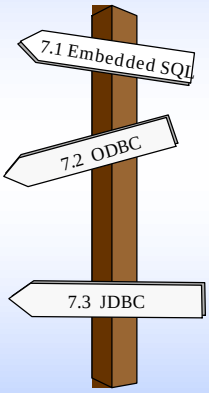
Inhaltsverzeichnis: Kapitel 7



7.1 Embedded SQL

- Ermöglicht Zugriff auf Datenbank aus einem Anwendungsprogramm
  - Erweiterung der Möglichkeiten von SQL durch Programmiersprache
  - Verbindung des Datenbanksystems zu anderen SW-Systemen
- Datenbankzugriff in SQL-Syntax
  - keine Spracherweiterung
  - Precompiler erforderlich
- Datentypanpassung zwischen SQL und Hostsprache
  - inkompatible Datentypen (impedance mismatch)
- Im ANSI SQL Standard enthalten
  - trotzdem herstellerspezifische Erweiterungen

7.1.1 Übersetzung

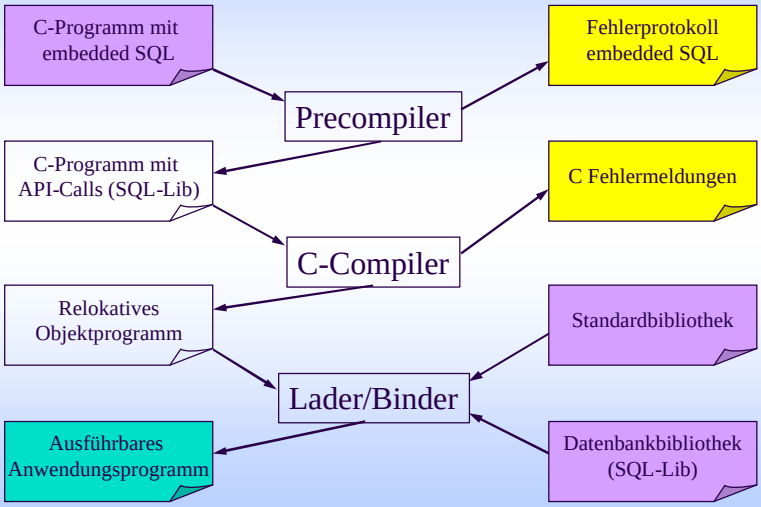


Bild 7 -3 Datenbank- und Informationssysteme © Sep-02 Prof. Dr. F. Laux

7.1.2 Minimalprogramm

- Declare Section

```
EXEC SQL BEGIN DECLARE SECTION;
        DCL* persno decimal(6);
        DCL ename char(20);
        DCL server char(20);
        DCL userid char(20);
        DCL passwd char(20);
EXEC SQL END DECLARE SECTION;
```

*Syntax ist abhängig von der Hostsprache (Cobol: {01|77}; PL/1: DCL; C, Fortran: entfällt)

- Verbindung herstellen

```
EXEC SQL CONNECT TO :server
        USER :userid [USING :passwd];*
```

*Syntax variiert je nach DB-System

- SQL-Statement erzeugen und ausführen

```
EXEC SQL SELECT Ename INTO :ename
        FROM Emp WHERE Persno = :persno;
```

- Datenbank schließen

```
DBCLOSE();
```

Bild 7 -4 Datenbank- und Informationssysteme © Sep-02 Prof. Dr. F. Laux

7.1.2 SQL Declare Section, SQLCA

Kommentare/Beispiele

- Begrenzer für Declare Section
  - EXEC SQL BEGIN DECLARE SECTION;
  - EXEC SQL END DECLARE SECTION;

  EXEC SQL und ; ist Markierung für Precompiler
- Definition von Host Variablen
  - <host_var> <SQL-Type>;
  - EXEC SQL INCLUDE SQLCA

  persno decimal(6);
  Kommunikationsbereich für Fehler-Meldungen
- Deklaration von Tabellen
  - EXEC SQL DECLARE <tab> TABLE (<attrDecl_list>);

  bei Oracle von Generator erzeugt
- Definition von Cursors
  - EXEC SQL DECLARE <curs> CURSOR FOR <SELECT-Clause>

  DECLARE <curs> CURSOR FOR
  select persno, ename from emp
  where deptid = :deptid;

7.1.2 Single Result Operation

Kommentare/Beispiele

- Insert/ Update/ Delete
  - EXEC SQL
    <SQL_insert/update/delete_stmt> ;

  EXEC SQL INSERT INTO emp
  VALUES (:persno, :ename);

  EXEC SQL UPDATE emp SET
  ename = :ename WHERE <cond>;

  EXEC SQL DELETE FROM emp
  WHERE persno =:persno;
- Single Row Query
  - EXEC SQL
    SELECT <attr_list> INTO <hostVar_list >
    FROM <tab> WHERE <1row_cond> ;

