

Programmietechnik

Prof. Dr. Stefan Enderle
NTA Isny

9. Datenspeicherung

Einleitung

- Benötigte Daten können auf viele Arten gespeichert werden.
- Entscheidend sind hierbei hauptsächlich
 - Datenumfang / Menge
 - Zugriffsgeschwindigkeit
 - Synchronisierung
- Mögliche Technologien (für größere Datenmengen):
 - Strukturierte Dateien
 - XML
 - Relationale Datenbanken (SQL)
 - (Objektorientierte Datenbanken)

9.1 Strukturierte Dateien

- Die zu speichernden Daten werden in eine Datei geschrieben.
- Oft wird das ASCII-Format benutzt.
- Trennung durch „White spaces“ (Leerzeichen, Tab, Newline)
- Problem: Strukturierung nur durch Abfolge der Daten
- **Beispiel:** Konfigurationsdatei

```
; lame configuration
[InputFileTypes]
Audio-Files=* .wav;* .aif;* .aiff;* .mp3;* .mp2;* .mp1
Wave-Files=* .wav
MP3-Files=* .mp3
MPx-Files=* .mp3;* .mp2;* .mp1
AIFF-Files=* .aif;* .aiff
...
```

Strukturierte Dateien

- Die Struktur der Datei muss in einem eigenen Dokument beschrieben werden
- Für komplexere Dateistrukturen bietet sich die Definition einer Grammatik an
- **Beispiel:** Grammatik
 - Datei = Dateikopf | Dateikopf Dateirumpf
 - Dateikopf = Autorennamen Datum Zweck
 - Dateirumpf = Ausdruck | Befehl |
Dateirumpf Ausdruck | Dateirumpf Befehl
 - Ausdruck = Zahl | Ausdruck + Zahl | Ausdruck – Zahl
 - Befehl = Text
 - Autorennamen = Text
 - Datum = tt.mm.jjjj
 - Zweck = Text

9.2 XML

- XML = Extensible Markup Language
- Sprache für Datenaustausch und Dokumentenmarkierung
- ASCII Datei

XML

- Ein XML-Dokument besteht aus *Elementen*.
- Ein Element ist Text, der in zueinander passende öffnende und schließende *Tags* eingeschlossen ist.
- Beispiel:

```
<book>  
    Mein Lieblingsbuch  
</book>
```

XML

- Innerhalb eines Elements kann stehen:
 - gewöhnlicher Text
 - wieder Elemente
 - beides
- Beispiel:

```
<book>  
  Mein Lieblingsbuch  
  <autor>  
    Stefan Enderle  
  </autor>  
</book>
```

XML

- Tags markieren die **Struktur** des Dokuments
- Der Text stellt den **Inhalt** dar.

XML

<bibliographie>

<buch>

<autor>

S. Abiteboul

</autor>

<autor>

R. Hull

</autor>

<titel>

Foundation of Databases

</titel>

<verlag>

Addison-Wesey

</verlag>

<jahr>

1985

</jahr>

</buch>

<artikel>

<autor>

E.F. Codd

</autor>

<titel>

A Relational Model of Data Banks

</titel>

<journal>

Communications of the ACM

</journal>

<jahr>

1970

</jahr>

</artikel>

</bibliographie>

DTD

- Die Gesamtstruktur ist immer ein **Baum**.
- Welche **Knoten** welche Unterknoten haben können wird durch eine separate Datei definiert:

Document Type Definition *DTD*

- Die DTD ist im wesentlichen eine kontextfreie Grammatik!

DTD

- Beispiel: Bisherige Schreibweise:

bibliographie	→	(buch artikel)*
buch	→	autoren titel verlag jahr
artikel	→	autoren titel journal jahr
autoren	→	autor autor autoren
autor	→	text
titel	→	text
journal	→	text
jahr	→	text
verlag	→	text

DTD

- Beispiel: DTD Schreibweise:

```
<!ELEMENT bibliographie (buch | artikel)+ >
```

```
<!ELEMENT buch (autor+, titel, verlag, jahr) >
```

```
<!ELEMENT artikel (autor+, titel, journal, jahr) >
```

```
<!ELEMENT autor PCDATA >
```

```
<!ELEMENT titel PCDATA >
```

```
<!ELEMENT journal PCDATA >
```

```
<!ELEMENT jahr PCDATA >
```

```
<!ELEMENT verlag PCDATA >
```

DTD

- Das heißt:
 - Die **DTD** ist die **Grammatik**, die eine Sprache definiert (nämlich die erlaubten XML Dateien).
 - Eine konkrete XML Datei ist dann EIN mögliches **Wort** aus dieser Sprache, mit einem bestimmten Ableitungsbaum.

