Kapitel 3: Das relationale Modell

Einführung
Typsystem und Konsistenzbedingungen
relationale Algebra
Interaktive Anfragesprache SQL
Schema und Sichten
Sicherheit

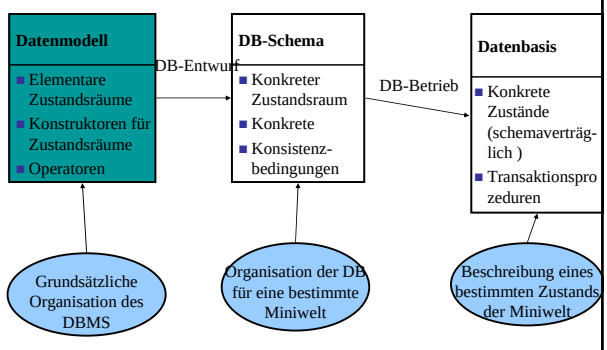
viele Folien: © Prof. Lockemann, IPD, Uni Karlsruhe

Kapitel 3: Das relationale Modell

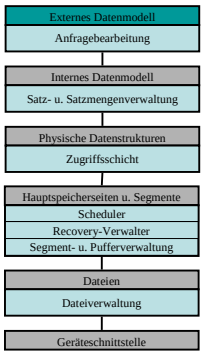
Einführung
Typsystem und Konsistenzbedingungen
relationale Algebra
Interaktive Anfragesprache SQL
Schema und Sichten
Sicherheit

viele Folien: © Prof. Lockemann, IPD, Uni Karlsruhe

Einordnung (1)



Einordnung (2)



Praxis der Datenmodelle

- Entwicklung eines Datenmodells verlangt Kompromisse:
  - Universelle Anwendbarkeit vs. Zuschnitt auf spezielle Modellierungsbedürfnisse
  - Ausdrucksmächtigkeit vs. Effizienz der Implementierung
- Heute eingesetzte Datenmodelle:
  - Hierarchisches und Netzwerk-Datenmodell (Altanwendungen)
  - Relationales Datenmodell (Sieg des Kompromisses)
  - Objektorientiertes Datenmodell (für anspruchsvollere Modellierungsbedürfnisse)
  - Objektrelationales Modell (Synthese der besten Eigenschaften)
  - XML (für Datenaustausch, Internetanwendungen)

Kapitel 3: Das relationale Modell

viele Folien: © Prof. Lockemann, IPD, Uni Karlsruhe

Polymorphe Typen

Atomare Typen

`char(*),int,numeric(p,q),float(p),date,time,timestamp`

Typkonstruktoren

`tupel` `::= [`*`sel`*`:`**`atomarerTyp`**`, ..., `*`sel`*`:`**`atomarerTyp`**`]`

`relation` `::= {`**`tupel`**`}`

Tupel: Sammlung als zusammengehörig betrachteter atomarer Datenelemente (**Tupelkomponente**). Die Zahl der Komponenten liegt für das Tupel fest, ihre Anordnung ist beliebig, da jede durch einen Selektor (**Attribut**) identifiziert wird. Der Komponentenwert entstammt einer für jedes Attribut vorgeschriebenen **Domäne**.

Relation: Menge im mathematischen Sinn aus gleichartigen Tupeln (Übereinstimmung in Attribut/Domäne-Paaren).

Relationen (anschaulich)

- **Relation** = endliche Menge von Tupeln gleichen Typs.
- Konsequenzen aus Definition: Relationen sind
  - duplikatfrei,
  - ungeordnet,
  - endlich,
  - in sogenannter **1. Normalform**, d.h., alle Attribute haben atomare Typen.
- Veranschaulichung in Tabellenform

Beispiel: Flugzeugtyp-Relation

ftypId	name	first	business	economy
744	Boeing 747-400	22	58	292
747	Boeing 747-200	23	36	326
D10	Boeing DC10	22	45	165
AB6	Airbus A300-600 Continental	10	54	137
319	Airbus_A319	10	42	63
737	Boeing 737	7	35	75
320	Airbus A320	10	62	63
314	Airbus A310-600 Continental	0	85	137
321	Airbus A321-100	10	45	116
SCC	British Airways Concorde	104	0	0
340	Airbus A340	10	48	170
380	Airbus A380-100	24	120	411

Polymorphe Konsistenzbedingungen (1)

- Bedingung 1: **Schlüsselbedingung**.
  key ::= **relation** × (sel^n → **atomarerTyp**^{n}) → **tupel** ($n \in \mathbb{N}$)
- Berücksichtigt häufige Gesetzmäßigkeit der Miniwelt: Objekte werden durch gewisse Attribute eindeutig charakterisiert, z.B. Flugzeugtypen durch ihren Code.
- Entsprechende Konsistenzbedingung: In der zugehörigen Relation dürfen keine zwei Tupel in all diesen Attributen übereinstimmen.
- Die Attribute dieser Kombination heißen **Schlüsselattribute** oder auch kurz der **Schlüssel** der Relation, die Wertekombination unter diesen Attributen in einem Tupel der **Schlüssel** dieses Tupels.

Polymorphe Konsistenzbedingungen (2)

- Bedingung 2: Erlaubt **Kopplung zweier Relationen**, indem man vorschreibt, dass die in einer Relation R_1 unter einer Attributfolge X_1 auftretenden Werte auch in einer Relation R_2 unter deren Attributfolge X_2 auftreten.
- Es besteht eine **referenzielle Konsistenz** von $R_1.X_1$ nach $R_2.X_2$. Ist X_2 Schlüssel von R_2, so nennt man X_1 **Fremdschlüssel** in R_1.
- Erläuterung: Objekte können sich aufeinander beziehen, z.B. Flüge auf den eingesetzten Flugzeugtyp.
- Somit: Tupel in einer Relation können Verweise auf Tupel einer anderen Relation enthalten, die nicht „ins Leere“ zeigen dürfen.

Relationale Datenbasisschemata (1)

Substitution der Variablen zu Tupel- und Relationstypen.

Für das Beispiel:

```
Typ  tupel Flugzeugtyp (
          ftypId:char(3), name:char(25),
          first:int, business:int,
          economy:int)
Typ  relation Flugzeugtyp (
          Flugzeugtyp key ftypId )
```

- Üblich: Zusammenfassung von Relationstyp und zugehörigem Tupeltyp zu einer einzigen Vereinbarung.
- Im Folgenden: Hervorhebung der Schlüssel durch Unterstreichung.

Laufendes Beispiel: DB für Flugbuchungen

Im Folgenden Betrachtung der Miniwelt „Flugplanung und Flugbuchungen“ mit Verwaltung von:

- Flugzeugtypen und ihren Sitzkapazitäten
- Flughäfen
- Flügen
- Kunden
- Passagieren (= gebuchte Kunden)
- Buchungen

Relation Flugzeugtyp

- Aufgabe:
  - Beschreibung der verfügbaren Flugzeugtypen.
- Typdefinition:
  - FLUGZEUGTYP (
    ftypId: CHAR(3), -- Eindeutiger Code, z.B. „737“
    name: CHAR(25), -- Name des Typs, z.B. „Boeing 737“
    first: INT, -- Anzahl Sitzplätze 1. Klasse
    business: INT, -- Anzahl Sitzplätze Business-Klasse
    economy: INT) -- Anzahl Sitzplätze Economy-Klasse

Ausprägung Flugzeugtyp

ftypId	name	first	business	economy
744	Boeing 747-400	22	58	292
747	Boeing 747-200	23	36	326
D10	Boeing DC10	22	45	165
AB6	Airbus A300-600 Continental	10	54	137
319	Airbus_A319	10	42	63
737	Boeing 737	7	35	75
320	Airbus A320	10	62	63
314	Airbus A310-600 Continental	0	85	137
321	Airbus A321-100	10	45	116
SCC	British Airways Concorde	104	0	0
340	Airbus A340	10	48	170
380	Airbus A380-100	24	120	411

Relation Flughafen

- Aufgabe:
  - Beschreibung der verfügbaren Flughäfen.
- Typdefinition:
  - FLUGHAFEN (
    flughCode: CHAR(3), -- Eindeutiger Code, z.B. „FRA“
    stadt: VARCHAR(25), -- Name der zugehörigen Stadt
    land: CHAR(3), -- Ländercode, z.B. „D“
    name: VARCHAR(25), -- Name des Flughafens
    zeitzone: INT) -- Differenz Lokalzeit-GMT in Std.

