DB-Entwurf & Anwendungsentwicklung

- Entwurfsaufgabe und Entwurfsprozeß
- Phasenmodell
- ER-Abbildung auf das Relationenmodell
- Relationaler Datenbankentwurf
  - Funktionale Abhängigkeiten
  - Normalformen
  - Transformationseigenschaften
- Anwendungsentwicklung mit SQL
  - Call-Level-Schnittstellen
  - Embedded SQL

Entwurfsaufgabe

- Anwendungsdaten sollen aus den in der Datenbank gespeicherten Daten (möglichst effizient) ableitbar sein
- Speicherung von *vernünftigen* Daten: Ermittlung des (aktuellen und zukünftigen) Informationsbedarfs
- möglichst nicht-redundante Darstellung der Anwendungsdaten (Speicherplatz sparen, Vermeidung von Anomalien)

Vorgehen und Eigenschaften

- **Entwurfsprozeß:**
  - Abfolge von Entwurfsdokumenten (Modellierungen),
  - die von einer abstrakten, anwendungsnahen Beschreibungsebene
  - hin zur tatsächlichen Realisierung der Datenbank führen
- **Eigenschaften:**
  - Informationserhaltung (Erhaltung der speicherbaren Informationen)
  - Konsistenzerhaltung (Erhaltung der Einschränkungen und Regeln)

Phasenmodell

- **Anforderungsanalyse**
  - Sammlung und Analyse des Informationsbedarfs
  - Ergebnis: informale Beschreibung
  - Methoden: Interview, Analyse existierender Arbeitsabläufe, Dokumente, Formulare usw.
- **Konzeptioneller Entwurf**
  - formale Beschreibung des Fachproblems unabhängig vom später zu verwendenden System
  - Teilschritte:
    - Modellierung von verschiedenen Sichten
    - Analyse bzgl. auftretender Konflikte
    - Integration der Sichten
  - Ergebnis: konzeptionelles Gesamtschema (ER, EER)

Phasenmodell

- **Verteilungsentwurf**
  - Festlegung der Art der Verteilung von Daten auf verschiedene Knoten
- **Logischer Entwurf**
  - Detailentwurf für ein „idealisierten“ DB-Modells
  - Beschränkung auf unterstützte Modellierungskonzepte
  - Verzicht auf systemspezifische Feinheiten
  - Teilschritte:
    - Transformation des konzeptionellen Schemas in Zieldatenbankmodell
    - Normalisierung
  - Ergebnis: logisches Schema

Phasenmodell

- **Datendefinition**
  - Umsetzung des logischen Schemas in konkretes Schema
  - Sichtdefinition
  - Verwendung der DDL und DML des Implementierungs-DBMS
  - Ergebnis: konkretes DB-Schema (konzeptuelles und externe Schemata)
- **Physischer Entwurf**
  - Definition der internen Ebene
  - Ergänzung der DB-Definition um Zugriffsunterstützung für Effizienzverbesserung (Definition von Indexen)
  - Ergebnis: physisches Schema
- **Implementierung und Wartung**

Abbildung des ER-Modells: Ziel

- Darstellung aller Informationen des ER-Diagramms im resultierenden DB-Schema
- Erhaltung der Informationskapazität
  - im DB-Schema sind genausoviele Instanzen darstellbar wie im ER-Diagramm
  - kapazitätserhöhend: mehr Relationen darstellbar als korrekte Beziehungen zwischen den Entities (Bsp.: 1:1-Beziehung)
  - kapazitätsvermindernd: weniger Relationen darstellbar (Bsp.: m:n-Beziehung)

Abbildung: Grundprinzipien

1. Entity-Typen und Beziehungstypen jeweils auf Relationenschemata
   - Attribute als Attribute des Relationenschemas
   - Schlüssel übernehmen
2. Kardinalitäten der Beziehungen durch Wahl der Schlüssel beim Relationenschemata ausdrücken
3. evtl. Relationenschemata von Entity- und Beziehungstypen verschmelzen

