Grundlagen der Informatik

Speicherung und Verwaltung von Informationen in Datenbanken

Prof. Dr.-Ing. Thomas Wiedemann
Fachgebiet Informatik / Mathematik

Überblick zur Vorlesung

Aktueller Stand der Datenverwaltung

- Einführung in die Thematik Datenbanken
- Grundlegende Komponenten von Datenbanken
- Grundlagen von SQL und Schnittstellen
- Datenbank-Programmierung mit Access

Einführung in die Thematik Datenbanken

Speicherung von Daten im historischen Kontext :

- erste Verwaltungsprogramme organisierten die Massenspeicher (Festplatten) in Eigenregie -> sehr hoher Aufwand und stark fehleranfällig
- Einführung einer Dateiorganisation und entsprechender Betriebssystembefehle (siehe VL Dateiarbeit) vereinfachten den Zugriff auf die Massenspeicher
- Grundfunktionen zum Strukturieren, Sortieren und Suchen von Daten innerhalb der Dateien mussten jedoch immer noch durch den Entwickler programmiert werden

Logische Konsequenz und Forderung :

- in Analogie zur Übernahme der Dateiverwaltung durch das Betriebssystem sollte ein weiteres System die Funktionen zur Verwaltung der Daten standardisieren :

Geburtsstunde der "Datenbanken"

- erste Datenbanken für Masseneinsatz (BWL, Ing.) in den 60iger Jahren
- ab Mitte der 80er Jahre erste Datenbanken für die PC-Technik
- **ab Mitte der 90 er Jahre haben sich Datenbanken als primäres Speicher-medium für betriebliche Informationen durchgesetzt ! -> Motivation der VL**
- Arbeit mit Dateien nur noch aus Performancegründen oder Sicherheitsfragen !

Grundlegende Anforderungen an Datenbanken

Mehrbenutzerfähigkeit

- **zu einem Zeitpunkt müssen mehrere Benutzer in einer Datenbank Daten ändern können (vgl. Buchungssysteme, Arbeitsamt, ...)**
- Den Benutzern müssen unterschiedliche Rechte zugewiesen werden können (einfacher Sachbearbeiter, Abteilungsleiter, Leiter, Admin)

Konsistente Arbeitsweise

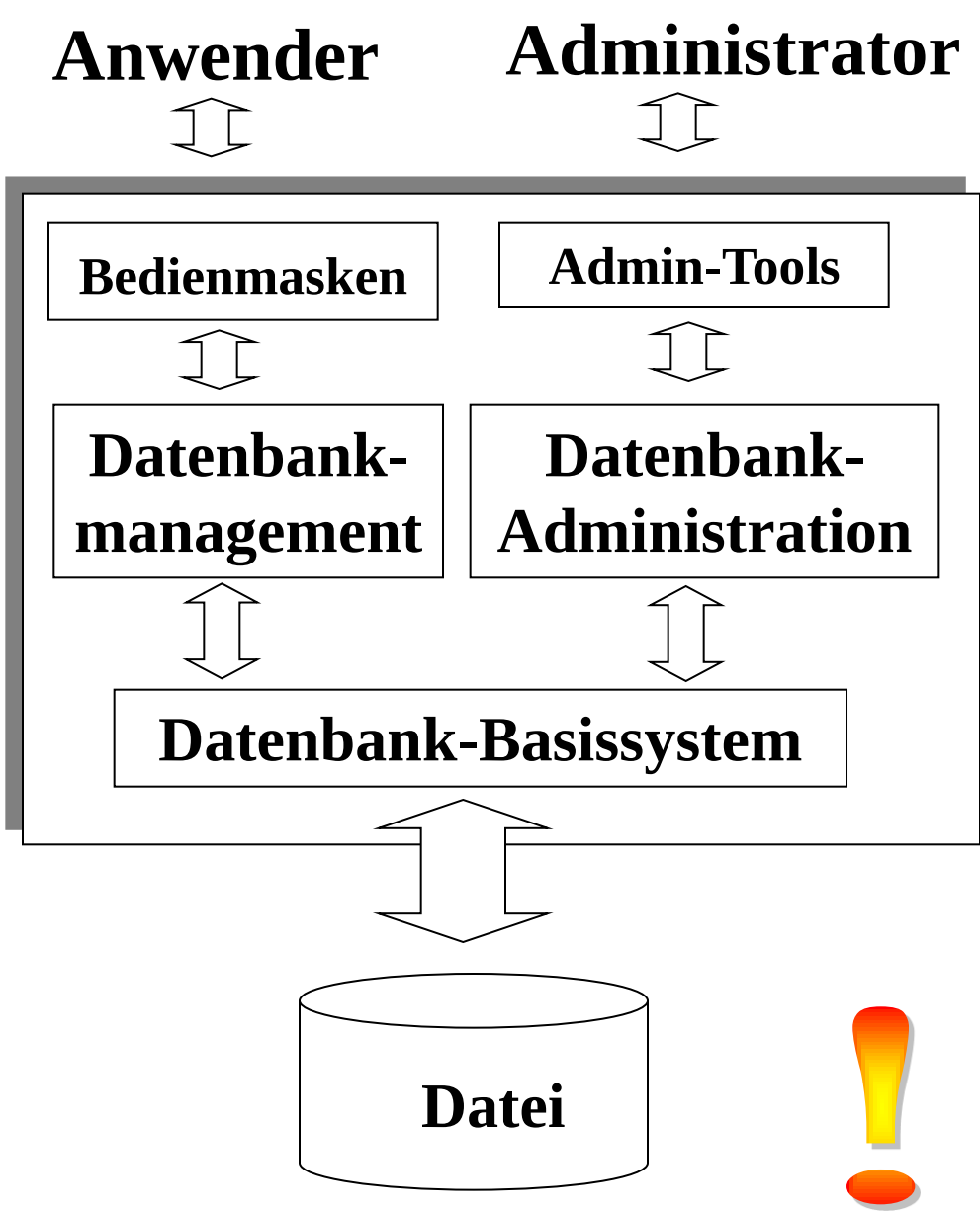
- Es muß zu JEDEM Zeitpunkt eine korrekte (konsistente) Version der Daten existieren ! Konzepte :
  - **Transaktionskonzept** : Jede Aktion wird entweder vollständig oder gar nicht durchgeführt (Gesamtrollback bei Fehlern in Einzelaktionen)
  - ausführliche Systemprotokolle (engl. logs bzw. log files) werden geschrieben, mit der Möglichkeit eines nochmaligen Ausführhrens der Aktionen bei schwerwiegenden Fehlern