Ausprägung Flughafen

flughCode	stadt	land	name	zeitzone
FRA	Frankfurt	D	Frankfurt/Main International	1
MUC	Muenchen	D	Franz Josef Strauss	1
DUS	Duesseldorf	D	Rhein-Ruhr	1
JFK	New York	USA	John F Kennedy International	-5
LHR	London	GB	Heathrow	0
BOS	Boston	USA	Logan	-5
HNL	Honolulu	USA	Honolulu International	-11
SFO	San Francisco	USA	San Francisco International	-8
NRT	Tokyo	J	Narita	9
GIG	Rio de Janeiro	BRA	Rio de Janeiro Internacional	-3
SIN	Singapore	SIN	Changi International	-3
TXL	Berlin	D	Tegel	1
SYD	Sydney	AUS	Kingsford Smith International	10
VIE	Wien	A	Schwechat	1
CDG	Paris	F	Charles de Gaulle	1
FKB	Karlsruhe/Baden	D	Baden-Airport	1
EWR	New York	USA	Newark International	-5
ALY	Alexandria	EGY	Alexandria	2

Relation Flug (1)

- Aufgabe:
  - Beschreibung der verfügbaren Direktverbindungen.
- Typdefinition:
  - FLUG (
    flugNr: CHAR(6), -- Zugehörige Flugnummer
    von: CHAR(3), -- Code des Startflughafens
    nach: CHAR(3), -- Code des Zielflughafens
    ftypId: CHAR(3), -- Code des eingesetzten Flugzeugtyps
    wochentage: CHAR(7), -- Liste der Wochentage, an denen der Flug angeboten wird
    abflugszeit: TIME, -- Geplante Abflugszeit
    ankunftszeit: TIME, -- Geplante Ankunftszeit
    entfernung: INT) -- Flugdistanz in km

Relation Flug (2)

- FLUG (
  flugNr: CHAR(6),
  von: CHAR(3),
  nach: CHAR(3),
  ftypId: CHAR(3),
  wochentage: CHAR(7),
  abflugszeit: TIME,
  ankunftszeit: TIME,
  entfernung: INT)
- Fremdschlüsselbedingungen:
  - FLUG.von $\subseteq$ FLUGHAFEN.flughCode
  - FLUG.nach $\subseteq$ FLUGHAFEN.flughCode
  - FLUG.ftypId $\subseteq$ FLUGZEUGTYP.ftypId

Ausprägung Flug (1)

flugNr	von	nach	ftypId	wochentage	abflzt.	ankzt.	entfernung
LH400	FRA	JFK	747	MDMDFSS	1035	1245	6188
LH401	JFK	FRA	747	MDMDFSS	1630	0550	6188
LH676	FRA	ALY	319	-D--FSS	1330	1825	2741
LH677	ALY	FRA	319	M-M--SS	0745	1100	2741
LH458	MUC	SFO	744	MDMDFSS	1220	1445	9130
LH459	SFO	MUC	744	MDMDFSS	1355	0990	9130
LH408	DUS	EWR	340	MDMDFSS	1015	1230	6021
LH403	EWR	FRA	747	MDMDFSS	1745	0720	6188
LH208	FRA	DUS	321	MDMDFSS	0910	0955	182
LH4616	FRA	LHR	AB6	MDMDFSS	1330	1410	654
LH4513	LHR	FRA	737	MDMDFSS	2105	2340	654
LH2005	DUS	TXL	737	MDMDF--	1125	1230	474
LH2419	TXL	FRA	AB6	MDMDFSS	0730	0840	421
LH710	FRA	NRT	744	MDMDFSS	1350	0745	9360
LH711	NRT	FRA	744	MDMDFSS	1010	1450	9360
LH778	FRA	SIN	744	MDMDFSS	2215	1605	10264
.....							
.....							

Ausprägung Flug (2)

flugNr	von	nach	ftypId	wochentage	abflzt.	ankzt.	entfernung
.....							
LH6390	SIN	SYD	744	MDMDFSS	2015	0550	6298
LH6391	SYD	SIN	744	MDMDFSS	1550	2150	6298
LH779	SIN	FRA	744	MDMDFSS	2305	0545	10264
LH454	FRA	SFO	744	MDMDFSS	0955	1205	9130
LH6488	SFO	HNL	D10	MDMDFSS	1745	2204	3853
LH6489	HNL	SFO	D10	MDMDFSS	2125	0520	3853
LH191	MUC	FRA	AB6	MDMDFSS	1100	1205	304
LH3724	FRA	VIE	320	MDMDFSS	1705	1820	597
LH3651	VIE	FRA	320	MDMDFSS	0725	0855	597
LH4080	FRA	CDG	314	MDMDFSS	1710	1820	478
LH4171	CDG	MUC	314	MDMDFS-	0855	1025	684
LH500	FRA	GIG	340	-D-D-S-	2235	0530	9585
LH501	GIG	FRA	340	--M-F-S	1325	0550	9585
BA001	LHR	JFK	SSC	MDMDFSS	1030	0925	5564
BA002	JFK	LHR	SSC	MDMDFSS	1345	2225	5564

Relation Kunde

- Aufgabe:
  - Beschreibung von Kundenstammdaten (hier nur Name und Telefon).
- Typdefinition:
  - KUNDE (
    name: CHAR(25), -- Eindeutiger Name in Standardform
    telefon: CHAR(15)) -- Telefonnummer

Ausprägung Kunde

name	telefon
Muelle_Mrs_J	(0621) 144
Hillebrand_Mr_G	(089) 3118
Weinand_Mr_C	(0351) 166
Ateyeh_Mr_K	(07621) 98
Schmitt_Mrs_B	(0621) 978
Lockemann_Mr_P	(0721) 654
Christoffel_Mr_C	(06227) 54
Nimis_Mr_J	(0711) 386
Bender_Mr_P	(0721) 265
Lukacs_Mr_G	(0721) 843
Nagi_Mr_K	(069) 5791
Kuhn_Mrs_E	(030) 8642
Pulkowski_Mr_S	(0711) 955
Witte_Mr_R	(07231) 44
Krakowski_Mrs_P	(07243) 62
Posselt_Mr_D	(030) 2398
Gimbel_Mr_M	(089) 5656
Simpson_Mr_B	(0721) 117

Relation Ticket

- Aufgabe:
  - Zuordnung von Ticketnummern zu Kunden.
- Typdefinition:
  - TICKET (
    ticketNr: INT, -- Nummer des Tickets
    name: CHAR(25)) -- Name des Ticketinhabers in Standardform
- Fremdschlüsselbedingung:
  - TICKET.name $\subseteq$ KUNDE.name.