  EXEC SQL
  SELECT ename INTO :ename
  FROM emp WHERE persno =:persno;

7.1.2 Multirow Query, Cursor

Kommentare/Beispiele

- Cursor deklarieren
  - DECLARE <curs> CURSOR FOR <select_stmt> [FOR UPDATE OF <attrib_list> ;

  DECLARE nyEmp CURSOR FOR select persno, ename from emp where deptid = :deptid ;
- Cursor öffnen (Query ausführen)
  - OPEN <curs> ;

  OPEN nyEmp ;
- Ergebnisse in Schleife verarbeiten
  - FETCH <curs> INTO <hostVar_list>

  while (sqlcode == 0) {
  FETCH nyEmp into :persno, :ename;
  ... }
  /* sqlcode == +100 no data,
  > 0 Warning, < 0 Error */
- Cursor schließen
  - CLOSE <curs>

Bild 7 -7 Datenbank- und Informationssysteme © Sep-02 Prof. Dr. F. Laux

7.1.2 Fehlerbehandlung

- Explizite Fehlerabfrage durch
  - SQLCODE (= 0 ok, +100 keine Daten, < 0 Error, > 0 Warning)
  - SQLSTATE (= ccuuu, cc = 2-stelliger Klassenwert z.B. 00 = successful completion, 01 = warning, 02 = no data found, 07 = dyn. SQL-Err. 08 = connection error, 0A = feature not supported, ...)
    uuu = 3-stelliger Unterklassenwert z.B. bei cc = 01, kann uuu = 003 (NULLs in Aggregat-Fkt. Ignoriert), uuu = 006 (Privilege not revoked) sein
- Implizite Fehlerabfrage durch
  - WHENEVER {NOT FOUND | SQLWARNING | SQLERROR} {CONTINUE | <goto_stmt>}
  - Precompiler erzeugt Fehlerabfrage für jede SQL-Anweisung und verzweigt zur angegebenen Sprungmarke

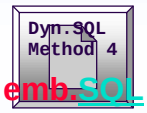
Bild 7 -8 Datenbank- und Informationssysteme © Sep-02 Prof. Dr. F. Laux

7.1.3 Dynamic SQL 92 (1)

- SQL-Anweisung ist erst zur Laufzeit bekannt
- Metainformation über SQL Description Area (SQLDA) erfragen
- Dynamic SQL Übersicht (Methode 4)

EXEX SQL ...

- INCLUDE SQLDA;
- Declare Hostvariable für <SQL_stmt>;
- DECLARE <stmt_name> STATEMENT;
- <SQL_stmt> = "SQL String";
- PREPARE <stmt_ prepName > FROM <SQL_stmt>;
- ALLOCATE DESCRIPTOR <descr> [WITH MAX <anzahl>];
- DESCRIBE {INPUT | OUTPUT} <stmt_prepName> USING SQL DESCRIPTOR <descr>;
  - Input gibt Infos über die Eingabeparameter
  - Output gibt Infos über das Ergebnis (z.B. Attributtypen)
- EXECUTE <stmt_prepName> [INTO <outHostVar_list>] [USING <inHostVar_list>];
- DECLARE <curs> [INSENSITIVE] [SCROLL] CURSOR FOR <stmt_prepName>;
- FETCH <curs> INTO <outHostVar_list>;



7.1.3 Dynamic SQL 92 (SQLDA)

- SQL Descriptor Area (SQLDA) Struktur

SQLDAID	Name des Deskriptors
SQLDALEN	Größe des Deskriptors
SQLN	Anzahl der Einträge in SQLVAR
SQLTYPE	Typ der SQL Anweisung
SQLVAR	Array[1 .. SQLN]

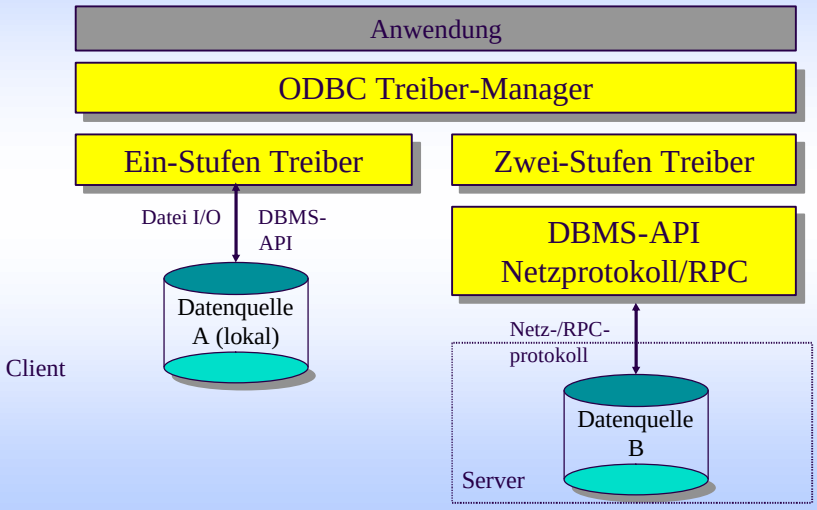
mit den Einträgen (Struktur) für jedes Attribut:

SQLNAME	Name Attributs
SQLTYPE	Datentyp Attributs
SQLLEN	Länge des Attributs
SQLDATA	Verweis auf Attributwert

7.2 ODBC

- Einheitliches API für den Datenzugriff auf verschiedene Datenquellen
  - relationale Datenbanken
  - nichtrelationale Datenbanken
  - Dateien
- Implementierung des Call Level Interface (CLI) der X/Open Gruppe
  - X/Open = Zusammenschluß von DBMS-Herstellern
  - Komponente von MS Data Access (MDAC)
  - Unterstützung durch ANSI und ISO
- Systemneutral
  - Einsatz unter verschiedenen Betriebssystemen
  - Unterstützung durch verschiedene Programmiersprachen
- unterstützt Client/Server-Architektur
  - Client = Anwendung und ODBC-Treiber
  - Server = lokale oder entfernte Datenquelle

7.2.1 Architektur



7.2.1 Minimalprogramm

◆ Variablen definieren	signed short rc; /* return code RETCODE */ void far * henv, hdbc, hstmt; /* handles */
◆ Handles allokieren	SQLAllocEnv(&henv); SQLAllocConnection(henv, &hdbc);
◆ Verbindung herstellen	SQLConnect(hdbc, "<dataSrc>", ...);
◆ Statement erstellen und ausführen	SQLAllocStmt(hdbc, &hstmt); SQLExecDirect(hstmt, "select * from ... ", ...);

◆ Ergebnis auslesen	rc = SQLFetch(hstmt);
	if (rc == SQL_SUCCESS)
	SQLGetData(hstmt, colno, SQL_C_CHAR,
	szData, sizeof(szData), &actLen);

◆ Ressourcen freigeben	SQLFreeStmt(hstmt, SQL_DROP); SQLDisconnect(hdbc); SQLFreeConnect(hdbc); SQLFreeEnv(henv);

7.2.2 SQLBindCol, SQLBindParmeter

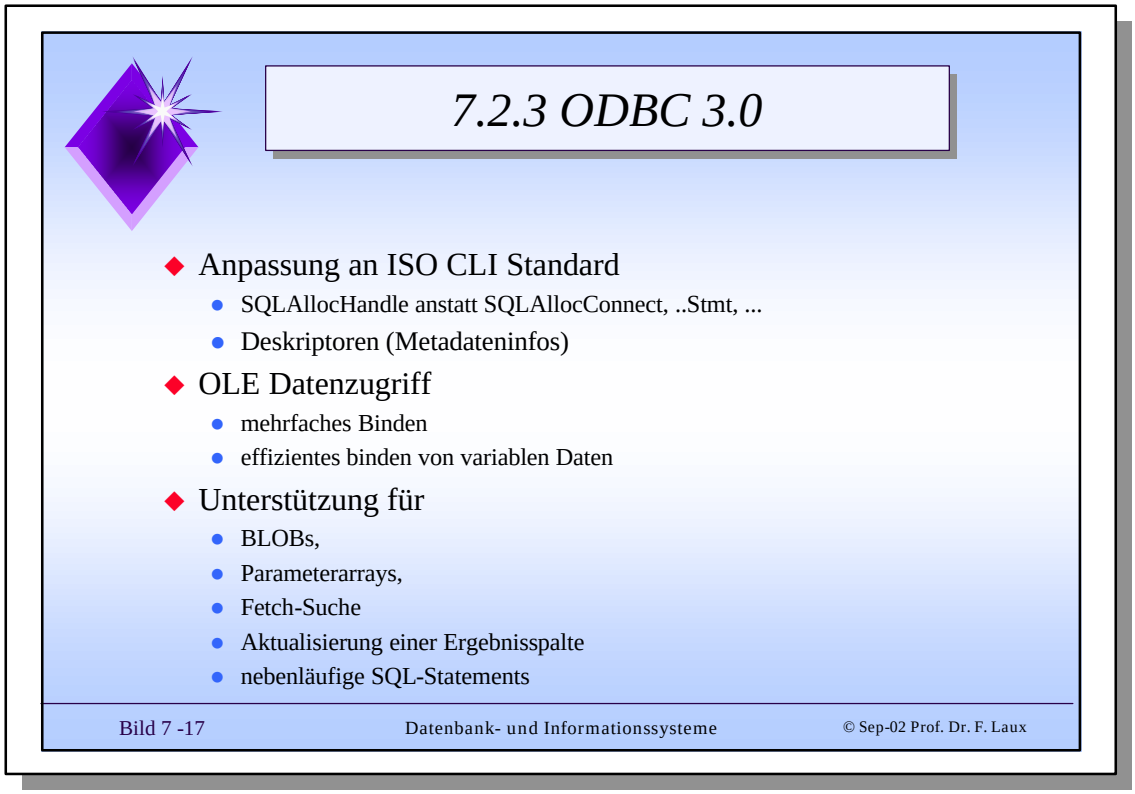
- Zuordnung von Datenbankwerten einer Programmvariablen
  - SQLBindCol(<hstmt>, <colno>, <C-type>, <var>, <var-len>, &<actlen>);
- Zuordnung von Parameterwerten einer Programmvariablen
  - SQLBindParameter(<hstmt>, <pno>, <I/O-ptype>, <C-type>, <SQL-type>, <coldef>, <scale>, <var>, <var-len>, &<actlen>);
- Beispiel
  - SQLBindCol(hstmt, 1, SQL_C_CHAR, f1, MAX_DATA, &cbData1);
  - SQLBindCol(hstmt, 2, SQL_C_LONG, &lf2, sizeof(lf2), &cbData2);
  - SQLExecDirect(hstmt, "select * from Tab1", SQL_NTS);
  - SQLFetch(hstmt); /* füllt Variablen f1 und lf2 */
  - SQLExecDirect(hstmt2, "create table Tab2 (f1 text(255), f2 long)",SQL_NTS);
  - SQLBindParameter(hstmt2, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 255, 0, f1, sizeof(f1), &cbData1);
  - SQLBindParameter(hstmt2, 2, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, &lf2, sizeof(lf2),&cbData2);
  - SQLExecDirect(hstmt2, "insert into Tab2(f1, f2) values (?,?)", SQL_NTS);

