Datenbank-basierte Webserver

Datenbank-Operationen unter Verwendung von PHP und MySQL

Karl Riedling
Institut für Sensor- und Aktuatorsysteme

Datenbank-Operationen unter Verwendung von PHP und MySQL

- Grundlagen von Datenbanken
- Allgemeines zu MySQL
- Installation und Wartung von MySQL-Datenbanken
- Die SQL-Schnittstelle von MySQL
- Datenbank-Zugriffe unter PHP

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenbank-Operationen unter Verwendung von PHP und MySQL

- **Grundlagen von Datenbanken**
- Allgemeines zu MySQL
- Installation und Wartung von MySQL-Datenbanken
- Die SQL-Schnittstelle von MySQL
- Datenbank-Zugriffe unter PHP

Grundlagen von Datenbanken

- Datenbank-Tabelle:

ID	Titel	Vorname	Nachname	Anrede
int(10)	char(10)	char(20)	char(30)	char(4)
1		Johann	Meier	Herr
2	Dipl.-Ing.	Josef	Müller	Herr
3	Dr.	Maria	Huber	Frau

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Grundlagen von Datenbanken

- Datenbank-Tabelle:

ID	Titel	Vorname	Nachname	Anrede
int(10)	char(10)	char(20)	char(30)	char(4)
1		Johann	Meier	Herr
2	Dipl.-Ing.	Josef	Müller	Herr
3	Dr.	Maria	Huber	Frau

Schlüssel
Feldbezeichnung
Datentype
Feldinhalt
Datensatz

Grundlagen von Datenbanken

- *Schlüssel*: Eindeutiger (meist numerischer) Wert zur Identifikation jedes *Datensatzes* (*Primärer Index*) (meist `INT` mit Zusatz `auto_increment`)
- *Feldbezeichnung*: Innerhalb der Tabelle eindeutige Bezeichnung für eine Spalte mit gleichartigen Inhalten
- *Datentype*: Identifiziert die Inhalte einer Tabellenspalte als z.B. Text, numerische Werte, Datum,…

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Grundlagen von Datenbanken

- *Feldinhalt*: Individuell verschiedener Inhalt einer Tabellenzelle mit einer durch die Datentype des Feldes (der Spalte) vorgegebenen Type; kann unter Umständen auch leer (NULL) sein
- *Datensatz*: Eine Zeile in der Tabelle mit zusammen-gehörigen Daten

Grundlagen von Datenbanken

- Einfache Datenbank: Alle Informationen in einer Tabelle

ID	Titel	Vorname	Nachname	Anrede
int(10)	char(10)	char(20)	char(30)	char(4)
1		Johann	Meier	Herr
2	Dipl.-Ing.	Josef	Müller	Herr
3	Dr.	Maria	Huber	Frau

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Grundlagen von Datenbanken

- *Relationale Datenbank*: verknüpfte Tabellen

ID	Titel	Vorname	Nachname	AnredeID
1		Johann	Meier	2
2	Dipl.-Ing.	Josef	Müller	2
3	Dr.	Maria	Huber	1

ID	Anrede
1	Frau
2	Herr

Grundlagen von Datenbanken

- *Relationale Datenbank*:
  - Mehrere miteinander verknüpfte (oder logisch zusammengehörige) Tabellen
  - Mehrfach in einer Tabellenspalte vorkommende Werte (z.B. im Feld „Anrede“) werden in einer eigenen Tabelle zusammengefasst und durch die Schlüssel-Werte der separaten Tabelle ersetzt

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Grundlagen von Datenbanken

- Relationale Datenbank – Vorteile:
  - Leichtere Wartbarkeit der Daten (jeder Datenwert kommt genau einmal in der Datenbank vor)
  - Daten können leichter gruppiert und Abfragen effizienter durchgeführt werden

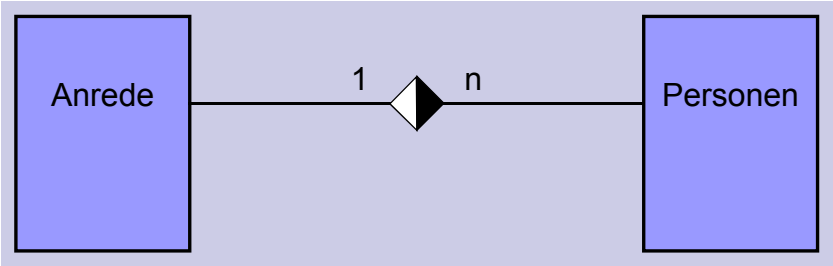
Grundlagen von Datenbanken

- Datenbank-Verknüpfungen:
  - *1:n*-Verknüpfungen: Der Schlüsselwert eines Datensatzes in einer Tabelle kann in einer anderen Tabelle beliebig oft referenziert werden, aber jeder Eintrag in der anderen Tabelle kann nur auf *einen* Eintrag in der ersten Tabelle verweisen
  - Beispiel: Beliebig viele Personen können die Anrede „Herr“ haben, aber jede Person kann nur *eine* Anrede haben („Frau“ oder „Herr“)

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Grundlagen von Datenbanken

- *1:n*-Verknüpfungen im ER- (*Entity Relation*) Diagramm:



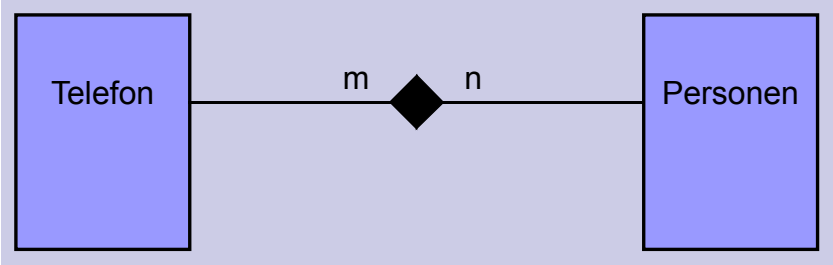
Grundlagen von Datenbanken

- Datenbank-Verknüpfungen:
  - *m:n*-Verknüpfungen: Beliebig viele Datensätze in einer Tabelle können mit beliebig vielen Datensätzen in einer anderen Tabelle verknüpft sein
  - Beispiel: Telefonnummern: Eine Person kann mehrere Telefonnummern haben, und eine Telefonnummer kann von mehreren Personen genutzt werden

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Grundlagen von Datenbanken

- *m:n*-Verknüpfungen im ER-Diagramm:



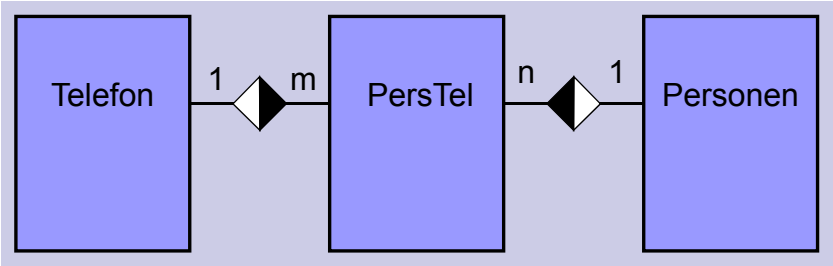
Grundlagen von Datenbanken

- Datenbank-Verknüpfungen:
  - Realisierung von *m:n*-Verknüpfungen: Hilfstabelle mit zwei Schlüsselwerten pro Datensatz (je einem für die beiden zu verknüpfenden Tabellen)
  - Unser Beispiel: Hilfstabelle enthält pro Datensatz je einen Schlüsselwert aus der Tabelle der Personen und der Tabelle der Telefonnummern

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Grundlagen von Datenbanken

- Realisierung einer *m:n*-Verknüpfung:



Grundlagen von Datenbanken

- *Datenbank-Management-System*: Serverseitige Einrichtung zur Verwaltung einer oder mehrerer Datenbanken
- Datenbank-Verwaltung
  - Kontrolle des Zugriffs auf Datenbanken
  - Funktionen zum Erstellen, Ändern und Abfragen von Datenbank-Inhalten (Datensätzen)
  - Funktionen zur Wartung der Datenbank (z.B. Prüfung der Datenintegrität)

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Grundlagen von Datenbanken

- Achtung: Bei einem Datenbank-basierten Webserver-System ist der Web*server Client* der Datenbank!
- Die *Client*-seitige Programmierung einer Datenbank erfolgt daher durch den Programm-Code des Web*servers* (z.B. durch PHP)!

Grundlagen von Datenbanken

- SQL (*Structured Query Language*):
  - Weit verbreitete Programmiersprache für Datenbank-Operationen
  - Es existiert eine Norm (ANSI-SQL/92), die aber nicht von allen Datenbank-Management-Systemen hundertprozentig unterstützt wird
  - Die meisten Datenbank-Management-Systeme verwenden proprietäre Erweiterungen von SQL – Portabilitätsprobleme!