Ausprägung Ticket

ticketNr	name
7216083969	Muelle_Mrs_J
7216082080	Hillebrand_Mr_G
7216083911	Weinand_Mr_C
7216084728	Ateyeh_Mr_K
7216084066	Schmitt_Mrs_B
7216083968	Lockemann_Mr_P
7216084069	Christoffel_Mr_C
7216087337	Nimis_Mr_J
7216088131	Bender_Mr_P
7216088132	Lukacs_Mr_G
7216087336	Nagi_Mr_K
7216087338	Kuhn_Mrs_E
7216084065	Pulkowski_Mr_S
7216082757	Witte_Mr_R
7216084316	Krakowski_Mrs_P
7216084317	Posselt_Mr_D
7216083495	Gimbel_Mr_M
7216083971	Muelle_Mrs_J
7216083970	Bender_Mr_P
7216080815	Lockemann_Mr_P
7216080816	Simpson_Mr_B
7216080817	Weinand_Mr_C

Relation Buchung

- Aufgabe:
  - Beschreibung der vorhandenen Flugbuchungen.
- Typdefinition:
  - BUCHUNG (
    flugNr: CHAR(6), -- Gebuchter Flug
    ticketNr: INT, -- Nummer des Tickets
    platzCode: CHAR(3), -- Nummer des Sitzplatzes
    datum: DATE) -- Gebuchtes Datum
- Konsistenzbedingungen:
  - {flugNr,ticketNr} ist Schlüssel (ticketNr alleine ist nicht eindeutig, da ein Ticket mehrere Flüge umfassen kann).
  - Fremdschlüssel:
    - BUCHUNG.flugNr $\subseteq$ FLUG.flugNr.
      BUCHUNG.ticketNr $\subseteq$ TICKET.ticketNr.

Ausprägung Buchung

flugNr	ticketNr	platzCode	datum	flugNr	ticketNr	platzCode	datum
LH4616	7216080816	20A	01-AUG-00	LH711	7216083495	02B	26-AUG-00
LH4513	7216080816	05F	02-AUG-00	LH3724	7216084316	08A	29-SEP-00
LH4616	7216083968	05E	02-AUG-00	LH3651	7216084316	14F	03-OCT-00
BA001	7216083968	01D	03-AUG-00	LH408	7216088131	04D	04-SEP-00
BA002	7216083968	11A	05-AUG-00	LH403	7216088131	05D	08-SEP-00
LH4513	7216083968	23D	06-AUG-00	LH208	7216088131	07C	09-SEP-00
LH676	7216087336	12A	04-AUG-00	LH2419	7216083969	02E	01-SEP-00
LH677	7216087336	15F	23-AUG-00	LH4080	7216084728	10K	07-AUG-00
LH454	7216084066	04K	01-SEP-00	LH4171	7216084728	07A	11-AUG-00
LH6488	7216084066	03A	01-SEP-00	LH191	7216084728	01K	11-AUG-00
LH6489	7216084066	02A	22-SEP-00	LH208	7216084069	05D	01-AUG-00
LH459	7216084066	04A	23-SEP-00	LH3724	7216088132	07E	14-AUG-00
LH191	7216084066	01H	24-SEP-00	LH458	7216080815	81K	03-SEP-00
LH778	7216084065	33K	08-SEP-00	LH710	7216082757	34D	10-SEP-00
LH6390	7216084065	24G	09-SEP-00	LH400	7216084317	05G	21-JUL-00
LH6391	7216084065	54J	20-OCT-00	LH401	7216084317	05D	05-AUG-00
LH779	7216084065	56B	20-OCT-00	LH500	7216087338	19D	12-AUG-00
LH458	7216082080	81A	01-AUG-00	LH500	7216083970	19G	12-AUG-00
LH459	7216082080	84K	19-AUG-00	LH500	7216080817	19E	12-AUG-00
LH500	7216087337	01C	02-SEP-00	LH778	7216083911	83K	05-AUG-00
LH501	7216087337	02K	22-SEP-00	LH6390	7216083911	82A	06-AUG-00
LH710	7216083495	01K	11-AUG-00				

Relationale Datenbasisschemata (2)

- Wesentliche Elemente eines **relationalen Schemas**:
  - Relationstypen
  - Schlüssel- und Fremdschlüsselbedingungen
  - Weitere Konsistenzbedingungen (später)
  - Sichten-Definitionen (später)
  - Zugriffsrechte (später)
- **Relationale Datenbasis**:
  - Eine Ausprägung zu diesem Schema, also eine Menge von Wurzelobjekten (**Relationen**), deren Typen im Schema definiert sind.
  - Zu jedem im Schema vereinbarten Relationstyp gibt es genau eine **aktuelle Ausprägung**.

Konsistenz

- Polymorphe Operatoren garantieren Schemakonsistenz.
- Solange wir Transaktionen ignorieren, schließt das ein:
  - Relation R vom Typ T **erfüllt** Schlüsselbedingung **S**, wenn es keine zwei Tupel in R gibt, die für alle Attribute aus S dieselben Werte aufweisen.
  - Relationen R_1 vom Typ T_1 und R_2 vom Typ T_2 **erfüllen** Fremdschlüsselbedingung **F**, wenn es zu jedem Tupel in R_1 ein Tupel in R_2 gibt, so dass beide Tupel für alle Attribute aus F dieselben Werte aufweisen und diese Werte in R_2 Schlüssel sind.

Kapitel 3: Das relationale Modell

viele Folien: © Prof. Lockemann, IPD, Uni Karlsruhe

Polymorphe Operatoren

- Zunächst: Beschränkung auf Operatoren zum Abfragen der in Relationen enthaltenen Information.
- Forderung nach mathematischer Exaktheit durch Formalisierung. Kandidaten dafür ergeben sich aus Relationsbegriff der Mathematik.

  $\Rightarrow$ Operatoren bilden **relationale Algebra**.

Relationale Algebra

σ	:	**relation** $\times$ Θ $\rightarrow$ **relation**	(Selektion)
π	:	**relation** $\times$ attributfolge $\rightarrow$ **relation**	(Projektion)
$\times$	:	**relation** $\times$ **relation** $\rightarrow$ **relation**	(Kartesisches Produkt)
$\cup$	:	**relation** $\times$ **relation** $\rightarrow$ **relation**	(Vereinigung)
$/$	:	**relation** $\times$ **relation** $\rightarrow$ **relation**	(Differenz)
$\cap$	:	**relation** $\times$ **relation** $\rightarrow$ **relation**	(Durchschnitt)
$\bowtie_\theta$	:	**relation** $\times$ Θ $\times$ **relation** $\rightarrow$ **relation**	(Theta-Verbindung)
$\bowtie$	:	**relation** $\times$ **relation** $\rightarrow$ **relation**	(natürliche Verbindung)
$\div$	:	**relation** $\times$ **relation** $\rightarrow$ **relation**	(Division)

Θ: Universum logischer Bedingungen

- Rein formal betrachtet reichen die fünf Grundoperatoren *Selektion*, *Projektion*, *Kartesisches Produkt*, *Vereinigung* und *Differenz* aus.
- Operatoren *Durchschnitt*, *Theta-Verbindung*, *natürliche Verbindung* und *Division* vermeiden Prozeduren (höhere Performanz!).

Relationale Algebra

σ	:	**relation** $\times$ Θ $\rightarrow$ **relation**	(Selektion)
π	:	**relation** $\times$ attributfolge $\rightarrow$ **relation**	(Projektion)
$\times$	:	**relation** $\times$ **relation** $\rightarrow$ **relation**	(Kartesisches Produkt)
$\cup$	:	**relation** $\times$ **relation** $\rightarrow$ **relation**	(Vereinigung)
$/$	:	**relation** $\times$ **relation** $\rightarrow$ **relation**	(Differenz)
$\cap$	:	**relation** $\times$ **relation** $\rightarrow$ **relation**	(Durchschnitt)
$\bowtie_\Theta$	:	**relation** $\times$ Θ $\times$ **relation** $\rightarrow$ **relation**	(Theta-Verbindung)
$\bowtie$	:	**relation** $\times$ **relation** $\rightarrow$ **relation**	(natürliche Verbindung)
$\div$	:	**relation** $\times$ **relation** $\rightarrow$ **relation**	(Division)

- Nicht Teil der Algebra sind die Operatoren χ zum Erzeugen von Relationen und ρ zum Umbenennen von Spalten.

