19.07.2002

DBS-Projekt: Ein internetgestütztes Musikarchiv

von Anja Jentzsch und Richard Cyganiak

Anwendungsprogrammierung

Unser Projekt ist eine internetgestützte Musikdatenbank. Die Java-Applikation könnte man sich als Eingabeinterface für die Betreiber der Datenbank vorstellen, während die Web-Anbindung die Sicht der Internet-User auf die Datenbank darstellt.

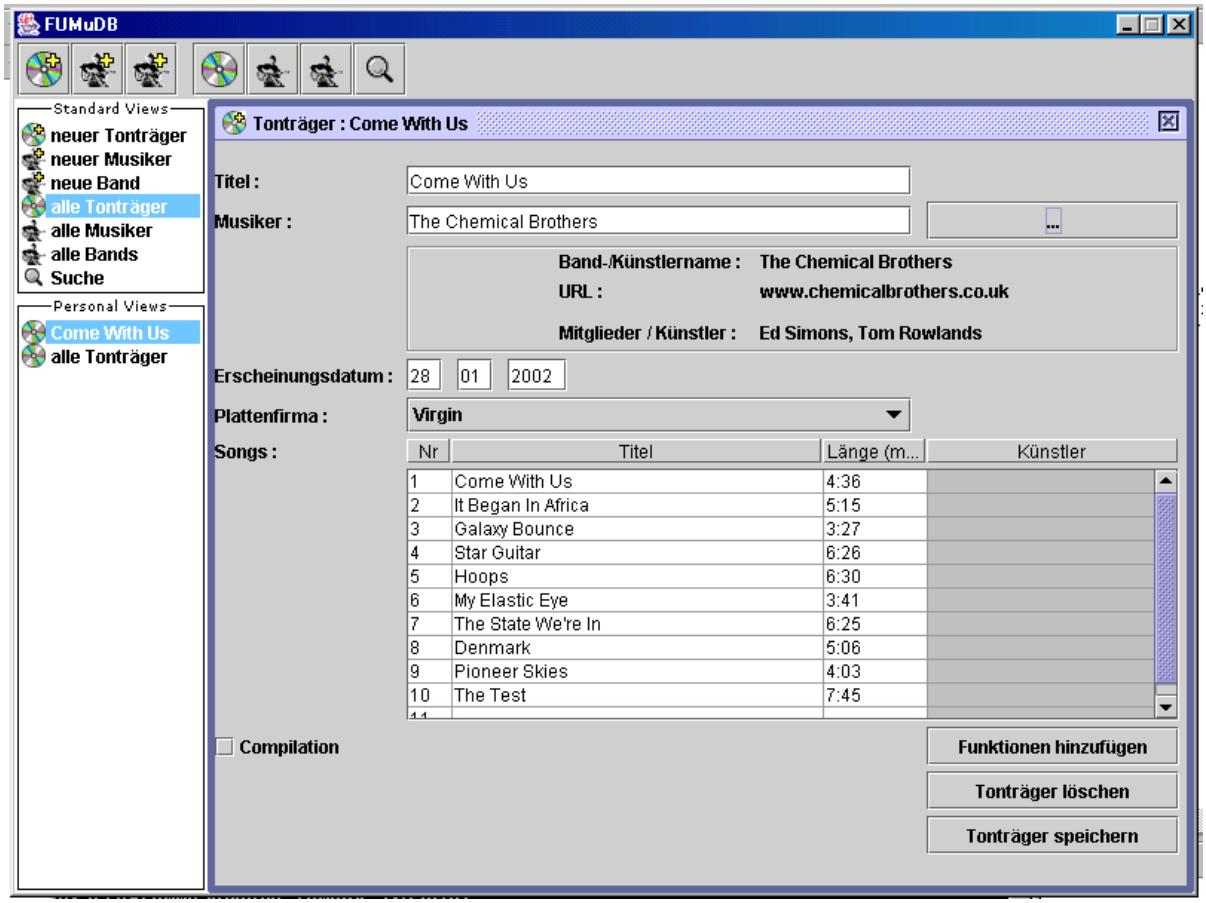
Für ein webgestütztes Projekt ist die Webanbindung natürlich der wichtigere Teil. Interessanter (und nach unseren Erfahrungen anspruchsvoller) ist aber die Frage, wie man die Daten am besten in die Datenbank bekommt. Wir haben daher auf unsere Applikation mehr Zeit verwendet als auf die Webanbindung.

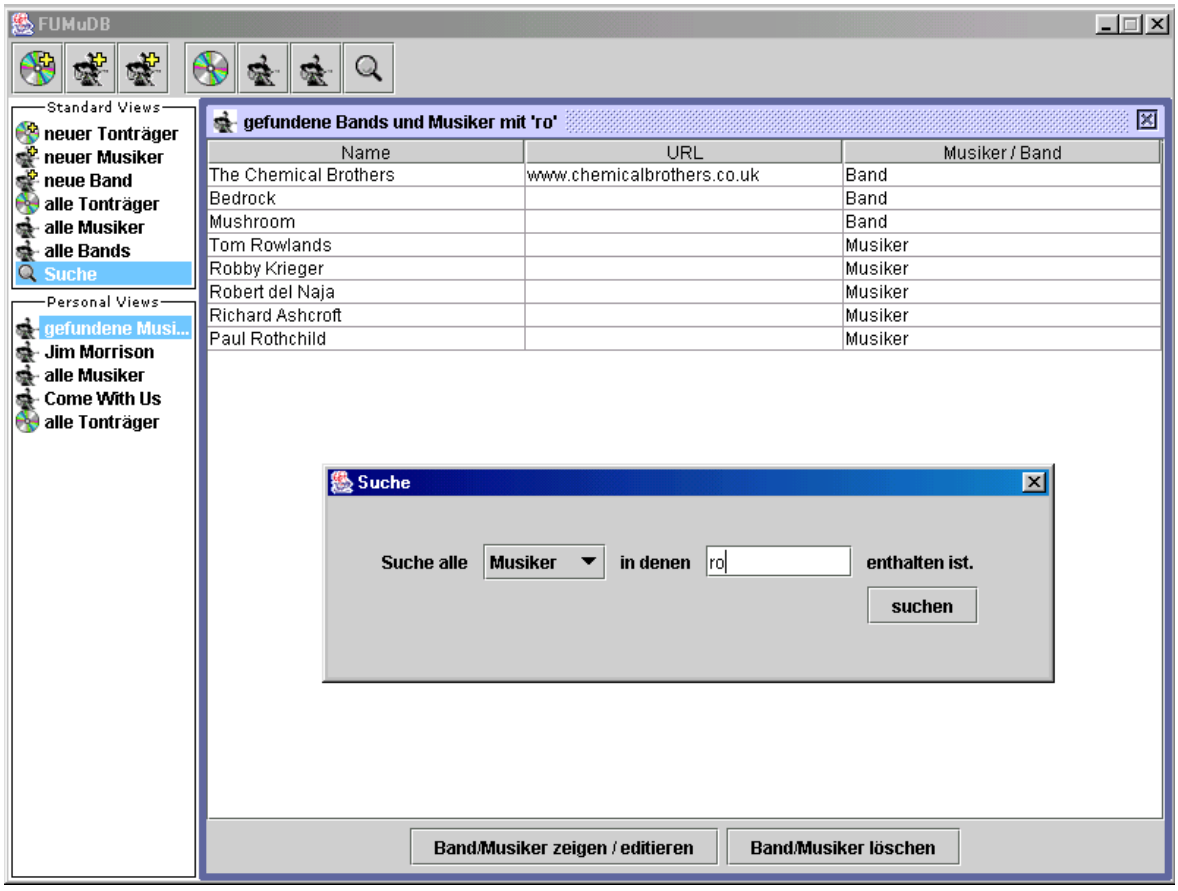
Implementierte Features der Applikation:

- Anzeigen aller Tonträger, Musiker und Bands
- Detailansichten für Tonträger, Musiker und Bands mit Editiermöglichkeit
- Formulare für die Eingabe neuer Tonträger, Musiker und Bands
- Suchfunktion für die genannten Entitäten

Eine Reihe anderer Features sind ansatzweise vorhanden. So können beispielsweise Compilations angezeigt, aber nicht editiert werden. Die Zugehörigkeit von Musikern zu Bands etc. wird ebenfalls angezeigt, aber Änderungen können nicht gespeichert werden.

Screenshots:




FUMuDB
Standard Views
neuer Tonträger
neuer Musiker
neue Band
alle Tonträger
alle Musiker
alle Bands
Suche
Personal Views
gefundene Musi...
Jim Morrison
alle Musiker
Come With Us
alle Tonträger
gefundene Bands und Musiker mit 'ro'
Name
URL
Musiker / Band
The Chemical Brothers
www.chemicalbrothers.co.uk
Band
Bedrock
Band
Mushroom
Band
Tom Rowlands
Musiker
Robby Krieger
Musiker
Robert del Naja
Musiker
Richard Ashcroft
Musiker
Paul Rothchild
Musiker
Suche
Suche alle
Musiker
in denen
ro
enthalten ist.
suchen
Band/Musiker zeigen / editieren
Band/Musiker löschen