Verteilte Speicherung der Daten, z.B. in globalen Konzernen

- Automatischer Abgleich von Datenbeständen
- Optionen für Betriebsabsicherung bei teilweisen (Mobilgeräte) oder vollständigen Ausfall der Kommunikationskanäle

Grundlegende Komponenten von Datenbanken

- über eine Benutzeroberfläche können Daten eingegeben und verwaltet werden
- die Daten werden durch ein Datenbankmanagementmodul verwaltet
- grundlegende Einstellungen werden mit einem Datenbank-administrationsmodul realisiert
- ein Datenbankbasissystem organisiert den Zugriff und die Ablage der Daten auf dem Massenspeicher (früher teilweise eigene Betriebssysteme, heute meist normale Dateien im Standardbetriebssystem)



Entwurf relationaler Datenbanken

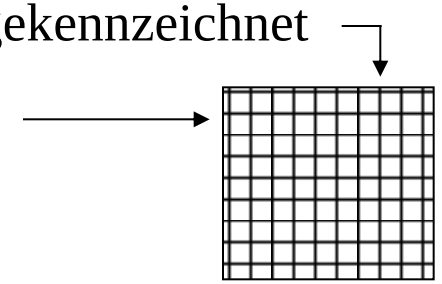
- Relationale Datenbanken haben sich aufgrund ihrer einfachen, generell in Tabellen erfolgenden Datenablage allgemein in der Praxis durchgesetzt
- für spezielle Aufgaben, welche meist durch sehr komplexe, stark heterogene Daten charakterisiert sind, werden objektorientierte Datenbanken verwendet

Hauptaufgaben beim Entwurf relationaler Datenbanken

- Zerlegung der Daten in mehrere Tabellen und Aufbau von Relationen zwischen diesen Tabellen (Normalisierung der Daten)
- Ziel ist die redundanzfreie Speicherung aller Daten, d.h. kein Datenattribut soll mehrfach in der Datenbank vorkommen (Normalformen)
- der Aufbau der Relationen wird meist durch spezielle Abfragen realisiert
- über die Abfragen werden Formularmasken oder Berichte zur komfortablen Verwaltung und zum Druck durch den Endanwender gesetzt
- optional können Programme die Datenbankinformationen verarbeiten

Die Speicherung in einer Datenbank aus Anwendersicht

- Bei den gegenwärtig am weitesten verbreiteten relationalen Datenbanken (siehe Folie 8) werden alle Daten generell in Tabellen gespeichert :
  - jede Tabelle speichert genau einen Datentyp (und kann daher als eine Art dynamische Liste von anwenderdefinierten Datenstrukturen verstanden werden)
  - die Tabellen sind generell als zweidimensionale Matrix mit einer definierten Anzahl von Spalten und einer fast beliebig großen Anzahl von Zeilen aufgebaut
  - die Spalten werden als Attribute bezeichnet und sind durch einen eindeutigen Attributnamen und einen Wertebereichsnamen (Typname) gekennzeichnet
  - die Menge aller Attribute wird in der Praxis als Datensatz bezeichnet



- die Ausprägung der Tabellen kann OHNE Programmierung im Adminmodus definiert werden :
  - wählbar sind Datentyp und Eigenschaften der Attribute
  - meist sind auch spezielle Verhaltensweisen und Standardwerte definierbar

Relationale Datenbanken

- Die Abbildung realer Daten erfordert meist verschiedene Tabellen, um Redundanzen bei der Speicherung zu vermeiden.

  Bsp.: Eine Firma hat mehrere Ansprechpartner.

  Die Adresse der Firma sollte aber trotzdem nur einmal gespeichert werden.

- Sehr häufig bestehen zwischen diesen Tabellen Beziehungen, welche als Relationen bezeichnet werden. Zum Beispiel kann eine Beziehung als "arbeitet bei" oder „enthält“ ausgedrückt werden.