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Grundlagen von Datenbanken

- Implementierung von Datenbanken
  - In der Regel in Form (binärer) Dateien
  - Alternative: XML-Dateien
  - Bei MyISAM-Tabellen (Standard-Tabellentype in MySQL bis MySQL 5.5.4): Ein Unterverzeichnis mit dem Namen der Datenbank; darin typisch je 3 (binäre) Dateien pro Tabelle (Daten, Index und Format-Informationen) mit Basisname = Tabellenname und unterschiedlichen Dateitypen

Grundlagen von Datenbanken

- Implementierung von Datenbanken
  - Bei InnoDB-Tabellen (ab MySQL 5.5.5 Standard-Tabellentype in MySQL): Ein Unterverzeichnis mit dem Namen der Datenbank; darin eine Datei pro Tabelle mit Format-Informationen mit Basisname = Tabellenname und Dateitype „`.frm`“; Daten und Indizes aller InnoDB-Tabellen in einer Datei `ibdata1` im MySQL-Datenverzeichnis (oder alternativ in je einer Datei „`.ibd`“ pro Tabelle)

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenbank-Operationen unter Verwendung von PHP und MySQL

- Grundlagen von Datenbanken
- **Allgemeines zu MySQL**
- Installation und Wartung von MySQL-Datenbanken
- Die SQL-Schnittstelle von MySQL
- Datenbank-Zugriffe unter PHP

Allgemeines zu MySQL

- Als (leistungsfähigerer) Nachfolger von mSQL (*Mini SQL* von *Hughes Technologies*) Mitte der 90er-Jahre von David Axmark, Allan Larsson und Michael Monty Widenius in Uppsala entwickelt
- Ursprünglich vertrieben durch MySQL AB, die Firma der drei Entwickler, mit einem weltweiten Team von (angestellten) Programmierern und einem weltweiten Vertriebs-Netzwerk (siehe `http://www.mysql.com/`)
- Seit Februar 2008 ist MySQL AB eine Tochterfirma von Sun Microsystems, die ihrerseits 2010 von Oracle gekauft wurden

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Allgemeines zu MySQL

- Gleichzeitig mit dem Verkauf von *Sun Microsystems* an *Oracle* hat Michael Monty Widenius mit einer Reihe von MySQL-Entwicklern eine eigene Firma gegründet, die als Klon von MySQL *MariaDB* entwickelt.
- *MariaDB* ist weitgehend kompatibel zu MySQL, aber unter *GNU General Public License* kostenlos verfügbar
- Von MySQL gibt es eine Reihe kostenpflichtiger Editionen (2.000 – 10.000 US-$/Jahr Lizenzgebühr), aber auch eine kostenlose Community Edition

Allgemeines zu MySQL

- Eigenschaften von MySQL:
  - Relationales Datenbank-Management-System
  - *Open Source*-Software; kommerzielle Editionen (zu Oracle-konformen Preisen) und eine kostenlose *Community Edition* unter *GNU Public License* (GPL)
  - Client-Server-System mit *multi-threaded* SQL-Server, der unterschiedliche Backends unterstützt, sowie diversen Client-Programmen und Bibliotheken, APIs und Verwaltungsprogrammen

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Allgemeines zu MySQL

- Eigenschaften von MySQL:
  - Auch als *multi-threaded* Bibliothek zum Einbinden in bestehende Programme verfügbar
  - Von einer großen Software-Basis unterstützt
  - Sehr portabel (C/C++-Quellcode)
  - *Multi-threaded* Kernel (mehrere CPUs unterstützt)
  - Auf Geschwindigkeit optimiert

Allgemeines zu MySQL

- Eigenschaften von MySQL:
  - Diverse Datentypen unterstützt
  - Feste oder variable Datensatzlängen
  - Voreingestellte Werte für alle Datentypen verfügbar
  - Volle Unterstützung von Funktionen und Operatoren in Datenbank-Abfragen
  - Tabellen aus mehreren Datenbanken können in einer Abfrage gemeinsam verwendet werden

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Allgemeines zu MySQL

- Eigenschaften ab MySQL 4:
  - Unterstützung von *Transaktionen* unter Verwendung der *InnoDB* Storage Engine (Transaktionen = Datenbank-Operationen können zurück genommen werden, z.B. bei einem Systemabsturz)
  - *Fulltext*-Suchfunktionen
  - Leicht anpassbare Zeichensätze und *collations* (Sortierordnungen) (z.B. für deutsche Umlaute)
  - Unicode-Unterstützung
  - Unter-Abfragen („*Subqueries*“)

Allgemeines zu MySQL

- Ab MySQL 5.0:
  - *Views* (Abfragen, die als Datenbank-Objekte interpretiert und in der Datenbank gespeichert werden)
  - *Stored Procedures* (Code-Prozeduren für Standard-Operationen)
  - *Triggers* (automatisch ausgeführte SQL-Operationen)
  - `INFORMATION_SCHEMA` (Meta-Informationen zu Datenbank-Tabellen)
  - ➔ MySQL-Dokumentation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Allgemeines zu MySQL

- Ab MySQL 5.1:
  - *Partitioning* (Aufteilung einer Tabelle in mehrere individuelle Tabellen über das Dateisystem)
  - Verbesserte Daten-Replikation und Kooperation von Datenbanken in einem Cluster
  - *Plugin API* (zum Laden von Komponenten ohne Neustart des Servers – verwendet z.B. für verbesserte Volltext-Suche)
  - ➔ MySQL-Dokumentation

Allgemeines zu MySQL

- Ab MySQL 5.5:
  - Verwendung der InnoDB *Storage Engine* als Default (statt MyISAM)
  - Verbesserte Authentifizierung mit *Plugins*
  - Verbesserte Leistungsfähigkeit der Engine (in der *Enterprise Edition*)
  - Verbesserter Unicode-Support
  - ➔ MySQL-Dokumentation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Allgemeines zu MySQL

- Ab MySQL 5.6:
  - Verbesserte Sicherheits-Features (z.B. Überprüfung von Passwörtern, …)
  - Diverse Verbesserungen der InnoDB-Storage Engine (z.B. verbesserte Volltextsuche)
  - Optimierung der Sortier-Funktionalität
  - Änderungen der Default-Werte von Server-Parametern
  - ➔ MySQL-Dokumentation

Allgemeines zu MySQL

- Ab MySQL 5.7:
  - Weitere verbesserte Sicherheits-Features
  - Weitere Verbesserungen der InnoDB-Storage Engine
  - Ende der Unterstützung des alten (MySQL 3.x-) Passwort-Verschlüsselungs-Algorithmus und einiger Datentypen (z.B. zweistellige Jahreszahl)
  - ➔ MySQL-Dokumentation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Allgemeines zu MySQL

- Ab MySQL 8:
  - Globales *Data Dictionary* (erleichtert Upgrades)
  - Verbesserte User-Authentifizierung (nur für Accounts, die unter MySQL 8 erstellt wurden)
  - Partitionierte Tabellen sind nur unter InnoDB oder NDB möglich
  - Default-Zeichensatz ist `utf8mb4`
  - Kompatibilität mit einigen alten SQL-Moden entfernt
  - ➔ MySQL-Dokumentation

Allgemeines zu MySQL

- Neue Features von MySQL:
  - Für Datenbank-basierte Webserver spielen die neuen Features oft keine Rolle (oft sehr einfache Datenbanken!)
  - Änderungen von Default-Einstellungen oder des zulässigen Wertebereichs von Datenbank-Inhalten können aber beim Upgrade auf eine neue MySQL-Version Probleme bereiten – am besten an einer Probe-Installation gründlich testen!

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Allgemeines zu MySQL

- Einschränkung von MySQL:
  - Bei MyISAM-Tabellen werden Verknüpfungen zwischen Tabellen nicht von MySQL selbst geprüft, sondern müssen vom Benutzer-Code (z.B. PHP) verifiziert werden (z.B. beim Löschen von Datensätzen)
  - Diese Einschränkung von MySQL limitiert nicht die Einsatzmöglichkeiten, sie erfordert aber aufwendigere Client-seitige (z.B. PHP-) Programmierung

Allgemeines zu MySQL

- Einschränkung von MySQL:
  - InnoDB erlaubt aber Definition von Verknüpfungen mit automatischen Lösch-Weitergaben oder Blockierungen von Lösch-Operationen

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenbank-Operationen unter Verwendung von PHP und MySQL

- Grundlagen von Datenbanken
- Allgemeines zu MySQL
- **Installation und Wartung von MySQL-Datenbanken**
- Die SQL-Schnittstelle von MySQL
- Datenbank-Zugriffe unter PHP

Installation und Wartung von MySQL-Datenbanken

- Installation von MySQL
- Inbetriebnahme von MySQL
- Wartung von MySQL

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Installation und Wartung von MySQL-Datenbanken

- **Installation von MySQL**
- Inbetriebnahme von MySQL
- Wartung von MySQL

Installation von MySQL

- In aktuellen Linux-Distributionen ist MySQL *nicht (mehr)* in der Default-Installation (wohl aber auf den Distributions-CDs) enthalten
- MySQL muss daher separat installiert werden
- MySQL-Programme werden unter Linux üblicherweise in `/usr/bin` installiert, Datenbanken in `/var/lib/mysql` (ein Unterverzeichnis pro Datenbank)

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Installation von MySQL

- Beim ersten Start von `mysqld` wird die Datenbank `mysql` für die Verwaltung der Zugriffsrechte auf die Datenbanken erstellt

Installation und Wartung von MySQL-Datenbanken

- Installation von MySQL
- **Inbetriebnahme von MySQL**
- Wartung von MySQL

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Inbetriebnahme von MySQL

- Ein User `root` wird mit allen Rechten für alle Datenbanken, allerdings ohne Passwort, automatisch eingerichtet; es sollte jedenfalls ehestmöglich dieser User mit einem Passwort versehen werden (➔ MySQL-Dokumentation)
- Ab MySQL 5.7 erhält der User `root` bei der Installation ein als abgelaufen markiertes Zufallszeichen-Passwort
- Grundsätzlich sollte je ein dedizierter User für jede Datenbank mit (möglichst restriktiven) Rechten (und nur Rechten für diese Datenbank) eingerichtet werden

Inbetriebnahme von MySQL

- MySQL-Usernamen und Passworte sind völlig unabhängig von den Betriebssystems- (z.B. Linux-) Usernamen (gilt auch für `root`)!
- MySQL-Dämon sollte standardmäßig als ein Linux-User mit möglichst restriktiven Rechten laufen (typisch „`mysql`“), *niemals* als `root`!

