Algorithmen und Datenstrukturen

Praktische Einführung und Programmierung

Stefan Bosse

Universität Koblenz - FB Informatik

Datenbanken

- Datenstrukturen
- Algorithmen
- Anwendungen

Motivation

Datenbanken sind eine wichtige "Anwendung" für Algorithmen und Datenstrukturen.

- Viele Algorithmen und Datenstrukturen aus dem Kurs finden sich in Datenbankprogrammen wieder
  - Extern sichtbar für die Datenorganisation durch/über die Nutzer
  - Intern unsichtbar aber elementar wichtig für die extern sichtbare Datenorganisation

Eine Datenbank zentralisiert Datenspeicherung und Datenorganisation (wenn auch DBs selbst verteilt sein können)!

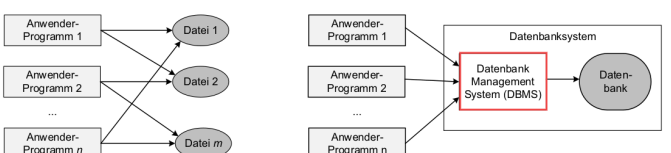


Abb. 1. Links: Mehrere Anwendungen verwalten ihre Daten selbst in Dateien. Rechts: Die Daten aller Anwendungen werden in einer gemeinsamen Datenbank von einem Datenbankmanagement-System verwaltet.

Motivation

Die wichtigsten Vorteile von Datenbankmanagement-Systemen im Vergleich zu einer dateiorientierten Strategie sind:

1. Die Daten werden in der Regel unabhängig von den Anwendungen gespeichert, die diese verwenden.
    - Die Daten sollten damit möglichst unabhängig von der verwendeten Programmiersprache sein, da sich diese im Laufe der Zeit eventuell ändert.
    - Die Anwendung kann durch eine neuere Software mit größerer Operationalität und Performant ersetzt oder ergänzt werden, dieselben Daten können aber langfristig weiter verwendet werden.
2. Die Daten werden in der Regel nur einmal zentral verwaltet und nicht redundant an mehreren unabhängigen Stellen. Damit sinkt das Risiko von Inkonsistenzen (mehrere unterschiedlich geänderte Kopien derselben Daten). Zusätzlich wird dadurch weniger Speicherplatz benötigt.

Motivation

3. Die Daten werden nicht nur persistent gespeichert, das Datenbankmanagement-System (DBMS) sorgt für die Konsistenz der Daten, auch wenn die Anwendung mitten in einer Operation abstürzt. Auch wenn die Datenbank zerstört ist, kann sie mithilfe von Backup und Recovery wieder hergestellt werden.

4. Das DBMS ermöglicht, dass viele Anwendungen bzw. Benutzer quasi gleichzeitig lesend und auch schreibend zugreifen können. Hierzu werden Konzepte wie **Transaktionen** angeboten.

5. Das DBMS kontrolliert die Zugriffe auf die Daten und sorgt dafür, dass definiert werden kann, welcher Nutzer bzw. welche Anwendung welche Daten ansehen oder ändern darf.

DBMS: Datenbankmanagement-System, die Softwareschnittstelle und das Ausführungsprogramm

Motivation

6. Die Programmierung der Anwendungen wird vereinfacht, da das DBMS das Anlegen, Ändern, Löschen und Suchen von Daten implementiert und optimiert. Die Implementierung der Operationen ist nicht festgelegt und kann geändert werden (z.B. Bäume statt Listen verwenden)
    - Über die Sektoren der Festplatte und die Speicherseiten des Betriebssystems zerbrechen sich jetzt die Entwickler des Herstellers des DBMS den Kopf. Die Benutzer und Entwickler der Anwendungen kennen nur eine konzeptuelle Sicht auf die Daten in Form von Knoten eines Graphen, von Zeilen in Tabellen oder von Schlüssel-Wert-Paaren.