- Zwischen 2 Tabellen sind folgende Relationen möglich:
  - 1:1 : –> ein Datensatz (DS) von Tabelle 1 bezieht sich auf 1 DS von Tabelle 2
  - 1:N : –> ein DS von Tabelle 1 bezieht sich auf **mehrere** DS von Tabelle 2

    Bsp.: eine Firma hat N Mitarbeiter, ein Angebot hat N Positionen
  - M:N : –> mehrere DS von Tabelle 1 beziehen sich auf **mehrere** DS von Tab. 2

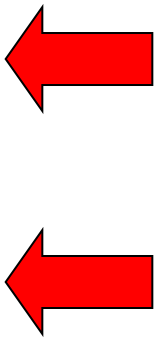
    Bsp.: M Gäste besuchen N Veranstaltungen, wobei für einen konkreten Fall auch der Fall 1:1 , 1:0 möglich ist -> gewertet wird immer der Worst Case

Aufbau von Relationen zwischen den Tabellen

Bsp.: Speicherung von Lieferantendaten :

- bei Ablage in **einer** Tabelle können die Firmendaten mehrfach auftreten

Firma	Adresse	Produkt
Siemens	**Nürnberg**	S5
Samsung	London	A7
Siemens	**Nürnberg**	S7



Nachteile:

- bei einer Änderung der Adresse bzw. einem Umzug müssten alle Datensätze durchsucht und mehrfach geändert werden
- die mehrfache Speicherung kostet unnötigen Speicherplatz

Lösung im Bsp.: Zerlegung der Daten in 2 Tabellen

Aufbau von Relationen zwischen den Tabellen II

Bsp.: Speicherung von Lieferantendaten :

- die Daten werden in eine Tabelle Firmen und eine Tabelle Produkte zerlegt :

FID	Firma	Adresse
101	**Siemens**	**Nürnberg**
102	Samsung	London
103	Hitech	Dresden

LieferID	Produkt	ProdID
101	Simatic 5	1
102	A7	2
101	Simatic 7	3

- die Relation zwischen dem Produkt und dem Lieferanten wird über dessen FirmenID (FID) durch die Datenbank hergestellt
- FID wird als Primärschlüssel (=eindeutiger Schlüsselwert) bezeichnet
- LieferID speichert diese ID in den Produktdaten und wird als Fremd-schlüssel bezeichnet (im Beispiel ist bereits eine 1:N Relation realisiert)

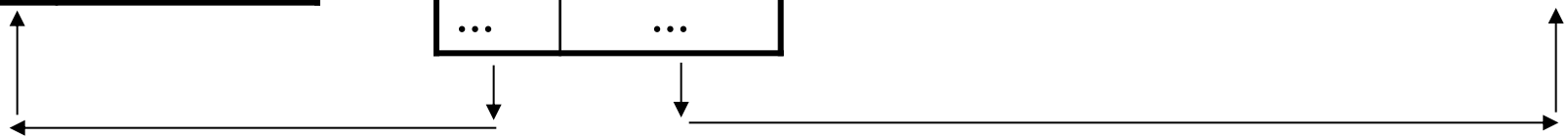
Aufbau von Relationen zwischen den Tabellen III

- 1:1 und 1:N Relationen lassen sich wie gezeigt realisieren
- problematisch sind M:N Relationen, z.B: Projekte verwenden Zuliefer-produkte, dabei kann 1 Produkt auch in N Projekten eingesetzt werden
- **Lösung:** eine Zwischentabelle baut zwischen den Projekten und Produkten die M:N-Beziehungen auf (Produkttabelle bleibt so)
- in der Zwischentabelle können beide Fremdschlüssel mehrfach auftreten

PID	Projekt
33	**P122**
34	P123-A
35	P124-C
...	

PID	ProdID
33	**2**
34	3
35	2
35	**1**
...	...

LieferID	Produkt	ProdID
101	S5	1
102	A7	2
101	S7	3
...		

