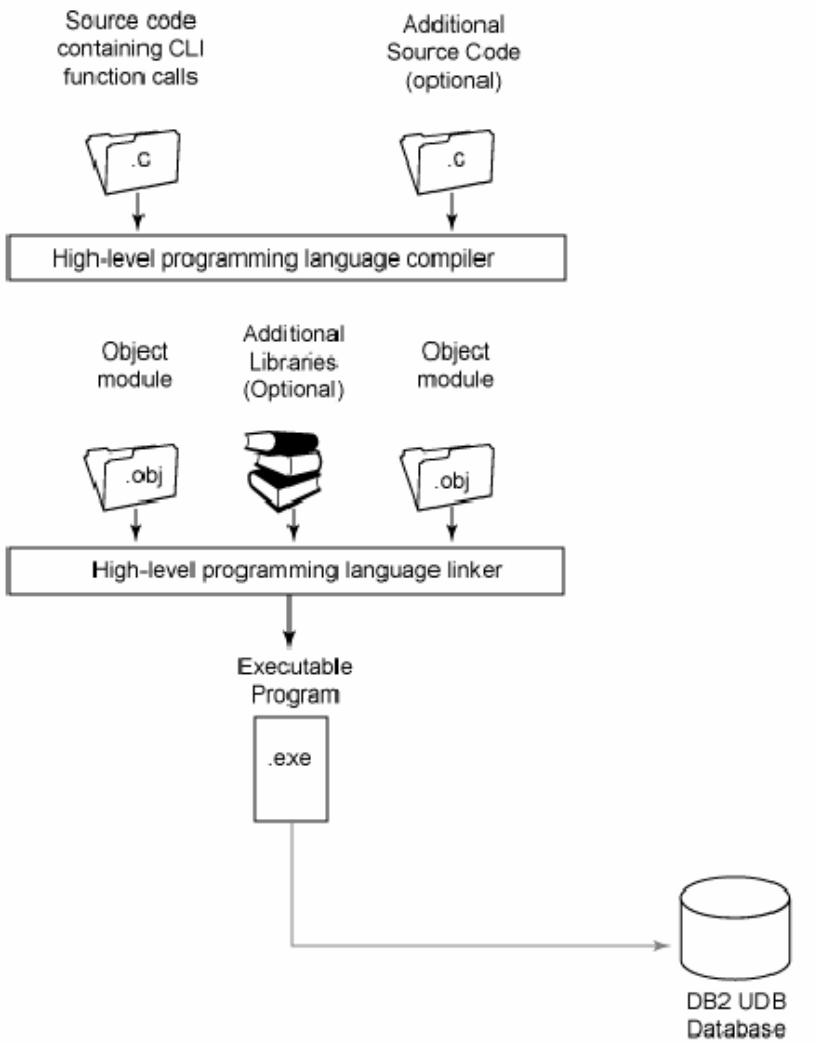
- **Beispiel**

```
SQLCHAR SQLState[6];
SQLCHAR ErrMsg[255];
cliRC = SQLConnect(ConHandle, (SQLCHAR *) "SAMPLE",
                   SQL_NTS, (SQLCHAR *) "foo",
                   SQL_NTS, (SQLCHAR *) "bar",
                   SQL_NTS);
if (cliRC != SQL_SUCCESS)
{
    SQLGetDiagField(SQL_HANDLE_DBC, hdbc, 0, SQL_DIAG_NUMBER,
                &NumRecords, SQL_IS_INTEGER, NULL);
    for (Counter = 1; Counter <= NumRecords; Counter++) {
        SQLGetDiagRec(SQL_HANDLE_DBC, hdbc, Counter,
                      SQLState, SQLCode, ErrMsg, sizeof(ErrMsg), 0);
        printf("SQLSTATE : %s\n", SQLState);
        printf("%s\n", ErrMsg);
     }
}
```

- **Ausgabe**

```
 SQLSTATE : 08001
 [IBM][CLI Driver] SQL30082N  Security processing failed with reason "24"
    ("USERNAME AND/OR PASSWORD INVALID").  SQLSTATE=08001
```

CLI-Anwendung erstellen




```
:~> /home/dbprog/build/build-app.sh
     mycliapp
Connected...
000150 ADAMSON
000160 PIANKA
000170 YOSHIMURA
000180 SCOUTTEN
000190 WALKER
000200 BROWN
000210 JONES
000220 LUTZ
200170 YAMAMOTO
200220 JOHN
Disconnected.
:~>
```

bldapp

```
DB2DIR=/opt/ibm/db2/V9.1

# Compile the program.
gcc -I$DB2DIR/include -c $1.c

# Link the program.
gcc -o $1 $1.o -L$DB2DIR/lib32 -l db2
```

Zusammenfassung

- **CLI / ODBC-Architektur**
- **CLI-Handle**
  - Environment-Handle, Connection-Handle, Descriptor-Handle, Statement-Handle
- **Initialisierung, Terminierung, Anfrageverarbeitung**
- **Attribute**
  - Datenquellen- und Treiberattribute
  - Anweisungsattribute
- **Fehlerbehandlung und Diagnose**

Alle Beispiele liegen unter: `/home/dbprog/examples/08/`

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut Systemarchitektur, Professur Datenbanken

Datenbankprogrammierung
9. Externe Funktionen

Externe UDFs

- **Externe nutzerdefinierte Funktionen**
  - in höherer Programmiersprache geschrieben (z.B. C, Java)
  - Warum extern?
    - nahezu beliebige Logik implementierbar
    - Zugriff auf bestehende Softwarekomponenten möglich
  - Arten
    - Externe Skalarfunktion (*external scalar*)
    - Externe Tabellenfunktion (*external table*)
    - OLE DB-Tabellenfunktion → Anbindung an OLE DB-Provider

- **Grundlegender Ablauf**
  1. Erstellen der Funktion
  2. Kompilierung
  3. optional: Erzeugen einer Library
  4. Speichern des Binärcodes im Datenbankserver
     - `sqllib/function` oder `sqllib/function/unfenced`
  5. Registrierung mittels **`CREATE FUNCTION`**

Externe Skalarfunktionen

Syntax

- **Syntax**

```
   CREATE FUNCTION <name> ([<par-name>] <par-type>, …)
neu RETURNS <type> [CAST FROM <type>]
   [SPECIFIC <unique-name>]
    EXTERNAL NAME <external-name>                    sprachabhängig
    LANGUAGE C|JAVA|OLE
    PARAMETER STYLE DB2GENERAL|JAVA|SQL
   [[NOT] DETERMINISTIC]
neu [[NOT] FENCED]
neu [RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT]
   [NO SQL | CONTAINS SQL | READS SQL DATA]
   [[NO] EXTERNAL ACTION]
neu [NO SCRATCHPAD | SCRATCHPAD <length>]
neu [[NO] FINAL CALL]
neu [ALLOW PARALLEL | DISALLOW PARALLEL]
neu [[NO] DBINFO]
```

Syntax

- **Optionen**
  - siehe “4. Nutzerdefinierte Funktionen”
  - **CAST FROM**: automatische Typkonvertierung
  - **EXTERNAL NAME**: Zugriffspfad auf externe UDF
  - **LANGUAGE**: verwendete Programmiersprache
  - **PARAMETER STYLE**: Art der Parameterübergabe
  - **FENCED**: DBMS-externe Ausführung
  - **RETURNS NULL ON NULL INPUT**: Aufruf nur, falls alle Argumente nicht-null
  - Datenänderungen generell verboten
  - **SCRATCHPAD**: Zwischenspeicherung von Daten
  - **FINAL CALL**: Abschließender Aufruf (ähnlich Destruktor)
  - **ALLOW PARALLEL**: mehrere, parallel verwendete Instanzen der UDF erlauben
    - Standardwert falls weder **NOT DETERMINISTIC**, **EXTERNAL ACTION**, **SCRATCHPAD**, oder **FINAL CALL** angegeben
  - **DBINFO**: Informationen über die Datenbank an UDF weitergeben

FENCED vs. UNFENCED

- **FENCED vs. UNFENCED**
  - **FENCED**-Modus (Standard)
    - UDF läuft in eigenem Prozess
    - kein Zugriff auf DB2-Interna
    - sicher, aber langsam
  - **NOT FENCED**-Modus
    - in DB2-Engine ausgeführt → schneller
    - für bereits erprobte, vertrauenswürdige UDFs
  - Wechsel des Ausführungsmodus
    - erfordert **SYSADM**-Privileg oder **DBADM**- bzw. **CREATE_NOT_FENCED_ROUTINE**-Rechte
    - **ALTER FUNCTION** `<external-functoin>` **NOT FENCED**
    - bestimmt Ablageverzeichnis: `sqllib/function` oder `sqllib/function/unfenced`

Beispiel

- **Beispiel**
  - **CREATE FUNCTION** HELLOWORLD()
    RETURNS VARCHAR(15)
    EXTERNAL NAME 'UDFJavaStyleExample!helloWorld‘
    LANGUAGE JAVA
    PARAMETER STYLE JAVA
    DETERMINISTIC
    NO SQL
    NO EXTERNAL ACTION;
  - **VALUES**(HELLOWORLD())

    1

    Hello World

Externe Tabellenfunktionen

Syntax

- **Syntax**

```
    CREATE FUNCTION <name> ([<par-name>] <par-type>, …)
neu RETURNS TABLE(<col-name> <col-type>, …)
   [SPECIFIC <unique-name>]
    EXTERNAL NAME <external-name>                    sprachabhängig
    LANGUAGE C|JAVA|OLE
neu PARAMETER STYLE DB2GENERAL|SQL
   [[NOT] DETERMINISTIC]
   [[NOT] FENCED]
   [RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT]
   [NO SQL | CONTAINS SQL | READS SQL DATA]
   [[NO] EXTERNAL ACTION]
   [NO SCRATCHPAD | SCRATCHPAD <length>]
   [[NO] FINAL CALL]
neu [DISALLOW PARALLEL | ALLOW PARALLEL EXECUTE ON ALL DATABASE PARTITIONS RESULT TABLE DISTRIBUTED]
   [[NO] DBINFO]
```

Syntax

- **Optionen**
  - siehe externe Skalarfunktionen
  - **`ALLOW PARALLEL EXECUTE ON ALL DATABASE PARTITIONS RESULT TABLE DISTRIBUTED`**
    - für partitionierte Datenbanken
    - Funktion wird auf jeder Partition ausgeführt, Ergebnisse vereinigt
    - Angabe von **`NO SQL`** erforderlich
  - **`DISALLOW PARALLEL`**
    - Funktion wird nur auf aktueller Partition ausgeführt
    - sollte bei nicht-partitionierter Datenbank angegeben werden
    - SQL-Abfragen möglich

Beispiel

- **Beispiel**
  - **CREATE FUNCTION** ALL_EMPLOYEES()
    RETURNS TABLE(FULLNAME **VARCHAR**(128))
    EXTERNAL NAME 'UDFDb2generalStyleExample!allEmployees‘
    LANGUAGE JAVA
    PARAMETER STYLE DB2GENERAL
    DETERMINISTIC
    READS SQL DATA
    NO SCRATCHPAD
    NO EXTERNAL ACTION
    DISALLOW PARALLEL;

  - **SELECT** * **FROM TABLE**(ALL_EMPLOYEES()) X

```
FULLNAME
--------------------
CHRISTINE HAAS
MICHAEL THOMPSON
…
```

Nutzerdefinierte Funktionen (Java)

Nutzerdefinierte Funktionen in Java

- **Nutzerdefinierte Funktionen in Java**
  - Ablage als Methode einer Klasse
    - Klasse und Methode müssen **`public`** sein
    - je nach Parameterart zusätzlich **`static`**
  - Unterstützte Parameterarten
    - **JAVA**: nur einfache UDFs
    - **DB2GENERAL**: beliebige UDFs möglich
  - nur **FENCED**-Modus
  - Zugriff auf Standard-I/O-Ströme nicht möglich (`System.`*`out`*, …)

- **Aufbau des externen Namens (`EXTERNAL NAME`)**
  - Angabe der Klasse und Methode, optional auch der Bibliothek
  - `[<jar-id>:]<class-id>!<method-id>`
  - Ablage der kompilierten Klasse/Bibliothek in `sqllib/function`

Parameterart **JAVA**

- **Parameterart JAVA**
  - nur externe Skalarfunktionen
  - kein **SCRATCHPAD**, **FINAL CALL**, **DBINFO**
  - Aufbau
    - Klasse als **public** definiert
    - Methode als **public static** definiert
    - Parameter als Java-Typen, automatisches Marshalling
    - Rückgabewert der Methode wird an Datenbanksystem weitergeleitet
  - Zugriff auf Datenbank
    - nur lesend
    - mittels Standardverbindung:
      `DriverManager.getConnection("jdbc:default:connection")`
    - Fehler werden an das DBMS weitergereicht
      → Definition mit **throws** SQLException

Beispiele

- ```
  public static String helloWorld() {
      return "Hello World";
  }
  ```

```
VALUES(HELLO_WORLD());
1
---------------
Hello World
```

- ```
  public static int mod10(int x) {
      return x % 10;
  }
  ```

```
VALUES(MOD10(7), MOD10(17));
1           2
----------- -----------
          7           7
```

- ```
  public static int noEmployees() throws SQLException {
      Connection con =
          DriverManager.getConnection("jdbc:default:connection");
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(
              "SELECT COUNT(*) FROM EMPLOYEE");
      rs.next();
      int noEmployees = rs.getInt(1);
      rs.close();
      stmt.close();
      con.close();
      return noEmployees;
  }
  ```

```
VALUES(NO_EMPLOYEES());
1
-----------
         42
```
```

Parameterart **DB2GENERAL**

- **Parameterart DB2GENERAL**
  - externe Skalar- und Tabellenfunktionen
  - Aufbau
    - Klassen erben von `COM.ibm.db2.app.UDF`
    - Klasse und Methode als **`public`** definiert
    - Fehler werden an das DBMS weitergereicht
      → Definition mit **`throws`** `Exception`
    - Ein- und Ausgabeparameter als Java-Typen, automatisches Marshalling

UDF-Klasse

- **Wichtige Methoden der `UDF`-Klasse**
  - `Connection getConnection()`
    - Liefert `Connection`-Objekt für aufrufende Datenbank
  - `boolean isNull(int index)`
    - Überprüfung eines Eingabearguments auf **`NULL`**-Wert
  - `void set(int index, *)`
    - Setzen von Ausgabeparametern (1-basiert)
  - `boolean needToSet(int index)`
    - Erkennung von nicht-verwendeten Ausgabeparametern (nur Tabellenfunktionen)
  - `setSQLstate(String) / setSQLmessage(String)`
    - Rückgabe von Fehlermeldungen
  - `byte[] getScratchpad() / setScratchpad(byte[])`
    - Auslesen/Setzen des Scratchpad (später mehr)
  - `int getCallType()`
    - Aufruftyp ermitteln
      - Skalarfunktionen: *`SQLUDF_FIRST_CALL, SQLUDF_NORMAL_CALL`*
      - Tabellenfunktionen: später…
  - `* getDB*()`
    - Informationen über aufrufende Datenbank (Angabe von **`DBINFO`** erforderlich)

Beispiele

```
public void helloWorld(String result) throws Exception {
    set(1, "Hello World");
}
```

```
public void mod10(int x, int result) throws Exception {
    set(2, x % 10);
}
```