Abbildung: Konzepte

ER-Konzept	relationales Konzept
Entity-Typ E_i	Relationenschema R_i
Attribute von E_i	Attribute von R_i
Primärschlüssel P_i	Primärschlüssel P_i
Beziehungstyp	Relationenschema Attribute: P_1, P_2
dessen Attribute	weitere Attribute
1:n	P_2 wird Primärschlüssel
1:1	P_1 und P_2 werden Schlüssel
m:n	$P_1 \cup P_2$ wird Primärschlüssel

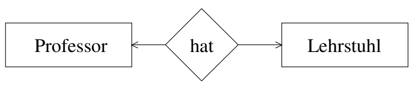
Entity-Typen

- Abbildung auf Relationenschema mit allen Attributen
- Auswahl eines (möglichst einfachen) Primärschlüssels
  - ein statt mehrere Attribute
  - **integer** statt **string**

Beziehungstypen

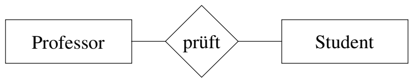
- Abbildung auf Relationenschema mit allen Attributen + Primärschlüssel der beteiligten Entity-Typen
  - m:n: beide Primärschlüssel werden zusammen Schlüssel
  - 1:n: Primärschlüssel der n-Seite wird Schlüssel
  - 1:1: beide Primärschlüssel werden je ein Schlüssel

Abbildung von 1:1-Beziehungen



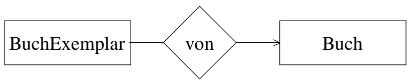
- Relationenschemata:
  Professoren (PANr, Stufe)
  Lehrstühle (Lehrstuhlbez, Anzahl_Stellen)
  Hat_Lehrstuhl (PANr, Lehrstuhlbez)
- Zusammenfassung zu einem Relationenschema,
  - wenn jeder Lehrstuhl immer durch einen Professor besetzt
  - jeder Professor immer einen Lehrstuhl hat

Abbildung von m:n-Beziehungen



- Relationenschemata
  Professoren (PANr, Stufe)
  Studenten (Matrikelnummer, Fach)
  Prüft (PANr, Matrikelnummer)

Abbildung von 1:n-Beziehungen



- Relationenschemata
  Buch_Exemplare (InvNr)
  Bücher(ISBN, Titel)
  Von(Invnr, ISBN)
- Zusammenfassung von Buch_Exemplar und Von,
  - da jedes Exemplar zu einem Buch gehören muß

Relationaler Datenbankentwurf

- Verfeinerung des logischen Entwurfs
- Ziel: Vermeidung von Redundanzen durch Aufspalten von Relationenschemata, ohne gleichzeitig
  - semantische Informationen zu verlieren (*Abhängigkeitstreue*)
  - die Möglichkeit zur Rekonstruktion der Relationen zu verlieren (*Verbundtreue*)
- Redundanzvermeidung durch Normalformen

Funktionale Abhängigkeiten

- *Funktionale Abhängigkeit* einer Relation zwischen Attributmengen X und Y, wenn in jedem Tupel der Relation der Attributwert unter den X-Komponenten den Attributwert unter den Y-Komponenten festlegt
- Schreibweise: $X \rightarrow Y$
- Beispiele (siehe folgende Folie)
  - ISBN $\rightarrow$ Titel, Verlag
  - ISBN $\rightarrow$ Autor, Stichwort (keine funktionale Abhängigkeit)
  - ISBN $\rightarrow$ ISBN (trivial)