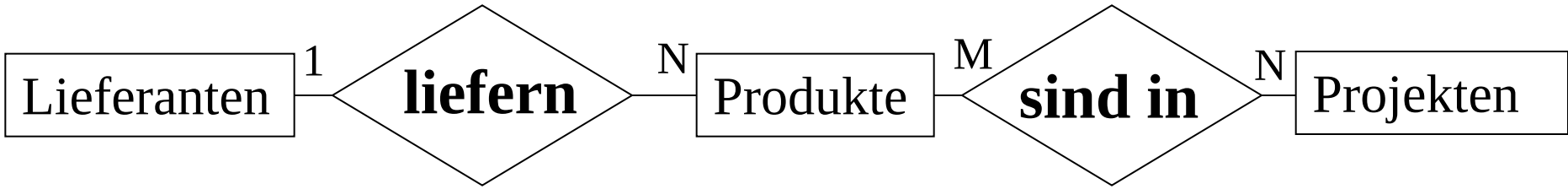


Entwurf komplexer Datenmodelle mit dem ER-Modell

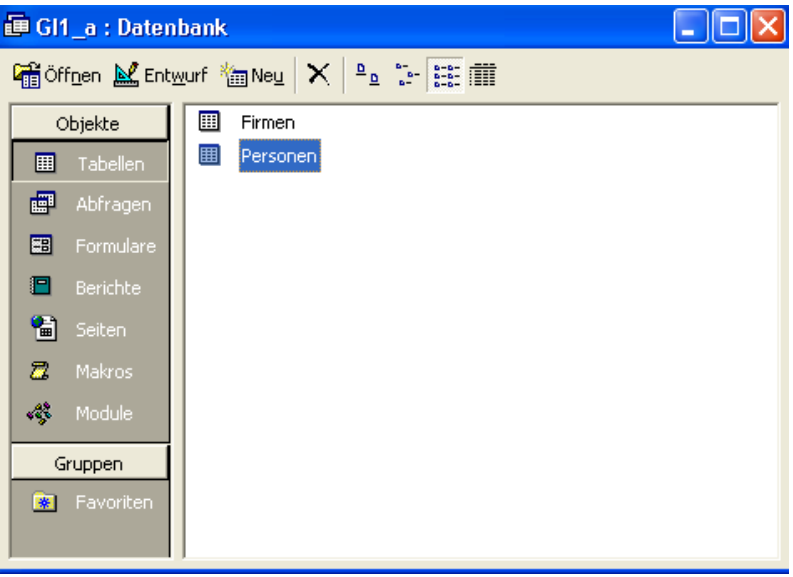
- praktische Aufgabenstellungen führen zu komplexe Verknüpfungen
- aus Performancegründen sind noch weiterführende Zerlegungen möglich (bitte unter Begriff "Normalisierung" in Fachbüchern nachschlagen)
- beim Entwurf sind auch alle zukünftigen Anforderungen zu berücksichtigen, da ein Übergang von 1:1 zu 1:N oder 1;N zu M:N sehr aufwendig ist

Sehr häufig wird das sogenannte Entity-Relationsship-Modell verwendet:

- mit den Entities werden die späteren Tabellen dargestellt (Kasten)
- Relationen werden unter Angabe der Kardinalität (=Relationstyp) als Rhombus dargestellt
- eine grafische Darstellung der Beispiele würde als ER-Modell ergeben:



Komponenten von Datenbanken unter Microsoft Access / VB



- nur die Tabellen speichern die Daten (sonst keine anderen Komponenten)
- Abfragen arbeiten über Tabellen und filtern, sortieren und verknüpfen diese
- mit den Formularen werden Bedien-oberflächen für Anwender ohne EDV-Kenntnisse entwickelt
- Berichte und Seiten drucken Daten

- Makros erlauben eine sehr einfache, lineare Art von Befehlsfolgen und werden für die Verknüpfung von Formularen verwendet (falls, vorhanden, wird das Makro „autoexec“ beim Start von Microsoft-Access automatisch gestartet)
- in den Modulen werden größere Visual Basic-Programme abgelegt

Datenbankabfragen

Über den Datenbanktabellen werden sogenannte Abfragen definiert:

- dies sind Verarbeitungsdefinitionen, welche wiederum eine Tabelle als Ergebnis haben

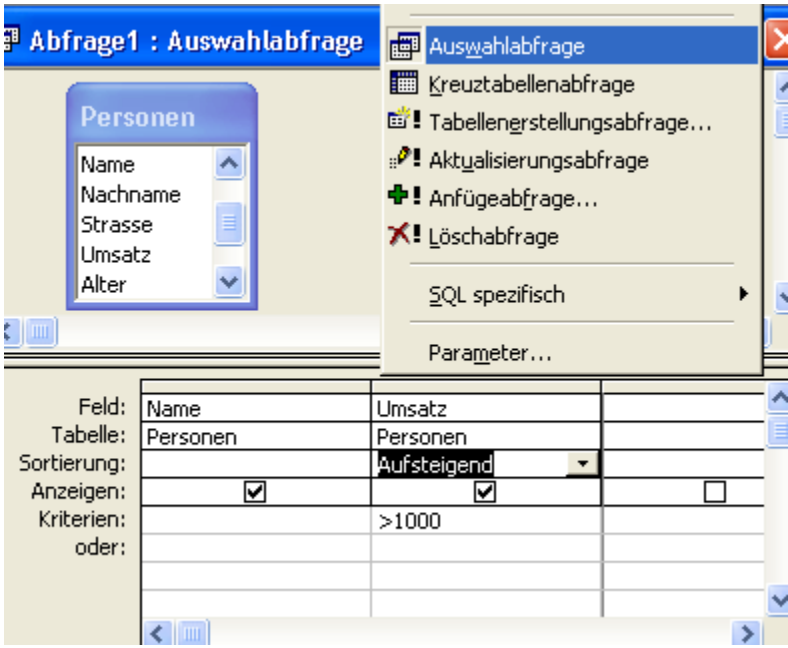
Möglich sind folgende Verarbeitungsdefinitionen:

- Auswahl (Selektion) von bestimmten Spaltenattributen
- mehrfaches Sortieren nach Attributen
- Kombiniertes Filtern nach Attributen
- Modifizieren, Löschen und Anfügen von Datensätzen auch in Kombination mit den vorher genannten Anzeigeoperationen

- Abfragen könnten geschachtelt und miteinander kombiniert werden
- in einer Abfrage kann gleichzeitig gefiltert, sortiert und selektiert werden
- die Operationen entsprechen den **Mengenoperationen** aus der Mathematik

Access – Datenbankabfragen

In Access sind (in der Relevanz ihrer Bedeutung) folgende Abfragetypen definiert :



- **Auswahlabfragen** zeigen Daten an und erlauben die komplexe Verknüpfungen, dabei können als Datenquelle auch wiederum Abfragen verwendet werden
- **Aktualisierungsabfragen** **ändern Daten** in beliebig großen Tabellen
- **Anfügeabfragen** hängen Daten einer Tabelle an eine zweite Tabelle und sind für Importe sinnvoll
- **Löschabfragen** löschen Daten, Kreuztabellen erlauben statistische Auswertungen
- **SQL-spezifische Abfragen** dienen zur Kombination aller Abfragen und zur Definition beliebiger Datenbankoperation mit der Datenbanksprache SQL
- **Parameter** können zur Anpassung aller Abfragen an wechselnde Aufgabenstellungen eingesetzt werden (z.B. Parameter = Umsatzgrenze ...)

