UNIVERSITÄT LEIPZIG

Enterprise Computing
Einführung in das Betriebssystem z/OS

Prof. Dr. Martin Bogdan
Prof. Dr.-Ing. Wilhelm G. Spruth

WS2012/2013

Transaktionsverarbeitung Teil 2

SQL

Arten von Datenbanken

Es existieren unterschiedliche Arten von Datenbanken. Die drei wichtigsten Grundtypen sind:

Relationale Datenbanken:

z.B. Oracle
DB2 – wichtigste Datenbank unter z/OS
Mircrosoft SQL

Nicht-relationale Datenbanken, unter z/OS häufig eingesetzt:

z.B. IMS
ADABAS

Objekt orientierte Datenbanken (wenig Bedeutung unter z/OS)

z.B. Poet

SQL

SQL (Structured Query Language) ist eine Programmiersprache für die Abfrage (query) und Abänderung (modify) von Daten und für die Administration von Datenbanken. SQL erlaubt

- **retrieval,**
- **insertion,**
- **updating, and**
- **deletion von Daten.**

In den 70er Jahren entwickelte eine Gruppe am IBM San Jose Research Laboratory das “System R” relationale Database Management System. Diese Arbeit basierte auf einer bahnbrechenden Veröffentlichung von IBM Fellow Edgar F. Codd: "A Relational Model of Data for Large Shared Data Banks". Donald D. Chamberlin und Raymond F. Boyce (beide IBM) entwickelten in 1974 die “Structured English Query Language” (SEQUEL) für die Manipulation von Daten des System R. Das Acronym SEQUEL wurde später in SQL abgeändert.

Oracle vs. DB2

Das erste relationale Datenbank Produkt wurde von der Firma Relational Software herausgebracht (später in Oracle umbenannt). IBM zögerte mit dem eigenen DB2 Produkt, weil das Leistungsverhalten gegenüber IMS als unzureichend angesehen wurde. Die Firma Oracle erbrachte aber den Nachweis, dass relationale Datenbanken ein attraktives Produkt sind trotz des deutlich schlechteren Leistungsverhaltens, und wurde damit Marktführer auf dem relationalen Datenbank-Sektor.

Das Leistungsverhalten von Relationalen Datenbanken wurde in den folgenden Jahrzehnten deutlich verbessert. IMS und Adabas haben aber nach wie vor ein besseres Leistungsverhalten, und werden deshalb nach wie vor eingesetzt.

Heute zerfällt der relationale Datenbankmarkt in 2 Teile: Relationale Datenbanken für Unix/Linux/Windows und für z/OS. Unter Unix/Linux/Windows hat Oracle nach wie vor einen höheren Marktanteil als DB2. Unter z/OS dominiert DB2; Oracle hat hier eine nur minimale Bedeutung. Ein interessantes Streitgespräch ist zu finden unter

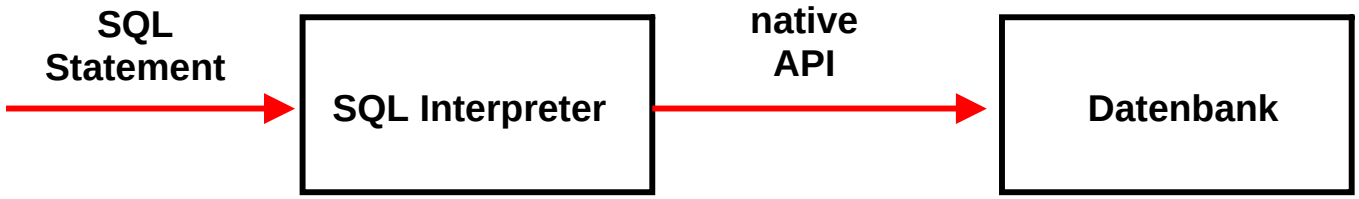
http://mainframeupdate.blogspot.com/2010/02/oracle-versus-ibm-and-db2.html

Die DB2 Version unter Unix/Linux/Windows ist kompatibel mit der DB2 Version unter z/OS. Die z/OS Version weist aber zahlreiche zusätzliche Funktionen auf, die in der Unix/Linux/Windows nicht vorhanden sind, z.B. Unterstützung für den Sysplex, die Coupling Facility, den Workload Manager, CICS usw.

Datenbank API

SQL wurde später von der ANSI und danach von der ISO standardisiert.. Die meisten Datenbank Management Systeme implementieren diese Standards, oft unter Hinzufügung proprietärer Erweiterungen. Diese proprietären Erweiterungen bringen in der Regel wesentliche Vorteile, und werden deshalb gerne benutzt. Trotz aller Standardisierungsbemühungen ist es aus diesem Grunde in der Regel nicht möglich, SQL Code von einer Datenbank auf eine andere Datenbank ohne Eingriffe zu portieren.

