

Datenbanken und Informationssysteme

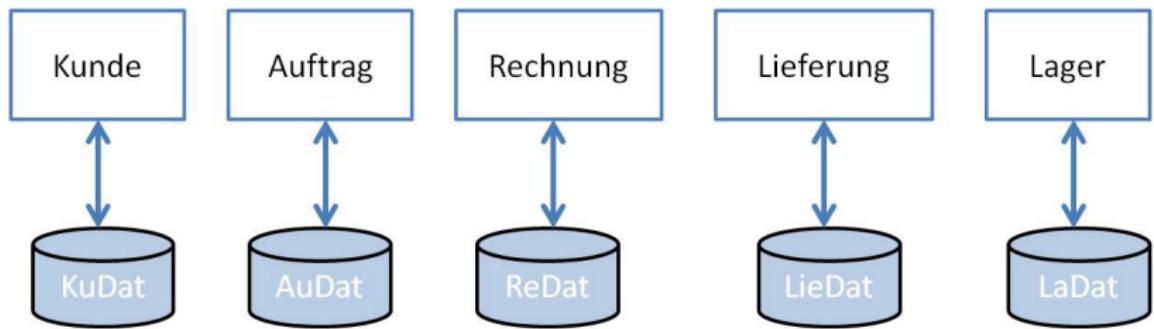
Georg Lausen

Lehrstuhl für Datenbanken und Informationssysteme
Universität Freiburg

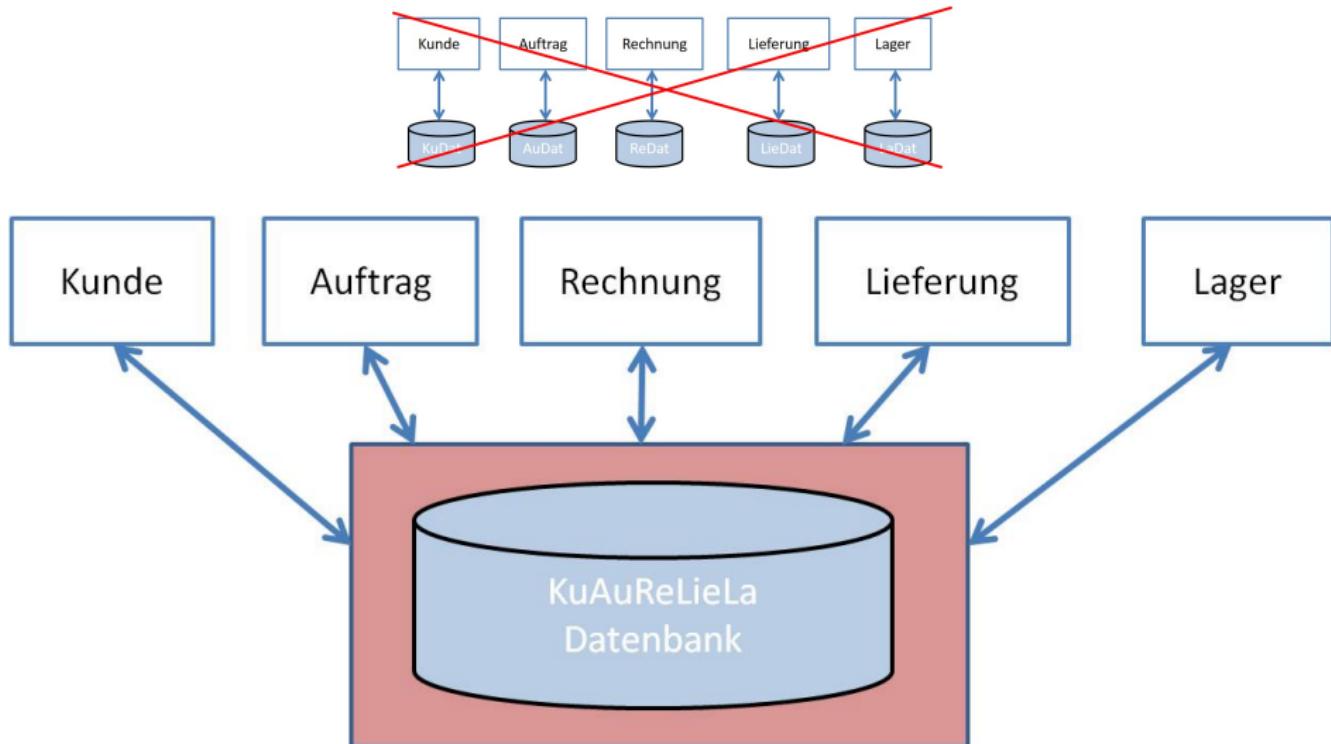
WS 2017/2018

Datenbanken warum?

Konventionelle Datenverarbeitung kommt mit vielen Problemen:



... Datenbanken sind die Lösung!



Gliederung:

- (1) Einführung
- (2) Grundlagen von Anfragesprachen
- (3) Einstieg in SQL
- (4) Der SQL-Standard
- (5) Konzeptueller Datenbankentwurf
- (6) Formaler Datenbankentwurf
- (7) Physischer Datenbankentwurf
- (8) Auswertung von Anfrageoperatoren
- (9) Transaktionen

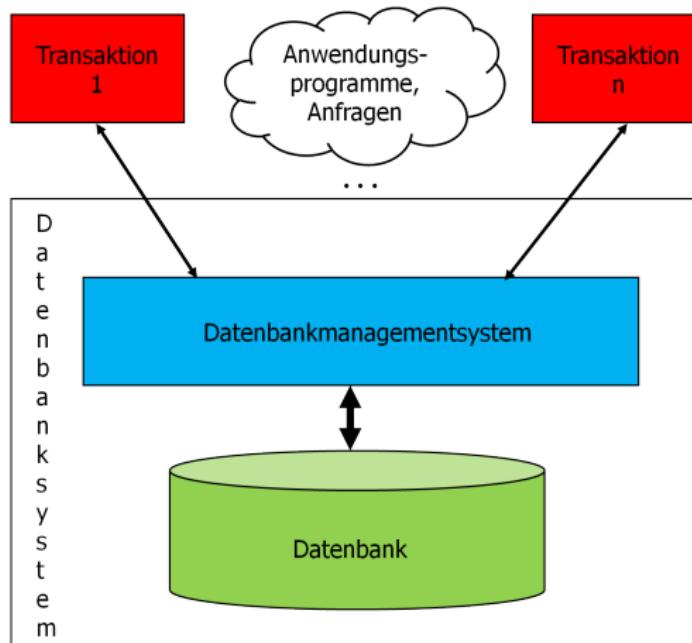
Literatur:

- ▶ *Datenbanken: Grundlagen und XML-Technologien.* Georg Lausen. Elsevier Spektrum Akademischer Verlag, 2005.
- ▶ *Datenbanken - Konzepte und Sprachen.* Andreas Heuer, Gunter Saake. International Thomson Publishing, 2. Auflage, 2000.
- ▶ *Datenbanksysteme - Eine Einführung.* Alfons Kemper, Andreas Eickler. Oldenbourg, 7. Auflage, 2009.
- ▶ *Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme.* Gottfried Vossen. Oldenbourg, 5. Auflage, 2008.

- ▶ *Database Management Systems.* Raghu Ramakrishnan, Johannes Gehrke. McGraw-Hill Higher Education, 3rd. Edition.

Kapitel 1: Einführung

1.1 Datenbanken?



1.2 Grundbegriffe relationaler Datenbanken

Repräsentieren von Daten einer Miniwelt



Studierenden-Datenbank

- ▶ Hans Eifrig hat die Matrikelnummer 1223. Seine Adresse ist Seeweg 20. Er ist im zweiten Semester.
- ▶ Lisa Lustig hat die Matrikelnummer 3434. Ihre Adresse ist Bergstraße 11. Sie ist im vierten Semester.
- ▶ Maria Gut hat die Matrikelnummer 1234. Ihre Adresse ist Am Bächle 1. Sie ist im zweiten Semester.

Repräsentation und Strukturierung von Daten in Relationen

Student

<u>MatrNr</u>	Name	Adresse	Semester
1223	Hans Eifrig	Seeweg 20	2
3434	Lisa Lustig	Bergstraße 11	4
1234	Maria Gut	Am Bächle 1	2

- ▶ Tabellarische Darstellungen dieser Art sind die Grundstrukturen *relationaler Datenbanken*.
- ▶ Begriffe: *Relation*, *Relationsbezeichner*, *Attribut*, *Tupel*, *Schlüssel*, *Primärschlüssel*.
oder auch *Tabelle*, *Tabellenbezeichner*, *Spalte*, *Zeile*.
- ▶ Ein *Schlüssel* einer Relation ist eine minimale Teilmenge der Attribute der Relation, mittels der die einzelnen Tupel der Relation unterschieden werden können.
Der Primärschlüssel ist ein ausgewählter Schlüssel - er wird durch Unterstrichen gekennzeichnet.

Struktur und Inhalt einer Relation

Die abstrakte Struktur einer Relation soll von ihrem Inhalt getrennt werden.

- ▶ *Relationsschema*. Struktur der Relation Student:

Student(MatrNr, Name, Adresse, Semester)

- ▶ *Relationsinstanz*. Inhalt/Zustand der Relation mit Schema Student:

Student

<u>MatrNr</u>	Name	Adresse	Semester
1223	Hans Eifrig	Seeweg 20	2
3434	Lisa Lustig	Bergstraße 11	4
1234	Maria Gut	Am Bächle 1	2

- ▶ Die identifizierende Eigenschaft des Primärschlüssels muss für jede Instanz der Relation gewährleistet sein.

Beispiel Student, Kurs und Belegung

Student

<u>MatrNr</u>	Name	Adresse	Semester
1223	Hans Eifrig	Seeweg 20	2
3434	Lisa Lustig	Bergstraße 11	4
1234	Maria Gut	Am Bächle 1	2

Kurs

<u>KursNr</u>	Institut	Name	Beschreibung
K010	DBIS	Datenbanken	Grundlagen von Datenbanken
K011	DBIS	Informationssysteme	Grundlagen von Informationssystemen

Belegung

<u>MatrNr</u>	<u>KursNr</u>	Semester	Note
1223	K010	WS2003/2004	2.3
1234	K010	SS2004	1.0

Datenbankschema und Datenbankinstanz

- ▶ Die Menge der Relationsschemata einer Miniwelt ergibt das (relationale) *Datenbankschema* der Miniwelt.
- ▶ Eine Menge von Instanzen der Relationen eines Datenbankschemas nennen wir *Datenbankinstanz*.
Die Elemente einer Datenbankinstanz müssen sich auf denselben Zustand der betreffenden Miniwelt beziehen.

1.3 Arbeiten mit einer Datenbanken

- ▶ Anwendungsprogramme (Transaktionen) kommunizieren mit einer Datenbank, indem sie Anfragen über den gespeicherten Zustand der Miniwelt stellen, bzw. diesen Zustand durch Ändern, Einfügen oder Löschen von Daten verändern.

Besonders von Interesse sind *Anfragen* (engl. queries) an eine Datenbank.

- ▶ Ausdrücke einer *Datenbankanfragesprache* haben eine *mengenwertige, deklarative Semantik*.
 - ▶ Das Ergebnis einer Anfrage ist eine Menge von Tupeln.
 - ▶ Die Anfrage definiert, was für Zusammenhänge aus den Daten der Datenbank gebildet werden sollen, ohne dass die algorithmische Vorgehensweise hierzu spezifiziert werden muss.
 - ▶ Es existiert eine standardisierte Anfragesprache: SQL.
- ▶ Anfrageoptimierer.

Anfragen in SQL, der Anfragesprache für Datenbanken:

Was wissen wir über die Studierenden?

```
SELECT *  
FROM Student
```

Welche Matrikelnummern haben die Studierenden, die den Kurs 'K010' belegt haben?

```
SELECT MatrNr  
FROM Belegung  
WHERE KursNr = 'K010'
```

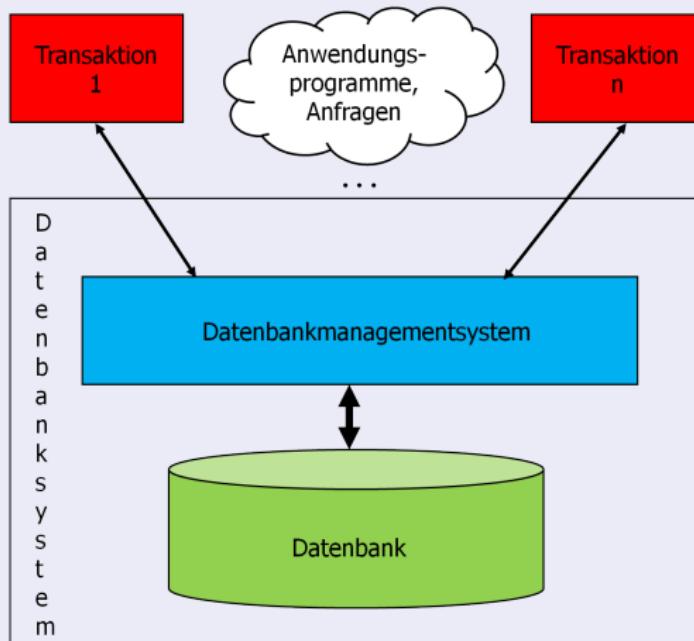
Wie heißen die Studierenden, die den Kurs 'K010' belegt haben?

```
SELECT S.Name  
FROM Student, Belegung  
WHERE Student.MatrNr = Belegung.MatrNr AND Belegung.KursNr = 'K010'
```

Welche Studierenden belegen welche Kurse?

```
SELECT S.Name, K.Name  
FROM Student S, Belegung B, Kurs K  
WHERE S.MatrNr = B.MatrNr AND B.KursNr = K.KursNr
```

1.4 Basisarchitektur



1.5 Diskussion

Abkürzungen

- ▶ - Datenbank: DB
- ▶ - Datenbanksystem: DBS
- ▶ - Datenbankmanagementsystem: DBMS

Vorteile eines DBS

- ▶ *Mehrfachnutzung der Daten.* Dateninseln, in denen einzelne Benutzer ihre Daten isoliert verwalten werden vermieden. Somit kann Redundanz der Daten ausgeschlossen werden und eine Standardisierung der Datenformate erreicht werden.
- ▶ *Unabhängigkeit von Programmen und Daten.* Programme kommunizieren mit der DB über vom DBMS angebotene Interfaces; Änderungen in den Strukturen der DB können vor den Programmen weitgehend verborgen werden.
- ▶ *Gewährleistete Datenintegrität.* Das DBMS hat das Wissen über die zulässigen Zustände der DB und kann Verletzungen der Integrität ausschließen.

weitere Vorteile eines DBS

- ▶ *Kontrollierter Mehrbenutzerbetrieb.* Zeitlich überlappender Zugriff unterschiedlicher Programme zu gemeinsamen Daten kann durch das DBMS kontrolliert werden.
- ▶ *Datenpersistenz.* Datenverlust durch Hardware- und Softwarefehler kann im laufenden Betrieb repariert werden; das DBMS verwaltet selbst die hierzu erforderlichen Informationen.
- ▶ *Datensichten.* Unterschiedliche Benutzergruppen benötigen auf ihre Bedürfnisse abgestimmte Sichten auf das zentrale konzeptuelle Schema. Das DBMS kann die erforderlichen Abbildungen zur Verfügung stellen.
- ▶ *Autorisierter Datenzugriff.* Zugriff zu den Daten ist nur über das DBMS möglich. Damit kann nicht autorisierter Zugriff vermieden werden.

Nachteile eines DBS

- ▶ *Komplexität.* Die effiziente und sichere Nutzung erfordert speziell ausgebildetes Personal und verursacht erhebliche Hardware- und Software-Kosten.
- ▶ *eingeschränkte Effizienz.* DBS sind universelle Softwaresysteme, die für unterschiedlichste Arten von Anwendungen die geforderte Leistung zur Verfügung stellen wollen. Für spezielle Anwendungen kann ihre Leistungsfähigkeit deutlich geringer sein im Vergleich zu für diese Anwendungen spezialisierte Systeme. Die Diskussionen hierzu laufen unter dem Stichwort *NoSQL*.

NoSQL-Systeme erreichen die Effizienzsteigerung typischerweise durch Aufgabe von für Datenbanken existentieller Eigenschaften wie die Gewährleistung der Datenintegrität oder der kontrollierte Mehrbenutzerbetrieb.

Die Lösung: NewSQL

Basierend auf modernen Multicore-Systemen mit sehr großem Interspeichern (> 1 TB) oder Computer-Clustern holen aktuell sogenannte *NewSQL*-Systeme wieder auf, wobei sie im Unterschied zu NoSQL die volle Funktionalität bieten.

empfohlene Lektüre

Technology | DOI:10.1145/1735223.1735231

Gary Antes

Happy Birthday, RDBMS!

The relational model of data management, which dates to 1970, still dominates today and influences new paradigms as the field evolves.

FORTY YEARS AGO this June, an article appeared in these pages that would shape the long-term direction of information technology like few other ideas in computer science. The opening sentence of the article, "A Relational Model of Data for Large Shared Data Banks," summed it up in a way as simple and elegant as the model itself: "Future users of large data banks must be protected from having to know how the data is organized in the machine," wrote Edgar F. Codd, a researcher at IBM.

And protect them it did. Programmers and users at the time dealt mostly with crude homegrown database systems or commercial products like IBM's Information Management Systems (IMS), which was based on a low-level, hierarchical data model. "These databases were very rigid, and they were hard to understand," recalls Ronald Fagin, a Codd protégé and now a computer scientist at IBM Almaden Research Center. The hierarchical "trees" in IMS were brittle. Adding a single data element, a common occurrence, or even tuning changes, could involve major reprogramming. In addition, the programming language



"People were stunned to learn that complex, page-long [IMS] queries could be done in a few lines of a relational language," says Raghu Ramakrishnan, chief scientist for audience and cloud computing at Yahoo! Codd's model came to dominate a multibillion-dollar database market, but it was hardly an overnight success. The model was just too simple to work, some said. And even if it did

management System (IDMS) from the company that would become Cullinet.

Contentious debates raged over the models in the CS community through much of the 1970s, with relational enthusiasts arrayed against CODASYL advocates while IMS users coasted along on waves of legacy software.

As brilliant and elegant as the relational model was, it might have remained confined to computer science curricula if it wasn't for three projects aimed at real-world implementation of the relational database management system (RDBMS). In the mid-1970s, IBM's System R project and the University of California at Berkeley's Ingres project set out to translate the relational concepts into workable, maintainable, and efficient computer code. Support for multiple users, locking, logging, error-recovery, and more were developed.

System R went after the lucrative mainframe market with what would become DB2. In particular, System R produced the Structured Query Language (SQL), which became the de facto standard language for relational databases. Meanwhile, Ingres was aimed at UNIX machines and Digital Equipment Corp.

DOI:10.1145/2366316.2366319

<http://cacm.acm.org/blogs/blog-cacm>

New Opportunities for New SQL

Michael Stonebraker expects a substantial increase in the number of New SQL engines using a variety of architectures in the near future.



Michael Stonebraker
"New SQL:
An Alternative to
NoSQL and Old SQL
for New OLTP Apps"
<http://cacm.acm.org/>
blogs.cacm.acm.org/
June 16, 2011

connected to one or more data warehouses is the gold standard in enterprise computing. I will term it "Old OLTP." By and large, this activity was supported by the traditional RDBMS vendors. In the past I have affectionately called them "the elephants"; in this posting I refer to them as "Old SQL."

As noted by most pundits, "the Web changes everything," and I have noticed a very different collection of OLTP requirements that are emerging for Web properties, which I will term "New OLTP." These sites seem to be driven by two customer requirements:

The need for far more OLTP throughput. Consider new Web-based applications such as multiplayer games, social networking sites, and online gambling networks. The aggregate number of interactions per second is skyrocketing for the successful Web properties in this category. In addition, the explosive growth of smartphones has created a market for applications that use the phone as a geographic sensor and provide location-based services. Again, successful applications are seeing explosive growth in transaction requirements. Hence, the Web and smartphones are driving the volume of

interactions with a DBMS through the roof, and New OLTP developers need vastly better DBMS performance and enhanced scalability.

The need for real-time analytics. Intermixed with a tidal wave of updates is the need for a query capability. For example, a Web property wants to know the number of current users playing its game, or a smartphone user wants to know "What is around me?" These are not the typical BI requests to consolidated data, but rather real-time inquiries to current data. Hence, New OLTP requires a real-time query capability.

In my opinion, these two characteristics are shared by quite a number of enterprise non-Web applications. For example, electronic trading firms often trade securities in several locations around the world. The enterprise wants to keep track of the global position for each security. To do so, all trading actions must be recorded, creating a fire hose of updates. Furthermore, there are occasional real-time queries. Some of these are triggered by risk exposure—i.e., alert the CEO if the aggregate risk for or against a particular security exceeds a certain monetary threshold. Others come from humans, e.g., "What is the current position of the firm with respect to security X?"

Hence, we expect New OLTP to be a substantial application area, driven by Web applications as the early adopters. These applications will be followed by more traditional enterprise systems. Let's look at the deployment options.

1. Traditional OLTP. This architec-

¹ In: Communications of the ACM, Volume 53 , Issue 5 (May 2010). Kann aus dem Institutsnetz heraus vom ACM-Portal heruntergeladen werden.

² In: Communications of the ACM, Volume 55 , Issue 11 (May 2012). Kann aus dem Institutsnetz heraus vom ACM-Portal heruntergeladen werden.

Kapitel 2: Grundlagen von Anfragesprachen

Sprachparadigmen

- ▶ Relationenalgebra (in der Vorlesung)
- ▶ Relationenkalkül (siehe Literatur)

SQL - vorgestellt in den folgenden Kapiteln - basiert auf der Algebra und dem Kalkül gleichermaßen und enthält weitere für die praktische Anwendung sehr nützliche Sprachkonzepte.

Relationen dargestellt als Tabellen

Student

<u>MatrNr</u>	Name	Adresse	Semester
1223	Hans Eifrig	Seeweg 20	2
3434	Lisa Lustig	Bergstraße 11	4
1234	Maria Gut	Am Bächle 1	2

Kurs

<u>KursNr</u>	Institut	Name	Beschreibung
K010	DBIS	Datenbanken	Grundlagen von Datenbanken
K011	DBIS	Informationssysteme	Grundlagen von Informationssystemen

Belegung

<u>MatrNr</u>	<u>KursNr</u>	Semester	Note
1223	K010	WS2003/2004	2.3
1234	K010	SS2004	1.0

2.1 Das relationale Datenmodell

Attribute und Tupel

- ▶ Sei $X = \{A_1, \dots, A_k\}$ eine endliche *Attributmenge*, wobei $k \geq 1$.
- ▶ Jedes Attribut $A \in X$ besitzt einen nicht-leeren *Wertebereich* $\text{dom}(A)$.
- ▶ Die Vereinigung aller Wertebereiche ergibt sich dann zu $\text{dom}(X) = \cup_{A \in X} \text{dom}(A)$.
- ▶ Ein *Tupel* μ über Attributmenge X ist eine Abbildung

$$\mu : X \longrightarrow \text{dom}(X),$$

wobei $(\forall A \in X)\mu(A) \in \text{dom}(A)$.

- ▶ Sei $\text{Tup}(X)$ im folgenden die Menge aller Tupel über X .

Student

<u>MatrNr</u>	Name	Adresse	Semester
1223	Hans Eifrig	Seeweg 20	2
:	:	:	:

Tupel als Abbildungen versus Tupel als Vektoren

$\mu = \{\text{MatrNr} \rightarrow 1223, \text{Name} \rightarrow \text{Hans Eifrig},$
 $\text{Adresse} \rightarrow \text{Seeweg 20}, \text{Semester} \rightarrow 2\}$

$\mu' = \{\text{MatrNr} \rightarrow 1223, \text{Adresse} \rightarrow \text{Seeweg 20},$
 $\text{Semester} \rightarrow 2, \text{Name} \rightarrow \text{Hans Eifrig}\}$

$\mu_1 = (1223, \text{Hans Eifrig}, \text{Seeweg 20}, 2)$

$\mu'_1 = (1223, \text{Seeweg 20}, 2, \text{Hans Eifrig})$

$$\mu = \mu', \text{ aber } \mu_1 \neq \mu'_1.$$

Relation

- ▶ Eine *Relation* r über einer Attributmenge X ist eine *endliche* Menge $r \subseteq \text{Tup}(X)$.
- ▶ Sei R ein *Relationsbezeichner*.
Ein (*Relations*)-*Schema* zu R hat die Form $R(X)$. X ist hier eine endliche Attributmenge, das so genannte *Format* des Schemas.
Anstatt $R(\{A_1, \dots, A_k\})$ schreiben wir auch $R(A_1, \dots, A_k)$.
 k ist die *Stelligkeit* des Relationsbezeichners.
Auch:

$$R(A_1 : \text{dom}(A_1), \dots, A_k : \text{dom}(A_k))$$

Datenbank

- ▶ Ein (*relationales*) *Datenbank-Schema* \mathcal{R} ist gegeben durch eine Menge von (Relations-) Schemata,

$$\mathcal{R} := \{R_1(X_1), \dots, R_m(X_m)\},$$

bzw. $\mathcal{R} = \{R_1, \dots, R_m\}$.

- ▶ Eine *Instanz* \mathcal{I} zu einem relationalen Datenbankschema $\mathcal{R} = \{R_1, \dots, R_m\}$ ist eine Menge von endlichen Relationen $\mathcal{I} := \{r_1, \dots, r_m\}$, wobei $r_i \subseteq \text{Tup}(X_i)$ Instanz zu R_i für $1 \leq i \leq m$.

2.2 Relationenalgebra

Warum?

was berechnet dieser SQL-Ausdruck:

```
SELECT DISTINCT MatrNr FROM Belegung
MINUS
SELECT MatrNr FROM (
    SELECT MatrNr, KursNr FROM (
        (SELECT MatrNr FROM Belegung)
        CROSS JOIN
        (SELECT KursNr FROM Kurs) )
    MINUS
    SELECT MatrNr, KursNr FROM Belegung
)
```

... Notwendigkeit von Auswertungsoperatoren mit formal definierter Semantik!

Basisoperatoren der Relationenalgebra

- ▶ Attribute aus Relationen herausstreichen: *Projektion* π ,
- ▶ Tupel aus Relationen auswählen: *Selektion* σ ,
- ▶ Relationen miteinander verknüpfen: *Verbund* \bowtie ,
- ▶ Relationen wie Mengen verarbeiten: *Vereinigung* \cup , *Differenz* $-$,
- ▶ Attribute umbenennen.