Anbindung der Access-Formulare an die Datenbank

- Die bisher gezeigten Formulare waren „Stand-alone“-Formulare, d.h. OHNE Datenbankanbindung

Zur Anbindung einer Datenbank sind folgende Schritte auszuführen :

- In der Eigenschaften des gesamten Formulars (links oben in die Ecke klicken) ist unter der Eigenschaft „**Datenherkunft**“ eine Abfrage oder Tabelle anzugeben
- danach muß in jedem Formularelement mit Datenanzeigemöglichkeit in der Eigenschaft „**Steuerelementinhalt**“ das gewünschte Feld aus der Datenquelle ausgewählt werden :
  - ein mehrfaches Verwenden ist zulässig
  - Bei sehr großen Formular kann diese Arbeit auch sehr effizient von Assistenten erledigt werden (siehe Formularassistenten)
  - Der Name „**Ungebunden**“ verweist auf ein NICHT an die Datenbank gebundenes Feld !!
- Mit den Navigationselementen von Formularen kann zwischen den Datensätzen gewechselt werden.

Einführung in SQL

Die Verwaltung von Daten in einer Datenbank kann auf 2 Arten erfolgen :

- Mit einem **Datenbankmanagementsystem** (wie z.B. Access) können ohne Programmierung Daten über Masken, Tabellen oder Webformulare (z.B. Tool phpmyadmin) eingegeben, geändert und teilweise auch deren Definition in der DB geändert werden
- Für den universellen Einsatz in Programmiersprachen wird ein möglichst standardisierter Zugang zu ALLEN Funktionen ALLER verfügbaren Datenbanken auf Basis einer Programmiersprache benötigt
  - Mit SQL existiert eine solche weitgehend standardisierte Sprache.
  - Fast jede Datenbank kann teilweise oder ganz den SQL-Sprachumfang verstehen und ausführen.
  - Das aufrufende Programm muss „nur“ die SQL-Befehle als Textkommando zusammen bauen und an eine Datenbankschnittstelle senden – die Datenbank analysiert die Befehle und gibt die Ergebnisse (i.d.R. eine Tabelle oder auch ein Einzelwerte) an das Programm zurück

Tip: Komplexe SQL-Befehle lassen sich günstig über die Tools erzeugen.

Einführung in SQL

- SQL steht für **Structured Query Language** - zu deutsch etwa: **strukturierte Abfragesprache**.

- SQL ist eine **nichtprozedurale Sprache**, d.h. im Gegensatz zu C oder ähnlichen Sprachen beschreibt SQL , **welche Daten** abzurufen, zu löschen oder einzufügen sind und nicht, wie das zu geschehen hat.
- SQL kann in vier Funktionsgruppen unterteilt werden :
  - SQL selbst als Sprache zur Datenabfrage (engl. Query)
  - DML - Data Manipulation Language zur Datenmanipulation (Einfügen, Ändern, Löschen von Daten)
  - DDL - SQL Data Definition Language zur Definition von Tabellen und Abfragen
  - DCL - SQL Data Control Language zur Vergabe von Rechten an Nutzer (spezifische Berechtigung zu den Funktionen)

Historie von SQL

- Entwicklung Ende der 70er Jahre bei IBM-Labor in San Jose (Kalifornien)
- ursprünglich für IBM-Produkt DB2 geplant (DB2 auch heute noch existent)
- Im Gegensatz zu früheren Ansätzen verfolgt SQL einen mengenorientierten Ansatz, d.h. die Operationen der Sprache basieren auf Mengenoperationen (auch mit mathematischer Grundlage) . Damit besteht eine gute theoretische Basis, erstellt durch Arbeiten von Edgar F. Codd mit seiner Forschungsarbeit am IBM Almaden Research Center in den 60 er und 70 iger Jahren .
- Der Erfolg von SQL führte zu einer starken Verbreitung und einer Standardisierung durch das ANSI (American National Standards Institute) und die ISO (International Standards Organization). (z.B. ANSI 92 – SQL – Standard, ANSI 2002 ##)
- **Resultierende Problem(e) :**
  - Leider sind die Standards (wie oft in der Informatik) sehr umfangreich und werden von Datenbankherstellern nicht zu 100% umgesetzt !
  - Damit besteht trotz SQL leider keine absolute Kompatibilität zwischen den Datenbanken (und wird wahrscheinlich von den Herstellern auch nicht gewünscht)

Funktionsweise von SQL

- das Programm generiert (mit Textoperationen) einen Kommandotext und sendet diesen an das Datenbanksystem (häufig als DB-Engine bezeichnet) => siehe Beispiele auf den Folgeseiten