```
public void noEmployees(int result) throws Exception {
    Connection con = getConnection();
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(
           "SELECT COUNT(*) FROM EMPLOYEE");
    rs.next();
    set(1, rs.getInt(1));
    rs.close();
    stmt.close();
}
```

Externe Tabellenfunktionen in Java

- **Externe Tabellenfunktionen in Java**
  - Parameterart **DB2GENERAL**
  - Iterator-Konzept
    - Menge der Ergebnistupel wird iterativ zurückgeliefert (*fetch*)
    - ähnlich Cursor
    - Aufruftyp
      - *SQLUDF_TF_OPEN* → Öffnen
      - *SQLUDF_TF_FETCH* → Tupel zurückliefern
      - *SQLUDF_TF_CLOSE* → Schließen
      - bei Angabe von **FINAL CALL** zusätzlich *SQLUDF_TF_FIRST*, *SQLUDF_TF_FINAL*
  - Attribute der Tabelle
    - zusätzliche Parameter am Ende der Parameterliste der Methodendeklaration
    - setzen mittels `set(int index, *)`

Beispiel mit Instanzvariablen

-
```
private Statement stmtEmployees;
private ResultSet rsEmployees;

public void allEmployees(String name) throws Exception {
    switch (getCallType()) {
    case SQLUDF_TF_OPEN:
        Connection con = getConnection();
        stmtEmployees = con.createStatement();
        rsEmployees = stmtEmployees.executeQuery(
            "SELECT FIRSTNME, LASTNAME FROM EMPLOYEE");
        break;
    case SQLUDF_TF_FETCH:
        if (!rsEmployees.next()) {
            setSQLstate("02000");
        } else {
            set(1, rsEmployees.getString("FIRSTNME") + " "
                + rsEmployees.getString("LASTNAME"));
        }
        break;
    case SQLUDF_TF_CLOSE:
        rsEmployees.close();
        stmtEmployees.close();
        break;
    }
}
```

Ausgabeparameter nach Schema des Ergebnisses

```
SELECT * FROM TABLE(ALL_EMPLOYEES()) X
FULLNAME
----------------------
CHRISTINE HAAS
MICHAEL THOMPSON
…
```

Externe Tabellenfunktionen in Java

- **Speicherung von Zwischenergebnissen**
  - 2 Varianten
    - mittels Instanzvariablen der Klasse
    - mittels Scratchpad

- **Scratchpad**
  - Option **`SCRATCHPAD`** in der **`CREATE FUNCTION`**-Anweisung
  - Zwischenspeicher mit fest vorgegebener Länge (in Byte)
  - Abfragen mittels `getScratchpad()`
  - Setzen mittels `setScratchpad(byte[])`, gleiche Länge!
  - für Java-UDFs nicht notwendig

Beispiel mit Scratchpad

```
public void seq(int from, int to, int cur) throws Exception {
    // variables to read from SCRATCHPAD area
    byte[] scratchpad = getScratchpad();
    ByteArrayInputStream byteArrayIn = new ByteArrayInputStream(scratchpad);
    DataInputStream dataIn = new DataInputStream(byteArrayIn);

    // variables to write into SCRATCHPAD area
    byte[] buffer;
    ByteArrayOutputStream byteArrayOut = new ByteArrayOutputStream(scratchpad.length);
    DataOutputStream dataOut = new DataOutputStream(byteArrayOut);

    switch (getCallType()) {
    case SQLUDF_TF_OPEN:
        cur = from;
        dataOut.writeInt(cur);
        buffer = byteArrayOut.toByteArray();
        System.arraycopy(buffer, 0, scratchpad, 0, buffer.length);
        setScratchpad(scratchpad);
        break;
    case SQLUDF_TF_FETCH:
        cur = dataIn.readInt();
        if (cur > to) {
            setSQLstate("02000");
        } else {
            set(3, cur);
            cur++;
            dataOut.writeInt(cur);
            buffer = byteArrayOut.toByteArray();
            System.arraycopy(buffer, 0, scratchpad, 0, buffer.length);
            setScratchpad(scratchpad);
        }
        break;
    case SQLUDF_TF_CLOSE:
        break;
    }
}
```

In Java typischerweise Umsetzung mittels Instanzvariablen!

```
SELECT * FROM TABLE(SEQ(5, 6)) X
CNT
-----------
          5
          6
```

Entwicklung von java-basierten UDFs

- **Aktualisierung der Klassen**
  - DB2 lädt bereits geladene Klassen nicht neu
    → Instanzparameter KEEPFENCED
    - YES: Weiterverwendung bereits geladener Klassen (Produktionssystem)
    - NO: bei jedem UDF-Aufruf Klasse neu laden (Entwicklungssystem)

- **Kompilierung**
  - Bibliotheken
    - UDFs mit **DB2GENERAL**-Parameterart benötigen DB2-Library
    - befindet sich unter `~/sqllib/java/db2java.zip`
    - aber: bereits im Standardklassenpfad
  - Kompilierung mittels `javac <filename>`
  - Kopieren nach: `~/sqllib/function`

Nutzerdefinierte Funktionen (C)

Nutzerdefinierte Funktionen in C

- **Nutzerdefinierte Funktionen in C**
  - Ablage als Funktion
  - Registrierung über **`PARAMETER STYLE SQL`**
  - **`FENCED`** und **`UNFENCED`**-Modus

- **Aufbau des externen Namens (`EXTERNAL NAME`)**
  - Angabe der Objekt-Datei und Name der Methode
  - `<object-id>!<method-id>`
  - Ablage der Objekt-Datei
    - **`FENCED`**: in `sqllib/function`
    - **`UNFENCED`**: in `sqllib/function/unfenced`

Definition einer UDF in C

- **Definition**
  - als: **void** SQL_API_FN <fn-name>(<parameter>)
  - Parameter (Reihenfolge beachten!)
    - alle Eingabeparameter
    - Ausgabeparameter
    - Nullwert-Indikatoren für alle Eingabeparameter
    - Nullwert-Indikator für Ausgabeparameter
    - Makro **SQLUDF_TRAIL_ARGS**
      - char *SQLUDF_STATE: Rückgabe SQLSTATE
      - char *SQLUDF_FNAME: Name der Funktion
      - char *SQLUDF_FSPEC: spezifischer Name der Funktion
      - char *SQLUDF_MSGTX: Rückgabe Nachricht
    - oder: Makro **SQLUDF_TRAIL_ARGS_ALL**
      - für Funktionen mit Scratchpad und **FINAL_CALL**
      - beinhaltet **SQLUDF_TRAIL_ARGS**
      - SQLUDF_SCRATCHPAD SQLUDF_SCRAT: das Scratchpad
      - SQLUDF_CALL_TYPE *SQLUDF_CALLT: Art der Aufrufs
        - » SQLUDF_FIRST_CALL, SQLUDF_NORMAL_CALL, SQLUDF_FINAL_CALL
        - » SQLUDF_TF_FIRST, SQLUDF_TF_OPEN, SQLUDF_TF_FETCH, SQLUDF_TF_CLOSE, SQLUDF_TF_FINAL
    - SQLUDF_DBINFO *dbinfo: falls **DBINFO** angegeben

Beispiele

- ```
  void SQL_API_FN helloWorld(SQLUDF_CHAR *result,
                             SQLUDF_NULLIND *resultNullInd,
                             SQLUDF_TRAIL_ARGS)
  {
    strcpy(result, "Hello World");
  }
  ```

- ```
  void SQL_API_FN mod10(SQLUDF_INTEGER *x,
                        SQLUDF_INTEGER *result,
                        SQLUDF_NULLIND *xNullInd,
                        SQLUDF_NULLIND *resultNullInd,
                        SQLUDF_TRAIL_ARGS)
  {
    if (SQLUDF_NOTNULL(xNullInd)) {
      *result = *x % 10;
    } else {
      *resultNullInd = -1;
    }
  }
  ```

Beispiele

```
void SQL_API_FN noEmployees(SQLUDF_INTEGER *result,
                            SQLUDF_NULLIND *resultNullInd,
                            SQLUDF_TRAIL_ARGS)
{
  SQLHANDLE henv, hdbc, hstmt;
  SQLCHAR sql[] = "SELECT COUNT(*) FROM EMPLOYEE";

  /* connect to database */
  SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
  SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
  SQLConnect(hdbc, NULL, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS);

  /* run query */
  SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
  SQLExecDirect(hstmt, sql, SQL_NTS);
  SQLBindCol(hstmt, 1, SQL_INTEGER, result, 0, NULL);
  SQLFetch(hstmt);
  SQLFreeHandle(SQL_HANDLE_STMT, hstmt);

  /* disconnect from database */
  SQLDisconnect(hdbc);
  SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
  SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
```

Aktuelle Datenbankverbindung wird genutzt.

Externe Tabellenfunktionen in C

- **Speicherung von Zwischenergebnissen**
  - nur mittels Scratchpad
  - typische Vorgehensweise
    - Definition eines eigenen Datentyps für das Scratchpad
    - Scratchpadgröße = Größe des Datentyps
    - bei Funktionsaufruf: Casten des Scratchpads auf definierten Datentyp

Beispiel

```
struct SEQ_SCRATCHPAD {
  int cur;
};

void SQL_API_FN seq(SQLUDF_INTEGER *from, SQLUDF_INTEGER *to, SQLUDF_INTEGER *result,
                    SQLUDF_NULLIND *fromNullInd,  SQLUDF_NULLIND *toNullInd,
                    SQLUDF_NULLIND *resultNullInd, SQLUDF_TRAIL_ARGS_ALL)
{
  struct SEQ_SCRATCHPAD *scratchpad = (struct SEQ_SCRATCHPAD *)SQLUDF_SCRAT->data;
  switch (SQLUDF_CALLT) {
  case SQL_TF_OPEN:
    scratchpad->cur = *from;
    break;
  case SQL_TF_FETCH:
    if (scratchpad->cur > *to) {
        strcpy(SQLUDF_STATE, "02000");
    } else {
      *result = scratchpad->cur;
    }
    scratchpad->cur++;
    break;
  default:
    break;
  }
}
```

```
SELECT * FROM TABLE(SEQ(5, 7)) X
CNT
-----------
          5
          6
          7
```

- **Kompilierung / Binden**
  - Benötigte Bibliotheken (dynamisch anbinden)
    - DB2-Library
    - libpthread
  - Vordefiniertes Build-Skript /home/dbprog/build/bld-rtn-c.sh
    - ```
      #!/bin/sh
      DB2DIR=/opt/ibm/db2/V9.1

      # compile
      gcc -Wall -I$DB2DIR/include -g -c -o $1.o $1.c

      # build shared library
      gcc -I$DB2PATH/include -D_REENTRANT -I$DB2DIR/include
      -L$DB2DIR/lib32 -ldb2 -lpthread -shared -o $1.so $1.o
      rm $1.o

      # copy into directory for fenced routines
      cp -f $1.so ~/sqllib/function/
      ```
      ```

Nutzerdefinierte Funktionen (OLE)

Externe OLE DB Funktionen

- **OLE DB** *(Object Linking and Embedding Database)*
  - von Microsoft als ODBC-Ersatz entwickelt
  - ermöglicht einheitlichen Zugriff auf Daten
    - relationale und nichtrelationale Datenbanken
    - Textdateien
    - Tabellenkalkulationen
  - Unterscheidung zwischen
    - Consumer: benötigt Daten
    - Provider: liefert Daten

- **Aufbau des externen Namens (`EXTERNAL NAME`)**
  - DB2 ist OLE DB-Datenquelle bekannt: `<server>![<rowset>]`
  - sonst : `![<rowset>]!<connect-string>[!<collating-sequence>]`

Externe OLE DB Funktionen

- **Beispiel**
  - Anbindung der Northwind-Datenbank (Microsoft Access)
  -
```
CREATE FUNCTION ORDERS()
RETURNS TABLE (ORDERID INTEGER,
               CUSTOMERID CHAR(5),
               EMPLOYEEID INTEGER,
               ORDERDATE TIMESTAMP,
               REQUIREDDATE TIMESTAMP,
               SHIPPEDDATE TIMESTAMP,
               SHIPVIA INTEGER,
               FREIGHT DECIMAL(19,4))
LANGUAGE OLEDB
EXTERNAL NAME
    '!orders!Provider=Microsoft.Jet.OLEDB.3.51;
     Data Source=c:\sqllib\samples\oledb\nwind.mdb
     !COLLATING_SEQUENCE=Y'
```

Zusammenfassung

- **Externe nutzerdefinierte Funktionen**
  - Skalar- und Tabellenfunktionen
  - Java, C, OLE DB
  - Ort der Funktion über **`EXTERNAL NAME`** angegeben
  - Zwischenergebnisse speichern über Instanzvariablen oder Scratchpad
  - **`FENCED`**- und **`UNFENCED`**-Modus

- **Quelltext der Beispiele**
  - `Java: /home/dbprog/examples/09/UDF*.*`
  - `C: /home/dbprog/examples/09/udf-example.*`

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut Systemarchitektur, Professur Datenbanken

Datenbankprogrammierung
10. Externe Stored Procedures

Externe Stored Procedures

- **Externe Stored Procedures**
  - Definition und Implementierung sehr ähnlich externen UDFs (siehe Vorlesung 9)

- **Grundlegender Ablauf**
  1. Erstellen der Stored Procedure
  2. Kompilierung
  3. optional: Erzeugen einer Library
  4. Speichern des Binärcodes im Datenbankserver
     - `sqllib/function` oder `sqllib/function/unfenced`
  5. Registrierung mittels **`CREATE PROCEDURE`**

Syntax

- **Syntax**

```
CREATE PROCEDURE <name>
     (IN|OUT|INOUT <par-name> <par-type>, … )
[SPECIFIC <unique-name>]
 EXTERNAL NAME <external-name>                          sprachabhängig
 LANGUAGE C|JAVA|OLE
neu PARAMETER STYLE JAVA|DB2GENERAL|DB2SQL|GENERAL|GENERAL WITH NULLS
[[NOT] DETERMINISTIC]
[DYNAMIC RESULT SETS <number>]
[NO SQL | CONTAINS SQL |
     READS SQL DATA | MODIFIES SQL DATA]
[CALLED ON NULL INPUT]
[[NO] DBINFO]
[[NOT] FENCED]
neu [PROGRAM TYPE SUB | MAIN]
```

Syntax

- **Optionen**
  - siehe “9. Externe Funktionen”
  - **DBINFO**
    - Informationen über die Datenbank (Plattform, Application ID) an Stored Procedure weitergeben
  - **PROGRAM TYPE SUB**
    - Parameter werden als eigenständige Argumente übergeben
  - **PROGRAM TYPE MAIN**
    - Parameterübergabe wie in Main-Funktion (`argc` und `argv[]`)
    - Aufruf beliebiger Programme möglich
    - nur in C

Stored Procedures (Java)

Stored Procedures in Java

- **Stored Procedures in Java**
  - Alles genau wie bei externen UDFs
  - Methoden besitzen keinen Rückgabewert

- **Parameterart** JAVA
  - **OUT** und **INOUT**-Parameter als einelementige Arrays definiert
  - andere Parameter als einfacher Java-Datentyp

```
public static void <methode>(
                <Datentyp>[] <parametername>, …){
     <parametername>[0] = <value>;
}
```

Parameterart **JAVA**

- **Beispiel**