Bücher-Relation mit Redundanzen

Bücher

ISBN	Titel	Autor	Version	Stichwort	Verlagsname
0-8053-1753-8	Princ. of DBS	Elmasri	1,1989	RDB	Benj./Cumm.
0-8053-1753-8	Princ. of DBS	Navathe	1,1989	RDB	Benj./Cumm.
0-8053-1753-8	Princ. of DBS	Elmasri	2,1994	RDB	Benj./Cumm.
0-8053-1753-8	Princ. of DBS	Navathe	2,1994	RDB	Benj./Cumm.
0-8053-1753-8	Princ. of DBS	Elmasri	1,1989	Lehrbuch	Benj./Cumm.
0-8053-1753-8	Princ. of DBS	Navathe	1,1989	Lehrbuch	Benj./Cumm.
0-8053-1753-8	Princ. of DBS	Elmasri	2,1994	Lehrbuch	Benj./Cumm.
0-8053-1753-8	Princ. of DBS	Navathe	2,1994	Lehrbuch	Benj./Cumm.
0-8053-1753-8	Princ. of DBS	Elmasri	1,1989	ER	Benj./Cumm.
0-8053-1753-8	Princ. of DBS	Navathe	1,1989	ER	Benj./Cumm.
0-8053-1753-8	Princ. of DBS	Elmasri	2,1994	ER	Benj./Cumm.
0-8053-1753-8	Princ. of DBS	Navathe	2,1994	ER	Benj./Cumm.

Schlüssel als Spezialfall

- für Beispiel auf folgende Folie
  PANr $\rightarrow$ Vorname, Nachname, PLZ, Ort, Straße,
  Hausnummer
- immer PANr $\rightarrow$ PANr und gesamtes Schema auf rechter Seite
- wenn linke Seite minimal: *Schlüssel*
- Schlüssel X liegt vor, wenn für Relationenschema R die funkt. Abhängigkeit $X \rightarrow R$ gilt und X minimal ist
- Ziel des Datenbankentwurfs: alle gegebenen funkt. Abhängigkeiten in „Schlüsselabhängigkeiten“ umformen, ohne dabei semantische Information zu verlieren

Beispiel: Schlüssel

Personen

PANr	Vorname	Nachname	PLZ	Ort	Straße	HNr
4711	Andreas	Heuer	18209	DBR	BHS	15
5588	Gunter	Saake	39106	MD	STS	55
6834	Michael	Korn	39104	MD	BS	41
7754	Andreas	Möller	18209	DBR	RS	31

Schema-Eigenschaften

- Relationenschemata, Schlüssel und Fremdschlüssel so wählen, daß
  1. alle Anwendungsdaten aus den Basisrelationen hergeleitet werden können,
  2. nur semantisch sinnvolle und konsistente Anwendungsdaten dargestellt werden können und
  3. die Anwendungsdaten möglichst nicht-redundant dargestellt werden.
- Forderung 3:
  - Redundanzen innerhalb einer Relation: Normalformen
  - globale Redundanzen: Minimalität

Update-Anomalien

- Redundanzen in Basisrelationen unerwünscht:
  - unnötiger Speicherplatz
  - Information redundant → Änderung muß diese Information in allen ihren Vorkommen verändern
- Beispiel: **insert**-Anomalie
  - in Bücher-Relation einfügen (funktionale Abhängigkeit verletzt)

Bücher

ISBN	Titel	Autor	Version	Stichwort	Verlagsname
0-8053-1753-8	Princ. of DBS	Elmasri	3,1996	RDB	Springer

Erste Normalform

- führt zunächst Redundanzen ein
- *Erste Normaform (1NF):* nur atomare Attribute in Relationenschemata
- Beispiel

Invnr	Titel	ISBN	Autoren
1201	Objekt-banken	3-111	Heuer, Scholl
4712	Daten-banken	3-891	Ullman

Invnr	Titel	ISBN	Autor
1201	Objekt-banken	3-111	Heuer
1201	Objekt-banken	3-111	Scholl
4712	Daten-banken	3-891	Ullman

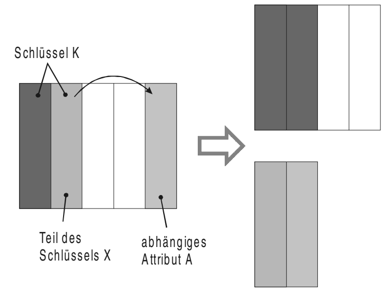
Zweite Normalform

- Zweite und weitere Normaformen: aufgrund der Struktur von Abhängigkeiten Redundanzen entdecken
- *Zweite Normalform (2NF):* keine partiellen Abhängigkeiten zwischen einem Schlüssel und weiteren Nicht-Primattributen (Attribute, die nicht in einem Schlüssel auftauchen)
- *Partielle Abhängigkeit* liegt vor, wenn ein Attribut funktional schon von einem Teil des Schlüssels abhängt