- die DB-Engine führt den Befehl aus

- Ergebnisse werden über Pointer oder spezielle Strukturen (Arrays, dyn. Listen oder Datenbankobjekte (db / rec in den Programmierbeispielen) zurück übergeben

Abfrage von Datenbanken mit dem Select-Befehl

- **Abfrage einer Tabelle mit dem Select-Befehl**

Syntax: "Select * from Produkte where Preis > 100 order by PName"

- Select gibt den Wunsch nach einer reinen Anzeige an
- mit dem * werden alle Spalten angezeigt (sonst Spalten aufzählen mit , , , ,)
- der from-Abschnitt definiert die Herkunftstabelle(n),
- der where-Abschnitt definiert einen Filter (hier alle Preise > 100 €),
- der order by -Abschnitt definiert eine Sortierung (hier nach Produktname)
- Es können bei where und order by auch mehrere Angaben erfolgen.
- Als Minimalversion reicht : select * from table

Typische Anwendungen

- Select Vorname, Nachname from ... - Spaltenselektion
- Select ... From Personen Where PID>100 and Alter < 20 - Filter

Abfrage von Datenbanken mit dem Select-Befehl - II

- **Vergleich von Texteinträgen mit dem like – Operator oder =**

unter Access :

- Select ... Where Nachname like “*mann*“ - sucht alle Namen mit mann
- ... Where Nachname like “*mann“ - sucht alle Namen mit mann am Ende
- ... Where Nachname like “*m?nn“ - sucht alle Namen mit Muster m*nn

Achtung – in anderen Datenbanken sind die Platzhalter ggf. anders – **MySQL** :

- Where Nachname like “%mann“ - sucht alle Namen mit mann am Ende
- ... Where Nachname like “%m_nn“ - sucht alle Namen mit Muster m*nn

Filter über Mengen

- Where Alter >16 **And** Alter < 30
- Where Alter **between** 16 And 30 - analog zu erster Option
- Where Alter **in** (16, 18, 20) - nur die angegebenen Altersgruppen

Abfrage von Datenbanken mit dem Select-Befehl - III

Sortieroptionen

- Select ... Order by Nachname, Vorname – Sortierung immer in der Reihenfolge der Felder von links nach rechts
- ... Order by Alter, PLZ ergibt andere Sort. Als ... Order by Alter, PLZ
- Sortierrichtung von ASC (Ascending – aufsteigend = Default) oder Desc
- ... Order by Alter Desc, PLZ Asc - zuerst nach Alter absteigend, dann bei gleichem Alter nach PLZ aufsteigend

Alias – Namen für Spalten (Um- oder Neubenennung von Spalten)

- SELECT feldName1 "Nachname", feldName2 "Vorname" ...
  - Alias-Vergabe ist sehr hilfreich bei Anpassung verschiedener DB mit unterschiedlichen Namensräumen, ohne daß die Namen in den Ursprungstabellen wirklich geändert werden müssen
- Select [Lagerbestand]*[Preis] AS Lagerwert - auch mit vorheriger Berechnung

Abfrage von Datenbanken mit dem Select-Befehl - 4

- **Verknüpfung von mehreren Tabellen mit JOIN :**

```
Select * FROM Autor INNER JOIN medien ON Autor.UID = medien.AutorID;
```

- Der Join verbindet zwei Tabellen über die mit ON-angegebenen Felder (i.d.R Primär- und Fremdschlüssel) – alle Filter/Sortierungen gelten analog
- Bei inner join müssen auf beiden Seiten die entsprechenden Schlüsselwerte vorhanden sein.
- Mit der zusätzlichen Option left join / right join können auch jeweils auf einer Seite Leermengen akzeptiert werden (Vorsicht dann beim Programmieren bei Zugriff auf diese mit NULL belegten Felder)
- Es können auch mehr als 2 Tabellen miteinander verbunden werden – das Verhalten der Datenbanken bzgl. des Schreibens in die Resultatmenge ist dabei aber leider unterschiedlich - Empfehlung : Joins suksessive aufbauen, auch zum Austesten der Ergebnisse

Abfrage von Datenbanken mit dem Select-Befehl - 5

Aggregatfunktionen (Mengenbasierte Rechenfunktionen)

- SELECT COUNT(*) "Anzahl" FROM Autor – liefert Autor-Anzahl
- Als Rechenoperationen stehen COUNT() (Anzahl), MIN(), MAX(), AVG() (Durchschnitt) und SUM() zur Verfügung.
- Mit dem Group by-Befehl können Datensätze zusammengefasst werden :

  SELECT AutorID, **Sum**(Lagerwert) AS [Summe von Lagerwert] ...

  GROUP BY Medien_Autoren.AutorID;

- Mit dem Operator Distinct können auch bei einfachen Tabellen gleiche Datensätze zusammengefasst werden.
- Select **Distinct** Nachname ... - jeden Nachnamen nur einmal

Manipulation von Daten

Einfügen von Daten

INSERT INTO Autor (Nr, Nachname, Vorname, GebJahr)
VALUES (1, 'Böll', 'Heinrich', 1917);
- die genaue Schreibweise ist wieder DB-abhängig (falls ein Feld nicht gesetzt werden soll – NULL angeben)

Ändern von Daten

- **UPDATE** Autor **SET** Vorn ame = Otto, GebJahr = 1954, Beruf = NULL **WHERE** Nr = 10;
  Achtung : Wichtig ist i.d.R. der Filter, da sonst ALLE Datensätze geändert werden !!!