```
public static void mod10 (int x,
                          int[] result)
   throws Exception {
    result[0] = x % 10;
}

public static void getNoEmployees(String job, int[] num)
  throws Exception {
    Connection con =
         DriverManager.getConnection("jdbc:default:connection");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT count(*) "
                   + "FROM employee WHERE job='" + job + "'");
    rs.next();
    num[0] = rs.getInt(1);
    rs.close();
    stmt.close();
}
```

```
CALL mod10(25,?)
Value of output parameters
--------------------------
Parameter Name  : RESULT
Parameter Value : 5
Return Status = 0
```

```
CALL getNoEmployees('DESIGNER',?)
Value of output parameters
--------------------------
Parameter Name  : NUM
Parameter Value : 10
Return Status = 0
```

Parameterart **JAVA**

- **Beispiel**

```
CREATE PROCEDURE mod10 (IN x int, OUT res int)
LANGUAGE Java
PARAMETER STYLE JAVA
EXTERNAL NAME 'SPJavaStyleExample.mod10'


CREATE PROCEDURE getNoEmployees ( IN job CHAR(15),
                                  OUT num int)
LANGUAGE Java
PARAMETER STYLE DB2GENERAL
EXTERNAL NAME 'SPJavaStyleExample.getNoEmployees'
```

Parameterart **DB2GENERAL**

- **Parameterart DB2GENERAL**
  - Aufbau
    - Klassen erben von `COM.ibm.db2.app.StoredProc`
    - Klasse und Methode als **`public`** definiert
    - Jeder Parameter als **`INOUT`** definiert

- **Wichtige Methoden der `StoredProc`-Klasse**
  - `Connection getConnection()`
    - Liefert `Connection`-Objekt für aufrufende Datenbank
  - **`boolean`** `isNull(`**`int`**` index)`
    - Überprüfung eines Eingabearguments auf **`NULL`**-Wert
  - **`void`** `set(`**`int`**` index, *)`
    - Setzen von Ausgabeparametern (1-basiert)

Parameterart **DB2GENERAL**

- **Beispiel**

```
public void getNoEmployees(String job,
            int num) throws Exception {
  Connection con = getConnection();
  Statement stmt = con.createStatement();
  ResultSet rs = stmt.executeQuery("SELECT count(*) "
                  + "FROM employee WHERE job='" + job + "'");
  rs.next();
  set(2, rs.getInt(1));
  rs.close();
  stmt.close();
}
```

```
CALL getNoEmployees('DESIGNER',?)

Value of output parameters
--------------------------
Parameter Name  : NUM
Parameter Value : 10
Return Status = 0
```

Rückgabe von Ergebnismengen

- **Rückgabe von Ergebnismengen**
  - Verwendung eines **`ResultSet`**-Array als Parameter
  - Statement ausführen und ResultSet übergeben

```
ResultSet rs[0] = stmt.executeQuery();
```

  - Statement offen lassen, Verbindung schließen
  - **`DYNAMIC RESULT SETS`** <number> (maximal 32767)

Rückgabe von Ergebnismengen

- **Beispiel**

```
public static void getEmployees(String job, ResultSet[] rs)
   throws SQLException {
   Connection con = DriverManager.getConnection("jdbc:default:connection");
   String query = "SELECT firstnme, lastname "
                + "FROM employee " + "WHERE job=?";
   PreparedStatement stmt = con.prepareStatement(query);
   stmt.setString(1, job);
   rs[0] = stmt.executeQuery();
   con.close();
}
```

Statement nicht schließen

```
CREATE PROCEDURE GETEMPLOYEES
        ( IN job VARCHAR(15) )
DYNAMIC RESULT SETS 1
LANGUAGE JAVA
EXTERNAL NAME 'MyClass.getEmployees'
PARAMETER STYLE JAVA
```

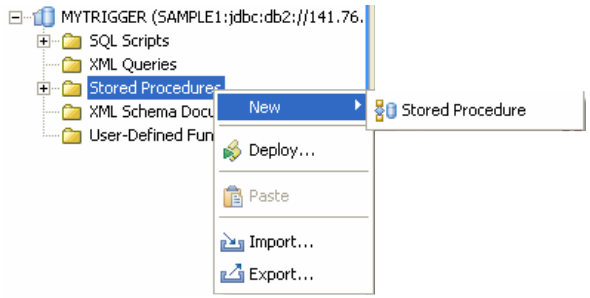
```
CALL getEmployees('DESIGNER')

FIRSTNME     LASTNAME
------------ ---------------
BRUCE        ADAMSON
ELIZABETH    PIANKA
MASATOSHI    YOSHIMURA
…            …
```

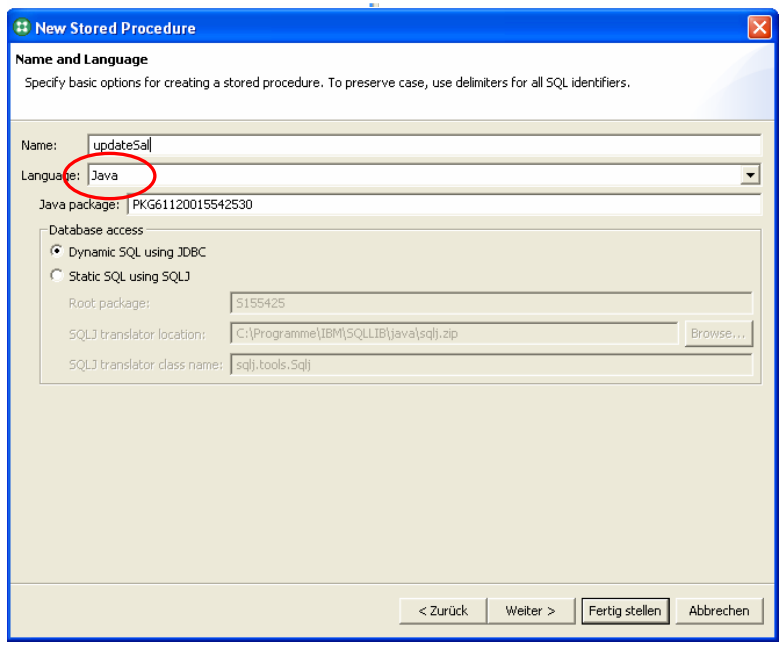
Stored Procedure-Entwicklung mit DB2DWB

- **Stored Procedure-Entwicklung mit DB2DWB**
  - Parameterart **JAVA**
    - Vorgehen siehe Abbildungen rechts
  - Parameterart **DB2GENERAL**
    - benötigen DB2-Library `~/sqllib/java/db2java.zip`
    - bereits im Standardklassenpfad

1



2



Stored Procedures (C)

- **Stored Procedures in C**
  - Unterstützte Parameterarten
    - **GENERAL:** Parameterübergabe über Argumente
    - **GENERAL WITH NULLS**: Zusätzliches NULL-Indikatorarray (**SHORTINT**-Array)
    - **DB2SQL**: Zusatzparameter müssen definiert werden
      - `nullinds[n]` (INOUT)
      - `char sqlst[6]` (OUT)
      - `char qualname[28]` (IN)
      - `char specname[19]` (IN)
      - `char diagmsg[71]` (OUT)
  - Zugriff auf Standard-I/O-Ströme nicht möglich (`printf()`, …)

Stored Procedures in C

- **Aufbau des externen Namens (`EXTERNAL NAME`)**
  - Angabe der Objekt-Datei und Name der Methode
  - `<object-id>!<method-id>`
  - Ablage der Objekt-Datei
    - **`FENCED`**: in `sqllib/function`
    - **`UNFENCED`**: in `sqllib/function/unfenced`

Beispiel

```
void SQL_API_FN getName(char *empNo, char *lastname)
{
   /* declare handles */

   SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
   SQLAllocHandle( SQL_HANDLE_DBC, henv, &hdbc);
   SQLConnect(hdbc, NULL, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS);
   SQLAllocHandle( SQL_HANDLE_STMT, hdbc, &hstmt);

   SQLPrepare(hstmt,"select lastname from employee where empno =?",
              SQL_NTS);

   SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
               SQL_CHAR, 0, 0, empNo, 6, NULL);
   SQLExecute(hstmt);
   SQLBindCol(hstmt, 1, SQL_C_CHAR, lastname, sizeof(lastname), NULL);
   SQLFetch(hstmt);

   /* release all handles */
}
```

Aktuelle Datenbankverbindung wird genutzt.

Beispiel

- **Beispiel**

```
CREATE PROCEDURE getName (IN empNo CHAR(6),
                          OUT lastname CHAR(20))
LANGUAGE C
PARAMETER STYLE GENERAL
PROGRAM TYPE SUB
EXTERNAL NAME 'sp-example.so!getName'
```

```
CALL getName('000010')

Value of output parameters
--------------------------
Parameter Name  : LASTNAME
Parameter Value : HAAS
Return Status = 0
```

PROGRAM TYPE MAIN

- **PROGRAM TYPE MAIN**
  - Erlaubt Parameterübergabe wie in Main-Funktion
  - Prozedur muss in Shared Library vorliegen
  - Prozedur muss zwei Parameter implementieren
    - Zähler, zum Beispiel `argc`
    - Array von Zeigern, zum Beispiel `char **argv`
  - `argv[0]` ist Name der Prozedur

PROGRAM TYPE MAIN

- **Beispiel**

```
void SQL_API_FN mainexample (int argc, char **argv)
  {
   SQLHANDLE henv, hdbc, hstmt;
   SQLCHAR sql[] = "SELECT AVG(salary) FROM employee WHERE job = ?";

   /* connect to database */

   /* run query */
   SQLPrepare(hstmt, sql, SQL_NTS);
   SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 0, 0,
                    (char *)argv[1], sizeof((char *)argv[1]), NULL);
   SQLExecute(hstmt);
   SQLBindCol(hstmt, 1, SQL_C_DOUBLE, (double *)argv[2], sizeof(double),
              NULL);
   SQLFetch(hstmt);
   SQLFreeHandle(SQL_HANDLE_STMT, hstmt);


   /* disconnect to database */
}
```

PROGRAM TYPE MAIN

- **Beispiel**

```
CREATE PROCEDURE MAIN_EXAMPLE (IN job CHAR(8),
                               OUT salary DOUBLE)
LANGUAGE C
PARAMETER STYLE GENERAL
PROGRAM TYPE MAIN
EXTERNAL NAME 'sp-example.so!mainexample'
```

```
CALL MAIN_EXAMPLE('DESIGNER',?)"

  Value of output parameters
  --------------------------
  Parameter Name  : SALARY
  Parameter Value : +5,7437000000000E+004
  Return Status = 0
```

Entwicklung C-basierter SPs

- **Kompilierung / Binden**
  - Genau wie UDFs (siehe Vorlesung 9)
  - Vordefiniertes Build-Skript unter `/home/dbprog/build/bld-rtn-c.java`

```
DB2DIR=/opt/ibm/db2/V9.1

gcc -Wall -I$DB2DIR/include -g -c -o $1.o $1.c

gcc -I$DB2PATH/include -D_REENTRANT -I$DB2DIR/include -
    L$DB2DIR/lib32 -ldb2 -lpthread -shared -o $1.so $1.o rm $1.o

cp -f $1.so ~/sqllib/function/
```

  - Ablage ebenfalls unter `~/sqllib/function/`

Zusammenfassung

- **Externe Stored Procedures**
  - Parameterarten
  - Rückgabe von Ergebnismengen
  - Ort der Funktion über **`EXTERNAL NAME`** angegeben
  - Stored Procedures in Java und C
  - **`PROGRAM TYPE MAIN`** und **`SUB`**

- **Quelltext der Beispiele**
  - `Java: /home/dbprog/examples/10/SP*.*`
  - `C: /home/dbprog/examples/10/sp-example.*`

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut Systemarchitektur, Professur Datenbanken

Datenbankprogrammierung
11. SQLJ

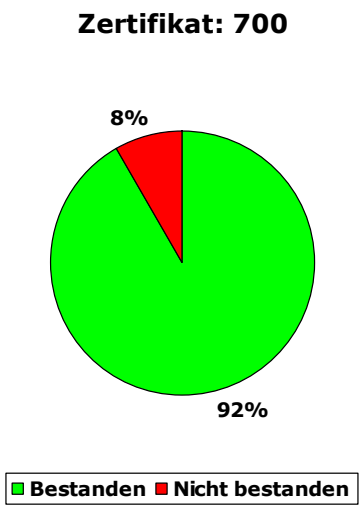
Zertifizierung

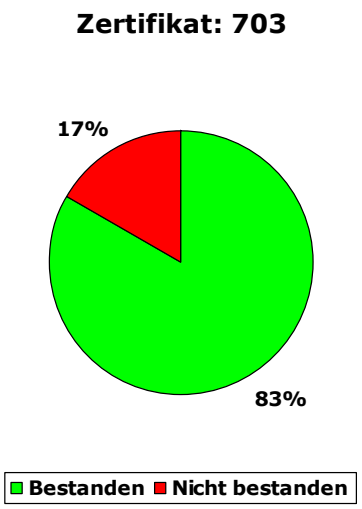
- **Wann?**
  - Dienstag, den 17.07.2007, ab 9:00
- **Wo?**
  - im Raum INF/E042
- **Voraussetzung**
  - Personalausweis (oder gleichwertiges Dokument)
  - 70 Punkte
- **Zertifikat-Aushändigung (online)**
  - http://www-03.ibm.com/certify/
  - Erfordert getrennte Registrierung
  - Prometric-ID aus der Ergebnis-E-Mail aufheben (!)

Online Test

- **Beispielfragen**
  - http://wwwdb.inf.tu-dresden.de/files/WS0607/DBPROG/tests/test700.php
  - http://wwwdb.inf.tu-dresden.de/files/WS0607/DBPROG/tests/test703a.php
  - http://wwwdb.inf.tu-dresden.de/files/WS0607/DBPROG/tests/test703b.php
  - Zugangsdaten in Vorlesung

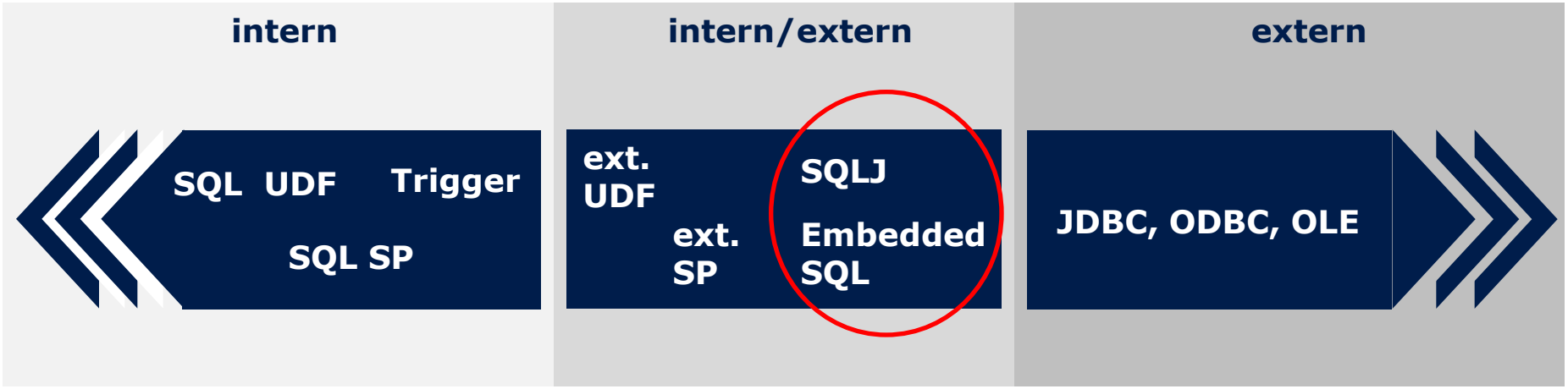
- **Offizieller Test**
  - kostenpflichtig, 10$
  - http://www-306.ibm.com/software/data/education/cert/assessment.html





SQLJ

Roadmap


intern
intern/extern
extern
SQL UDF
Trigger
SQL SP
ext. UDF
ext. SP
SQLJ
Embedded SQL
JDBC, ODBC, OLE


Kommunikationslast
Datenbankabhängigkeit

- **SQLJ**
  - Spracherweiterung aufbauend auf JDBC
  - ermöglicht Einbettung von SQL in Java
    - native Unterstützung von statischem SQL
    - dynamisches SQL weiterhin über JDBC
  - Standardisierung
    - entwickelt von Konsortium aus Oracle, Compaq/Tandem, JavaSoft, IBM, Informix und Sybase
    - später ANSI-Standard X3.135
      - Teil 0: Syntax für statisches SQL
      - Teil 1: Portierbare UDFs & Stored Procedures
      - Teil 2: Schnittstelle zwischen DBMS und Java
  - einfaches Beispiel