Dies gilt für die Nutzung unter Cobol, Fortran, REXX, C++, PL/1 oder ADA. Für die Nutzung unter Java existieren die Java Standards SQLJ und JDBC mit besserer Kompatibilität.



Neben SQL existiert für jede Datenbank eine nicht standardisierte, proprietäre, native Application Programming Interface (API). Ein Interpreter übersetzt bei jeder Anfrage das SQL Statement in die native API. Da dies Aufwand bedeutet, sind Zugriffe direkt in der nativen API in der Regel performanter als Zugriffe in SQL.

Embedded SQL, Beispiel für C++ (1)

Ein Anwendungsprogramm implementiert Datenbankzugriffe über eingebettete SQL-Anweisungen. Diese werden durch "exec sql" eingeleitet und durch ein spezielles Symbol (Delimiter, hier ";") beendet. Dies erlaubt einem Precompiler die Unterscheidung der exec sql Anweisungen von anderen Anweisungen.

```
main( )
{
     ........
     ........
     exec sql insert into PERS (PNR, PNAME) values
                    (4711, ´Ernie´);
     ........
     ........
}
```

```
exec sql include sqlca; /* SQL Communication Area*/
main ()
{
exec sql begin declare section;
    char X[8];
    int   GSum;
exec sql end declare section;
exec sql connect to dbname;
exec sql insert into PERS (PNR, PNAME) values (4711, 'Ernie');
exec sql insert into PERS (PNR, PNAME) values (4712, 'Bert');
printf ( "ANR ? ") ; scanf(" %s", X);
exec sql select sum (GEHALT) into :GSum from PERS     where ANR = :X;
printf ("Gehaltssumme %d\n", GSum)
exec sql commit work;
exec sql disconnect;
}
```

Embedded SQL, Beispiel für C++ (2)

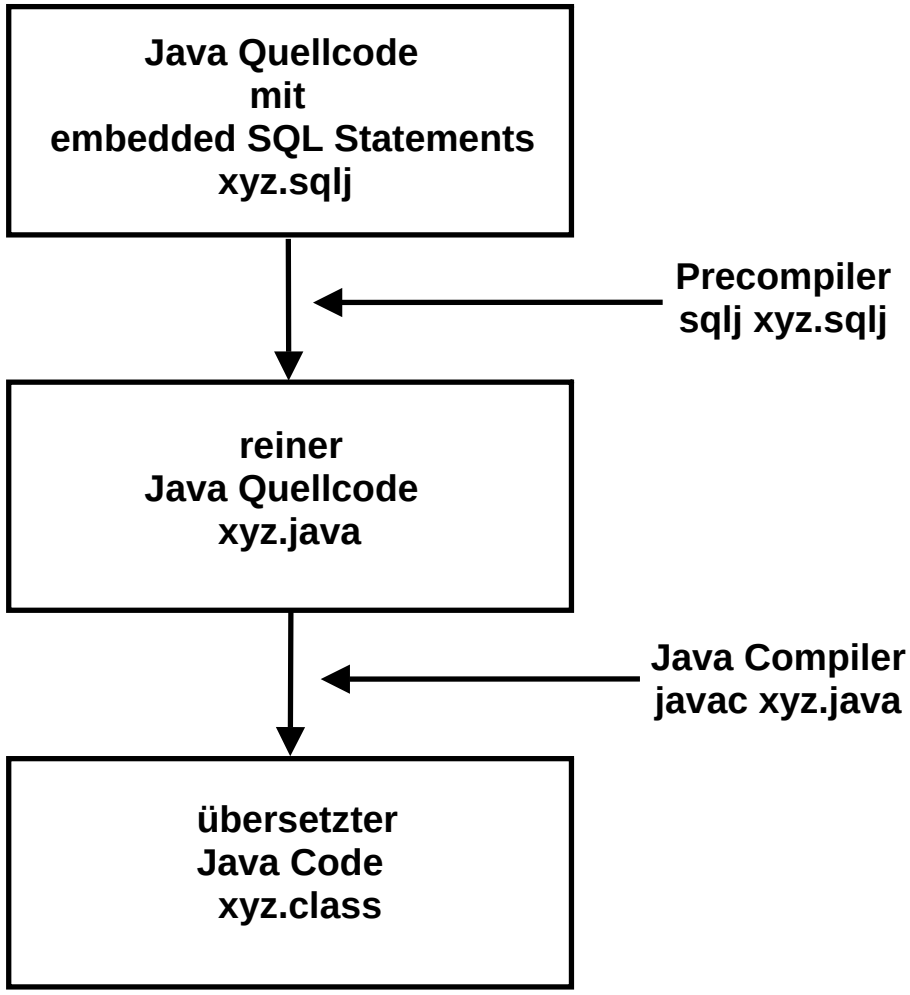
Eingebettete SQL-Anweisungen werden durch "EXEC SQL" eingeleitet und durch spezielles Symbol (hier ";") beendet, um einem Precompiler die Unterscheidung von anderen Anweisungen zu ermöglichen

Der Oracle oder DB/2 Precompiler greift sich die exec sql Befehle heraus und übersetzt sie in Anweisungen, die der C-Compiler versteht.