Daten löschen

- **DELETE FROM** Autor **WHERE** Geburtsdatum < (SYSDATE - 36500);
- Hier Löschen von (veralteten) Datensätzen

Typische Datenbank-Programmierung mit SQL unter VB

- unter Anwendung der SQL-Befehle und einiger Grundbefehle lassen sich Datenbanken relativ einfachen durchlaufen und bearbeiten
- Für den Zugriff werden nachfolgende Objekte definiert

  Database = Zugriffsobjekt auf die gesamte Datenbank und

  Recordset = Zugriff auf einzelne Tabelle oder Abfrage verwendet

  Dim db as database , rec as recordset, sql$: Rem db, rec sind Variablen

  set Db = Currentdb() : Rem Erzeugt Verweis auf aktuelle DB

- mit normalen Textfunktionen wird die Zugriffsquelle bestimmt !
- der Filterwert (hier Umsatz) wird meist durch die Benutzeroberfläche oder einen Browserrequest an den Webserver bestimmt !

  sql= “select * from Personen where Umsatz > “ & Umsatzvorgabe

  set rec = db.openrecordset(sql) : Rem öffnet Abfrage zu Umsatz

Typische Datenbank-Programmierung mit SQL unter VB

- unter der Voraussetzung, das das Öffnen der Abfrage fehlerfrei erfolgte, können dann alle Datensätze sequentiell durchlaufen werden :

```
Set rec = db.openrecordset( sq ) : Rem öffnet Abfrage zu Umsatz
Dim Name1, Umsatzgruppe%
Do until rec.eof
    Name1 = rec!Vorname & " " & rec!Nachname
     if rec!Umsatz >= 1000 then Umsatzgruppe = 1
     if rec!Umsatz >= 10000 then Umsatzgruppe = 2
     if rec!Umsatz >= 100000 then Umsatzgruppe = 3
     generiere_report( Name1 & " hat Umsatz " & Umsatzgruppe )
  rec.movenext : Rem Schaltet weiter zum nächsten Datensatz
loop
rec.close : db.close : Rem Alles wieder schliessen !
```

Typische Datenbank-Programmierung mit SQL unter VB

Für das Bewegen (Navigieren) in Datenbankabfragen stehen folgende Befehle zur Verfügung :

movenext - geht zum nächsten Datensatz

movePrevious - geht zum vorhergehenden Datensatz

movefirst - geht zum ersten Datensatz

moveLast - geht zum letzten Datensatz

Achtung: Nach jeder dieser Befehle ist auf das Erreichen des Endes bzw. Anfangs der Daten zu prüfen – ein erneutes Bewegen in gleicher Richtung führt sonst zu Laufzeitfehlern !

Typische Datenbank-Programmierung mit SQL unter VB

Zum Ändern von Daten muß der entsprechende Datensatz mit den Befehlen edit geöffnet oder mit addnew angelegt werden und nach Ende aller Änderungen oder Eingaben mit update gespeichert werden :

```
rec.addnew : Rem Start Neueingabe eines DS
   rec!Vorname = "Jan" : rec!Nachname = "Mustermann": rec!Umsatz = 0
rec.update
Do until rec.eof                    : Rem Ändere ALLE Datensätze
    rec.edit
          rec!Umsatz = rec!Umsatz + NeuerMonatsumsatz
    rec.update : Rem Speichert die neuen Wert in DB
    rec.movenext : Rem Schaltet weiter zum nächsten Datensatz
Loop:
rec.close : db.close : Rem Alles wieder schliessen !
```

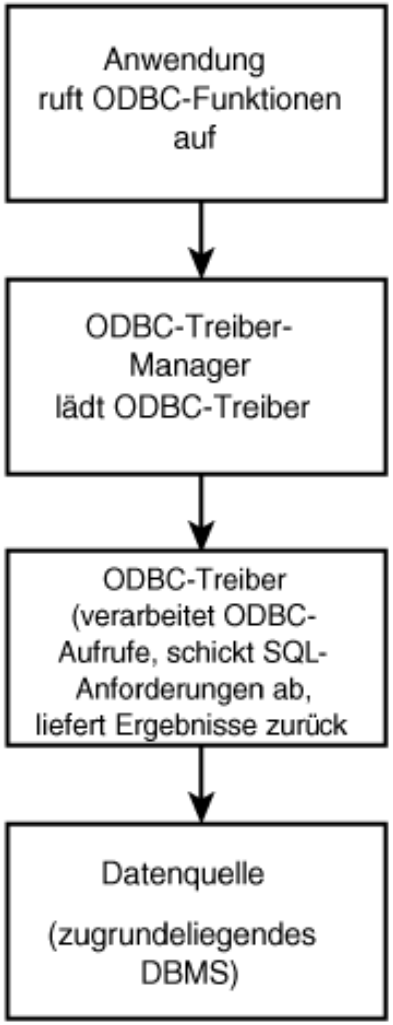
Weitere Begriffe und Technologien in der Datenbanktechnik