```
int count;
#sql { SELECT COUNT(*) INTO :count FROM LINEITEM };
```

SQLJ-Syntax

SQLJ-Anweisungen

- **SQLJ-Anweisungen**
  - beginnen immer mit `#sql`
  - 2 Arten
    - Deklaration (direkt nach **`import`**-Anweisungen)
    - Ausführung (in Methodenrumpf)
  - Deklarationen, z.B. `#sql context Ctx;`
    - Verbindungskontext
    - Benannter Iterator
    - Positionierter Iterator
  - Ausführung, z.B. `#sql { DROP TABLE EMPLOYEE };`
    - SQL-Anweisung (keine Ausgabewerte)
    - SQL-Zuweisung (>0 Ausgabewerte)

Verbindungskontext

- **Verbindungskontext** (*connection context*)
  - repräsentiert Datenbankverbindung
  - Voraussetzung für SQLJ-Ausführungsanweisungen
  - explizites Schließen notwendig: `close()`
  - JDBC-Verbindung mittels `getConnection()` ermittelbar

  - Variante 1: Setzen des Standard-Verbindungskontextes

```
Connection con = DriverManager.getConnection(<jdbc-url>);
DefaultContext defaultContext = new DefaultContex(con);
DefaultContext.setDefaultContext(defaultContext);
```

  - Variante 2: explizite Definition des Verbindungskontextes
    - Klassendeklaration: `#sql context <class-name>`
    - Instanziierung mittels Konstruktoren
      - `<class-name>(String jdbcUrl, boolean autoCommit)`
      - `<class-name>(String jdbcUrl, String username, String password, boolean autoCommit)`

Verbindungskontext

- **Beispiel**

```
#sql context Ctx;
…
// Verbindungskontext erstellen
Class.forName("com.ibm.db2.jcc.DB2Driver");
Ctx ctx = new Ctx("jdbc:db2:sample", true);

// JDBC verwenden
Connection con = ctx.getConnection();
DatabaseMetaData dbmd = con.getMetaData();
System.out.println("Datenbanksystem: "
    + dbmd.getDatabaseProductName());
System.out.println("Version          : "
    + dbmd.getDatabaseMajorVersion() + "."
    + dbmd.getDatabaseMinorVersion());

// Verbindung abbauen
ctx.close();
```

SQLJ-Ausführungsanweisungen

- **SQLJ-Ausführungsanweisungen**
  - generelle Syntax
    - `#sql [[<connection-context>]] { <sql-statement> };`
    - bei fehlender Angabe des Verbindungskontexts wird der Standard-Verbindungskontext verwendet

- **Hostvariablen**
  - können in SQLJ-Ausführungsanweisungen verwendet werden
  - Restriktionen wie bei SQL-basierten Stored Procedures
  - Syntax: `:[<modus> ]<name>`
    - Angegebene Java-Variable muss typkompatibel sein
    - Modus (meist implizit)
      - `IN`: lesender Zugriff
      - `OUT`: schreibender Zugriff
      - `INOUT`: lesender & schreibender Zugriff

SQLJ-Ausführungsanweisungen

- **Beispiel**

```
String name = "HAAS";
double salary;
#sql [ctx] { SELECT SALARY
             INTO :salary
             FROM EMPLOYEE
             WHERE LASTNAME=:name };
System.out.println(salary);
```

schreibender Zugriff

lesender Zugriff

- **Ausgabe**

```
152750
```

Iteratorkonzept

- **Iteratorkonzept**
  - Iteratoren dienen zur Abfrage von mengenwertigen Ergebnissen
  - Deklaration
    - Iteratoren verwenden eigene Klasse
    - Deklaration mittels
      `#sql iterator <class-name>(<type> [<name>], …)`
  - Verwendung
    - Zuweisung eines mengenwertigen Ergebnisses an Iterator
      `<class-name> <it-name>;`
      `#sql [[<con-context>]] <it-name> = { <select-stmt> };`
  - Auswertung (2 Arten)
    - positionierte Iteratoren
      - nur Attributtypen deklariert
      - nächstes Tupel: `#sql { FETCH :<it-name> into :<var1>,… };`
      - `endFetch()` ist wahr, wenn keine weiteren Tupel
    - benannte Iteratoren
      - Attributnamen & -typen deklariert
      - Abfrage mittels `<it-name>.<attribute-name>()`
      - nächstes Tupel: `<it-name>.next()`
      - `next()` ist falsch, falls keine weiteren Tupel

Iteratorkonzept

- **Beispiel: positionierter Iterator**

```
#sql iterator NamePIterator(String);
…
NamePIterator nameIt;
String name = null;
#sql [ctx] nameIt = { SELECT LASTNAME FROM EMPLOYEE };
#sql { FETCH :nameIt into :name };
while (!nameIt.endFetch()) {
  System.out.print(name + "; ");
  #sql { FETCH :nameIt into :name };
}
nameIt.close();
```

- **Ausgabe**

```
HAAS; THOMPSON; KWAN; GEYER; …
```

Iteratorkonzept

- **Beispiel: benannter Iterator**

```
#sql iterator NameNIterator(String lastname);
…
NameNIterator nameIt;
#sql [ctx] nameIt = { SELECT LASTNAME FROM EMPLOYEE };
while (nameIt.next()) {
  System.out.print(nameIt.lastname() + "; ");
}
nameIt.close();
```

- **Ausgabe**

```
HAAS; THOMPSON; KWAN; GEYER; …
```

Dynamisches SQL

- **Dynamisches SQL**
  - nicht unterstützt
  - Rückgriff auf JDBC notwendig

- **Dynamische Ergebnismengen**
  - Stored Procedures können dynamisch Ergebnismengen zurückliefern
  - Aufruf der Stored Procedure → SQLJ
  - Abfrage der Ergebnismengen → JDBC
  - Ausführungskontext
    - steuert Ausführung von SQL-Anweisungen
    - kapselt Ergebnis der Ausführung
    - Klasse `sqlj.runtime.ExecutionContext`
    - Methode `getExecutionContext()` des Verbindungskontextes
    - Zugriff auf Ergebnismengen über `getNextResultSet()`

Dynamische Ergebnismengen

- **Beispiel**

liefert Inhalt der Relation

liefert SQL-Anweisung

```
String sql = null;
#sql [ctx] { CALL GET_TABLE('EMPLOYEE', :OUT sql) };
System.out.println("PARAMETER 2 (SQL)           : " + sql);
System.out.print("RESULTSET (LASTNAME COLUMN): ");

ExecutionContext execCtx = ctx.getExecutionContext();
ResultSet rs = execCtx.getNextResultSet();
while (rs.next()) {
  System.out.print(rs.getString("LASTNAME") + " ");
}
System.out.println();
rs.close();
```

- **Ausgabe**

```
PARAMETER 2 (SQL): SELECT * FROM EMPLOYEE
RESULTSET (LASTNAME COLUMN): HAAS THOMPSON KWAN GEYER …
```

- **Vergleich SQLJ & JDBC**
  - Vorteile
    - höheres Abstraktionsniveau
    - einfachere Syntax
    - bessere Autorisierungskontrolle (Programm statt Nutzer)
    - Binden der Anweisungen in Datenbank (Vorkompilierung)
      - Effizienz
      - Gültigkeitsüberprüfung
      - performanzkritische Anweisungen aufspürbar
  - Nachteile
    - Präprozessor notwendig (aber: auch „standardisiert“)
    - mangelnde Unterstützung durch IDEs
    - keine Integration mit Persistenzframeworks (wie Hibernate)
    - (Dynamisches SQL nur über JDBC)

UDFs und SP mit SQLJ

- **SQLJ in nutzerdefinierten Funktionen und Stored Procedures**
  - Vorgehensweise
    - Deklaration eines Verbindungskontextes
    - Instanziierung des Verbindungskontext mit der JDBC-URL: `jdbc:default:connection`

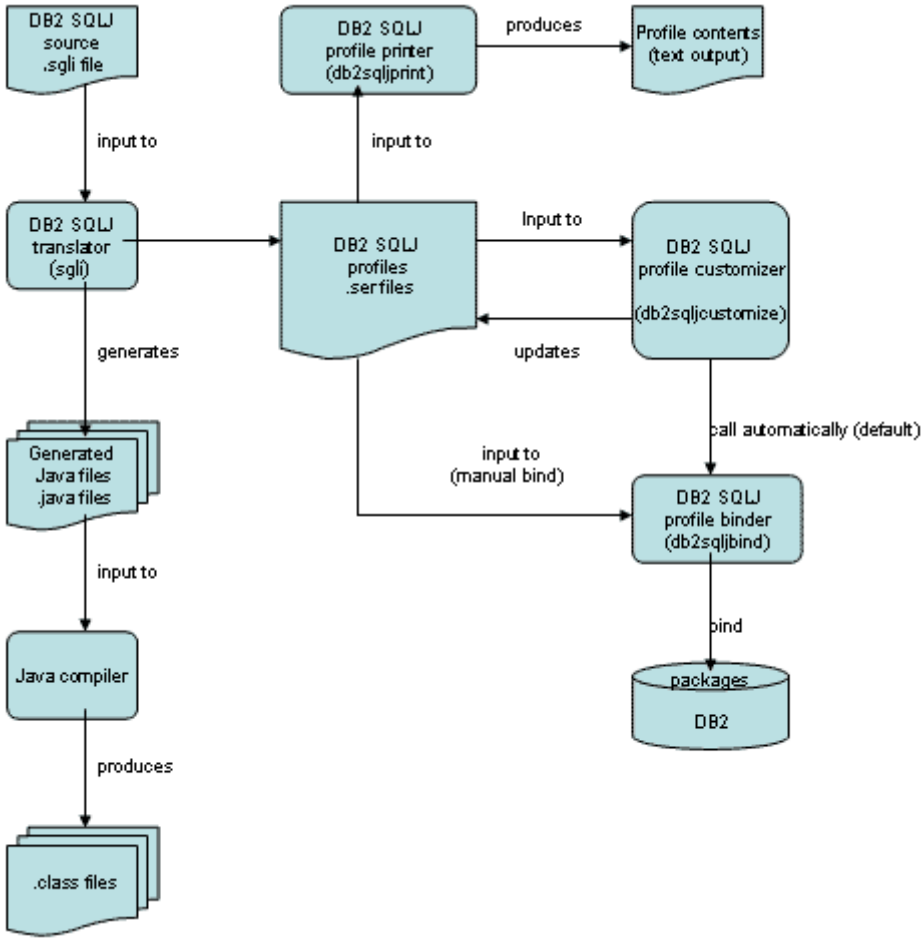
- **Beispiel (UDF mit Parameterart JAVA)**

```
#sql context Ctx;
…
public static int noEmployees() throws SQLException {
  Ctx ctx = new Ctx( "jdbc:default:connection", false);
  int noEmployees;
  #sql [ctx] { SELECT COUNT(*) INTO :noEmployees
               FROM EMPLOYEE };
  ctx.close();
  return noEmployees;
}
```

Entwicklung von SQLJ-Anwendungen

Grundlegende Vorgehensweise

- **Translator**
  - Präprozessor
  - standardisiert
  - erzeugt Java-Quellcode
  - erzeugt Profile
- **Profile**
  - binäre Repräsentation der SQL-Anweisungen
  - standardisiert → portierbar auf andere DBMS
  - Dateiendung `.ser`
- **Package**
  - Datenbankobjekt
  - gebundene SQL-Anweisungen
  - **`REBIND`** → erneutes Binden



Entwicklungsunterstützung

- **DB2 Developer Workbench**
  - unterstützt SQLJ-Syntax (Dateiendung `.sqlj`)
  - kann SQLJ-Programme nicht kompilieren/binden/ausführen
  - eigener Quellcode liegt unter `~/workspace/<project-name>`

- **Hilfsskript zur Anwendungserstellung**
  - liegt unter `/home/dbprog/build/build-app-sqlj.sh`
  - Inhalt
    - Aufruf des Präprozessor
    - Erzeugen eines Archives mit den kompilierten Dateien
    - Binden des Profils an die Datenbank(en)
    - Ausführen der Anwendung
  - Aufruf mittels: `build-app-sqlj.sh <class-name> <dbname> [<dbname2>]`
    - Angabe der SQLJ-Datei (`class-name`)
    - Angabe der (ersten) verwendeten Datenbank (`dbname`)
    - opt: Angabe der (zweiten) verwendeten Datenbank (`dbname2`)
  - Beispiel: `build-app-sqlj.sh SqljExamples SAMPLE TPCH`

Entwicklungsunterstützung

- **Hilfsskript zur Erstellung von UDFs/SPs**
  - liegt unter `/home/dbprog/build/build-rtn-sqlj.sh`
  - Inhalt
    - Aufruf des Präprozessor
    - Erzeugen eines Archives mit den kompilierten Dateien
    - Binden des Profils an die Datenbank
    - Installation des Archives in der Datenbank
  - Aufruf mittels `build-rtn-sqlj.sh <class-name> <dbname>`
  - Registrierung der Routine an DBMS trotzdem erforderlich
    - Aufbau des externen Namens: `[<jar-id>:]<class-id>!<method-id>`
    - hier: Angabe des Archivs erforderlich
  - Beispiel
    - `bldsqljrtn.sh SqljUDFExample SAMPLE`
    - ```
      CREATE FUNCTION NO_EMPLOYEES()
      RETURNS INTEGER
      EXTERNAL NAME 'SqljUDFExample:SqljUDFExample!noEmployees'
      LANGUAGE JAVA
      PARAMETER STYLE JAVA
      ```
      ```

Zusammenfassung

- **SQLJ**
  - Einbettung von SQL in Java-Programmen
  - standardisierte Spracherweiterung
  - Vorkompilierung der SQL-Anweisung
  - unterstützt Anwendungen und UDFs / Stored Procedures
  - Vorgehensweise: Translator → Javaquellen & Profile → Packages
  - dynamisches SQL nicht unterstützt
  - z.Zt. mangelnde Unterstützung durch IDEs

Alle Beispiele unter `/home/dbprog/examples/11`

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut Systemarchitektur, Professur Datenbanken

Datenbankprogrammierung
12. Embedded SQL

Embedded SQL

- **Embedded SQL**
  - ermöglicht Einbettung von SQL in C
  - keine API, sondern Spracherweiterung
  - unterstützt statisches und dynamisches SQL (anders als SQLJ)
  - Vor- und Nachteile wie bei SQLJ (siehe letzte Vorlesung)
  - Anweisungen beginnen immer mit `EXEC SQL`
  - Endung von Embedded SQL-Dateien ist "`.sqc`„

- **Beispiel**