Daten und Datenstrukturen

je nach Datenbankmodell können verschiedene Daten und Datenstrukturen gespeichert und verarbeitet werden:

1. **Name / Wert-Paare** dienen dazu Daten wiederzufinden. Dazu müssen diese eindeutig identifizierbar sein. Im einfachsten Fall ist das ein eindeutiger Name (wie die Pfad- und Dateinamen in einem Dateisystem). Das kann auch ein aus den Daten selbst berechneter Hash-Wert sein. Damit wäre jede Entität bzw. jedes Objekt jeweils ein Wert. Als identifizierender Name kann eventuell eines der Attribute verwendet werden.

2. Die **Tabelle** ist eine sehr häufige Darstellungsform von Daten, besonders im Bereich betriebswirtschaftlicher Anwendungen. Aus einem Entitätstyp bzw. einer Klasse wird im einfachsten Fall eine Tabelle. Aus jedem Attribut wird eine Spalte. Objekte bzw. Entitäten sind jeweils die Zeilen. Tabellen sind die Grundlage der **relationalen Datenbankmanagement-Systeme**.

3. Die **Baumstruktur**: Von einer Wurzel aus sind die anderen Knoten erreichbar. Aus einem Entitätstyp bzw. einer Klasse wird eine Menge von Bäumen. Ein Objekt bzw. eine Entität wird als Baum dargestellt werden: Jedes Attribut ist ein innerer Knoten oder ein Blatt. Baumstrukturen bieten verbesserte Sucheigenschaften.

Daten und Datenstrukturen

4. Ein **Gerichteter Graph**: Wenn die Objekte bzw. Entitäten sich untereinander häufig referenzieren, kann dies gut als gerichteter Graph dargestellt werden mit den Objekten bzw. Entitäten als Knoten und den jeweiligen Referenzen als (annotierte) Kanten.

5. **Zeitreihe** und zeitabhängige Daten: Ein Sensor misst eine physikalische Größe, zu speichern sind Zeitpunkt und Messwert. Das wäre ein Objekt bzw. eine Entität. Eventuell kommen noch Metadaten dazu wie die Geoposition und die physikalische Einheit in der der Sensor misst. Regelmäßiges Messen erzeugt so eine Zeitreihe.

6. **(Text) Dokumente**: (Internet-) Anwendungen erzeugen teilweise große Mengen von mehr oder weniger strukturierten Textdokumenten, auch die Log-Ausgaben zählen dazu. Damit wäre ein Textdokument, z. B. ein Text über einen Mitarbeiter ein Objekt bzw. die Entität. Möglicherweise gibt es dazu eine Klasse, die Strukturvorgaben an das Dokument macht.

Daten und Datenstrukturen

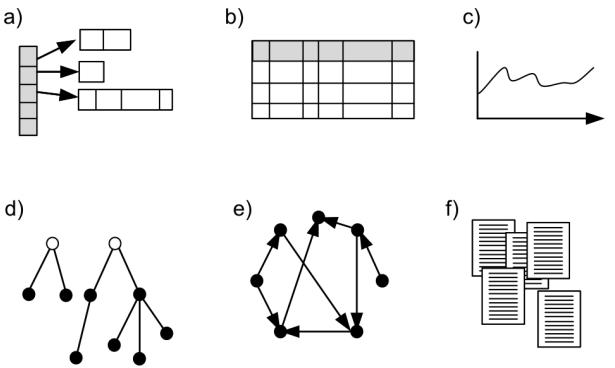


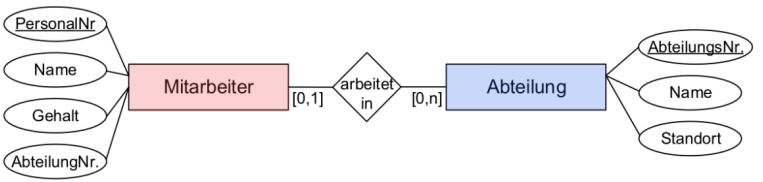
Abb. 2. Überblick über verschiedene Möglichkeiten Daten darzustellen. a) als Menge von Name-Wert Paaren, b) als Tabelle oder in mehreren Tabellen, c) als Zeitreihe, d) als Menge von Bäumen, e) als ungerichteten oder gerichteten annotierten Graphen oder f) als Menge von (Text-) Dokumenten

Zugriffsmuster auf gespeicherte Daten

Was sind nun Kriterien zur Wahl einer bestimmten Repräsentation der Daten als Baum, Graph oder Tabelle?