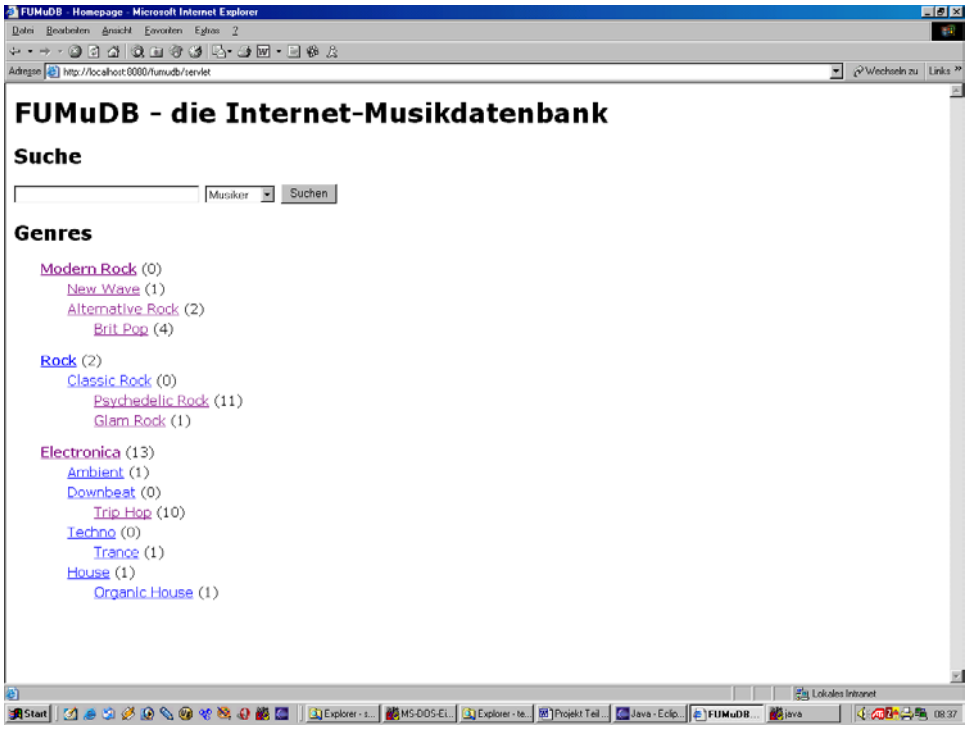
Web-Anbindung der Datenbank

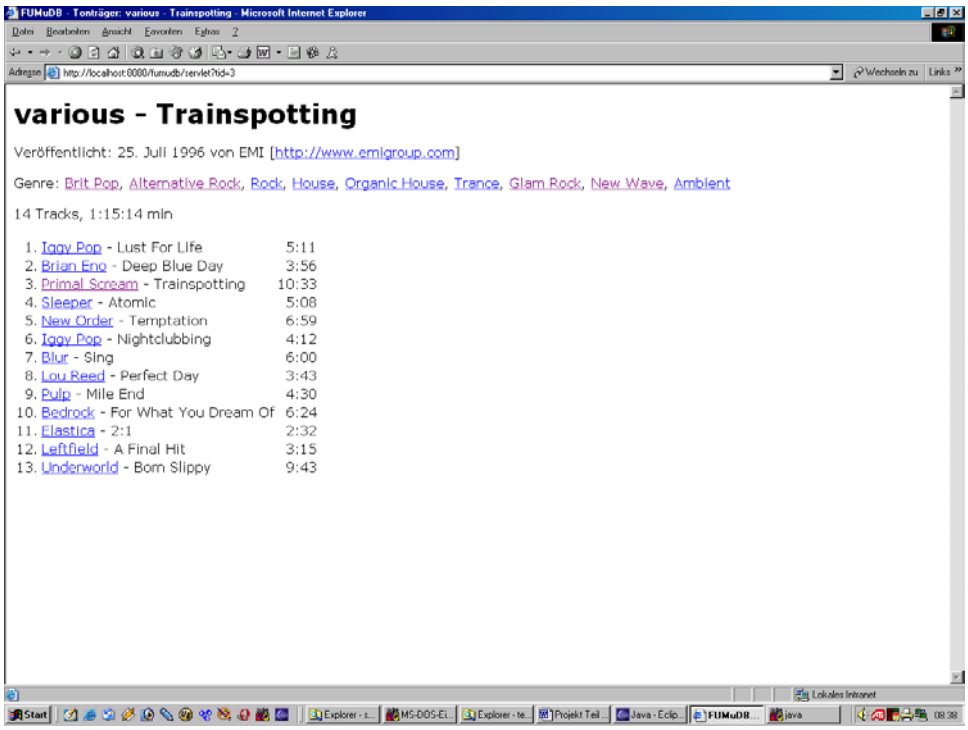
Implementierte Features der Webanbindung:

- Gliederung des Datenbestandes in Genres
- Detailansichten für Tonträger, Musiker und Bands
- Suchfunktion für die Relationen Musiker und Tonträger

Die Webanbindung wurde mit Hilfe von Servlets realisiert.

Screenshots:





Architektur

Wir haben unseren Code in mehreren Schichten aufgebaut, damit wir möglichst große Teile für beide Teilprojekte verwenden konnten:

fumdb.sql: Kommunikation mit der Datenbank
fumdb.domain: Klassen zur Repräsentierung der Entitäten (Tonträger, Musiker, Song…)
fumdb.gui und fumdb.servlet: Java-Swing-Applikation und Java-Servlet

Dieser Ansatz hat sich als recht praktisch erwiesen. Besonders die domain-Klassen konnten fast ohne Änderungen für beide Teilprojekte übernommen werden. Die Datenbankanbindung hingegen musste doch für das Webinterface noch deutlich erweitert werden, da die Webanwendung andere Anfragen benötigt.

Quellcode

Es folgt der Quellcode unserer Klasse fumdb.sql.Persistence. Es ist die einzige Klasse der Package fumdb.sql und ist für die gesamte Kommunikation mit dem Datenbankserver zuständig. Die Klasse wird sowohl von unserer Java-Applikation, als auch vom Servlet verwendet.