Beispiel Projektion einer Relation

$$r = \begin{array}{c} \begin{array}{ccc} A & B & C \\ \hline a & b & c \\ a & a & c \\ c & b & d \end{array} \end{array} \quad \pi[A, C]r = \begin{array}{c} \begin{array}{cc} A & C \\ \hline a & c \\ c & d \end{array} \end{array}$$

Projektion einer Relation

- Sei $r \subseteq \text{Tup}(X)$ eine Relation und $\emptyset \subset Y \subseteq X$.
- Der Ausdruck $\pi[Y]r$ heißt *Projektion* der Relation r auf Y . Es gilt:

$$\pi[Y]r := \{\mu \in \text{Tup}(Y) \mid \exists \mu' \in r, \text{ so dass } \mu(A) = \mu'(A), A \in Y\}.$$

Beispiel Selektion

	A	B	C
r	a	b	c
	d	a	f
	c	b	d

$$\sigma[B = b]r =$$

	A	B	C
r	a	b	c
	c	b	d

Selektion

- Sei $r \subseteq \text{Tup}(X)$ eine Relation und α eine Selektionsbedingung zu X .
- Der Ausdruck $\sigma[\alpha]r$ heißt *Selektion* der Relation r bezüglich α . Es gilt:

$$\sigma[\alpha]r := \{\mu \in \text{Tup}(X) \mid \mu \in r \wedge \mu \text{ erfüllt } \alpha\}.$$

Selektionsbedingung

- ▶ Seien $A, B \in X$, $a \in \text{dom}(A)$ und sei $\theta \in \{=, \neq, <, \leq, >, \geq\}$ ein arithmetischer Vergleichsoperator.
- ▶ Eine (atomare) *Selektionsbedingung* α (bezüglich X) ist ein Ausdruck der Form $A \theta B$, bzw. $A \theta a$, bzw. $a \theta A$.
- ▶ Ein Tupel $\mu \in \text{Tup}(X)$ erfüllt eine Selektionsbedingung α , wenn gerade $\mu(A) \theta \mu(B)$, bzw. $\mu(A) \theta a$, bzw. $a \theta \mu(A)$.
- ▶ Atomare Selektionsbedingungen können mittels \wedge , \vee , \neg , $(,)$ zu Formeln verallgemeinert werden.

Beispiel

$$X = \{A, B, C\}.$$

$$\mu_1 = (A \rightarrow 2, B \rightarrow 2, C \rightarrow 1), \quad \mu_2 = (A \rightarrow 2, B \rightarrow 3, C \rightarrow 2)$$

$$\alpha_1 = (A = B),$$

$$\alpha_2 = ((B > 1) \wedge (C > 1))$$

Welche Tupel erfüllen welche Selektionsbedingungen?

Beispiel Vereinigung und Differenz

A	B	C
a	b	c
d	a	f
c	b	d

A	B	C
b	g	a
d	a	f

$r \cup s =$

A	B	C
a	b	c
d	a	f
c	b	d
b	g	a

A	B	C
a	b	c
d	a	f
c	b	d

A	B	C
b	g	a
d	a	f

$r - s =$

A	B	C
a	b	c
c	b	d

Vereinigung \cup und Differenz –

- ▶ Seien X, Y Attributmengen, wobei $X = Y$ und seien weiter $r \subseteq \text{Tup}(X), s \subseteq \text{Tup}(Y)$ zwei entsprechende Relationen.
- ▶

$$r \cup s = \{\mu \in \text{Tup}(X) \mid \mu \in r \vee \mu \in s\}.$$

$$r - s = \{\mu \in \text{Tup}(X) \mid \mu \in r, \text{ wobei } \mu \notin s\}.$$

Beispiel Verbund

$$r = \begin{array}{c|ccc} & A & B & C \\ \hline 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 6 \end{array}$$

$$s = \begin{array}{c|cc} & C & D \\ \hline 3 & 1 \\ 6 & 2 \\ 4 & 5 \end{array}$$

$$r \bowtie s =$$

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 2 \\ 7 & 8 & 6 & 2 \end{array}$$

Sei $\mu \in \text{Tup}(X)$ ein Tupel über X , $Y \subseteq X$.

$$\mu = \{A \rightarrow 1, B \rightarrow 2, C \rightarrow 3, D \rightarrow 1\}$$

$$Y = \{B, D\}.$$

$$\mu[Y] = \{B \rightarrow 2, D \rightarrow 1\}$$

Der Ausdruck $\mu[Y]$ heißt *Projektion* des Tupels μ auf Y . Es gilt:

$$\mu[Y] \in \text{Tup}(Y),$$

wobei $\mu[Y](A) = \mu(A)$, $A \in Y$.

Beispiel Verbund

A	B	C
1	2	3
4	5	6
7	8	6

C	D
3	1
6	2
4	5

$$r \bowtie s =$$

A	B	C	D
1	2	3	1
4	5	6	2
7	8	6	2

Verbund

- ▶ Seien X, Y Attributmengen; XY sei im Folgenden eine Kurzschreibweise für $X \cup Y$.
- ▶ Seien weiter $r \subseteq \text{Tup}(X), s \subseteq \text{Tup}(Y)$ zugehörige Relationen.
- ▶ Der (*natürliche*) *Verbund* \bowtie von r und s ist dann definiert:

$$r \bowtie s := \{\mu \in \text{Tup}(XY) \mid \mu[X] \in r \wedge \mu[Y] \in s\}.$$

Der (*natürliche*) **Verbund** \bowtie von r und s ist definiert:

$$r \bowtie s := \{\mu \in \text{Tup}(XY) \mid \mu[X] \in r \wedge \mu[Y] \in s\}.$$

Verbund fortgesetzt

Seien X_i , $1 \leq i \leq n$ Formate.

- ▶ $\bowtie_{i=1}^n r_i := \{\mu \in \text{Tup}(\cup_{i=1}^n X_i) \mid \mu[X_i] \in r_i, 1 \leq i \leq n\}.$

- ▶ Sei $X_1 \cap X_2 = \emptyset$ und bezeichne \times das kartesische Produkt.

$$r_1 \times r_2 = r_1 \bowtie r_2.$$

Beispiel Umbenennung

$X = \{A, B, C\}$, $Y = \{D, E, C\}$ und $\delta = \{A \rightarrow D, B \rightarrow E, C \rightarrow C\}$.

$$r = \begin{array}{c} \begin{array}{ccc} A & B & C \\ \hline a & b & c \\ d & a & f \\ c & b & d \end{array} \end{array}$$

$$\delta[X, Y]r = \begin{array}{c} \begin{array}{ccc} D & E & C \\ \hline a & b & c \\ d & a & f \\ c & b & d \end{array} \end{array}$$

Umbenennung

- ▶ Seien $X = \{A_1, \dots, A_k\}$, $Y = \{B_1, \dots, B_k\}$ Formate.
- ▶ Sei δ eine umkehrbar eindeutige Abbildung von X nach Y , wobei $\text{dom}(A) = \text{dom}(\delta(A))$. Gilt $\delta(A) = B$, so schreiben wir $A \rightarrow B$.
- ▶ Sei $r \subseteq \text{Tup}(X)$ eine Relation zu X .
- ▶ Die Umbenennung $\delta[X, Y]$ bezüglich r ist wie folgt:

$$\delta[X, Y]r := \{\mu \in \text{Tup}(Y) \mid \exists \mu' \in r, \text{ so dass } \mu'(A_i) = \mu(\delta(A_i)), 1 \leq i \leq k\}$$

Basisoperatoren

- ▶ Selektion, Projektion, Vereinigung, Differenz, Verbund und Umbenennung sind die Basisoperatoren der Relationenalgebra.
- ▶ Die Anwendung dieser Operatoren auf Relationen liefert als Ergebnis wiederum eine Relation.
- ▶ Die zulässigen Ausdrücke der Relationenalgebra können ausgehend von den Basisoperatoren induktiv definiert werden.
- ▶ Wir können andere nützliche Operatoren definieren.

weitere Operatoren

Seien X_i , $1 \leq i \leq n$, Formate und seien $r_i \subseteq \text{Tup}(X_i)$, $1 \leq i \leq n$, Relationen.

- ▶ *Durchschnitt*. Sei $X_1 = X_2$.

$$r_1 \cap r_2 := r_1 - (r_1 - r_2).$$

Bemerkung: Da $X_1 = X_2$ gilt $r_1 \cap r_2 = r_1 \bowtie r_2$.

- ▶ *θ -Verbund*. Sei $X_1 \cap X_2 = \emptyset$ und sei α eine beliebige Selektionsbedingung über $X_1 \cup X_2$.

$$r \bowtie_\alpha s := \sigma[\alpha](r \bowtie s).$$

Enthält α ausschließlich Gleichheitsvergleiche, dann redet man von einem *Equi-Verbund*.

Beispiel Division

$$\begin{array}{c}
 \begin{array}{cccc}
 A & B & C & D \\
 \hline
 a & b & c & d \\
 a & b & e & f \\
 r_1 = & b & c & e & f \\
 & e & d & c & d \\
 & e & d & e & f \\
 a & b & d & d
 \end{array} &
 r_2 = \frac{\begin{array}{cc} C & D \\ c & d \\ e & f \end{array}}{\begin{array}{cc} A & B \\ a & b \\ e & d \end{array}} &
 r_1 \div r_2 = \frac{\begin{array}{cc} A & B \\ a & b \\ e & d \end{array}}{\begin{array}{cc} A & B \\ a & b \\ e & d \end{array}}
 \end{array}$$

Division

Seien X_1, X_2 Formate, $X_2 \subset X_1$, $Z = X_1 - X_2$ und weiter $r_1 \subseteq \text{Tup}(X_1)$ (Dividend), $r_2 \subseteq \text{Tup}(X_2)$ (Divisor), wobei $r_2 \neq \emptyset$.

$$\begin{aligned}
 r_1 \div r_2 &:= \{\mu \in \text{Tup}(Z) \mid \{\mu\} \times r_2 \subseteq r_1\} \\
 &= \pi[Z]r_1 - \pi[Z](\pi[Z](\pi[Z]r_1) \times r_2) - r_1.
 \end{aligned}$$

Beispiel: Welche Studierenden belegen alle Kurse?

Kurs(KursNr, Institut, Name, Beschreibung)

Belegung(MatrNr, KursNr, Semester, Note)

$$\pi[\text{MatrNr}, \text{KursNr}] \text{Belegung} \div \pi[\text{KursNr}] \text{Kurs}$$

empfohlene Lektüre

The 1981 ACM Turing Award Lecture

Delivered at ACM '81, Los Angeles, California, November 9, 1981



The 1981 ACM Turing Award was presented to Edgar F. Codd, an IBM Fellow of the San Jose Research Laboratory, by President Peter Denning on November 9, 1981 at the ACM Annual Conference in Los Angeles, California. It is the Association's foremost award for technical contribution to the computing field.

Codd was selected by the ACM General Technical Achievement Award Committee for his "fundamental and卓著的 contributions to the theory and practice of database management systems." The originator of the relational model for databases, Codd has made further important contributions in the development of relational algebra, relational calculus, and normalization of relations.

Edgar F. Codd joined IBM in 1949 to prepare programs for the Selective Sequence Electronic Calculator. Since then, his work in computing has encompassed logical design of computers (IBM 701 and Stretch), managing a computer center in Canada, heading the development of one of the first operating systems with a modular design, writing one of the first books on the topic of self-reproducing automata, developing high level techniques for software specification and synthesizing subsystem for causal users of relational databases. He is also the author of *Cellular Automata*, an early volume in the ACM Monograph Series.

Codd received his B.A. and M.A. in Mathematics from Oxford University in England, and his M.Sc. and Ph.D. in Computer and Communication Sciences from the University of Michigan. He is a Member of the National Academy of Engineering (USA) and a Fellow of the British Computer Society.

The ACM Turing Award is presented each year in commemoration of A. M. Turing, the English mathematician who made major contributions to the computing sciences.

Relational Database: A Practical Foundation for Productivity

E. F. Codd
IBM San Jose Research Laboratory

It is well known that the growth in demands from end users for new applications is outstripping the capability of data processing systems to handle the corresponding application programs. There are two complementary approaches to attacking this problem (and both approaches are needed): one is to put end users into direct touch with the information stored in computers; the other is to increase the productivity of data processing professionals in the development of application programs. It is less well known that a single technology,

relational database management, provides a practical foundation for both approaches. It is explained why this is so.

While developing this productivity theme, it is noted that the time has come to draw a very sharp line between relational and non-relational database systems, so that the label "relational" will not be used in misleading ways. The key to drawing this line is something called a "relational processing capability."

CR Categories and Subject Descriptors: H.2.0 [Database Management]; General; H.2.1 [Database Management]; Logical Design—data models; H.2.4 [Database Management]; Systems

General Terms: Human Factors, Languages

Additional Key Words and Phrases: database, relational database, relational model, data structure, data manipulation, data integrity, productivity

Author's Present Address: E. F. Codd, IBM Research Laboratory, 5600 La Jolla Village Drive, San Diego, CA 92121.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and this notice appear, and either this notice or the copyright notice is given when copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission. © 1982 ACM 0001-0782/82/0200-0109 \$00.75

Kapitel 3: Einstieg in SQL

- ▶ SQL (Structured Query Language) ist die in der Praxis am weitesten verbreitete Datenbanksprache für relationale Datenbanken.
- ▶ Die Historie von SQL geht zurück bis 1974, die Anfangszeit der Entwicklung relationaler Datenbanken.
- ▶ Alles begann mit SEQUEL, der *Structured English Query Language*.
- ▶ Der Sprachumfang von SQL ist einer permanenten Weiterentwicklung und Standardisierung unterworfen. Derzeit relevant sind der Stand von 1992, 1999, 2003, 2008, 2011 und 2016 entsprechend bezeichnet mit SQL-92, SQL:1999, SQL:2003, SQL:2008, SQL:2011 und SQL:2016.

3.1 Beispiele-Datenbank

Mondial-Datenbank Teil 1

Land

<u>LName</u>	<u>LCode</u>	<u>HStadt</u>	<u>Fläche</u>
Austria	A	Vienna	84
Egypt	ET	Cairo	1001
France	F	Paris	547
Germany	D	Berlin	357
Italy	I	Rome	301
Russia	RU	Moscow	17075
Switzerland	CH	Bern	41
Turkey	TR	Ankara	779

Provinz

<u>PName</u>	<u>LCode</u>	<u>Fläche</u>
Baden	D	15
Bavaria	D	70,5
Berlin	D	0,9
Ile de France	F	12
Franken	D	null
Lazio	I	17

Stadt

<u>SName</u>	<u>LCode</u>	<u>PName</u>	<u>Einwohner</u>	<u>LGrad</u>	<u>BGrad</u>
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

Mondial-Datenbank Teil 2

Lage

<u>LCode</u>	<u>Kontinent</u>	<u>Prozent</u>
D	Europe	100
F	Europe	100
TR	Asia	68
TR	Europe	32
ET	Africa	90
ET	Asia	10
RU	Asia	80
RU	Europe	20

Mitglied

<u>LCode</u>	<u>Organisation</u>	<u>Art</u>
A	EU	member
D	EU	member
D	WEU	member
ET	UN	member
I	EU	member
I	NAM	guest
TR	UN	member
TR	CERN	observer

3.2 Einfache Anfragen

Ein *Anfrageausdruck* in SQL besteht aus einer SELECT-Klausel, gefolgt von einer FROM-Klausel, gefolgt von einer WHERE-Klausel.

SFW-Ausdruck

```
SELECT A1, ..., An (...Attribute der Ergebnisrelation)
  FROM R1, ..., Rm (...benötigte Relationen)
 WHERE F          (...Auswahlbedingung)
```

Anfragen über einer Relation

(a) gesamter Inhalt

Gib den vollständigen Inhalt der Tabelle Stadt.

```
SELECT * FROM Stadt
```

Stadt

SName	LCode	PName	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

(b) einzelne Spalten (Projektion)

In welchen Provinzen liegen die Städte der Tabelle Stadt?

```
SELECT PName FROM Stadt
```

PName
Berlin
Baden
Baden
Bavaria
Franken
Ile de France
Lazio

In welchen *unterschiedlichen* Provinzen liegen die Städte der Tabelle Stadt?

```
SELECT DISTINCT PName FROM Stadt
```

PName
Berlin
Baden
Bavaria
Franken
Ile de France
Lazio

(c) einzelne Zeilen (Selektion)

Stadt

<u>SName</u>	<u>LCode</u>	<u>PName</u>	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

Wie heißen die Städte, die mehr als 1 Mio. Einwohner haben?

```
SELECT * FROM Stadt
WHERE Einwohner > 1000
```

Stadt

<u>SName</u>	<u>LCode</u>	<u>PName</u>	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Munich	D	Bavaria	1244	11,56	48,15
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

(d) geänderte Spaltenbezeichnungen und neue Spalten

Stadt

SName	LCode	PName	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

Kennzeichne die Tatsache, dass eine Stadt mehr als 1 Mio. Einwohner hat, durch den Wert 'Großstadt' einer neuen Spalte mit Namen `StadtKategorie`; die Spalte `SName` soll die Bezeichnung `Stadt` erhalten.

```
SELECT SName AS Stadt, 'Großstadt' AS StadtKategorie FROM Stadt
WHERE Einwohner > 1000
```

Stadt	StadtKategorie
Berlin	Großstadt
Munich	Großstadt
Paris	Großstadt
Rome	Großstadt

(e) Pattern Matching (1)

Land			
LName	LCode	HStadt	Fläche
Austria	A	Vienna	84
Egypt	ET	Cairo	1001
France	F	Paris	547
Germany	D	Berlin	357
Italy	I	Rome	301
Russia	RU	Moscow	17075
Switzerland	CH	Bern	41
Turkey	TR	Ankara	779

Erstelle eine Namensliste der Länder, deren Namen mit 'G' anfängt oder mit 'y' aufhört?

```
SELECT LName FROM Land WHERE LName LIKE 'G%' OR LName LIKE '%y'
```

LName
Germany
Italy
Turkey

(e) Pattern Matching (2)

Land			
LName	LCode	HStadt	Fläche
Austria	A	Vienna	84
Egypt	ET	Cairo	1001
France	F	Paris	547
Germany	D	Berlin	357
Italy	I	Rome	301
Russia	RU	Moscow	17075
Switzerland	CH	Bern	41
Turkey	TR	Ankara	779

Erstelle eine Namensliste der Länder, deren dritter Buchstabe des Namens 'y' ist?

```
SELECT LName FROM Land WHERE LName LIKE '_ _y%'
```

LName
Egypt

Anfragen über mehreren Relationen

Anfragen mit mehreren Relationen in der `FROM`-Klausel sind sogenannte *Verbund*-Anfragen (engl. join-queries).

(a) einfacher Verbund

Land				Stadt							
LName	LCode	HStadt	Fläche	SName	LCode	PName	Einwohner	LGrad	BGrad	Land	Stadt
Austria	A	Vienna	84	Berlin	D	Berlin	3472	13,2	52,45	Germany	Berlin
Egypt	ET	Cairo	1001	Freiburg	D	Baden	198	7,51	47,59	Germany	Freiburg
France	F	Paris	547	Karlsruhe	D	Baden	277	8,24	49,03	Germany	Karlsruhe
Germany	D	Berlin	357	Munich	D	Bavaria	1244	11,56	48,15	Germany	Munich
Italy	I	Rome	301	Nuremberg	D	Franken	495	11,04	49,27	Germany	Nuremberg
Russia	RU	Moscow	17075	Paris	F	Ile de France	2125	2,48	48,81	France	Paris
Switzerland	CH	Bern	41	Rome	I	Lazio	2546	12,6	41,8	Italy	Rome
Turkey	TR	Ankara	779								

Gib zu jedem Land die zugehörigen Städte an.

```
SELECT Land.LName AS Land, Stadt.SName AS Stadt
  FROM Land, Stadt
 WHERE Land.LCode = Stadt.LCode
```

Was passiert, wenn man die Auswahlbedingung (WHERE) entfernt?

```
SELECT Land.LName AS Land, Stadt.SName AS Stadt  
FROM Land, Stadt
```

?

... man berechnet das kartesische Produkt der angegebenen Relationen!

(b) Korrelationsnamen

... Verwende optionale Korrelationsnamen.

```
SELECT S.SName, L.LName  
      FROM Stadt S, Land L  
     WHERE S.LCode = L.LCode
```

(c) Verbund mit Auswahlbedingung

Land

LName	<u>LCode</u>	HStadt	Fläche
Austria	A	Vienna	84
Egypt	ET	Cairo	1001
France	F	Paris	547
Germany	D	Berlin	357
Italy	I	Rome	301
Russia	RU	Moscow	17075
Switzerland	CH	Bern	41
Turkey	TR	Ankara	779

Stadt

SName	<u>LCode</u>	PName	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

Gib zu jedem Land die zugehörigen Städte mit mehr als 1 Mio Einwohner an.

```
SELECT L.LName AS Land, S.SName AS Stadt
  FROM Land L, Stadt S
 WHERE L.LCode = S.LCode AND S.Einwohner > 1000
```

Land	Stadt
Germany	Berlin
Germany	Munich
France	Paris
Italy	Rome

(d) Verbund einer Relation mit sich selbst

Lage		
<u>LCode</u>	<u>Kontinent</u>	Prozent
D	Europe	100
F	Europe	100
TR	Asia	68
TR	Europe	32
ET	Africa	90
ET	Asia	10
RU	Asia	80
RU	Europe	20

Bestimme alle Paare von Ländern, die in einem gemeinsamen Kontinent liegen.

```
SELECT L1.LCode AS Land1, L2.LCode AS Land2  
FROM Lage L1, Lage L2  
WHERE L1.Kontinent = L2.Kontinent  
AND L1.LCode < L2.LCode
```

(1) FROM Lage L1, Lage L2

Lage L1 (8 Tupel)

LCode	Kontinent	Prozent
D	Europe	100
F	Europe	100
TR	Asia	68
TR	Europe	32
ET	Africa	90
ET	Asia	10
RU	Asia	80
RU	Europe	20

 \times

Lage L2 (8 Tupel)

LCode	Kontinent	Prozent
D	Europe	100
F	Europe	100
TR	Asia	68
TR	Europe	32
ET	Africa	90
ET	Asia	10
RU	Asia	80
RU	Europe	20

 $=$ $L_1 \times L_2$ (64 Tupel)

L1.LCode	L1.Kontinent	L1.Prozent	L2.LCode	L2.Kontinent	L2.Prozent
D	Europe	100	D	Europe	100
D	Europe	100	F	Europe	100
...
D	Europe	100	RU	Asia	80
...
RU	Asia	20	D	Europe	100
...
RU	Europe	20	RU	Europe	20

(2) FROM Lage L1, Lage L2
 WHERE L1.Kontinent = L2.Kontinent

$L_1 \times L_2$ (64 Tupel)

L1.LCode	L1.Kontinent	L1.Prozent	L2.LCode	L2.Kontinent	L2.Prozent
D	Europe	100	D	Europe	100
D	Europe	100	F	Europe	100
...
D	Europe	100	RU	Asia	80
...
RU	Asia	20	D	Europe	100
...
RU	Europe	20	RU	Europe	20

↓
 (26 Tupel)

L1.LCode	L1.Kontinent	L1.Prozent	L2.LCode	L2.Kontinent	L2.Prozent
D	Europe	100	D	Europe	100
D	Europe	100	F	Europe	100
...
D	Europe	100	RU	Europe	20
...
RU	Europe	20	D	Europe	100
...
RU	Europe	20	RU	Europe	20

(3) FROM Lage L1, Lage L2

WHERE L1.Kontinent = L2.Kontinent
AND L1.LCode < L2.LCode

(9 Tupel)

L1.LCode	L1.Kontinent	L1.Prozent	L2.LCode	L2.Kontinent	L2.Prozent
D	Europe	100	F	Europe	100
ET	Asia	10	TR	Asia	68
RU	Asia	80	TR	Asia	68
D	Europe	100	TR	Europe	32
F	Europe	100	TR	Europe	32
RU	Europe	20	TR	Europe	32
ET	Asia	10	RU	Asia	80
D	Europe	100	RU	Europe	20
F	Europe	100	RU	Europe	20

```
(4) SELECT DISTINCT L1.LCode AS Land1,  
          L2.LCode AS Land2  
     FROM Lage L1, Lage L2  
    WHERE L1.Kontinent = L2.Kontinent  
  AND L1.LCode < L2.LCode
```

(8 Tupel)

Land1	Land2
D	F
ET	TR
RU	TR
D	TR
F	TR
ET	RU
D	RU
F	RU

(e) impliziter und expliziter Verbund

Implizit:

Gib zu jedem Land die zugehörigen Städte an.

```
SELECT S.SName, L.LName  
      FROM Stadt S, Land L  
     WHERE S.LCode = L.LCode
```

Explizit:

```
SELECT S.SName, L.LName  
      FROM Stadt S JOIN Land L  
        ON S.LCode = L.LCode
```

Wird Gleichheit über Attributen mit identischen Bezeichnern gefordert redet man von einem *natürlichen Verbund* (engl. natural join) und schreibt kürzer:

```
SELECT S.SName, L.LName  
      FROM Stadt S NATURAL JOIN Land L
```

Spezialfall *kartesisches Produkt*:

```
SELECT S.SName, L.LName  
      FROM Stadt S CROSS JOIN Land L
```

Auswertung einfacher SQL-Anfragen

SFW-Ausdruck

```
SELECT A1, ..., An (...Attribute der Ergebnisrelation)
  FROM R1, ..., Rm (...benötigte Relationen)
 WHERE F          (...Auswahlbedingung)
```

intuitive deklarative Semantik

Das Ergebnis besteht aus den Tupeln des kartesischen Produkts $R_1 \times \dots \times R_m$, die die Bedingung der WHERE-Klausel F erfüllen, wobei nur die Werte der Attribute A_1, \dots, A_n ausgegeben werden.