XML Verarbeitung

- Verarbeitung:
 - Lesen und Auswerten der XML Datei
 - Speichern von Daten als XML Datei
- XML Dateien werden meist durch eines der beiden Frameworks gelesen:
 - SAX
 - DOM

9.2.1 SAX

- SAX = „Simple API for XML“
- Erlaubt Verarbeitung einer XML-Datei schon *während* des Einlesens.
- Hierzu werden **Events** generiert, die dem XML-Stream entsprechen.
→ „Ereignisbasiert“

SAX

- Beispiel:

```
<book>  
  Mein Lieblingsbuch  
</book>
```

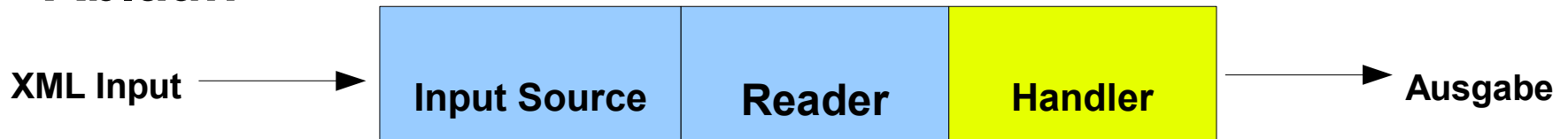
- Events:
 - start-tag(„book“)
 - character-data(„Mein Lieblingsbuch“)
 - end-tag(„book“)

SAX

- **Vorteil:**
 - Kein Einlesen der gesamten Datei in den Hauptspeicher nötig
- **Nachteil:**
 - Kein späterer Zugriff auf Daten möglich

SAX in Qt

- **Ablauf:**



- **XML Input:**

Beliebiger Input, z.B. `QFile`

- **Input Source:**

`QXmlInputSource` gibt an, was als XML-Stream benutzt wird (z.B. File).

- **Reader:**

`QXmlSimpleReader` implementiert einen kompletten XML Reader, dem man noch *ContentHandler* und *ErrorHandler* mitgeben muss.

- **Handler:**

Eigenen Handler ableiten von `QXmlDefaultHandler`.

Beispiel: handler.h

```
#ifndef Handler_H
#define Handler_H

#include <QXmlDefaultHandler>
#include <QXmlSimpleReader>
#include <QString>

class Handler : public QXmlDefaultHandler
{
public:
    Handler() { };

    bool startElement(const QString &namespaceURI, const QString &localName,
        |           |           |           const QString &qName, const QXmlAttributes &attributes);
    bool endElement(const QString &namespaceURI, const QString &localName,
        |           |           |           const QString &qName);
    bool characters(const QString &str);
};

#endif
```

Beispiel: handler.cpp

```
#include "handler.h"
#include <iostream>

using namespace std;

bool Handler::startElement(const QString& /* namespaceURI */,
                          const QString& /* localName */,
                          const QString& qName,
                          const QXmlAttributes& /* attributes */)
{
    cout << "startElement:  qname=" << qName.toString() << endl;
    return true;
}

bool Handler::endElement(const QString& /* namespaceURI */,
                        const QString& /* localName */,
                        const QString& qName)
{
    return true;
}

bool Handler::characters(const QString &str)
{
    return true;
}
```

Beispiel: main.cpp

```
#include "handler.h"
#include <QtXml\QXmlInputSource>
#include <iostream>

using namespace std;

int main()
{
    Handler handler; // Eigenen Handler benutzen

    QXmlSimpleReader reader; // QXmlSimpleReader benutzen
    reader.setContentHandler(&handler); // auf eigenen Handler umbiegen
    reader.setErrorHandler(&handler); // auf eigenen Handler umbiegen

    QFile file("test.xml"); // Datei test.xml öffnen
    if (!file.open(QFile::ReadOnly | QFile::Text)) {
        cout << "could not open file" << endl;
        return -1;
    }

    QXmlInputSource inputSource(&file); // Input source = Datei test.xml

    reader.parse(inputSource); // XML Parsen
}
```