Persistence.java:

```
package fumdb.sql;

import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.HashMap;
import java.util.Map;
import java.util.Vector;

import fumdb.domain.Band;
import fumdb.domain.Function;
import fumdb.domain.Genres;
import fumdb.domain.Member;
import fumdb.domain.Musician;
import fumdb.domain.Person;
import fumdb.domain.Record;
import fumdb.domain.RecordCompany;
import fumdb.domain.Song;
import fumdb.domain.SongId;
import fumdb.domain.Tracklist;

/**
 * @author Richard Cyganiak
 */
public class Persistence {

     private static final String DRIVER
               = "oracle.jdbc.driver.OracleDriver";

     private static final String URL
               = "jdbc:oracle:thin:@kuh.inf.fu-berlin.de:1521:INTRODBS";

     private static final String USERNAME = "tz21";

     private static final String PASSWORD = "tz21";

     private static final int NO_ID = 0;

     private Connection connection;
```

```
private PreparedStatement selectRecordStatement;
private PreparedStatement selectMusicianStatement;
private PreparedStatement selectTracklistStatement;

private SQLErrorListener errorListener;

private Genres genres;
private Vector recordCompanies;
private Vector functions;

public Persistence(SQLErrorListener errorListener) {
     this.errorListener = errorListener;
}

public void connect() throws ClassNotFoundException, SQLException {
     if (connection == null) {
          Class.forName(DRIVER);
          connection = DriverManager.getConnection(URL, USERNAME, PASSWORD);
     }
     selectRecordStatement = connection.prepareStatement(getSelectRecordByIdSql());
     selectMusicianStatement = connection.prepareStatement(getSelectMusicianByIdSql());
     selectTracklistStatement = connection.prepareStatement(getSelectTracklistSql());

     recordCompanies = readRecordCompanies();
     genres = readGenres();
     functions = readFunctions();
}

public void disconnect() throws SQLException {
     connection.close();
     connection = null;
}

public void raiseError(String message) {
     errorListener.catchSQLException(new SQLException(message));
}

public int getRecordCount() {
     try {
          Statement statement = connection.createStatement();
          ResultSet resultSet = statement.executeQuery(getCountRecordsSql());
          resultSet.next();
          return resultSet.getInt(1);
     } catch (SQLException e) {
          errorListener.catchSQLException(e);
          return 0;
     }
}

public Record getRecord(int id) {
     if (id == NO_ID) {
          return null;
     }
     try {
          selectRecordStatement.setInt(1, id);
          ResultSet resultSet = selectRecordStatement.executeQuery();
          resultSet.next();
          return readRecordFromResultSet(resultSet);
     } catch (SQLException e) {
          errorListener.catchSQLException(e);
          return null;
     }
}

private Record readRecordFromResultSet(ResultSet rs) throws SQLException {
     Record result = new Record(this, rs.getInt("TID"));
     result.setTitle(rs.getString("Titel"));
     result.setMusician(rs.getInt("MID"), rs.getString("musiker_name"));
     result.setPublishedDate(rs.getDate("Erscheinungsdatum"));
     result.setRecordCompany(rs.getString("Plattenfirma"));
     return result;
}

public Musician getMusician(int id) {
     if (id == NO_ID) {
          return null;
     }
     try {
          selectMusicianStatement.setInt(1, id);
```

```
            ResultSet resultSet = selectMusicianStatement.executeQuery();
            resultSet.next();
            return readMusicianFromResultSet(resultSet);
        } catch (SQLException e) {
            errorListener.catchSQLException(e);
            return null;
        }
    }

    public Vector getAllMusicians() {
        return getMusicianCollection(getSelectAllMusiciansSql());
    }

    public Vector getSimilarMusicians(String name) {
        if (name == null) {
            return new Vector();
        }
        return getMusicianCollection(getSelectMusiciansByNameSql(name));
    }

    public Vector getSimilarRecords(String name) {
        if (name == null) {
            return new Vector();
        }
        return getRecordCollection(getSelectRecordsByNameSql(name));
    }

    public Musician getMusicianByName(String name) {
        return (Musician) getSimilarMusicians(name).firstElement();
    }

    private Vector getMusicianCollection(String sql) {
        try {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(sql);
            Vector result = new Vector();
            while (resultSet.next()) {
                result.add(readMusicianFromResultSet(resultSet));
            }
            return result;
        } catch (SQLException e) {
            errorListener.catchSQLException(e);
            return null;
        }
    }

    public Vector getBandMembers(int bandID) {
        return getMemberCollection(getSelectBandMembersSql(bandID));
    }

    public Vector getBandMemberships(int personID) {
        return getMemberCollection(getSelectBandMembershipsSql(personID));
    }

    private Vector getMemberCollection(String sql) {
        try {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(sql);
            Vector result = new Vector();
            while (resultSet.next()) {
                result.add(readMembershipFromResultSet(resultSet));
            }
            return result;
        } catch (SQLException e) {
            errorListener.catchSQLException(e);
            return null;
        }
    }

    private Member readMembershipFromResultSet(ResultSet rs) throws SQLException {
        Member member = new Member(this, rs.getInt("MID1"), rs.getInt("MID2"));
        member.setInstrument(rs.getString("Instrument"));
        member.setStartDate(rs.getDate("von"));
        member.setEndDate(rs.getDate("bis"));
        return member;
    }

    private Musician readMusicianFromResultSet(ResultSet rs) throws SQLException {
        boolean isBand = (rs.getString("Name") != null);
        if (isBand) {
```

```
            Band band = new Band(this, rs.getInt("MID"));
            band.setName(rs.getString("Name"));
            band.setURL(rs.getString("URL"));
            return band;
        } else {
            Person person = new Person(this, rs.getInt("MID"));
            person.setFirstName(rs.getString("Vorname"));
            person.setLastName(rs.getString("Nachname"));
            person.setURL(rs.getString("URL"));
            person.setMale("M".equals(rs.getString("Geschlecht")));
            person.setBirthday(rs.getDate("Geburtstag"));
            return person;
        }
    }

    public Vector getRecordCompanies() {
        return recordCompanies;
    }

    public Vector getRecordsByCompany(String company) {
        return getRecordCollection(getSelectRecordsByCompanySql(company));
    }

    public Vector getRecordsByMusician(int musicianID) {
        return getRecordCollection(getSelectRecordsByMusicianSql(musicianID));
    }

    public Vector getAllRecords() {
        return getRecordCollection(getSelectAllRecordsSql());
    }

    private Vector getRecordCollection(String sql) {
        try {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(sql);
            Vector result = new Vector();
            while (resultSet.next()) {
                result.add(readRecordFromResultSet(resultSet));
            }
            return result;
        } catch (SQLException e) {
            errorListener.catchSQLException(e);
            return null;
        }
    }

    private Vector readRecordCompanies() throws SQLException {
        Statement statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery(getSelectAllRecordCompaniesSql());
        Vector result = new Vector();
        while (resultSet.next()) {
            String name = resultSet.getString("firmen_name");
            String url = resultSet.getString("url");
            result.add(new RecordCompany(this, name, url));
        }
        return result;
    }

    private Vector readFunctions() throws SQLException {
        Statement statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery(getSelectAllFunctionsSql());
        Vector result = new Vector();
        while (resultSet.next()) {
            result.add(resultSet.getString("Funktion"));
        }
        return result;
    }

    public Genres getGenres() {
        return genres;
    }

    private Genres readGenres() throws SQLException {
        Statement statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery(getSelectAllGenresSql());
        Map genresAndParents = new HashMap();
        while (resultSet.next()) {
            String genre = resultSet.getString("genre_name");
            String parent = resultSet.getString("obergenre_name");
            genresAndParents.put(genre, parent);
```

```
        }
        return new Genres(this, genresAndParents);
    }

    public Map getSongCountsByGenre() {
        try {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(getSelectGenreNamesAndCountsSql());
            Map songCounts = new HashMap();
            while (resultSet.next()) {
                String genre = resultSet.getString("genre_name");
                int songs = resultSet.getInt("songs");
                songCounts.put(genre, new Integer(songs));