Nested-Loop-Semantik

```
FOR each Tupel t1 in Relation R1 DO
  FOR each Tupel t2 in Relation R2 DO
    :
    FOR each Tupel tm in Relation Rm DO
      IF WHERE-Klausel ist erfüllt nach Ersetzen der Attributnamen in F
        durch die entsprechenden Werte der gerade betrachteten Tupel t1, ..., tm.
      THEN Bilde ein Antwort-Tupel aus den Werten der in der
        SELECT-Klausel angegebenen Attributen A1, ..., An
        bezüglich der gerade betrachteten Tupel t1, ..., tm.
```

Sortierung

Stadt

<u>SName</u>	<u>LCode</u>	<u>PName</u>	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

Sortiere die Zeilen der Tabelle Stadt aufsteigend nach LCode und für gemeinsame Werte zu LCode absteigend nach dem Breitengrad.

```
SELECT * FROM Stadt
ORDER BY LCode DESC, BGrad ASC
```

<u>SName</u>	<u>LCode</u>	<u>PName</u>	Einwohner	LGrad	BGrad
Rome	I	Lazio	2546	12,6	41,8
Paris	F	Ile de France	2125	2,48	48,81
Freiburg	D	Baden	198	7,51	47,59
Munich	D	Bavaria	1244	11,56	48,15
Karlsruhe	D	Baden	277	8,24	49,03
Nuremberg	D	Franken	495	11,04	49,27
Berlin	D	Berlin	3472	13,2	52,45

3.3 Basis-Datentypen und Built-in Functions

Basis-Datentypen

SQL bietet eine Fülle von unterschiedlichen Datentypen an, mittels derer die Wertebereiche der Spalten einer Tabelle festgelegt werden können.

- ▶ INTEGER, SMALLINT
- ▶ NUMERIC, DECIMAL
Angabe Anzahl Ziffern insgesamt und Anzahl Kommastellen.
- ▶ REAL, DOUBLE PRECISION, FLOAT
- ▶ CHARACTER, CHARACTER VARYING, CHARACTER LARGE OBJECT
- ▶ BIT, BIT VARYING, BINARY LARGE OBJECT
- ▶ BOOLEAN
- ▶ DATE, TIME, TIMESTAMP, INTERVALL, . . .

Built-in Functions

Zur Manipulation der Werte der einzelnen Datentypen können spezielle Funktionen verwendet werden. Während der SQL-Standard hier sehr sparsam ist, bieten die einzelnen Hersteller von Datenbanksystemen einen großen Vorrat an. Die folgenden Beispiele basieren auf *Oracle Database SQL Reference 11g*. Es handelt sich um eine Auswahl sowohl der Funktionstypen als auch der jeweiligen Funktionen des jeweiligen Typs.

- ▶ Numeric:
ABS, ACOS, ASIN, ATAN, ATAN2, BITAND, CEIL, COS, COSH, EXP, FLOOR, LN, LOG, MOD, POWER, REMAINDER, ROUND, SIGN, SIN, SINH, SQRT, TAN, TANH, TRUNC
- ▶ Character:
CHR, CONCAT, INITCAP, LOWER, LPAD, LTRIM, REGEXP_REPLACE, REGEXP_SUBSTR, REPLACE, RPAD, RTRIM, SUBSTR, TRANSLATE, TREAT, TRIM, UPPER
ASCII, INSTR, LENGTH, REGEXP_INSTR
- ▶ Aggregate:
AVG, COLLECT, CORR, COUNT, FIRST, LAST, MAX, MEDIAN, MIN, RANK, REGRESSION, STATS_BINOMIAL_TEST, STDDEV, SUM, VARIANCE

Beispiele

- ▶

```
SELECT UPPER(LName)
FROM Land
```
- ▶

```
SELECT CONCAT(LName,LCode) AS LNameAndLCode
FROM Land
```
- ▶

```
SELECT CONCAT(CONCAT(LName, '>'),LCode) AS LNameAndLCode
FROM Land
```
- ▶

```
SELECT LName || '>' || LCode) AS LNameAndLCode
FROM Land
```
- ▶

```
SELECT Flaeche, LN(EXP(Flaeche)) AS Interessant
FROM Land WHERE Flaeche < 100
```
- ▶

```
SELECT A.SName, B.SName,
SQRT(power((A.BGrad-B.BGrad)*111.13,2) +
      power((A.LGrad-B.LGrad)*71.44,2)) AS Luli
FROM Stadt A, Stadt B
WHERE A.SName = 'Freiburg' AND B.SName = 'Karlsruhe'
```

CAST und CASE

- ▶ CAST erlaubt eine *explizite* Typkonversionen zwischen unterschiedlichen Typen;
- ▶ CASE beschränkt die Konversionen im Wesentlichen auf Wertumwandlungen innerhalb eines Typs.

Erstelle eine Tabelle mit einer Spalte LNameUndLCode. Zu jedem Land wird der Name auf die ersten zwei Zeichen reduziert und anstatt 'D' wird der Wert 'BRD' von LCode verwendet. Beide Werte sollen konkateniert werden getrennt durch '>'.

```
SELECT CONCAT(CONCAT(CAST(LName AS VARCHAR(2)), '>'),  
CASE LCode  
    WHEN 'D' THEN 'BRD'  
    ELSE LCode  
END) AS LNameAndLCode  
FROM Land
```

3.4 empfohlene Lektüre

SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

by

Donald D. Chamberlin
Raymond F. Boyce

IBM Research Laboratory
San Jose, California

ABSTRACT: In this paper we present the data manipulation facility for a structured English query language (SEQUEL) which can be used for accessing data in an integrated relational data base. Without resorting to the concepts of bound variables and quantifiers SEQUEL identifies a set of simple operations on tabular structures, which can be shown to be of equivalent power to the first order predicate calculus. A SEQUEL user is presented with a consistent set of keyword English templates which reflect how people use tables to obtain information. Moreover, the SEQUEL user is able to compose these basic templates in a structured manner in order to form more complex queries. SEQUEL is intended as a data base sublanguage for both the professional programmer and the more infrequent data base user.

1

¹ In: Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control. Kann aus dem Institutsnetz heraus vom ACM-Portal heruntergeladen werden.

Kapitel 4 Der SQL-Standard

4.1 Nullwerte

Die Problematik

- ▶ Liegt zu einem Attribut kein Wert vor, so kann dies durch Verwendung des *Nullwerts null* ausgedrückt werden.
- ▶ Als mögliche Interpretationen eines Nullwertes können wir unterscheiden:
Wert existiert, jedoch zur Zeit unbekannt - *Wert existiert erst in der Zukunft* - *Wert prinzipiell unbekannt* - oder auch *Attribut nicht anwendbar*.

Beispiel

Student

MatrNr	Name	Adresse	Semester	Exmatrikulationsdatum	Mutterschutz
1223	Hans Eifrig	null	2	null	null
3434	Lisa Lustig	Bergstraße 11	4	null	ja
1234	Maria Gut	Am Bächle 1	null	null	nein

Nullwerte und SQL

- ▶ SQL bietet die Prädikate IS NULL und IS NOT NULL an, um auf Existenz von Nullwerten prüfen zu können.

Bestimme alle Provinzen, zu denen die Fläche bekannt ist.

```
SELECT * FROM Provinz  
WHERE Fläche IS NOT NULL
```

- ▶ In Ausdrücken der Form A+B, A+1, etc. ist das Resultat null, wenn einer der Operanden null ist.
- ▶ Ausdrücke mit Vergleichsoperatoren der Form A=B, A<>B, A<B, etc. haben den Wahrheitswert UNKNOWN, wenn mindestens einer der beteiligten Operanden den Wert null besitzt.
- ▶ SQL liegt eine dreiwertige Logik zugrunde:
(t=TRUE, f=FALSE, u=UNKNOWN).

Wahrheitswerte

AND	t	u	f	OR	t	u	f	NOT	
t	t	u	f	t	t	t	t	t	f
u	u	u	f	u	t	u	u	u	u
f	f	f	f	f	t	u	f	f	t

Was liefern:

```
SELECT * FROM Provinz  
    WHERE (Fläche > 0)  
  
SELECT * FROM Provinz  
    WHERE NOT (Fläche > 0)
```

Vermeide Nullwerte wann immer es geht!

... denn sie verkomplizieren die Anfrageformulierung und ihre Semantik bzgl. der realen Welt ist nicht eindeutig.

äußerer Verbund (engl. Outer Join)

Land

LName	LCode	HStadt	Fläche
Austria	A	Vienna	84
Egypt	ET	Cairo	1001
France	F	Paris	547
Germany	D	Berlin	357
Italy	I	Rome	301
Russia	RU	Moscow	17075
Switzerland	CH	Bern	41
Turkey	TR	Ankara	779

Stadt

SName	LCode	PName	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

Wie viele Einwohner haben die Hauptstädte der einzelnen Länder?

```
SELECT L.LName AS Land, L.HStadt, S.Einwohner
  FROM Land L LEFT OUTER JOIN Stadt S
    ON L.HStadt = S.SName AND L.LCode = S.LCode
```

LName	HStadt	Einwohner
Austria	Vienna	null
Egypt	Cairo	null
France	Paris	2125
Germany	Berlin	3472
Italy	Rome	2546
Russia	Moscow	null
Switzerland	Bern	null
Turkey	Ankara	null

Land

LName	LCode	HStadt	Fläche
Austria	A	Vienna	84
Egypt	ET	Cairo	1001
France	F	Paris	547
Germany	D	Berlin	357
Italy	I	Rome	301
Russia	RU	Moscow	17075
Switzerland	CH	Bern	41
Turkey	TR	Ankara	779

Stadt

SName	LCode	PName	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Île de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

Dieselbe Anfrage mit RIGHT OUTER JOIN

```
SELECT L.LName AS Land, L.HStadt, S.Einwohner
  FROM Land L RIGHT OUTER JOIN Stadt S
    ON L.HStadt = S.SName AND L.LCode = S.LCode
```

LName	HStadt	Einwohner
France	Paris	2125
Germany	Berlin	3472
Italy	Rome	2546
null	null	198
null	null	277
null	null	1244
null	null	495

Varianten des äußeren Verbunds

- ▶ Bei einem LEFT OUTER JOIN bleiben die Tupel der linken Relation erhalten; rechts werden bei fehlenden Verbundpartnern Nullwerte ergänzt.
- ▶ Bei einem RIGHT OUTER JOIN werden analog gegebenenfalls links Nullwerte ergänzt.
- ▶ Ein FULL OUTER JOIN berechnet die Vereinigung des entsprechenden LEFT OUTER und RIGHT OUTER JOIN.

4.2 Aggregierung und Gruppierung

Aggregationsfunktionen: COUNT, MIN, MAX, SUM und AVG

Wie viele Länder gibt es in der Tabelle Land, wie groß ist die maximale, die minimale Fläche und die durchschnittliche Fläche aller Länder?

```
SELECT COUNT(LCode), MAX(Flaeche), MIN(Flaeche), AVG(Flaeche)  
      FROM Land
```

COUNT(LCode)	MAX(Flaeche)	MIN(Flaeche)	AVG(Flaeche)
8	17075	41	2523,125

Wie viele Länder haben eine Mitgliedschaft bzgl. der EU?

```
SELECT COUNT(*) AS AnzEU  
      FROM Mitglied  
     WHERE Organisation = 'EU' AND Art = 'member'
```

AnzEU

3

Wie viele unterschiedliche Organisationen werden in Mitglied aufgeführt?

```
SELECT COUNT(DISTINCT Organisation) AS AnzMitglied FROM Mitglied
```

AnzMitglied
5

Besonderheiten

- ▶ COUNT(*) liefert die Anzahl Zeilen der Tabelle, die sich nach Auswerten der FROM- und WHERE-Klausel ergeben hat.
 - ▶ Aggregationsfunktionen ignorieren für ihre Berechnungen Nullwerte.
 - ▶ Eine Ausnahme ist COUNT(*); hier werden auch alle Zeilen, in denen alle Spalten null sind, mitgezählt.
-
- ▶

```
SELECT LCode, AVG(Einwohner) FROM Stadt
```

 ist syntaktisch nicht zulässig, da ein Aggregationsoperator eine Menge von Zeilen auf einen einzigen Wert reduziert. Zulässig wäre:

```
SELECT AVG(Einwohner) FROM Stadt
```

Gruppierung

GROUP BY

- ▶ Mittels einer *Gruppierung* können wir eine virtuelle Struktur über einer Tabelle definieren.
- ▶ Die Gruppierungsattribute fassen alle Zeilen der Tabelle jeweils zu einer Gruppe zusammen, die bezüglich aller Gruppierungsattribute die gleichen Werte haben und zusätzlich die in einer optionalen HAVING-Klausel festgelegten Bedingungen erfüllt.
- ▶ Anfragen über einer gruppierten Tabelle betrachten die einzelnen Gruppen zusammen mit den Gruppierungsattributen analog zu einer Zeile einer Tabelle.

Konsequenterweise dürfen Attribute, die nicht als Gruppierungsattribute verwendet wurden, nur als Parameter für Aggregierungsfunktionen verwendet werden.

Wie groß ist die durchschnittliche Einwohnerzahl der Städte der jeweiligen Länder?

```
SELECT LCode, AVG(Einwohner) FROM Stadt  
    GROUP BY LCode
```

LCode	AVG(Einwohner)
D	1137,2
F	2125
I	2546

In welchen Ländern ist die durchschnittliche Einwohnerzahl kleiner 2 Mio.?

```
SELECT LCode, AVG(Einwohner) FROM Stadt  
    GROUP BY LCode  
    HAVING AVG(Einwohner) < 2000
```

LCode	AVG(Einwohner)
D	1137,2

Wie groß ist die durchschnittliche Einwohnerzahl der Provinzen der jeweiligen Länder?

```
SELECT LCode, PName, AVG(Einwohner), COUNT(*) AS Anzahl FROM Stadt  
GROUP BY (LCode, PName)
```

LCode	PName	AVG(Einwohner)	Anzahl
D	Berlin	3472	1
D	Baden	237,5	2
D	Bavaria	1244	1
D	Franken	495	1
F	Ile de France	2125	1
I	Lazio	2546	1

Was ist die größte Einwohnerzahl der Städte in einer Provinz?

```
SELECT LCode, PName, MAX(Einwohner) FROM Stadt  
GROUP BY (LCode, PName)
```

LCode	PName	MAX(Einwohner)
D	Berlin	3472
D	Baden	277
D	Bavaria	1244
D	Franken	495
F	Ile de France	2125
I	Lazio	2546

Auswertungsreihenfolge

SFW-Ausdruck (erweitert)

SELECT A_1, \dots, A_n	Liste der Attribute
FROM R_1, \dots, R_m	Liste der Relationen
WHERE F	Bedingung
GROUP BY B_1, \dots, B_k	Liste der Gruppierungsattribute
HAVING G	Gruppierungsbedingung
ORDER BY H	Sortierordnung

Für die Auswertungsreihenfolge gilt: FROM-Klausel vor WHERE-Klausel vor GROUP-Klausel vor HAVING-Klausel vor ORDER-Klausel vor SELECT-Klausel.

4.3 Mengenoperatoren

UNION, INTERSECT und EXCEPT (MINUS)

- ▶ Die beteiligten Tabellen müssen zueinander *kompatible* Spaltentypen haben.
- ▶ Die Resultatspalte bekommt dann jeweils den allgemeineren Typ.

Welche Ländercodes treten in der Relation Stadt oder der Relation Lage auf?

```
SELECT LCode, 'Stadt' AS Kategorie FROM Stadt  
UNION  
SELECT LCode, 'Lage' AS Kategorie FROM Lage
```

LCode	Kategorie
D	Lage
D	Stadt
F	Lage
F	Stadt
TR	Lage
ET	Lage
RU	Lage
I	Stadt

Welche Länder sind Teil von Europa und Asien?

```
SELECT LCode FROM Lage WHERE Kontinent = 'Europe'  
INTERSECT  
SELECT LCode FROM Lage WHERE Kontinent = 'Asia'
```

LCode
RU
TR

Welche Ländercodes treten in der Relation Land und nicht in der Relation Lage auf?

```
SELECT LCode FROM Land  
EXCEPT  
SELECT LCode FROM Lage
```

LCode
A
CH
I

Hinweis: Oracle verwendet das Schlüsselwort MINUS anstelle von EXCEPT!

Duplikate

- ▶ Duplikate werden berücksichtigt, sofern die Varianten UNION ALL, INTERSECT ALL, EXCEPT ALL verwendet werden. Andernfalls wird standardmäßig DISTINCT angenommen.
- ▶ Im Falle einer Verwendung von ALL verhalten sich die Operatoren wie folgt: Hat der erste Operand n Duplikate einer Zeile und der zweite Operand m , wobei $0 \leq n, m$, dann hat das Ergebnis bei UNION $n + m$, bei INTERSECT $\min(n, m)$, und bei EXCEPT $\max(n - m, 0)$ Duplikate dieses Tupels.

Hinweis: Oracle unterstützt nur UNION ALL!

4.4 Geschachtelte Anfragen

Eine Anfrage heißt *geschachtelt*, wenn sie in der SELECT-, FROM-, oder WHERE-, bzw. HAVING-Klausel selbst wieder eine SQL-Anfrage enthält.

Zum Testen des Ergebnisses einer Teilanfrage (engl. subquery) existieren die Operatoren: IN, ANY, ALL, EXISTS und NOT.

Welche Länder befinden sich im gleichen Kontinent wie Russland?

```
SELECT DISTINCT L2.LCode  
  FROM Lage L1, Lage L2  
 WHERE L1.Kontinent = L2.Kontinent AND L1.LCode = 'RU'
```

```
SELECT DISTINCT LCode FROM Lage  
 WHERE Kontinent IN  
 (SELECT Kontinent FROM Lage WHERE LCode = 'RU')
```

Welche Länder haben eine größere Fläche als mindestens ein anderes Land?

```
SELECT LName FROM Land  
WHERE Fläche > ANY  
(SELECT Fläche FROM Land)
```

LName
Austria
Egypt
France
Germany
Italy
Russia
Turkey

Welche Länder haben eine größere Fläche als alle anderen Länder?

```
SELECT LName FROM Land  
WHERE Fläche > ALL  
(SELECT Fläche FROM Land)
```

FALSCH! Alle anderen Länder!

Korrelationsvariablen

Bei Verwendung von *Korrelationsvariablen*, (*-namen*) wird die Teilanfrage pro möglicher Wertekombination der Korrelationsvariablen ihrer übergeordneten Anfragen genau einmal ausgeführt.

Welche Länder haben eine größere Fläche als alle anderen Länder?

```
SELECT LName FROM Land L1
  WHERE Fläche > ALL
        (SELECT Fläche FROM Land L2
          WHERE L1.LCode <> L2.LCode)
```

LName
Russia

Zu welchen Ländern ist mindestens ein Kontinent bekannt? (Variante 1)

```
SELECT LName FROM Land L1  
WHERE EXISTS  
(SELECT L2.Kontinent FROM Lage L2  
WHERE L1.LCode = L2.LCode)
```

LName
Germany
Egypt
France
Russia
Turkey

Zu welchen Ländern ist mindestens ein Kontinent bekannt? (Variante 2)

```
SELECT LName FROM Land L1  
WHERE 1 <=  
(SELECT COUNT(L2.Kontinent) FROM Lage L2  
WHERE L1.LCode = L2.LCode)
```

Division

Beschreiben Sie das Ergebnis der folgenden Anfrage:

```
SELECT DISTINCT LCode
  FROM Mitglied M
 WHERE NOT EXISTS (
    (SELECT Organisation FROM Mitglied
     WHERE LCode = 'A')
 EXCEPT /*MINUS*/
    (SELECT Organisation FROM Mitglied
     WHERE LCode = M.LCode))
```

Es werden die Länder berechnet, die in denselben Organisationen wie Österreich vertreten sind (member, guest oder observer).

Anfragen von dieser Struktur implementieren die relationale *Division*.
Zur Erinnerung:

- ▶ Seien a, b ganze Zahlen. Dann ist $a \div b$ die größte ganze Zahl q , so dass $q * b \leq a$.
- ▶ Seien A, B Mengen (Relationen). Dann ist $A \div B$ die größte Relation Q , so dass $Q \times B \subseteq A$.

$$\pi[LCode, Organisation]Mitglied \div \pi[Organisation](\sigma[LCode = 'A']Mitglied)$$

Welche Länder sind in denselben Organisationen wie Österreich vertreten?

```
SELECT DISTINCT LCode
  FROM Mitglied M
 WHERE NOT EXISTS
   ((SELECT Organisation FROM Mitglied
     WHERE LCode = 'A')
    MINUS
   (SELECT Organisation FROM Mitglied
     WHERE LCode = M.LCode))
```

(SELECT Organisation FROM Mitglied WHERE LCode = 'A')

{ EU }

(SELECT Organisation FROM Mitglied WHERE LCode = M.LCode)

M.LCode	Organisationen
A	{ EU }
D	{ EU, WEU }
ET	{ UN }
I	{ EU, NAM }
TR	{ UN, CERN }

⇒ {A, D, I}

Division (alternative Variante)

Welche Länder sind in denselben Organisationen wie Österreich vertreten?

```
SELECT DISTINCT M1.LCode
FROM Mitglied M1, Mitglied M2
WHERE M2.LCode = 'A' AND
      M1.Organisation = M2.Organisation
GROUP BY M1.LCode
HAVING COUNT(M1.Organisation) = (
    SELECT COUNT(Organisation)
    FROM Mitglied WHERE LCode = 'A' )
```

Wie verhalten sich die unterschiedlichen Varianten bei einer Division durch eine leere Tabelle oder bei der Existenz von Duplikaten?

Gleichheit

Welche Länder sind in GENAU denselben Organisationen wie Österreich vertreten?

Zwei Mengen A, B sind gleich genau dann, wenn $A \subseteq B$ und $B \subseteq A$;
 $A \subseteq B$ genau dann, wenn $A - B = \emptyset$.

```
SELECT DISTINCT LCode FROM Mitglied M WHERE
    NOT EXISTS (
        (SELECT Organisation FROM Mitglied WHERE LCode = 'A')
        MINUS
        (SELECT Organisation FROM Mitglied WHERE LCode = M.LCode) )
    AND NOT EXISTS (
        (SELECT Organisation FROM Mitglied WHERE LCode = M.LCode)
        MINUS
        (SELECT Organisation FROM Mitglied WHERE LCode = 'A') )
```

empfohlene Lektüre

SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

by

Donald D. Chamberlin
Raymond F. Boyce

IBM Research Laboratory
San Jose, California

ABSTRACT: In this paper we present the data manipulation facility for a structured English query language (SEQUEL) which can be used for accessing data in an integrated relational data base. Without resorting to the concepts of bound variables and quantifiers SEQUEL identifies a set of simple operations on tabular structures, which can be shown to be of equivalent power to the first order predicate calculus. A SEQUEL user is presented with a consistent set of keyword English templates which reflect how people use tables to obtain information. Moreover, the SEQUEL user is able to compose these basic templates in a structured manner in order to form more complex queries. SEQUEL is intended as a data base sublanguage for both the professional programmer and the more infrequent data base user.

1

¹ In: Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control.
Kann gegoogled werden.

4.5 Struktur der Syntax

SFW-Ausdruck

SELECT A_1, \dots, A_n	Liste der Attribute
FROM R_1, \dots, R_m	Liste der Relationen
WHERE F	Bedingung
GROUP BY B_1, \dots, B_k	Liste der Gruppierungsattribute
HAVING G	Gruppierungsbedingung
ORDER BY H	Sortierordnung

Orthogonalität der Syntax

- ▶ Ein *tabellenwertiger* Ausdruck ist überall dort zulässig, wo eine Tabelle stehen darf.
- ▶ Ein *skalarer* Ausdruck ist überall dort zulässig, wo ein skalarer Wert stehen darf.
- ▶ Ein *bedingter* Ausdruck ist überall dort zulässig, wo ein Wahrheitswert stehen darf.

(1) tabellenwertiger Ausdruck

Tabellen in SQL

Ein Anfrageausdruck in SQL definiert im Allgemeinen eine Tabelle.

- ▶ Jeder Tabellenbezeichner ist ein Anfrageausdruck.
- ▶ Jeder SFW-Ausdruck ist ein Anfrageausdruck.
- ▶ Ein Verbundausdruck ist ebenfalls ein Anfrageausdruck.
- ▶ Die üblichen Mengenoperatoren können verwendet werden, um Anfrageausdrücke zu bilden.
- ▶ Ein *Tabellen-Konstruktor* der Form
`VALUES ('a1', ..., 'an'), ('b1', ..., 'bn'), ...` ist ein Anfrageausdruck.

Liste die Namen, die für Städte und Länder verwendet werden.

```
SELECT Name  
FROM ( SELECT SName AS Name FROM Stadt  
      UNION  
      SELECT LName AS Name FROM Land ) T
```

Berechne die Anzahl der Menschen aller Länder, die in der größten Stadt ihres Landes leben.

```
SELECT SUM(GroßStädter)  
FROM ( SELECT LCode, MAX(Einwohner) AS Großstädter  
      FROM Stadt  
      GROUP BY LCode ) T
```

Division à la Algebra:

Seien X_1, X_2 Formate, $X_2 \subset X_1$, $Z = X_1 - X_2$
 und weiter $r_1 \subseteq \text{Tup}(X_1)$ (Dividend), $r_2 \subseteq \text{Tup}(X_2)$ (Divisor), wobei $r_2 \neq \emptyset$.

$$r_1 \div r_2 = \pi[Z]r_1 - \pi[Z](((\pi[Z]r_1) \times r_2) - r_1)$$

Welche Länder sind in denselben Organisationen wie Österreich vertreten?

$$\pi[LCode, Organisation]Mitglied \div \pi[Organisation](\sigma[LCode = 'A']Mitglied)$$

```

SELECT DISTINCT LCode FROM Mitglied
MINUS /* EXCEPT */
SELECT LCode FROM (
  SELECT LCode, Organisation FROM (
    (SELECT LCode FROM Mitglied)
    CROSS JOIN
    (SELECT Organisation FROM Mitglied WHERE LCode = 'A') )
  MINUS /* EXCEPT */
  SELECT LCode, Organisation FROM Mitglied
)
  
```

(2) skalarer Anfrageausdruck

Anstelle eines Wertes, bzw. Spaltenbezeichners, ist auch ein geklammerter Tabellenausdruck zulässig, sofern er *skalar* ist, d.h. genau einen Wert definiert.