- **Die nachfolgenden Begriff werden nicht von allen DB unterstützt, sondern meist nur von den größeren (und teueren) DB-Systemen.**
- Stored Procedures
  - immer wiederkehrende Abfragen können in der DB in einem internen Format gespeichert werden und sind durch die entfallende Syntaxanalyse schneller bei der Ausführung
  - Zusätzlich ist auch die Manipulationsgefahr (Hacker) geringer (-> Web)
- Trigger
  - Es können Bedingungen (Trigger) auf den Daten definiert werden, bei denen automatisch eine Datenbankaktion ausgelöst wird (z.B. Neuanlage eines Kunden, oder Eintreten von Fehlerzuständen ...)
- Transaktionskontrolle
  - Der Befehl **COMMIT** startet eine Transaktion
  - Alle DML-Befehle (INSERT, UPDATE, DELETE) werden erst beim nächsten **COMMIT** wirklich ausgeführt.
  - Mit **ROLLBACK** kann der gesamte Vorgang rückgängig gemacht werden.

Weitere Begriffe und Technologien in der Datenbanktechnik - 2

Spezielle Datentypen

- **BLOB – Binary Large Objects (dt.: Binäre große Objekte)**
  - zur Speicherung von größeren Datenmengen, z.B. Multimediadaten (Fotos, Musikstücke, verschlüsselte Daten, auch XML-Daten)
  - DB eignen sich jedoch (noch) nicht zur allgemeinen Ablage großer Binärdaten – sinnvoll ist immer eine Verarbeitung (z.B. internes Auslesen der Daten im aufrufenden Programm)
  - Alternativ kann auch ein Pfadname als Text in der DB gespeichert werden und dann extern die Anzeige und Verarbeitung erfolgen
  - immer wiederkehrende Abfragen können in der DB in einem internen Format
- **Datetime, Timestamp**
  - Zur Ablage von Datumsangaben und Zeitstempeln (z.B. letzte Änderung)
  - einige DB erfordern für Änderungen immer einen Zeitstempel
- **Mengen**
  - Enum - Aufzählung von eineindeutigen Werten (“Mo“, “Di“, ..)
  - Set -String-Objekt mit max. 64 Elementen



Datenbankschnittstellen - ODBC und JDBC

- Zur universellen Anbindung von DB unter Windows wurde von Microsoft der ODBC-(Open Database Connectivity Standard eingeführt.
- Es gibt ODBC-Treiber für nahezu jede Datenbank.
- **Vorteil:** Datenbanken sind austauschbar, ohne die Applikation umschreiben zu müssen (falls Standard-SQL verwendet, ANSI 89).
- **Nachteil:** Performance geringer als bei datenbank-spezifischen APIs
- In ähnlicher Weise bieten JDBC – Treiber eine universelle Schnittstelle zu Java-Programmen an.

Marktübersicht zu Datenbanksystemen

Kommerzielle Datenbanken (meist auch eingeschränkte Freeware-Versionen)

- **Oracle (Marktführer) - Oracle Database 11g**
  - sehr komplexes System, Preise bis zu 20.000 €
  - Führend bzgl. Funktionalität und Performance, harter Kampf gegen Microsoft
  - Neuorientierung auf Produktions-Planungs-Systeme (ERP) als Konkurrenz zu SAP
- **Microsoft** - Access und Microsoft SQL Server
  - MS Access als kleine Bürodatenbank mit bis zu 100.000 Datensätze, Entwicklungsoberfläche aber sehr effizient
  - bei größeren Datenvolumen Migration auf MS SQL Server sinnvoll
  - MS SQL Server ca. 3000 –6000 € , sehr günstig im Paket (Small Business Server) für Windows-Applikationen
- **IBM**
  - DB2 – ebenfalls sehr leistungsfähiges System
  - Weiterhin bietet IBM mit Informix eine weitere, kommerzielle Datenbank an.

Marktübersicht zu Datenbanksystemen

Freie / Kostenlose Datenbanken

- **MySQL - MySQL 5.1 - jetzt MariaDB benannt**
  - freie DB der schwedischen Firma MySQL AB , **www.mysql.com**
  - optimiert für Einsatz als Webdatenbank (sehr viele kleine Zugriffe)
  - im Januar 2008 von Sun übernommen
- **PostgreSQL 7.2 ... 8.0**
  - freie DB
  - schnell bei vielen gleichzeitigen Benutzern und komplexeren Operationen;
  - Verarbeitung geographischer Daten (GIS)
- **MaxDB**
  - Nachfolger der SAP® DB (und damit auch von Adabas) und prädestiniert für ERP-Anwendungen.
  - auch gut geeignet für andere komplexe Anwendungen im kommerziellen Umfeld; z.B. Data Mining,

Zusammenfassung zu Datenbanksystemen

- **Fast alle professionellen IT-ysteme speichern ihre Daten in Datenbanken**
  - Die größte betriebswirtschaftliche Anwendung SAP speichert die betrieblichen Daten Tausenden verknüpften Tabellen
  - Auch alle Web-Anwendungen legen ihre Daten in Datenbanken an (meist MariaDB/ MySQL)

- **Mit Datenbanken werden die Daten**
  - Redundanzfrei und konsistent gespeichert (auch bei Stromausfällen oder Abstürzen)
  - durch Mehrbenutzerzugriff und verteilte DB lassen sich globale IT-Systeme aufbauen, welche Tausende von Nutzer gleichzeitig bedienen können