            }
            return songCounts;
        } catch (SQLException e) {
            errorListener.catchSQLException(e);
            return null;
        }
    }

    public Song getSong(SongId songId) {
        if (songId == null) {
            return null;
        }
        try {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(getSelectSongSql(songId));
            resultSet.next();
            return readSongFromResultSet(resultSet);
        } catch (SQLException e) {
            errorListener.catchSQLException(e);
            return null;
        }
    }

    public Vector getSongsByMusician(int musicianId) {
        try {
            Statement statement = connection.createStatement();
            ResultSet resultSet =
                    statement.executeQuery(getSelectSongsByMusicianSql(musicianId));
            Vector result = new Vector();
            while (resultSet.next()) {
                result.add(readSongFromResultSet(resultSet));
            }
            return result;
        } catch (SQLException e) {
            errorListener.catchSQLException(e);
            return null;
        }
    }

    private Song readSongFromResultSet(ResultSet rs) throws SQLException {
        SongId songId = new SongId(rs.getInt("mid"), rs.getInt("sid"));
        Song song = new Song(this, songId);
        song.setName(rs.getString("name"));
        song.setGenreName(rs.getString("genre_name"));
        SongId original = new SongId(rs.getInt("original_mid"), rs.getInt("original_sid"));
        if (!rs.wasNull()) {
            song.setOriginal(original);
        }
        return song;
    }

    public Map getAllMusicianNames() {
        try {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(getSelectAllMusicianNamesSql());
            HashMap result = new HashMap();
            while (resultSet.next()) {
                result.put(resultSet.getString("musiker_name"), new
                        Integer(resultSet.getInt("mid")));
            }
            return result;
        } catch (SQLException e) {
            errorListener.catchSQLException(e);
            return null;
        }
    }
```

```
public Map getAllBandNames() {
     try {
          Statement statement = connection.createStatement();
          ResultSet resultSet = statement.executeQuery(getSelectAllBandNamesSql());
          HashMap result = new HashMap();
          while (resultSet.next()) {
               result.put(resultSet.getString("Name"), new
                         Integer(resultSet.getInt("mid")));
          }
          return result;
     } catch (SQLException e) {
          errorListener.catchSQLException(e);
          return null;
     }
}

public Map getAllPersonNames() {
     try {
          Statement statement = connection.createStatement();
          ResultSet resultSet = statement.executeQuery(getSelectAllPersonNamesSql());
          HashMap result = new HashMap();
          while (resultSet.next()) {
               result.put(resultSet.getString("Name"), new
                         Integer(resultSet.getInt("mid")));
          }
          return result;
     } catch (SQLException e) {
          errorListener.catchSQLException(e);
          return null;
     }
}

public Tracklist getTracklist(int recordId) {
     try {
          selectTracklistStatement.setInt(1, recordId);
          ResultSet resultSet = selectTracklistStatement.executeQuery();
          Tracklist result = new Tracklist(this, recordId);
          while (resultSet.next()) {
               Song song = readSongFromResultSet(resultSet);
               int trackId = resultSet.getInt("tracknr");
               int length = resultSet.getInt("laenge");
               result.addSong(trackId, song, length);
          }
          return result;
     } catch (SQLException e) {
          errorListener.catchSQLException(e);
          return null;
     }
}

public Vector getAllFunctionsByRecord(int recordId) {
     try {
          Statement statement = connection.createStatement();
          ResultSet resultSet = statement.executeQuery(getSelectFunctionSql(recordId));
          Vector result = new Vector();
          while(resultSet.next()) {
               Song song = readSongFromResultSet(resultSet);
               Musician musician = readMusicianFromResultSet(resultSet);
               String function = resultSet.getString("funktion");
               result.add(new Function(this, song, musician, function));
          }
          return result;
     } catch (SQLException e) {
          errorListener.catchSQLException(e);
          return null;
     }
}

public int createRecord(Record record) {
     return create(getCreateSql(record), "Tontraeger", "TID");
}

public boolean updateRecord(Record record) {
     return update(getUpdateSql(record));
}

public boolean deleteRecord(int recordId) {
     return delete("DELETE FROM Tontraeger WHERE tid=" + recordId);
}
```

```
public boolean deleteSong(SongId songId) {
     return delete("DELETE FROM Song WHERE mid=" + songId.getMusicianId() + " AND sid="
               + songId.getSongId());
}

public boolean updateSong(Song song) {
     return update(getUpdateSql(song));
}

public int createSong(Song song) {
     try {
          Statement statement = connection.createStatement();
          statement.executeUpdate(getCreateSql(song,
                    getSongAutoIncrementValue(song.getMusicianId())+1));
          return getSongAutoIncrementValue(song.getMusicianId());
     } catch (SQLException e) {
          errorListener.catchSQLException(e);
          return 0;
     }
}

public int createPerson(Person person) {
     try {
          Statement statement = connection.createStatement();
          statement.executeUpdate(getCreateMusicianSql(person.getURL()));
          int id = getAutoIncrementValue("Musiker", "mid");
          statement.executeUpdate(getCreateSql(person, id));
          return id;
     } catch (SQLException e) {
          errorListener.catchSQLException(e);
          return 0;
     }
}

public boolean updatePerson(Person person) {
     try {
          Statement statement = connection.createStatement();
          statement.executeUpdate(getUpdateMusicianSql(person));
          statement.executeUpdate(getUpdateSql(person));
          return true;
     } catch (SQLException e) {
          errorListener.catchSQLException(e);
          return false;
     }
}

public int createBand(Band band) {
     try {
          Statement statement = connection.createStatement();
          statement.executeUpdate(getCreateMusicianSql(band.getURL()));
          int id = getAutoIncrementValue("Musiker", "mid");
          statement.executeUpdate(getCreateSql(band, id));
          return id;
     } catch (SQLException e) {
          errorListener.catchSQLException(e);
          return 0;
     }
}

public boolean updateBand(Band band) {
     try {
          Statement statement = connection.createStatement();
          statement.executeUpdate(getUpdateMusicianSql(band));
          statement.executeUpdate(getUpdateSql(band));
          return true;
     } catch (SQLException e) {
          errorListener.catchSQLException(e);
          return false;
     }
}

public boolean deleteMusician(int musicianId) {
     return delete("DELETE FROM Musiker WHERE mid=" + musicianId);
}

private int create(String sql, String table, String keyColumn) {
     try {
          Statement statement = connection.createStatement();
          statement.executeUpdate(sql);
          return getAutoIncrementValue(table, keyColumn);
```

```
        } catch (SQLException e) {
            errorListener.catchSQLException(e);
            return 0;
        }
    }

    private boolean update(String sql) {
        try {
            Statement statement = connection.createStatement();
            statement.executeUpdate(sql);
            return true;
        } catch (SQLException e) {
            errorListener.catchSQLException(e);
            return false;
        }
    }

    private boolean delete(String sql) {
        try {
            Statement statement = connection.createStatement();
            statement.executeUpdate(sql);
            return true;
        } catch (SQLException e) {
            errorListener.catchSQLException(e);
            return false;
        }
    }

    private int getAutoIncrementValue(String table, String column) throws SQLException {
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery(sqlAutoIncrementValue(table, column));
        rs.next();
        return rs.getInt(1);
    }

    private int getSongAutoIncrementValue(int mid) throws SQLException {
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery(sqlSongAutoIncrementValue(mid));
        rs.next();
        return rs.getInt(1);
    }

    private String sqlId(int id) {
        if (id == 0) return "NULL";
        return Integer.toString(id);
    }

    private String sqlString(String s) {
        if (s == null) return "NULL";
        return "'" + s + "'";
    }

    private String sqlDate(Date d) {
        if (d == null) return "NULL";
        return "TO_DATE('" + d.toString() + "', 'YYYY-MM-DD')";
    }

    private String sqlAutoIncrement(String table, String column) {
        return "(" + sqlAutoIncrementValue(table, column) + ") + 1";
    }