Selektion

- **Selektion** = Auswahl von *Zeilen* (Tupeln) aus einer Relation.
  - Auswahlkriterium ist vorgegebene Selektionsbedingung.
  - Ausgewählt werden Tupel, deren Attributwerte Bedingung Θ erfüllen:
    - Elementare Selektionsbedingung: Vergleich zweier Attributwerte oder eines Attributwerts und einer Konstante mittels $=, \neq, \leq, <, >, \geq$.
    - Zusammengesetzte Selektionsbedingung: disjunktive ($\vee$) oder konjunktive ($\wedge$) Verknüpfung elementarer Selektionsbedingungen.

Selektion: Beispiel 1

Alle in der Datenbasis enthaltenen deutschen Flughäfen:

$$\sigma_{land = "D"}(FLUGHAFEN)$$

flughCode	stadt	land	name	zeitzone
FRA	Frankfurt	D	Frankfurt/Main International	1
MUC	Muenchen	D	Franz Josef Strauss	1
DUS	Duesseldorf	D	Rhein-Ruhr	1
TXL	Berlin	D	Tegel	1
FKB	Karlsruhe/Baden	D	Baden-Airport	1

Selektion: Beispiel 2

Alle Flugzeugtypen mit mehr Sitzen in der First- als der Business-Klasse:

$$\sigma_{first > business}(FLUGZEUGTYP)$$

ftypId	name	first	business	economy
SCC	British Airways Concorde	104	0	0

Selektion: Beispiel 3

Alle von Frankfurt ausgehenden Fernflüge:

$$\sigma_{von = "FRA" \wedge entfernung > 6000}\ (FLUG)$$

oder auch

$$\sigma_{entfernung > 6000}\ (\sigma_{von = "FRA"}\ (FLUG))$$

Ergebnis:

flugNr	von	nach	ftypId	wochentage	abflzt.	ankzt.	entfernung
LH400	FRA	JFK	747	MDMDFSS	1035	1245	6188
LH710	FRA	NRT	744	MDMDFSS	1350	0745	9360
LH778	FRA	SIN	744	MDMDFSS	2215	1605	10264
LH454	FRA	SFO	744	MDMDFSS	0955	1205	9130
LH500	FRA	GIG	340	-D-D-S-	2235	0530	9585

Projektion

- **Projektion** = Auswahl von *Spalten* (Attributen) aus einer Relation.
  - Auswahlkriterium ist Liste von Attributnamen.
  - Von jedem Tupel der Original-Relation werden nur Werte für die ausgewählten Attribute beibehalten.
  - Entstehende Duplikate werden eliminiert ⇒ Tupelanzahl des Resultats kann geringer als in Ausgangsrelation sein.

Projektion: Beispiel 1

Flughäfen unter Verzicht auf Zeitzonen-Info:

$$\pi_{flughCode,stadt,land,name}\ (FLUGHAFEN)$$

flughCode	stadt	land	name
FRA	Frankfurt	D	Frankfurt/Main International
MUC	Muenchen	D	Franz Josef Strauss
DUS	Duesseldorf	D	Rhein-Ruhr
JFK	New York	USA	John F Kennedy International
LHR	London	GB	Heathrow
BOS	Boston	USA	Logan
HNL	Honolulu	USA	Honolulu International
SFO	San Francisco	USA	San Francisco International
NRT	Tokyo	J	Narita
GIG	Rio de Janeiro	BRA	Rio de Janeiro Internacional
SIN	Singapore	SIN	Changi International
TXL	Berlin	D	Tegel
SYD	Sydney	AUS	Kingsford Smith International
VIE	Wien	A	Schwechat
CDG	Paris	F	Charles de Gaulle
FKB	Karlsruhe/Baden	D	Baden-Airport
EWR	New York	USA	Newark International
ALY	Alexandria	EGY	Alexandria

Projektion: Beispiel 2

Nur die Länder der Flughäfen:

$$\pi_{land}\ (FLUGHAFEN)$$

land
D
USA
GB
J
BRA
SIN
AUS
A
F
EGY

Kartesisches Produkt

- **Kartesisches Produkt** = „Multiplikation“ von zwei Relationen.
  - Jedes Tupel der 1. Relation wird mit jedem Tupel der 2. Relation verkettet und in Ergebnis aufgenommen.
  - Attributmenge der Ergebnis-Relation ist disjunkte Vereinigung der Attribute der Ausgangsrelationen (gemeinsame Attribute sind umzubenennen).
  - Tupelzahl der Ergebnisrelation ist Produkt der Tupelzahlen der Ausgangsrelationen ⇒ eher theoretischer Operator; in der Praxis nur für kleine Relationen durchführbar.

Kartesisches Produkt: Beispiel

Relation BUCHUNG:

flugNr	ticketNr	platzCode	datum
LH4513	7216083968	23D	06-AUG-00
LH676	7216087336	12A	04-AUG-00
LH677	7216087336	15F	23-AUG-00
LH454	7216084066	04K	01-SEP-00
...			

Relation KUNDE:

ticketNr	name
7216084066	Schmitt_Mrs_B
7216083968	Lockemann_Mr_P
7216084069	Christoffel_Mr_C
7216087337	Nimis_Mr_J
...	

Kartesisches Produkt: Beispiel

Relation BUCHUNG × TICKET:

flugNr	B.ticketNr	platzCode	datum	T.ticketNr	name
LH4513	7216083968	23D	06-AUG-00	7216084066	Schmitt_Mrs_B
LH4513	7216083968	23D	06-AUG-00	7216083968	Lockemann_Mr_P
LH4513	7216083968	23D	06-AUG-00	7216084069	Christoffel_Mr_C
LH4513	7216083968	23D	06-AUG-00	7216087337	Nimis_Mr_J
LH676	7216087336	12A	04-AUG-00	7216084066	Schmitt_Mrs_B
LH676	7216087336	12A	04-AUG-00	7216083968	Lockemann_Mr_P
LH676	7216087336	12A	04-AUG-00	7216084069	Christoffel_Mr_C
LH676	7216087336	12A	04-AUG-00	7216087337	Nimis_Mr_J
LH677	7216087336	15F	23-AUG-00	7216084066	Schmitt_Mrs_B
LH677	7216087336	15F	23-AUG-00	7216083968	Lockemann_Mr_P
LH677	7216087336	15F	23-AUG-00	7216084069	Christoffel_Mr_C
LH677	7216087336	15F	23-AUG-00	7216087337	Nimis_Mr_J
LH454	7216084066	04K	01-SEP-00	7216084066	Schmitt_Mrs_B
LH454	7216084066	04K	01-SEP-00	7216083968	Lockemann_Mr_P
LH454	7216084066	04K	01-SEP-00	7216084069	Christoffel_Mr_C
LH454	7216084066	04K	01-SEP-00	7216087337	Nimis_Mr_J
...					

Kartesisches Produkt: Bewertung

- Daseinszweck: Verknüpfung von bisher getrennten Informationen.
- Verknüpfung erfolgt allerdings „blindlings“ ⇒ Resultat enthält viele sinnlose Kombinationen.
- I.d.R. anschließende Selektion der sinnvollen Kombinationen erforderlich.

- Hier z.B. durch Gleichheit der Ticket-Nummern gegeben:

$$\sigma_{\text{B.ticketNr} = \text{T.ticketNr}} (\text{BUCHUNG} \times \text{TICKET})$$

- Ergebnis:

flugNr	B.ticketNr	platzCode	Datum	T.ticketNr	name
LH4513	7216083968	23D	06-AUG-00	7216083968	Lockemann_Mr_P
LH454	7216084066	04K	01-SEP-00	7216084066	Schmitt_Mrs_B
...					