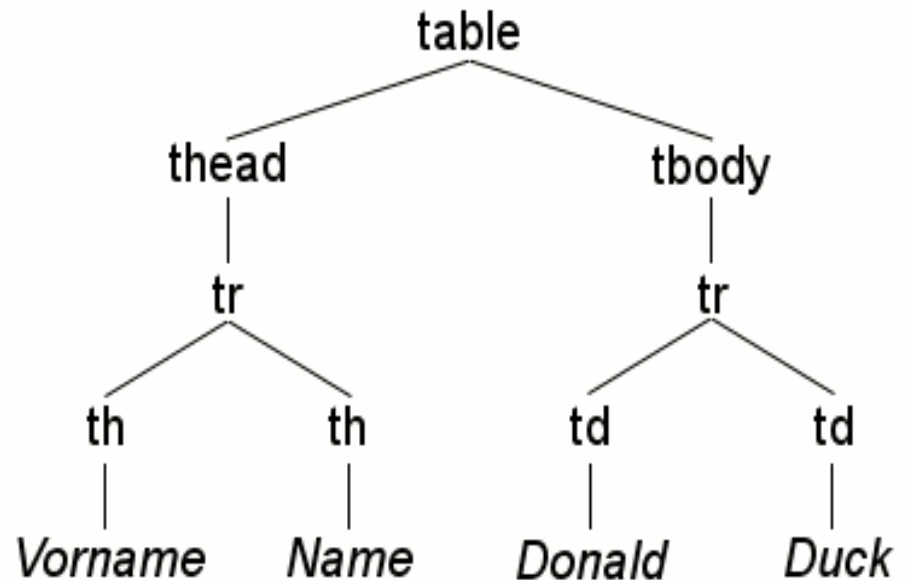
9.2.2 DOM

- DOM = „Document Object Model“
- Liest komplette XML-Datei ein ...
- und wandelt sie in einen **Baum** um.

DOM

- Beispiel:
 - Ausschnitt aus HTML-Datei
 - DOM Baum

```
<table>
  <thead>
    <tr>
      <th>Vorname</th>
      <th>Name</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Donald</td>
      <td>Duck</td>
    </tr>
  </tbody>
</table>
```



DOM

- Zugriff auf die Baumstruktur:

Elemente des Baumes können dann mit bestimmten Funktionen verarbeitet werden:

- Elemente lesen
- Elemente suchen
- Elemente verändern
- Elemente löschen

DOM

- Beispiel: Zugriff auf Elemente (Qt):
 - Dokument spezifizieren:
`QDomDocument doc("mydocument");`
 - Oberstes Element lesen:
`QDomElement docElem = doc.documentElement();`
 - Ersten Kind-Knoten lesen:
`QDomNode n = docElem.firstChild();`
 - Nächsten Kind-Knoten lesen:
`n = n.nextSibling();`

9.3 Datenbanken

- **XML:**
 - XML dient der strukturierten Speicherung von Daten in separaten Dateien (XML Datei).
 - Die Inhalte der XML Dateien können baumartig strukturiert sein.
- **(Relationale) Datenbanken:**
 - Datenbank unterstützen ebenfalls die strukturierte Speicherung von Dateien.
 - Zusätzliche Features:
 - Einfaches Suchen und Einfügen von Datensätzen
 - Client-Server-Betrieb
 - Replikation
 - Relationale Datenbank: Tabellen zur Strukturierung

Datenbanken

- **Querbezug: Hierarchisches Datenbankmodell**
 - Beispiel: DOS oder UNIX Dateisystem entspricht dem hierarchischen Datenbankmodell
 - Hierarchie = Baum
 - Es sind nur 1:n Beziehungen möglich
 - Jedes Verzeichnis kann beliebig viele Unterverzeichnisse (mit Daten) enthalten. Umgekehrt jedoch nur genau eines!
 - DOS / UNIX Kommandos können als Tools zur Benutzung der „Datenbank“ betrachtet werden.