```
EXEC SQL BEGIN DECLARE SECTION;
  int hvCount;
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT COUNT(*) INTO :hvCount FROM LINEITEM;
```

Verbindungsaufbau

- **Verbindungsaufbau**
  - Syntax
    ```
    EXEC SQL CONNECT TO <database> [USER <user> USING <pwd>]
    ```
  - eine Datenbankverbindung pro Transaktion (RUW, remote unit of work)

- **Verbindungsabbau**
  - `EXEC SQL CONNECT TO <another-database>` (nur RUW)
  - `EXEC SQL CONNECT RESET`
  - `EXEC SQL DISCONNECT`

- **Verteilte Transaktionen**
  - Mehrere Datenbankverbindungen pro Transaktion (DUW, distributed unit of work)
  - Aktivierung mittels Präcompileroptionen (benötigt sqlenv.h)
  - Wechseln der aktuellen Datenbank
    ```
    EXEC SQLSET CONNECTION <database>
    ```
  - Beispiel
    ```
    EXEC SQL CONNECT TO TPCH;
    EXEC SQL CONNECT TO SAMPLE;
    EXEC SQL SET CONNECTION TPCH;
    ```

Hostvariablen

- **Hostvariablen**
  - Kommunikation zwischen Anwendung und Datenbank
  - Deklaration in Deklarationsblock (mehrere erlaubt)

```
EXEC SQL BEGIN DECLARE SECTION
  <var-c-type> <var-name>; …
EXEC SQL END DECLARE SECTION
```

  - Hostvariablen müssen eindeutig in gesamter Anwendung sein
  - Namenskonvention
    - Präfix `hvIn` für Eingabevariablen
    - Präfix `hvOut` für Ausgabevariablen
  - Verwendung der Hostvariable in SQL-Blocks mit Doppelpunkt

```
:<var-name>
```

- **Beispiel**

```
EXEC SQL BEGIN DECLARE SECTION
  char hvInEmployeeID[7];
  double hvOutSalary;
EXEC SQL END DECLARE SECTION
```

SQL und C-Datentypen

SQL-Datentyp	C-Datentyp
SMALLINT	short
INTEGER	long
DECIMAL	-
DOUBLE	double

CHAR(n)	char[n+1] oder
	struct {
	  short length;
	  char data[n];
	}

VARCHAR(n)	char[n+1]
DATE	char[11]
TIME	char[9]
TIMESTAMP	char[27]

Beispiel

- **Beispiel**

```
EXEC SQL BEGIN DECLARE SECTION;
  char hvEmpno[7];
  struct {
    short length;
    char  data[10];
  } hvPhotoformat;
EXEC SQL END DECLARE SECTION;

EXEC SQL CONNECT TO SAMPLE;
EXEC SQL SELECT EMPNO, PHOTO_FORMAT
         INTO :hvEmpno, :hvPhotoformat
         FROM emp_photo
         WHERE empno='000130' and photo_format='bitmap';

printf("%-6s %-9s\n", hvEmpno, hvPhotoformat.data);

EXEC SQL CONNECT RESET;
```

db2dclgn

- **Automatische Generierung von Deklarationsblöcken**
  - Tool: `db2dclgn`
  - erzeugt automatisch Datenstruktur zur Deklaration einer Tabelle/View
  - Aufruf: `db2dclgn -d <database> -t <table> -o <file.h>`

- **Beispiel**

  `db2dclgn -d sample -t emp_photo -o sampledcl.h`

sampledcl.h

```
struct
{ char empno[7];
  struct
  { short length;
    char  data[10]; } photo_format;
  SQL TYPE IS BLOB(102400) picture;
  char emp_rowid[41];
} emp_photo;
```

Anfrageergebnisse verarbeiten

- **Anfrageergebnisse verarbeiten**
  - Einzeilige Ergebnismenge
    - Verwendung von **`SELECT INTO`** und **`VALUES INTO`**
  - Mehrzeilige Ergebnismenge
    - Mittels Cursor (vgl. SQL-Stored Procedures)
    - Name des Cursors muss anwendungsweit eindeutig sein

- **Indikatorvariablen**
  - Dienen zur Rück-/Übergabe von Nullwerten
  - Deklaration im `DECLARE`-Abschnitt, Datentyp `short`
  - Werden bei Anfrageausführung mit Hostvariable assoziert
    `[…] :<var-name> :<var-ind> […]` (kein Komma!)
  - Nullwert wenn Indikatorvariable < 0

- **Beispiel mit Cursor**

```
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
  char hvOutEmployeeNo[7];
  double hvOutSalary;
  short hvOutSalaryNI;
EXEC SQL END DECLARE SECTION;

[…]

EXEC SQL DECLARE C1 CURSOR FOR
         SELECT EMPNO, SALARY FROM EMPLOYEE;
EXEC SQL OPEN C1;
while (sqlca.sqlcode != 100) {
  EXEC SQL FETCH C1 INTO :hvOutEmployeeNo,
                         :hvOutSalary :hvOutSalaryNI;
  if (sqlca.sqlcode != 100 && hvOutSalaryNI < 0) {
    printf("Keine Gehaltsinformatinen für %s.",
        hvOutEmployeeNo);
  }
}
EXEC SQL CLOSE C1;
[…]
```

Enthält SQL-Code der letzten Anweisung

Hostvariable direkt gefolgt von Indikatorvariable

Dynamisches SQL

- **Dynamisches SQL**
  - Vorkompilieren einer SQL-Anweisung

    ```
    EXEC SQL PREPARE <stmt-name> FROM :<sql>;
    ```
  - Ausführen einer **`SELECT`**-Anweisung

    ```
    EXEC SQL DECLARE <cursor-name> CURSOR FOR <stmt-name>
    EXEC SQL OPEN <cursor-name> [USING :<var>, …]
    ```
  - Ausführen einer DDL/DML/DCL-Anweisung

    ```
    EXEC SQL EXECUTE <stmt-name> [USING :<var>, …]
    ```
  - Kompilieren und Ausführen in einem Schritt
    - kein **`SELECT`**, **`USING`** oder **`INTO`**
    - `SQL EXECUTE IMMEDIATE :stmt;`

Dynamisches SQL

- **Beispiel**

```
EXEC SQL BEGIN DECLARE SECTION;
    char hvInSqlStmt[80];
    char hvInJobType[10];
EXEC SQL END DECLARE SECTION;

EXEC SQL CONNECT TO SAMPLE;
strcpy(hvInSqlStmt, "UPDATE EMPLOYEE SET JOB = ? \
                      WHERE JOB = 'DESIGNER'");
strcpy(hvInJobType, "MANAGER");

EXEC SQL PREPARE sqlstmt FROM :hvInSqlStmt;
EXEC SQL EXECUTE sqlstmt USING :hvInJobType;
```

UDFs und SP mit Embedded SQL

- **UDFs und SP mit Embedded SQL**
  - Wie mit DB2 CLI
  - Verbindung wird implizit hergestellt
- **Beispiel** (Stored Procedure, Parameterart GENERAL)

```
SQL_API_RC SQL_API_FN getSalary(long *id, double *salary) {
  EXEC SQL INCLUDE SQLCA;
  EXEC SQL BEGIN DECLARE SECTION;
    long   hvInId;
    double hvOutsalary;
  EXEC SQL END DECLARE SECTION;

  hvInId = *id;
  EXEC SQL SELECT salary INTO :hvOutsalary
           FROM staff
           WHERE id=:hvInId;
  *salary = hvOutsalary;
  return 0;
}
```

Fehlerbehandlung

SQL Communications Area

- **SQL Communications Area** (SQLCA)
  - enthält Verbindung- und Fehlerinformationen
  - Definiert in `sqlca.h`, aber einbinden mit
    ```
    EXEC SQL INCLUDE SQLCA
    ```
  - stellt Variable `sqlca` zur Verfügung
  - nach Ausführung jeder SQL-Anweisung aktualisiert

- **Beispiel**

```
EXEC SQL INCLUDE SQLCA
…
EXEC SQL CONNECT TO SAMPLE
if (sqlca.sqlcode<0) { // Fehler!
  …
}
```

```
SQL_STRUCTURE  sqlca
{
  _SQLOLDCHAR    sqlcaid[8];
  sqlint32       sqlcabc;
  #ifdef DB2_SQL92E
  sqlint32       sqlcade;
  #else
  sqlint32       sqlcode;
  #endif
  short          sqlerrml;
  _SQLOLDCHAR    sqlerrmc[70];
  _SQLOLDCHAR    sqlerrp[8];
  sqlint32       sqlerrd[6];
  _SQLOLDCHAR    sqlwarn[11];
  #ifdef DB2_SQL92E
  _SQLOLDCHAR    sqlstat[5];
  #else
  _SQLOLDCHAR    sqlstate[5];
  #endif
};
```

SQLCA-Elemente

- **SQLCA-Elemente**
  - `char sqlcaid[8]`
    - enthält Kennung "SQLCA"
  - `int sqlcabc`
    - Größe der SQLCA-Datenstruktur (136 Byte)
  - `int sqlcode`
    - SQL-Return Code (`=0`: Erfolg, `>0`: Erfolg+Warnung, `<0`: Fehler)
  - `struct sqlerrm`
    - `char sqlerrmc[70]`
      - Fehlertokens, durch `FF` getrennt
    - `short sqlerrml`
      - Größe des `sqlerrmc`-Elements in Byte (0-70)
  - `char sqlerrp[8]`
    - Verwendeter DB2-Server, pppvvrrm (Beispiel `SQL09014`)
      - ppp: Produkt (SQL für DB2 unter UNIX/Windows)
      - vv: Version
      - rr: Release
      - m: Modifikations-Level

SQLCA-Elemente

- **SQLCA-Elemente**
  - `int sqlerrd[6]`
    - Diagnoseinformationen im Fehlerfall
    - Abhängig von Anweisung, z.B.
      - `sqlerrd[0]`: Rückgabewert einer Stored Procedure
  - `char sqlwarn[11]`
    - Indikator für Warnungen
    - jede Position entspricht bestimmter Warnung, z.B.
      - `sqlwarn[0]=='W'`: mindestens eine Warnung
      - `sqlwarn[4]=='W'`: **`UPDATE`**- oder **`DELETE`**-Anweisung ohne **`WHERE`**-Klausel
  - `sqlstate char[5]`
    - **`SQLSTATE`** der zuletzt ausgeführten SQL-Anweisung

Fehlerbeschreibungen

- **Fehlerbeschreibungen**
  - Get Error Message API
  - liefert textuelle Fehlerbeschreibung
  - `int sqlaintp (char *pBuffer, short sBufferSize, short sLineWidth, struct sqlca *pSQLCA);`
  - Argumente
    - `pBuffer`: Zieladresse des zu speichernden Textes
    - `sBufferSize`: Größe des Nachrichtenpuffers in Byte
    - `sLineWidth`: Anzahl der Zeichen vor Einschub eines Zeilenumbruchs (`=0` kein Umbruch)
    - `pSQLCA`: Zeiger auf die `SQLCA`-Datenstruktur
  - Negativer Rückgabewert → Fehler
  - Deklariert in `sql.h`

- **Beispiel**

```
int ret;
char msg[1024];
ret = sqlaintp(msg, sizeof(msg), 70, &sqlca);
```

WHENEVER-Anweisung

- **WHENEVER-Anweisung**
  - nach Ausführung jeder SQL-Anweisung sollte `sqlca.sqlcode` überprüft werden → hoher Programmieraufwand
  - `WHENEVER`-Anweisung legt Fehlerbehandlung global fest
  - `WHENEVER [SQLERROR|SQL WARNING|NOT FOUND] GOTO <label>;`
    - Präcompiler generiert Code und Sprungmarken für
      - Fehler
      - Warnungen
      - keine weiteren Daten (`sqlcode=100` bzw. `sqlstate='02000'`)
  - `WHENEVER [SQLERROR|SQL WARNING|NOT FOUND] CONTINUE;`
    - deaktiviert Fehlerbehandlung

WHENEVER-Anweisung

- **Beispiel**

```
EXEC SQL WHENEVER NOT FOUND GOTO error_handler;

EXEC SQL CONNECT TO SAMPLE;
EXEC SQL SELECT EMPNO INTO :hvInEmployeeNo
         FROM EMPLOYEE
         WHERE EMPNO = '000099';
…

EXEC SQL WHENEVER NOT FOUND CONTINUE;
goto exit;

error_handler:
  printf("SQL Code = %d\n", sqlca.sqlcode);
  EXEC SQL ROLLBACK;
  goto exit;

exit:
  EXEC SQL CONNECT RESET;
```

Anweisungsinformationen

SQL Descriptor Area (SQLDA)

- **SQL Descriptor Area (SQLDA)**
  - Informationen zu Hostvariablen & Spalten von `PREPARE`, `DESCRIBE`, `OPEN`, `FETCH` und `EXECUTE`-Anweisungen
  - Definiert in Header-Datei `sqlda.h`, aber Einbinden mit
    `EXEC SQL INCLUDE SQLDA;`
  - optional, außer bei Verwendung von `DESCRIBE`-Anweisungen

- **SQLDA-Header**
  - `char sqldaid[8]`
    - enthält Kennung "SQLDA"
  - `long sqldabc`
    - Größe der `SQLDA`-Datenstruktur (16 + 44*`sqln` Bytes)
  - `short sqln`
    - Anzahl der Elemente im `sqlvar`-Array
  - `short sqld`
    - Anzahl der Spalten bei `DESCRIBE`- oder `PREPARE`-Anweisungen oder Anzahl der Hostvariablen

```
SQL_STRUCTURE  sqlda
{
  _SQLOLDCHAR    sqldaid[8];
  long           sqldabc;
  short          sqln;
  short          sqld;
  struct sqlvar  sqlvar[1];
};