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Installation und Wartung von MySQL-Datenbanken

- Installation von MySQL
- Inbetriebnahme von MySQL
- **Wartung von MySQL**

Wartung von MySQL

- Mit MySQL-Command-Line-Tools (z.B. `mysql` oder einem der spezialisierten Wartungsprogramme, z.B. `mysqladmin`) ➔ MySQL-Dokumentation
- Mit dem Web-Interface von `phpMyAdmin` (`http://www.phpmyadmin.net/`) oder ähnlichen Produkten (`phpMyAdmin` erfordert funktionsfähiges PHP)

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Wartung von MySQL

- `phpMyAdmin`
  - Web-Interface
  - Sehr komfortable, intuitiv verwendbare Benutzeroberfläche
  - Erlaubt praktisch alle Manipulationen an Datenbanken (Erstellung und Änderung von Datenbanken, Datenbank-Tabellen und Datenbank-Inhalten)

Wartung von MySQL

- `phpMyAdmin`
  - **Vorsicht:** Unbedingt Zugangsbeschränkung vorsehen (Default: Name + Passwort eines MySQL-Users, z.B. `root`; evtl. Einschränkung in der Apache-Konfiguration auf gewisse IP-Adressen)
  - Sehr leistungsfähig und daher gefährlich – `phpMyAdmin` *niemals* ungeschulten Usern zugänglich machen!

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenbank-Operationen unter Verwendung von PHP und MySQL

- Grundlagen von Datenbanken
- Allgemeines zu MySQL
- Installation und Wartung von MySQL-Datenbanken
- **Die SQL-Schnittstelle von MySQL**
- Datenbank-Zugriffe unter PHP

Die SQL-Schnittstelle von MySQL

- MySQL-Sprachstruktur
- Zeichensätze
- Datentypen
- Funktionen
- Datenmanipulation
- Tabellen-Manipulation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Die SQL-Schnittstelle von MySQL

- **MySQL-Sprachstruktur**
- Zeichensätze
- Datentypen
- Funktionen und Operatoren
- Datenmanipulation
- Tabellen-Manipulation

MySQL-Sprachstruktur

- Strings und numerische Konstanten
- Namen
- Variable
- Kommentare

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

MySQL-Sprachstruktur

- **Strings und numerische Konstanten**
- Namen
- Variable
- Kommentare

MySQL-Sprachstruktur

- Strings:
  - Dargestellt in einfachen oder doppelten Anführungszeichen (nur einfache Anführungszeichen sind ANSI-kompatibel!):

```
'Das ist ein ANSI-kompatibler String'
"Dieser String ist zulässig, aber nicht ANSI-kompatibel"
```

  - *Escape*-Sequenzen für Sonderzeichen

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

MySQL-Sprachstruktur

- *Escape*-Sequenzen für Sonderzeichen:

`\0`	ASCII-NUL (`%00`)	`\t`	Tabulator (`%09`)
`\'`	Einf. Anführungszeichen	`\Z`	Ctrl-Z (*End of File* – `%1b`)
`\"`	Dopp. Anführungszeichen	`\\`	Backslash („`\`“ – `%5c`)
`\b`	*Backspace* (`%08`)	`\%`	Prozentzeichen („%“)
`\n`	*Line Feed* (`%0a`)	`\_`	Unterstreichung („_“)
`\r`	*Carriage Return* (`%0d`)		

MySQL-Sprachstruktur

- Anführungszeichen in Strings:
  - Gleiche Anführungszeichen wie die für den String verwendeten:
    - Verdoppeln: `"Ein ""Anführungszeichen"" im String"`
    - Als *Escape*-Sequenz darstellen: `"Ein \"Anführungszeichen\" im String"`
  - Unterschiedliche Anführungszeichen wie die für den String verwendeten: Normal verwenden: `'Ein "Anführungszeichen" im String'`

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

MySQL-Sprachstruktur

- Unzulässige Zeichen in allen Strings vor der Übergabe an MySQL (z.B. in einer Abfrage) durch *Escape*-Sequenzen ersetzen
  - „zu Fuß“
  - mit geeigneter API-Funktion, z.B. mit PHP Funktionen `mysql(i)_real_escape_string()` (ab PHP 4.3) oder `addslashes()` (ab PHP 3)

MySQL-Sprachstruktur

- Zahlen
  - Ganze Zahlen: Vorzeichen zulässig: „`0`“, „`10`“, „`-4711`“
  - Gleitkommazahlen: Dezimalpunkt; Vorzeichen und Exponent sind zulässig: „`123.456`“, „`1.00`“, „`1.602e-19`“
  - Ganze Zahlen können auch dort verwendet werden, wo Gleitkommawerte erwartet werden

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

MySQL-Sprachstruktur

- Zahlen
  - Hexadezimalzahlen ➔ MySQL-Dokumentation
  - Boolesche Werte (ab MySQL 4.1) ➔ MySQL-Dokumentation
  - Bitfelder (ab MySQL 5.0.3) ➔ MySQL-Dokumentation
  - NULL („keine Daten“): Kann durch „`NULL`“ (*case insensitive*) oder „`\N`“ (*case sensitive*) dargestellt werden (bei Datenimporten und -exporten durch „`\N`“)

MySQL-Sprachstruktur

- Strings und numerische Konstanten
- **Namen**
- Variable
- Kommentare

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

MySQL-Sprachstruktur

- Namen:
  - Namen von Datenbanken, Tabellen und Spalten sind auf 64 Zeichen beschränkt (Aliase auf 255 Zeichen)
  - Datenbank-Namen: Alle Zeichen, die für Verzeichnisnamen zulässig sind, außer „/“, „\“ und „.“
  - Tabellen-Namen: Alle Zeichen, die für Dateinamen zulässig sind, außer „/“ und „.“

MySQL-Sprachstruktur

- Namen:
  - Namen dürfen auch mit einer Ziffer beginnen, aber nicht nur aus Ziffern bestehen!
  - Reservierte Namen in verkehrte einfache Anführungszeichen setzen:

```
SELECT * FROM `select` WHERE `select`.id > 100;
```

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

MySQL-Sprachstruktur

- Namen:
  - Datenbank- und Tabellennamen sind unter Linux *case sensitive*, unter Windows nicht (➔ MySQL-Dokumentation)
  - Aliase für Tabellennamen sind immer *case sensitive*, solche für Spaltennamen nicht
  - Alle anderen Namen sind nicht *case sensitive* (Spaltennamen, aber auch Schlüsselwörter und Namen von Funktionen)

MySQL-Sprachstruktur

- Reservierte Namen:
  - Namen von Datentypen und Funktionen dürfen als Namen von Spalten, Tabellen und Datenbanken verwendet werden; es darf aber beim Aufruf einer Funktion kein Leerzeichen zwischen dem Namen der Funktion und den Funktionsklammern stehen; z.B. „`ABS(-1)`“
  - Reservierte Namen in verkehrte einfache Anführungszeichen („`` ` ``“) setzen
  - Liste reservierter Namen ➔ MySQL-Dokumentation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

MySQL-Sprachstruktur

- Zugriff auf Tabellenspalten:
  - `col_name`: Eine Spalte mit dem (eindeutigen) Namen `col_name`
  - `tbl_name.col_name`: Spalte `col_name` der Tabelle `tbl_name`
  - `db_name.tbl_name.col_name`: Spalte `col_name` der Tabelle `tbl_name` aus Datenbank `db_name`

MySQL-Sprachstruktur

- Strings und numerische Konstanten
- Namen
- **Variable ➔ MySQL-Dokumentation**
- Kommentare

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

MySQL-Sprachstruktur

- Strings und numerische Konstanten
- Namen
- Variable
- **Kommentare**

MySQL-Sprachstruktur

- Kommentare:
  - „#“: Kommentar bis zum Zeilenende
  - „`-- `“: Kommentar bis zum Zeilenende; nach dem zweiten „`-`“ *muss* ein Leerzeichen stehen!
  - „`/* ... */`“: Kommentar innerhalb einer Zeile, oder über mehrere Zeilen. Vorsicht bei Anführungszeichen oder Strichpunkten in diesen Kommentaren! ➔ MySQL-Dokumentation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Die SQL-Schnittstelle von MySQL

- MySQL-Sprachstruktur
- **Zeichensätze**
- Datentypen
- Funktionen und Operatoren
- Datenmanipulation
- Tabellen-Manipulation

Zeichensätze

- MySQL unterstützt ab Version 4.0 dynamisch unterschiedliche Zeichensätze (*character sets*) und Sortierordnungen (*collations*)
- Zeichensatz: ein Satz von Symbolen und ihren internen Codierungen
- Sortierordnung: Gibt an, wie Elemente des Zeichensatzes sortiert und miteinander verglichen werden
- Ab MySQL 4.1 werden Unicode (Zeichensatz `ucs2`) und UTF-8 (Zeichensatz `utf8`) unterstützt

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Zeichensätze

- Sortierordnungen (*collations*):
  - Binäre (z.B. nach Zeichencodes): „1“ < „A“ < „B“ < „a“
  - Unabhängig von Groß- und Kleinschreibung (*case insensitive*): „a“ < „B“
- 8-Bit-Zeichensatz für deutsche Texte: `latin1`
  - Default-*Collation*: `latin1_swedish_ci`: „ü“ gleichbedeutend mit „y“
  - *Collation* `latin1_german1_ci`: „ü“ gleichbedeutend mit „u“
  - *Collation* `latin1_german2_ci`: „ü“ gleichbedeutend mit „ue“