Relationale Datenbanken

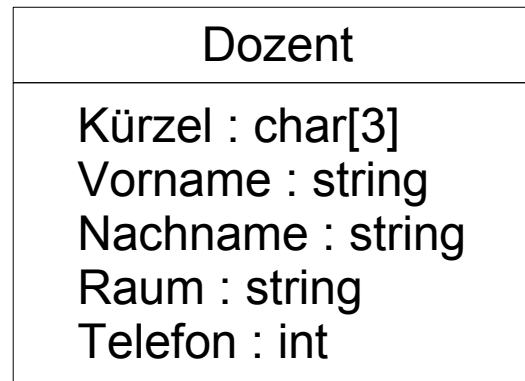
- Ein ***Relationales Datenbanksystem*** besteht aus
 - **Datenbank**
Die „eigentliche Datenmasse“. Besteht ggfs. aus vielen Dateien, die jeweils Tabellen mit Daten beinhalten.
 - **Datenbank Management System (DBMS)**
Vielzahl von Werkzeugen, mit denen die Daten
 - eingegeben
 - geändert,
 - verwaltet
 - geschützt
 - gesichert
 - abgefragt bzw. gesucht werden können.

Relationale Datenbanken

- Die Haupt-Beschreibungseinheit einer relationalen Datenbank ist eine **Tabelle**.
- In den Tabellen werden Objekte mit deren **Attributen** gespeichert.
- Die Überschriften der Tabelle entsprechen den **Attributen** einer Klassendefinition
 - Für jedes Attribut genau eine Spalte.
 - In jeder Zeile der Tabelle befindet sich dann genau ein Objekt der Klasse

Relationale Datenbanken

- Beispiel
 - Klasse in UML:



- Datenbank-Schema: Tabelle „Dozenten“

Kürzel	Name	Vorname	Raum	Telefon
BNR	Bannier	Ulrich	112	3028
DKT	Dankert	Helga	226f	2571
DNK	Dankert	Jürgen	226f	2571
HDN	Haidan	Rainer	226b	4341
WBE	Wiebe	Erhard	129	3001

Relationale Datenbanken

Kürzel	Name	Vorname	Raum	Telefon
BNR	Bannier	Ulrich	112	3028
DKT	Dankert	Helga	226f	2571
DNK	Dankert	Jürgen	226f	2571
HDN	Haidan	Rainer	226b	4341
WBE	Wiebe	Erhard	129	3001

- **Primärschlüssel:**

- Für den **eindeutigen Zugriff** auf ein Objekt, muß mindestens eine Spalte das Schlüsselattribut *Primärschlüssel* enthalten.
- Durch den Primärschlüssel müssen die Zeilen der Tabelle eindeutig voneinander zu unterscheiden sein.
- Gegebenenfalls muß ein Schlüsselattribut zusätzlich "erfunden" werden, wenn keines der Attribute die Bedingung der Eindeutigkeit erfüllt. Beispiel: Index

Relationale Datenbanken

Kürzel	Name	Vorname	Raum	Telefon
BNR	Bannier	Ulrich	112	3028
DKT	Dankert	Helga	226f	2571
DNK	Dankert	Jürgen	226f	2571
HDN	Haidan	Rainer	226b	4341
WBE	Wiebe	Erhard	129	3001

- **Problem:**
 - Manche Dozenten halten sich oft in zwei oder mehr Räumen auf.
- **Schlechte Lösungen:**
 - Noch eine Zeile mit anderem Raum (→ fast gleiches Objekt nochmal)
 - Weitere Spalten „Raum 2, Telefon 2, ...“ (→ Wieviele?)

Relationale Datenbanken

- **Gute Lösung:**
 - Aufteilen der Tabelle in „Wiederholungsdaten“ und sich nicht wiederholende Daten.
 - Tabelle „Dozenten“

Kürzel	Name	Vorname
BNR	Bannier	Ulrich
DKT	Dankert	Helga
DNK	Dankert	Jürgen
HDN	Haidan	Rainer
WBE	Wiebe	Erhard

Tabelle „Räume“

Kürzel	Raum	Telefon
BNR	112	3028
DKT	226f	2571
DNK	226f	2571
DNK	338	3075
HDN	226b	4341
WBE	129	3001
WBE	128	3002

Relationale Datenbanken

- Tabelle „Dozenten“

Kürzel	Name	Vorname
BNR	Bannier	Ulrich
DKT	Dankert	Helga
DNK	Dankert	Jürgen
HDN	Haidan	Rainer
WBE	Wiebe	Erhard

- Tabelle „Räume“

Kürzel	Raum	Telefon
BNR	112	3028
DKT	226f	2571
DNK	226f	2571
DNK	338	3075
HDN	226b	4341
WBE	129	3001
WBE	128	3002

- In „Dozenten“ ist „Kürzel“ der Primärschlüssel
- In „Räume“ ist „Kürzel“ der „**Fremdschlüssel**“ und kann auch mehrfach vorkommen.

