Vielen Dank an Dennis Riehle
für die Bereitstellung dieser Folien

1.1 Definition Datenbank

Ein **Datenbanksystem** (DBS) ist ein System zur elektronischen Datenverwaltung. Die wesentliche Aufgabe eines DBS ist es, große Datenmengen effizient, widerspruchsfrei und dauerhaft zu speichern und benötigte Teilmengen in unterschiedlichen, bedarfsgerechten Darstellungsformen für Benutzer und Anwendungsprogramme bereitzustellen.

Ein DBS besteht aus zwei Teilen: der Verwaltungssoftware, genannt **Datenbankmanagementsystem** (DBMS) und der Menge der zu verwaltenden Daten, der eigentlichen **Datenbank** (DB). Die Verwaltungssoftware organisiert intern die strukturierte Speicherung der Daten und kontrolliert alle lesenden und schreibenden Zugriffe auf die Datenbank. Zur Abfrage und Verwaltung der Daten bietet ein Datenbanksystem eine **Datenbanksprache** an.

Quelle: http://de.wikipedia.org/wiki/Datenbank

1.2 Verschiedene DBMS

MySQL
Bietet eine kostenlose Version an, sowie eine kommerzielle Version
www.mysql.com

MsSQL
Nur für Windows, kostenlose Version mit starken Einschränkungen
www.microsoft.com/sql/default.mspx

PostgreSQL
Open-Source, kostenlos, großer Funktionsumfang
www.postgresql.org

1.3 Structured Query Language

SQL ist eine Datenbanksprache, welche u.a. folgende Operationen unterstützt:

Datenabfrage
Datenmanipulation
Datendefinition
Rechteverwaltung

Alle vorhin erwähnten DBMS verwenden eine mehr oder weniger modifizierte Version von ANSI-SQL (standardisiertes SQL).

Die nachfolgenden Informationen beziehen sich auf MySQL, sollten prinzipiell jedoch ohne größere Änderungen auf andere DBMS übertragbar sein.

Für MySQL findet man das Handbuch mit allen wichtigen Erklärungen unter:
http://dev.mysql.com/doc/refman/5.1/de/

2.1 Einfache SQL Abfragen

```
SELECT
     *
FROM
   Users
```

Mit einem Sternchen werden alle Spalten der Tabelle selektiert, es wird somit der komplette Inhalt der Tabelle angezeigt.

```
SELECT
     Username,
     Password
FROM
     Users
```

Durch die Angabe von Spaltennamen kann auf einzelne Spalten projiziert werden.

```
SELECT
     Username AS Benutzername,
     Password AS Passwort
FROM
     Users
LIMIT
   50, 10
```

Spalten können beim Selektieren beliebig umbenannt werden, was für die weitere Verarbeitung sinnvoll sein kann.

Mit LIMIT kann die Ergebnismenge auf eine Teilmenge reduziert werden. In diesem Beispiel werden die Datensätze 50 – 59 zurückgegeben (es wird bei 0 angefangen zu zählen).

2.2 Erweiterte SQL Abfragen

```
SELECT
      Username,
      Password
FROM
   Users
WHERE
      Username = 'Thomas'
```

Durch Angabe einer WHERE Bedingung kann die Ergebnismenge gefiltert werden, was eine Selektion entspricht. Hierbei können bei Bedarf auch mehrere Bedingungen angegeben und entsprechend verknüpft werden (vgl. Kapitel 2.3).

```
SELECT
      User_Id,
      Username
FROM
      Users
WHERE
      User_Id > 20
ORDER BY
      Username ASC
```

Durch die Angabe von ORDER BY kann die Ergebnismenge sortiert werden. ASC sortiert aufsteigend (A-Z), DESC sortiert absteigend (Z-A).
Integer-Spalten werden numerisch sortiert, Text-Spalten werden lexikalisch sortiert.
Es können mehrere Angabe durch Komma getrennt notiert werden um bei gleicher erster Spalte weiter zu sortieren.