SQL_STRUCTURE  sqlvar
{
  short          sqltype;
  short          sqllen;
  _SQLOLDCHAR   *SQL_POINTER sqldata;
  short         *SQL_POINTER sqlind;
  struct sqlname sqlname;
};
```

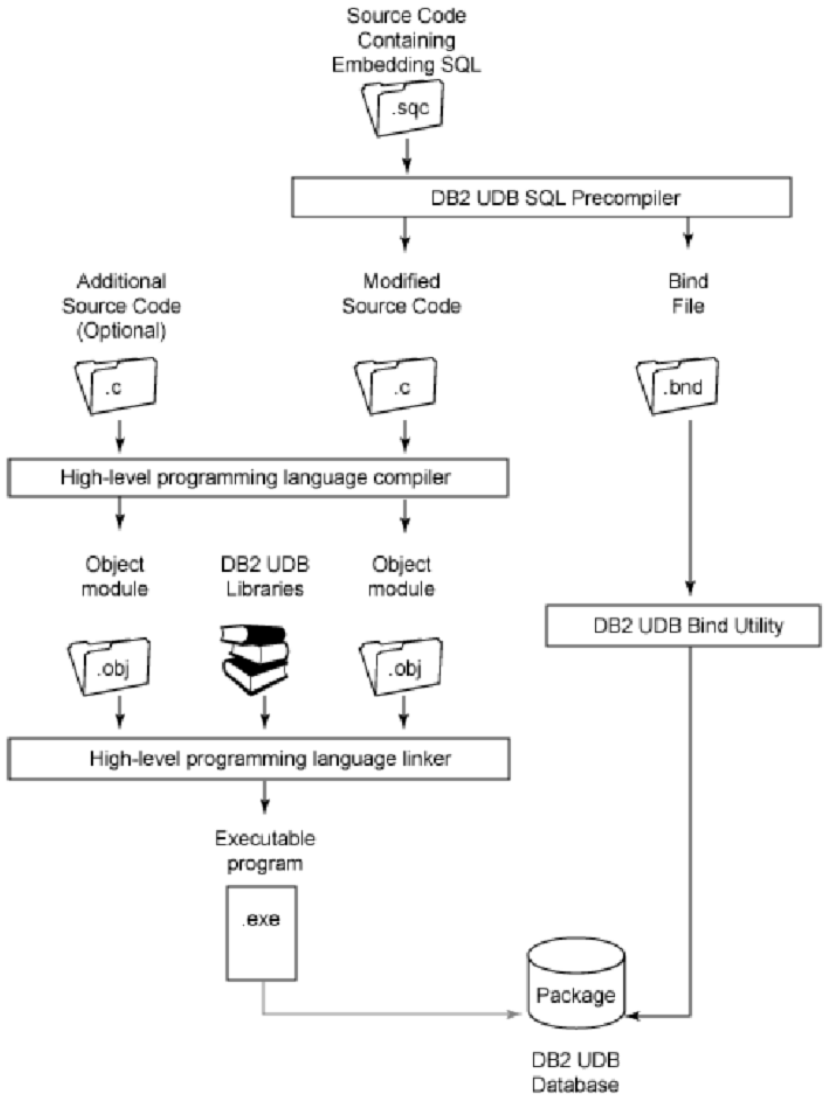
SQLDA-Elemente

- **SQLDA-Elemente**
  - `struct sqlvar sqlvar[]`
    - Informationen über die Hostvariablen bzw. Spalten
    - Je ein Eintrag pro Hostvariable
    - Aufbau
      - `short sqltype:` Datentyp
      - `short sqllen:` Größe in Byte
      - `char *sqldata:` Zeiger zur Speicheradresse
      - `short *sqlind:` Zeiger zum Nullindikator
      - `char sqlname[31]:` Name
  - Bei Verwendung nutzerdefinierter oder LOB-Datentypen
    - Element `sqldaid` enthält Wert "2" an siebter Stelle
    - Doppelt soviele Einträge im `sqlvar`-Array
    - Anderer Aufbau der zusätzlichen Einträge (Secondary SQLVAR)

Entwicklung von Embedded SQL-Anwendungen

Entwicklung von Embedded SQL-Anwendung

- **Präprozessor**
  - Datenbankverbindung notwendig
  - erzeugt C-Code
  - erzeugt Bind-Dateien (Anfrageinformationen)
  - automatisch an Datenbank gebunden oder später (*deferred*) mit DB2 UDB Bind Utility
  - `LIST PACKAGES`
- **Kompilieren**
  - erzeugt Objektmodul (Endung .o)
- **Linken**
  - Linken mit DB2 UDB- und anderen verwendeten Bibliotheken
  - Ergebnis ist ausführbares Programm



Hilfsskripte

- **Hilfsskripte**
  - wie gewohnt…
  - zur Anwendungserstellung
    - unter `/home/dbprog/build/build-app-sqc.sh`
    - Aufruf mittels: `build-app-sqc.sh <sqc-name> <dbname>`
    - Beispiel: `build-app-sqc.sh emb_appl.sqc SAMPLE`
  - zur Erstellung von UDFs/SPs
    - unter `/home/dbprog/build/build-rtn-sqc.sh`
    - Aufruf mittels: `build-rtn-sqc.sh <sqc-name> <dbname>`
    - Beispiel: `build-rtn-sqc.sh emb_sp.sqc SAMPLE`

```
CREATE PROCEDURE GETSALARY (IN id INT,
                            OUT salary DOUBLE)
LANGUAGE C
PARAMETER STYLE GENERAL
EXTERNAL NAME 'emb_sp.so!getSalary';
```

Zusammenfassung

- **Embedded SQL**
  - Spracherweiterung zur Einbettung von SQL in C-Programme
  - Statisches und dynamisches SQL
  - Anwendungen und UDFs / Stored Procedures

- **Fehlerbehandlung**
  - SQLCA-Datenstruktur
  - `WHENEVER`-Anweisung

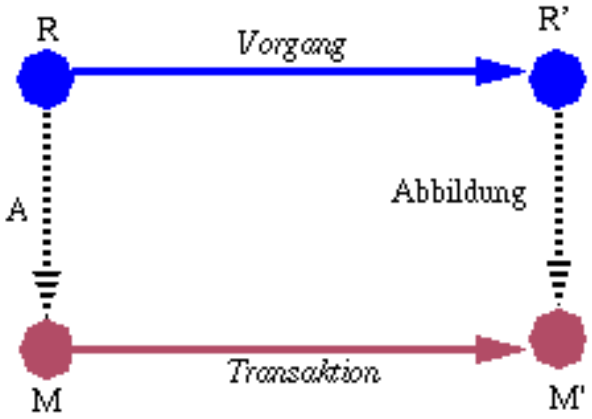
- **Anweisungsinformationen**
  - SQLDA-Datenstruktur

Datenbankprogrammierung
13. Potpourri

Synchronisation

Transaktion

- **Transaktion**
  - Folge von Datenbankoperationen
  - bildet Vorgang in der modellierten Miniwelt ab
  - alle Zugriffe erfolgen durch Transaktionen



R: Realitätsausschnitt (Miniwelt)

M: Modell der Miniwelt (beschrieben durch DB-Schema)

A: Abbildung aller wichtigen Objekte und Beziehungen

[Härder, Rahm 1999]

- **Probleme**
  - Nebenläufigkeit
  - Fehlersituationen (Systemfehler, Medienfehler, …)

ACID-Paradigma

- **Atomarität** (*atomicity*)
  - Alles-oder-Nichts-Prinzip
- **Konsistenz** (*consistency*)
  - logische Konsistenz
  - physische Konsistenz
  - gilt vor und nach der Transaktion
- **Isolation** (*isolation*)
  - virtueller Einbenutzerbetrieb
- **Dauerhaftigkeit** (*durability*)
  - Persistenz erfolgreicher Transaktionen

Mehrbenutzeranomalien

- **Lost Update**
  - mehrere Transaktionen lesen & ändern dieselbe Ressource
  - nur letzte Änderung bleibt sichtbar
- **Dirty Read** (uncommited read)
  - Lesen von nicht festgeschriebenen Daten
  - nicht festgeschriebene Daten werden zurückgesetzt
- **Nonrepeatable Read**
  - mehrfaches Lesen derselben Ressource
  - unterschiedliche Werte bei wiederholtem Lesen
- **Phantom**
  - bei wiederholtem Lesen erfüllen mehr Tupel die Selektionsbedingung

Verhinderung von Mehrbenutzeranomalien

- **Isolationsstufen** (*isolation levels*)
  - Sperren von Daten für andere Transaktionen
  - Zielkonflikt: Nebenläufigkeit vs. Isolation

- **Setzen der Isolationsstufe**
  - vor Datenbankverbindung
    - **`CHANGE ISOLATION TO {UR | CS | RS | RR}`**
  - während Datenbankverbindung
    - **`SET CURRENT ISOLATION {UR | CS | RS | RR | RESET}`**
  - für einzelne Anfragen (ab Version 8.1)
    - `<sql-stmt>` **`WITH {UR | CS | RS | RR}`**

Isolationsstufen

- **Uncommited Read (UR)**
  - Lesen von nicht festgeschrieben Änderungen
  - Schreiben nur auf bereits festgeschriebene Änderungen
  - Anomalien
    - Dirty Read, Nonrepeatable Read, Phantom
  - für Transaktionen
    - auf Tabellen, auf denen nicht geschrieben wird
    - für die Lesen unbestätigter Änderungen unkritisch ist

Isolationsstufen

- **Cursor Stability (CS)**
  - Sperren des vom Cursor aktuell referenzierten Tupels → kann von keiner anderen TA verändert werden
  - Freigabe bei
    - Repositionierung des Cursors (mittels **`FETCH`**)
    - Schließen des Cursors bzw. Transaktionsende
  - zusätzliches Sperren aller veränderter Tupel
  - Anomalien
    - Nonrepeatable Read, Phantom
  - Standard-Isolationstufe

Isolationsstufen

- **Read Stability (RS)**
  - Sperren aller gelesenen oder geänderten Tupel für gesamte Transaktion
  - Anomalien
    - Phantom

- **Repeatable Read (RR)**
  - Sperren aller referenzierten Tupel für gesamte Transaktion
  - Anomalien: keine

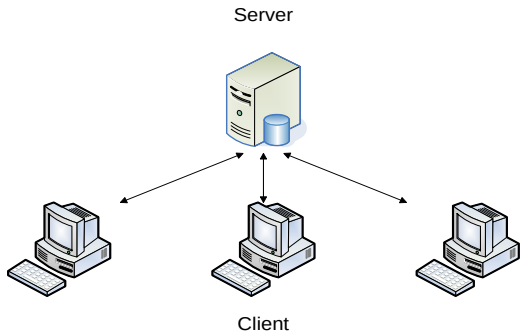
Sperren

- **Sperren**
  - Zuordnung von Ressourcen mit Transaktion
  - Ziel: Zugriff anderer Transaktion beschränken
    - Sperrenvergabe bei Zugriff auf Ressource
    - inkompatibler Zugriffsversuch → Warten bis sperrende Transaktion beendet
  - Attribute
    - **`OBJECT`**: gesperrte Ressource
    - **`DURATION`**: Zeitraum der Sperrenaktivität (vgl. Isolationsstufe)
    - **`MODE`**: Art des Zugriffs für Sperrinhaber und konkurrierende Transaktionen
  - Sperrmodi
    - X, S
    - weitere Sperrmodi → Datenbankadministration

Sicherheit: Authentifikation

Authentifikation

- **Authentifikation (*authentication*)**
  - Verifizierung der Identität eines Nutzers (Nutzername/Passwort)
  - durch Betriebssystem oder separates Produkt (Kerberos)
  - durch Server oder Client
- **Art der Authentifikation (*authentication types*)**
  - werden am Server und am Client gesetzt (müssen übereinstimmen)



  - **`UPDATE DBM CFG USING AUTHENTICATION`** `<auth_type>`
  - **`CATALOG DB`** `<db_name>` **`AT NODE`** `<node_name>` **`AUTHENTICATION`** `<auth_type>`

Authentifikation

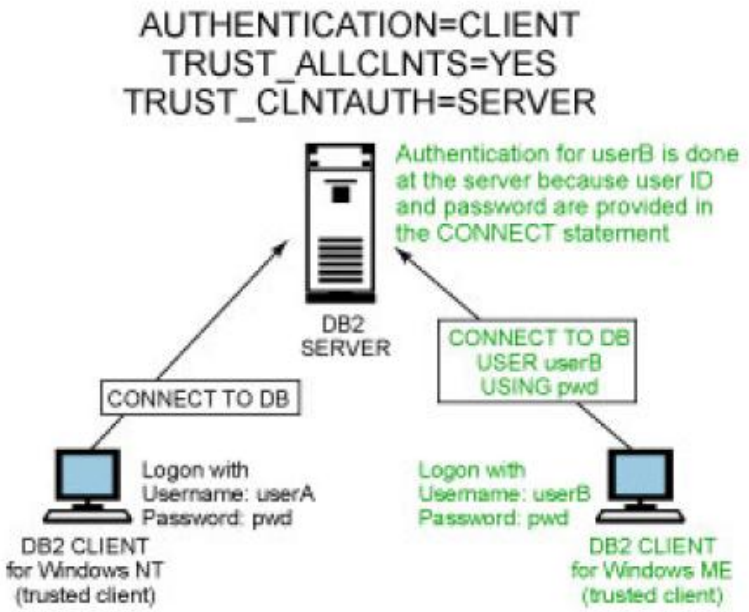
- **Arten der Authentifikation** (<auth_type>)
  - **SERVER / SERVER_ENCRYPT**
    = Authentifikation am Server
  - **CLIENT**
    = Authentifikation am Client
  - **KERBEROS**
  - **KRB_SERVER_ENCRYPT**
    = Kerberos & Server Encrypt; nur für Server
  - **DCE**
    = Authentifikation durch Fremdsoftware; nur für Client

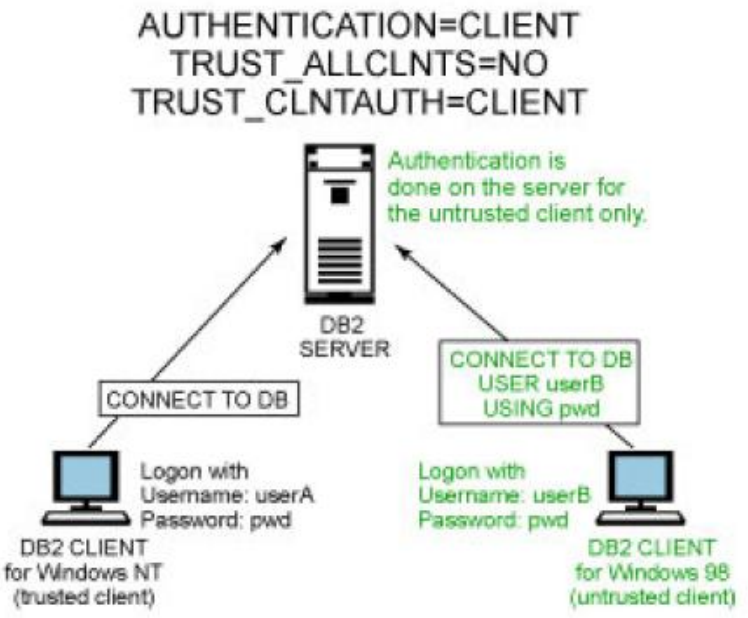
Client-Authentifikation

- **Client-Authentifikation**
  - **`TRUST_ALLCLNTS`**
    - Vertrauenswürdigkeit des Client-Betriebssystems (!)
    - **`UPDATE DBM CFG USING TRUST_ALLCLNTS YES|NO|DRDAONLY`**
      - **`YES`** ⇨ allen Clients vertrauen (unabh. vom Betriebssystem)
      - **`NO`** ⇨ nicht-vertrauenswürdige Clients am Server authentifizieren (z.B. Windows 98 und früher)
      - **`DRDAONLY`** ⇨ Distributed Relational Database Architecture-fähige Hostrechner
  - **`TRUST_CLNTAUTH`**
    - Festlegung des Authentifikationsortes, falls vertrauenswürdige Clients trotzdem Logininformationen angegeben
    - **`UPDATE DBM CFG USING TRUST_CLNTAUTH CLIENT|SERVER`**

Client-Authentifikation

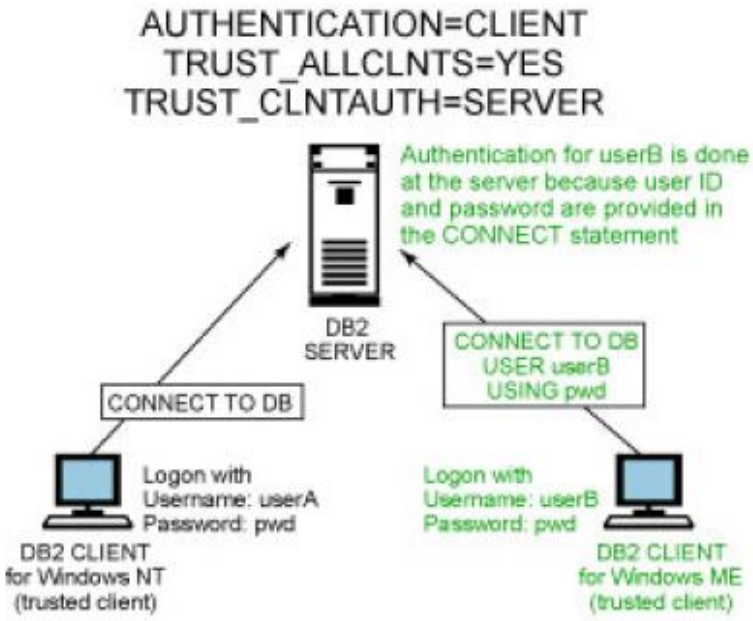
- **Beispiele**

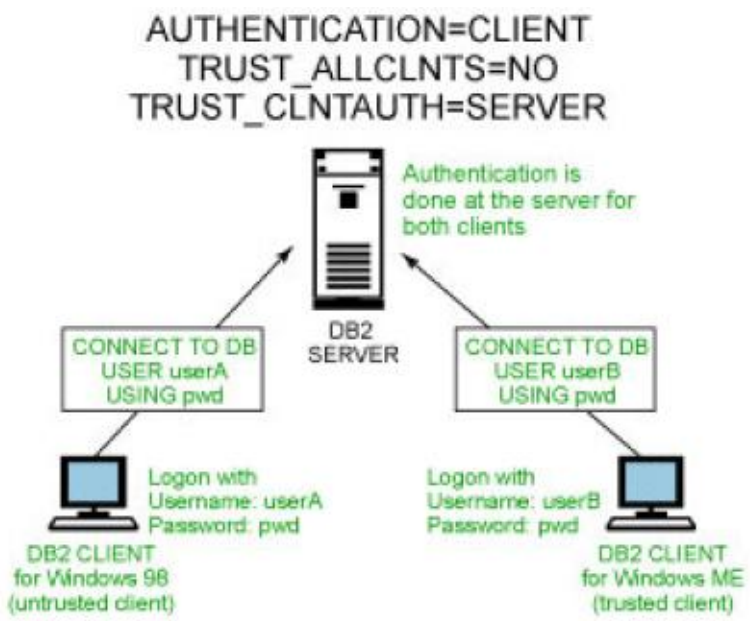




Client-Authentifikation

- **Beispiele**





Sicherheit: Autorisierung

Autoritäten

- **Autorisierung** (*authorization*)
  - Zugriffs- bzw. Ausführungsrechte für alle (!) Datenbankobjekte
  - Unter Kontrolle des DB2 Datenbank-Managers
  - Autoritäten

Zuweisung zu einer Gruppe

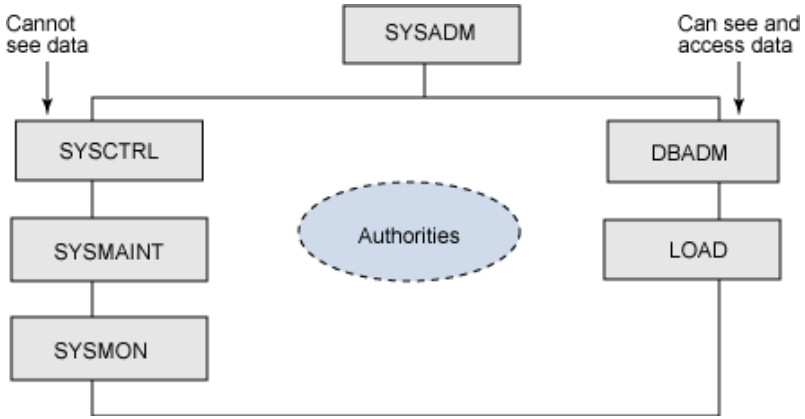
- **`SYSADM`**: Systemadministrator
  - Ausführen sämtlicher Verwaltungsaufgaben der DB2 Instanz
- **`SYSCTRL`**, **`SYSMAINT`**: System Control, System Maintenance
  - Keine Berechtigung zum Modifizieren der Instanzkonfiguration
  - Komplette Kontrolle über alle Datenbankobjekte (keine Leserechte !)
  - Ausführen von Verwaltungsaufgaben (Backup & Recovery, Ändern der Datenbankkonfiguration, etc.)

Zuweisung zu Nutzer/Gruppe

- **`DBADM`**: Datenbankadministrator
  - Komplette Kontrolle über eine Datenbank
  - Wird dem Ersteller einer Datenbank automatisch zugewiesen
- **`LOAD`**
  - Berechtigung zum Laden von Daten
  - Zusätzlich Privilegien auf entsprechende Datenbankobjekte nötig
    - » z.B.: Relation laden nur mit **`INSERT`** Privileg

Autoritäten

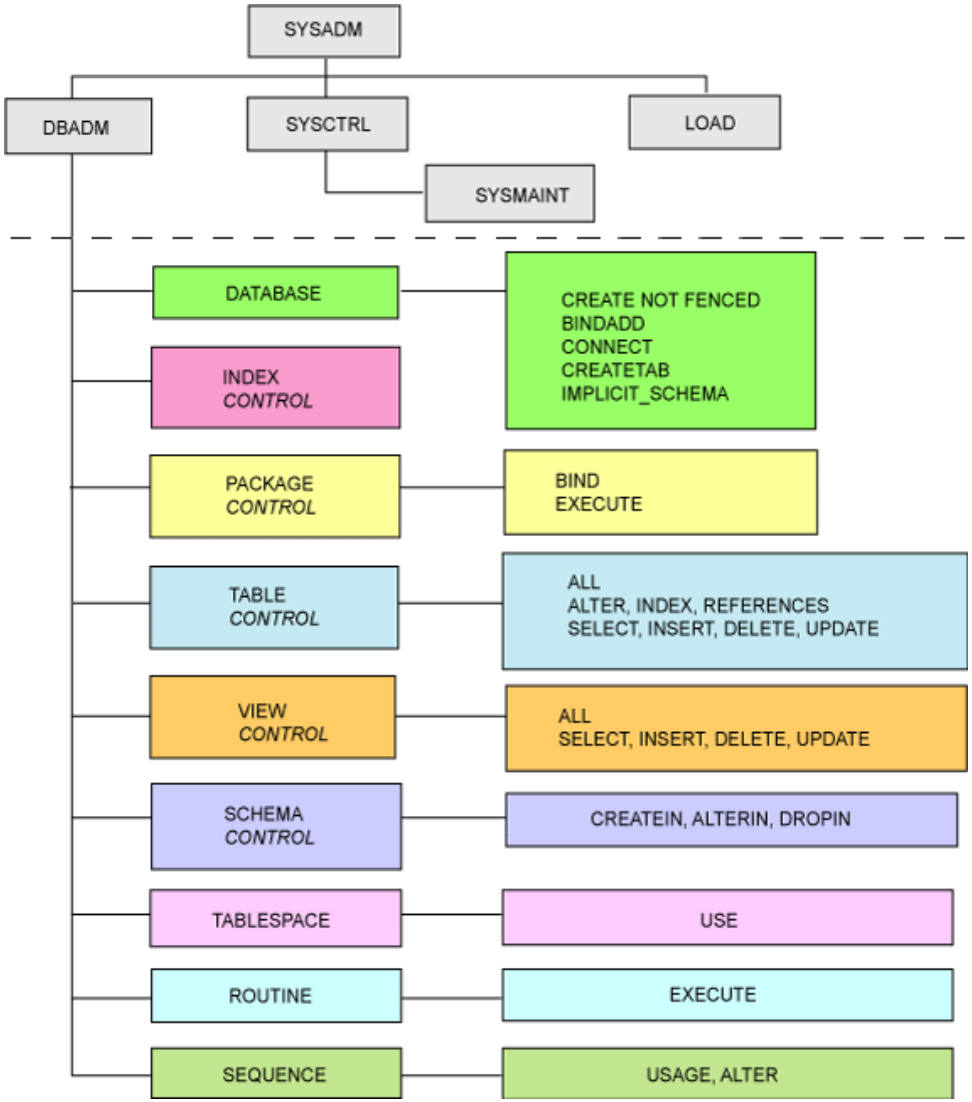
- **Zuweisen von Autoritäten**
  - **SYSADM, SYSCTRL, SYSMAINT**
    - Auf Instanzebene
    - **UPDATE DBM CFG USING**
      SYSADM_GROUP | **SYSCTRL_GROUP** | **SYSMAINT_GROUP** <group_name>
  - **DBADM, LOAD**
    - Auf Datenbankebene
    - **GRANT LOAD|DBADM**
      ON DATABASE <db_name>
      TO USER|GROUP <auth_name>
    - **GRANT INSERT**
      ON TABLE <tab_name>
      TO USER|GROUP <auth_name>



Privilegien

- **Privilegien** (*privileges*)
  - Recht zum Anlegen bzw. Zugriff auf Datenbankobjekte
  - Zuweisung
    - **GRANT** (DB2 Dokumentation)
    - **GRANT** <privilege> **ON** <obj> <obj_name>
      TO USER|GROUP <auth_name>
  - **WITH GRANT OPTION:** das Recht darf weiter vergeben werden (aber: Entziehen des Rechts damit nicht erlaubt)
  - Entzug
    - **REVOKE** (DB2 Dokumentation)
    - **REVOKE** <privilege> **ON** <obj> <obj_name> **FROM USER|GROUP** <auth_name>
  - Eigentümer bzw. Erzeuger eines Datenbankobjektes erhält **CONTROL**-Privileg und kann Rechte weitergeben

Privilegien



Privilegien

- **Implizite Privilegien**
  - Bei Vergabe von Privilegien/Autoritäten werden automatisch **einige** niedere Prioritäten mit vergeben = implizite Privilegien
  - bei Entzug des höherwertigen Privileges verbleiben die expliziten und impliziten Privilegien
  - z.B. `CONTROL` auf Table impliziert alle untergeordneten Privilegien, bei Entfernen des `CONTROL`-Privileges bleiben diese aber erhalten

Sequenzen

Sequenzen

- **Sequenz** (*sequence*)
  - Datenbankobjekt zur Generierung fortlaufender Werte
  - transaktional geschützt
  - nicht an eine Tabellenspalte gebunden (anders als *identity columns*)