Darstellung als Tabellen

- Ein Entitätstype besteht aus einer Klasse mit Attributen
- Im einfachsten Fall wird aus jedem Entitätstyp eine Tabelle modelliert
- Die Attribute werden jeweils zu Spalten. Ein identifizierendes Attribut ist dabei der Primärschlüssel der Tabelle.



Bsp. 1. Entity-Relationship-Modell der Relationen Mitarbeiter und Abteilung

- Die Beziehung zwischen Mitarbeiter und Abteilung ist 1 zu N,
  - ein Mitarbeiter ist genau in einer Abteilung und in einer Abteilung können beliebig viele Mitarbeiter sein.
- Dies wird über einen Fremdschlüssel dargestellt, die Tabelle Mitarbeiter hat eine Spalte mit Abt. Nr. Werten, diese sind Primärschlüssel in der Abteilung-Tabelle.

Dies war ein Beispiel für die Übersetzung eine logischen Datenmodells in Form eine ER-Modells bzw. Klassenmodells auf ein physisches Datenmodell in Form von Tabellen.

[kap16db]

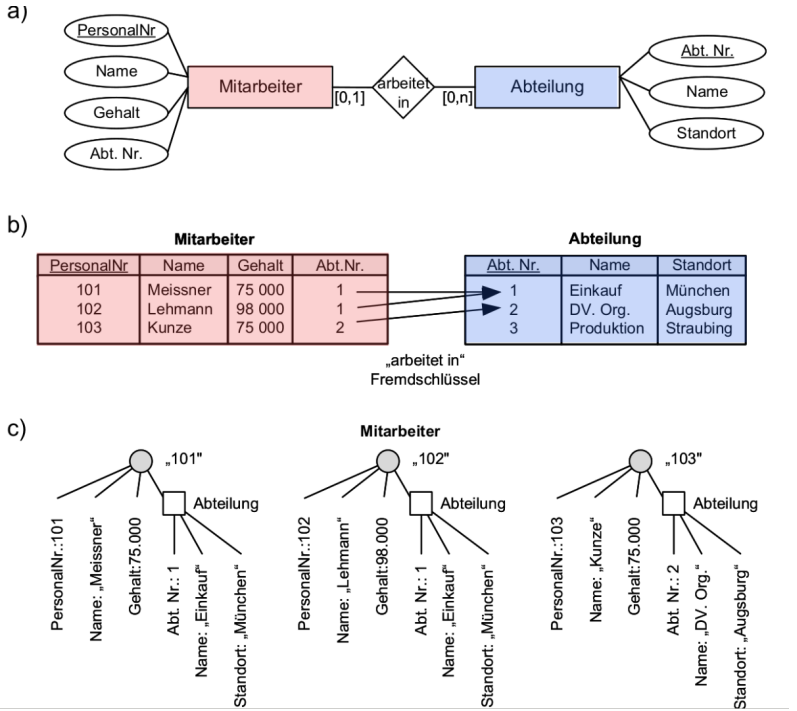


Abb. 3. Abbildung des ER-Modells auf Tabellen und auf eine Baumstruktur

Relationale Datenbankmanagement-Systeme

- Das relationale Datenbankmodell und darauf aufbauende relationale Datenbankmanagement-Systeme (RDBMS) gehen auf eine in 1970 veröffentlichte Arbeit von E. F. Codd zurück.
  - Codd legte damit das mathematische Fundament in Form der relationalen Algebra
- Unter einer Relation versteht man im Zusammenhang mit dem relationalen Datenmodell eine logische Zusammenfassung von Informationen in einer Form, die in etwa einer Tabelle vergleichbar ist.
  - Gemeint ist auch eine Relation im mathematischen Sinne, also eine Menge von Tupeln.
- Zur eindeutigen Identifikation aller Zeilen der Relation sollte diese einen Primärschlüssel (Pri-mary Key) besitzen (vorhandenes Attribut oder erzeugte Nummer ID).
  - Um den Zugriff zu beschleunigen, kann man neben dem Primärschlüssel weitere eindeutige Schlüssel einführen, sog. Zweitschlüssel (Secondary Keys).