    private String sqlAutoIncrementValue(String table, String column) {
        return "SELECT MAX(" + column + ") FROM " + table;
    }

    private String sqlSongAutoIncrement(int mid) {
        return "(" + sqlSongAutoIncrementValue(mid) + ") + 1";
    }

    private String sqlSongAutoIncrementValue(int mid) {
        return "SELECT MAX(sid) FROM Song WHERE mid=" + mid;
    }

    private String getSelectAllRecordsSql() {
        return "SELECT * FROM Tontraeger NATURAL LEFT OUTER JOIN MusikerName";
    }

    private String getSelectRecordByIdSql() {
        return getSelectAllRecordsSql() + " WHERE TID=?";
    }
```

```
private String getSelectRecordsByCompanySql(String companyName) {
     return getSelectAllRecordsSql() + " WHERE Plattenfirma='" + companyName + "';";
}

private String getSelectRecordsByMusicianSql(int musicianID) {
     return getSelectAllRecordsSql() + " WHERE MID=" + musicianID;
}

private String getSelectRecordsByNameSql(String name) {
     return getSelectAllRecordsSql()
               + " WHERE LOWER(Titel) LIKE LOWER('%" + name + "%')";
}

private String getCountRecordsSql() {
     return "SELECT COUNT(*) FROM Tontraeger";
}

private String getSelectAllRecordCompaniesSql() {
     return "SELECT * FROM Plattenfirma";
}

private String getSelectAllFunctionsSql() {
     return "SELECT * FROM Funktion";
}

private String getSelectAllMusiciansSql() {
     return "SELECT * FROM Musiker "
               + "NATURAL LEFT OUTER JOIN Band_Kuenstlername "
               + "NATURAL LEFT OUTER JOIN Person";
}

private String getSelectMusicianByIdSql() {
     return getSelectAllMusiciansSql() + " WHERE MID=?";
}

private String getSelectMusiciansByNameSql(String name) {
     return getSelectAllMusiciansSql()
               + " WHERE LOWER(Band_Kuenstlername.name) LIKE LOWER('%" + name + "%') "
               + "OR LOWER(CONCAT(Person.vorname, CONCAT(' ', Person.nachname))) "
               + "LIKE LOWER('%" + name + "%')";
}

private String getSelectBandMembersSql(int bandID) {
     return "SELECT * FROM IstMitglied WHERE mid2=" + bandID;
}

private String getSelectBandMembershipsSql(int personID) {
     return "SELECT * FROM IstMitglied WHERE mid1=" + personID;
}

private String getSelectAllGenresSql() {
     return "SELECT * FROM Genre ORDER BY genre_name";
}

private String getSelectGenreNamesAndCountsSql() {
     return "SELECT genre_name, COUNT(*) AS songs FROM Genre NATURAL JOIN Song "
               + "GROUP BY genre_name";
}

private String getSelectSongSql(SongId songId) {
     return "SELECT * FROM Song "
               + "WHERE mid=" + songId.getMusicianId()
               + " AND sid=" + songId.getSongId();
}

private String getSelectSongsByMusicianSql(int musicianId) {
     return "SELECT * FROM Song "
               + "WHERE mid=" + musicianId;
}

private String getSelectAllMusicianNamesSql() {
     return "SELECT mid, musiker_name FROM MusikerName";
}

private String getSelectAllBandNamesSql() {
     return "SELECT MID, Name FROM Band_Kuenstlername";
}

private String getSelectAllPersonNamesSql() {
```

```
    return "SELECT MID, CONCAT(Vorname, CONCAT(' ', Nachname)) AS Name FROM Person";
}

private String getSelectTracklistSql() {
    return "SELECT * FROM Enthaelt NATURAL JOIN Song WHERE TID=? ORDER BY tracknr";
}

private String getSelectFunctionSql(int recordId) {
    return "SELECT * FROM HatFunktion "
             + "NATURAL JOIN Musiker "
             + "NATURAL LEFT OUTER JOIN Band_Kuenstlername "
             + "NATURAL LEFT OUTER JOIN Person, "
             + "Song, "
             + "Enthaelt "
             + "WHERE Song.mid = HatFunktion.song_mid AND Song.sid = HatFunktion.sid "
             + "AND Enthaelt.mid = Song.mid AND Enthaelt.sid = Song.sid "
             + "AND Enthaelt.tid = " + recordId;
}

private String getCreateSql(Record record) {
    return "INSERT INTO Tontraeger (TID, titel, mid, erscheinungsdatum, plattenfirma) "
             + "VALUES ("
             + sqlAutoIncrement("Tontraeger", "TID") + ", "
             + sqlString(record.getTitle()) + ", "
             + sqlId(record.getMusicianID()) + ", "
             + sqlDate(record.getPublishedDate()) + ", "
             + sqlString(record.getRecordCompanyName()) + ")";
}

private String getUpdateSql(Record record) {
    return "UPDATE Tontraeger "
             + "SET titel=" + sqlString(record.getTitle()) + ", "
             + "mid=" + sqlId(record.getMusicianID()) + ", "
             + "erscheinungsdatum=" + sqlDate(record.getPublishedDate()) + ", "
             + "plattenfirma=" + sqlString(record.getRecordCompanyName()) + " "
             + "WHERE tid = " + record.getId();
}

private String getCreateSql(Song song, int sid) {
    return "INSERT INTO Song (mid, sid, name, genre_name, original_mid, original_sid) "
             + "VALUES ("
             + song.getMusicianId() + ", "
             + sid + ", "
             + sqlString(song.getName()) + ", "
             + sqlString(song.getGenreName()) + ", "
             + ((song.getOriginalId() == null) ? "NULL" :
                     sqlId(song.getOriginalId().getMusicianId())) + ", "
             + ((song.getOriginalId() == null) ? "NULL" :
                     sqlId(song.getOriginalId().getSongId())) + ")";
}

private String getUpdateSql(Song song) {
    return "UPDATE Song "
             + "SET name=" + sqlString(song.getName()) + ", "
             + "genre_name=" + sqlString(song.getGenreName()) + " "
             + "original_mid=" + sqlId(song.getOriginalId().getMusicianId()) + ", "
             + "original_sid=" + sqlId(song.getOriginalId().getSongId()) + " "
             + "WHERE mid = " + song.getMusicianId() + " AND sid = " + song.getSongId();
}

private String getCreateMusicianSql(String url) {
    return "INSERT INTO Musiker (MID, URL) VALUES ("
             + sqlAutoIncrement("Musiker", "MID") + ", "
             + sqlString(url) + ")";
}

private String getCreateSql(Person person, int personId) {
    return "INSERT INTO Person (MID, Vorname, Nachname, Geburtstag, Geschlecht) VALUES ("
             + personId + ", "
             + sqlString(person.getFirstName()) + ", "
             + sqlString(person.getLastName()) + ", "
             + sqlDate(person.getBirthday()) + ", "
             + (person.isMale() ? "'M'" : "'W'") + ")";
}

private String getCreateSql(Band band, int bandId) {
    return "INSERT INTO Band_Kuenstlername (MID, Name) VALUES ("
             + bandId + ", "
             + sqlString(band.getName()) + ")";
}
```

```
    private String getUpdateMusicianSql(Musician musician) {
        return "UPDATE Musiker SET "
                + "url=" + sqlString(musician.getURL()) + " "
                + "WHERE mid = " + musician.getId();
    }

    private String getUpdateSql(Person person) {
        return "UPDATE Person SET "
                + "vorname=" + sqlString(person.getFirstName()) + ", "
                + "nachname=" + sqlString(person.getLastName()) + ", "
                + "geburtstag=" + sqlDate(person.getBirthday()) + ", "
                + "geschlecht=" + (person.isMale() ? "'M'" : "'W'") + " "
                + "WHERE mid=" + person.getId();
    }

    private String getUpdateSql(Band band) {
        return "UPDATE Band_Kuenstlername SET "
                + "name=" + sqlString(band.getName()) + " "
                + "WHERE mid = " + band.getId();
    }

    public Vector getMusiciansByGenre(String genre) {
        return getMusicianCollection(getSelectMusiciansByGenreSql(genre));
    }

    private String getSelectMusiciansByGenreSql(String genre) {
        return "SELECT * FROM "
                + "(SELECT mid, COUNT(*) AS SongsInGenre FROM Song "
                + " WHERE genre_name=" + sqlString(genre) + " GROUP BY mid) "
                + "NATURAL JOIN (Musiker NATURAL LEFT OUTER JOIN Band_Kuenstlername "
                + "NATURAL LEFT OUTER JOIN Person) "
                + "ORDER BY SongsInGenre";
    }

    private String getSelectRecordsByGenreSql(String genre) {
        return "SELECT * FROM Tontraeger "
                + "NATURAL JOIN ("
                + "SELECT tid, ROUND(songsInGenre/songs*100) AS genreQuota FROM "
                + "(SELECT tid, COUNT(*) AS songs FROM Enthaelt GROUP BY tid) "
                + "NATURAL JOIN "
                + "(SELECT tid, COUNT(*) AS songsInGenre FROM Enthaelt NATURAL JOIN Song "
                + "WHERE genre_name=" + sqlString(genre) + " GROUP BY tid) "
                + "WHERE songsInGenre/songs > 0.25) "
                + "NATURAL LEFT OUTER JOIN MusikerName "
                + "ORDER BY genreQuota DESC, musiker_name ASC";
    }

    public Map getRecordsByGenre(String genre) {
        try {
            Statement statement = connection.createStatement();
            ResultSet rs = statement.executeQuery(getSelectRecordsByGenreSql(genre));
            Map result = new HashMap();
            while (rs.next()) {
                Record record = readRecordFromResultSet(rs);
                Integer quota = new Integer(rs.getInt("genreQuota"));
                result.put(record, quota);
            }
            return result;
        } catch (SQLException e) {
            errorListener.catchSQLException(e);
            return null;
        }
    }
}
```

