Datenbanken

1. Vorlesung: Konzepte zum Speichern von Information, Relationen und SQL

Prof. Dr. Andreas de Vries

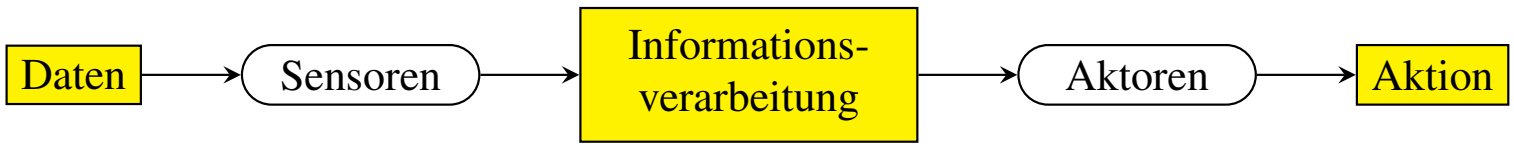
Fachhochschule Südwestfalen, Standort Hagen

Version: 20. September 2019

Inhaltsübersicht

Die Rolle von Information in der Natur

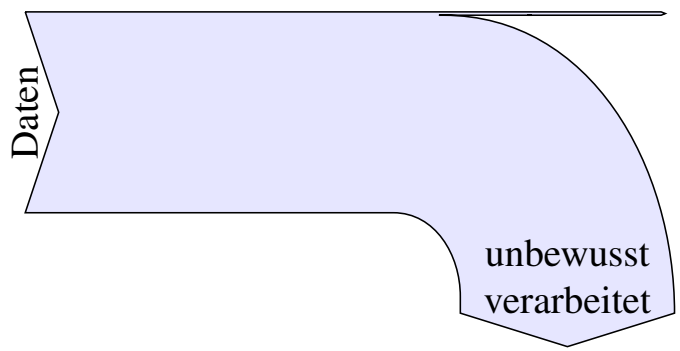
- Informationsverarbeitung biologischer Organismen:



  - Erfassung von Reizen der Umgebung („Daten“) über Sinnesorgane („Sensoren“)
  - Informationsverarbeitung im zentralen Nervensystem
  - Reaktionen über Muskeln („Aktoren“)
- Maß für eine Datenmenge: Bits, die Einheit des Informationsgehalts
- Informationsfluss der Sinnesorgane und des Bewusstseins beim Menschen:

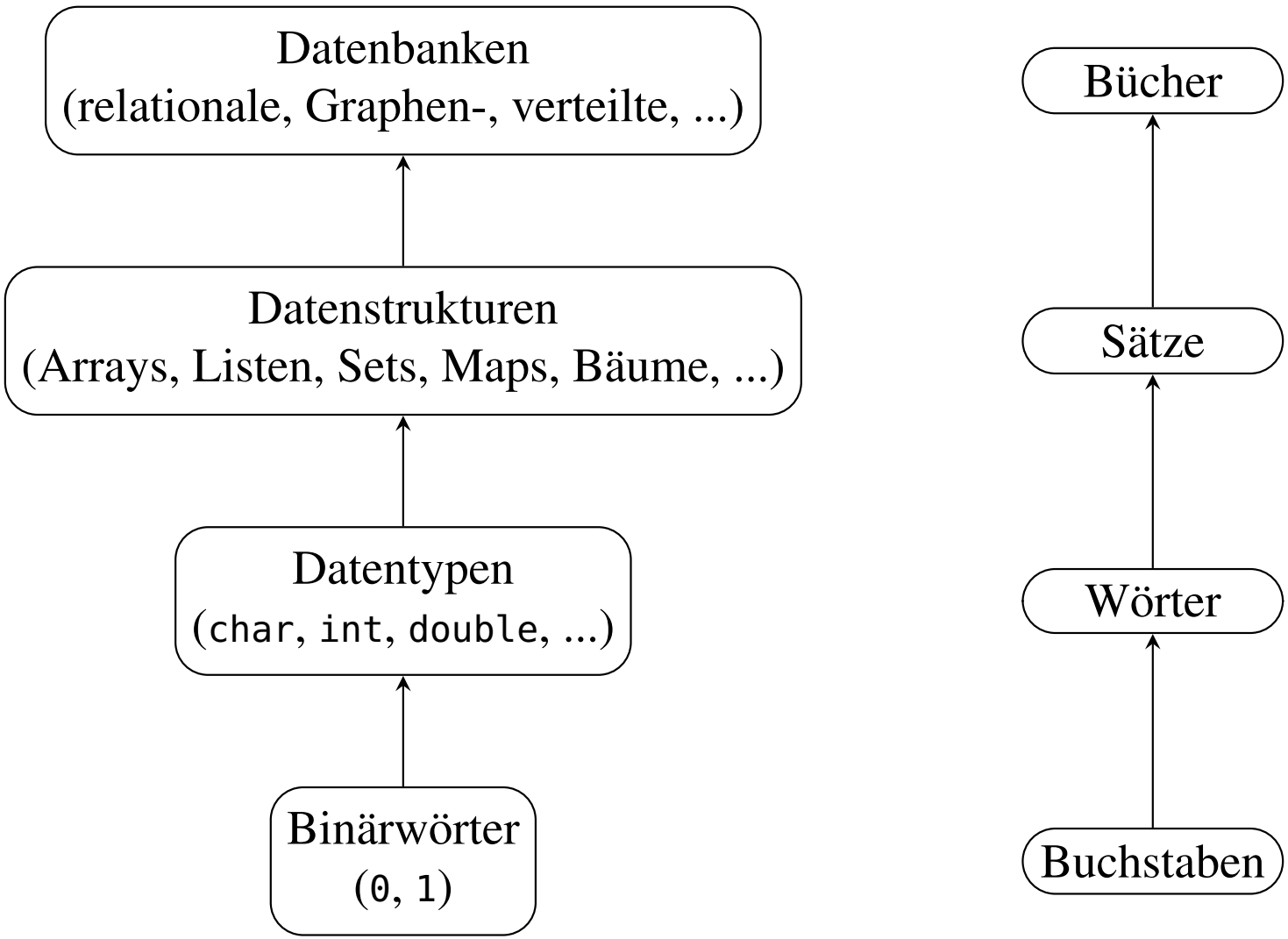
Sinnessystem	**Datenrate [bit/s]**	
	Kapazität	**Bewusstsein**
Augen	10 000 000	40
Ohren	100 000	30
Haut	1 000 000	5
Geschmack	1 000	1 (?)
Geruch	100 000	1 (?)
Gesamt	11 201 000	77

“(?)” = Schätzwerte, Quelle: ZIMMERMANN (1993)



Konzepte der Informationsspeicherung

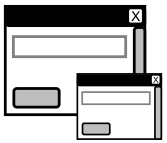
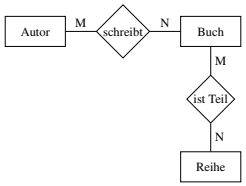
Die Hierarchie der Speicherkonzepte in Informatik & Literatur:



Inhaltsübersicht

Datenbanken als dreischichtige Systeme

Datenbanksysteme basieren auf der *ANSI-SPARC-Referenzarchitektur* (1975):

Externe Ebene		Benutzeroberflächen, Views
Logische Ebene		ER-Diagramme, Datenmodell, Normalisierung
Interne Ebene		Speicherformate und -verfahren, Datenzugriffssteuerung

Definition

Ein *Datenbanksystem* ist ein Softwaresystem zur Speicherung und Verwaltung strukturierter Daten, das die drei Ebenen der ANSI-SPARC-Architektur realisiert. Die Menge der gespeicherten Daten heißt *Datenbank* oder auch *Datenbasis*, die Menge an Programmen zum Zugriff auf die Daten heißt *Datenbankmanagementsystem (DBMS)*. Vgl. (KRCMAR, 2015, §6.1.3.1).