Relationen

Mitarbeiter

PersonalNr	Name	Gehalt	AbteilungsNr
101	Meissner	75 000	1
102	Lehmann	98 000	1
103	Kunze	75 000	2

Abteilungen

AbteilungsNr	Name	Standort
1	Einkauf	München
2	DV-Org.	Augsburg
3	Produktion	Straubing
4	Entwicklung	Augsburg
5	Verwaltung	München

Abb. 4. Die Relation Mitarbeiter hat vier Attribute (Namen PersonalNr , Name , Gehalt und AbteilungsNr) drei Zeilen (also insgesamt drei 4-Tupel). Die Relation Abteilung hat drei Attribute (AbteilungsNr , Name und Standort) sowie vier Zeilen (also insgesamt fünf 3-Tupel). Die (Primär)Schlüssel der beiden Tabellen sind jeweils für jede Zeile eindeutig: PersonalNr ist Schlüssel der Relation Mitarbeiter und AbteilungsNr ist Schlüssel von Abteilung.

Beziehungen (Relationships)

- Relationen und ihre Tupel bzw. Tabellen und ihre Zeilen können nicht isoliert betrachtet werden.
- Relationen können also in Beziehung zu einander stehen, wie dies schon in den ER-Modellen in Form der Beziehungen (Relationships) zwischen den Entitäten zum Ausdruck kommt. Abhängig von den Kardinalitäten unterscheidet man:
- 1:1 Beziehungen ⇒ Zu einem Tupel gehört genau ein anderes Tupel.
- 1:N Beziehung ⇒ Zu einem Tupel gehört mindestens ein (≥ 1) anderes Tupel.
- M:N Beziehung ⇒ Zu einem Tupel gehören beliebig viele (≥ 0) andere Tupel umgekehrt gilt dasselbe.

Relationale Algebra

Eine Algebra besteht aus einer Grundmenge und einer Menge von Operationen, die auf dieser Grundmenge definiert sind und deren Ergebnisse wieder in der Grundmenge liegen.

- In unserem Fall ist die Grundmenge die Menge aller Relationen (Tabellen), die Operationen bilden eine oder mehrere Relationen wieder auf Relationen ab und können flexibel miteinander verknüpft werden.
- Die relationale Algebra legt ein mathematisches Fundament für relationale DBMS.
- Eine Datenbank-anfrage kann mathematisch vergleichsweise einfach modelliert werden.
    - Mithilfe der Operationen der Algebra kann diese dann z. B. durch Rechenregeln in eine andere Anfrage umgeformt werden, die dasselbe Ergebnis liefert aber effizienter ausgeführt werden kann.

Relationale Algebra

Selektion σ

Mit der Selektion σ werden aus einer Relation entsprechend einer Bedingung (Prädikat) ein oder mehrere Tupel herausgegriffen.

Mitarbeiter

PersonalNr	Name	Gehalt	AbteilungsNr
101	Meissner	75 000	1
102	Lehmann	98 000	1
103	Kunze	75 000	2

Selektion: $\sigma_{Gehalt=75.000}(Mitarbeiter)$

PersonalNr	Name	Gehalt	AbteilungsNr
101	Meissner	75 000	1
103	Kunze	75 000	2

Abb. 5. Anwendung der Selektion auf die Relation Mitarbeiter

Relationale Algebra

Projektion π

Mit der Projektion π können Attribute $Attr_i$ (in einer Tabelle wären das Spalten) aus einer Relation herausgegriffen werden.