Theta-Verbindung

- Häufige Kombination von kart. Produkt und Selektion motiviert Definition des abgeleiteten Operators **Theta-Verbindung**:

$$R \bowtie_\theta S := \sigma_\theta (R \times S)$$

- Vorteil:
  - Einfachere Formulierung von Anfragen,
  - Implementierung kann umfangreiche Zwischenrelation durch unmittelbare Auswertung der Selektionsbedingung vermeiden.

Theta-Verbindung: Beispiel

Selektion der Buchungen für den 6. August 2000 mit den zugehörigen Passagieren:

$$\sigma_{\text{datum = 06-AUG-00} \wedge \text{B.ticketNr = T.ticketNr}} (\text{BUCHUNG} \times \text{TICKET})$$

Formulierung mit Theta-Verbindung:

$$\sigma_{\text{datum = 06-AUG-00}} (\text{BUCHUNG} \bowtie_{\text{B.ticketNr = T.ticketNr}} \text{TICKET})$$

Ergebnis:

flugNr	B.ticketNr	platzCode	datum	T.ticketNr	name
LH4513	7216083968	23D	06-AUG-00	7216083968	Lockemann_Mr_P
LH6390	7216083911	82A	06-AUG-00	7216083911	Weinand_Mr_C

Natürliche Verbindung

- Beobachtung:
  - Inhaltliche Zusammenhänge zwischen Tupeln häufig durch gleich benannte Attribute gegeben.
  - Daher nützlicher Spezialfall: Theta-Verbindung prüft auf Wertegleichheit unter gleich benannten Attributen (siehe Beispiel: BUCHUNG.ticketNr = TICKET.ticketNr).
  - Resultat einer solchen Theta-Verbindung enthält redundante Spalten.
- Konsequenz: **Natürliche Verbindung** (engl.: natural join) als abgeleiteter Operator $\bowtie$, der
  - Kartesisches Produkt bildet,
  - auf Wertegleichheit von namensgleichen Attributen selektiert,
  - redundante Spalten herausprojiziert.

Natürliche Verbindung: Beispiel 1

Betrachte wieder Selektion der Buchungen für den 6. August 2000 mit den zugehörigen Passagieren:

$$\sigma_{\text{datum = 06-AUG-00} \wedge \text{B.ticketNr = T.ticketNr}} (\text{BUCHUNG} \times \text{TICKET})$$

Formulierung mit natürlicher Verbindung:

$$\sigma_{\text{datum = 06-AUG-00}} (\text{BUCHUNG} \bowtie \text{TICKET})$$

Ergebnis:

flugNr	ticketNr	platzCode	datum	name
LH4513	7216083968	23D	06-AUG-00	Lockemann_Mr_P
LH6390	7216083911	82A	06-AUG-00	Weinand_Mr_C

Natürliche Verbindung: Formale Definition

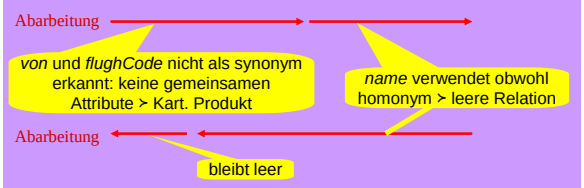
- Gegeben sei:
  - Relation R vom Typ $(A_1{:}t_1,...,A_n{:}t_n,B_1{:}u_1,...,B_m{:}u_m)$,
  - Relation S vom Typ $(B_1{:}u_1,...,B_m{:}u_m,C_1{:}v_1,...,C_l{:}v_l)$, wobei $m \geq 0$.
- Dann gilt:
  - Typ von $R \bowtie S$ ist $(A_1{:}t_1,...,A_n{:}t_n,B_1{:}u_1,...,B_m{:}u_m,C_1{:}v_1,...,C_l{:}v_l)$,
  - Elemente von $R \bowtie S$ sind alle Tupel t, für die Tupel $t_R \in R$, $t_S \in S$ existieren mit:
    - t und t_R stimmen in $A_1,...,A_n,B_1,...,B_m$ überein,
    - t und t_S stimmen in $B_1,...,B_m,C_1,...,C_l$ überein.
- Äquivalente Definition:
  - $R \bowtie S$ ist größte Relation über Attribute(R) $\cup$ Attribute(S), für die noch $\pi_{\text{Attribute}(R)}\ (R \bowtie S) \subseteq R$ und $\pi_{\text{Attribute}(S)}\ (R \bowtie S) \subseteq S$ gilt.

Natürliche Verbindung: Beispiel 2

Vorsicht bei natürlicher Verbindung!

Alle Flüge zusammen mit ausführlichen Angaben zu ihren Flugzeugtypen und Startflughäfen:

$$\text{FLUG} \bowtie \text{FLUGHAFEN} \bowtie \text{FLUGZEUGTYP}$$



Abhilfe durch Umbenennung :

$$\text{FLUG}_{\text{von}\rightarrow\text{flughCode}} \bowtie \text{FLUGHAFEN}_{\text{name}\rightarrow\text{fhname}} \bowtie \text{FLUGZEUGTYP}$$

Halbverbindung

- Effekt der natürlichen Verbindung $R \bowtie S$:
  - Kombination derjenigen Tupel, die „Partner“ in jeweils anderer Relation haben,
  - Elimination der „partnerlosen“ (engl.: dangling) Tupel.
- Manchmal interessiert nur Existenz, nicht aber Attributwerte, eines „Partners“.
- **Halbverbindung** $R \ltimes S$ selektiert Tupel aus R, die an natürlicher Verbindung teilnehmen würden:

$$R \ltimes S := \pi_{\text{Attribute}(R)}\ (R \bowtie S) = R \bowtie \pi_{\text{Attribute}(R)}\ (S)$$

Halbverbindung: Beispiel

Suche alle Buchungen des Passagiers Lockemann:

$$\pi_{\text{flugNr,ticketNr,platzCode,datum}}\ (\sigma_{\text{name = "Lockemann_Mr_P"}}\ (\text{BUCHUNG} \bowtie \text{TICKET}))$$

Formulierung mit Halbverbindung:

$$\text{BUCHUNG} \ltimes (\sigma_{\text{name = "Lockemann_Mr_P"}}\ (\text{TICKET}))$$

Ergebnis:

flugNr	ticketNr	platzCode	datum
LH4616	7216083968	05E	02-AUG-00
BA001	7216083968	01D	03-AUG-00
BA002	7216083968	11A	05-AUG-00
LH4513	7216083968	23D	06-AUG-00
LH458	7216080815	81K	03-SEP-00

Division

- **Division** = Umkehroperation zu kartesischem Produkt.
  - Sei T Relation vom Typ $(A_1{:}t_1,...,A_n{:}t_n,B_1{:}u_1,...,B_m{:}u_m)$.
  - Sei S Relation vom Typ $(B_1{:}u_1,...,B_m{:}u_m)$.
  - Dann ist $T \div S$ vom Typ $(A_1{:}t_1,...,A_n{:}t_n)$,
  - und der Inhalt von $T \div S$ ist die größte Relation R über diesem Typ, für die $R \times S \subseteq T$ gilt, die also innerhalb T ein kartesisches Produkt mit S darstellt.
- Äquivalente Definition:
  - $T \div S$ ist Teilmenge von $\pi_{A_1,...,A_n}(T)$,
  - ein Tupel t aus $\pi_{A_1,...,A_n}(T)$ ist dann in $T \div S$, wenn $\{t\} \times S \subseteq T$ gilt, wenn t also gepaart mit *allen* Tupeln aus S in T auftritt.

Division: Beispiel

Auf welchen Flughäfen starten *alle* Flugzeugtypen?