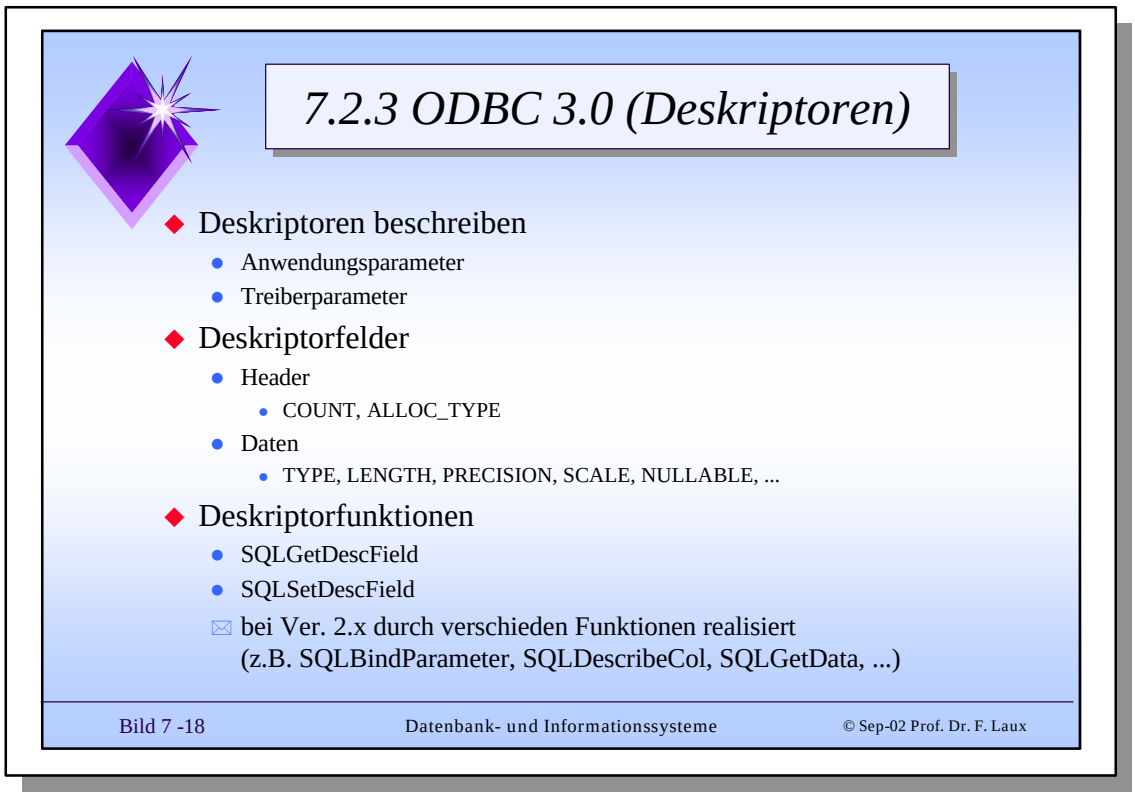
7.2.2 SQLPrepare, SQLExecute

- Optimierung der Ausführungszeiten
  - SQLExecDirect versus SQLPrepare und SQLExecute
    - **SQLExecDirect** parst SQL-String, erstellt Ausführungsplan und führt Befehl aus
    - **SQLPrepare** parst SQL-String und erstellt Ausführungsplan
    - **SQLExecute** führt vorbereiteten SQL-Befehl aus
- SQLPrepare(<hstmt>, <SQL-string>, <SQL-strType>);
- SQLExecute(<hstmt>);
- Beispiel
  - SQLPrepare(hstmt2, "insert into MyFiles(f1, f2, f3, f4) values (?, ?, ?, ?)", SQL_NTS);
  - rc = SQLExecDirect(hstmt, "select * from MyFiles", SQL_NTS);
  - for(rowcount = 0, rc = SQLFetch(hstmt); rc == SQL_SUCCESS; rc = SQLFetch(hstmt)) {
    SQLExecute(hstmt2);
    ...;
    };

7.2.2 ODBC Fehlercodes

#define SQL_INVALID_HANDLE	(-2)
#define SQL_ERROR	(-1)
#define SQL_SUCCESS	0
#define SQL_SUCCESS_WITH_INFO	1
#define SQL_NO_DATA_FOUND	100