Mitarbeiter

PersonalNr	Name	Gehalt	AbteilungsNr
101	Meissner	75 000	1
102	Lehmann	98 000	1
103	Kunze	75 000	2

Projektion: $\pi_{Name,Gehalt}(Mitarbeiter)$

Name	Gehalt
Meissner	75 000
Lehmann	98 000
Kunze	75 000

Abb. 6. Anwendung der Projektion auf die Relation Mitarbeiter

Relationale Algebra

Kartesisches Produkt (Kreuzprodukt ×)

Mit dem (kartesischen) Produkt × wird analog zum kartesischen Produkt zweier Mengen das Produkt zweier Relationen gebildet. Die resultierende Relation Rel 1 × Rel 2 enthält dann sämtliche Kombinationen, die sich aus den Tupeln der beiden Relationen bilden lassen.

Rel_1

Attribut 1.1	Attribut 1.2
1	A
2	B
3	C

Rel_2

Attribut 2.1
x
y

Produkt: $Rel_1 \times Rel_2$

Attribut 1.1	Attribut 1.2	Attribut 2.1
1	A	x
1	A	y
2	B	x
2	B	y
3	C	x
3	C	y

Abb. 7. Aus den Relationen Rel 1 und Rel 2 entsteht durch Rel 1 × Rel 2 eine neue Relation

Relationale Algebra

Vereinigung (Union, ∪)

Die Operation Vereinigung ∪ (Union) vereinigt zwei Relationen Rel 1 und Rel 2 zu einer Relation Rel 1 ∪ Rel 2 , in der alle Tupel der ersten Relation und der zweiten Relation enthalten sind. In beiden Relationen enthaltene Tupel erscheinen im Ergebnis nur einmal. Diese Operation ist nur ausführbar, wenn die Anzahl der Attribute in den beiden zu verknüpfenden Relationen gleich sind und wenn die Attribute miteinander kompatibel sind

$Abteilungen_A$

Name	Standort
Einkauf	München
DV-Org.	Augsburg
Entwicklung	Augsburg
Verwaltung	München

$Abteilungen_B$

Name	Standort
DV-Org.	Augsburg
Produktion	Straubing
Marketing	München

$Abteilungen_A \cup Abteilungen_B$

Name	Standort
Einkauf	München
DV-Org.	Augsburg
Entwicklung	Augsburg
Produktion	Straubing
Marketing	München
Verwaltung	München

Abb. 8. Aus den Relationen Abteilungen A und Abteilungen B entsteht die rechts abgebildete Relation durch Abteilungen A ∪ Abteilungen B

Relationale Algebra

Mengendurchschnitt (Intersection, ∩)

Durch die Operation Mengendurchschnitt ∩ (Intersection) wird aus zwei Relationen Rel 1 und Rel 2 als Ergebnis eine Relation Rel 1 ∩ Rel 2 gebildet, die nur diejenigen Tupel enthält, die sowohl in der ersten Relation als auch in der zweiten Relation enthalten sind. Diese Operation ist nur ausführbar, wenn die beiden Relationen in dem oben bei der Operation Union erläuterten Sinne union-kompatibel sind.

$Abteilungen_A$

Name	Standort
Einkauf	München
DV-Org.	Augsburg
Entwicklung	Augsburg
Verwaltung	München

$Abteilungen_B$

Name	Standort
DV-Org.	Augsburg
Produktion	Straubing
Marketing	München

$Abteilungen_A \cap Abteilungen_B$

Name	Standort
DV-Org.	Augsburg

Abb. 9. Aus den Relationen Abteilungen A und Abteilungen B entsteht die rechts abgebildete Relation durch Abteilungen A ∩ Abteilungen B

Relationale Algebra

Mengendifferenz (Difference, −)

Durch die Operation Differenz (Difference) wird aus zwei Relationen Rel 1 und Rel 2 als Ergebnis eine Relation Rel 1 − Rel 2 gebildet, die nur diejenigen Tupel enthält, die in der ersten Relation enthalten sind, in der zweiten Relation jedoch nicht. Diese Operation ist nur ausführbar, wenn die beiden Relationen union-kompatibel sind.