Bestimme zu jeder Stadt den Mittelwert der Einwohnerzahl aller Städte, die weniger Einwohner haben als sie selbst.

```
SELECT SName, Einwohner,
       ( SELECT AVG(Einwohner) FROM Stadt S2
         WHERE S2.Einwohner < S1.Einwohner )
       AS kleinerMittelwert
  FROM Stadt S1
```

Bestimme diejenigen asiatischen Länder, deren Flächenanteil in Asien kleiner ist als der Anteil der Türkei in Asien.

```
SELECT LCode, Prozent FROM Lage
 WHERE Kontinent = 'Asia' AND
       Prozent <
       ( SELECT Prozent FROM Lage
         WHERE LCode = 'TR' AND Kontinent = 'Asia' )
```

(3) bedingter Ausdruck

Welche Länder befinden sich im gleichen Kontinent wie Russland?

```
SELECT DISTINCT L2.LCode  
  FROM Lage L1, Lage L2  
 WHERE L1.Kontinent = L2.Kontinent AND L1.LCode = 'RU'
```

```
SELECT DISTINCT LCode FROM Lage  
 WHERE Kontinent IN  
 (SELECT Kontinent FROM Lage WHERE LCode = 'RU')
```

- ▶ Ein Ausdruck, dem ein Wahrheitswert zugeordnet werden kann, ist ein *bedingter* Ausdruck.
- ▶ Bedingte Ausdrücke sind Teil einer WHERE-, HAVING- oder ON-Klausel.
- ▶ Wesentlich für die korrekte Verwendung bedingter Ausdrücke ist die Miteinbeziehung des Auftretens von Nullwerten.

Probleme mit Nullwerten.

Vergleiche unter Annahme Tabelle Land enthält Wunderland mit Nullwert für HStadt:

Welche Städte sind keine Hauptstadt, d.h., heißen nicht so, wie die Hauptstadt irgendeines Landes?

```
SELECT SName FROM Stadt S  
WHERE S.SName NOT IN ( SELECT HStadt FROM Land )
```

Resultat: leere Tabelle

```
SELECT SName FROM Stadt S  
WHERE NOT EXISTS (  
    SELECT HStadt FROM Land  
    WHERE HStadt = S.SName )
```

Resultat: Freiburg, Munich, Nuremberg, Karlsruhe

Welche Städte sind keine Hauptstadt?

```
INSERT INTO Land VALUES ('Wunderland', 'WU', null, 0)
SELECT SName FROM Stadt S
WHERE S.SName NOT IN ( SELECT HStadt FROM Land )
```

SELECT HStadt FROM Land

Vienna
Bern
Berlin
Cairo
Paris
Rome
Moscow
Ankara
null

Tabelle:

'Freiburg' IN (SELECT HStadt FROM Land) ≡ UNKNOWN
⇒ 'Freiburg' NOT IN (SELECT HStadt FROM Land) ≡ UNKNOWN

da der Vergleich 'Freiburg' = null den Wahrheitswert UNKNOWN liefert.

4.6 Datentypen

Basis-Datentypen

SQL bietet eine Fülle von unterschiedlichen Datentypen an, mittels derer die Wertebereiche der Spalten einer Tabelle festgelegt werden können.

- ▶ INTEGER, SMALLINT
- ▶ NUMERIC, DECIMAL
Angabe Anzahl Ziffern insgesamt und Anzahl Kommastellen.
- ▶ REAL, DOUBLE PRECISION, FLOAT
- ▶ CHARACTER, CHARACTER VARYING, CHARACTER LARGE OBJECT
- ▶ BIT, BIT VARYING, BINARY LARGE OBJECT
- ▶ BOOLEAN
- ▶ DATE, TIME, TIMESTAMP, INTERVALL,

Definition einer Tabelle am Beispiel

```
CREATE TABLE Stadt (
    SName      VARCHAR(50),
    PName      VARCHAR(50),
    LCode      CHAR(4),
    Einwohner   INTEGER,
    LGrad      NUMBER,
    BGrad      NUMBER,
    PRIMARY KEY (SName ,PName ,LCode) )
```

DESCRIBE liefert die Attribute einer Tabelle inklusive Typen.

```
DESCRIBE Stadt;
```

Objektrelationale Datenbanken: Konstruierte Datentypen

Ein Datentyp heißt *konstruiert*, sofern seine Werte aus Werten anderen Typen, sogenannter *Element-Typen*, zusammengesetzt sind.

- ▶ Der Datentyp **ARRAY** fasst mehrere Werte seines Element-Typs geordnet zusammen, die über einen Index referenziert werden können.
- ▶ Der Typ **ROW** lässt hingegen zu, dass Werte unterschiedlicher Element-Typen geordnet zusammengefasst werden und das der Zugriff auf die einzelnen Komponenten über Bezeichner, analog zu Spaltenbezeichnern, ermöglicht wird.
- ▶ Der Datentyp **MULTISET** fasst mehrere Werte eines Element-Typs, möglicherweise mit Duplikaten, zu einer ungeordneten Menge zusammen.

```
CREATE TABLE Land (
    :
    Provinzen VARCHAR(50) ARRAY[20]
    Organisationen
        ROW(Organisation VARCHAR(50), Art VARCHAR(20)) MULTISET
);
CREATE TABLE Stadt (
    :
    Koordinaten ROW(LGrad NUMBER, BGrad NUMBER )
);

```

- ▶ Die fünfte Provinz eines Landes wird mittels Provinzen[5] referenziert.
- ▶ Der Längengrad innerhalb der Koordinaten einer Stadt kann mittels eines Pfadausdrucks der Form Koordinaten.LGrad angesprochen werden.
- ▶ Mittels ('EU', 'member') ELEMENT Organisationen wird getestet, ob die Mitgliedschaften eines Landes bezüglich der EU den *member*-Status hat.

4.7 Einfügen, Löschen und Ändern

Einfügen

Aufnahme eines neuen Mitgliedes in die EU.

```
INSERT INTO Mitglied (LCode, Organisation, Art)
VALUES ('PL', 'EU', 'member')
```

Alle Länder, die in irgendwelchen Organisationen vertreten sind aber noch nicht in der Relation Land auftreten, werden in diese Relation übernommen.

```
INSERT INTO Land (LCode)
SELECT DISTINCT M.LCode
FROM Mitglied M
WHERE NOT EXISTS (
    SELECT L.LCode
    FROM Land L
    WHERE L.LCode = M.LCode )
```

- ▶ Mittels `INSERT` kann eine neue Zeile, oder eine Menge von neuen Zeilen in eine Tabelle `T` eingefügt werden.
- ▶ Werden nicht zu allen Attributen von `T` Werte gegeben, so werden für die fehlenden Werte möglicherweise vorgesehene Default-Werte, bzw. der Nullwert `null` genommen.
- ▶ Die Angabe der Spaltennamen kann entfallen, wenn die Werte in der Reihenfolge der Spaltenamen in der `CREATE`-Anweisung definiert werden.

Sequenznummern

Beim Einfügen von Zeilen muss die Eindeutigkeit des Primärschlüssels gewährleistet sein.

Beispiel Identitätsspalte

```
CREATE TABLE Land (
    LandNr INTEGER GENERATED ALWAYS AS IDENTITY (
        START WITH 1
        INCREMENT BY 1
        MINVALUE 1
        MAXVALUE 100000
        NO CYCLE ),
    :
)
```

```
INSERT INTO Land (LName, HStadt, Fläche)
VALUES ('Bavaria', 'Munich', 70)
```

generierte Spalten

Der Wert eines Attributes ergibt sich automatisch aus den Werten anderer Attribute desselben Tupels.

Beispiel

```
CREATE TABLE Land (
    LandNr INTEGER GENERATED ALWAYS AS IDENTITY ( ... ),
    :
    Fläche      NUMBER,
    Einwohner   NUMBER,
    Dichte GENERATED ALWAYS AS (Einwohner / Fläche)
)
```

Löschen

- ▶ `DELETE FROM T WHERE P`
- ▶ Es werden alle Tupel aus T, für die der bedingte Ausdruck P wahr ist, markiert und anschließend aus T entfernt.

Löschen des gesamten Inhalts der Tabelle Stadt.

```
DELETE FROM Stadt
```

Löschen von ausgewählten Zeilen.

```
DELETE FROM Stadt  
WHERE Einwohner < (  
    SELECT AVG(Einwohner) FROM Stadt )
```

Ändern

- ```
UPDATE T
▶ SET A_1 = val_1, ..., A_n = val_n
 WHERE P
▶ Anstelle eines direkten Wertes innerhalb einer Zuweisung kann auch ein
 skalarer Ausdruck stehen.
```

Im Zuge der Euro-Umstellung werden die Angaben des Bruttonsozialprodukts angepasst.

```
UPDATE Land
SET BruttoSP =
CASE BruttoSP
 WHEN LCode = 'D' THEN BruttoSP * 0,5
 WHEN LCode = 'F' THEN BruttoSP * 0,16
 ELSE NULL
END
```

## 4.8 Sichten

- ▶ Eine Sicht  $V$  ist eine durch einen Anfrageausdruck  $E$  definierte Tabelle:
- ▶ `CREATE VIEW V AS  
 <E>`
- ▶ Im Unterschied zu den als Sicht definierten Tabellen bezeichnen wir die mittels `CREATE TABLE` definierten Tabellen als *Basistabellen*.
- ▶ Bezeichner von Sichten dürfen in SQL überall stehen, wo ein Tabellenbezeichner stehen darf.

Definiere zu der Tabelle Benachbart eine bezüglich Symmetrie abgeschlossene Tabelle symBenachbart in Form einer Sicht.

Benachbart

| <u>LCode1</u> | <u>LCode2</u> |
|---------------|---------------|
| CH            | D             |
| CH            | F             |
| CH            | I             |
| D             | F             |
| I             | F             |

```
CREATE VIEW symBenachbart AS
(SELECT LCode1 AS Von, LCode2 AS Nach
 FROM Benachbart)
UNION
(SELECT LCode2 AS Von, LCode1 AS Nach
 FROM Benachbart)
```

Welche Länder sind zu Deutschland benachbart?

```
SELECT Nach FROM symBenachbart
WHERE Von = 'D'
```

## Materialisierte und virtuelle Sichten

- ▶ Ein Datenbanksystem kann Sichten entweder bei Bedarf jeweils neu berechnen, oder eine einmal berechnete Sicht für weitere Bearbeitungen permanent speichern. Im ersten Fall redet man von einer *virtuellen* Sicht, im zweiten Fall von einer *materialisierten* Sicht.
- ▶ Soll eine Anfrage bearbeitet werden, die sich auf eine virtuelle Sicht bezieht, so wird vor Ausführung der Anfrage der Name der Sicht durch den sie definierenden Ausdruck ersetzt (*Anfrage-Modifizierung*).
- ▶ Gegenüber einer materialisierten Sicht hat eine virtuelle Sicht den Vorteil, dass ihr Inhalt garantiert dem aktuellen Zustand der Datenbank entspricht.
- ▶ Standardmäßig ist eine Sicht einer Datenbank virtuell. Materialisierte Sichten werden typischerweise für sogenannte *Datenlager* (engl. *Data-Warehouses*) eingesetzt; zu ihrer Aktualisierung ist häufig ein erheblicher organisatorischer und systemtechnischer Aufwand vonnöten.
- ▶ Im Folgenden betrachten wir ausschließlich virtuelle Sichten.
- ▶ Eine interessante Frage ist, ob in einer virtuellen Sicht Einfügen, Löschen und Ändern von Zeilen erlaubt sein kann, oder nicht.

## Ändern von Sichten

Sei  $\mathcal{R}$  ein Datenbank-Schema.

- ▶ Eine *Datenbankänderung* ist eine Funktion  $t$  von der Menge der Instanzen zu  $\mathcal{R}$  auf sich selbst.  $t : \mathcal{I}^{\mathcal{R}} \rightarrow \mathcal{I}^{\mathcal{R}}$
- ▶ Eine *Sicht* ist eine Funktion  $f$  von der Menge aller Instanzen zu  $\mathcal{R}$  in die Menge aller Instanzen zu  $S$ , wobei  $S$  das durch die Sicht definierte Relationsschema ist.  $f : \mathcal{I}^{\mathcal{R}} \rightarrow \mathcal{I}^S$
- ▶ Eine *Sichtänderung* ist eine Funktion  $u$  von der Menge aller Instanzen zu  $S$  auf sich selbst.  $u : \mathcal{I}^S \rightarrow \mathcal{I}^S$
- ▶ Sei  $u$  eine Sichtänderung und  $t$  eine Datenbankänderung, so dass für jede Datenbank-Instanz  $\mathcal{I}^{\mathcal{R}}$  gilt:

$$u(f(\mathcal{I}^{\mathcal{R}})) = f(t(\mathcal{I}^{\mathcal{R}})),$$

dann nennen wir  $t$  eine *Transformation* von  $u$ .

- ▶ Auf einer Sicht sind grundsätzlich nur solche Änderungen zulässig, zu denen eine Transformation existiert.

### Beispiel

| A | B |
|---|---|
| a | b |
| x | b |

| B | C |
|---|---|
| b | c |
| b | z |

| A | B | C |
|---|---|---|
| a | b | c |
| a | b | z |
| x | b | c |
| x | b | z |

### INSERT (y,b,c) in v

Es existiert keine Transformation.

### DELETE (a,b,c) in v

Es existiert keine Transformation.

### UPDATE (a,b,c) zu (y,b,c) in v

Es existiert keine Transformation.

## Projektionssicht

Einfügen, Löschen und Ändern ist nur dann erlaubt, wenn der Schlüssel der Basistabelle komplett in der Sicht vorhanden ist.

Informationen über Städte sind nur anonymisiert erlaubt.

```
CREATE VIEW StadtInfo AS
 SELECT PName, Einwohner
 FROM Stadt

INSERT INTO StadtInfo VALUES (Baden,90)
```

INSERT nicht zulässig!

## Selektionssicht

- ▶ Aufgrund von Einfügungen und Änderungen können als unerwünschter Seiteneffekt Zeilen aus der Sicht herausfallen und damit in anderen Sichten erkennbar werden.
- ▶ Dieser Seiteneffekt kann durch Hinzunahme der Klausel `WITH CHECK OPTION` zu der Sichtdefinition verhindert werden.

### Beschränkung auf Großstädte.

```
CREATE VIEW Großstadt AS
 SELECT *
 FROM Stadt
 WHERE Einwohner >= 1000
```

```
UPDATE Großstadt SET Einwohner = Einwohner * 0.9
```

## Verbundsicht

In SQL-92 und in SQL:1999 werden eine Reihe von Regeln diskutiert, deren Erfülltsein die Existenz einer Transformation sichern. Diese Regeln garantieren im Wesentlichen ein eindeutiges Rückverfolgen der Sichtänderung zu einzelnen Zeilen in den Basistabellen.

Ordne jeder Stadt ihren Kontinent zu.

```
CREATE VIEW StadtInfo AS
 SELECT S.SName, L.Kontinent
 FROM Stadt S, Lage L
 WHERE S.LCode = L.LCode
```

```
INSERT INTO StadtInfo VALUES ('Freiburg', 'DreiLänderEck')
```

INSERT nicht zulässig!

# empfohlene Lektüre

## Efficiently Updating Materialized Views<sup>\*</sup>

José A. Blakeley, Per-Åke Larson, Frank Wm. Tompa

Data Structuring Group,  
Department of Computer Science,  
University of Waterloo,  
Waterloo, Ontario, N2L 3G1

### Abstract

Query processing can be sped up by keeping frequently accessed users' views materialized. However, the need to access base relations in response to queries can be avoided only if the materialized view is adequately maintained. We propose a method in which all database updates to base relations are first filtered to remove from consideration those that cannot possibly affect the view. The conditions given for the detection of updates of this type, called *irrelevant updates*, are necessary and sufficient and are independent of the database state. For the remaining database updates, a *differential* algorithm can be applied to re-evaluate the view expression. The algorithm proposed exploits the knowledge provided by both the view definition expression and the database update operations.

derived relation or *view* is defined by a relational expression (i.e., a query evaluated over the base relations). A derived relation may be *virtual*, which corresponds to the traditional concept of a view; or *materialized*, which means that the resulting relation is actually stored. As the database changes because of updates applied to the base relations, the materialized views may also require change. A materialized view can always be brought up to date by re-evaluating the relational expression that defines it. However, complete re-evaluation is often wasteful, and the cost involved may be unacceptable.

The need for a mechanism to update materialized views efficiently has been expressed by several authors. Gardarin et al. [GSV84] consider *concrete views* (i.e., materialized views) as a candidate approach for the support of real time queries. However, they discard this approach because of the lack

1

<sup>1</sup> In: Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data.

## 4.9 SQL und Programmiersprachen

- ▶ SQL hat eine tabellenorientierte Semantik.
- ▶ SQL hat eine im Vergleich zu einer Programmiersprache eingeschränkte Mächtigkeit, so dass es typischerweise ermöglicht wird, SQL von einem in einer gängigen Programmiersprache geschriebenen Programm aus aufzurufen.
- ▶ Anwendungen können so unter Ausnutzung der vollen Mächtigkeit einer Programmiersprache Datenbank-Instanzen verarbeiten.
- ▶ *Impedance-Mismatch*

## Ansätze der Integration

### (A) **SQL-Erweiterung:**

Erweiterung von SQL um imperative Sprachelemente zur Formulierung von *in der Datenbank gespeicherten* benutzerdefinierten Funktionen und Prozeduren.

### (B) **Statisches SQL:**

Einbettung von SQL-Ausdrücken in eine Programmiersprache.

### (C) **Dynamisches SQL:**

Übergabe eines datenabhängig gebildeten SQL-Ausdrucks an eine Datenbank während der Ausführung eines Programms.

## (A) SQL-Erweiterung

- ▶ Deklaration von *Variablen*
- ▶ Zuweisung von Werten an Variable
- ▶ Sequenz von Anweisungen
- ▶ bedingte Anweisungen und Wiederholungsanweisungen
- ▶ Funktionen und Prozeduren.<sup>1</sup>

Funktionen können als parametrisierbare virtuelle Sichten betrachtet werden.

*Hinweis:* Oracle verwendet teilweise eine andere Syntax als der SQL-Standard!

---

<sup>1</sup>In Oracle sind keine tabellenwertigen Parameter erlaubt.

## Funktionen als parametrisierbare virtuelle Sichten

| Benachbart |        | nachbarVon('D') |
|------------|--------|-----------------|
| LCode1     | LCode2 | LCode           |
| CH         | D      | CH              |
| CH         | F      | F               |
| CH         | I      |                 |
| D          | F      |                 |
| I          | F      |                 |
| :          | :      | :               |
| :          | :      |                 |

Berechne die benachbarten Länder zu einem gegebenen Land.

```
CREATE FUNCTION nachbarVon(X CHAR(4))
RETURNS TABLE (LCode CHAR(4))
RETURN (
 SELECT LCode2 AS LCODE FROM Benachbart WHERE LCODE1 = X
 UNION
 SELECT LCode1 AS LCode FROM Benachbart WHERE LCODE2 = X)
```

```
SELECT * FROM TABLE (nachbarVon('D')) T
```

*Hinweis:* Nicht direkt umsetzbar in Oracle, da Funktionen in Oracle keine Tabellen zurückgeben können!

## SQL-Standard vs. Oracle Syntax

Berechne die Anzahl Städte zu einem gegebenen Land. (SQL-Standard)

```
CREATE FUNCTION anzahlStaedte(Land CHAR(2))
RETURNS NUMBER
RETURN (
 SELECT count(*) FROM Stadt WHERE LCode = Land
)
```

Berechne die Anzahl Städte zu einem gegebenen Land. (Oracle)

```
CREATE FUNCTION anzahlStaedte(Land CHAR)
RETURN NUMBER IS
numCities NUMBER;
BEGIN
 SELECT count(*) INTO numCities FROM Stadt WHERE LCode = Land;
 RETURN numCities;
END;
```

Aufrufen der Funktion mittels Dummy Tabelle

```
SELECT anzahlStaedte('D') FROM Dual
```

Gib zu einem Land alle erreichbaren Länder mit der Mindestanzahl Grenzübergänge an  
(SQL-Standard; Oracle Version in den Übungen).

```
CREATE FUNCTION ErreichbarVon(Start CHAR(4))
RETURNS TABLE (Nach CHAR(4), Anzahl INTEGER)
BEGIN
CREATE TABLE Erreichbar (Nach CHAR(4), Anzahl INTEGER);
DECLARE alt, neu INTEGER;

/* Initialisiere mit den direkt erreichbaren Ländern */
INSERT INTO Erreichbar
 SELECT T.LCode2 AS Nach, 1 AS Anzahl
 FROM symBenachbart T WHERE T.LCode1 = Start;

/* Initialisiere Abbruchbedingung */
SET alt = 0;
SET neu = (SELECT COUNT(*) FROM Erreichbar);

/* Berechne iterativ die indirekt erreichbaren Länder */
WHILE (alt <> neu) DO
 SET alt = neu;
 INSERT INTO Erreichbar
 SELECT DISTINCT B.LCode2, (A.Anzahl + 1)
 FROM Erreichbar A, symBenachbart B
 WHERE A.Nach = B.LCode1
 AND B.LCode2 <> Start
 AND B.LCode2 NOT IN (SELECT Nach FROM Erreichbar);
 SET neu = (SELECT COUNT(*) FROM Erreichbar);
END WHILE;

RETURN Erreichbar;
END
```

Initialisiere Erreichbar mit den *direkt* erreichbaren Ländern.

```
INSERT INTO Erreichbar
 SELECT T.LCode2 AS Nach, 1 AS Anzahl
 FROM symBenachbart T WHERE T.LCode1 = Start;
```

Die Berechnung soll terminieren, wenn nach einer Iteration kein neues *indirekt* erreichbares Land hinzugekommen ist.

```
SET neu = (SELECT COUNT(*) FROM Erreichbar);
WHILE (alt <> neu) DO
 SET alt = neu;
 INSERT INTO Erreichbar
 ...
 SET neu = (SELECT COUNT(*) FROM Erreichbar)
END WHILE;
```

Zyklen und Mehrfachberechnungen sollen vermieden werden.

- ▶ DISTINCT berücksichtigt den Fall, dass zum selben Land mehrere Wege gleicher Länge existieren können.
- ▶ Zyklen werden berücksichtigt, indem ein weiteres Land nur dann hinzugefügt wird, wenn das neu berechnete Land nicht bereits als erreichbares Land bekannt ist.
- ▶ Zyklen zum Start werden ebenfalls ausgeschlossen.

```
INSERT INTO Erreichbar
SELECT DISTINCT B.LCode2, (A.Anzahl + 1)
FROM Erreichbar A, symBenachbart B
WHERE A.Nach = B.LCode1
 AND B.LCode2 <> Start
 AND B.LCode2 NOT IN (SELECT Nach FROM Erreichbar);
```

## (B) Statisches SQL

- ▶ Stehen die auszuführenden SQL-Anfragen bereits zur Übersetzungszeit eines Programms als Teil des Programmcodes fest, dann redet man von einer *statischen* Einbettung von SQL in eine Programmiersprache.
- ▶ Eine datenabhängige Änderung der Anfragen während der Ausführung des Programms ist dann nicht mehr möglich.
- ▶ Die Ergebnisse einer SQL-Anfrage werden innerhalb des Programms mittels eines Cursors zugänglich gemacht.

### Anlegen eines Cursors

```
EXEC SQL DECLARE StadtCursor CURSOR FOR
 SELECT DISTINCT S.SName, L.LName
 FROM Stadt S, Land L
 WHERE S.LCode = L.LCode;
```

### Öffnen einen Cursors

```
EXEC SQL OPEN StadtCursor;
```

### Aktuelle Zeile eines Cursors auslesen

```
EXEC SQL FETCH StadtCursor INTO :stadtName, :landName;
```

### Cursor schließen

```
EXEC SQL CLOSE StadtCursor;
```

## (C) Dynamisches SQL

- ▶ Können wir einen SQL-Ausdruck während der Ausführung eines Programms in Form einer Zeichenkette an das Datenbanksystem übergeben, so redet man von *dynamischem* SQL.
- ▶ Standardisierte Schnittstellen: ODBC und JDBC.  
Erweiterung von Java: SQLJ.

### Beispiel: Berechnen einer Adjazenzmatrix in PHP

```
<?php
sql_connect('Mondial','lausen','buch');
$AdjazenzMatrix = array();
$query = 'SELECT * FROM Benachbart';
$result = sql_query($query);
while ($row = sql_fetch_assoc($result)) {
 $i = $row['von']; $j = $row['nach'];
 $AdjazenzMatrix[$i][$j] = 1;
}
?>
```

## 4.10 Integrität und Trigger

- ▶ Im Allgemeinen sind nur solche Instanzen einer Datenbank erlaubt, deren Relationen die der Datenbank bekannten *Integritätsbedingungen* (IB) erfüllen.
- ▶ Integritätsbedingungen können *explizit* durch den Benutzer definiert werden, durch die Definition von konkreten Schemata *implizit* erzwungen werden, oder bereits dem relationalen Datenmodell *inhärent* sein.
- ▶ *Inhärente Bedingungen*: Attributwerte sind skalar; Relationen, abgesehen von Duplikaten, verhalten sich wie Mengen, d.h. ohne weitere Angaben haben sie insbesondere keine Sortierung.
- ▶ *Implizite Bedingungen*: Werte der Attribute eines Primärschlüssels dürfen keine Nullwerte enthalten
- ▶ *Explizite Bedingungen*: Werden als Teil der CREATE TABLE-Klausel, bzw. der CREATE SCHEMA-Klausel definiert.

- ▶ Integritätsbedingungen sind von ihrer Natur aus deklarativ: sie definieren die zulässigen Instanzen, ohne auszudrücken, wie eine Gewährleistung der Integrität implementiert werden kann.
- ▶ Eine wichtige Klasse von deklarativen Integritätsbedingungen sind *Fremdschlüsselbedingungen*, die gewährleisten, dass keine *dangling* Referenzen zwischen den Tupeln in den Tabellen bestehen.
- ▶ Komplementär zu den deklarativen Bedingungen bieten Datenbanksysteme einen *Trigger*-Mechanismus an, mit dem in Form von Regeln definiert werden kann, welche Aktionen zur Gewährleistung der Integrität vorgenommen werden sollen, bzw. wie Verletzungen behandelt werden sollen.
- ▶ Mittels Trigger können wir insbesondere die Zulässigkeit von *Zustandsübergängen* kontrollieren, was mit Integritätsbedingungen aufgrund ihres Bezugs zu gerade einem Zustand nicht möglich ist.

## 4.10.1 Schlüsselbedingungen

Zu jeder Tabelle werden typischerweise ein *Primärschlüssel* und möglicherweise weitere Schlüssel festgelegt (**UNIQUE**-Klausel).

