

# SQL für Trolle

mag.e

Qt-Seminar

Dienstag, 10.2.2009

# SQL ist...

- ▶ ...die Abkürzung für „Structured Query Language“ (früher `SEQUEL` für „Structured English Query Language“)
- ▶ ...ein ISO und ANSI Standard (aktuell SQL:2008)
- ▶ ...eine Befehls- und Abfragesprache für →relationale Datenbanken

# Relationale Datenbanksysteme

- ▶ Eine relationale Datenbank besteht aus einzelnen Tabellen (Relationen), die durch Beziehungen (Joins) miteinander verknüpft sind.
- ▶ Ein Datensatz (Zeile einer Tabelle) wird meist durch ein Feld (eine Spalte, der sog. Primärschlüssel) eindeutig identifiziert.
- ▶ Beziehungen werden durch Verknüpfung mit einem Fremdschlüssel in einer anderen Tabelle hergestellt.
- ▶ Um Datensätze schneller sortieren und suchen zu können, kann man Spalten als Index festlegen.
- ▶ Von Qt unterstützte Datenbanksysteme:  
IBM DB2, MySQL, Oracle, ODBC, PostgreSQL, SQLite („in-process“-Datenbank, also auch ohne Server verwendbar), u.v.a.m.

# SQL Datentypen (Übersicht für MySQL)

- ▶ INT
- ▶ DECIMAL: Fixkommazahl
- ▶ FLOAT: Fließkommazahl
- ▶ DATE, DATETIME: Datum und Uhrzeit
- ▶ CHAR, VARCHAR: kurze Strings
- ▶ TEXT (CLOB): lange Strings („character large object“)
- ▶ BLOB: binäre Daten („binary large object“)
- ▶ undefinierte Felder enthalten NULL

# SQL Ausdrücke

- ▶ "Hier ein sog. \"String\"."
- ▶ 'auch so'
- ▶  $6 * (4 + 2.34)$
- ▶ `cat = 'Buch' AND price >= 50`

# SQL Befehlssatz

- ▶ CREATE, ALTER, DROP  
Manipulation der Datenbankstruktur
- ▶ INSERT, UPDATE, DELETE  
Manipulation des Datenbankinhalts
- ▶ SELECT  
Abfrage von Daten
- ▶ GRANT, REVOKE  
Rechteverwaltung

# INSERT

```
INSERT INTO movies
  (title, director, rating)
VALUES
  ('The Matrix', 'Wachowski', 8.2);
```

# UPDATE

```
UPDATE movies
SET
    director = 'Wachowski Brothers',
    rating = 9.2
WHERE id = 5;
```

# DELETE

```
DELETE FROM movies  
WHERE id = 5;
```

# SELECT

```
SELECT * FROM movies;
```

```
SELECT title, director FROM movies  
WHERE rating > 8;
```

```
SELECT title, director, rating FROM movies  
ORDER BY rating DESC;
```

Beachte: Groß-/Kleinschreibung spielt keine Rolle, dafür ist die Reihenfolge der SQL-Schlüsselworte fest vorgegeben.

# Aggregatfunktionen

- ▶ COUNT(\*), COUNT()
- ▶ MIN(), MAX()
- ▶ SUM(), AVG()

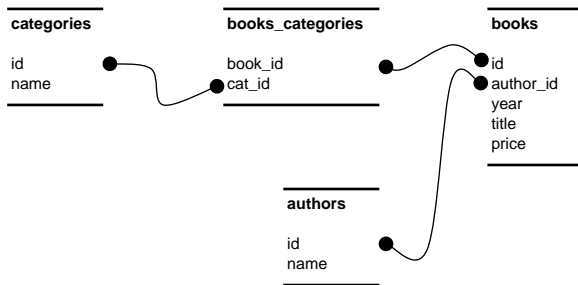
```
SELECT AVG(rating) AS average FROM movies;
```

```
SELECT COUNT(*) AS number FROM movies  
WHERE rating <= 8;
```

# Verbindungsdaten für die Beispieldatenbank

Driver: QMYSQL  
Database Name: qt-test  
Username: qt-troll  
Password: olletrolle  
Hostname: infmathphys.inter.at

# Schema der Beispieldatenbank



# Beispiel 1

Problem:

Alle Bücher mit ihrem Autor auflisten und alphabetisch zuerst nach Autor, dann nach Buchtitel sortieren.

Lösung:

```
SELECT name, title
FROM books INNER JOIN authors
ON author_id = authors.id
ORDER BY name, title;
```

## Beispiel 2

Problem:

Anzahl der Bücher pro Autor in der Datenbank.

Lösung:

```
SELECT name, COUNT(books.id) AS number
  FROM authors LEFT JOIN books
    ON authors.id = author_id
  GROUP BY authors.id;
```

Achtung: COUNT(\*) würde ein anderes Ergebnis liefern.

# Beispiel 3

Problem:

Alle Autoren mit mehr als 3 Büchern.

Lösung:

```
SELECT name, COUNT(books.id) AS number
  FROM authors LEFT JOIN books
    ON authors.id = author_id
 GROUP BY authors.id
  HAVING number >= 3;
```

## Beispiel 4

Problem:

Alle Kategorien, die auch tatsächlich verwendet werden.

Lösung:

```
SELECT DISTINCT name
  FROM categories INNER JOIN books_categories
    ON id = cat_id;
```

Achtung: Ohne das Schlüsselwort `DISTINCT` würde jede Kategorie so oft aufscheinen, wie es Bücher darin gibt.

# Beispiel 5

Problem:

Anzahl der Bücher pro Kategorie.

Lösung:

```
SELECT name, COUNT(book_id) AS number
  FROM categories LEFT JOIN books_categories
    ON id = cat_id
 GROUP BY categories.id;
```

# Beispiel 6

Problem:

Alle Bücher deren Titel mit „D“ beginnt.

Lösung:

```
SELECT title FROM books
WHERE title LIKE 'd%';
```

# Beispiel 7

Problem:

Die 3 teuersten Bücher mit dem jeweiligen Autor.

Lösung:

```
SELECT name, title, price
  FROM books INNER JOIN authors
    ON author_id = authors.id
  ORDER BY price DESC
  LIMIT 0, 3;
```

## Beispiel 8

Problem:

Alle Bücher, die mehr als der Preis-Durchschnitt kosten.

Lösung:

```
SELECT title, price FROM books
  WHERE price > (SELECT AVG(price) FROM books);
```

Beachte:

Solche sog. Subqueries sind erst ab MySQL Version 4.1 implementiert, deshalb kann man das Beispiel als Übung an der SQLite-Movies Datenbank mit `rating` statt `price` ausprobieren.

NULL-Werte werden bei den Aggregatfunktionen ausgeklammert.

# Ende

Man dankt für die Aufmerksamkeit.  
Fragen?