Relationen – Definition

- Die meisten Datenbanksysteme sind „relational“ und werden entsprechend RDBMS abgekürzt.
- Was ist eine „Relation“ in der Mathematik?

Definition

Eine *Relation* R ist eine Teilmenge des kartesischen Produkts von Mengen A_1, ..., A_n:

$$R \subseteq A_1 \times \ldots \times A_n. \tag{1}$$

Formal lassen sich die Elemente der Relation als n-Tupel schreiben:

$$(x_1, \ldots, x_n) \qquad \text{mit} \qquad x_j \in A_j \text{ für } j = 1, \ldots, n. \tag{2}$$

Der „*Relationentyp*“ wird mit

$$R(A_1, \ldots, A_n)$$

bezeichnet.

Relationen – Beispiel

Beispiel

Gegeben zwei Mengen A_1 = „Farbe“ und A_2 = „Karte“, die Spielkarten beschreiben:

$$A_1 = \{\text{Kreuz, Pik, Herz, Karo}\},$$
$$A_2 = \{2, 3, 4, 5, 6, 7, 8, 9, 10, \text{Bube, Dame, König, Ass}\}$$

So ist $A_1 \times A_2$ die Menge aller $4 \cdot 13 = 52$ Kombinationen der beiden Eigenschaften, also ein komplettes Pokerblatt. Das folgende „Full House“ $R \subset A_1 \times A_2$ ist eine Relation von $A_1 \times A_2$:

$$R = \{(\text{Pik, Ass}), (\text{Kreuz, Ass}), (\text{Herz, Ass}), (\text{Herz, König}), (\text{Karo, König})\}, \tag{3}$$

R

A_1	A_2
Pik	Ass
Kreuz	Ass
Herz	Ass
Herz	König
Karo	König

oder eben:

Full House

Farbe	Karte
Pik	Ass
Kreuz	Ass
Herz	Ass
Herz	König
Karo	König

Die Menge *aller* Full Houses ist ebenfalls eine Relation von $A_1 \times A_2$.

Relationen und Tabellen

Eine Relation von (endlichen) Mengen mit m Elementen lässt sich stets als eine Tabelle darstellen, deren Spalten die n Mengen des kartesischen Produkts sind und deren Zeilen die Kombinationen der Werte sind.

$$R = \{(x_{i1}, \ldots, x_{in}) : i = 1, \ldots, m \text{ und } x_{ij} \in A_j \text{ für } j = 1, \ldots, n\} \quad \Longleftrightarrow \quad \begin{array}{|cccc|} \hline A_1 & A_2 & \cdots & A_n \\ \hline x_{11} & x_{12} & \cdots & x_{1n} \\ \vdots & \vdots & & \vdots \\ x_{i1} & x_{i2} & \cdots & x_{in} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \\ \hline \end{array} \tag{4}$$

Umgekehrt ist jede Tabelle eine endliche Relation, wenn man vereinbart, dass nur Tabellen ohne doppelte Zeilen betrachtet werden und zudem zwei Tabellen als gleich angesehen werden, wenn sie bis auf Zeilenvertauschungen gleich sind, vgl. (PIEPMEYER, 2011, S. 37).

Relationen in verschiedenen Fachdisziplinen

Relationales Modell (Mathematik)	**Tabellenmodell** (Angewandte Informatik)	**Objektmodell** (Programmierung)
Relation *(relation)*	Tabelle *(table)*	Klasse *(class)*
Attribut *(attribute)*	Spalte *(column)* / Feld *(field)*	Attribut *(attribute)*
Tupel *(tuple)*	Zeile *(row)* / Datensatz *(record set)*	Objekt *(object)*

- Mit der Notation

$$R(A_1, A_2, \ldots, A_n) \tag{5}$$

bezeichnen wir den *Relationentyp* der Relation R. Im Zusammenhang mit Datenbanken spricht man oft auch von dem *Schema* der Relation.