Zeichensätze

- Sortier-Beispiel für drei `latin1`-*Collations*

`latin1_`**`swedish`**`_ci`	`latin1_`**`german1`**`_ci`	`latin1_`**`german2`**`_ci`
Muffe	Muffe	Müller
MX Systems	Müller	Muffe
Müller	MX Systems	MX Systems
MySQL	MySQL	MySQL
	„*Wörterbuch*“	„*Telefonbuch*“

- Details ➔ MySQL-Dokumentation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Die SQL-Schnittstelle von MySQL

- MySQL-Sprachstruktur
- Zeichensätze
- **Datentypen**
- Funktionen und Operatoren
- Datenmanipulation
- Tabellen-Manipulation

Datentypen

- Ganzzahlige numerische Typen
- Gleitkomma-Typen
- Datum und Zeit
- Strings
- Aufzählungen und Mengen
- Geoinformations-Datentypen
- JSON-Datentype

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- **Ganzzahlige numerische Typen**
- Gleitkomma-Typen
- Datum und Zeit
- Strings
- Aufzählungen und Mengen
- Geoinformations-Datentypen
- JSON-Datentype

Datentypen

- Ganzzahlige numerische Typen
  - Optionale Attribute bei der Deklaration:
    - Maximale Anzahl der angezeigten Stellen: z.B. `INT(5)` (hat nur Einfluss auf die Ausgabe, nicht auf die gespeicherten Werte!)
    - `UNSIGNED`: Wert als positive Zahl interpretiert; um Faktor 2 größerer Wertebereich
    - `ZEROFILL`: Nicht benutzte Stellen werden mit Nullen aufgefüllt ausgegeben, z.B. Wert „7“ in einer „`INT(3) ZEROFILL`“-Spalte als „`007`“ statt „ 7“

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Ganzzahlige numerische Typen
  - Optionale Attribute bei der Deklaration:
    - `SERIAL`: Alias für `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE` (Datentype für Schlüssel-Feld) (ab MySQL 5).

Datentypen

- Ganzzahlige numerische Typen
  - `BIT` (1 … 64 Bit): Bitfeld ➔ MySQL-Dokumentation
  - `TINYINT` (8 Bit): Wertebereich `-128` … `127` bzw. `0` … `255`; `BOOLEAN` und `BOOL`: Synonyme für `TINYINT(1)` (nur MySQL!)
  - `SMALLINT` (16 Bit): Wertebereich `-32768` … `32767` bzw. `0` … `65536`
  - `MEDIUMINT` (24 Bit): Wertebereich `-8388608` … `8388607` bzw. `0` … `16777215` (nur MySQL!)

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Ganzzahlige numerische Typen
  - `INT` oder `INTEGER` (32 Bit): Wertebereich `-2147483648` … `2147483647` bzw. `0` … `4294967295`
  - `BIGINT` (64 Bit): Wertebereich `-9223372036854775808` … `9223372036854775807` bzw. `0` … `18446744073709551615` (nur MySQL!)

Datentypen

- Ganzzahlige numerische Typen
  - Ganzzahlige Arithmetik in MySQL erfolgt mit vorzeichenbehafteten `BIGINT`s
  - Werte, die den zulässigen Wertebereich überschreiten, werden auf den oberen bzw. unteren Grenzwert begrenzt

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Ganzzahlige numerische Typen
  - Vorsicht mit PHP-Code: PHP unterstützt (bis einschließlich PHP 5.x) nur vorzeichenbehaftete 32-Bit-Werte (also MySQL-Type `INT`)!
  - `BIGINT`-Werte können aber als Strings übergeben werden

Datentypen

- Ganzzahlige numerische Typen
- **Gleitkomma-Typen**
- Datum und Zeit
- Strings
- Aufzählungen und Mengen
- Geoinformations-Datentypen
- JSON-Datentype

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Gleitkomma-Typen
  - Exakte Repräsentation (`NUMERIC`, `DECIMAL`) oder Näherung (`FLOAT`, `REAL`, `DOUBLE PRECISION`)
  - Optionale Parameter:
    - Maximale Anzahl der angezeigten Stellen und Dezimalstellen: z.B. `FLOAT(10,4)` (hat bei exakten Datentypen Einfluss auf die gespeicherten Werte!)
    - `UNSIGNED` und `ZEROFILL` wie bei ganzzahligen Typen (`UNSIGNED` hat keinen Einfluss auf positiven Wertebereich!)

Datentypen

- Gleitkomma-Typen
  - `FLOAT(precision)`: Für `precision <= 24` äquivalent zu `FLOAT`, für `24 < precision <= 53` äqivalent zu `DOUBLE` (ODBC-Kompatibilität)
  - `FLOAT` (*single precision* 32 Bit-Gleitkomma): Wertebereich `-3.402823466E+38` … `-1.175494351E-38`, `0`, `1.175494351E-38` … `3.402823466E+38`

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Gleitkomma-Typen
  - `DOUBLE`, `DOUBLE PRECISION`, `REAL` (*double precision* 64 Bit-Gleitkomma): Wertebereich `-1.7976931348623157E+308` … `-2.2250738585072014E-308`, `0`, `2.2250738585072014E-308` … `1.7976931348623157E+308`
  - Gleitkomma-Operationen werden intern mit `DOUBLE` vorgenommen – Vorsicht beim Vergleich von `FLOAT`s mit `DOUBLE`s!

Datentypen

- Gleitkomma-Typen
  - `DECIMAL`, `DEC`, `NUMERIC`, `FIXED`: Exakte Darstellung, Werte werden intern als Strings gespeichert.
    - Maximaler Wertebereich: wie bei `DOUBLE`s!
    - Exakte Darstellung mit vorgegebener (nicht zu großer) Anzahl von Stellen vermeidet Rundungsfehler bei Gleitkomma-Operationen (`1.2299999…` → `1.23`)

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Ganzzahlige numerische Typen
- Gleitkomma-Typen
- **Datum und Zeit**
- Strings
- Aufzählungen und Mengen
- Geoinformations-Datentypen
- JSON-Datentype

Datentypen

- Datum und Zeit
  - `DATE`: Datum im Format „`JJJJ-MM-TT`“; Wertebereich `'1000-01-01'` … `'9999-12-31'`; Werte können als Strings oder (**Vorsicht!**) Zahlen übergeben werden ➔ MySQL-Dokumentation
  - `DATETIME`: Datum und Zeit im Format „`JJJJ-MM-TT HH:MM:SS`“; Wertebereich `'1000-01-01 00:00:00'` … `'9999-12-31 23:59:59'`; Werte können als Strings oder Zahlen übergeben werden ➔ MySQL-Dokumentation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Datum und Zeit
  - `TIME`: Zeit im Format „`HH:MM:SS`“; Wertebereich `'-838:59:59'` … `'838:59:59'`; Werte können als Strings oder (**Vorsicht!**) Zahlen übergeben werden ➔ MySQL-Dokumentation
  - `YEAR`, `YEAR(2)`, `YEAR(4)`: Jahr in zwei- oder (Default) vierstelliger Darstellung; Wertebereich `0000`, `1901` … `2155` (4 Stellen) bzw. `1970` … `2069` (2 Stellen); Werte können als Strings oder Zahlen übergeben werden ➔ MySQL-Dokumentation

Datentypen

- Datum und Zeit
  - `TIMESTAMP`: Darstellung von Datum und Zeit als Anzahl der Sekunden seit 1.1.1970 0 Uhr (32 Bit-Wert); Werte bis ins Jahr 2038 darstellbar
    - Unter MySQL vor 4.1 erlaubte ein optionaler Längenparameter (z.B. `TIMESTAMP(6)`) die Ausgabe als `JJJJMMTTHHMMSS` (14; Default), `JJMMTTHHMMSS` (12), `JJJJMMTT` (8) oder `JJMMTT` (6)
    - Ab MySQL 4.1 werden `TIMESTAMP`-Felder im gleichen Format wie `DATETIME` ausgegeben

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Datum und Zeit
  - `TIMESTAMP`:
    - Nicht explizit initialisierte `TIMESTAMP`-Spalten werden auf den Zeitpunkt des letzten `INSERT` oder `UPDATE` gesetzt
    - Details ➔ MySQL-Dokumentation

Datentypen

- Datum und Zeit
  - Ungültige Datums- oder Zeit-Werte werden auf `0` (z.B. für `DATETIME '0000-00-00 00:00:00'`) gesetzt
  - Vor MySQL 3.23 waren in `DATE-` oder `DATETIME-` Feldern Monats- und Tagesangaben von „`0`“ verboten
  - MySQL 3.23 – 4.1 tolerierte sowohl „`2014-00-00`“ als auch „`2014-02-31`“
  - MySQL 5.+ erlaubt zwar „`2014-00-00`“, aber nicht mehr ungültige Daten wie „`2014-02-31`“ (auch Schaltjahre werden berücksichtigt!)