- Ermittle zunächst, welche Flugzeugtypen wo starten:

$$\text{TYP\_START} := \pi_{\text{von,ftypId}}(\text{FLUG})$$

- Ergebnis:

von	ftypId
FRA	747
JFK	747
FRA	319
ALY	319
MUC	744
SFO	744
DUS	340
EWR	747
FRA	321
FRA	AB6

von	ftypId
LHR	737
DUS	737
TXL	AB6
FRA	744
NRT	744
SIN	744
SYD	744
SFO	D10
HNL	D10
MUC	AB6

von	ftypId
FRA	320
VIE	320
FRA	314
CDG	314
FRA	340
GIG	340
JFK	SSC

Division: Beispiel

Auf welchen Flughäfen starten *alle* Flugzeugtypen?

- Ermittle dann, welche Flugzeugtypen es überhaupt gibt:

$$\text{TYPEN} := \pi_{\text{ftypId}}(\text{FLUGZEUGTYP})$$

- Ergebnis:

ftypId
744
747
D10
AB6
319
737
320
314
321
SCC
340
3XX

Division: Beispiel

Auf welchen Flughäfen starten *alle* Flugzeugtypen?

- Ermittle schließlich, welche Flughafencodes gepaart mit *allen* Tupeln aus TYPEN in TYP_START auftreten:

$$\pi_{\text{von,ftypId}}(\text{FLUG}) \div \pi_{\text{ftypId}}(\text{FLUGZEUGTYP})$$

- Dividend (TYP_START): Divisor (TYPEN):

von	ftypId
FRA	747
JFK	747
FRA	319
ALY	319
MUC	744
SFO	744
DUS	340
EWR	747
FRA	321
FRA	AB6

von	ftypId
LHR	737
DUS	737
TXL	AB6
FRA	744
NRT	744
SIN	744
SYD	744
SFO	D10
HNL	D10
MUC	AB6

von	ftypId
FRA	320
VIE	320
FRA	314
CDG	314
FRA	340
GIG	340
JFK	SSC

ftypId
744
747
D10
AB6
319
737
320
314
321
SCC
340
3XX

Division: Beispiel

Auf welchen Flughäfen starten *alle* Flugzeugtypen?

- E … aus T…

Ergebnis: leer, da im Dividenden kein Flughafencode gepaart mit allen Tupeln des Divisors auftritt.

```
von
---
```

- Dividend (TYP_START) gruppiert nach „von“: Divisor (TYPEN):

von	FtypId
ALY	319
CDG	314
DUS	340
DUS	737
EWR	747
FRA	314
FRA	319
FRA	320
FRA	321
FRA	340

von	FtypId
FRA	744
FRA	747
FRA	AB6
GIG	340
HNL	D10
JFK	747
JFK	SSC
LHR	737
MUC	744
MUC	AB6

von	FtypId
NRT	744
SFO	744
SFO	D10
SIN	744
SYD	744
TXL	AB6
VIE	320

FtypId
744
747
D10
AB6
319
737
320
314
321
SCC
340
3XX

Division als abgeleiteter Operator

- Fingerübung: Man zeige, dass

$$T \div S = \pi_A(T) \setminus \pi_A((\pi_A(T) \times S) \setminus T),$$

- wobei A := Attribute(T) – Attribute(S).

Vereinigung, Differenz, Durchschnitt

- Relationen sind Tupelmengen ⇒ klassische Mengenoperationen möglich.
- Datenmodell verlangt jedoch: Beteiligte Relationen müssen typgleich oder zumindest typkompatibel sein.

Mengenoperationen: Beispiel 1

Finde alle Flughäfen, auf denen Nachtflüge stattfinden:

$$\pi_{von}(\sigma_{abflugszeit > 2200 \vee abflugszeit < 0600}(FLUG)) \cup \pi_{nach}(\sigma_{ankunftszeit > 2200 \vee ankunftszeit < 0600}(FLUG))$$

```
1
---
FRA
SIN
SYD
HNL
SFO
GIG
LHR
```

Mengenoperationen: Beispiel 2

Finde alle Flughäfen, auf denen *keine* Nachtflüge stattfinden:

$$\pi_{flughCode}\,(FLUGHAFEN) \setminus (\pi_{von}\,(\sigma_{abflugszeit > 2200 \vee abflugszeit < 0600}\,(FLUG)) \cup \pi_{nach}\,(\sigma_{ankunftszeit > 2200 \vee ankunftszeit < 0600}\,(FLUG)))$$

```
1
---
MUC
DUS
JFK
BOS
NRT
TXL
VIE
CDG
FKB
EWR
ALY
```


Mengenoperationen: Beispiel 3

Finde alle Flughäfen, auf denen Nachtflüge sowohl starten als auch landen:

$$(\pi_{von}\,(\sigma_{abflugszeit > 2200 \vee abflugszeit < 0600}\,(FLUG)) \cap \pi_{nach}\,(\sigma_{ankunftszeit > 2200 \vee ankunftszeit < 0600}\,(FLUG)))$$

```
1
---
FRA
```


Mengenoperationen: Beispiel 4

- Einfügen eines neuen Flugs:

  FLUG = FLUG ∪
  {⟨flugNr: "LH6024",
  von: "FRA",
  nach: "JFK",
  ftypId: "747",
  wochentage: "MDMFSS",
  abflugszeit: 1000,
  ankunftszeit: 1200,
  entfernung: 6188⟩}
- Löschen analog via „\“.

Domänenerweiterung um NULL-Werte

- Häufiges Problem: Informationen über Miniwelt unvollständig.
  - Beispiel: Fehlende Kunden-Telefonnummer.
  - Fehlende Information sollte in Datenbasis nicht durch willkürliche Werte (z.B. Telefonnummer (999) 999-9999) abgebildet werden.
- Konsequenz:
  - Erweiterung aller Domänen um speziellen Wert **NULL**.
  - Bedeutung 1: Wert „unbekannt“. Beispiel: fehlende Telefonnummer.
  - Bedeutung 2: Wert „undefiniert“. Beispiel: Genau genommen gilt dies für die Business- und Economy-Plätze in der Concorde.
- Vorsicht: Die Semantik der Operatoren wird bei Berücksichtigung von NULL-Werten erheblich komplizierter.

Kapitel 3: Das relationale Modell

viele Folien: © Prof. Lockemann, IPD, Uni Karlsruhe

Deklarative Sprachen (1)

- Schwäche der Relationalen Algebra:
  - Für den ungeschulten Benutzer zu unhandlich.
- Mengenorientierung und damit Relationales Modell bildet aber natürliche Grundlage für **deklarative Sprachen**:
  - Deklarative Sprachen beschreiben lediglich gewünschtes Ziel (benutzernah).
  - Imperative Sprachen beschreiben Weg zum Ziel (implementierungsnah).
- An der Nutzerschnittstelle daher Ersatz der imperativen relationalen Algebra durch deklarative Sprache SQL.
- Ansatz: Der relationalen Algebra äquivalenter Kalkül auf Basis der Prädikatenlogik (**Tupelkalkül**).

Deklarative Sprachen (2)

Nutzungsarten:

- Ad-hoc-Anfragesprachen für interaktiven Dialog zwischen menschlichem Nutzer und DBMS.
- In herkömmliche Programmiersprache eingebettete DML für Weiterverarbeitung der Daten durch Anwendungsprogramme.
- Datenbankprogrammiersprache mit vollständiger Integration des Datenmodells einschließlich DDL in eine Programmiersprache.
- In jedem Fall Sprachkonstrukte für Transaktionsprozeduren erforderlich: Beginn und Abschluss der Ausführung, Abbruch mit Rückkehr zum Ausgangszustand.