Beispiel

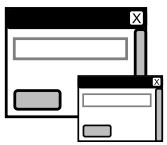
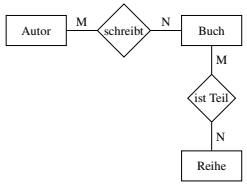
Der Relationentyp der Relation „Full House“ aus dem obigen Beispiel lautet

$$\text{Full House (Farbe, Karte)}. \tag{6}$$

An dieser Notation kann man ablesen, dass die Relation ein 2-Tupel (x_1, x_2) ist, dessen erster Wert x_1 eine Farbe und dessen zweiter x_2 eine Karte ist.

Relationale Datenbanksysteme (RDBMS)

- In einem später berühmt gewordenen Fachartikel veröffentlichte CODD (1970) seine Gedanken zu Relationen als Grundlage für ein Datenbankmodell.
- Das Modell fand schnell weite Beachtung.
- Heute werden auf Tabellen basierende Datenbanken *relational* genannt,
- das Verwaltungssystem entsprechend *RDBMS* (für *relationales Datenbankmanagementsystem*).
- Die Begriffe der Relationen und der Tabellen beziehen sich dabei genau genommen auf die logische Ebene der ANSI-SPARC-Architektur

Externe Ebene		Benutzeroberflächen, Views
Logische Ebene		ER-Diagramme, Datenmodell, Normalisierung
Interne Ebene		Speicherformate und -verfahren, Datenzugriffssteuerung

Inhaltsübersicht

Was ist SQL? *SQL (Structured Query Language) ...*

- ... ist eine Programmiersprache zur Definition von Datenstrukturen und zur Datenverwaltung relationaler Datenbanken.
- ... erschien 1974, wurde 1986 Standard der ANSI und ein Jahr später der ISO. Die 2019 aktuellste Version ist SQL:2016.
- ... ist eine „deklarative“ Programmiersprache
  - d.h. es wird „ergebnisorientiert“ die Logik und die Funktionalität des Programms programmiert.
  - Bei der imperativen Programmierung steht das *Wie* im Vordergrund, bei der deklarativen Programmierung das *Was*, das berechnet werden soll.
- ... hat die Merkmale einer Interpretersprache, d.h. der Quelltext wird zur Laufzeit direkt übersetzt und ausgeführt.
  - Die meisten RDBMS übersetzen den Quelltext aber intern in einen „Query Execution Plan“, der als temporäre Datei gespeichert und dann ausgeführt wird (also ähnlich dem Bytecode von Java oder C#).
  - In diesen Fällen ist SQL also eigentlich auch eine Compilersprache, allerdings ist die kompilierte Datei nur DBMS-intern verfügbar.
- ... ist „Turing-vollständig“, d.h. ermöglicht logische `WHILE`-Schleifen, so dass alle berechenbaren Funktionen implementiert werden können.[1] Vgl. (DE VRIES, 2019, §5.4)

[1] `https://stackoverflow.com/questions/900055/`, `https://modern-sql.com/de`

SQL: Wichtige reservierte Wörter und Literale

Wie jede Programmiersprache hat auch SQL

- *reservierte Wörter* mit festen Bedeutungen, die nach einer bestimmten *Syntax* („Grammatik“) angeordnet werden dürfen,
- *Literale*, d.h. Symbole mit festem Wert, wie Zahlen oder *Strings*, also von Apostrophs eingerahmte Zeichenfolgen (`'Abc'`),
- besondere Zeichen und Zeichenfolgen wie Wildcards, Kommentarmarkierungen oder Rechenoperationen.

Wichtige reservierte Wörter in SQL

ABS	ADD	ALL	ALTER	AND	ANY	AS	AVG	BETWEEN	BY
CHECK	COLLATE	CONCAT	CONSTRAINT	COUNT	CREATE	DATE	DOMAIN	DOUBLE	DECIMAL
DELETE	DESC	DISTINCT	DROP	EXCEPT	EXISTS	FOREIGN	FROM	GRANT	GROUP
HAVING	IN	INNER	INSERT	INTEGER	INTERSECT	INTO	IS	JOIN	LEFT
LIKE	MAX	MIN	NOT	NULL	OF	ON	OR	ORDER	OUTER
POSITION	POWER	PRIMARY	RECURSIVE	REFERENCES	RESTRICT	REVOKE	RIGHT	ROUND	SELECT
SET	SMALLINT	SQRT	START	STDDEV	SUBSTRING	SUM	TABLE	TIME	TIMESTAMP
UNION	UNIQUE	UPDATE	USER	VALUES	VARCHAR	VARIANCE	VIEW	WHERE	WITH