#define SQL_STILL_EXECUTING	2
#define SQL_NEED_DATA	99


7.2.3 ODBC 3.0
Anpassung an ISO CLI Standard
SQLAllocHandle anstatt SQLAllocConnect, ..Stmt, ...
Deskriptoren (Metadateninfos)
OLE Datenzugriff
mehrfaches Binden
effizientes binden von variablen Daten
Unterstützung für
BLOBs,
Parameterarrays,
Fetch-Suche
Aktualisierung einer Ergebnisspalte
nebenläufige SQL-Statements
Bild 7 -17
Datenbank- und Informationssysteme
© Sep-02 Prof. Dr. F. Laux

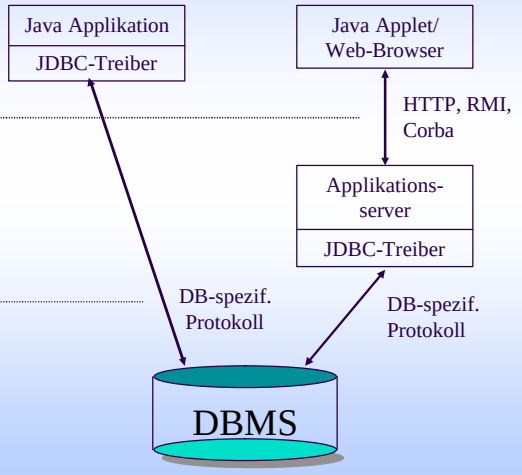

7.2.3 ODBC 3.0 (Deskriptoren)
Deskriptoren beschreiben
Anwendungsparameter
Treiberparameter
Deskriptorfelder
Header
COUNT, ALLOC_TYPE
Daten
TYPE, LENGTH, PRECISION, SCALE, NULLABLE, ...
Deskriptorfunktionen
SQLGetDescField
SQLSetDescField
bei Ver. 2.x durch verschieden Funktionen realisiert
(z.B. SQLBindParameter, SQLDescribeCol, SQLGetData, ...)
Bild 7 -18
Datenbank- und Informationssysteme
© Sep-02 Prof. Dr. F. Laux