Relationale Datenbanken

- Tabelle „Dozenten“

Kürzel	Name	Vorname
BNR	Bannier	Ulrich
DKT	Dankert	Helga
DNK	Dankert	Jürgen
HDN	Haidan	Rainer
WBE	Wiebe	Erhard

- Tabelle „Räume“

Kürzel	Raum	Telefon
BNR	112	3028
DKT	226f	2571
DNK	226f	2571
DNK	338	3075
HDN	226b	4341
WBE	129	3001
WBE	128	3002

- 1. Problem gelöst:
Ein neuer Raum für einen Dozenten wird einfach als weitere Zeile in „Räume“ hinzugefügt.

Relationale Datenbanken

- Tabelle „Dozenten“

Kürzel	Name	Vorname
BNR	Bannier	Ulrich
DKT	Dankert	Helga
DNK	Dankert	Jürgen
HDN	Haidan	Rainer
WBE	Wiebe	Erhard

- Tabelle „Räume“

Kürzel	Raum	Telefon
BNR	112	3028
DKT	226f	2571
DNK	226f	2571
DNK	338	3075
HDN	226b	4341
WBE	129	3001
WBE	128	3002

- Neues Problem:
Räume und zugehörige Telefonnummern tauchen mehrfach auf.
Was tun, wenn sich die Tel.Nr. eines Raumes ändert?

Relationale Datenbanken

- Man sieht, dass ein sauberer Entwurf einer Datenbank wichtig ist.
- Einige Regeln:
 - Zeilen können jederzeit hinzugefügt werden, Spalten nicht!
 - Keine Redundanz!
Jede Angabe über eine Person oder Sache sollte nur einmal gespeichert sein.
 - Unveränderliche Daten immer bevorzugen
(Geburtstag anstatt Alter, usw.)

Relationale Datenbanken

- **Beispiel:**
Ein Dozent will für sich selbst eine kleine Datenbank mit Studenten und deren Prüfungsleistungen anlegen.

Er benötigt folgende Daten:

- Matrikelnummer
 - Name
 - Vorname
 - Geburtsdatum
 - Fach
 - Semester
 - Prüfungsdatum
 - Note
 - Note in Worten
- Wie könnte(n) die Tabelle(n) aussehen?

SQL

- **Zugriff auf eine Relationale Datenbank**
 - PC Anwendungen mit Oberfläche:
 - früher Borland mit dBase
 - jetzt Access
 - weitere: FoxPro, Paradox
 - Auf Großrechnern und Client-Server Lösungen:
 - **SQL** „Structured Query Language“
 - SQL findet man in Oracle, DB/2, Informix,...
 - Seit 1989 genormt (ANSI und ISO)

SQL

- **Wichtige Befehle in SQL**
 - **Anlegen einer neuen Datenbank:**
CREATE DATABASE dbname ;
 - **Löschen einer Datenbank:**
DROP DATABASE dbname ;

SQL

- **Wichtige Befehle in SQL**

- **Anlegen einer neuen Tabelle:**

CREATE TABLE <Tabelle>

(<Attribut> <Datentyp> [, <Attribut> <Datentyp> , ...]);

- **Einige Datentypen:**

- INTEGER Ganze Zahlen
 - DECIMAL(i,n) Dezimalzahl mit insgesamt i Stellen (einschließlich Dezimalpunkt), davon n Nachpunktstellen
 - DATE Datum, Format entsprechend Landeseinstellung, z. B.: tt.mm.jjjj
 - CHAR(n) String mit n Zeichen

SQL

- **Beispiel:**

```
CREATE TABLE Student  
  ( MatNum CHAR(7) , Name CHAR(20) ,  
    VName CHAR(15) , GebDat DATE ) ;
```