Atomare Typen

- SQL/92-Standard sieht vor:
  - Zeichenfolge: CHARACTER(n), CHARACTER VARYING(n)
  - Bitfolge: BIT(n), BIT VARYING(n)
  - Ganzzahl: INTEGER
  - Festkommazahl: NUMERIC(p,q)
  - Gleitkommazahl: FLOAT(p)
  - Datum: DATE
  - Uhrzeit: TIME
  - Zeitpunkt: TIMESTAMP
- Oft weitere produktspezifische Typen (nicht portabel!)
- Neuer Standard (allmählich voll verfügbar): SQL/99
- Ganz neuer Standard: SQL 2003

Grundstruktur

- Grundstruktur eines SQL-Ausdrucks:
  select $A_1, A_2, ..., A_n$
  from $R_1, R_2, ..., R_m$
  where B
- Gedankliches Abarbeitungsmodell:
  - Bilde das kartesische Produkt der Relationen in der **from**-Klausel,
  - wähle hieraus die Tupel aus, die die Bedingung B erfüllen,
  - und projiziere das so erhaltene Ergebnis auf die in der **select**-Klausel aufgeführten Attribute.
- Äquivalenter Ausdruck der relationalen Algebra:

$$\pi_{A_1, A_2, ..., A_n} (\sigma_B (R_1 \times R_2 \times ... \times R_m)) .$$

Beispiele (1)

- Anzeige aller Flughäfen:
  select *
  from FLUGHAFEN
- entspricht (trivialem) relationenalgebraischem Ausdruck
  FLUGHAFEN .

Beispiele (2)

- Flughäfen unter Verzicht auf die Angabe der Zeitzone:
  select flughCode, stadt, land, name
  from FLUGHAFEN
- entspricht

$$\pi_{flughCode,stadt,land,name} (FLUGHAFEN) .$$

Beispiele (3)

- Weiter beschränkt auf die Flughäfen Deutschlands:
  select flughCode, stadt, land, name
  from FLUGHAFEN
  where land = "D"
- entspricht

$$\pi_{flughCode,stadt,land,name} (\sigma_{land = "D"} (FLUGHAFEN)) .$$

Beispiele (4)

- Alle Länder mit Flughäfen:

  select land
  from FLUGHAFEN

- Ergebnis:

```
land
----
D
D
D
USA
GB
USA
USA
USA
J
BRA
...
```

- mit Duplikaten!

Beispiele (5)

- Duplikat-Elimination durch Angabe von **distinct**:

  select **distinct** land
  from FLUGHAFEN

- entspricht

  π_{land} (FLUGHAFEN) .

- Ergebnis:

```
land
----
D
USA
GB
J
BRA
SIN
AUS
A
F
EGY
```


Beispiele (6)

- Alle von Frankfurt ausgehenden Fernflüge:

  select *
  from FLUG
  where von = "FRA"
  and entfernung > 6000

- entspricht

  $\sigma_{von = "FRA" \wedge entfernung > 6000}$ (FLUG) .

Beispiele (7)

- Selektion der Passagiere, die am 6. August 2000 fliegen:

  select name
  from TICKET, BUCHUNG
  where datum = 06-AUG-00
  and TICKET.ticketNr = BUCHUNG.ticketNr

- entspricht

  π_{name} ($\sigma_{datum = 06\text{-}AUG\text{-}00 \wedge TICKET.ticketNr = BUCHUNG.ticketNr}$ (BUCHUNG $\times$ TICKET))

- oder auch (interne Anfrageoptimierung!)

  π_{name} ($\sigma_{datum = 06\text{-}AUG\text{-}00}$ (BUCHUNG $\bowtie$ TICKET)) .

Beispiele (8)

- Beobachtung: Im letzten Beispiel hat BUCHUNG am Ergebnis nicht teil, sollte daher auch nicht in **from**-Klausel eingehen.
- Alternative Formulierung mit geschachtelter Anfrage

```
select  name
from    TICKET
where   ticketNr  in
        (select   ticketNr
         from     BUCHUNG
         where    datum = 06-AUG-00)
```

Gültigkeitsbereich eines Attributbezeichners: lokale select-Klausel

- entspricht relationaler Halbverbindung

$$\pi_{name}\,(\text{TICKET} \ltimes (\sigma_{datum = 06\text{-}AUG\text{-}00}\,(\text{BUCHUNG}))) .$$

Geschachtelte Anfragen (1)

- Abarbeitungsmodell für allgemeine geschachtelte Anfragen:
  - Bilde Kart. Produkt der Relationen in der äußeren **from**-Klausel.
  - Äußere Schleife: Für jedes Tupel *t* des Kart. Produkts:
    - Berechne Kart. Produkt der Relationen in der inneren **from**-Klausel.
    - Innere Schleife: Selektiere alle Tupel des inneren Kart. Produkts, die die innere **where**-Klausel erfüllen (diese kann Bezug auf *t* nehmen).
    - Projiziere Resultat auf Attribute der inneren **select**-Klausel.
    - Werte äußere **where**-Klausel mit *t* und dem Resultat der inneren Anfrage aus, bei positivem Ergebnis selektiere *t*.
  - Projiziere Resultat der äußeren Schleife auf Attribute der äußeren **select**-Klausel.

Geschachtelte Anfragen (2)

- Optimierer erstellt natürlich effizienteren Ablaufplan!
- Im Beispiel sofort ersichtlich: Kommt *t* in der inneren Schleife nicht vor,so muss die innere Schleife nur einmal berechnet werden!

Beispiele (9)

- Mehrfach geschachtelte Anfrage: Suche alle Passagiere, die Flüge von Frankfurt gebucht haben:

```
select  name
from    TICKET
where   ticketNr  in
        (select   ticketNr
         from     BUCHUNG
         where    flugNr   in
                  (select  flugNr
                   from    FLUG
                   where   von = "FRA")).
```

Beispiele (10)

- Anfrage mit Negation:

```
select   name
from     TICKET
where    ticketNr   not in
         (select    ticketNr
          from      BUCHUNG
          where     flugNr     in
                    (select    flugNr
                     from      FLUG
                     where     von = "FRA")).
```

- Liefert nicht etwa alle Passagiere, die *nicht* ab oder über Frankfurt fliegen, sondern Passagiere, die *mindestens ein* Ticket besitzen, dessen sämtliche Flüge Frankfurt nicht berühren.

Beispiele (11)

- Variante von **in**: Existenzprüfung mit **exists** bzw. **not exists**.
- Suche versehentliche Doppelbuchungen:

```
select   distinct flugNr, datu
from     BUCHUNG B1
where    exists
         (select   *
          from     BUCHUNG
          where    B1.flugNr = B2.flug
          and      B1.datum = B2.datum
          and      B1.platzCode = B2.platzCode
          and      B1.ticketNr <> B2.ticketNr).
```

Liegt im Gültigkeitsbereich der äußeren Schleife, daher zu qualifizieren

Liegt im Gültigkeitsbereich der inneren Schleife, daher Qualifizierung optional

- Anfrage ersetzt fehlende Konsistenzbedingung: Neben {flugNr,ticketNr} ist auch {flugNr,datum,platzCode} Schlüssel von BUCHUNG.

Beispiele (12)

- Konstrukte **θany** und **θall** (mit θ Vergleichsoperator):
- Es qualifiziert sich jedes Tupel, das gemäß Vergleichsoperator θ positiven Vergleich mit irgendeinem Wert bzw. allen Werten der berechneten Menge liefert.
- Beispiel: Suche weiteste Flugstrecke

```
select   von, nach, entfernung
from     FLUG
where    entfernung >=all
         (select    entfernung
          from      FLUG)
```

- Bemerkung: **in** ist Abkürzung von **=any**, **not in** Abkürzung von **<>all**.