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Ganzzahlige numerische Typen
- Gleitkomma-Typen
- Datum und Zeit
- **Strings**
- Aufzählungen und Mengen
- Geoinformations-Datentypen
- JSON-Datentype

Datentypen

- Strings
  - Längenangaben von Strings sind in Bytes (vor MySQL 4.1) bzw. Zeichen (ab MySQL 4.1)
  - Bei vielen String-Datentypen (und auch bei Aufzählungs- und Mengentypen) ist bei der Definition eines Feldes die Angabe eines Zeichensatzes und einer Sortierordnung möglich ➔ MySQL-Dokumentation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Strings
  - `CHAR(M)`, `NATIONAL CHAR(M)`, `NCHAR(M)`: Datenfeld der Länge *M*; `0 < M <= 255`
    - Strings mit einer Länge < *M* werden rechts mit Leerzeichen aufgefüllt, die beim Auslesen wieder entfernt werden
    - Daten der Type `CHAR` werden *case insensitive* sortiert und verglichen, wenn nicht das optionale Attribut `BINARY` definiert wurde
    - `CHAR BYTE` ist ein Alias für `BINARY`
    - `CHAR` ist ein Synonym für `CHAR(1)`

Datentypen

- Strings
  - `VARCHAR(M)`, `NATIONAL VARCHAR(M)`: Datenfeld mit der maximalen Länge *M*; `0 < M <= 255` (unter MySQL 5.0.3); `0 < M <= 65535` (ab MySQL 5.0.3); maximale effektive Länge ist 65,532 Bytes!
    - Leerzeichen am Ende von Strings werden vor MySQL 5.1 beim Abspeichern entfernt, ab MySQL 5.1 nicht (Kompatibilität zum SQL-Standard!)

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Strings
  - `VARCHAR(`*`M`*`)`, `NATIONAL VARCHAR(`*`M`*`)` :
    - Kürzere Strings brauchen weniger Speicherplatz, aber ein bis zwei zusätzliche Bytes für die Länge
    - Daten der Type `VARCHAR` werden *case insensitive* sortiert und verglichen, wenn nicht das optionale Attribut `BINARY` definiert wurde

Datentypen

- Strings
  - `BINARY(`*`M`*`)`:
    - Ähnlich zur Type `CHAR`, aber für Binärdaten vorgesehen
  - `VARBINARY(`*`M`*`)`:
    - Ähnlich zur Type `VARCHAR`, aber für Binärdaten vorgesehen

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Strings
  - Wenn in einem Ausdruck mindestens eine Spalte mit dem Attribut `BINARY` versehen ist, werden alle in diesem Ausdruck vorkommenden Spalten *case sensitive* sortiert und verglichen!

Datentypen

- Strings
  - Die Verwendung der Type `CHAR` ist vorzuziehen, wenn:
    - Sonst nur Felder fixer Länge in der Tabelle vorkommen
    - Die Länge der Daten in allen Datensätzen gleich ist (z.B. *Hashes* für Passwörter, Session-IDs o.ä.)
  - In allen anderen Fällen ist in der Regel die Type `VARCHAR` effizienter

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Strings
  - `TINYBLOB`, `TINYTEXT`: Stringfeld variabler Länge mit maximal 255 (`2^8 - 1`) Zeichen
  - `BLOB`, `TEXT`: Stringfeld variabler Länge mit maximal 65535 (`2^16 - 1`) Zeichen
  - `MEDIUMBLOB`, `MEDIUMTEXT`: Stringfeld variabler Länge mit maximal 16777215 (`2^24 - 1`) Zeichen
  - `LONGBLOB`, `LONGTEXT`: Stringfeld variabler Länge mit maximal 4294967295 (`2^32 - 1`) Zeichen

Datentypen

- Strings
  - Der Unterschied zwischen `BLOB`- (*Binary Large OBject*) und `TEXT`-Typen liegt darin, dass `BLOB`s *case sensitive* und `TEXT`s *case insensitive* sortiert und verglichen werden
  - Strings länger als die maximal zulässige Länge werden abgeschnitten
  - Die tatsächlich nutzbare maximale Länge von `LONGBLOB`s ist auf < 16 MB begrenzt (Client-Server-Protokoll!)

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Ganzzahlige numerische Typen
- Gleitkomma-Typen
- Datum und Zeit
- Strings
- **Aufzählungen und Mengen**
- Geoinformations-Datentypen
- JSON-Datentype

Datentypen

- Aufzählungen und Mengen
  - `ENUM('Wert1','Wert2',...)`: Stringfeld, das genau einen Wert aus der Liste der bei der Deklaration angegebenen (maximal 65535) Werte haben kann, z.B. `ENUM('ja','nein')`; Wert im Fehlerfall: leerer String
    - `ENUM`s werden über einen Index, beginnend mit `1`, dargestellt; der leere String hat den Index `0`
    - Die Eingabe von `ENUM`s ist *case insensitive*, die Ausgabe erfolgt aber immer mit der bei der Definition angegebenen Schreibweise

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Aufzählungen und Mengen
  - `SET('Wert1','Wert2',...)`: Stringfeld, das einen oder mehrere Werte aus der Liste der bei der Deklaration angegebenen (maximal 64) Stringwerte haben oder den leeren String enthalten kann
    - SETs werden auf die Bits eines ganzzahligen Datenelements abgebildet
    - Die Reihenfolge der Eingabe der Elemente eines SETs ist gleichgültig; Elemente können auch mehrfach angegeben werden

Datentypen

- Ganzzahlige numerische Typen
- Gleitkomma-Typen
- Datum und Zeit
- Strings
- Aufzählungen und Mengen
- **Geoinformations-Datentypen**
- JSON-Datentype

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- Geoinformations-Datentypen: Spezielle Datentypen für die Verarbeitung von Geoinformations-Daten (ab MySQL 5.5) ➔ MySQL-Dokumentation

Datentypen

- Ganzzahlige numerische Typen
- Gleitkomma-Typen
- Datum und Zeit
- Strings
- Aufzählungen und Mengen
- Geoinformations-Datentypen
- **JSON-Datentype**

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datentypen

- JSON-Datentype: Spezielle Datentype für die Speicherung von Daten in JSON- (*JavaScript Object Notation*) Dokumenten (ab MySQL 5.7) ➔ MySQL-Dokumentation

Die SQL-Schnittstelle von MySQL

- MySQL-Sprachstruktur
- Zeichensätze
- Datentypen
- **Funktionen und Operatoren**
- Datenmanipulation
- Tabellen-Manipulation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Funktionen und Operatoren

- Funktionen und Operatoren werden in `SELECT`- und `WHERE`-Konstrukten von MySQL verwendet
- Sie können Inhalte von Tabellen(spalten) und externe Parameter bearbeiten oder verknüpfen

Funktionen und Operatoren

- Ausdrücke, bei denen ein Operand `NULL` ist, ergeben im Allgemeinen `NULL`
- Zwischen dem Namen einer Funktion und der öffnenden Klammer darf kein Leerraum (*White Space*) liegen!
- MySQL enthält zahlreiche Funktionen für die Bearbeitung allgemeiner Daten, numerischer Daten, von Strings und von Datums-Zeit-Werten

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Funktionen und Operatoren

- Für Datenbank-basierte Webserver werden im Allgemeinen nur Vergleichs- und logische Verknüpfungsoperatoren in MySQL-Abfragen benötigt
- Für einen vollständigen Überblick über die MySQL-Funktionen und Operatoren ➔ MySQL-Dokumentation

Funktionen und Operatoren

- Vergleichsoperatoren (Auswahl):
  - Strings und Zahlen können miteinander verglichen werden
  - Strings werden *case insensitive* verglichen, außer einer der beiden Strings ist als `BINARY` definiert
  - Vergleichsoperatoren liefern als Ergebnis `1` für einen wahren, `0` für einen falschen Ausdruck, und `NULL` für einen Ausdruck, bei dem einer der Operanden `NULL` ist (außer bei „<=>“)

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Funktionen und Operatoren

- Vergleichsoperatoren (Auswahl):
  - „=“: Gleich (**Vorsicht**: Der Gleichheits-Operator ist in PHP „==“, in MySQL „=“!
  - „<>“, „!=“: Ungleich
  - „<=“: Kleiner gleich
  - „<“: Kleiner
  - „>=“: Größer gleich
  - „>“: Größer
  - „<=>“: NULL-sicherer Gleichheits-Operator (`NULL <=> NULL` → `1`)

Funktionen und Operatoren

- Vergleichsoperatoren (Auswahl):
  - „`IS NULL`“, „`IS NOT NULL`“ für Test auf Werte, die `NULL` sind

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Funktionen und Operatoren

- Vergleichsoperatoren (Auswahl):
  - „`LIKE`“: Vergleich von Strings unter Berücksichtigung von *Wildcards* im rechten Operanden:
    - „%“ steht für eine beliebige Anzahl (einschließlich 0) von Zeichen
    - „_“ steht für *genau ein* Zeichen
    - Beim Vergleich mit `LIKE` müssen „%“ und „_“ im rechten Operanden durch „`\%`“ bzw. „`\_`“ ersetzt werden, wenn sie nicht als *Wildcards* gelten sollen
    - In allen anderen Fällen gilt „`\%`“ als „`\%`“, nicht „%“!