Die connect Anweisung baut die Verbindung zwischen Klienten und Server auf.

Es wird eine Kopie von einem Teil der DB Tabelle erstellt, gegen die alle SQL Befehle Änderungen vornehmen. Die commit Anweisung macht die vorhergehenden SQL Befehle entgültig.

Der Kommunikationsbereich SQLCA dient der Rückgabe von Statusanzeigern u.a..



Embedded SQL bedeutet, wir fügen SQL Kommandos in normale Programmiersprachen wie Cobol, PL/1, C/C++ oder Java ein.

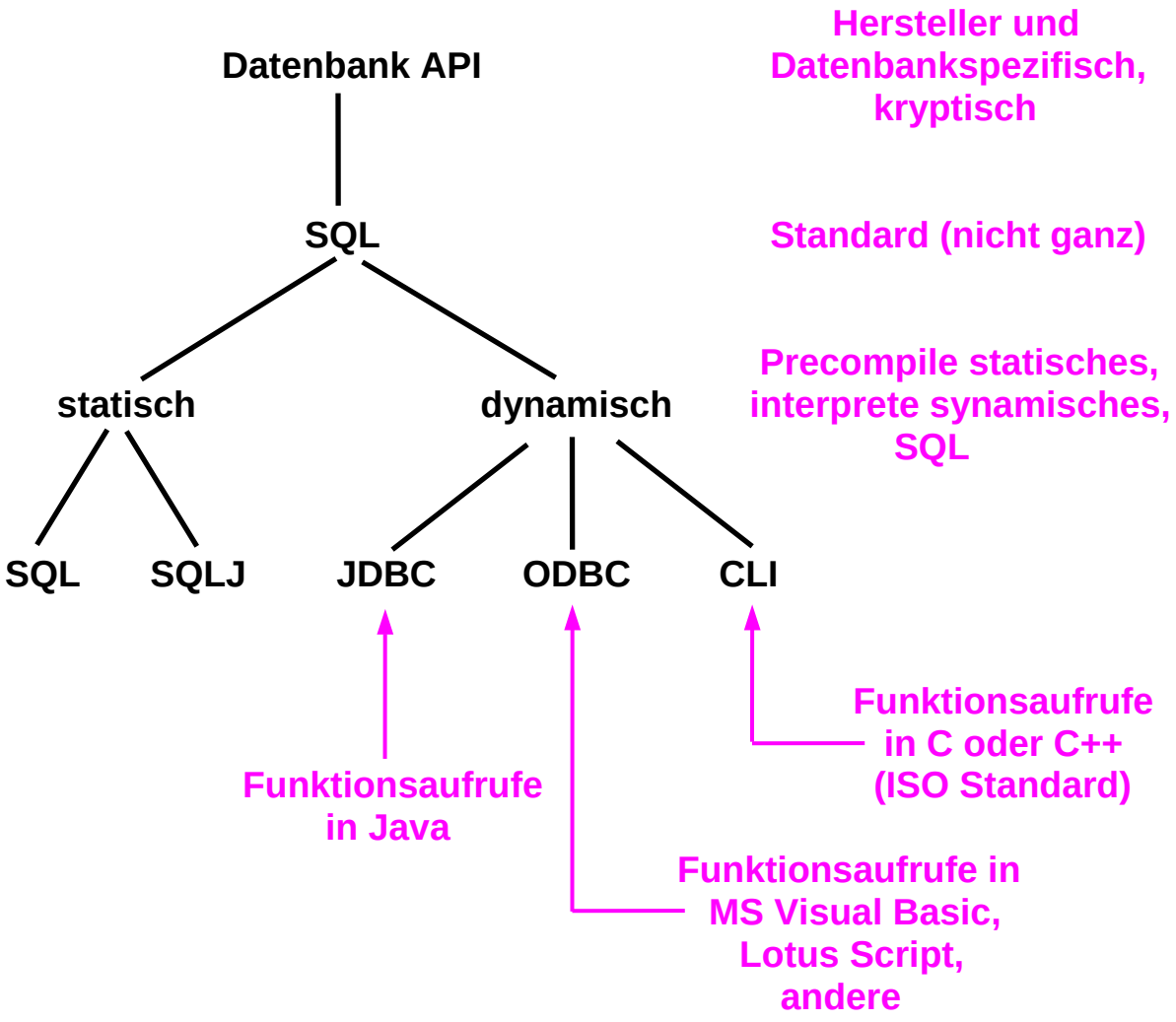
Der Cobol, C/C++ oder Java Compiler versteht die SQL Kommandos nicht. Sie müssen zunächst mit einem *Pre-Compile*r in Funktionsaufrufe der entsprechenden Programmiersprache übersetzt werden.