$Abteilungen_A$

Name	Standort
Einkauf	München
DV-Org.	Augsburg
Entwicklung	Augsburg
Verwaltung	München

$Abteilungen_B$

Name	Standort
DV-Org.	Augsburg
Produktion	Straubing
Marketing	München

$Abteilungen_A - Abteilungen_B$

Name	Standort
Einkauf	München
Entwicklung	Augsburg
Verwaltung	München

Abb. 10. Aus den Relationen Abteilungen A und Abteilungen B entsteht durch Abteilungen A − Abteilungen B die rechts abgebildete Relation

Relationale Algebra

Division ÷

Es wird die Division einer Relation Rel 1 durch eine Relation Rel 2 bezüglich eines Attributes A aus Rel 1 betrachtet, wobei Rel 2 ebenfalls ein Attribut mit demselben Wertebereich enthalten muss wie das bei der Division verwendete Attribut A. Aus der Ergebnisrelation wird zunächstdas Attribut A gelöscht. Sodann verbleiben in der Ergebnis-Relation Rel 1 ÷ Rel 2 nur diejenigen Tupel, für die irgendein Attribut der Relation Rel 1 alle Werte von Relation Rel 2 enthält.

Rel_1

Attribut 1.1	Attribut 1.2
1	A
2	A
2	B
2	C
3	D

Rel_2

Attribut 2.1
A
B

Division: $Rel_1 \div Rel_2$

Attribut 1.1
2

Abb. 11. In der Relation Rel 1 gibt es nur einen Wert für Attribut 1.1, der mit allen Werten von Attribut 2.1 aus Rel 2 vorkommt: Tupel sind (2, A) und (2, B), der Wert ist damit 2. Dieser Wert findet sich in der Ergebnisrelation

Relationale Algebra

Verbund (Join, $\trianglelefteq\triangleright$)

Bei der Operation Join (Verbund) wird in dessen allgemeinster Form zunächst das Produkt $\times$ zweier Relationen gebildet, danach über eine Selektion mit der Verknüpfung zweier Attribute aus je einer der Relationen eine Anzahl von Zeilen herausgegriffen und schließlich über eine Projektion eine Reihe von Attributen ausgewählt. Vorausgesetzt wird dabei, dass die beiden im Join verwendeten Attribute denselben Wertebereich haben.

Umbenennung β

Um vollständig mit Relationen rechnen zu können, fehlt noch eine Operation zur Umbenennung von Spalten. Diese wird mit β bezeichnet.

Die Datenbanksprache SQL

Mit SQL (Structured Query Language) steht eine Sprache zur Verfügung, mit der alle Funktionen auf relationalen Datenbanken ausgeführt werden können. Eine Basis von SQL ist die im vorangegangenen Abschnitt eingeführte Relationenalgebra.

- SQL ist eine **deklarative Sprache**

Der wesentliche Unterschied zwischen Sprachen wie Java oder C und SQL liegt in der Art und Weise wie Anweisungen formuliert werden. In SQL-Anweisungen wird nicht beschrieben wie ein Datenbankzugriff zu erfolgen hat (algorithmisch, prozedural), es wird vielmehr beschrieben, was der Anwender beabsichtigt (deklarativ).

Die Datenbanksprache SQL

- Wenn ein DBMS eine SQL-Anweisung ausführt, muss daher eine Umsetzung in prozedurale Anweisungen erfolgen, da nur solche durch einen Computer ausführbar sind.
- Dies kann geschehen durch Umwandlung von SQL-Ausdrücken in Ausdrücke der relationalen Algebra, die dann ihrerseits in einer Sprache wie C unter Verwendung von Konzepten wie B-Bäumen oder Hashing ausgeführt werden können.
- SQL basiert zwar auf der relationalen Algebra, besteht aber keineswegs nur aus einfachen Zuordnungen von SQL-Anweisungen zu den Operationen der relationalen Algebra.
- Auch wurden die mathematisch geprägten Ausdrücke der relationalen Algebra durch eher umgangssprachliche Formulierungen ersetzt und die Operationen durch Schlüsselworte der Sprache SQL:

`Relation → Tabelle, Tupel → Satz oder Zeile und Attribut → Spalte.`

Die Datenbanksprache SQL

- SQL-Anweisungen lassen sich zwei wesentlichen Gruppen zuordnen:
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language), sowie noch die
  - DCL (Data Contol Language) zur Wahrung der Datenintegrität.