Funktionen und Operatoren

- Logische Operatoren:
  - Logische Operatoren ergeben „`1`“ für einen wahren und „`0`“ für einen falschen Ausdruck; „`NULL`“ wird meist als „`0`“ interpretiert
  - „`NOT`“, „`!`“: Logisches `NICHT`
  - „`OR`“, „`||`“: Logisches `ODER`
  - „`AND`“, „`&&`“: Logisches `UND`
  - „`XOR`“: Logisches `XOR` (Exklusiv-Oder)

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Funktionen und Operatoren

- Volltext-Suchfunktionen
  - Verfügbar ab MySQL 3.23
  - Nur Suche nach ganzen Worten (bzw. im *Boole*-schen Modus evtl. nach Beginn eines Worts)
  - Erfordert speziellen `FULLTEXT`-Index für die durchsuchten Datenfelder
  - Ist (speziell bei großen Tabellen) signifikant schneller als Suche mit `LIKE`
  - Ergebnisse nach „Relevanz“ gereiht
  - Details ➔ MySQL-Dokumentation

Die SQL-Schnittstelle von MySQL

- MySQL-Sprachstruktur
- Zeichensätze
- Datentypen
- Funktionen und Operatoren
- **Datenmanipulation**
- Tabellen-Manipulation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Befehle für die Manipulation von Daten innerhalb einer Tabelle:
  - Abfrage von Daten(sätzen) – `SELECT`
  - Erstellen eines neuen Datensatzes – `INSERT`
  - Ändern eines bestehenden Datensatzes – `UPDATE`
  - Löschen eines Datensatzes – `DELETE`

Datenmanipulation

- Befehle für die Manipulation von Daten innerhalb einer Tabelle (2. Teil):
  - Löschen aller Daten in der Tabelle – `TRUNCATE`
  - Ersetzen von Datensätzen – `REPLACE`
  - Import-Datei laden – `LOAD DATA INFILE`

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Befehle für die Manipulation von Daten innerhalb einer Tabelle:
  - **Abfrage von Daten(sätzen) – SELECT**
  - Erstellen eines neuen Datensatzes – `INSERT`
  - Ändern eines bestehenden Datensatzes – `UPDATE`
  - Löschen eines Datensatzes – `DELETE`

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - `SELECT`-Befehle geben grundsätzlich das Ergebnis eines (arithmetischen oder Datenbank-Abfrage-) Ausdrucks zurück, z.B.:
    `SELECT 3 * 4;` → `12`
    (**Alle Befehle in MySQL mit einem Strichpunkt abschließen!**)
  - Hier nur einfachste Formen von `SELECT`-Befehlen behandelt; Details ➔ MySQL-Dokumentation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - `SELECT [DISTINCT[ROW]] <ausdruck1, ausdruck2, ...>`
    `FROM <tabelle1, tabelle2, ...>`
    `WHERE <bedingungen>`
    `GROUP BY <spalte1, spalte2, ...>`
    `ORDER BY <spalte1, spalte2, ...> [ASC | DESC];`
  - (Syntax vereinfacht; ➔ MySQL-Dokumentation)

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - `SELECT <`**`ausdruck1, ausdruck2, ...`**`>`
  - `ausdruck` kann sein:
    - Spaltenname in der Form `col_name`, `tbl_name.col_name` oder `db_name.tbl_name.col_name`
    - Ergebnis eines Ausdrucks oder einer Funktion mit oder ohne Verwendung von Tabelleninformationen
    - Aliase sind zulässig
    - „*“ als `ausdruck` liefert bei Abfragen aus Tabellen *alle* Felder aller (passenden) Datensätze

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - **`FROM <tabelle1, tabelle2, ...>`**
  - Gibt an, aus welcher Tabelle bzw. aus welchen Tabellen die Daten für die Abfrage kommen sollen

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - **`WHERE <bedingungen>`**
  - Legt fest, welche Datensätze ausgewählt werden sollen
  - Wenn keine `WHERE`-Klausel angegeben ist, werden alle Datensätze aus den angegebenen Tabellen ausgewählt
  - **Vorsicht**: Wenn mit `FROM` mehrere Tabellen gewählt wurden, *muss* mit `WHERE` (oder `JOIN`) eine Bedingung angegeben werden, die die Tabellen miteinander verknüpft!

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Abfrage von Daten(sätzen) – SELECT:
  - **GROUP BY <spalte1, spalte2, ...>**
  - Fasst Datensätze zusammen, die gleiche Inhalte in den angegebenen Spalten aufweisen
  - GROUP BY unter Angabe einer Schlüssel-Spalte kann den selben Effekt haben wie ein SELECT DISTINCT- oder SELECT DISTINCTROW-Befehl, kann aber je nach Abfrage bei einer großen Anzahl von Datensätzen viel schneller sein

Datenmanipulation

- Abfrage von Daten(sätzen) – SELECT:
  - **ORDER BY <spalte1, spalte2, ...> [ASC | DESC]**
  - Sortiert die Datensätze nach den angegebenen Spalten in aufsteigender (ASC) oder absteigender (DESC) Reihenfolge

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Beispiel: Tabellen „`Name`“ und „`Anrede`“

ID	Titel	Vorname	Nachname	AnredeID
1		Hermann	Maier	2
2	Dipl.-Ing.	Josef	Müller	2
3	Dr.	Johanna	Huber	1

ID	Anrede
1	Frau
2	Herr

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - 1. Beispiel:
    Alle Datensätze von Damen (`AnredeID` = 1 entspricht Anrede „Frau“) aus der Tabelle `Namen`, alphabetisch sortiert zuerst nach Nach- und dann nach Vornamen:

```
SELECT * FROM Namen WHERE AnredeID=1
ORDER BY Nachname, Vorname;
```


© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - 2. Beispiel (einfache *1:n*-Verknüpfung – alphabetische Liste aller Personen, deren Vornamen mit „J“ beginnen):

```
SELECT CONCAT(t2.Anrede, ' ', t1.Titel,
' ', t1.Vorname, ' ', t1.Nachname) AS
NameLang FROM Namen AS t1, Anrede AS t2
WHERE t1.AnredeID = t2.ID &&
LEFT(t1.Vorname, 1) = 'J' ORDER BY
Nachname, Vorname;
```

Datenmanipulation

- Beispiel: Tabellen „`Name`“ und „`Anrede`“

ID	Titel	Vorname	Nachname	AnredeID
1		Hermann	Maier	2
2	Dipl.-Ing.	Josef	Müller	2
3	Dr.	Johanna	Huber	1

ID	Anrede
1	Frau
2	Herr

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - Ergebnis:

NameLang
Frau Dr. Johanna Huber
Herr Dipl.-Ing. Josef Müller

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - *Aliase*:

```
SELECT CONCAT(t2.Anrede, ' ', t1.Titel,
' ', t1.Vorname, ' ', t1.Nachname) AS
NameLang FROM Namen AS t1, Anrede AS t2
WHERE t1.AnredeID = t2.ID &&
LEFT(t1.Vorname, 1) = 'J' ORDER BY
Nachname, Vorname ASC;
```

Aliase

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - Beispiel (alphabetische Liste aller Personen, deren Vornamen mit „J“ beginnen):

```
SELECT CONCAT(t2.Anrede, ' ', t1.Titel,
' ', t1.Vorname, ' ', t1.Nachname) AS
NameLang FROM Namen AS t1, Anrede AS t2
WHERE t1.AnredeID = t2.ID &&
LEFT(t1.Vorname, 1) = 'J' ORDER BY
Nachname, Vorname ASC;
```

Verknüpfung der Tabellen „`Namen`“ und „`Anrede`“

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - Alternative für 2. Beispiel (einfache *1:n*-Verknüpfung – alphabetische Liste aller Personen, deren Vornamen mit „J“ beginnen):

```
SELECT CONCAT(t2.Anrede, ' ', t1.Titel,
' ', t1.Vorname, ' ', t1.Nachname) AS
NameLang FROM Namen AS t1 LEFT JOIN
Anrede AS t2 ON t1.AnredeID = t2.ID
WHERE LEFT(t1.Vorname, 1) = 'J' ORDER BY
Nachname, Vorname;
```

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - Wichtigster Unterschied zwischen „`FROM t1, t2 WHERE t1.AnredeID = t2.ID`“ und „`FROM t1 LEFT JOIN t2 ON t1.AnredeID = t2.ID`“: „`WHERE`“ liefert nur jene Datensätze, für die in *beiden* Tabellen Daten existieren (also der Wert von `t1.AnredeID` ein Äquivalent in `t2.ID` hat); „`LEFT JOIN`“ liefert *alle* Datensätze aus `t1`, auch jene, bei denen kein (gültiger) Wert in `t1.AnredeID` steht

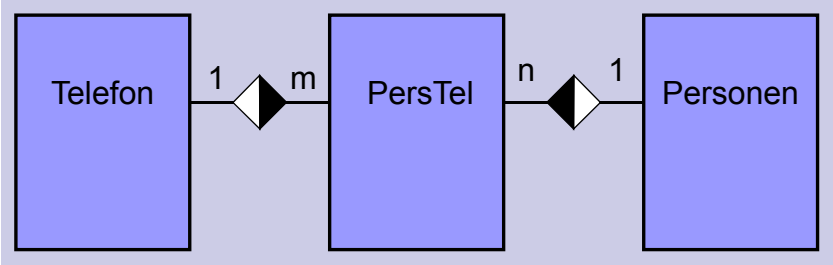
Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - Details zur Syntax von „`JOIN`“ ➔ MySQL-Dokumentation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - 3. Beispiel (*m:n*-Verknüpfung – Alle Personen mit einer bestimmten Telefonnummer („4711“)):



Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - 3. Beispiel (*m:n*-Verknüpfung):

```
SELECT p.* FROM PersTel AS pt, Personen
AS p, Telefon as t WHERE pt.TelefonID =
t.ID && pt.PersonenID = p.ID && t.Nummer
= 4711 ORDER BY Nachname, Vorname ASC;
```


© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Abfrage von Daten(sätzen) – SELECT:
  - 3. Beispiel (*m:n*-Verknüpfung):

```
SELECT p.* FROM PersTel AS pt, Personen
AS p, Telefon as t WHERE pt.TelefonID =
t.ID && pt.PersonenID = p.ID && t.Nummer
= 4711 ORDER BY Nachname, Vorname ASC;
```

Verknüpfungs-Bedingungen

Datenmanipulation

- Abfrage von Daten(sätzen) – SELECT:
  - 3. Beispiel (*m:n*-Verknüpfung):