Literale

NULL | 0, 1, -2 | 1.2, 3.1E8	'Abc'

Besondere Zeichen

-- | * | +, - , /, % | =, !=, <>, >, <, >=, <=

- SQL ist nicht schreibungssensitiv *(case insensitive)*,
- In dieser Vorlesung gilt die Konvention: *Reservierte Wörter in Großbuchstaben.* (... bis auf Datentypen)

SQL: Verschiedene Dialekte

- Trotz der Standards existieren jedoch viele Dialekte für die verschiedenen Datenbanksysteme, so dass ein einmal geschriebener SQL-Quelltext nicht unbedingt auf allen Datenbanken läuft.
- Der zentrale Grund ist, dass die reservierten Wörter zu einem gewissen Teil nicht identisch sind. Allein schon die Anzahl der reservierten Wörter variiert zwischen den verschiedenen Datenbanksystemen:

Anzahl reservierter Wörter

OpenOffice Base*	Microsoft Access	MariaDB* / MySQL	PostgreSQL*	Azure SQL / SQL Server	Oracle
61	252	238	93	185	110

* Open Source / freie Software

Zum Vergleich: Übliche (imperative) Programmiersprachen haben deutlich weniger reservierte Wörter, z.B. hat Java 49 reservierte Wörter, Python nur 30.

SQL: Kommentare

- Wie in jeder Programmiersprache gibt es auch in SQL die Möglichkeit, Kommentare einzufügen, um anderen Programmierern (und sich selbst) Hinweise zum Zweck oder zur Funktionsweise des Programms zu geben.
- Kommentare werden von dem Interpreter ignoriert und daher nicht ausgeführt.
- Es gibt zwei Arten von Kommentaren in SQL, den einzeiligen Kommentar,

```
-- Dies ist ein Kommentar
```

der durch zwei Minuszeichen für den nachfolgenden Rest der Zeile markiert wird, und der mehrzeilige (oder Block-) Kommentar,

```
/* Dies ist ein Kommentar,
   der über mehrere Zeilen geht
 */
```

dessen Bereich mit `/*` geöffnet und mit `*/` geschlossen wird.

SQL: Datentypen

- *Datentypen* sind die elementaren und die zusammengesetzten Speichereinheiten einer Programmiersprache, die jeweils einen bestimmten Speicherbereich zugewiesen bekommen. Datentypen definieren die Wertebereiche dieser Einheiten.
- Es gibt in der Regel drei Kategorien von Datentypen, Zeichen *(character)*, Ganzzahlen *(integer)* und Gleitkommazahlen *(float, double)*.
- Aus der Aneinanderreihung von Zeichen entsteht Text *(string)* als ein zusammengesetzter Datentyp.