Zweite Normalform

- Beispiel:
  Invnr $\rightarrow$ Titel
  Invnr, Autor $\rightarrow$ Invnr, Titel, ISBN, Autor
  - **Invnr** und **Autor** zusammen Schlüssel
  - **Titel** hängt allein von **Invnr** ab
- Zweite Normaform durch Elimination der rechten Seite der partiellen Abhängigkeit und Kopie der linken Seite

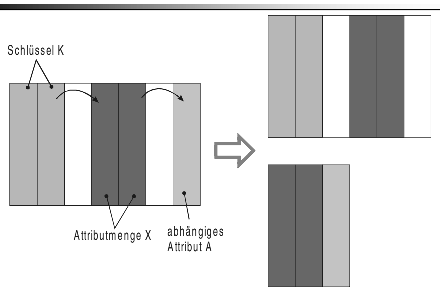
Zweite Normalform



Dritte Normalform

- *Dritte Normalform (3NF):* keine transitiven Abhängigkeiten zwischen einem Schlüssel und weiteren Nicht-Primattributen
- *Transitive Abhängigkeit:* Schlüssel K bestimmt Attributmenge X funktional, diese aber auch eine Attributmenge Y ($K \rightarrow X \rightarrow Y$)
- Beispiel: PANr → PLZ und PLZ → Ort
  Information, daß zur PLZ "18209" der Ort "DBR" gehört, ist redundant
- 3NF durch Elimination von Y und Kopie von X

Dritte Normalform



Transformationseigenschaften

- Erreichen von Normalformen durch Zerlegung von Relationenschemata
- Zu beachten:
  1. nur semantisch sinnvolle und konsistente Anwendungsdaten darstellen (*Abhängigkeitstreue*)
  2. alle Anwendungsdaten sollen aus Basisrelationen hergeleitet werden können (*Verbundtreue*)

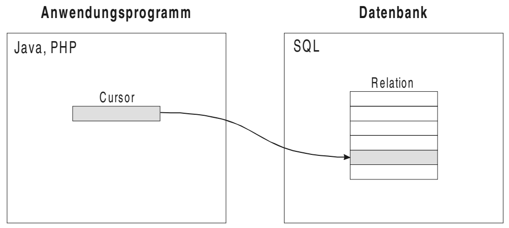
DB-Anwendungsentwicklung

- Kopplung von SQL und prozeduraler Programmiersprache
  - prozedurale Schnittstelle (Call-Level-Interface)
    - PHP + SQL-Funktionen
    - JDBC
  - Einbettung von SQL
    - Embedded SQL, SQLJ
  - Spracherweiterungen, neue Sprachentwicklungen
    - Persistente Programmiersprachen
    - PL/SQL

Cursor-Konzept

- Motivation
  - SQL: Konzept der Relation als Menge von Tupeln
  - Programmiersprache: Tupel/Objekt als Datenstruktur
- Verbindung Relation - Tupel
  - tupelweiser Zugriff auf Elemente einer Menge (Relation)
  - Navigation über Menge

Cursor-Konzept



SQL-Zugriff mit PHP

Funktionen
- DB-Verbindung öffnen und schließen
- DB auswählen
- Anfrage ausführen
- Zugriff auf Metadaten (Katalog, Schema des Anfrageergebnisses)

- Anfrageergebnis als Feld
- Beispiele
  - für MySQL (mysql_xxx)
  - fur Sybase (sybase_xxx)

SQL-Zugriff mit PHP

- **`int mysql_connect (string host, string user, string passwd)`**
  - Öffnen einer DB-Verbindung zum Server *host* für Benutzer *user* mit Passwort *passwd*
  - liefert Link-Identifier (>0)
- **`int mysql_select_db (string dbname, [int link_id])`**
  - Auswahl einer Datenbank *dbname*
- **`int mysql_close ([int link_id])`**
  - schließt DB-Verbindung