7.3 JDBC

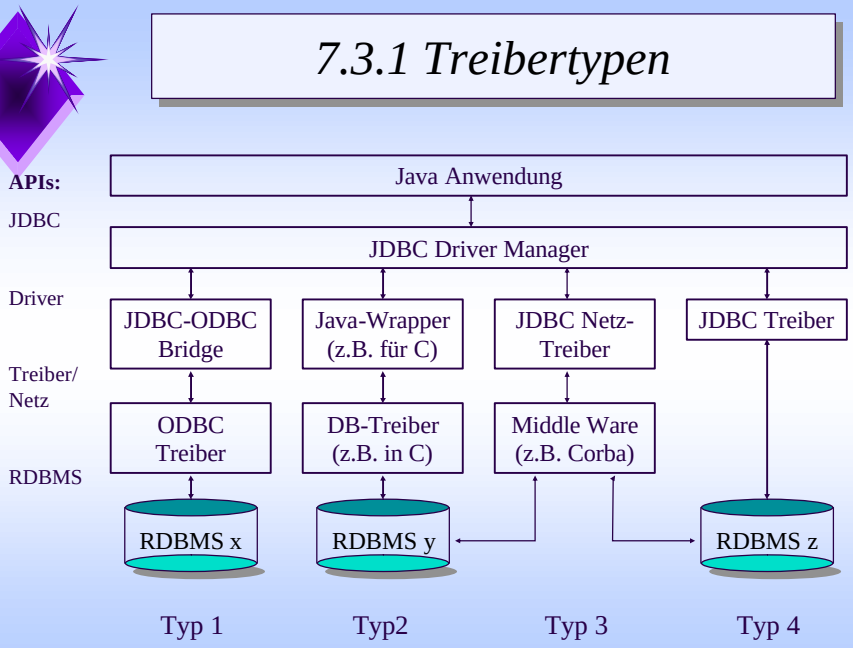
- Was ist JDBC ?
  - Java API zur Ausführung von SQL-Befehlen
  - Besteht aus Klassen und Schnittstellen, die in Java geschrieben sind
  - JDBC ermöglicht DB-Anwendungen, die ausschließlich in Java geschrieben sind
  - JDBC™ ist eine Schutzmarke, kein Akronym

7.3.1 JDBC Architektur

- Client
- Middle Ware
- DB-Server



7.3.1 Treibertypen



7.3.1 Minimalprogramm

- JDBC Treiber laden
  ```
  Class.forName ("path.to.jdbc.Driver");
  ```
- Verbindung aufbauen
  ```
  Connection conn;
  conn = DriverManager.getConnection(url,uid,pwd);
  ```
- SQL-Statement erzeugen
  ```
  Statement stmt = conn.createStatement();
  ```
- Statement ausführen
  ```
  ResultSet rs = stmt.executeQuery("select * from ...");
  ```
- Ergebnisse auslesen
  ```
  while (rs.next())
  {"for each attribute-index (i) "
    variable(i) = rs.getObject(i);
    .. "do something with variable(i) "... }
  ```
- Statement schließen
  ```
  stmt.close;
  ```
- Verbindung schließen
  ```
  conn.close;
  ```

7.3.2 Verbindung herstellen

- Treiber laden

  Kommentare/Beispiele

  - Class.forName("<pfad.fuer>.<JdbcDriver>");

    erzeugt ClassNotFoundException
    oracle.thin.JdbcDriver
    solid.JdbcDriver

- mit DB verbinden
  - URL aufbauen
    - String url = "jdbc:<subprotocol>:<subname>";

      jdbc:odbc:myDataSource
      jdbc:oracle:thin:@dblabor1/dbName
      jdbc:solid://dblabor1.fh-rt.de:1313
  - verbinden und anmelden
    - Connection conn =
      DriverManager.getConnection(
      url, "<uid>", "<pwd>");

      erzeugt SQLException

7.3.2 Tabellen anlegen

- Statement erzeugen

  Kommentare/Beispiele

  - Statement stmt = conn.createStatement();

    erzeugt SQLException
    wird für alle Datenbank-
    zugriffe benötigt

- Statement ausführen
  - mit explizitem String-Objekt
    - String sqlCreate = "create table <table>" +
      "(<col1> <datatype1> <kb1>, ...)";

      SQL-create-table Statement
      als String-Objekt
    - stmt.executeUpdate(sqlCreate);
  - mit implizitem String-Objekt
    - stmt.executeUpdate("create table <table>" +
      "(<col1> <datatype1> <kb1>, ...)");

7.3.2 Daten manipulieren

- Statement erzeugen (siehe Bild 7-24)
- Statement ausführen (siehe Bild 7-24 für die Variante mit explizitem String-Objekt)
  - Daten eingeben
    - stmt.executeUpdate("insert into <table>" +
      "values (<val1>, <val2>, ...)");
  - Daten ändern
    - stmt.executeUpdate("update <table>" +
      "set <col1> = <val1>, ...");
  - Daten löschen
    - stmt.executeUpdate("delete from <table>" +
      "where <col1> = <val1>, ...");

7.3.2 Tabellen abfragen

- Statement erzeugen (siehe Bild 7-24) — Kommentar/Beispiel
- Query ausführen
  - ResultSet rs = stmt.executeQuery("select <col_list>" +
    "from <table> where ...");
- „ResultSet" auslesen
  - über getObject — idx = rs.findColumn("Adresse");
    - Object obj = rs.getObject(<index>|"<AttrName>") ; — Adresse adr = rs.getObject(idx);
    - Objekttyp umwandeln (falls geeignet)
      var = new <Type>(obj.toString()).<type>Value(); — fl = new Float(dm.toString()).floatValue();
  - über getInt, getFloat, getString, getDate, getTime, getAsciiStream, getBinaryStream, ... — BLOB stückweise auslesen
    - var = rs.getXXX(<index>|"<AttrName>"); — fl = rs.getFloat("betrag");
    - if (var = 0 && rs.wasNull) — NULL-Marke erkennen ! Bei echten Objekten, kann auf var = NULL abgefragt werden.