```
CREATE SEQUENCE <seq-name>
[START WITH <numeric-constant>]
[INCREMENT BY <numeric-constant>]
[NO MINVALUE | MINVALUE <numeric-constant>]
[NO MAXVALUE | MAXVALUE <numeric-constant>]
[NO CYCLE | CYCLE]
[NO CACHE | CACHE <integer-constant>]
[NO ORDER | ORDER]
```

Sequenzen

- **Sequenz-Optionen**
  - **`INCREMENT BY`**: Intervall zwischen aufeinander folgenden Werten (Standard: 1)
  - **`MINVALUE`**: Haltepunkt für eine absteigende Sequenz
  - **`MAXVALUE`**: Haltepunkt für eine aufsteigende Sequenz
  - **`CYCLE`**: Neustart der Sequenz bei Erreichen des minimalen/maximalen Werts → Mehrfachgenerierung möglich
  - **`CACHE`**
    - Vorhalten von Sequenzwerten im Speicher aus Performanzgründen (Standard: 20)
    - gehen bei Systemfehler verloren → Lücken möglich
  - **`ORDER`**: Sortierung erzwingen

Sequenzen

- **Verwendung**
  - **NEXTVAL FOR** *<name>*: liefert nächsten Wert
  - **PREVVAL FOR** *<name>*: liefert aktuellen Wert

- **Beispiel**

```
CREATE SEQUENCE ORG_SEQ
START WITH 1
INCREMENT BY 1
NO MAXVALUE
NO CYCLE
CACHE 24

VALUES(NEXTVAL FOR ORG_SEQ);
```

1. Ausführung	2. Ausführung
1	1
--	--
1	2

Unicode

Unicode

- **Unicode und Unicode Transformation Format** (UTF)
  - Früher
    - eine Sprache in einer Datenbank
    - ASCII-Kodierung (1 Byte=1 Zeichen)
  - Heute
    - Repräsentation vieler Sprachen in einer Datenbank
    - neuer Zeichensatz (wurde) notwendig
  - Unicode-Standard
    - Entwickelt von ISO und Unicode-Consortium
    - Mehrere Bytes pro Zeichen
    - Kodierungen in DB2: UTF-8 und UTF-16
    - Sortierung von Zeichenketten über eine gemeinsame Textvergleichsfolge (*unicode collation algorithm*)

UTF-8

- **UTF-8**
  - 1-4 Byte pro Zeichen
  - Vorteile
    - ASCII ist Untermenge
    - einfache Portierung von ASCII (ohne Vergrößerung der DB)
    - in DB2 als **`CHAR`**, **`VARCHAR`** und **`LONG VARCHAR`** gespeichert
  - Nachteile
    - Zeichenkettenfunktionen arbeiten byte-basiert → Problem durch unterschiedliche Bytelängen je Zeichen
    - Java verwendet UTF-16 → aufwendige Konvertierung notwendig

UTF-16

- **UTF-16**
  - 2 Byte pro Zeichen
  - Vorteile
    - Einfache Zeichenkettenmanipulationen
    - keine Konvertierung bei Verwendung von Java
  - Nachteile
    - Größe der ACSII-Daten verdoppelt sich
    - in DB2 als **`GRAPHIC`**, **`VARGRAPHIC`** oder **`LONG VARGRAPHIC`**-Datentyp gespeichert
    - Portierung der Anwendungslogik notwendig

Verwendung von Unicode in DB2

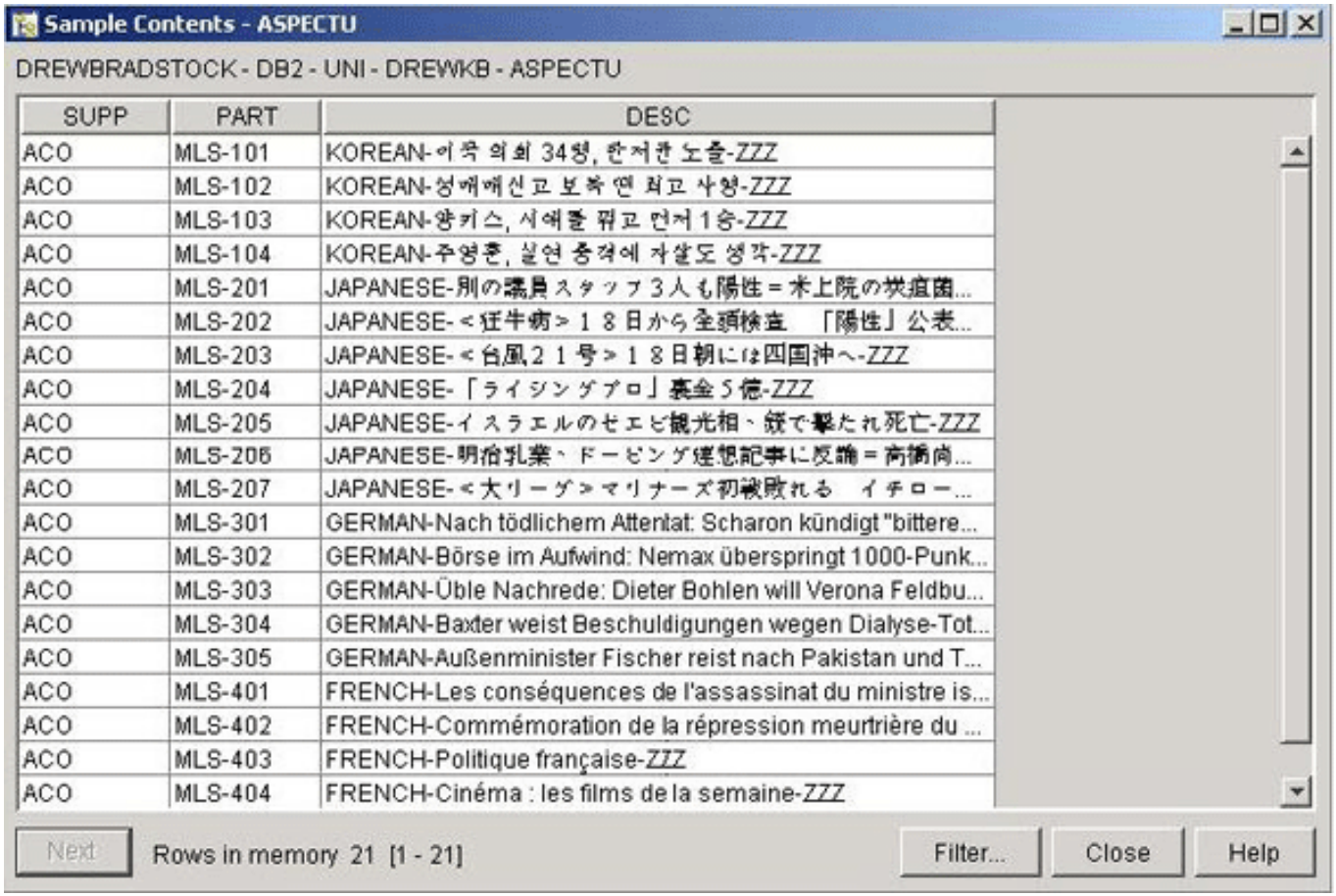
- **Verwendung von Unicode in DB2**
  - Festlegung der Kodierung nur bei Anlegen der Datenbank

    ```
    CREATE DATABASE <name>
    USING CODESET <codeset> TERRITORY <territory-name>
    ```
  - UTF-8 oder UTF-16 möglich (<codeset>)
  - Standardzeichensatz ist der des Betriebssystems
  - Ermitteln des Zeichensatzes → Datenbankkonfiguration