2.3 Vergleichsoperatoren

`Username = 'Thomas'`	Gleichheit
`Username <> 'Markus'`	Ungleich
`Username != 'Markus'`	Ungleich
`User_Id > 10`	Größer als
`User_Id >= 10`	Größer gleich
`User_Id < 10`	Kleiner als
`User_Id <= 10`	Kleiner gleich
`User_Id BETWEEN 1 AND 10`	Zwischen X und Y
`User_Id >= 1 AND User_Id <= 10`	Zwischen X und Y
`User_Id NOT BETWEEN 1 AND 10`	Nicht zwischen X und Y
`Username LIKE 'An%'`	Username beginnt mit 'An', gefolgt von beliebig vielen Zeichen, Findet: An, Anna, Anika, Anton, etc.
`Username NOT LIKE 'An%'`	
`Username LIKE 'Jo_'`	Username beginnt mit 'Jo', gefolgt von genau einem Zeichen, findet 'Joe', aber nicht 'Jonathan'

Formuliere folgende SQL-Anfragen:

Welche Ausleiher-Nr hat die Person „Anton Abel“?

Wie lautet der Nachname der Person mit der Ausleiher-Nr. 219?

Welche Buchtyp-Nummern sind vom Autor „Microsoft“ geschrieben?

Welche Bücher (Inventar-Nr) wurden im Jahr 2000 ausgeliehen?

Welche Bücher sind vom Verlag „Borland“ vorhanden?

Welche Bücher vom Verlag „Microsoft“ sind vor 1993 erschienen?

Welche Bücher enthalten im Titel das Wort „Informatik“?

2.4 Aggregat-Funktionen

Aggregat-Funktionen liefern bestimmte Informationen über eine Gruppe an Datensätzen. Sofern die Datensätze nicht nach einem Kriterium gruppiert wurden (vgl. Kapitel 2.5), beziehen sich die Funktionen auf die gesamte Ergebnismenge.

COUNT(*)	Zählt die Anzahl der Datensätze in der Ergebnismenge
MIN(<feld>)	Liefert das Minimum der Spalte <feld>
MAX(<feld>)	Liefert das Maximum der Spalte <feld>
AVG(<feld>)	Liefert den Durchschnitt der Spalte <feld>
SUM(<feld>)	Berechnet die Summe über die Werte der Spalte <feld>

Beispiele:

```
SELECT COUNT(*) AS anzahl_datensaetze FROM Tabelle
SELECT MAX(gehalt) AS hoechstes_gehalt FROM Mitarbeiter
SELECT SUM(gehalt) AS ausgaben FROM Mitarbeiter
```

2.4 Datensätze gruppieren

Mit GROUP BY lassen sich Datensätze gruppieren, welche in bestimmten Spalten dieselben Werte stehen haben. Über die restlichen Spalten lassen sich mit den Aggregat-Funktionen (vgl. Kapitel 2.3) Informationen errechnen.

```
SELECT
      Stufe,
      COUNT(*) AS Anzahl_Schueler,
      AVG(Punkte) AS Durchschnitt
FROM
      Schueler
GROUB BY
      Stufe
ORDER BY
      Stufe ASC
```

Dieses Beispiel geht von einer Tabelle Schueler aus, mit den Spalten Vorname, Nachname, Stufe und Punkte.

Der SQL-Query fasst alle Schüler einer Stufe zusammen, zählt die Anzahl der Schüler einer Stufe und berechnet den Punktedurchschnitt dieser Stufe.

Wird eine Spalte ohne Aggregat-Funktion selektiert, nach welcher nicht gruppiert wurde, so wird als Wert für diese Spalte ein zufälliger Datensatz herangezogen.
Beispiel: `SELECT Vorname, Stufe […] GROUP BY Stufe`

Formuliere folgende SQL-Anfragen:

Wie viele Schüler sind im System registriert?

Wie lautet die niedrigste Ausleiher-Nr eines beliebigen Ausleihers?

Wie lautet die höchste Ausleiher-Nr, welche einem Schüler zugeordnet ist?

Wie lautet der Name des Ausleihers mit der niedrigsten Ausleiher-Nr.?

Wie lautet der Name des Schülers mit der höchsten Ausleiher-Nr?

2.5 Produkt zweier Tabellen

Das Produkt von zwei Tabellen lässt sich mit einem CROSS JOIN bilden:

```
SELECT
      *
FROM
      tabelle1
CROSS JOIN
      tabelle2
```

```
SELECT
      a.spalte1,
      a.spalte2,
      b.spalte1,
      b.spalte2
FROM
      tabelle1 AS a
CROSS JOIN
      Tabelle2 AS b
```

MySQL erlaubt es hierbei auch, dass Wort CROSS auszulassen. Ein Produkt wird in der Praxis selten benötigt, da es keinerlei Beziehung zwischen den Daten der Tabellen herstellt.