Der Precompiler übersetzt embedded SQL Kommandos in entsprechende Datenbank API Calls.

Relationale Datenbanken wie DB2, Oracle, Sybase erfordern unterschiedliche Precompiler

Beispiel: SQLJ Precompiler

Literatur: IBM Redbook *e-business Cookbook for z/OS Volume II: Infrastructure,*July 2002, section 4.5.



Wir unterscheiden zwischen statischem und dynamischem SQL.

Statisches SQL arbeitet mit einem Precompiler. Alle Angaben zu dem Datenbankzugriff müssen zur Compile Zeit festgelegt werden.

Dynamisches SQL ermöglicht Angabe von Parametern erst zur Laufzeit. Z.B. Benutzer gibt gewünschten Datenbanknamen und Tabellennamen in Datenfeld einer HTML Seite ein.

JDBC, ODBC und CLI ist dynamisches SQL für unterschiedliche Programmiersprachen

DB2connect ist ein Klient für statische SQL Zugriffe auf eine DB2 Datenbank. SQLJ ist die Java - spezifische Version von DB2connect.

Zugriff auf eine SQL Datenbank

```
#sql iterator SeatCursor(Integer row, Integer col,
       String type, int status);
Integer status = ?;
SeatCursor sc;
#sql sc = {
    select rownum, colnum from seats where status <= :status
   };
while(sc.next())
     {
     #sql { insert into categ values(:(sc.row()),
            :(sc.col())) };
     }
sc.close();
```

SQLJ

Statisches SQL Programmierbeispiel

In vielen Anwendungen benötigt statisches SQL weniger Zeilen Code als dynamisches SQL. Gezeigt sind hier zwei einfache Beispiele zum Vergleich von SQLJ mit JDBC.

```
Integer status = ?;
PreparedStatement stmt = conn.prepareStatement("select
        row, col from seats where status <= ?");
if (status == null) stmt.setNull(1,Types.INTEGER);
else   stmt.setInt(1,status.intValue());
ResultSet sc = stmt.executeQuery();
while(sc.next())
{
int row = sc.getInt(1);
boolean rowNull = sc.wasNull();
int col = sc.getInt(2);
boolean colNull = sc.wasNull();
PreparedStatement stmt2 =
        conn.prepareStatement("insert into categ
        values(?, ?)");
if (rowNull) stmt2.setNull(3,Types.INTEGER);
else stmt2.setInt(3,rownum);
if (colNull) stmt2.setNull(4,Types.INTEGER);
else stmt2.setInt(4,colnum);
stmt2.executeUpdate();
stmt2.close();
}
sc.close();
stmt.close();
```

JDBC

Ein dynamischer Datenbankzugriff auf eine relationale Datenbank aus einem Java Anwendungsprogramm heraus verwendet das JDBC (Java Data Base Connectivity) Zugriffsverfahren. Es bietet höhere Flexibilität als das statische Verfahren. Z.B. muss der Name der Datenbank erst im Augenblick des Zugriffs und nicht schon zur Compile-Zeit angegeben werden.

Ein Nachteil ist häufig eine schlechtere Performance. Auch ist der Programmieraufwand größer, wie ein Vergleich mit dem vorhergehenden statischen SQLJ Beispiel zeigt.

JDBC Programmierbeispiel

Vor- und Nachteile von statischem gegenüber dynamischem SQL

Was sind die Vor- und Nachteile von SQLJ gegenüber JDBC in Bezug auf Performance?

Die Antwort ist, es hängt sehr stark von den Umständen ab. Ein guter Vergleich ist zu finden unter http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.doc/ad/c0005785.htm .

Im Allgemeinen hat ein Anwendungsprogramm mit dynamischen SQL größere Start-up (oder initiale) Kosten pro SQL Statement, weil das SQL Statement vor der Benutzung erst übersetzt (compiled) werden muss. Nach der Übersetzung sollte die Ausführungszeit beim dynamischen SQL ähnlich sein wie beim statischen SQL. Wenn mehrere Klienten das gleiche dynamische Programm mit den gleichen Statements aufrufen, benötigt nur der erste Benutzer den zusätzlichen Übersetzungsaufwand.