```
DB2 => GET DB CFG FOR <database-name>
Database territory = DE
Database code page = 1252
Database code set = IBM-1252
Database country code = 49
```

Anzeige von Unicode-Daten

- **Anzeige von Unicode-Daten**
  - Wichtigsten Fonts bereits in DB2 vorhanden



Zertifizierung

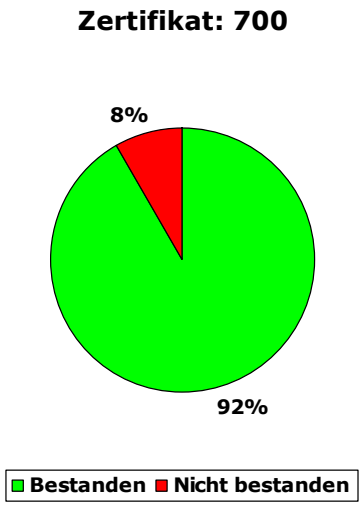
Zertifizierung

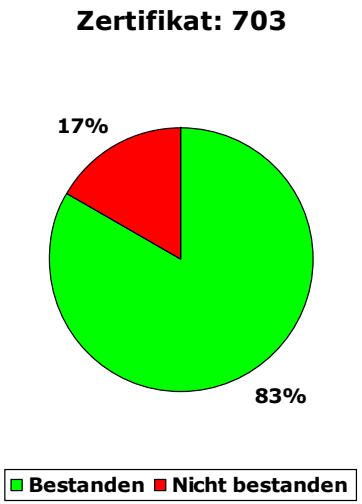
- **Wann?**
  - Dienstag, den 17.07.2007, ab 9:00 Uhr
- **Wo?**
  - im Raum INF/E042
- **Voraussetzung**
  - Personalausweis (oder gleichwertiges Dokument)
  - 70 Punkte
- **Zertifikat-Aushändigung (online)**
  - http://www-03.ibm.com/certify/
  - Erfordert getrennte Registrierung
  - Prometric-ID aus der Ergebnis-E-Mail aufheben (!)

Online Test

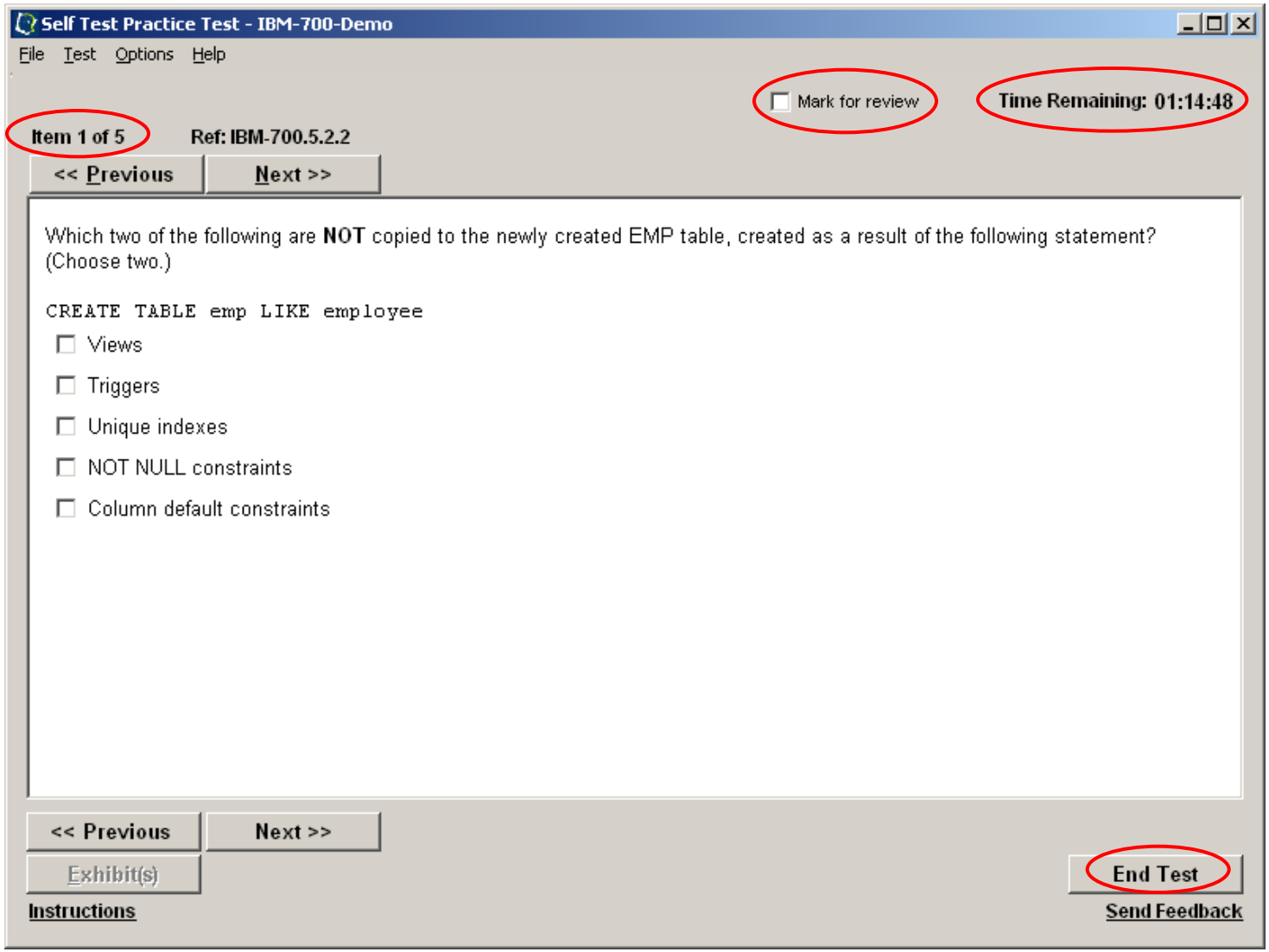
- **Beispielfragen**
  - http://wwwdb.inf.tu-dresden.de/files/WS0607/DBPROG/tests/test700.php
  - http://wwwdb.inf.tu-dresden.de/files/WS0607/DBPROG/tests/test703a.php
  - http://wwwdb.inf.tu-dresden.de/files/WS0607/DBPROG/tests/test703b.php
  - Zugangsdaten in Vorlesung

- **Offizieller Test**
  - kostenpflichtig, 10$
  - http://www-306.ibm.com/software/data/education/cert/assessment.html





Testoberfläche



Test 700
DB2 UDB V8.1 Family Fundamentals

Testdetails

- **Test**
  - 54 Fragen in 6 Teilen
  - 75 Minuten
  - 61% oder besser → bestanden
- **Webseite**
  - http://www-03.ibm.com/certify/tests/obj700.shtml

Section 1 - Planning (15%)

- **Section 1 - Planning**
  - Knowledge of DB2 UDB products (client, server, etc.) 
  - Knowledge of the features in DB2 tools such as: DB2 Extenders, Configuration Assistant, Visual Explain, Command Center, Control Center, Relational Connect, Replication Center, Development Center, and Health Center
  - Knowledge of Data warehouse and OLAP concepts
  - Knowledge of non-relational data concepts (extenders, etc)

Section 2 - Security (9%)

- **Section 2 - Security**
  - Knowledge of restricting data access ✓
  - Knowledge of different privileges ✓

- **Section 3 - Accessing DB2 UDB Data**
  - Ability to identify and locate DB2 UDB servers 
  - Ability to access and manipulate DB2 UDB objects 
  - Ability to create basic DB2 UDB objects 

- **Section 4 - Working with DB2 UDB Data**
  - Knowledge of transactions ✓
  - Given a DDL SQL statement, knowledge to identify results ✓
  - Given a DML SQL statement, knowledge to identify results ✓
  - Given a DCL SQL statement, knowledge to identify results ✓
  - Ability to use SQL to SELECT data from tables ✓
  - Ability to use SQL to SORT or GROUP data ✓
  - Ability to use SQL to UPDATE, DELETE, or INSERT data ✓
  - Ability to call a procedure ✓

- **Section 5 - Working with DB2 UDB Objects**
  - Ability to demonstrate usage of DB2 UDB data types ✓
  - Given a situation, ability to create table ✓
  - Knowledge to identify when referential integrity should be used ✓
  - Knowledge to identify methods of data constraint ✓
  - Knowledge to identify characteristics of a table, view or index ✓

- **Section 6 - Data Concurrency**
  - Knowledge to identify factors that influence locking ✓
  - Ability to list objects on which locks can be obtained ✓
  - Knowledge to identify characteristics of DB2 UDB locks
  - Given a situation, knowledge to identify the isolation levels that should be used ✓

Test 703
DB2 UDB V8.1 Family Application Development

Testdetails

- **Test**
  - 63 Fragen in 7 Teilen
  - 75 Minuten
  - 57% oder besser → bestanden
- **Webseite**
  - http://www-03.ibm.com/certify/tests/obj703.shtml

- **Section 1 - Database objects and Programming Methods**
  - Knowledge of naming conventions of DB2 objects (aliases, views, etc.)
  - Knowledge of the authorities needed to access data in an application ✓
  - Knowledge of complex database objects
  - Knowledge to identify the differences between dynamic and static embedded SQL ✓
  - Skill in determining when to use CLI/ODBC ✓
  - Skill in determining when to use JDBC, SQLJ ✓
  - Ability to determine when to use SQL routines and functions ✓
  - Determine when to use OLEDB ✓

Section 2 - Data Manipulation (26%)

- **Section 2 - Data Manipulation**
  - Ability to query database across multiple tables ✓
  - Ability to query tables across multiple databases (federated databases)
  - Knowledge of changing data ✓
  - Ability to use DB2 SQL functions ✓
  - Ability to use common table expressions ✓
  - Knowledge to identify when to use cursors in an SQL program ✓
  - Knowledge to identify types of cursors
  - Knowledge to identify the scopes of cursors ✓
  - Ability to manipulate cursors
  - Ability to manage a unit of work (transaction management) ✓

- **Section 3 - Embedded SQL Programming**
  - Knowledge of identifying steps and output involved in creating an embedded SQL programming application ✓
  - Knowledge to identify when host variables are used (begin-declare) ✓
  - Skill in declaring host variables ✓
  - Skill in utilizing host variable in queries ✓
  - Ability to explain/analyze the content of the SQLCA ✓
  - Knowledge of common errors, prep, and BIND database programs
  - Ability to connect to databases within an embedded SQL programming application ✓

- **Section 4 - ODBC/CLI Programming**
  - Knowledge of the different handle types ✓
  - Knowledge of configuring DB2 ODBC driver
  - Knowledge of problem determination (diagnostic records) ✓
  - Knowledge of the correct sequence for calling ODBC/CLI functions ✓
  - Knowledge of various CLI cursor types and when to use them
  - Ability to connect to databases within an ODBC/CLI programming application ✓

- **Section 5 - Java Programming**
  - Knowledge of various JDBC objects ✓
  - Knowledge of the difference between SQLJ and JDBC ✓
  - Knowledge of problem determination (JDBC Trace, SQL exceptions, JDBC error log) ✓
  - Skill in performing the steps to build SQLJ applications ✓
  - Ability to manage transactions across multiple databases (JTA)
  - Ability to connect to databases within a JAVA programming application ✓
  - Ability to determine which connection types should be used to connect to the database ✓

- **Section 6 - Advanced Programming**
  - Utilize dynamic and static SQL within programs ✓
  - Skill in casting UDTs within a program
  - Knowledge to identify when to use Compound SQL ✓
  - Knowledge of concurrency considerations within an application ✓
  - Knowledge of distributed unit of work
  - Knowledge of using parameter markers ✓
  - Determine the approaches to programming using Unicode ✓
  - Use performance enhancement features (buffered inserts)

- **Section 7 - User Defined Routines**
  - Knowledge to identify usage of UDFs 
  - Knowledge to identify when to use stored procedures 
  - Skill in using the DB2 Development Center
  - Knowledge of programming languages using the DB2 Development Center

Zusammenfassung

- **Synchronisation**
  - ACID
  - Mehrbenutzeranomalien
  - Isolationsstufen
- **Authentifikation**
  - Am Client vs. am Server
  - Vertrauenswürdige Clients
- **Autorisierung**
  - Instanz- und Datenbankautoritäten
  - Privilegien
- **Sequenzen**
  - Generierung eindeutiger Werte
- **Unicode**
  - Mehrbytezeichen
  - UTF-8 und UTF-16

Zusammenfassung

- **Zertifierung**
  - Test 700
  - Test 703

Fragen?