SQL-Zugriff mit PHP

- `int mysql_query (string query, [int link_id])`
  - Senden einer SQL-Anfrage *query* an das DBMS
  - liefert Result-Identifier (>0) bei erfolgreicher Ausführung
- `array mysql_fetch_row (int result)`
  - liefert ein Ergebnistupel (Zeile) als Feld
  - bei weiteren Aufrufen jeweils nächstes Tupel
  - liefert false, wenn keine weiteren Tupel vorhanden sind

SQL-Zugriff mit PHP - Beispiel

```
<?php
 mysql_connect ($host, $user, $passwd);
 mysql_select_db ("my_db");
 $result =
 mysql_query ("SELECT titel FROM buch");
 while ($row = mysql_fetch_row ($result)) {
     echo $row[0];
 }
 ?>
```

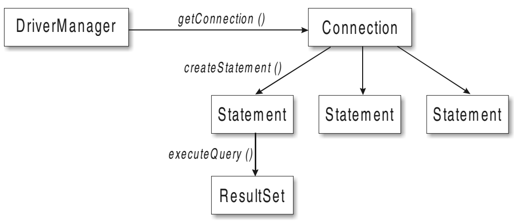

Datenbankzugriff mit Java™

- Einsatz von Java in Web-DB-Anwendungen
  - auf Client-Seite:
    - Applets (Browser-Erweiterungen)
  - auf Server-Seite:
    - Servlets (Erweiterungen des Web-Servers)
    - gespeicherte Prozeduren (DB-Erweiterungen)
- Anbindung Java - Datenbank (SQL)
  - JDBC (Call-Level-Schnittstelle)
  - SQLJ (Embedded SQL)

JDBC: Überblick

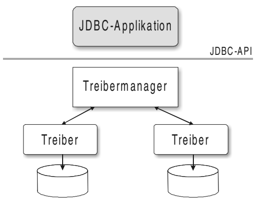
- Datenbankzugriffsschnittstelle für Java
- abstrakte, datenbankneutrale Schnittstelle
- vergleichbar mit ODBC
- Low-Level-API: direkte Nutzung von SQL
- Java-Package **java.sql**
- Klassen
  - **DriverManager**: Einstiegspunkt, Laden von Treibern
  - **Connection**: Datenbankverbindung
  - **Statement**: Ausführung von Anweisungen über eine Verbindung
  - **ResultSet**: verwaltet Ergebnisse einer Anfrage, Zugriff auf einzelne Spalten

JDBC-Struktur



Treiberkonzept

- Treibermanager
- abstrakte Klassen für Datenbankzugriff
- dynamisches Laden von Treibern
- Auswahl des geeigneten Treibers über Verbindungsdaten



Ablauf

1. Aufbau einer Verbindung zur Datenbank
   - Angabe der Verbindungsinformationen
   - Auswahl und Laden des Treibers
2. Senden einer SQL-Anweisung
   - Definition der Anweisung
   - Belegung von Parametern
3. Verarbeiten der Anfrageergebnisse
   - Navigation über Ergebnisrelation
   - Zugriff auf Spalten

Verbindungsaufbau

1. Treiber laden
```
Class.forName ("com.company.DBDriver");
```
2. Verbindung herstellen
```
Connection con;
String url = "jdbc:subprotocol:datasource";

con = DriverManager.getConnection (url,
        "scott", "tiger");
```
- JDBC-URL spezifiziert
  - Datenquelle/Datenbank
  - Verbindungsmechanismus (Protokoll, Server-Host und Port)

Anfrageausführung

1. Anweisungsobjekt (Statement) erzeugen
```
Statement stmt = con.createStatement ();
```
2. Anweisung ausführen
```
String query = "SELECT titel, preis FROM buch";
ResultSet rset = stmt.executeQuery (query);
```
- Klasse **`java.sql.Statement`**
  - Ausführung von Anfragen (SELECT) mit **`executeQuery`**
  - Ausführung von Änderungsanweisungen (DELETE, INSERT, UPDATE) mit **`executeUpdate`**