2.6 Tabellen mit JOIN verknüpfen

In SQL lassen sich zwei Tabellen mit einem INNER JOIN verknüpfen, was einem Join in der Datenbank-Theorie entspricht. Aus diesem Grund bezeichnet man den INNER JOIN oft auch nur als JOIN.

```
SELECT
      *
FROM
      tabelle1
INNER JOIN
      Tabelle2
ON
      tabelle1.id =
      tabelle2.id
```

```
SELECT
      a.spalte1,
      a.spalte2,
      b.spalte1,
      b.spalte2
FROM
      tabelle1 AS a
INNER JOIN
      Tabelle2 AS b
ON
      a.id = b.id
```

MySQL erlaubt es hierbei auch, dass Wort INNER auszulassen. Die hinter ON notierte Bedingung legt fest, wie die Tabellen verknüpft werden.

2.7 LEFT und RIGHT JOIN

In manchen Fällen möchte man sicherstellen, dass bei einem Join aus einer Tabelle auf jeden Fall alle Datensätze in die Ergebnismenge gelangen. Hierfür gibt es den LEFT OUTER JOIN und den RIGHT OUTER JOIN.

```
SELECT
      *
FROM
      tabelle1
LEFT OUTER JOIN
      Tabelle2
ON
      tabelle1.id =
      tabelle2.id
```

Alle Datensätze aus tabelle1 gelangen in die Ergebnismenge. Finden sich keine Datensätze aus tabelle2, die dem ON-Kriterum entsprechen, werden die Felder mit NULL augefüllt.

```
SELECT
      *
FROM
      tabelle1
RIGHT OUTER JOIN
      Tabelle2
ON
      tabelle1.id =
      tabelle2.id
```

Alle Datensätze aus tabelle2 gelangen in die Ergebnismenge. Finden sich keine Datensätze aus tabelle1, die dem ON-Kriterum entsprechen, werden die Felder mit NULL augefüllt.

MySQL erlaubt es hierbei, statt LEFT OUTER JOIN und RIGHT OUTER JOIN jeweils nur LEFT JOIN und RIGHT JOIN zu schreiben.

2.8 FULL OUTER JOIN

Der FULL OUTER JOIN ist ähnlich wie RIGHT und LEFT OUTER JOIN (vgl. Kapitel 2.7). Im Gegensatz zu letzteren beiden stellt der FULL OUTER JOIN jedoch sicher, dass alle Datensätze von beiden Tabellen in die Ergebnismenge gelangen.

Finden sich zu einem Datensatz aus der einen Tabelle jeweils keine Daten aus der anderen Tabelle, welche der über ON definierten Beziehung entsprechen, so werden die restlichen Felder mit NULL aufgefüllt.

```
SELECT
      *
FROM
      tabelle1
FULL OUTER JOIN
      Tabelle2
ON
      tabelle1.id =
      tabelle2.id
```

MySQL erlaubt es, dass Wort FULL wegzulassen und stattdessen nur OUTER JOIN zu schreiben.

Ein FULL OUTER JOIN, ist der Join, welcher dem ursprünglichen Produkt von zwei Tabellen am nächsten kommt.

Formuliere folgende SQL-Anfragen:

Selektiere Name, Vorname und Ausleiher-Nr. aller Ausleiher. Sofern der Ausleiher ein Schüler ist, sollen Klasse und Tutor hinzugefügt werden.

Selektiere alle Fremdbücher mit den Informationen über ihren Ort, ihre Anzahl, Ihr Thema und ihr Erscheinungsjahr.

Erzeuge eine Ergebnismenge, welche die Informationen enthält welcher Ausleiher (Vorname, Name) welches Buch (Inventar-Nr., Titel) ausgeliehen hat. Jede Ausleihe soll einer Zeile in der Ergebnismenge entsprechen.

Erzeuge eine Ergebnismenge, welche die Informationen enthält welcher Ausleiher (Vorname, Name) welches Buch (Inventar-Nr., Titel, Autor) ausgeliehen hat.

Erzeuge eine Ergebnismenge, welche die Informationen enthält, welcher Schüler (Vorname, Name, Klasse) welches Buch (Inventar-Nr., Titel, Autor) ausgeliehen hat.