DDL-Anweisungen	Definition eines Datenbankschemas
`CREATE TABLE`	Erstellen einer Tabelle
`CREATE VIEW`	Definieren einer Sicht (logische Tabelle, die aus anderen Tabellen mit Daten befüllt wird)
`CREATE INDEX`	Definition eines Index um eventuell damit Anfragen zu beschleunigen
`DROP TABLE` / `VIEW` / `INDEX`	Löschen von Tabellen, Sichten, Indizes, Constraints etc.
`ALTER TABLE`	Hinzufügen, Ändern, Löschen einer Spalte oder eines Constraints zu einer Tabelle

Tab. 1. Zusammenstellung der wichtigsten SQL-Anweisungen der DDL

Die Datenbanksprache SQL

Name	Beschreibung
CHAR (n)	Zeichenkette mit fester Länge von *n*-Zeichen
VARCHAR (n)	Zeichenkette mit einer Länge von bis zu *n*-Zeichen
INT	Ganze Zahl, Vorzeichenbehaftet, Größe Abhängig von der Implementierung
NUMERIC (p,s)	Festpunkt Zahl, mit *p*-Stellen insgesamt und davon *s*-Stellen nach dem Komma.
FLOAT (n)	Gleitpunkt Zahl, mit der Genauigkeit *n* (d. h. Mantisse verwendet *n* Bit, z. B. 53)
DATE	Datumsangaben
TIMESTAMP	Zeitstempel mit Datum und Uhrzeit

Tab. 2. Zusammenstellung der wichtigsten SQL-Datentypen

DML-Anweisungen	Änderungen (Update) der Daten
INSERT	Einfügen von Zeilen in eine Tabelle
DELETE	Löschen von Zeilen aus einer Tabelle
UPDATE	Ändern von Zeilen einer Tabelle
SELECT	Suchen von Zeilen in einer oder mehreren Tabellen

Tab. 3. Zusammenstellung der DML Anweisungen in SQL

Erzeugen einer Tabelle

- Es muss ein Schema angegeben werden dass die Spalten definiert (Name, Typ, weitere Attribute)
- Zu jeder Spalte der Datenbank wird ein Name und ein Wertebereich (Datentyp) angegeben. SQL gibt ein eigenes Typsystem vor.

```
CREATE TABLE Mitarbeiter (
  PersonalNr int primary key,
  Name varchar(100) not null,
  Gehalt int,
  AbteilungsNr int,
  foreign key (AbteilungsNr) references Abteilungen (AbteilungsNr)
);
```

Def. 1. Erzeugung einer nicht existierenden Tabelle "Mitarbeiter"

Einfügen, Ändern und Löschen von Daten

Um Daten in eine Tabelle einzufügen, zu ändern oder zu löschen bietet SQL die drei Anweisungen INSERT , UPDATE und DELETE an. Diese Anweisungen zählen wie das Suchen zur Data Manipulation Language (DML) innerhalb von SQL.

```
INSERT INTO Mitarbeiter VALUES (104, ’Beneken’, 38000, 5);
INSERT INTO Mitarbeiter (PersonalNr, Name, Abteilung) VALUES (106, ’Schmidt’, 5);
```

Def. 2. Einfügen von Zeilen in die Tabelle "Mitarbeiter"

Suchen mit SELECT

```
SELECT <Liste von Spalten evtl. mit Berechnungen>
  FROM <Liste mit Tabellen und Sichten>
   [WHERE <Bedingung>]
```

Def. 3. SELECT -Anweisung oder auch SELECT-FROM-WHERE -Anweisung (SFW).

- Die nach der SELECT -Anweisung spezifiziert die Spaltennamen (Attribute) einer Tabelle oder mehrerer Tabellen.
  - Die Ergebnistabelle umfasst genau diese genannten Spalten. Dies ist vergleichbar mit der Projektion π aus der relationalen Algebra. Erlaubt ist auch ein Stern (*), der für alle Attribute steht.
  - Der Name der Spalte in der Ergebnistabelle kann dabei angepasst werden (SELECT PersonalNr AS Nummer FROM ...).
  - Weiterhin können die Werte in den Spalten auch berechnet sein, etwa durch Zusammenfassung von mehreren Werten der Ergebniszeilen, dazu werden Funktionen wie min , max , sum , avg oder count verwendet.