Indexierung

Wir haben alle unsere Anfragen aus Teil 5 mit SET TIMING ON noch einmal ausgeführt. Ergebnisse:

```
Elapsed: 00:00:00.01
Elapsed: 00:00:00.05
Elapsed: 00:00:00.02
Elapsed: 00:00:00.06
Elapsed: 00:00:08.62
Elapsed: 00:00:00.53
Elapsed: 00:00:00.09
Elapsed: 00:00:00.08
```

Offenbar ist bei den getesteten Datenmengen nur die fünfte Anfrage interessant für Optimierungen. Wir konzentrieren unsere Bemühungen daher auf diese Anfrage.

```
SQL> CREATE INDEX bewertet_tid_idx ON Bewertet(tid);

Index created.

Elapsed: 00:00:00.19
SQL> SELECT k.name, b.bewertung
  2  FROM
  3     Band_Kuenstlername k,
  4     (SELECT AVG(b.note) AS bewertung, f.mid
  5     FROM Bewertet b, Enthaelt e, HatFunktion f
  6     WHERE b.tid = e.tid
  7             AND e.mid = f.song_mid AND e.sid = f.sid
  8             AND f.funktion = 'Produzent'
  9     GROUP BY f.mid) b
 10  WHERE k.mid = b.mid AND b.bewertung IN
 11     (SELECT MAX(AVG(b.note))
 12     FROM Bewertet b, Enthaelt e, HatFunktion f
 13     WHERE b.tid = e.tid
 14             AND e.mid = f.song_mid AND e.sid = f.sid
 15             AND f.funktion = 'Produzent'
 16     GROUP BY f.mid);

NAME                                               BEWERTUNG
-------------------------------------------------- ----------
bruce springsteen                                          10
kmc                                                        10
shidapu                                                    10
vectrolab                                                  10
ukw                                                        10

Elapsed: 00:00:01.03
```