In jeder Instanz zu der Tabelle Land können alle Zeilen eindeutig durch ihren Spaltenwert zu LCode, oder alternativ, durch ihren Spaltenwert zu LName identifiziert werden.

```
CREATE TABLE Land (
 LName VARCHAR(35) UNIQUE,
 LCode VARCHAR(4) PRIMARY KEY,
 ...
```

Identifizierendes Kriterium für die Tabelle Stadt sei SName, LCode, PName, bzw. alternativ, LGrad, BGrad.

```
CREATE TABLE Stadt (
 ...
 PRIMARY KEY (SName,LCode,PName),
 UNIQUE (LGrad,BGrad))
```

## 4.10.2 Statische Integrität

### CHECK-Klausel

- ▶ *Statische* Integrität definiert unter Verwendung der CHECK-Klausel, welche Instanzen eines Schemas zulässig sind.
- ▶ Mittels der CHECK-Klausel können die zulässigen Werte eines Datentyps und die für eine Spalte einer konkreten Tabelle zu verwendenden Werte weiter eingeschränkt werden.
- ▶ Darüberhinaus können beliebige, mittels SQL-Anfrageausdrücken gebildete, Bedingungen über den Instanzen der Tabellen eines Schemas ausgedrückt werden.

## Wertebereichsbedingungen mittels NONNULL, DEFAULT und CREATE DOMAIN

```
LName VARCHAR(35) NONNULL
Prozent NUMBER DEFAULT 100
```

```
CREATE DOMAIN meineStädte VARCHAR(35) DEFAULT '?'
CREATE TABLE Stadt (
 SName meineStädte,
 :
)
```

## Wertebereichsbedingungen mittels CHECK

```
CREATE DOMAIN meineStädte VARCHAR(35),
 DEFAULT 'Paris',
 CHECK (VALUE IN ('Berlin', 'Paris',
 'London', 'Rom'))
```

## Spaltenbedingung mittels CHECK

```
CREATE TABLE Stadt (
 SName meineStädte,
 :
 LGrad NUMBER
 CHECK (LGrad BETWEEN -180 AND 180),
 BGrad NUMBER
 CHECK (BGrad BETWEEN -90 AND 90),
 ...)
```

## Spalten- und Tabellenbedingung mittels CHECK

Die Summe aller Anteile an unterschiedlichen Kontinenten eines Landes muss 100 ergeben.

```
CREATE TABLE Lage (
 LCode VARCHAR(4),
 Kontinent VARCHAR(35),
 Prozent NUMBER
 CHECK (Prozent BETWEEN 0 AND 100),
 CHECK (100 = (SELECT SUM(L.Prozent) FROM Lage L
 WHERE LCode = L.LCode)))
```

## Assertion

- ▶ Spalten- und Tabellenbedingungen sind erfüllt, wenn jede Zeile der betreffenden Tabelle sie erfüllt.
- ▶ Spalten- und Tabellenbedingungen sind somit *implizit*  $\forall$ -quantifiziert über den Zeilen der Tabelle.
- ▶ Alternativ können wir die explizitere Form einer ASSERTION wählen

### CHECK Assertion:

Die Summe aller Anteile an unterschiedlichen Kontinenten eines Landes muss 100 ergeben.

```
CREATE ASSERTION AssertLage (
 CHECK (NOT EXISTS (
 SELECT LCode FROM Lage
 GROUP BY LCode
 HAVING (SUM(Prozent) <> 100))
)
)
```

## CHECK und ASSERTION in ORACLE, SQL Server, etc.

- ▶ Typischerweise sind Sub-Queries in der CHECK-Klausel verboten.
- ▶ ASSERTION wird nicht unterstützt.

**Gewährleistung von Integritätsbedingungen in solchen Fällen mittels  
TRIGGER**

## 4.10.3 Fremdschlüsselbedingungen

- ▶ *Fremdschlüsselbedingungen* werden als Teil der CREATE TABLE-Klausel definiert.
- ▶ Sie sind formal sogenannte *Inklusionsabhängigkeiten*: zu jedem von null verschiedenen Fremdschlüsselwert in einer Zeile der *referenzierenden* Tabelle, der C- (child-) Tabelle, existiert ein entsprechender Schlüsselwert in einer Zeile der *referenzierten* Tabelle, der P- (parent-) Tabelle.
- ▶ Man redet hier auch von *referentieller* Integrität. Zur Definition von Fremdschlüsselbedingungen steht die FOREIGN KEY-Klausel zur Verwendung in der C-Tabelle zur Verfügung.

Die Spalte LCode innerhalb der Tabelle Provinz enthält Werte des Schlüssels LCode der Tabelle Land. Zu jeder Zeile in Provinz muss eine Zeile in Land existieren, deren Schlüsselwert gleich dem Fremdschlüsselwert ist.

```
CREATE TABLE Provinz (
 PName VARCHAR(35),
 LCode VARCHAR(4),
 Fläche NUMBER
 PRIMARY KEY (PName, LCode),
 FOREIGN KEY (LCode) REFERENCES Land (LCode))
```

Für die Tabelle Stadt sind zwei Fremdschlüsselbeziehungen relevant. Einmal müssen die referenzierten Länder in der Tabelle zu Land existieren, und zum andern entsprechend die Provinzen. Letzterer Fremdschlüssel besteht aus zwei Spalten. Die Zuordnung der einzelnen Spalten des Fremd- und Primärschlüssels ergeben sich aus der Reihenfolge des Hinschreibens.

```
CREATE TABLE Stadt (
 :
 PRIMARY KEY (SName, LCode, PName),
 FOREIGN KEY (LCode) REFERENCES Land (LCode),
 FOREIGN KEY (LCode, PName) REFERENCES Provinz (LCode, PName))
```

## referentielle Aktionen

Zur Gewährleistung der referentiellen Integrität werden sogenannte *referentielle Aktionen* zur Ausführung bezüglich der C-Tabellen definiert. Aufgabe dieser Aktionen ist die Kompensierung von durch DELETE- und UPDATE-Operationen auf der zugehörigen P-Tabelle verursachten Verletzungen der Integrität.

- ▶ Änderungen der P-Tabelle werden auf die C-Tabelle übertragen.  
(CASCADE)
- ▶ Die Änderung der P-Tabelle wird im Falle einer Verletzung der referentiellen Integrität einer C-Tabelle abgebrochen.  
(NO ACTION oder RESTRICT)
- ▶ Der Fremdschlüsselwert der C-Tabelle wird angepaßt.  
(SET NULL oder SET DEFAULT)

*Hinweis:* Oracle unterstützt bei UPDATE-Operationen nur die Aktion NO ACTION und bei DELETE-Operationen die Aktionen CASCADE, SET NULL und NO ACTION (keine Angabe entspricht NO ACTION).

Wird der Code eines Landes geändert oder das Land gelöscht, so sollen die neuen Codes bei den zugehörigen Provinzen nachgezogen werden, bzw. auch die Provinzen des gelöschten Landes gelöscht werden.

```
CREATE TABLE Provinz (
 :
 FOREIGN KEY (LCode) REFERENCES Land (LCode)
 ON DELETE CASCADE ON UPDATE CASCADE)
```

Werden Provinzen gelöscht, so sollen ihre Städte weiter in der Datenbank bestehen bleiben, wobei der betreffende Fremdschlüsselwert Nullwerte erhält. Änderungen eines Provinzschlüssels sollen auf die betroffenen Städte übertragen werden.

```
CREATE TABLE Stadt (
 :
 PRIMARY KEY (SNAME)
 FOREIGN KEY (LCode, PName)
 REFERENCES Provinz (LCode, PName)
 ON DELETE SET NULL ON UPDATE CASCADE)
```

## Warum werden nur DELETE- und UPDATE-Operationen auf den entsprechenden P-Tabellen betrachtet?

- ▶ Einfügen bezüglich der P-Tabelle ist für die referentielle Integrität immer unkritisch.
- ▶ Löschen bezüglich der C-Tabelle ist für die referentielle Integrität immer unkritisch.
- ▶ Einfügen bezüglich der C-Tabelle oder Ändern bezüglich der C-Tabelle, die einen Fremdschlüsselwert erzeugen, zu dem kein Schlüssel in der P-Tabelle existiert, sind immer primär unzulässig, da von Änderungen in den C-Tabellen im Allgemeinen kein sinnvoller Rückschluss auf Änderungen der P-Tabellen möglich ist; anderenfalls sind die Änderungen unkritisch.

## referentielle Aktionen im Überblick

- NO ACTION:** Die Operation auf der P-Tabelle wird zunächst ausgeführt; ob Dangling References in der C-Tabelle entstanden sind wird erst nach Abarbeitung aller durch die Operation auf der P-Tabelle direkt oder indirekt ausgelösten referentiellen Aktionen überprüft.
- RESTRICT:** Die Operation auf der P-Tabelle wird nur dann ausgeführt, wenn durch ihre Anwendung keine Dangling References in der C-Tabelle entstehen.
- CASCADE:** Die Operation auf der P-Tabelle wird ausgeführt. Erzeugt die DELETE/UPDATE-Operation Dangling References in der C-Tabelle, so werden die entsprechenden Zeilen der C-Tabelle ebenfalls mittels DELETE entfernt, bzw. mittels UPDATE geändert. Ist die C-Tabelle selbst P-Tabelle bezüglich einer anderen Bedingung, so wird das DELETE/UPDATE bezüglich der dort festgelegten Lösch/Änderungs-Regel weiter behandelt.
- SET DEFAULT:** Die Operation auf der P-Tabelle wird ausgeführt. In der C-Tabelle wird der entsprechende Fremdschlüsselwert durch die für die betroffenen Spalten in der C-Tabelle festgelegten DEFAULT-Werte ersetzt; es muss jedoch gewährleistet sein, daß entsprechende Schlüsselwerte in den P-Tabellen existieren.
- SET NULL:** Die Operation auf der P-Tabelle wird ausgeführt. In der C-Tabelle wird der entsprechende Fremdschlüsselwert spaltenweise durch NULL ersetzt. Voraussetzung ist hier, daß Nullwerte zulässig sind.

Bei Verwendung von RESTRICT können in Abhängigkeit von der Reihenfolge der Abarbeitung der FOREIGN KEY-Klauseln in Abhängigkeit vom Inhalt der Tabellen potentiell unterschiedliche Ergebnisse resultieren.

```
CREATE TABLE T1 (... PRIMARY KEY K1)
```

```
CREATE TABLE T2 (... PRIMARY KEY K2,
 FOREIGN KEY (K1) REFERENCES T1 (K1)
 ON DELETE CASCADE)
```

```
CREATE TABLE T3 (... PRIMARY KEY K3,
 FOREIGN KEY (K1) REFERENCES T1 (K1)
 ON DELETE CASCADE)
```

```
CREATE TABLE T4 (... PRIMARY KEY K4,
 FOREIGN KEY (K2) REFERENCES T2 (K2)
 ON DELETE CASCADE,
 FOREIGN KEY (K3) REFERENCES T3 (K3)
 ON DELETE RESTRICT)
```

Das Beispiel `DELETE FROM T1 WHERE K1 = 1` demonstriert, dass bzgl. T4 die RESTRICT-Aktion scheitert, sofern nicht vorher bzgl. T4 die CASCADE-Aktion durchgeführt wurde.

| T1 | K1 | T2 | K2 | K1 | T3 | K3 | K1 | T4 | K4 | K2 | K3 |
|----|----|----|----|----|----|----|----|----|----|----|----|
| 1  |    |    | a  | 1  |    | b  | 1  |    | c  | a  | b  |

## zum potentiellen Nichtdeterminismus

- ▶ Um nichtdeterministische Ausführungen dieser Art auszuschließen, wird vorgeschlagen, die Implementierung nach einer Strategie vorzunehmen, in der im Wesentlichen vor Berücksichtigung einer RESTRICT-Aktion alle CASCADE-Aktionen ausgeführt werden.
- ▶ Diese Strategie klärt offensichtlich obige Unbestimmtheit.
- ▶ Alternativ können gewisse Kombinationen von referentiellen Aktionen verboten werden. Ersetzt man RESTRICT durch NO ACTION in obigem Beispiel, so wird das Endergebnis wieder eindeutig, unabhängig von der Reihenfolge der betrachteten referentiellen Aktionen.

## 4.10.4 Dynamische Integrität und Trigger

- ▶ Komplementär zu deklarativen Integritätsbedingungen bieten Datenbanksysteme einen *Trigger*-Mechanismus an, mit dem in Form von Regeln definiert werden kann, welche Aktionen zur Gewährleistung der Integrität vorgenommen werden sollen, bzw. wie Verletzungen behandelt werden sollen.
- ▶ Damit können auch die Einschränkungen kommerzieller DBS bei der Verwendung der CHECK-Klausel umgangen werden.
- ▶ Mittels Trigger können wir insbesondere die Zulässigkeit von *Zustandsübergängen* kontrollieren, was mit deklarativen Integritätsbedingungen bisher aufgrund ihres Bezugs zu gerade einem Zustand nicht möglich ist.

- ▶ *Dynamische Integrität* beschäftigt sich somit mit der Formulierung von Integritätsbedingungen, die definieren, welche Zustandsübergänge auf den Tabellen zu einem Datenbank-Schema erlaubt sind.
- ▶ Sie müssen es uns dazu ermöglichen, in einem Ausdruck sowohl den alten, wie auch den neuen Zustand der Instanzen ansprechen zu können.
- ▶ Zur Gewährleistung der dynamischen Integrität bietet SQL einen mächtigen *Trigger-Mechanismus*. Trigger sind ein Spezialfall *aktiver Regeln*, in denen in Abhängigkeit von eingetretenen Ereignissen, sofern gewisse Bedingungen erfüllt sind, definierte Aktionen auf einer Datenbank ausgeführt werden (**Event-Condition-Action-Paradigma**).
- ▶ Innerhalb von SQL sind die auslösenden Operationen gerade *Einfügungen*, *Löschen* und *Änderungen* von Zeilen der Tabellen.

## Anwendungen

- ▶ Prüfen der Zulässigkeit von Werten vor der Durchführung von Änderungen, um so im Falle von Integritätsverletzungen diese korrigieren zu können.
- ▶ Protokollieren von auf sicherheitskritischen Tabellen vorgenommene Änderungen, z.B. mit Angabe der Benutzeridentifikation und Zugriffszeit.
- ▶ Implementierung von Änderungsoperationen auf Sichten.
- ▶ Definition von für Anwendungen verbindlichen (Geschäfts-)Regeln.

*Hinweis:* Oracle hat für Trigger eine leicht andere Syntax.

Ändert sich die Fläche einer Provinz, dann soll auch das entsprechende Land angepaßt werden.  
(SQL-Standard)

```
CREATE TRIGGER FlaecheAnpassen
 AFTER UPDATE OF Flaeche ON Provinz
 REFERENCING OLD AS Alt NEW AS Neu
 FOR EACH ROW
 UPDATE Land L
 SET L.Flaeche = L.Flaeche - Alt.Flaeche + Neu.Flaeche
 WHERE L.LCode = Alt.LCode
```

(Oracle)

```
CREATE TRIGGER FlaecheAnpassen
 AFTER UPDATE OF Flaeche ON Provinz
 REFERENCING OLD AS Alt NEW AS Neu
 FOR EACH ROW
 BEGIN
 UPDATE Land L
 SET L.Flaeche = L.Flaeche - :Alt.Flaeche + :Neu.Flaeche
 WHERE L.LCode = :Alt.LCode;
 END;
```

Die Tabelle Grenze soll antisymmetrisch sein; d.h. für je zwei Länder darf eine Nachbarschaftsbeziehung nur einmal enthalten sein.

```
CREATE TRIGGER antiSymGrenze
 BEFORE INSERT ON Grenze
 REFERENCING NEW AS Neu
 FOR EACH ROW
 WHEN EXISTS (
 SELECT * FROM Grenze G
 WHERE G.LCode1=Neu.LCode2 AND
 G.LCode2=Neu.LCode1)
 BEGIN
 SIGNAL SQLSTATE '75001';
 SET Message='Grenze bereits vorhanden'
 END
```

*Hinweis:* Oracle bietet eine Funktion zum Erzeugen eines Laufzeitfehlers:

```
raise_application_error(error_number, message)
```

Die Fehlernummer muss eine negative Nummer kleiner -20000 sein, z.B.:

```
raise_application_error(-20001, 'Grenze bereits vorhanden')
```

*Hinweis:* WHEN darf kein SFW enthalten.

Ergibt die Summe der Anteile an den Kontinenten für ein Land einen kleineren Wert als 100, so wird die Differenz dem Kontinent Atlantis zugeordnet.

```
CREATE TRIGGER Atlantis
 AFTER INSERT ON Lage
 FOR EACH STATEMENT
 WHEN EXISTS (
 SELECT * FROM Lage
 GROUP BY LCode
 HAVING (SUM(Prozent) < 100))
BEGIN
 INSERT INTO Lage
 SELECT LCode, 'Atlantis' AS Kontinent,
 (100 - SUM(Prozent)) AS Prozent
 FROM Lage
 GROUP BY LCode
 HAVING (SUM(Prozent) < 100)
END
```

*Hinweis:* Statement Trigger sind der Standardfall in Oracle. Die Angabe von FOR EACH STATEMENT entfällt. Weitere Einschränkungen in Oracle.

## Eigenschaften von Triggern

- ▶ Ein Trigger ist einer Tabelle zugeordnet. Er wird aktiviert durch das Eintreten eines Ereignisses (SQL-Anweisung): Einfügung, Änderung und Löschung von Zeilen.
- ▶ Der Zeitpunkt der Aktivierung ist entweder vor oder nach der eigentlichen Ausführung der entsprechenden aktivierenden Anweisung in der Datenbank. Ein Trigger kann die Ausführung der ihn aktivierenden Anweisung verhindern.
- ▶ Ein Trigger kann einmal pro aktivierender Anweisung (Statement-Trigger) oder einmal für jede betroffene Zeile (Row-Trigger) seiner Tabelle ausgeführt werden.
- ▶ Ein aktiverter Trigger wird ausgeführt, wenn seine Bedingung erfüllt ist.
- ▶ Der Rumpf eines Triggers enthält die auszuführenden SQL-Anweisungen.
- ▶ Mittels Transitions-Variablen OLD/NEW bzw. OLD/NEW TABLE<sup>2</sup> kann auf die Zeilen- bzw. Tabellen-Inhalte vor und nach der Ausführung der aktivierenden Aktion zugegriffen werden. Im Falle von Tabellen-Inhalten handelt es sich dabei um hypothetische Tabellen, die alle betroffenen Zeilen enthalten.
- ▶ Bei einem BEFORE-Trigger sind die *einzufügenden* Tupel nicht sichtbar in der betreffenden Tabelle; bei einem AFTER-Trigger sind sie sichtbar.
- ▶ Bei einem BEFORE-Trigger sind die zu *löschen* Tupel sichtbar in der betreffenden Tabelle; bei einem AFTER-Trigger sind sie nicht sichtbar.

<sup>2</sup>OLD/NEW TABLE wird von Oracle nicht unterstützt.

## Bemerkungen

- ▶ Trigger können selbst weitere Trigger aktivieren, wenn ein ausgelöster Trigger eine Tabelle modifiziert, über der selbst Trigger definiert sind. Eine Transaktion kann somit während ihrer Ausführung eine ganze Reihe von Triggern auslösen.
- ▶ Die Reihenfolge der Ausführung dieser Trigger ist ohne weitere Kontrolle nicht vorhersehbar.
- ▶ Um eine deterministische Ausführung zu gewährleisten, sind Einschränkungen an die möglichen Trigger-Definitionen, bzw. Anforderungen an ihre Ausführung zu berücksichtigen.
- ▶ Eine Aktivierungsfolge von Triggern kann insbesondere zyklisch sein (*rekursive Trigger*); die Terminierung einer solchen Folge ist im Allgemeinen nicht gesichert.
- ▶ Seit SQL:2008 können auch `INSTEAD OF`-Trigger verwendet werden.

Wird ein solcher Trigger aktiviert, dann werden *anstelle* der auslösenden Operation die Operationen des Triggers ausgeführt.

Ein sinnvolles Anwendungsgebiet für `INSTEAD OF`-Trigger ist die Realisierung von Änderungsoperationen auf Sichten auf den zugehörenden Basistabellen.

## 4.9 SQL und Programmiersprachen

- ▶ SQL hat eine tabellenorientierte Semantik.
- ▶ SQL hat eine im Vergleich zu einer Programmiersprache eingeschränkte Mächtigkeit, so dass es typischerweise ermöglicht wird, SQL von einem in einer gängigen Programmiersprache geschriebenen Programm aus aufzurufen.
- ▶ Anwendungen können so unter Ausnutzung der vollen Mächtigkeit einer Programmiersprache Datenbank-Instanzen verarbeiten.
- ▶ *Impedance-Mismatch*

## Ansätze der Integration

### (A) **SQL-Erweiterung:**

Erweiterung von SQL um imperative Sprachelemente zur Formulierung von *in der Datenbank gespeicherten* benutzerdefinierten Funktionen und Prozeduren.

### (B) **Statisches SQL:**

Einbettung von SQL-Ausdrücken in eine Programmiersprache.

### (C) **Dynamisches SQL:**

Übergabe eines datenabhängig gebildeten SQL-Ausdrucks an eine Datenbank während der Ausführung eines Programms.

## (A) SQL-Erweiterung

- ▶ Deklaration von *Variablen*
- ▶ Zuweisung von Werten an Variable
- ▶ Sequenz von Anweisungen
- ▶ bedingte Anweisungen und Wiederholungsanweisungen
- ▶ Funktionen und Prozeduren.<sup>1</sup>

Funktionen können als parametrisierbare virtuelle Sichten betrachtet werden.

*Hinweis:* Oracle verwendet teilweise eine andere Syntax als der SQL-Standard!

---

<sup>1</sup>In Oracle sind keine tabellenwertigen Parameter erlaubt.

## Funktionen als parametrisierbare virtuelle Sichten

| Benachbart |        | nachbarVon('D') |
|------------|--------|-----------------|
| LCode1     | LCode2 | LCode           |
| CH         | D      | CH              |
| CH         | F      | F               |
| CH         | I      |                 |
| D          | F      |                 |
| I          | F      |                 |
| :          | :      | :               |
| :          | :      |                 |

Berechne die benachbarten Länder zu einem gegebenen Land.

```
CREATE FUNCTION nachbarVon(X CHAR(4))
RETURNS TABLE (LCode CHAR(4))
RETURN (
 SELECT LCode2 AS LCODE FROM Benachbart WHERE LCODE1 = X
 UNION
 SELECT LCode1 AS LCode FROM Benachbart WHERE LCODE2 = X)
```

```
SELECT * FROM TABLE (nachbarVon('D')) T
```

*Hinweis:* Nicht direkt umsetzbar in Oracle, da Funktionen in Oracle keine Tabellen zurückgeben können!

## SQL-Standard vs. Oracle Syntax

Berechne die Anzahl Städte zu einem gegebenen Land. (SQL-Standard)

```
CREATE FUNCTION anzahlStaedte(Land CHAR(2))
RETURNS NUMBER
RETURN (
 SELECT count(*) FROM Stadt WHERE LCode = Land
)
```

Berechne die Anzahl Städte zu einem gegebenen Land. (Oracle)

```
CREATE FUNCTION anzahlStaedte(Land CHAR)
RETURN NUMBER IS
numCities NUMBER;
BEGIN
 SELECT count(*) INTO numCities FROM Stadt WHERE LCode = Land;
 RETURN numCities;
END;
```

Aufrufen der Funktion mittels Dummy Tabelle

```
SELECT anzahlStaedte('D') FROM Dual
```

Gib zu einem Land alle erreichbaren Länder mit der Mindestanzahl Grenzübergänge an  
(SQL-Standard; Oracle Version in den Übungen).

```
CREATE FUNCTION ErreichbarVon(Start CHAR(4))
RETURNS TABLE (Nach CHAR(4), Anzahl INTEGER)

BEGIN
CREATE TABLE Erreichbar (Nach CHAR(4), Anzahl INTEGER);
DECLARE alt, neu INTEGER;

/* Initialisiere mit den direkt erreichbaren Ländern */
INSERT INTO Erreichbar
 SELECT T.LCode2 AS Nach, 1 AS Anzahl
 FROM symBenachbart T WHERE T.LCode1 = Start;

/* Initialisiere Abbruchbedingung */
SET alt = 0;
SET neu = (SELECT COUNT(*) FROM Erreichbar);

/* Berechne iterativ die indirekt erreichbaren Länder */
WHILE (alt <> neu) DO
 SET alt = neu;
 INSERT INTO Erreichbar
 SELECT DISTINCT B.LCode2, (A.Anzahl + 1)
 FROM Erreichbar A, symBenachbart B
 WHERE A.Nach = B.LCode1 AND B.LCode2 <> Start
 AND B.LCode2 NOT IN (SELECT Nach FROM Erreichbar);
 SET neu = (SELECT COUNT(*) FROM Erreichbar);
END WHILE;

RETURN Erreichbar;
END
```

Initialisiere Erreichbar mit den *direkt* erreichbaren Ländern.

```
INSERT INTO Erreichbar
 SELECT T.LCode2 AS Nach, 1 AS Anzahl
 FROM symBenachbart T WHERE T.LCode1 = Start;
```

Die Berechnung soll terminieren, wenn nach einer Iteration kein neues *indirekt* erreichbares Land hinzugekommen ist.

```
SET neu = (SELECT COUNT(*) FROM Erreichbar);
WHILE (alt <> neu) DO
 SET alt = neu;
 INSERT INTO Erreichbar
 ...
 SET neu = (SELECT COUNT(*) FROM Erreichbar)
END WHILE;
```

Zyklen und Mehrfachberechnungen sollen vermieden werden.

- ▶ DISTINCT berücksichtigt den Fall, dass zum selben Land mehrere Wege gleicher Länge existieren können.
- ▶ Zyklen werden berücksichtigt, indem ein weiteres Land nur dann hinzugefügt wird, wenn das neu berechnete Land nicht bereits als erreichbares Land bekannt ist.
- ▶ Zyklen zum Start werden ebenfalls ausgeschlossen.

```
INSERT INTO Erreichbar
 SELECT DISTINCT B.LCode2, (A.Anzahl + 1)
 FROM Erreichbar A, symBenachbart B
 WHERE A.Nach = B.LCode1 AND B.LCode2 <> Start
 AND B.LCode2 NOT IN (SELECT Nach FROM Erreichbar);
```

## (B) Statisches SQL

- ▶ Stehen die auszuführenden SQL-Anfragen bereits zur Übersetzungszeit eines Programms als Teil des Programmcodes fest, dann redet man von einer *statischen* Einbettung von SQL in eine Programmiersprache.
- ▶ Eine datenabhängige Änderung der Anfragen während der Ausführung des Programms ist dann nicht mehr möglich.
- ▶ Die Ergebnisse einer SQL-Anfrage werden innerhalb des Programms mittels eines Cursors zugänglich gemacht.