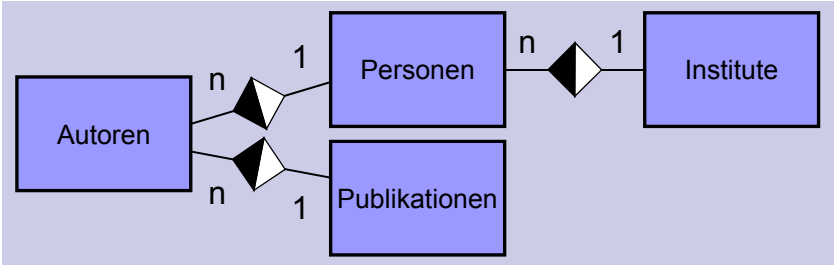
```
SELECT p.* FROM PersTel AS pt, Personen
AS p, Telefon as t WHERE pt.TelefonID =
t.ID && pt.PersonenID = p.ID && t.Nummer
= 4711 ORDER BY Nachname, Vorname ASC;
```

Abfrage-Bedingung

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - 4. Beispiel (komplexe Verknüpfung – Alle Publikationen von Angehörigen eines Instituts):



Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - 4. Beispiel (komplexe Verknüpfung – Alle Publikationen von Angehörigen des Instituts E 366, keine Sortierung):

```
SELECT t1.* FROM Autoren t0,
Publikationen t1, Personen t2, Institute
t3 WHERE t0.PublikationID = t1.ID AND
t0.PersonenID = t2.ID AND t2.InstitutID
= t3.ID AND t3.InstName = 'E 366' GROUP
BY t1.ID;
```


© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - Ohne „`GROUP BY t1.ID`“ würden alle Publikationen, bei denen mehrere Autoren dem Institut E 366 angehören, mehrfach in der Ausgabeliste aufscheinen (einmal für jeden Autor)
  - Alternative zu `GROUP BY t1.ID` : `SELECT DISTINCT`

Datenmanipulation

- Abfrage von Daten(sätzen) – `SELECT`:
  - Variante für Ausgabe eindeutiger Datensätze:

```
SELECT DISTINCT t1.* FROM Autoren t0,
Publikationen t1, Personen t2, Institute
t3 WHERE t0.PublikationID = t1.ID AND
t0.PersonenID = t2.ID AND t2.InstitutID
= t3.ID AND t3.InstName = 'E 366';
```

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Befehle für die Manipulation von Daten innerhalb einer Tabelle:
  - Abfrage von Daten(sätzen) – `SELECT`
  - **Erstellen eines neuen Datensatzes – INSERT**
  - Ändern eines bestehenden Datensatzes – `UPDATE`
  - Löschen eines Datensatzes – `DELETE`

Datenmanipulation

- Erstellen eines neuen Datensatzes – `INSERT`:
  - `INSERT INTO <tabelle> [(<spalte1>, <spalte2>, ...)] VALUES (<ausdruck1>, <ausdruck2>, ...);`
  - `INSERT INTO <tabelle> SET <spalte1>=<ausdruck1>, <spalte2>=<ausdruck2>,...;`
  - Andere Varianten ➔ MySQL-Dokumentation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Erstellen eines neuen Datensatzes – `INSERT`:
  - Wenn beim `INSERT`-Befehl eine Liste von Spalten angegeben ist, muss nach „`VALUES`“ zu jeder Spalte genau ein (typmäßig geeigneter) Wert angegeben werden
  - Wird keine Liste von Spalten angegeben, so muss für *jede* Spalte der Tabelle ein Wert angegeben werden

Datenmanipulation

- Erstellen eines neuen Datensatzes – `INSERT`:
  - Wenn eine Spalte nicht explizit gesetzt wird, wird sie im neuen Datensatz auf ihren Default-Wert gesetzt
  - Spalten mit dem Attribut „`NOT NULL`“ werden auf ihren Default-Wert gesetzt, wenn für sie der Wert `NULL` übergeben wurde (**Vorsicht**: „`NULL`“, nicht „`'NULL'`“)

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Erstellen eines neuen Datensatzes – `INSERT`:
  - Wenn die Übergabewerte mit PHP übergeben werden, kann ein Wert von `NULL` für die für die Initialisierung verwendete Variable zu einem MySQL-Syntaxfehler führen – Werte für numerische Spalten aufbereiten:

```
<?php
if ($wert === NULL) $wert = "NULL";
?>
```


Datenmanipulation

- Erstellen eines neuen Datensatzes – `INSERT`:
  - Alternative für vorige Lösung: Übergabewert `NULL` in gültigen numerischen Wert umwandeln:

```
<?php
$wert = (int) $wert;
    // macht aus NULL den Zahlenwert 0
?>
```


© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Beispiel: Einfügen einer Zeile in Tabelle „Name“:

ID	Titel	Vorname	Nachname	AnredeID
1		Hermann	Maier	2
2	Dipl.-Ing.	Josef	Müller	2
3	Dr.	Johanna	Huber	1

```
INSERT INTO Name VALUES NULL, 'Prof.Dr.', 'Lise',
'Meitner', 1;
```


Datenmanipulation

- Beispiel: Einfügen einer Zeile in Tabelle „Name“:

ID	Titel	Vorname	Nachname	AnredeID
1		Hermann	Maier	2
2	Dipl.-Ing.	Josef	Müller	2
3	Dr.	Johanna	Huber	1
4	Prof.Dr.	Lise	Meitner	1

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Befehle für die Manipulation von Daten innerhalb einer Tabelle:
  - Abfrage von Daten(sätzen) – `SELECT`
  - Erstellen eines neuen Datensatzes – `INSERT`
  - **Ändern eines bestehenden Datensatzes – `UPDATE`**
  - Löschen eines Datensatzes – `DELETE`

Datenmanipulation

- Ändern eines bestehenden Datensatzes – `UPDATE:`
  - `UPDATE <tabelle> SET`
    `<spalte1>=<ausdruck1>,`
    `<spalte2>=<ausdruck2>, ...`
    `[WHERE <bedingung>] [LIMIT <anzahl>];`
  - `<bedingung>` und `LIMIT` schränken ein, welche bzw. wie viele Datensätze geändert werden sollen

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Ändern eines bestehenden Datensatzes – `UPDATE`:
  - Die Ausdrücke in einem `UPDATE`-Befehl können auch Werte von Datenfeldern beinhalten:
    `UPDATE personen SET alter=alter+1;`
    (elegante Methode für die Implementierung von Zugriffszählern u.dgl.)
  - Solche Ausdrücke werden von links nach rechts abgearbeitet ➔ MySQL-Dokumentation

Datenmanipulation

- Befehle für die Manipulation von Daten innerhalb einer Tabelle:
  - Abfrage von Daten(sätzen) – `SELECT`
  - Erstellen eines neuen Datensatzes – `INSERT`
  - Ändern eines bestehenden Datensatzes – `UPDATE`
  - **Löschen eines Datensatzes – `DELETE`**

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Löschen eines Datensatzes – `DELETE`:
  - `DELETE FROM <tabelle> [WHERE <bedingung>] [LIMIT <anzahl>];`
  - `<bedingung>` und `LIMIT` schränken ein, welche bzw. wie viele Datensätze gelöscht werden sollen
  - Ohne Angabe einer `<bedingung>` werden *alle* Datensätze gelöscht
  - **MySQL prüft *nicht*, ob der zu löschende Datensatz in einer anderen Tabelle referenziert wird!**

Datenmanipulation

- Befehle für die Manipulation von Daten innerhalb einer Tabelle:
  - **Löschen aller Daten in der Tabelle – `TRUNCATE`**
  - Ersetzen von Datensätzen – `REPLACE`
  - Import-Datei laden – `LOAD DATA INFILE`

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Löschen aller Daten in der Tabelle – `TRUNCATE:`
  - `TRUNCATE TABLE <tabelle>;`
  - `TRUNCATE` löscht die Tabelle und erstellt sie neu – schneller als `DELETE`
  - Nicht *Transaction*-fest
  - Kann verwendet werden, um defekte Tabellen zu reparieren

Datenmanipulation

- Befehle für die Manipulation von Daten innerhalb einer Tabelle:
  - Löschen aller Daten in der Tabelle – `TRUNCATE`
  - **Ersetzen von Datensätzen – `REPLACE`**
  - Import-Datei laden – `LOAD DATA INFILE`

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Ersetzen von Datensätzen – `REPLACE`:
  - Syntax wie bei `INSERT`
  - Funktioniert wie `INSERT`; wenn jedoch ein Datensatz existiert, dessen primärer Schlüsselwert der gleiche ist wie beim neuen Datensatz, wird der bestehende Datensatz überschrieben

Datenmanipulation

- Befehle für die Manipulation von Daten innerhalb einer Tabelle:
  - Löschen aller Daten in der Tabelle – `TRUNCATE`
  - Ersetzen von Datensätzen – `REPLACE`
  - **Import-Datei laden – `LOAD DATA INFILE`**

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenmanipulation

- Import-Datei laden – `LOAD DATA INFILE:`
  - Datensätze werden aus einer Textdatei eingelesen, die (z.B.) mit `SELECT ... INTO OUTFILE` oder `SELECT ... INTO DUMPFILE` erstellt wurde ➔ MySQL-Dokumentation

Die SQL-Schnittstelle von MySQL

- MySQL-Sprachstruktur
- Zeichensätze
- Datentypen
- Funktionen und Operatoren
- Datenmanipulation
- **Tabellen-Manipulation**

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Tabellen-Manipulation

- Befehle zum Verwalten von Datenbanken und Tabellen
  ➔ MySQL-Dokumentation:
  - `CREATE DATABASE` – Neue Datenbank erstellen
  - `DROP DATABASE` – Datenbank löschen
  - `CREATE TABLE` – Neue Tabelle erstellen
  - `ALTER TABLE` – Tabellenstruktur ändern
  - `RENAME TABLE` – Tabelle umbenennen
  - `DROP TABLE` – Tabelle löschen
  - `CREATE INDEX` – Neuen Index erstellen
  - `DROP INDEX` – Index entfernen

Datenbank-Operationen unter Verwendung von PHP und MySQL

- Grundlagen von Datenbanken
- Allgemeines zu MySQL
- Installation und Wartung von MySQL-Datenbanken
- Die SQL-Schnittstelle von MySQL
- **Datenbank-Zugriffe unter PHP**

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenbank-Zugriffe unter PHP

- Unter PHP sind seit PHP-Version 3 fast 50 API (*Application Program Interface*) -Funktionen für Zugriffe auf eine MySQL-Datenbank verfügbar (Funktionen „`mysql_...`“)
- Mit PHP 5.0 wurde die „*Improved MySQL Extension*“ eingeführt, die objektorientierte (z.B. Objekt „`mysqli`“) oder prozedurale Zugriffe (Funktionen „`mysqli_...`) auf die erweiterten Funktionalitäten von MySQL 4.1+ erlaubt.
- Objektorientierte und prozedurale `mysqli`-Aufrufe sollten aber nicht gemischt werden!