Suchen mit SELECT

- Nach dem Schlüsselwort FROM folgt ein Tabellenname oder eine Liste von Tabellennamen. Wenn keine weiteren Bedingungen hinter WHERE spezifiziert sind, wird das kartesische Produkt (×) aller angegebenen Tabellen gebildet.

- Die Zeilen in der Ergebnistabelle werden mit WHERE eingeschränkt: Eine Bedingung liefert für die Zeilen, die in das Ergebnis übernommen werden sollen true und für alle anderen false . In der relationalen Algebra ist dies die Selektion σ Bedingung .

```
SELECT Name, Gehalt
 FROM Mitarbeiter
  WHERE AbteilungsNr = 3 AND Gehalt > 80000
```

Bsp. 2. Beispiel einer selektiven und konditionalen Suchanweisung

SQLite3 Live

Ein Datenbank Dateisystem

- Ein Dateisystem ist i.A. ein hierarchischer Baum (oder Graph)
  - Verzeichnisse sind Knoten für weitere Teilbäume (Verzeichnisse) oder Dateien
  - Dateien sind Blätter
  - Verzeichnisse werden durch / getrennt
- Ein Pfad beginnend beim Wurzelknoten / ist eine Referenz zu einem Verzeichnis oder einer Datei.

Naive DB Implementierung

1. Jedes Verzeichnis ist eine Tabelle
   - Eine Zeile besteht aus einem Namen und Inhalt
   - Der Inhalt kann ein Verweis auf eine tiefere folgende Verzeichnistabelle oder der Inhalt einer Datei sein.

Ein Datenbank Dateisystem

```
CREATE TABLE FolderNNN (
  Name char,
  Type int,
  Site int,
  Date datetime,
  Content char
);
```

Bsp. 3. Type=1: Verzeichnis, Type=2 (Text): Datei, NNN = Fortlaufende Nummerierung 001, 002, ..

Ein Datenbank Dateisystem

```
CREATE TABLE Folder000 (Name char, Type int, Size int,
                        Date datetime, Content char);
INSERT INTO Folder000 values
  ("foo",  2 , 11 , "Today" ,     "Hello World"),
  ("foo2", 2 , 3  , "Yesterday" , "Joe"),
  ("dir1", 1 , 0  , "1.1.1970" ,  "Folder001");
CREATE TABLE Folder001 (Name char, Type int, Size int,
                        Date datetime, Content char);
INSERT INTO Folder001 values
  ("foo",  2 , 4  , "Today" ,     "End."),
```

Bsp. 4. Ein einfaches Dateisystem

Wie kann man nun aber Pfade auflösen und an die Inhalte von Dateien gelangen oder tiefere Verzeichnisse lesen (listen)?

Rekursive SQL Queries

Man kann einen solchen Dateisystempfad nur **rekursiv** mit Variablen über mehrere Tabellen hinweg auflösen und das Ergebnis ermitteln.

- SQL bietet rekursive Statements

Beispiel Stammbaum

```
CREATE TABLE family(
  name TEXT PRIMARY KEY,
  mom TEXT REFERENCES family,
  dad TEXT REFERENCES family,
  born DATETIME,
  died DATETIME -- NULL if still alive
  -- other content
);
```

Bsp. 5. Beispiel Stammbaum

Rekursive SQL Queries

```
WITH RECURSIVE
  parent_of(name, parent) AS
    (SELECT name, mom FROM family UNION SELECT name, dad FROM family),
  ancestor_of_alice(name) AS
    (SELECT parent FROM parent_of WHERE name='Alice'
     UNION ALL
     SELECT parent FROM parent_of JOIN ancestor_of_alice USING(name))
SELECT family.name FROM ancestor_of_alice, family
 WHERE ancestor_of_alice.name=family.name
   AND died IS NULL
 ORDER BY born;
```

Bsp. 6. Rekursive Stammbaumiteration