### Anlegen eines Cursors

```
EXEC SQL DECLARE StadtCursor CURSOR FOR
 SELECT DISTINCT S.SName, L.LName
 FROM Stadt S, Land L
 WHERE S.LCode = L.LCode;
```

### Öffnen einen Cursors

```
EXEC SQL OPEN StadtCursor;
```

### Aktuelle Zeile eines Cursors auslesen

```
EXEC SQL FETCH StadtCursor INTO :stadtName, :landName;
```

### Cursor schließen

```
EXEC SQL CLOSE StadtCursor;
```

## (C) Dynamisches SQL

- ▶ Können wir einen SQL-Ausdruck während der Ausführung eines Programms in Form einer Zeichenkette an das Datenbanksystem übergeben, so redet man von *dynamischem* SQL.
- ▶ Standardisierte Schnittstellen: ODBC und JDBC.  
Erweiterung von Java: SQLJ.

### Beispiel: Berechnen einer Adjazenzmatrix in PHP

```
<?php
sql_connect('Mondial','lausen','buch');
$AdjazenzMatrix = array();
$query = 'SELECT * FROM Benachbart';
$result = sql_query($query);
while ($row = sql_fetch_assoc($result)) {
 $i = $row['von']; $j = $row['nach'];
 $AdjazenzMatrix[$i][$j] = 1;
}
?>
```

## 4.10 Integrität und Trigger

- ▶ Im Allgemeinen sind nur solche Instanzen einer Datenbank erlaubt, deren Relationen die der Datenbank bekannten *Integritätsbedingungen* (IB) erfüllen.
- ▶ Integritätsbedingungen können *explizit* durch den Benutzer definiert werden, durch die Definition von konkreten Schemata *implizit* erzwungen werden, oder bereits dem relationalen Datenmodell *inhärent* sein.
- ▶ *Inhärente Bedingungen*: Attributwerte sind skalar; Relationen, abgesehen von Duplikaten, verhalten sich wie Mengen, d.h. ohne weitere Angaben haben sie insbesondere keine Sortierung.
- ▶ *Implizite Bedingungen*: Werte der Attribute eines Primärschlüssels dürfen keine Nullwerte enthalten
- ▶ *Explizite Bedingungen*: Werden als Teil der CREATE TABLE-Klausel, bzw. der CREATE SCHEMA-Klausel definiert.

- ▶ Integritätsbedingungen sind von ihrer Natur aus deklarativ: sie definieren die zulässigen Instanzen, ohne auszudrücken, wie eine Gewährleistung der Integrität implementiert werden kann.
- ▶ Eine wichtige Klasse von deklarativen Integritätsbedingungen sind *Fremdschlüsselbedingungen*, die gewährleisten, dass keine *dangling* Referenzen zwischen den Tupeln in den Tabellen bestehen.
- ▶ Komplementär zu den deklarativen Bedingungen bieten Datenbanksysteme einen *Trigger*-Mechanismus an, mit dem in Form von Regeln definiert werden kann, welche Aktionen zur Gewährleistung der Integrität vorgenommen werden sollen, bzw. wie Verletzungen behandelt werden sollen.
- ▶ Mittels Trigger können wir insbesondere die Zulässigkeit von *Zustandsübergängen* kontrollieren, was mit Integritätsbedingungen aufgrund ihres Bezugs zu gerade einem Zustand nicht möglich ist.

## 4.10.1 Schlüsselbedingungen

Zu jeder Tabelle werden typischerweise ein *Primärschlüssel* und möglicherweise weitere Schlüssel festgelegt (**UNIQUE**-Klausel).

In jeder Instanz zu der Tabelle Land können alle Zeilen eindeutig durch ihren Spaltenwert zu LCode, oder alternativ, durch ihren Spaltenwert zu LName identifiziert werden.

```
CREATE TABLE Land (
 LName VARCHAR(35) UNIQUE,
 LCode VARCHAR(4) PRIMARY KEY,
 ...
```

Identifizierendes Kriterium für die Tabelle Stadt sei SName, LCode, PName, bzw. alternativ, LGrad, BGrad.

```
CREATE TABLE Stadt (
 ...
 PRIMARY KEY (SName,LCode,PName),
 UNIQUE (LGrad,BGrad))
```

## 4.10.2 Statische Integrität

### CHECK-Klausel

- ▶ *Statische* Integrität definiert unter Verwendung der CHECK-Klausel, welche Instanzen eines Schemas zulässig sind.
- ▶ Mittels der CHECK-Klausel können die zulässigen Werte eines Datentyps und die für eine Spalte einer konkreten Tabelle zu verwendenden Werte weiter eingeschränkt werden.
- ▶ Darüberhinaus können beliebige, mittels SQL-Anfrageausdrücken gebildete, Bedingungen über den Instanzen der Tabellen eines Schemas ausgedrückt werden.

## Wertebereichsbedingungen mittels NONNULL, DEFAULT und CREATE DOMAIN

```
LName VARCHAR(35) NONNULL
Prozent NUMBER DEFAULT 100
```

```
CREATE DOMAIN meineStädte VARCHAR(35) DEFAULT '?'
CREATE TABLE Stadt (
 SName meineStädte,
 :
)
```

## Wertebereichsbedingungen mittels CHECK

```
CREATE DOMAIN meineStädte VARCHAR(35),
 DEFAULT 'Paris',
 CHECK (VALUE IN ('Berlin', 'Paris',
 'London', 'Rom'))
```

## Spaltenbedingung mittels CHECK

```
CREATE TABLE Stadt (
 SName meineStädte,
 :
 LGrad NUMBER
 CHECK (LGrad BETWEEN -180 AND 180),
 BGrad NUMBER
 CHECK (BGrad BETWEEN -90 AND 90),
 ...)
```

## Spalten- und Tabellenbedingung mittels CHECK

Die Summe aller Anteile an unterschiedlichen Kontinenten eines Landes muss 100 ergeben.

```
CREATE TABLE Lage (
 LCode VARCHAR(4),
 Kontinent VARCHAR(35),
 Prozent NUMBER
 CHECK (Prozent BETWEEN 0 AND 100),
 CHECK (100 = (SELECT SUM(L.Prozent) FROM Lage L
 WHERE LCode = L.LCode)))
```

## Assertion

- ▶ Spalten- und Tabellenbedingungen sind erfüllt, wenn jede Zeile der betreffenden Tabelle sie erfüllt.
- ▶ Spalten- und Tabellenbedingungen sind somit *implizit*  $\forall$ -quantifiziert über den Zeilen der Tabelle.
- ▶ Alternativ können wir die explizitere Form einer ASSERTION wählen

### CHECK Assertion:

Die Summe aller Anteile an unterschiedlichen Kontinenten eines Landes muss 100 ergeben.

```
CREATE ASSERTION AssertLage (
 CHECK (NOT EXISTS (
 SELECT LCode FROM Lage
 GROUP BY LCode
 HAVING (SUM(Prozent) <> 100))
)
)
```

## CHECK und ASSERTION in ORACLE, SQL Server, etc.

- ▶ Typischerweise sind Sub-Queries in der CHECK-Klausel verboten.
- ▶ ASSERTION wird nicht unterstützt.

**Gewährleistung von Integritätsbedingungen in solchen Fällen mittels  
TRIGGER**

## 4.10.3 Fremdschlüsselbedingungen

- ▶ *Fremdschlüsselbedingungen* werden als Teil der CREATE TABLE-Klausel definiert.
- ▶ Sie sind formal sogenannte *Inklusionsabhängigkeiten*: zu jedem von null verschiedenen Fremdschlüsselwert in einer Zeile der *referenzierenden* Tabelle, der C- (child-) Tabelle, existiert ein entsprechender Schlüsselwert in einer Zeile der *referenzierten* Tabelle, der P- (parent-) Tabelle.
- ▶ Man redet hier auch von *referentieller* Integrität. Zur Definition von Fremdschlüsselbedingungen steht die FOREIGN KEY-Klausel zur Verwendung in der C-Tabelle zur Verfügung.

Die Spalte LCode innerhalb der Tabelle Provinz enthält Werte des Schlüssels LCode der Tabelle Land. Zu jeder Zeile in Provinz muss eine Zeile in Land existieren, deren Schlüsselwert gleich dem Fremdschlüsselwert ist.

```
CREATE TABLE Provinz (
 PName VARCHAR(35),
 LCode VARCHAR(4),
 Fläche NUMBER
 PRIMARY KEY (PName, LCode),
 FOREIGN KEY (LCode) REFERENCES Land (LCode))
```

Für die Tabelle Stadt sind zwei Fremdschlüsselbeziehungen relevant. Einmal müssen die referenzierten Länder in der Tabelle zu Land existieren, und zum andern entsprechend die Provinzen. Letzterer Fremdschlüssel besteht aus zwei Spalten. Die Zuordnung der einzelnen Spalten des Fremd- und Primärschlüssels ergeben sich aus der Reihenfolge des Hinschreibens.

```
CREATE TABLE Stadt (
 :
 PRIMARY KEY (SName, LCode, PName),
 FOREIGN KEY (LCode) REFERENCES Land (LCode),
 FOREIGN KEY (LCode, PName) REFERENCES Provinz (LCode, PName))
```

## referentielle Aktionen

Zur Gewährleistung der referentiellen Integrität werden sogenannte *referentielle Aktionen* zur Ausführung bezüglich der C-Tabellen definiert. Aufgabe dieser Aktionen ist die Kompensierung von durch DELETE- und UPDATE-Operationen auf der zugehörigen P-Tabelle verursachten Verletzungen der Integrität.

- ▶ Änderungen der P-Tabelle werden auf die C-Tabelle übertragen.  
(CASCADE)
- ▶ Die Änderung der P-Tabelle wird im Falle einer Verletzung der referentiellen Integrität einer C-Tabelle abgebrochen.  
(NO ACTION oder RESTRICT)
- ▶ Der Fremdschlüsselwert der C-Tabelle wird angepaßt.  
(SET NULL oder SET DEFAULT)

*Hinweis:* Oracle unterstützt bei UPDATE-Operationen nur die Aktion NO ACTION und bei DELETE-Operationen die Aktionen CASCADE, SET NULL und NO ACTION (keine Angabe entspricht NO ACTION).

Wird der Code eines Landes geändert oder das Land gelöscht, so sollen die neuen Codes bei den zugehörigen Provinzen nachgezogen werden, bzw. auch die Provinzen des gelöschten Landes gelöscht werden.

```
CREATE TABLE Provinz (
 :
 FOREIGN KEY (LCode) REFERENCES Land (LCode)
 ON DELETE CASCADE ON UPDATE CASCADE)
```

Werden Provinzen gelöscht, so sollen ihre Städte weiter in der Datenbank bestehen bleiben, wobei der betreffende Fremdschlüsselwert Nullwerte erhält. Änderungen eines Provinzschlüssels sollen auf die betroffenen Städte übertragen werden.

```
CREATE TABLE Stadt (
 :
 PRIMARY KEY (SNAME)
 FOREIGN KEY (LCode, PName)
 REFERENCES Provinz (LCode, PName)
 ON DELETE SET NULL ON UPDATE CASCADE)
```

## Warum werden nur DELETE- und UPDATE-Operationen auf den entsprechenden P-Tabellen betrachtet?

- ▶ Einfügen bezüglich der P-Tabelle ist für die referentielle Integrität immer unkritisch.
- ▶ Löschen bezüglich der C-Tabelle ist für die referentielle Integrität immer unkritisch.
- ▶ Einfügen bezüglich der C-Tabelle oder Ändern bezüglich der C-Tabelle, die einen Fremdschlüsselwert erzeugen, zu dem kein Schlüssel in der P-Tabelle existiert, sind immer primär unzulässig, da von Änderungen in den C-Tabellen im Allgemeinen kein sinnvoller Rückschluss auf Änderungen der P-Tabellen möglich ist; anderenfalls sind die Änderungen unkritisch.

## referentielle Aktionen im Überblick

- NO ACTION:** Die Operation auf der P-Tabelle wird zunächst ausgeführt; ob Dangling References in der C-Tabelle entstanden sind wird erst nach Abarbeitung aller durch die Operation auf der P-Tabelle direkt oder indirekt ausgelösten referentiellen Aktionen überprüft.
- RESTRICT:** Die Operation auf der P-Tabelle wird nur dann ausgeführt, wenn durch ihre Anwendung keine Dangling References in der C-Tabelle entstehen.
- CASCADE:** Die Operation auf der P-Tabelle wird ausgeführt. Erzeugt die DELETE/UPDATE-Operation Dangling References in der C-Tabelle, so werden die entsprechenden Zeilen der C-Tabelle ebenfalls mittels DELETE entfernt, bzw. mittels UPDATE geändert. Ist die C-Tabelle selbst P-Tabelle bezüglich einer anderen Bedingung, so wird das DELETE/UPDATE bezüglich der dort festgelegten Lösch/Änderungs-Regel weiter behandelt.
- SET DEFAULT:** Die Operation auf der P-Tabelle wird ausgeführt. In der C-Tabelle wird der entsprechende Fremdschlüsselwert durch die für die betroffenen Spalten in der C-Tabelle festgelegten DEFAULT-Werte ersetzt; es muss jedoch gewährleistet sein, daß entsprechende Schlüsselwerte in den P-Tabellen existieren.
- SET NULL:** Die Operation auf der P-Tabelle wird ausgeführt. In der C-Tabelle wird der entsprechende Fremdschlüsselwert spaltenweise durch NULL ersetzt. Voraussetzung ist hier, daß Nullwerte zulässig sind.

Bei Verwendung von RESTRICT können in Abhängigkeit von der Reihenfolge der Abarbeitung der FOREIGN KEY-Klauseln in Abhängigkeit vom Inhalt der Tabellen potentiell unterschiedliche Ergebnisse resultieren.

```
CREATE TABLE T1 (... PRIMARY KEY K1)
```

```
CREATE TABLE T2 (... PRIMARY KEY K2,
 FOREIGN KEY (K1) REFERENCES T1 (K1)
 ON DELETE CASCADE)
```

```
CREATE TABLE T3 (... PRIMARY KEY K3,
 FOREIGN KEY (K1) REFERENCES T1 (K1)
 ON DELETE CASCADE)
```

```
CREATE TABLE T4 (... PRIMARY KEY K4,
 FOREIGN KEY (K2) REFERENCES T2 (K2)
 ON DELETE CASCADE,
 FOREIGN KEY (K3) REFERENCES T3 (K3)
 ON DELETE RESTRICT)
```

Das Beispiel `DELETE FROM T1 WHERE K1 = 1` demonstriert, dass bzgl. T4 die RESTRICT-Aktion scheitert, sofern nicht vorher bzgl. T4 die CASCADE-Aktion durchgeführt wurde.

| T1 | K1 | T2 | K2 | K1 | T3 | K3 | K1 | T4 | K4 | K2 | K3 |
|----|----|----|----|----|----|----|----|----|----|----|----|
| 1  |    |    | a  | 1  |    | b  | 1  |    | c  | a  | b  |

## zum potentiellen Nichtdeterminismus

- ▶ Um nichtdeterministische Ausführungen dieser Art auszuschließen, wird vorgeschlagen, die Implementierung nach einer Strategie vorzunehmen, in der im Wesentlichen vor Berücksichtigung einer RESTRICT-Aktion alle CASCADE-Aktionen ausgeführt werden.
- ▶ Diese Strategie klärt offensichtlich obige Unbestimmtheit.
- ▶ Alternativ können gewisse Kombinationen von referentiellen Aktionen verboten werden. Ersetzt man RESTRICT durch NO ACTION in obigem Beispiel, so wird das Endergebnis wieder eindeutig, unabhängig von der Reihenfolge der betrachteten referentiellen Aktionen.

## 4.11 Arbeiten mit Schema-Definitionen

Alle mit CREATE definierten Konstrukte sind Teil eines *Datenbankschemas*.

- ▶ SQL bietet Anweisungen an, mit denen existierende Schemata erweitert, oder auch einmal festgelegte Definitionen innerhalb eines Schemas wieder entfernt oder geändert werden können.
- ▶ Um einen nachträglichen Bezug zu existierenden Definitionen zu haben, müssen diese Definitionen mit einem Namen versehen werden.
- ▶ Die Zuordnung eines Namens ist auch sinnvoll, um im Falle von auftretenden Datenbankfehlern, wie Integritätsverletzungen, einen konkreten Bezug innerhalb einer Fehlernachricht zu bekommen.

### Definition eines Schemas.

```
CREATE SCHEMA MondialDatenbank
```

## Änderungen eines Schemas

- ▶ Mittels einer **DROP**-Anweisung können existierende (mit CREATE erzeugte) Wertebereiche, Tabellen, Sichten und Assertions entfernt werden.
- ▶ Mittels **ALTER** können nachträglich Änderungen vorgenommen werden.
- ▶ Spalten und Integritätsbedingungen können mittels **DROP** entfernt, bzw. mittels **ADD** nachträglich hinzugefügt werden.

Die Tabelle Land wird um eine Spalte Einwohner erweitert; des Weiteren wird die Spalte Hauptstadt entfernt.

```
ALTER TABLE Land
 ADD COLUMN Einwohner NUMBER

ALTER TABLE Land
 DROP COLUMN HStadt
```

## Zyklische Fremdschlüssel Definitionen

- ▶ Bei zyklischen Beziehungen kann zunächst die zuerst erstellte Tabelle ohne REFERENCES-Klausel definiert werden.
- ▶ Nach erfolgter Definition der zweiten Tabelle wird die REFERENCES-Klausel dann mittels ALTER Table nachträglich hinzugefügt.

## DEFERRED und IMMEDIATE

- ▶ Zu jeder Integritätsbedingung kann mittels IMMEDIATE und DEFERRED festgelegt werden, ob sie direkt nach Ausführung einer SQL-Anweisung, oder nach Ausführung einer sie enthaltenden Transaktion überprüft werden soll.
- ▶ Diese Angaben können nachträglich modifiziert werden. Bedingungen können als DEFERRABLE oder NOT DEFERRABLE definiert werden. INITIALLY definiert den gültigen Modus für eine Transaktionen zu Beginn ihres Ablaufs. Mittels der Anweisung SET CONSTRAINTS kann während der Ausführung einer Transaktion eine Bedingung auf IMMEDIATE oder DEFERRED gesetzt werden.

## Referentielle Integrität: Handhabung zyklischer Definitionen

```
CREATE TABLE Z1 (
K1 CHAR(2),
K2 CHAR(2),
PRIMARY KEY (K1));
```

```
CREATE TABLE Z2 (
K2 CHAR(2),
K1 CHAR(2),
PRIMARY KEY (K2));
```

```
/* INSERT */
```

```
ALTER TABLE Z1 ADD CONSTRAINT cyclic1
FOREIGN KEY (K2)
REFERENCES Z2 (K2) ON DELETE CASCADE;
```

```
ALTER TABLE Z2 ADD CONSTRAINT cyclic2
FOREIGN KEY (K1)
REFERENCES Z1 (K1) ON DELETE CASCADE;
```

```
/* DELETE */
/* COMMIT */
```

```
CREATE TABLE Z2 (
K2 CHAR(2),
K1 CHAR(2),
PRIMARY KEY (K2));
```

```
CREATE TABLE Z1 (
K1 CHAR(2),
K2 CHAR(2),
PRIMARY KEY (K1),
CONSTRAINT cyclic1
FOREIGN KEY (K2) REFERENCES Z2 (K2)
ON DELETE CASCADE
DEFERRABLE INITIALLY DEFERRED);
```

```
ALTER TABLE Z2 ADD CONSTRAINT cyclic2
FOREIGN KEY (K1) REFERENCES Z1 (K1)
ON DELETE CASCADE
DEFERRABLE INITIALLY DEFERRED;
```

```
/* INSERT, DELETE */
```

```
SET CONSTRAINTS cyclic1, cyclic2 IMMEDIATE;
```

```
/* INSERT, DELETE, UPDATE */
/* COMMIT */
```

## LIKE- und AS-Klausel

- ▶ SQL:2003 beinhaltet die beiden Klauseln  
`CREATE TABLE LIKE` bzw. `CREATE TABLE AS`.
- ▶ Im ersten Fall wird die komplette Spaltendefinition einer existierenden Tabelle in die neu zu definierende Tabelle übernommen, wobei zusätzlich weitere neue Spalten hinzugenommen werden können.
- ▶ Im zweiten Fall wird die neue Tabelle mittels einer beliebigen SFW-Anweisung definiert. Es können somit beliebige Spalten aus existierenden Tabellen ausgewählt werden und es wird gleichzeitig eine Instanz der neuen Tabelle erzeugt.
- ▶ In beiden Varianten der CREATE-Klausel sind die neuen Tabellen unabhängig von ihren Ursprüngen.

Die Tabelle Stadt\_1 ist wie Stadt definiert und enthält zusätzlich eine Spalte Fläche.

```
CREATE TABLE Stadt_1 (
 LIKE Stadt
 Fläche NUMBER)
```

Die Tabelle Stadt\_2 hat den Inhalt von Stadt und zusätzlich für jede Stadt den Anteil an der Gesamtbevölkerung ihres Landes.

```
CREATE TABLE Stadt_2 AS (
 SELECT S1.*,
 (SELECT S1.Einwohner/SUM(S2.Einwohner)
 FROM Stadt S2 WHERE S1.LCode = S2.LCode) AS Anteil
 FROM Stadt S1)
WITH DATA
```

## 4.12 Verschiedenes

### FROM DUAL

In manchen Situationen ist das Resultat einer SFW-Anfrage unabhängig von den Tabellen der FROM-Klausel. Um der SFW-Syntax zu genügen verwendet man dann als Tabelle eine Dummy-Tabelle (in Oracle existent mit Namen DUAL), die genau eine Zeile enthält.

```
SELECT (ln(50) * sin(300)) / tan(0.5) FROM DUAL;
```

```
SELECT (SELECT min(Einwohner) FROM Stadt) AS Klein,
 (SELECT max(Einwohner) FROM Stadt) AS Gross
FROM DUAL;
```

## In-line/temporary-table Sichten mittels WITH

- ▶ Verwendet wie lokal definierte virtuelle Sicht (*in-line view*), bzw. wie eine lokale materialisierte Sicht (*temporäre Tabelle*)
- ▶ Führt zu einer klareren Struktur

Welches Land hat die kleinste durchschnittliche StadtEinwohnerzahl?

Vergleiche:

```
SELECT DISTINCT S.LCode FROM Stadt S
WHERE (SELECT AVG(Einwohner) FROM Stadt WHERE LCode = S.LCode) =
 (SELECT MIN(Einw) FROM (SELECT AVG(Einwohner) AS Einw
 FROM Stadt GROUP BY Lcode));
```

mit:

WITH

```
avgEinwohner AS (SELECT LCode, AVG(Einwohner) AS Einw
 FROM Stadt GROUP BY Lcode),
minEinwohner AS (SELECT MIN(Einw) as minE FROM avgEinwohner)
```

```
SELECT LCode FROM avgEinwohner S
WHERE S.Einw = (SELECT minE FROM minEinwohner);
```

## Online Analytical Processing (OLAP) mittels ROLLUP und CUBE

- ▶ *Online Transaction Processing (OLTP):*  
Anwendungen auf den aktuellen operationalen Daten.
- ▶ *Online Analytical Processing (OLAP):*  
Datawarehouse-Anwendungen auf ausgelagerten, typischerweise historischen Daten.