Anfrageformen

- Statement
  - Basisklasse für alle Anweisungen
  - Verarbeitung einfacher SQL-Anweisungen ohne Parameter
- PreparedStatement
  - Kapselung einer vorkompilierten Anweisung
  - parametrisierbar
  - Verwendung: wiederholte Ausführung einer Anweisung (mit verschiedenen Parametern)
- CallableStatement
  - Aufruf von gespeicherten Prozeduren mit Parametern

Ergebnisverarbeitung

1. Navigation über Ergebnismenge (Cursor-Prinzip)
```
while (rset.next ()) {
   // Verarbeitung der einzelnen Tupel
   ...
}
```
2. Zugriff auf Spaltenwerte über `getXXX`-Methoden
   - über Spaltenindex
     `String titel = rset.getString (1);`
   - über Spaltenname
     `String titel = rset.getString ("titel");`

Fehlerbehandlung

- Fehlerbehandlung mittels Exception-Mechanismus
- `SQLException` für alle SQL- und DBMS-Fehler

```
try {
   // Aufruf von JDBC-Methoden
   ...
} catch (SQLException exc) {
   System.out.println ("SQLException: " +
        exc.getMessage ());
}
```

Änderungsoperationen

- DDL- und DML-Operationen mittels **`executeUpdate`**
- liefert Anzahl der betroffenen Zeilen (für DML-Operationen)

```
Statement stmt = con.createStatement ();
int rows = stmt.executeUpdate (
   "UPDATE buch SET bestand = bestand-1" +
   " WHERE isbn = '12345' ");
```


Transaktionssteuerung

- Methoden von Connection
  - **`commit ()`**
  - **`rollback ()`**
- Auto-Commit-Modus
  - implizites Commit nach jeder Anweisung
  - Transaktion besteht nur aus einer Anweisung
  - Umschalten mittels **`setAutoCommit (boolean)`**

Transaktionssteuerung

```
try {
   // AutoCommit-Modus ausschalten
   con.setAutoCommit (false);

   // mehrere DML-Anweisungen
   ...

   // COMMIT ausführen
   con.commit ();
} catch (SQLException exc) {
   // Änderungen zurücknehmen
   con.rollback ();
}
```

Vorübersetzte Anweisungen

- `java.sql.PreparedStatement`
  - Beschleunigung der Ausführung von Anweisungen durch Vorübersetzung und
  - wiederholten Aufruf mit verschiedenen Parametersätzen
- Parameter in Anweisung durch "`?`" markiert
- Belegung der Parameter mittels `setXXX`-Methoden

Vorübersetzte Anweisungen

1. Anweisungsobjekt (PreparedStatement) erzeugen
```
String insStmt =
  "INSERT INTO buch VALUES (?, ?)";
PreparedStatement stmt =
  con.prepareStatement (insStmt);
```
2. Parameter setzen
```
stmt.setString (1, "123456");
stmt.setFloat (2, 56.99);
```
3. Anweisung ausführen
```
stmt.executeUpdate ();
```


Aufruf von gespeicherten Prozeduren

- `java.sql.CallableStatement`
  - Aufruf von gespeicherten Prozeduren (Stored Procedures)
  - Übergabe von IN-, INOUT- und OUT-Parametern
  - IN-Parameter
    - wie bei PreparedStatement
  - OUT-Parameter
    - Registrierung des SQL-Typs
    - Auslesen der Ergebniswerte mit `getXXX`-Methoden

Aufruf von gespeicherten Prozeduren

1. Anweisungsobjekt erzeugen
```
String callStmt = "{ call TestProc (?, ?) }";
CallableStatement stmt = con.prepareCall
(callStmt);
```
2. IN- und INOUT-Parameter setzen
```
stmt.setInt (1, 42);
```
3. Typen der INOUT- und OUT-Parameter registrieren
```
stmt.registerOutParameter (2,
java.sql.Types.FLOAT);
```
4. Anweisung ausführen
```
stmt.executeUpdate ();
```
5. Ergebnisse auslesen
```
double res = stmt.getDouble (2);
```