Datenbank-Zugriffe unter PHP

- Im Wesentlichen bilden die „`mysqli_...`“-Funktionen die Funktionalität der alten „`mysql_...`“-Funktionen ab,
- ABER: Die Reihenfolge der Aufrufparameter unterscheidet sich bei den „`mysqli_...`“-Funktionen von der bei den „`mysql_...`“-Funktionen!

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenbank-Zugriffe unter PHP

- Die Umstellung alter, auf den „`mysql_`“-Funktionen basierender Anwendung ist daher nicht trivial möglich
- Die „`mysqli_`“-Funktionen benötigen z.T. auch unterschiedliche Aufrufs-Strukturen (z.B.: Funktion **`mysql`**`_free_result()` galt als optional und nicht unbedingt erforderlich; **`mysqli`**`_free_result()` wird hingegen dringend empfohlen)
- Ab PHP 5.5 gelten die „`mysql_...`“-Funktionen als *deprecated*

Datenbank-Zugriffe unter PHP

- Die PHP-Klassen bzw. -Funktionen erlauben sämtliche unter MySQL zulässige Operationen an Datensätzen, Tabellen und Datenbanken (➔ PHP-Dokumentation)
- Hier nur die wichtigsten Funktionen behandelt
- Für die Schnittstelle PHP – MySQL existieren einige Konfigurationseinstellungen in `php.ini`, die zum Teil dynamisch geändert werden können ➔ PHP-Dokumentation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenbank-Zugriffe unter PHP

- Zugriff auf eine (MySQL-) Datenbank unter PHP:
  - Verbindung zum Datenbank-Server aufnehmen
  - Datenbank auswählen
  - SQL-Abfragen vornehmen oder SQL-Befehle ausführen
  - evtl. Verbindung zum Datenbank-Server wieder abbauen

Datenbank-Zugriffe unter PHP

- Verbindung zum Datenbank-Server aufnehmen
  - `mysqli::__construct()`; `mysqli_connect()`
  - Zwei Varianten: *nicht persistente* oder *persistente* Verbindung (`host`-Parameter "`p:`“ voranstellen)
  - Nicht persistente Verbindungen werden beim Verlassen der PHP-Seite wieder abgebaut, persistente bleiben bestehen
  - Persistente Verbindungen sind effizienter, brauchen aber mehr Server-Ressourcen und spezielle Konfiguration ➔ Dokumentation

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenbank-Zugriffe unter PHP

- Verbindung zum Datenbank-Server aufnehmen – objektorientiert:
  ```
  $mysqli = new mysqli('localhost', $username, $passwort, $datenbank)
  ```
- Alle Parameter sind optional; Defaults können in `php.ini` festgelegt werden
- Das Objekt `$mysqli` wird für alle weiteren Datenbank-Zugriffe benötigt
- Es ist möglich (und sinnvoll), Benutzer-definierte MySQL-Klassen von der Klasse `mysqli` abzuleiten

Datenbank-Zugriffe unter PHP

- Verbindung zum Datenbank-Server aufnehmen – prozedural
  - `mysqli_connect()` gibt als Ergebnis ein Objekt der Type `mysqli` zurück:
    ```
    <?php
    $mysqli = mysqli_connect("localhost", $username, $passwort, $datenbank);
    ?>
    ```
  - Das *Link Identifier*-Objekt `$mysqli` wird für alle weiteren Datenbank-Zugriffe benötigt

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenbank-Zugriffe unter PHP

- Datenbank auswählen (bei Änderung der aktuellen Datenbank; Default sollte mit `mysqli::_construct()` bzw. `mysqli_connect()` festgelegt werden):
  - Funktionen `select_db()` bzw. `mysqli_select_db()`:

```
<?php
$mysqli->select_db($datenbank);
// bzw.
mysqli_select_db ($mysqli, $datenbank);
?>
```


Datenbank-Zugriffe unter PHP

- Nur *eine* Datenbank kann zu einem gegebenen Zeitpunkt aktiv sein. Für Abfragen aus mehreren Datenbanken die SQL-Syntax `db_name.tbl_name` verwenden (➔ PHP-Dokumentation)!

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenbank-Zugriffe unter PHP

- SQL-Abfragen vornehmen oder SQL-Befehle ausführen
  - Beliebige SQL-Befehle können mit den PHP-Funktionen `query()` bzw. `mysqli_query()` abgesetzt werden

```
<?php
$result = $mysqli->query ($query);
```

bzw.

```
$result = mysqli_query ($mysqli, $query);
?>
```

  - SQL-Befehlsstring `$query` in `mysqli_query()` *nicht* mit „`;`“ abschließen!

Datenbank-Zugriffe unter PHP

- SQL-Abfragen vornehmen oder SQL-Befehle ausführen
  - (`mysqli_`)`query()` gibt als Ergebnis die folgenden Daten zurück:
    - Im Falle eines Fehlers: immer `false`
    - Bei SQL-Abfragen und -Befehlen, die Daten als Ergebnis liefern, ein Objekt der Type `mysqli_result`, das das Ergebnis der Abfrage bzw. des Befehls enthält (bzw. den Zugriff darauf erlaubt)
    - Bei erfolgreich ausgeführten Befehlen, die *keine* Daten als Ergebnis haben, `true`

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenbank-Zugriffe unter PHP

- SQL-Abfragen vornehmen oder SQL-Befehle ausführen
  - Beispiel (objektorientiert):

```
<?php
$result = $mysqli->query ("SELECT * FROM Namen
ORDER BY Nachname, Vorname") or die ("Ungültige
Abfrage: ".$mysqli->error);
?>
```

  - `die()` bewirkt Abbruch des PHP-Scripts
  - `$error` enthält MySQL-Fehlermeldung

Datenbank-Zugriffe unter PHP

- SQL-Abfragen vornehmen oder SQL-Befehle ausführen
  - Beispiel (prozedural):

```
<?php
$result = mysqli_query ($mysqli, "SELECT * FROM
Namen ORDER BY Nachname, Vorname") or die
("Ungültige Abfrage: ".mysqli_error($mysqli));
?>
```

  - `die()` bewirkt Abbruch des PHP-Scripts
  - `mysqli_error()` liefert MySQL-Fehlermeldung

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenbank-Zugriffe unter PHP

- SQL-Abfragen vornehmen oder SQL-Befehle ausführen
  - (Syntaktisch) gültige Abfragen können trotzdem keine Daten als Ergebnis liefern!
  - Die Ergebnisse einer Abfrage werden MySQL-intern vorgehalten und können von dort unter Angabe von `$result` mit einer der (`mysqli_`)`fetch_...()`-Funktionen abgerufen werden (➔ PHP-Dokumentation)

Datenbank-Zugriffe unter PHP

- SQL-Abfragen vornehmen oder SQL-Befehle ausführen
  - Beispiel (objektorientiert):

```
<?php
$result = $mysqli->query ($query);
while ($zeile = $result->fetch_object())
{
  echo "$zeile->Vorname $zeile->Nachname<br>\n";
}
$result->free();
?>
```

  - Abfrageergebnis immer mit `free()` freigeben!

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenbank-Zugriffe unter PHP

- SQL-Abfragen vornehmen oder SQL-Befehle ausführen
  - Beispiel (prozedural):

```
<?php
$result = mysqli_query ($mysqli, $query);
while ($zeile = mysqli_fetch_object($result))
{
  echo "$zeile->Vorname $zeile->Nachname<br>\n";
}
mysqli_free_result($result);
?>
```

  - Abfrageergebnis immer mit `mysqli_free_result()` freigeben!

Datenbank-Zugriffe unter PHP

- SQL-Abfragen vornehmen oder SQL-Befehle ausführen
  - Anzahl der mit einem `SELECT`-SQL-Befehl erhaltenen Datensätze kann mit (`mysqli_`)`num_rows` erhalten werden:

```
<?php
$result = $mysqli->query ($query);
echo $result->num_rows()." Datensätze";
```

bzw.

```
$result = mysqli_query ($mysqli, $query);
echo mysqli_num_rows ($result)." Datensätze";
?>
```

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme

Datenbank-Zugriffe unter PHP

- Verbindung zum Datenbank-Server wieder abbauen
  - Nicht persistente Verbindungen zum Datenbank-Server können mit (`mysqli_`)`close()` abgebaut werden:

```
<?php
$mysqli->close();
```

bzw.

```
mysqli_close ($mysqli);
?>
```

  - Eigentlich unnötig, weil die Verbindungen am Ende des Scripts ohnehin automatisch abgebaut werden (sollten)

© Karl Riedling, TU Wien, Institut für Sensor- und Aktuatorsysteme