### Beispiel: Analyse von Verkaufszahlen

```
SELECT Model, Year, Color, sum(Sales)
FROM SALES
GROUP BY ROLLUP(Model, Year, Color)
```

```
SELECT Model, Year, Color, sum(Sales)
FROM SALES
GROUP BY CUBE(Model, Year, Color)
```

# ROLLUP

| SALES |      |       |       |
|-------|------|-------|-------|
| Model | Year | Color | Sales |
| Chevy | 1990 | red   | 5     |
| Chevy | 1990 | white | 87    |
| Chevy | 1990 | blue  | 62    |
| Chevy | 1991 | red   | 54    |
| Chevy | 1991 | white | 95    |
| Chevy | 1991 | blue  | 49    |
| Chevy | 1992 | red   | 31    |
| Chevy | 1992 | white | 54    |
| Chevy | 1992 | blue  | 71    |
| Ford  | 1990 | red   | 64    |
| Ford  | 1990 | white | 62    |
| Ford  | 1990 | blue  | 63    |
| Ford  | 1991 | red   | 52    |
| Ford  | 1991 | white | 9     |
| Ford  | 1991 | blue  | 55    |
| Ford  | 1992 | red   | 27    |
| Ford  | 1992 | white | 62    |
| Ford  | 1992 | blue  | 39    |

⇒  
ROLLUP

| ROLLUP |      |       |       |
|--------|------|-------|-------|
| Model  | Year | Color | Sales |
| Chevy  | 1990 | blue  | 62    |
| Chevy  | 1990 | red   | 5     |
| Chevy  | 1990 | white | 87    |
| Chevy  | 1990 | ALL   | 154   |
| Chevy  | 1991 | blue  | 49    |
| Chevy  | 1991 | red   | 54    |
| Chevy  | 1991 | white | 95    |
| Chevy  | 1991 | ALL   | 198   |
| Chevy  | 1992 | blue  | 71    |
| Chevy  | 1992 | red   | 31    |
| Chevy  | 1992 | white | 54    |
| Chevy  | 1992 | ALL   | 156   |
| Chevy  | ALL  | ALL   | 508   |
| Ford   | 1990 | blue  | 63    |
| Ford   | 1990 | red   | 64    |
| Ford   | 1990 | white | 62    |
| Ford   | 1990 | ALL   | 189   |
| Ford   | 1991 | blue  | 55    |
| Ford   | 1991 | red   | 52    |
| Ford   | 1991 | white | 9     |
| Ford   | 1991 | ALL   | 116   |
| Ford   | 1992 | blue  | 39    |
| Ford   | 1992 | red   | 27    |
| Ford   | 1992 | white | 62    |
| Ford   | 1992 | ALL   | 128   |
| Ford   | ALL  | ALL   | 433   |
| ALL    | ALL  | ALL   | 941   |

# CUBE

SALES

| Model | Year | Color | Sales |
|-------|------|-------|-------|
| Chevy | 1990 | red   | 5     |
| Chevy | 1990 | white | 87    |
| Chevy | 1990 | blue  | 62    |
| Chevy | 1991 | red   | 54    |
| Chevy | 1991 | white | 95    |
| Chevy | 1991 | blue  | 49    |
| Chevy | 1992 | red   | 31    |
| Chevy | 1992 | white | 54    |
| Chevy | 1992 | blue  | 71    |
| Ford  | 1990 | red   | 64    |
| Ford  | 1990 | white | 62    |
| Ford  | 1990 | blue  | 63    |
| Ford  | 1991 | red   | 52    |
| Ford  | 1991 | white | 9     |
| Ford  | 1991 | blue  | 55    |
| Ford  | 1992 | red   | 27    |
| Ford  | 1992 | white | 62    |
| Ford  | 1992 | blue  | 39    |

⇒ CUBE

DATA CUBE

| Model | Year | Color | Sales |
|-------|------|-------|-------|
| Chevy | 1990 | blue  | 62    |
| Chevy | 1990 | red   | 5     |
| Chevy | 1990 | white | 87    |
| Chevy | 1990 | ALL   | 154   |
| Chevy | 1991 | blue  | 49    |
| Chevy | 1991 | red   | 54    |
| Chevy | 1991 | white | 95    |
| Chevy | 1991 | ALL   | 198   |
| Chevy | 1992 | blue  | 71    |
| Chevy | 1992 | red   | 31    |
| Chevy | 1992 | white | 54    |
| Chevy | 1992 | ALL   | 156   |
| Chevy | ALL  | blue  | 182   |
| Chevy | ALL  | red   | 90    |
| Chevy | ALL  | white | 236   |
| Chevy | ALL  | ALL   | 508   |
| Ford  | 1990 | blue  | 63    |
| Ford  | 1990 | red   | 64    |
| Ford  | 1990 | white | 62    |
| Ford  | 1990 | ALL   | 189   |
| Ford  | 1991 | blue  | 55    |
| Ford  | 1991 | red   | 52    |
| Ford  | 1991 | white | 9     |
| Ford  | 1991 | ALL   | 116   |
| Ford  | 1992 | blue  | 39    |
| Ford  | 1992 | red   | 27    |
| Ford  | 1992 | white | 62    |
| Ford  | 1992 | ALL   | 128   |
| Ford  | ALL  | blue  | 157   |
| Ford  | ALL  | red   | 143   |
| Ford  | ALL  | white | 133   |
| Ford  | ALL  | ALL   | 433   |

# empfohlene Lektüre

## Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals\*

JIM GRAY  
SURAJIT CHAUDHURI  
ADAM BOSWORTH  
ANDREW LAYMAN  
DON REICHART  
MURALI VENKATRAO  
*Microsoft Research, Advanced Technology Division, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052*

Gray@Microsoft.com  
SurajitC@Microsoft.com  
AdamB@Microsoft.com  
AndrewL@Microsoft.com  
DonRei@Microsoft.com  
MuraliV@Microsoft.com

FRANK PELLOW  
HAMID PIRAHESH  
*IBM Research, 500 Harry Road, San Jose, CA 95120*

Pellow@vnet.IBM.com  
Pirahesh@Almaden.IBM.com

**Editor:** Usama Fayyad

*Received July 2, 1996; Revised November 5, 1996; Accepted November 6, 1996*

**Abstract.** Data analysis applications typically aggregate data across many dimensions looking for anomalies or unusual patterns. The SQL aggregate functions and the GROUP BY operator produce zero-dimensional or one-dimensional aggregates. Applications need the  $N$ -dimensional generalization of these operators. This paper defines that operator, called the **data cube** or simply **cube**. The cube operator generalizes the histogram, cross-tabulation, roll-up, drill-down, and sub-total constructs found in most report writers. The novelty is that cubes are relations. Consequently, the cube operator can be imbedded in more complex non-procedural data analysis programs. The cube operator treats each of the  $N$  aggregation attributes as a dimension of  $N$ -space. The aggregate of a particular set of attribute values is a point in this space. The set of points forms an  $N$ -dimensional cube. Super-aggregates are computed by aggregating the  $N$ -cube to lower dimensional spaces. This paper (1) explains the cube and roll-up operators, (2) shows how they fit in SQL, (3) explains how users can define new aggregate functions for cubes, and (4) discusses efficient techniques to compute the cube. Many of these features are being added to the SQL Standard.

1

<sup>1</sup> In: **Data Mining and Knowledge Discovery 1, 1997.**

## 4.13 Zugriffskontrolle

- ▶ Datenbanken enthalten häufig vertrauliche Informationen, die nicht jedem Anwender zur Verfügung stehen dürfen.
- ▶ Außerdem wird man im Allgemeinen nicht allen Anwendern dieselben Möglichkeiten zur Verarbeitung der Daten einräumen wollen, da Änderungen der Daten unter Umständen kritisch sind, auch wenn die Daten an sich nicht vertraulich sind.
- ▶ Zugriffsrechte können nicht nur einzelnen Benutzern zugewiesen werden, sondern es können Zugriffsrechte auch an *Rollen* gebunden werden.

### Rollen

- ▶ CREATE ROLE <Rollenname>
- ▶ DROP ROLE <Rollenname>
- ▶ GRANT <Rollenname> TO <Benutzerliste>
- ▶ REVOKE <Rollenname> FROM <Benutzerliste>

## Benutzer und Objekte

PUBLIC erteilte Rechte sind automatisch für alle Benutzer gültig.

- ▶ Zugriffskontrolle mittels GRANT und REVOKE.
- ▶ Objekte, die mit Zugriffsrechten versehen werden können, sind unter anderem: Tabellen, Spalten, Sichten, Wertebereiche (Domains) und Routinen (Funktionen und Prozeduren).

## Rechte

- ▶ Die möglichen Rechte sind SELECT, INSERT, UPDATE, DELETE, REFERENCES, USAGE, TRIGGER und EXECUTE, wobei nicht jedes Recht für jede Art von Objekten angewendet werden kann.
- ▶ Syntax:

```
GRANT <Liste von Rechten>
ON <Objekt>
TO <Liste von Benutzern> [WITH GRANT OPTION]
REVOKE [GRANT OPTION FOR] <Liste von Rechten>
ON <Objekt>
FROM <Liste von Benutzern> {RESTRICT | CASCADE}
```

- ▶ Mit GRANT OPTION erhaltene Rechte können weitergereicht werden.

## Verwaltung von Rechten

Der Erzeuger einer Basistabelle hat zu dieser Tabelle alle für eine Tabelle möglichen Rechte, d.h. die Rechte SELECT, INSERT, UPDATE, DELETE, REFERENCES und TRIGGER.

Beispiel:

Angenommen der Benutzer Admin hat alle Tabellen der Mondial-Datenbank erzeugt und besitzt somit alle Rechte.

Das Leserecht zur Tabelle Land soll allen Benutzern (PUBLIC) erteilt werden.

Außerdem sollen den Benutzern Assistent und Tutor die Rechte zum Lesen, Einfügen, Löschen und Ändern zugeteilt werden in der Weise, dass diese Benutzer diese Rechte auch anderen Benutzern erteilen dürfen.

Schließlich soll der Benutzer SysProg die Rechte REFERENCES und TRIGGER erhalten.

```
GRANT SELECT ON Land TO PUBLIC
```

```
GRANT SELECT, INSERT, DELETE, UPDATE
ON Land TO Assistent, Tutor WITH GRANT OPTION
```

```
GRANT REFERENCES, TRIGGER
ON Land TO SysProg
```

## Bemerkungen

- ▶ Die Definition von Fremdschlüsseln, Integritätsbedingungen und Triggern darf nur bei Besitz entsprechender Rechte erlaubt sein, da sonst indirekt auf den Inhalt einer Tabelle geschlossen werden könnte.
- ▶ Jeder Benutzer, der das SELECT-Recht besitzt, darf eine Sicht über der entsprechenden Tabelle definieren.
- ▶ Wird eine Sicht über mehreren Tabellen definiert, dann muss das SELECT-Recht zu allen diesen Tabellen zugeteilt sein. Weitere Rechte zu der Sicht existieren nur dann, wenn diese Rechte auch für alle der Sicht zugrunde liegenden Tabellen zutreffen.

## REVOKE - verlassene Rechte

Ein Recht  $R$  heißt *verlassen*, wenn das Recht, das für seine Zuteilung erforderlich war, zurückgezogen wurde und keine weitere Zuteilung von  $R$  vorgenommen wurde, deren erforderliche Rechte noch existieren.

- ▶ Die Option CASCADE veranlaßt zusätzlich zu der Rücknahme des in der REVOKE-Klausel benannten Rechts auch die Zurücknahme aller verlassenen Rechte.
- ▶ die Option RESTRICT führt zum Abbruch der REVOKE-Anweisung, wenn verlassene Rechte resultieren.

Benutzer Assistent teilt dem Benutzer Tutor ein INSERT-Recht für Land zu.

```
GRANT INSERT ON Land TO Tutor
```

Es folgen eine Reihe durch Benutzer Admin vorgenommene REVOKE-Anweisungen.

```
REVOKE INSERT ON Land FROM Tutor
```

Tutor behält das Recht, da er es unabhängig auch von Assistent erhielt.

```
REVOKE INSERT ON Land FROM Assistent CASCADE
```

Jetzt verliert sowohl Assistent, als auch Tutor das Recht.

Angenommen, Admin führt anstatt der letzten Anweisung die folgende Anweisung aus.

```
REVOKE GRANT OPTION FOR INSERT ON Land
FROM Assistent CASCADE
```

Jetzt behält Assistent das INSERT-Recht, jedoch Tutor verliert es, da die Erlaubnis für die Vergabe des Rechts an ihn zurückgezogen wurde.

# Kapitel 5: Konzeptueller Datenbankentwurf

- ▶ Der Entwurf des konzeptuellen Schemas ist Teil eines übergeordneten Softwareentwurfsprozesses.
- ▶ Im Pflichtenheft eines zu entwickelnden Anwendungssystems werden die Informationsbedürfnisse definiert.
- ▶ Häufige Grundlage ist das *Entity-Relationship-Modell (ERM)*,
- ▶ oder auch die *Unified Modeling Language (UML)*.

## Datenbankentwurf

### ► Konzeptueller Entwurf:

Finde eine umfassende Strukturierung der gesamten Informationsanforderungen der Miniwelt.

Resultat: *konzeptuelles Schema*.

### ► Logischer Entwurf:

Bilde die Zusammenhänge des konzeptuellen Schemas in Relationsschemata ab.  
Resultat: *logisches Schema*.

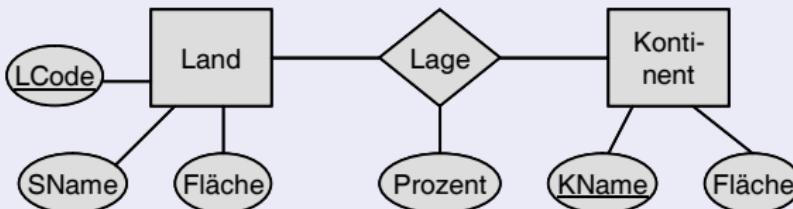
Das logische Schema ist Teil des zum Einsatz kommenden (relationalen) Datenbanksystems.

### ► Physischer Entwurf:

Definiere die Abspeicherung der Relationen auf den Speichermedien und lege Indexstrukturen zur Effizienzsteigerung der Anwendungen fest.

# 5.1 Entity-Relationship-Modell

## Entitäts- und Beziehungstypen



## Entitäts- und Beziehungsmengen

Land

| <u>LCode</u> | <u>LName</u> | Fläche |
|--------------|--------------|--------|
| Austria      | A            | 84     |
| Egypt        | ET           | 1001   |
| Germany      | D            | 357    |
| Turkey       | TR           | 779    |

Kontinent

| <u>KName</u> | Fläche |
|--------------|--------|
| Europe       | 3234   |
| Asia         | 44400  |
| Africa       | 30330  |

Lage

| Land         | Kontinent    |         |
|--------------|--------------|---------|
| <u>LCode</u> | <u>KName</u> | Prozent |
| A            | Europe       | 84      |
| ET           | Africa       | 90      |
| ET           | Asia         | 10      |
| D            | Europe       | 100     |
| TR           | Europe       | 32      |
| TR           | Asia         | 68      |

## Entitäts- und Beziehungstypen

- ▶ Entitätstypen ( $\approx$  Objekttypen)  $\leftrightarrow$  Rechtecke
- ▶ Beziehungstypen  $\leftrightarrow$  Rauten
- ▶ Attribute  $\leftrightarrow$  Ovale

- ▶ Attribute eines Entitätstyps, die einen Schlüssel ergeben, werden durch Unterstreichung gekennzeichnet.
- ▶ Beziehungstypen dürfen nur über Entitätstypen definiert sein.
- ▶ Der Schlüssel eines Beziehungstyps ist implizit durch die Schlüssel der an ihm beteiligten Entitätstypen gegeben.
- ▶ Auch Beziehungstypen können Attribute besitzen.

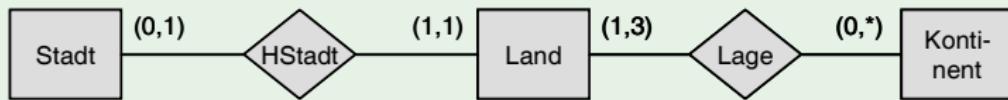
## Schema und Instanz

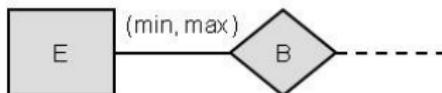
- ▶ Ein aus Entitäts- und Beziehungstypen gebildetes Schema nennen wir *ER-Schema*.
- ▶ Ein konkreter Zustand der betrachteten Miniwelt wird durch eine Menge von Entitäten und Beziehungen gemäß den Strukturen des Schemas repräsentiert.
- ▶ Zu jedem Typ existiert somit eine *Entitäts-* bzw. *Beziehungsmenge*, in denen die jeweiligen Entitäten und Beziehungen zusammengefasst sind; wir nennen dies eine *Instanz* des ER-Schemas.
- ▶ In jeder Instanz müssen die Entitäts- und Beziehungsmengen die folgenden Bedingungen erfüllen:
  - (1) Je zwei Entitäten, die gleiche Schlüsselwerte besitzen, sind auch in den übrigen Attributen gleich. Die Werte der einzelnen Schlüsselattribute müssen verschieden von null sein. (*Schlüsselbedingung für Entitätsmengen*)
  - (2) Je zwei Beziehungen, in denen die Werte der Schlüssel der an ihnen beteiligten Entitäten gleich sind, sind auch in den übrigen Attributen gleich. (*Schlüsselbedingung für Beziehungsmengen*)
  - (3) Jede, in einer Beziehung über ihren Schlüssel referenzierte Entität, existiert auch in der zugehörigen Entitätsmenge. (*referentielle Integrität, Fremdschlüsselbedingung*)

## Kardinalitäten: Beziehungskomplexitäten

- ▶ Sei  $E — B$  eine Kante, die einen Entitätstyp  $E$  mit einem Beziehungstyp  $B$  verbindet und mit einer Beziehungskomplexität  $(min, max)$ ,  $min \leq max$ , beschriftet ist.
- ▶ Seien  $e, b$  eine Entitäts- und Beziehungsmenge einer Instanz.
- ▶ Jede Entität  $\mu \in e$  ist dann in mindestens  $min$  und maximal  $max$  Beziehungen  $\nu \in b$  involviert.
- ▶ Wählen wir für  $max$  anstelle einer Zahl  $*$ , so steht dies für *beliebig viele*.

### Beispiel





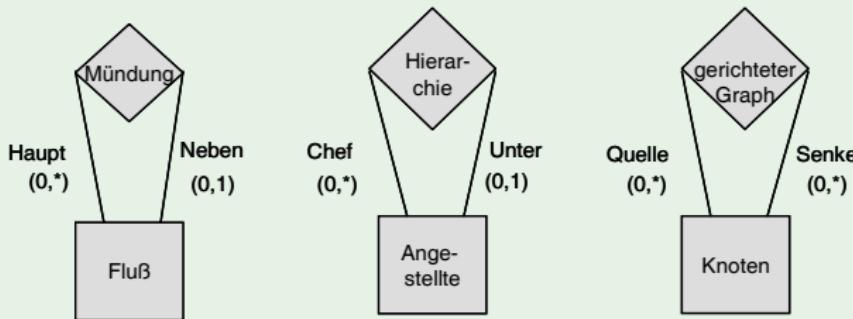
## partielle Beziehungen und Schlüsselbedingungen

- ▶  $\min = 0$ : es sind Instanzen zulässig, in denen Entitäten  $\mu \in e$  auftreten können, die in keiner Beziehung  $\nu \in b$  involviert sind.  
*(partielle Beziehung)*
- ▶  $\min = 0, \max = 1$ : in jeder Instanz darf jede Entität  $\mu \in e$  nur in höchstens einer Beziehung  $\nu \in b$  involviert sein.  
*(partielle Schlüsselbedingung)*
- ▶  $\min = 1, \max = 1$ : in jeder Instanz muss jede Entität  $\mu \in e$  in genau einer Beziehung  $\nu \in b$  involviert sein.  
*(totale Schlüsselbedingung)*

## rekursive Beziehungstypen

- ▶ Ist derselbe Entitätstyp mehrmals in demselben Beziehungstyp involviert, so reden wir von einem *rekursiven* Beziehungstyp.
- ▶ Es wird eine Unterscheidung von unterschiedlichen *Rollen* des Entitätstyps erforderlich.

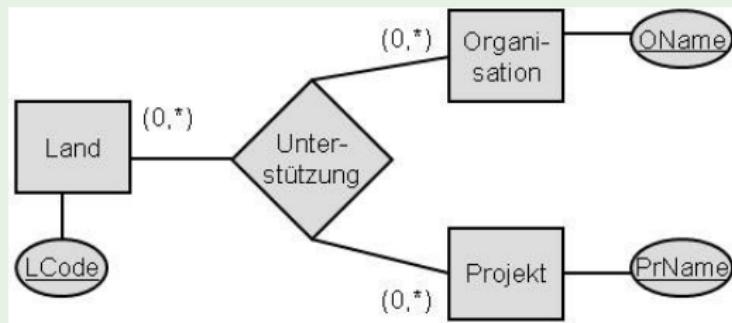
### Beispiele



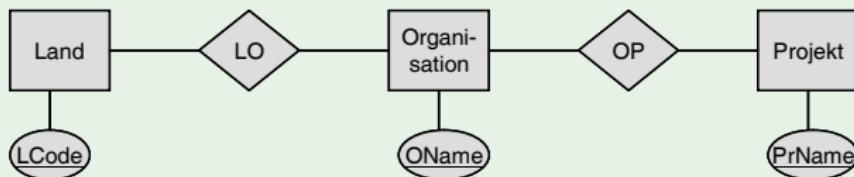
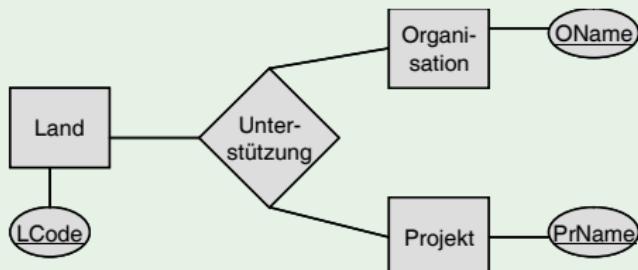
## mehrstellige Beziehungstypen

- ▶ Bisher waren Beziehungstypen binär.
- ▶ Ein Beziehungstyp kann im Allgemeinen über beliebig vielen Entitätstypen definiert sein.

### Beispiel



Beziehungstyp über drei Entitätstypen und eine mögliche Zerlegung des ternären Beziehungstyps in zwei binäre Beziehungstypen.



Ist die Zerlegung unabhängig von der Wahl der Beziehungskomplexitäten?

## Beziehungsmengen zum Beispiel

### Unterstützung

| Land         | Organisation | Projekt       |
|--------------|--------------|---------------|
| <u>LCode</u> | <u>OName</u> | <u>PrName</u> |
| EgoLand      | Invest       | Economy       |
| AltruLand    | Invest       | Economy       |
| AltruLand    | Invest       | Education     |

⇓

### LO

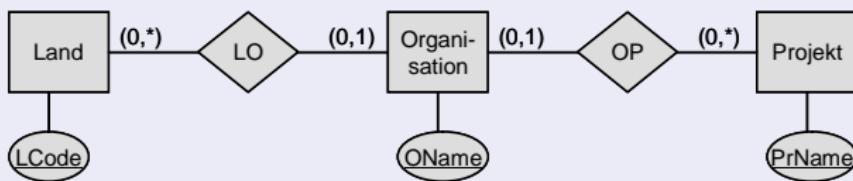
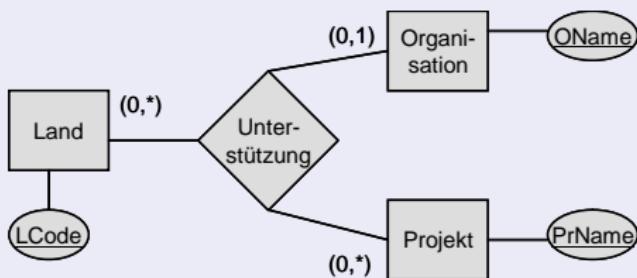
| Land         | Organisation |
|--------------|--------------|
| <u>LCode</u> | <u>OName</u> |
| EgoLand      | Invest       |
| AltruLand    | Invest       |

### OP

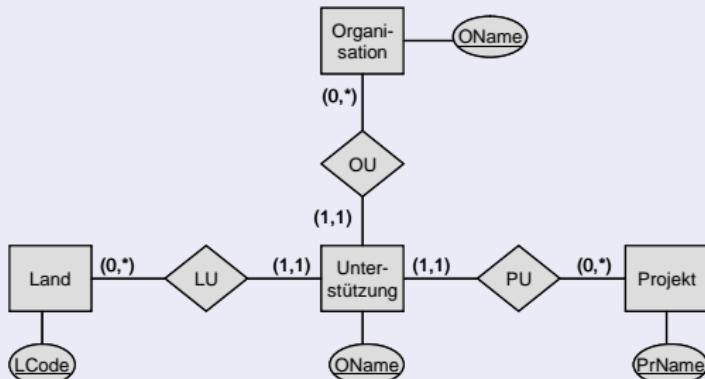
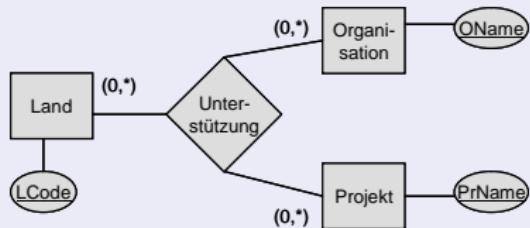
| Organisation | Projekt       |
|--------------|---------------|
| <u>OName</u> | <u>PrName</u> |
| Invest       | Economy       |
| Invest       | Education     |

Zerlegung unzulässig!

Zerlegung eines ternären in zwei binäre Beziehungstypen ist zulässig, falls mindestens eine beteiligte Beziehungskomplexität die Form  $(0/1,1)$  hat.



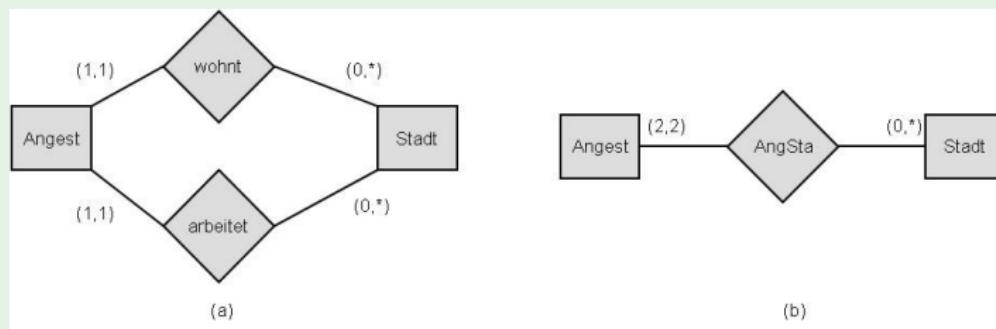
Zerlegung in binäre Beziehungstypen immer möglich unter Zuhilfenahme eines zusätzlichen Entitätstyps, der den mehrstelligen Beziehungstyp ersetzt.



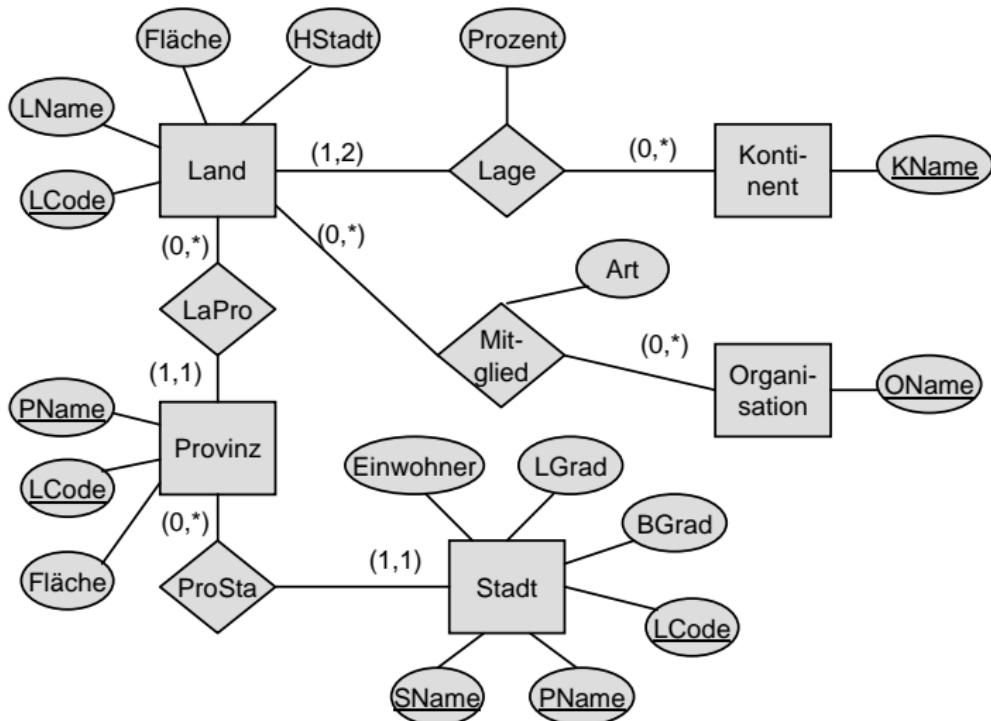
## Beziehungstypen über gleichen Entitätstypen

- Mehrere Beziehungstypen sind zwischen denselben Entitätstypen möglich.
- (a) und (b) repräsentieren im Allgemeinen unterschiedliche Miniwelten.