JDBC-Beispiel

```
try {
      // Verbindungsaufbau
      Connection con = DriverManager.getConnection (
                        url, user, passwd);
      // Anfrageausführung
      Statement stmt = con.createStatement ();
      ResultSet rs = stmt.executeQuery (
            "SELECT titel, preis FROM buch");
      while (rs.next ()) {
            System.out.println (rs.getString (1) +
                  ", " + rs.getDouble (2));
      }
} catch (SQLException exc) {
      // Fehlerbehandlung
      System.out.println (exc);
}
```

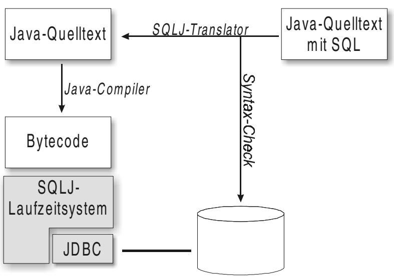

Zugriff auf Metadaten

- Metadaten: Informationen über Nutzdaten (Tabellen, Sichten, Spalten, ...)
- **`java.sql.ResultSetMetaData`**
  - Informationen über Struktur des Anfrageergebnisses
- **`java.sql.DatabaseMetaData`**
  - Informationen über Eigenschaften/Fähigkeiten des DBMS
  - Daten aus dem Schemakatalog

SQLJ: Überblick

- Einbettung von SQL-Anweisungen in Java-Quelltext
- Vorübersetzung des erweiterten Quelltextes in „echten“ Java-Code durch Translator **sqlj**
- Überprüfung der SQL-Anweisungen
  - korrekte Syntax
  - Übereinstimmung der Anweisungen mit DB-Schema
  - Typkompatibilität der für Datenaustausch genutzten Variablen
- Nutzung von JDBC-Treibern

SQLJ: Prinzip



SQLJ-Anweisungen

- Kennzeichnung durch `#sql`
- Deklarationen
  - Klassendefinitionen für Iteratoren
- SQL-Anweisungen
  - Anfragen, DML- und DDL-Anweisungen
    `#sql { SQL-Operation };`
  - Beispiel

```
#sql { INSERT INTO buch VALUES
  ("Datenbanken", 59.00, 123456, MITP) };
```

Host-Variablen

- Variablen einer Host-Sprache (hier Java), die in SQL-Anweisungen auftreten können
- Verwendung: Austausch von Daten zwischen Host-Sprache und SQL
- Kennzeichnung durch "`:variable`"
- Beispiel:

```
String titel, isbn = "123456";
#sql { SELECT titel INTO :titel FROM buch
        WHERE isbn = :isbn };
```


Iteratoren

- Implementierung des Cursor-Konzeptes
- Formen:
  - *benannte Iteratoren*
    - Zugriff auf Spalten des Ergebnisses über Methode mit dem Spaltennamen
  - *Positionsiteratoren*
    - Zugriff über Host-Variablen und FETCH-Anweisung
    - Zuordnung durch Position in der Anweisung

Iteratoren

1. Deklaration des Iterators

```
#sql public iterator BookIter (String titel,
                               double preis);
```

2. Definition des Iteratorobjektes

```
BookIter iter;
```

3. Ausführung der Anweisung

```
#sql iter = { SELECT titel, preis FROM buch };
```

4. Navigation

```
while (iter.next ()) {
  System.out.println (iter.titel () + " " +
                      iter.preis ());
}
```

SQLJ: Beispiel

```
#sql public iterator BookIter (String titel,double preis);
try {
       // Verbindungsaufbau
       Connection con = DriverManager.getConnection (
                               url, user, passwd);
       DefaultContext ctx = new DefaultContext (con);
       DefaultContext.setDefaultContext (ctx);

       // Anfrageausführung
       BookIter iter;
#sql iter = { SELECT titel, preis FROM buch");
       while (iter.next ())
              System.out.println (iter.titel () +
                     ", " + iter.preis ());
} catch (SQLException exc) {
       System.out.println (exc);
}
```


Vergleich

- JDBC:
  - Call-Level-Schnittstelle
  - dynamisches SQL
  - hohe Flexibilität
  - Fehlererkennung zur Laufzeit
- SQLJ:
  - Embedded-SQL-Lösung
  - statisches SQL
  - Fehlererkennung zur Compile-Zeit