SQL

- **Wichtige Befehle in SQL**

- **Einfügen von Daten in eine Tabelle:**

- INSERT INTO <Tabelle> VALUES (<Wert> [, <Wert> ...]);

- **Beispiel:**

- ```
INSERT INTO Student
VALUES ('1230815' , 'Korn' , 'Klara' ,
{14.04.1970}) ;
```

# SQL

- **Wichtige Befehle in SQL**

- **Ändern einer Zeile:**

- UPDATE <Tabelle> SET <Attribut>=<Ausdruck>  
[ , <Attribut>=<Ausdruck> ... ] WHERE <Bedingung> ;

- Beispiel:**

- UPDATE Student  
SET GebDat={12.04.70} WHERE Name='Korn' ;

- **Achtung:** Die Aktion kann mehrere Zeilen betreffen!

# SQL

- **Wichtige Befehle in SQL**

- **Löschen einer Zeile:**

- DELETE FROM <Tabelle> WHERE <Bedingung> ;

- Beispiel:**

- DELETE FROM Student WHERE MatNum='1230815' ;

- **Achtung: Die Aktion kann mehrere Zeilen betreffen!**

# SQL

- **Wichtige Befehle in SQL**

- **Auswählen / Suchen von Daten:**

```
SELECT * FROM <Tabelle>
[WHERE <Bedingung>] [ORDER BY <Attribut>] ;
```

```
SELECT <Ausdruck> [, <Ausdruck> ...]
FROM <Tabelle>
[WHERE <Bedingung>] [ORDER BY <Attribut>] ;
```

# SQL

- **Beispiele:**

- `SELECT * FROM Student;`

- `SELECT Name , VName , MatNum  
FROM Student  
WHERE GebDat < {01.01.1980} ORDER BY Name;`

# SQL

- **SELECT genauer:**

```
SELECT Ausdruck [, Ausdruck ...] | *
FROM tablename
[WHERE Bedingung] [ORDER BY Attribut] ;
```

– in **Ausdruck** kann vorkommen:

- **Attribute** (Spalten-Überschriften),
- **Operatoren** (z. B.: + - \* / ),
- **Klammern**,
- **Funktionen** und
- komplette **SELECT-Anweisungen** (in Klammern).

# SQL

- **SELECT genauer:**
  - **Funktionen** können sein:
    - **Spaltenfunktionen**
      - MIN(Attribut)      Minimum der Spalte
      - MAX(Attribut)      Maximum der Spalte
      - AVG(Attribut)      Durchschnitt der Spalte
      - SUM(Attribut)      Summe der Spalte
      - COUNT(\*)            Anzahl der Zeilen
    - **Mathematische Funktionen**
    - **String-Funktionen**
    - **Datumsfunktionen**
    - ...

# SQL

- **Weitere Beispiele:**

- `SELECT COUNT (*) FROM Student ;`  
**Anzahl der Eintragungen in der Tabelle Student.**

- `SELECT INT ((DATE ()-GebDat)/365)`  
`FROM Student WHERE MatNum='1230816' ;`  
**Alter eines Studenten**

- `SELECT AVG (Note) FROM Pruefungen`  
`WHERE Fach='Mathe' ;`  
**Durchschnitt aller Noten der Mathe-Prüfung.**

- `SELECT AVG (Note) FROM Pruefungen`  
`WHERE (Fach = 'Mathe1' OR Fach = 'Mathe2')`  
`AND Note <= 4 ;`  
**Notendurchschnitt aller bestandenen Prüfungen in den Fächern Mathe1 und Mathe2.**

# SQL in C++ / Qt

- **Beispiele:**

- **Datenbank-Verbindung herstellen:**

```
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
db.setHostName("bigblue");
db.setDatabaseName("flightdb");
db.setUserName("acarlson");
db.setPassword("1uTbSbAs");
bool ok = db.open();
```

- **Anfrage:**

```
QSqlQuery query;
query.exec("SELECT name, salary FROM employee WHERE
salary > 50000");
```

- **Anfrage auswerten/anzeigen:**

```
while (query.next()) {
 QString name = query.value(0).toString();
 int salary = query.value(1).toInt();
 qDebug() << name << salary;
}
```