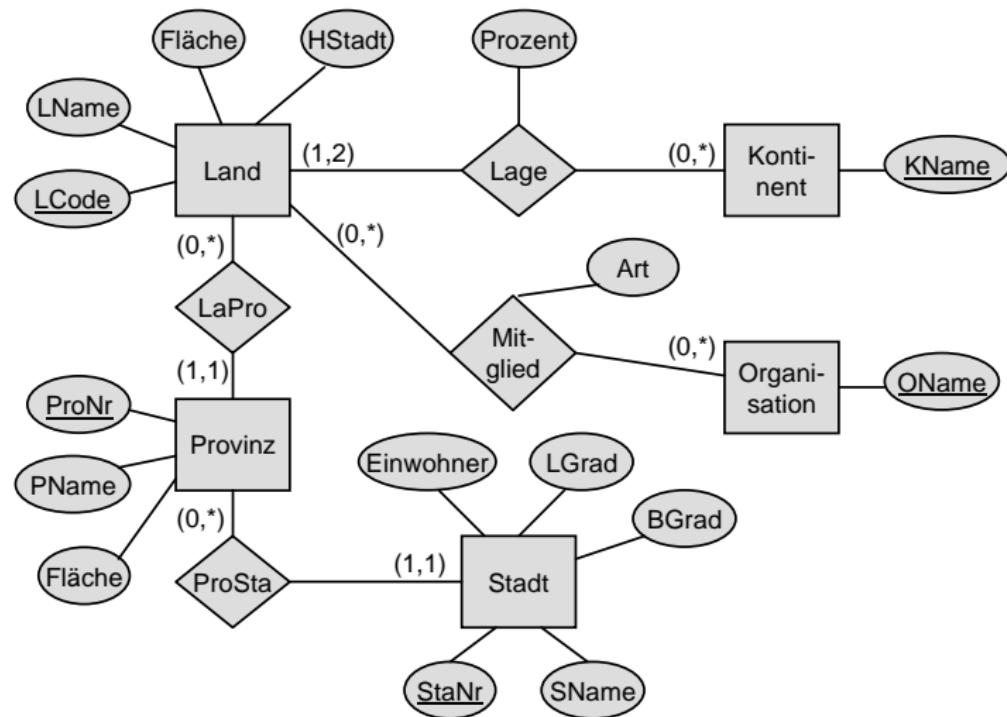
### Beispiel



## Mondial ER-Schema



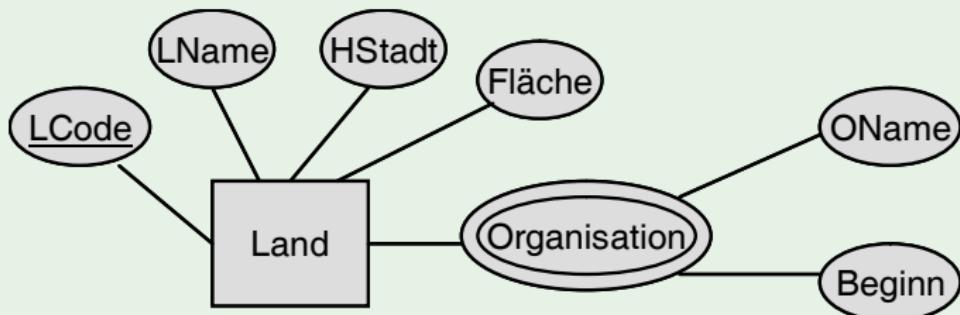
## Alternatives Mondial ER-Schema



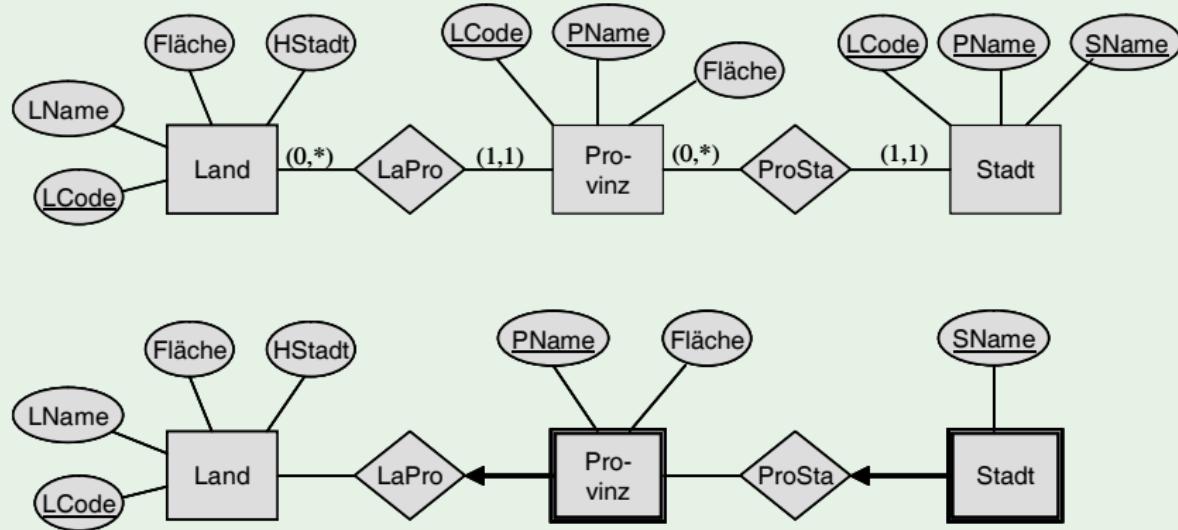
## 5.2 ER Erweiterungen

- ▶ Die Erfahrungen beim Einsatz des Entity-Relationship-Modells haben gezeigt, dass die Modellierung mittels Entitäts- und Beziehungstypen in gewissen Situation zu unbefriedigenden Resultaten führen kann.
- ▶ Mengenwertige und strukturierte Attribute
- ▶ Schwache Entitätstypen
- ▶ Generalisierung
- ▶ Aggregation

## Mengenwertige und strukturierte Attribute

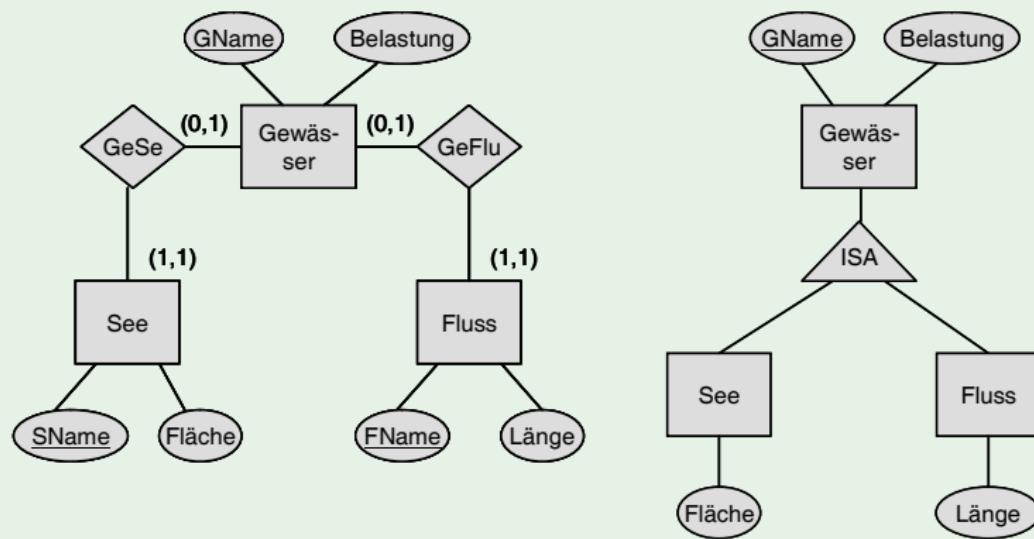


## Schwache Entitätstypen



Vererbung des Primärschlüssels an die angeschlossenen schwachen Entitätstypen.

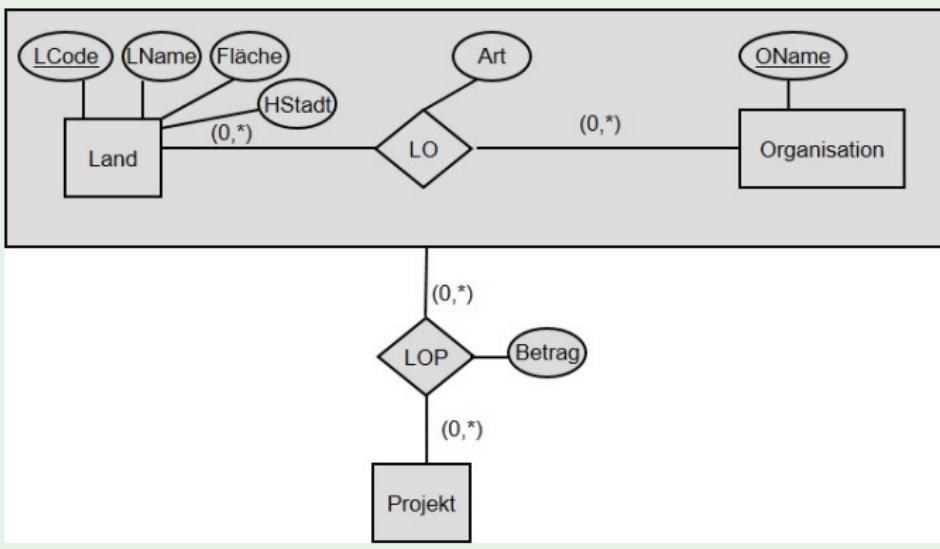
## Generalisierung



*ISA (Is A) Beziehungstyp:*

Jede Entität eines Unter(Sub)-Typs ist auch Entität des Ober(Super)-Typs.

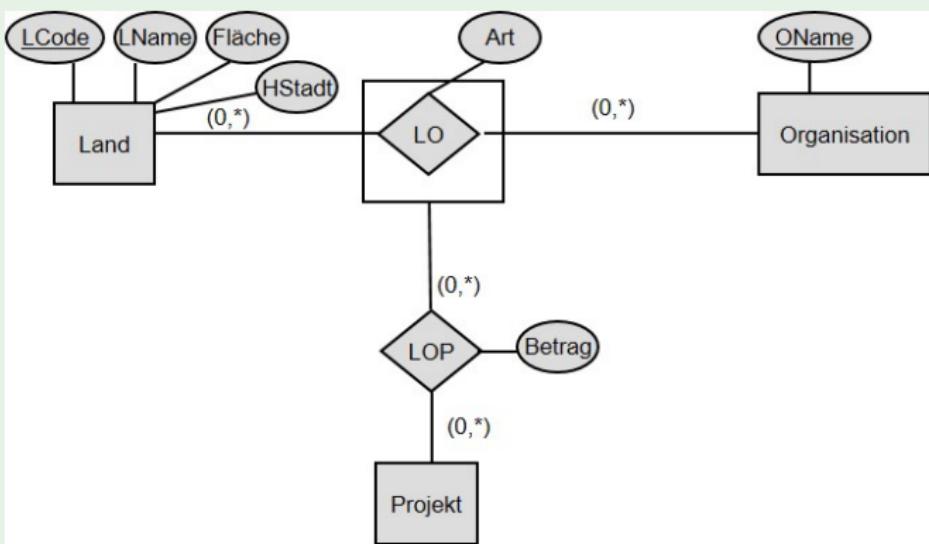
## Aggregation



Beziehungstypen werden als ein *aggregierter* Entitätstyp betrachtet:

⇒ Möglichkeit mehrstellige Beziehungstypen zu repräsentieren, die nicht immer für alle beteiligten Entitätstypen definiert sind.

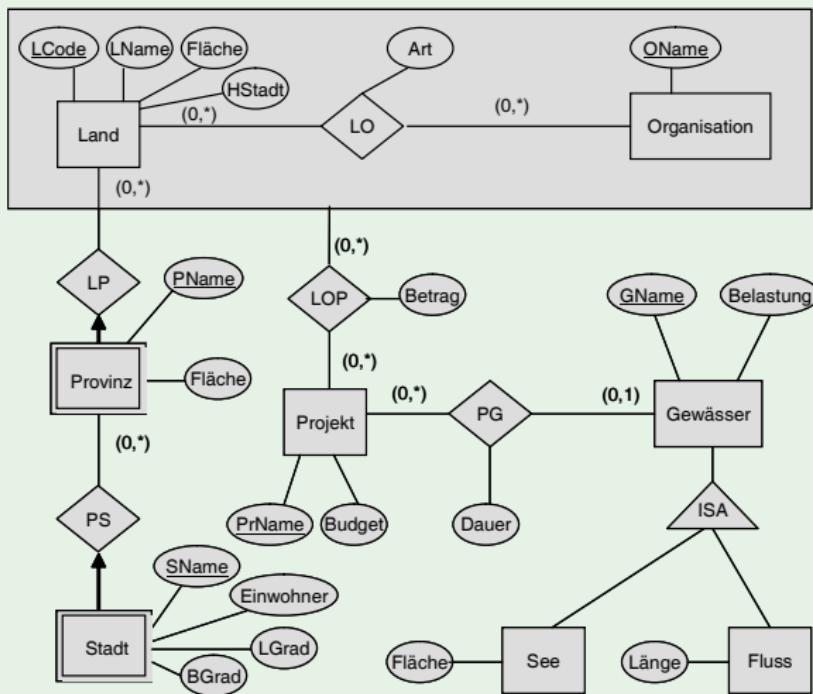
## Aggregation (alternative graphische Repräsentation)



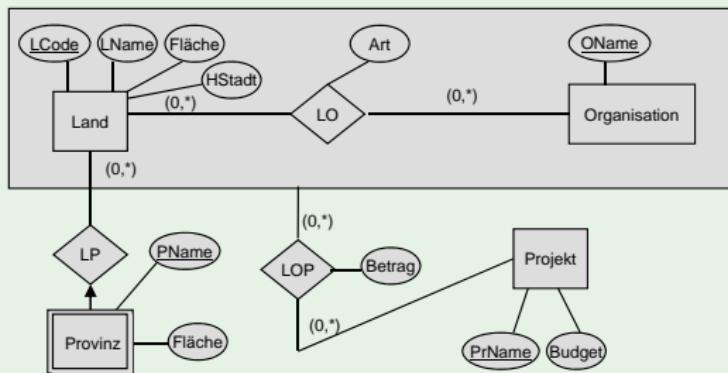
## 5.3 Transformation in Relationsschemata

- ▶ Wir ordnen zunächst jedem Entitätstyp und jedem Beziehungstyp ein Relationsschema zu.
- ▶ Unter Umständen können mehrere Relationsschemata wieder zu einem Relationsschema zusammengefasst werden, um die Anzahl Schemata nicht unnötig groß werden zu lassen. Beispiele sind:
  - ▶ Das Relationsschema des Beziehungstyps eines schwachen Entitätstypen kann in dessen Relationsschema integriert werden.
  - ▶ Ist ein Beziehungstyp mit einem Entitätstyp über eine Kardinalität mit  $\max = 1$  verbunden, dann kann das Relationsschema des Beziehungstyps in das Schema des Entitätstyps integriert werden.
- ▶ Ein so erhaltenes logisches Schema ist als Ausgangspunkt für eine weitere Verfeinerung zu verstehen, in die dann auch weiteres Wissen über die geplanten Anwendungen der Miniwelt eingehen kann, das im ER-Schema nicht repräsentiert ist.

## Mondial ER-Schema



## Transformation - Teil 1



**Land**(LCode, HStadt, LName, Fläche)

**Provinz**(PName, LCode, Fläche)

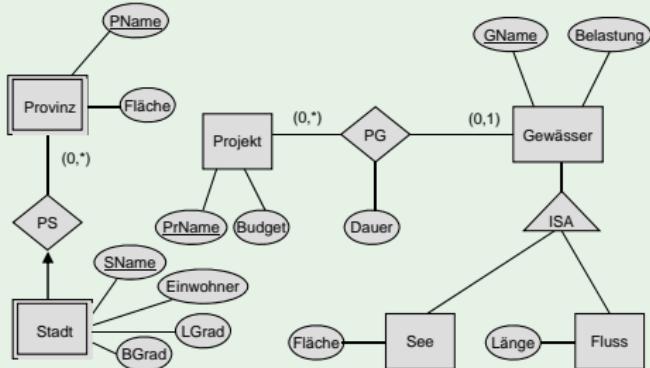
**Organisation**(OName)

**Projekt**(PrName, Budget)

**LO**(LCode, OName, Art)

**LOP**(LCode, OName, PrName, Betrag)

## Transformation - Teil 2



Provinz(PName, LCode, Fläche)

Stadt(SName, LCode, PName, Einwohner, LGrad, BGrad)

Projekt(PrName, Budget)

Gewässer(GName, Belastung)

See(GName, Fläche)

Fluss(GName, Länge)

PG(GName, PrName, Dauer)

## Mondial relational

Land(LCode, HStadt, LName, Fläche)

Provinz(PName, LCode, Fläche)

Stadt(SName, LCode, PName, Einwohner, LGrad, BGrad)

Organisation(OName)

Projekt(PrName, Budget)

Gewässer(GName, Belastung, PrName, Dauer)

See(GName, Fläche)

Fluss(GName, Länge)

LO(LCode, OName, Art)

LOP(LCode, OName, PrName, Betrag)

### Bemerkung:

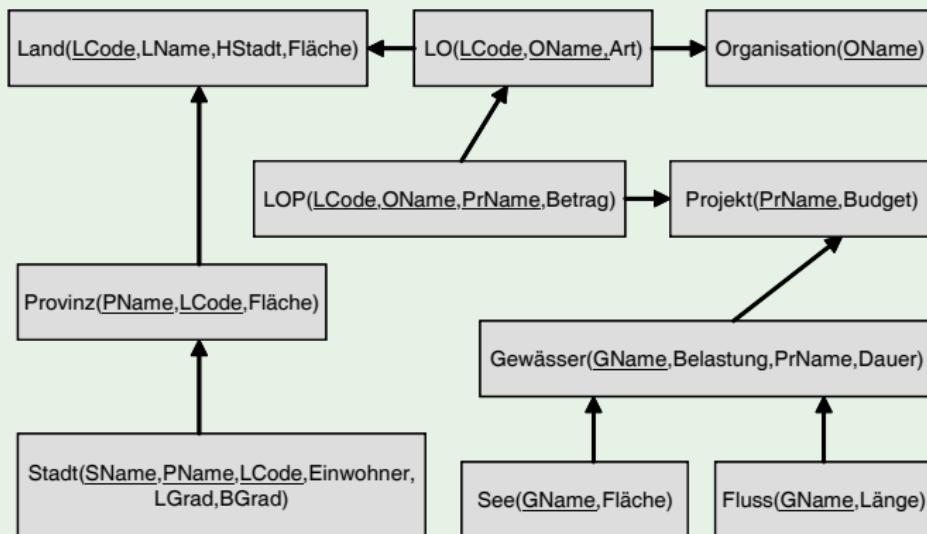
Beziehungstyp PG ist in Entitätstyp Gewässer integriert

⇒ Nullwerte für PrName und Dauer sind jetzt möglich!

# Relationsgraph

Fremdschlüsselbeziehungen unter den relationalen Schemata kennzeichnen wir graphisch durch gerichtete Kanten (*child* → *parent*).

## Mondial Relationsgraph



# empfohlene Lektüre

## The Entity-Relationship Model—Toward a Unified View of Data

PETER PIN-SHAN CHEN

Massachusetts Institute of Technology

---

A data model, called the entity-relationship model, is proposed. This model incorporates some of the important semantic information about the real world. A special diagrammatic technique is introduced as a tool for database design. An example of database design and description using the model and the diagrammatic technique is given. Some implications for data integrity, information retrieval, and data manipulation are discussed.

The entity-relationship model can be used as a basis for unification of different views of data: the network model, the relational model, and the entity set model. Semantic ambiguities in these models are analyzed. Possible ways to derive their views of data from the entity-relationship model are presented.

Key Words and Phrases: database design, logical view of data, semantics of data, data models, entity-relationship model, relational model, Data Base Task Group, network model, entity set model, data definition and manipulation, data integrity and consistency

CR Categories: 3.50, 3.70, 4.33, 4.34

---

1

---

<sup>1</sup> In: ACM Transactions on Database Systems (ToDS), Volume 1 Issue 1, March 1976.

# Kapitel 6: Formaler Datenbankentwurf

- ▶ Die Schwierigkeiten der konzeptuellen Modellierung sind zu einem großen Teil dadurch begründet, dass sich die relevanten Strukturen einer Miniwelt erst in Diskussionen mit den Anwendern oder durch Analyse von Dokumenten erfassen lassen.
- ▶ Mit der Transformation eines konzeptuellen Schemas in ein relationales Schema, dem logischen Entwurf, ändert sich diese Situation.
- ▶ Die Konzepte des relationalen Datenmodells werden dahingehend erweitert, dass sich die Güte eines logischen Entwurfs formal überprüfen lässt.
- ▶ Grundlage hierfür sind sogenannte *funktionale Abhängigkeiten*.

# 6.1 Funktionale Abhangigkeiten

## Definition

- ▶ Sei ein Relationsschema gegeben durch sein Format  $V$  und seien  $X, Y \subseteq V$ .
- ▶ Sei  $\text{Rel}(V)$  die Menge aller moglichen Relationen uber  $V$ .
- ▶ Sei  $r \in \text{Rel}(V)$ .  $r$  erfullt eine *funktionale Abhangigkeit (FA)*  $X \rightarrow Y$ , wenn fur alle  $\mu, \nu \in r$  gilt:

$$\mu[X] = \nu[X] \Rightarrow \mu[Y] = \nu[Y].$$

Wir sagen auch die FA  $X \rightarrow Y$  gilt in  $r$ .

|     | A     | B     | C     |
|-----|-------|-------|-------|
| r = | $a_1$ | $b_1$ | $c_1$ |
|     | $a_2$ | $b_2$ | $c_2$ |
|     | $a_2$ | $b_2$ | $c_3$ |
|     | $a_3$ | $b_2$ | $c_4$ |

Welche FA werden durch  $r$  erfüllt?

| FA                        | gilt in $r$ |
|---------------------------|-------------|
| $A \rightarrow B$         | 1           |
| $C \rightarrow A$         | 1           |
| $C \rightarrow B$         | 1           |
| $AC \rightarrow B$        | 1           |
| $C \rightarrow AB$        | 1           |
| <hr/>                     |             |
| $A \rightarrow C$         | 0           |
| $B \rightarrow A$         | 0           |
| $B \rightarrow C$         | 0           |
| $AB \rightarrow C$        | 0           |
| $A \rightarrow BC$        | 0           |
| $B \rightarrow AC$        | 0           |
| <hr/>                     |             |
| $A \rightarrow \emptyset$ | 1           |
| $A \rightarrow A$         | 1           |
| $AB \rightarrow A$        | 1           |
| $ABC \rightarrow A$       | 1           |
| <hr/>                     |             |
| ⋮                         | ⋮           |
| <hr/>                     |             |
| $\emptyset \rightarrow A$ | 0           |
| <hr/>                     |             |
| ⋮                         | ⋮           |

## Sat( $V, \mathcal{F}$ )

- ▶ Sei  $\mathcal{F}$  eine Menge funktionaler Abhangigkeiten uber  $V$  und  $X, Y \subseteq V$ .
- ▶ Sei dann  $\text{Rel}(V)$  die Menge aller moglichen Relationen uber  $V$ .
- ▶ Die Menge aller Relationen  $r \in \text{Rel}(V)$ , die alle funktionalen Abhangigkeiten in  $\mathcal{F}$  erfullen, bezeichnen wir mit  $\text{Sat}(V, \mathcal{F})$ .

## Membership-Test

- ▶  $\mathcal{F}$  impliziert die funktionale Abhangigkeit  $X \rightarrow Y$ ,  $\mathcal{F} \models X \rightarrow Y$ , wenn jede Relation  $r \in \text{Sat}(V, \mathcal{F})$  auch  $X \rightarrow Y$  erfullt.
- ▶ Die Menge  $\mathcal{F}^+ = \{X \rightarrow Y \mid \mathcal{F} \models X \rightarrow Y\}$  nennen wir die *Hulle* von  $\mathcal{F}$ .
- ▶ Der Test  $X \rightarrow Y \in \mathcal{F}^+$  ist der *Membership-Test*.

## Armstrong-Axiome

Sei  $r \in \text{Sat}(V, \mathcal{F})$ .

(A1) **Reflexivitat:** Wenn  $Y \subseteq X \subseteq V$ , dann erfullt  $r$  die FA  $X \rightarrow Y$ .

(A2) **Augmentation:**

Wenn  $X \rightarrow Y \in \mathcal{F}, Z \subseteq V$ , dann erfullt  $r$  auch die FA  $XZ \rightarrow YZ$ .

(A3) **Transitivitat:**

Wenn  $X \rightarrow Y, Y \rightarrow Z \in \mathcal{F}$ , dann erfullt  $r$  auch die FA  $X \rightarrow Z$ .

(A1) erlaubt die Herleitung funktionaler Abhangigkeiten, ohne Bezug auf  $\mathcal{F}$ .  
Wir bezeichnen solche FA als *triviale funktionale Abhangigkeiten*.

## Korrektheit und Vollstandigkeit

- ▶ Die Armstrong-Axiome sind *korrekt* in dem Sinn, dass die mit ihnen herleitbaren funktionalen Abhangigkeiten in der Tat Elemente der Hulle  $\mathcal{F}^+$  sind.
- ▶ Die Armstrong-Axiome sind auch *vollstandig*, d.h. jede funktionale Abhangigkeit in  $\mathcal{F}^+$  kann auch mit ihnen hergeleitet werden.