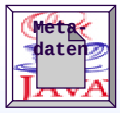
7.3.3 Metadaten („ResultSet“ Infos)

- Verbindung herstellen, Query ausführen (siehe Bild 7-24, 7-26)
- ResultSet für Metadaten erzeugen
  - ResultSetMetaData rsmd = rs.getMetaData();
- Spaltenanzahl ermitteln
  - int numCols = rsmd.getColumnCount();
- Spaltenüberschrift ermitteln
  - for (int i = 1; i <= numCols; i++)
    { String colName = rsmd.getColumnLabel(i); ...}
- Attributtyp ermitteln
  - int colType = rsmd.getColumnType(i); oder
    String colTName = rsmd.getColumnTypeName(i);
  - JDBC- und Java Types siehe Bild 7-36

7.3.3 Metadaten (Datenbank Infos)

- Verbindung herstellen (siehe Bild 7-24)
- Metadaten Objekt erzeugen
  - DatabaseMetaData dbmd = conn.getMetaData();
- Metadaten auslesen
  - DB Produktname und Versionsnummer
    - String dbName = dbmd.getDatabaseProductName();
    - String dbVersion = dbmd.getDatabaseProductVersion();
  - Meta-Meta-Daten
    - int maxLength = dbmd.getMaxTableNameLength();
  - Schema/Kataloginfos
    - ResultSet dbSchema = dbmd.getTables("<cat>", "<schema>", "<tab>", <types>[]);

    liefert eine Tabelle mit den Attributen *Table_Cat, Table_Schema, Table_Name, Table_Type, Remarks*, die mit den ResultSet-Methoden auszulesen sind. <cat>, <schema> und <tab> sind Matchcodes für Katalog-, Schema und Tabellennamen. <types> ist ein Array mit Tabellentypen.



7.3.3 SQL-Exception

- SQLException können bei fast allen JDBC-Klassen auftreten
- muss abgefangen werden, da Operation nicht ausgeführt wurde
- ist eine von java.lang.Exception abgeleitete Klasse

- stellt Informationen über Datenzugriffsfehler im ***catch***-Block bereit
  - catch (SQLException ex) {
    while (ex != null) {
    System.out.println("Java Exception Message: " + ex.getMessage());
    System.out.println("SQL State: " + ex.getSQLState());
    System.out.println("vendorspecific code: " + ex.getErrorCode());
    ex = ex.getNextException();
    }
    conn.rollback;
    }
  - SQL State ist ein Fehlercode, der in X/Open SQL Specification definiert ist

Bild 7 -29 Datenbank- und Informationssysteme © Sep-02 Prof. Dr. F. Laux

7.3.3 SQL-Warning

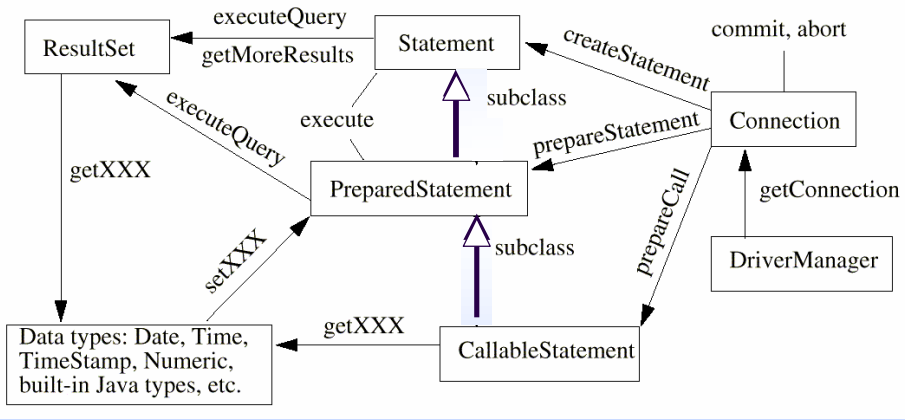
- SQLWarning können bei Connection, Statement und ResultSet auftreten
- unterbricht das Programm nicht
- muss mit getWarnings bzw. getNextWarning abgefragt werden

- stellt Informationen in „besonderen“ Situationen bereit

  - SQLWarning warn = conn.getWarnings();
    while (warn != null) {
    System.out.println("Java Warning Message: " + warn.getMessage());
    System.out.println("SQL State: " + warn.getSQLState());
    System.out.println("Vendorspecific code: " + warn.getErrorCode());
    warn = warn.getNextException();
    }
  - SQLWarnings: DataTruncation, Privilege not revoked, Error during disconnect