Der erste Index auf der Bewertet-Relation reduziert die Ausführungszeit schon deutlich von 8.62 Sekunden auf 1.03 Sekunden. Wir fügen zwei weitere zur Enthaelt-Relation hinzu:

```
SQL> CREATE INDEX enthaelt_sid_idx ON Enthaelt(sid);

Index created.

Elapsed: 00:00:00.13
SQL> SELECT k.name, b.bewertung
  2  FROM
  3     Band_Kuenstlername k,
  4     (SELECT AVG(b.note) AS bewertung, f.mid
  5     FROM Bewertet b, Enthaelt e, HatFunktion f
  6     WHERE b.tid = e.tid
  7             AND e.mid = f.song_mid AND e.sid = f.sid
  8             AND f.funktion = 'Produzent'
  9     GROUP BY f.mid) b
 10  WHERE k.mid = b.mid AND b.bewertung IN
 11     (SELECT MAX(AVG(b.note))
 12     FROM Bewertet b, Enthaelt e, HatFunktion f
 13     WHERE b.tid = e.tid
 14             AND e.mid = f.song_mid AND e.sid = f.sid
 15             AND f.funktion = 'Produzent'
 16     GROUP BY f.mid);

NAME                                               BEWERTUNG
```

```
-------------------------------------------------- ----------
bruce springsteen                                          10
kmc                                                        10
shidapu                                                    10
vectrolab                                                  10
ukw                                                        10

Elapsed: 00:00:00.15
SQL> CREATE INDEX enthaelt_mid_idx ON Enthaelt(mid);

Index created.

Elapsed: 00:00:00.14
SQL> SELECT k.name, b.bewertung
  2  FROM
  3     Band_Kuenstlername k,
  4     (SELECT AVG(b.note) AS bewertung, f.mid
  5     FROM Bewertet b, Enthaelt e, HatFunktion f
  6     WHERE b.tid = e.tid
  7             AND e.mid = f.song_mid AND e.sid = f.sid
  8             AND f.funktion = 'Produzent'
  9     GROUP BY f.mid) b
 10  WHERE k.mid = b.mid AND b.bewertung IN
 11     (SELECT MAX(AVG(b.note))
 12     FROM Bewertet b, Enthaelt e, HatFunktion f
 13     WHERE b.tid = e.tid
 14             AND e.mid = f.song_mid AND e.sid = f.sid
 15             AND f.funktion = 'Produzent'
 16     GROUP BY f.mid);

NAME                                                BEWERTUNG
-------------------------------------------------- ----------
bruce springsteen                                          10
kmc                                                        10
shidapu                                                    10
vectrolab                                                  10
ukw                                                        10

Elapsed: 00:00:00.06
SQL>
```

Damit ist die Anfragezeit von ursprünglich 8.62 Sekunden auf 0.06 Sekunden reduziert. Die Ausführungsgeschwindigkeit ist um etwa Faktor 143 gestiegen.

Transaktionen

Wir testen die Transaktionen, indem wir zuerst AUTOCOMMIT deaktivieren, und dann einen Datensatz in unsere Genre-Tabelle einfügen:

```
SQL> SET AUTOCOMMIT OFF;
SQL> SELECT genre_name FROM Genre WHERE Obergenre_name='Classical';

GENRE_NAME
----------------------------------------
Avant Classical
Chamber Music
Classical Guitar
Composers
Opera
Solo Instrumental
Symphony

7 rows selected.

SQL> INSERT INTO Genre VALUES ('Operette', 'Classical');

1 row created.

SQL> SELECT genre_name FROM Genre WHERE Obergenre_name='Classical';

GENRE_NAME
----------------------------------------
Avant Classical
```

```
Chamber Music
Classical Guitar
Composers
Opera
Solo Instrumental
Symphony
Operette

8 rows selected.

SQL> COMMIT;

Commit complete.
```

Eine zweite Transaktion wollen wir durch ROLLBACK abbrechen:

```
SQL> INSERT INTO Genre VALUES ('Piano', 'Classical');

1 row created.

SQL> SELECT genre_name FROM Genre WHERE Obergenre_name='Classical';

GENRE_NAME
----------------------------------------
Avant Classical
Chamber Music
Classical Guitar
Composers
Opera
Solo Instrumental
Symphony
Operette
Piano

9 rows selected.

SQL> ROLLBACK;

Rollback complete.

SQL> SELECT genre_name FROM Genre WHERE Obergenre_name='Classical';

GENRE_NAME
----------------------------------------
Avant Classical
Chamber Music
Classical Guitar
Composers
Opera
Solo Instrumental
Symphony
Operette

8 rows selected.
```

Wie man sieht, enthält die Tabelle nach dem INSERT-Kommando neun Datensätze. Wir brechen die Transaktion mit ROLLBACK ab, wodurch der neu einzufügende Datensatz wieder entfernt wird. Nach dem Rollback sind wieder acht Datensätze vorhanden.

Aufgabenverteilung und Zeiteinteilung

Aufgabenverteilung: In Anjas Aufgabenbereich fielen ein Großteil der SQLPlus-Arbeiten, die Erstellung der Skripte zur Erzeugung der "großen" Datenbasis, und die Programmierung der Java-Applikation. Richard übernahm die Erstellung der "kleinen" Datenbasis in Handarbeit, die Tomcat-Installation, Servlet-Programmierung, und der größere Teil der JDBC-Programmierung.

Zeiteinteilung und Planeinhaltung: Wir haben beide dieses Semester das Softwarepraktikum besucht, und haben festgestellt, dass die Programmierung eines Computerspiels letztlich doch spannender war als die Entwicklung einer Datenbank. Zumindest haben wir für das SWP eine Menge Zeit geopfert, worunter besonders das DBS-Projekt gelitten hat. Wir haben dementsprechend unseren eigenen Zeitplan immer wieder kräftig nach hinten korrigieren müssen.