SQL: Datentypen

Datentyp	Speicher	Beschreibung
`varchar(`n`)`	n Byte	Text (String) mit Maximallänge n; wird mit Apostrophs `'Abc'` gekennzeichnet
`text`	x Byte	Text beliebiger Länge; wird mit Apostrophs `'Abc'` gekennzeichnet
`smallint`	2 Byte	Ganzzahl im Bereich von $-32\,768$ bis $32\,767$ ($= -2^{15}$ bis $2^{15}-1$)
`int`	4 Byte	Ganzzahl im Bereich von $-2\,147\,483\,648$ bis $2\,147\,483\,647$ ($= -2^{31}$ bis $2^{31}-1$)
`bigint`	8 Byte	Ganzzahl im Bereich von $-9\,223\,372\,036\,854\,775\,808$ bis $9\,223\,372\,036\,854\,775\,807$ ($= -2^{63}$ bis $2^{63}-1$)
`float`	4 Byte	Gleitkommazahl im Bereich von `1.0E-38` bis `3E+38` ($= 2^{-126}$ bis 2^{128}), auf bis zu 8 Dezimalstellen genau
`double`	8 Byte	Gleitkommazahl mit einem Betrag von `2E-308` bis `2E+308` ($= 2^{-1022}$ bis 2^{1024}) auf bis zu 16 Dezimalstellen genau
`decimal`(p,s)	$f(p)$ Byte*	Festkommazahl mit maximal p Stellen, davon s Nachkommastellen; Restriktionen: $p \leqq 38$ bei SQL Server ($p \leqq 65$ bei MariaDB) und $0 \leqq s \leqq p$
`date`	3 Byte	Datum, je nach DBMS im Format yyyy-mm-dd oder #mm/dd/yyyy#
`time`	4 Byte	Zeit, je nach DBMS meist im Format HH:MM:SS
`datetime`	8 Byte	Datum und Uhrzeit

* Erläuterung: $f(p) = 1 + 4 \cdot \left\lceil \frac{p}{9{,}5} \right\rceil$

SQL: Anweisungen

- Ein SQL-Programm besteht aus einer oder mehreren Anweisungen, auch Befehle oder Statements genannt.
- Zwei Anweisungen werden in SQL mit einem Semikolon getrennt. (Für die letzte auszuführende Anweisung darf es weggelassen werden.)
- SQL kann
  - die Struktur der Datenbank verwalten → DDL
  - Daten editieren → DML
  - Zugriffsrechte verwalten → DCL
- Auf den folgenden Folen werden wir die Syntax dieser drei Sprachteile kurz vorstellen.

SQL: Die *Data Definition Language* (DDL)

- Mit der Data Definition Language werden die Datenbank und die Struktur ihrer Tabellen angelegt und verwaltet.
- Dazu stehen die Ausdrücke **`CREATE`**, **`ALTER`** und **`DROP`** zur Verfügung, die kombiniert werden können mit den Ausdrücken **`DATABASE`**, **`TABLE`** und **`USER`**:

$$\left\{\begin{array}{c}\texttt{CREATE}\\ \texttt{ALTER}\\ \texttt{DROP}\end{array}\right\} + \left\{\begin{array}{c}\texttt{DATABASE}\\ \texttt{TABLE}\\ \texttt{USER}\end{array}\right\} + \mathit{Name} + \mathit{[Zusatzoptionen]};$$

- So werden Datenbanken, Tabellen und User angelegt, verändert und gelöscht.

Vgl. `https://de.wikibooks.org/wiki/Einf%C3%BChrung_in_SQL:_DDL_-_Struktur_der_Datenbank`

SQL: Die *Data Manipulation Language* (DML)

Mit der *DML* werden Daten angelegt, gelesen, aktualisiert und gelöscht (CRUD für *create*, *read*, *update* und *delete*). Die Syntax:

- Daten anlegen mit **INSERT**:

```
INSERT INTO tabelle (spalte_1, ..., spalte_n) VALUES
  (wert_11,  ...,  wert_1n),
      ...
  (wert_m1,  ...,  wert_mn);
```

- Daten lesen mit **SELECT**:

```
SELECT spalte_1, ..., spalte_n FROM tabelle [... Klauseln ...];
```

  Die Ausdrücke in eckigen Klammern sind optional. Statt der Spaltenliste kann einfach ein Sternchen $*$ stehen.

- Daten aktualisieren mit **UPDATE**:

```
UPDATE tabelle SET
  spalte_1 = wert_1,
    ...
  spalte_n = wert_n
WHERE Bedingung;
```

- Daten löschen mit **DELETE**:

```
DELETE FROM tabelle WHERE Bedingungen;
```

SQL: Die *Data Control Language* (DCL)

- Mit der *DCL* werden die Zugriffsrechte auf die Datenbank verwaltet.
- Sie besteht im Wesentlichen aus den zwei Befehlen
  - **GRANT** für die Vergabe von Zugriffsrechten,
  - **REVOKE** für deren Entzug.
- Für weitere Details siehe

  `https://de.wikibooks.org/wiki/Einf%C3%BChrung_in_SQL:_DCL_-_Zugriffsrechte`
- Wir werden in dieser Vorlesung mit der DCL nicht weiter arbeiten.