Bild 7 -30 Datenbank- und Informationssysteme © Sep-02 Prof. Dr. F. Laux

7.3.4 JDBC 2.0 API Übersicht



7.3.4 Connection, DriverManager

- DriverManager
  - getConnection(String <url>[, String <usr>, String <pwd>]);
  - [de]registerDriver(Driver <dvr>)
- Connection
  - createStatement(); prepareStatement(String <sql>); prepareCall(String <sql>);
  - getMetaData();
  - commit(); rollback();
  - {set|get}TransactionIsolation(int <level>);
    - <level> ∈ {TRANSACTION_SERIALIZABLE, ...REPEATABLE_READ, ...READ_COMMITTED, ..READ_UNCOMMITTED}
  - setReadOnly(boolean <t/f>); boolean isReadOnly();

7.3.4 Statement und Subklassen

- Statement
  - ResultSet executeQuery(String <sql>); int executeUpdate(String <sql>);
  - boolean execute(String <sql_script>);
    - erzeugt mehrere ResultSets und Update counts. Abgreifen mit:
    - ResultSet **getResultSet();** boolean **getMoreResults();** int **getUpdateCount();**
  - SQLWarning getWarnings();
  - {set|get}QueryTimeout(int <sec>); cancel();
- PreparedStatement
  - setXXX(int <parmIdx>, <JavaTypeXXX> <val>);
  - setNull(int <paramIdx>, int <sqlTypeNo>); clearParameters();
- CallableStatement
  - registerOutParameter(int <paramIdx>, int <sqlType>[, int <scale>]);
  - <JavaTypeXXX> getXXX(int <parmIdx>);
  - boolean wasNull();

Bild 7 -33 Datenbank- und Informationssysteme © Sep-02 Prof. Dr. F. Laux

7.3.4 ResultSet

- ResultSet
  - boolean next(); close();
  - Nach Spaltennummer
    - <JavaTypeXXX> getXXX(int <colIdx>); Object getObject(int <colIdx>);
  - Nach Spaltennamen
    - <JavaTypeXXX> getXXX(String <colName>);
      Object getObject(String <colName>);
  - Als Teilergebnis
    - InputStream get{ASCII|Unicode|Binary}Stream(String <colName>);
  - boolean wasNull();
  - SQLWarning getWarnings(); clearWarnings();
  - ResultSetMetaData getMetaData();
  - int findColumn(String <colName>);

Bild 7 -34 Datenbank- und Informationssysteme © Sep-02 Prof. Dr. F. Laux

7.3.4 Data Types

- Java Datatypes
  - boolean, byte, double, float, int, long, short, String
- Date, Time
  - Date d = new Date(<yy>, <mm>, <dd>); Time t = new Time(<hh>, <mm>, <ss>);
    - yy := Jahr -1900, mm := 0 (Jan.) ... 11 (Dez.)
  - valueOf(String "<hh:mm:ss>"); valueOf(String "<yyyy-mm-dd>");
  - String toString();
- TimeStamp
  - Timestamp ts = new Timestamp(<yy>, <mm>, <dd>, <hh>, <mm>, <ss>, <ns>);
    - ns := Nanosekunden (1/1000000000sec)
  - valueOf(String "<yyyy-mm-dd hh:mm:ss.ns>"); String toString();
  - boolean {after|before|equals}(Timestamp ts);
  - setNanos(int n); getNanos();

7.3.4 Java vs. JDBC Typen

Java Typ	JDBC Typ
String	CHAR, VARCHAR, LONGVARCHAR
java.math.BigDecimal	NUMERIC
boolean	BIT
byte, short, int, long	TINYINT, SMALLINT, INTEGER, BIGINT
float, double	DEAL, DOUBLE
java.sql.Date, ...Time, ...Timestamp	DATE, TIME, TIMESTAMP

7.3.4 Neue Funktionen in Version 2.0

- geordnetes (scrollable) ResultSet
  - beforeFirst(), next(); relative(int n); afterLast();
- Stapelverarbeitung (batch updates) im Statement-Objekt
  - addBatch(String <sql>);
  - executeBatch();
- Persistente Objekte
  - ResultSet rs = stmt.executeQuery("select Mitarbeiter from Personal");
  - Mitarbeiter ma = (Mitarbeiter) rs.getObject(1);
- Neue Datentypen
  - BLOB, CLOB, Bsp.: Ref ref = rs.getRef(<colIdx));
  - create type Point(x FLOAT, y FLOAT); create type Währung as NUMERIC(10,2);
- Namens- und Verzeichnisdienste (JNDI, Java Naming and Directory Interface)

Bild 7 -37 Datenbank- und Informationssysteme © Sep-02 Prof. Dr. F. Laux