Inhaltsübersicht

Erzeugen eines Relationentyps als Tabelle in SQL

- Um eine Tabelle in SQL zu erzeugen, muss ihr Relationentyp klar sein.
- Der Relationentyp

  `tabelle (datentyp_1, ..., datentyp_n)`

  wird mit **`CREATE TABLE`** definiert:

```
CREATE TABLE tabelle (
   spalte_1 datentyp_1,
   spalte_2 datentyp_2,
     ...
   spalte_n datentyp_n,
   PRIMARY KEY(spalte_x, ..., spalte_y)
);
```

- **`PRIMARY KEY`** bestimmt, welche Spalten der Tabelle den „Primärschlüssel“ bilden.
- Die Zeile mit **`PRIMARY KEY`** kann auch weggelassen werden, dann hat die Tabelle eben keinen Primärschlüssel.

Wie kann man eine Relation mit SQL implementieren?

Betrachten wir die Relation „Full House“ $\subset$ Farbe $\times$ Karte,

„Full House“ = {(Pik, Ass), (Kreuz, Ass), (Herz, Ass), (Herz, König), (Karo, König)}

der zwei Mengen

$$\text{Farbe} = \{\text{Kreuz}, \text{Pik}, \text{Herz}, \text{Karo}\},$$
$$\text{Karte} = \{2, 3, 4, 5, 6, 7, 8, 9, 10, \text{Bube}, \text{Dame}, \text{König}, \text{Ass}\}$$

Wie können wir diese Relation als Tabelle in SQL speichern?

Full House

Farbe	Karte
Pik	Ass
Kreuz	Ass
Herz	Ass
Herz	König
Karo	König

Beispiel: Full House auf der Hand

- Den Relationentyp Full House implementieren wir als Tabelle:

```
CREATE TABLE full_house (
  farbe varchar(5),
  karte varchar(5)
) DEFAULT CHARSET=utf8;
```

- Daten einfügen:

```
INSERT INTO full_house (farbe,karte) VALUES
('Kreuz','Ass'), ('Pik','Ass'), ('Herz','König'), ('Herz','Ass'), ('Karo','König');
```

  werden die Daten darin gespeichert.

- Daten ansehen:

`SELECT * FROM full_house;`

farbe	karte
Kreuz	Ass
Pik	Ass
Herz	König
Herz	Ass
Karo	König

`SELECT farbe FROM full_house;`

farbe
Kreuz
Pik
Herz
Herz
Karo

`SELECT karte FROM full_house;`

karte
Ass
Ass
König
Ass
König

Inhaltsübersicht

Das war's für heute!

Wir haben heute behandelt:

- Konzepte zur Strukturierung und Speicherung von Information
- Relationen und relationale Datenbanken
- Einführung in SQL
- Implementierung von Relationen mit SQL

Haben Sie noch Fragen

Literatur

CODD, E. F. (1970): A relational model of data for large shared data banks, in: Communications of the ACM, 13(6), S. 387, DOI 10.1145/357980.358007.

DE VRIES, A. (2019): Algorithmik, Vorlesungsskript, Hagen, `https://www4.fh-swf.de/media/downloads/fbtbw/download_8/devries_1/algorithmik.pdf`.

KRCMAR, H. (2015): Informationsmanagement, 6. Aufl., Springer Gabler, Berlin Heidelberg, DOI 10.1007/978-3-662-45863-1.

PIEPMEYER, L. (2011): Grundkurs Datenbanksysteme, Carl Hanser Verlag, München Wien.

ZIMMERMANN, M. (1993): 'Das Nervensystem – nachrichtentechnisch gesehen', in: SCHMIDT, R. F./THEWS, G. (Hrsg.), Physiologie des Menschen, S. 176–183, Springer Verlag, Berlin Heidelberg.