Gruppierung, Aggregierung, Sortierung (1)

- Finde für jeden Flugzeugtyp die Zahl aller von Frankfurt abgehenden Flüge und sortiere Ergebnis nach Flugzeugtyp:

```
select    ftypId, count (flugNr)
from      FLUG
where     von = "FRA"
group by  ftypId
order by  ftypId
```

Gruppierung, Aggregierung, Sortierung (2)

- Abarbeitungsmodell bei Gruppierung:
  - Werte **from**- und **where**-Klauseln aus wie bisher (Bildung von Kart. Produkt und Selektion).
  - Fasse in der Ergebnisrelation alle Tupel, die gleiche Werte für die Gruppierungsattribute aufweisen, zu einer Gruppe zusammen.
  - Werte nun für jede einzelne Gruppe die **select**-Klausel aus. Dadurch entsteht für jede Gruppe genau ein Tupel.
  - Falls eine **having**-Klausel angegeben wurde, werte diese für jede Gruppe aus und eliminiere nicht-qualifizierende Gruppen.
  - Wenn **select distinct** angegeben wurde, eliminiere Duplikate aus der Menge der so entstandenen Tupel.
  - Sortiere ggf. die Ergebnistupel gemäß **order by**-Klausel.

Gruppierung, Aggregierung, Sortierung (3)

```
select      ftypId, count(flugNr)
from        FLUG
where       von = "FRA"
group by    ftypId
order by    ftypId
```

Auswertung der **from**- und **where**-Klauseln:

flugNr	von	nach	ftypId	wochentage	abflzt.	ankzt.	entfernung
LH400	FRA	JFK	747	MDMDFSS	1035	1245	6188
LH676	FRA	ALY	319	-D--FSS	1330	1825	2741
LH208	FRA	DUS	321	MDMDFSS	0910	0955	182
LH4616	FRA	LHR	AB6	MDMDFSS	1330	1410	654
LH710	FRA	NRT	744	MDMDFSS	1350	0745	9360
LH778	FRA	SIN	744	MDMDFSS	2215	1605	10264
LH454	FRA	SFO	744	MDMDFSS	0955	1205	9130
LH3724	FRA	VIE	320	MDMDFSS	1705	1820	597
LH4080	FRA	CDG	314	MDMDFSS	1710	1820	478
LH500	FRA	GIG	340	-D-D-S-	2235	0530	9585

Gruppierung, Aggregierung, Sortierung (3)

```
select      ftypId, count(flugNr)
from        FLUG
where       von = "FRA"
group by    ftypId
order by    ftypId
```

Gruppierung nach FtypId:

flugNr	von	nach	ftypId	wochentage	abflzt.	ankzt.	entfernung
LH400	FRA	JFK	747	MDMDFSS	1035	1245	6188
LH676	FRA	ALY	319	-D--FSS	1330	1825	2741
LH208	FRA	DUS	321	MDMDFSS	0910	0955	182
LH4616	FRA	LHR	AB6	MDMDFSS	1330	1410	654
LH710	FRA	NRT	744	MDMDFSS	1350	0745	9360
LH778	FRA	SIN	744	MDMDFSS	2215	1605	10264
LH454	FRA	SFO	744	MDMDFSS	0955	1205	9130
LH3724	FRA	VIE	320	MDMDFSS	1705	1820	597
LH4080	FRA	CDG	314	MDMDFSS	1710	1820	478
LH500	FRA	GIG	340	-D-D-S-	2235	0530	9585

Gruppierung, Aggregierung, Sortierung (3)

```
select      ftypId, count(flugNr)
from        FLUG
where       von = "FRA"
group by    ftypId
order by    ftypId
```

Projektion auf die Attribute der **select**-Klausel:

ftypId	flugNr
747	LH400
319	LH676
321	LH208
AB6	LH4616
744	LH710
744	LH778
744	LH454
320	LH3724
314	LH4080
340	LH500

Gruppierung, Aggregierung, Sortierung (3)

```
select    ftypId, count(flugNr)
from      FLUG
where     von = "FRA"
group by  ftypId
order by  ftypId
```

Auswertung der Funktion **count**(flugNr):

ftypId	count
747	1
319	1
321	1
AB6	1
744	3
320	1
314	1
340	1

Gruppierung, Aggregierung, Sortierung (3)

```
select    ftypId, count(flugNr)
from      FLUG
where     von = "FRA"
group by  ftypId
order by  ftypId
```

Auswertung der Funktion **count**(flugNr):

ftypId	count
747	1
319	1
321	1
AB6	1
744	3
320	1
314	1
340	1

Gruppierung, Aggregierung, Sortierung (3)

```
select    ftypId, count(flugNr)
from      FLUG
where     von = "FRA"
group by  ftypId
order by  ftypId
```

Sortierung gemäß **order by**-Klausel:

ftypId	count
314	1
319	1
320	1
321	1
340	1
744	3
747	1
AB6	1

Gruppierung, Aggregierung, Sortierung (4)

- Auswahl unter Gruppen:
- Beschränkung des vorigen Beispiels auf Flugzeugtypen mit mehr als einem Einsatz ab Frankfurt.

```
select    ftypId, count(flugNr)
from      FLUG
where     von = "FRA"
group by  ftypId
having    count(flugNr) > 1
order by  ftypId
```

- Ergebnis:

FtypId	Count
744	3

Gruppierung, Aggregierung, Sortierung (5)

- Weitere Aggregatfunktionen neben **count**:
  - Mittelwertbildung (**avg**),
  - Minimum (**min**),
  - Maximum (**max**),
  - Aufsummieren (**sum**).
- Anwendung stets auf die Tupel einer Gruppe.
- Argumente können Attributnamen oder auch arithmetische Ausdrücke sein, z.B.

```
select   max(first + business + economy)
from     FLUGZEUGTYP

max
---
555
```


Allgemeine Form der Sortierklausel

- Ergebnis eines Tabellenausdrucks kann für interaktive Präsentation oder programmgesteuertes Auslesen sortiert werden, indem hinter Ausdruck eine Klausel

  order by col_1 [**asc** | **desc**], ..., col_m [**asc** | **desc**]

  mit col_i Attributname oder Spaltennummer eingefügt wird.
- Schlüsselwort **asc** oder **desc** gibt aufsteigende (Default) bzw. absteigende Sortierfolge an.
- Beachte: Sortierung ist *Eigenschaft der Präsentation*, nicht der gespeicherten Tabellen.

Allgemeine Tabellenausdrücke (1)

- **select**-Anweisung ist wesentliches Konstrukt zur Bildung von Tabellen, aber es gibt weitere:
  - **union**, **intersect**, **except** dienen zur Bildung von Vereinigung, Durchschnitt und Differenz.
  - Beispiel: Suche nach Flügen, die noch keine Buchungen haben:

```
(select   flugNr from FLUG)
except
(select   flugNr from BUCHUNG)
```


Allgemeine Tabellenausdrücke (2)

- **select**-Anweisung ist wesentliches Konstrukt zur Bildung von Tabellen, aber es gibt weitere:
  - [**natural**] [**left** | **right** | **full**] [**outer**] **join** und **cross join** - Klauseln ermöglichen Bildung von natürlicher Verbindung, äußerer Verbindung und Kart. Produkt.
  - Beispiel: Reprise von „Suche Passagiere, die Buchungen für den 6. August 2000 haben“:

```
select   BT.name
from     (BUCHUNG natural join TICKET) as BT
where    BT.datum = 06-AUG-2000
```

Erweiterung: Einbezug von Tupeln, die sich ansonsten nicht qualifizieren

SQL/99: Allgemeine select-Syntax (1)

- Allgemeine Form der **select**-Anweisung:

  select [**distinct**] $item_1$ [[**as**] A_1], ..., $item_k$ [[**as**] A_k]
  from $table\text{-}expr_1$ [[**as**] R_1], ..., $table\text{-}expr_l$ [[**as**] R_l]
  [**where** *cond-expr*]
  [**group by** col_1, ..., col_m]
  [**having** *cond-expr*]

- Dabei hat jedes $item_i$ eine der Formen
  - e_i, wobei e_i ein Attributname oder ein arithmetischer Ausdruck ist,
  - $R_i.*$ (Auswahl aller Attribute von R_i),
  - f(e_i) mit f ∈ {**count**,**sum**,**avg**,**min**,**max**} und e_i wie oben,
  - f(**distinct** e) mit f und e wie oben,
  - **count**(*).
- „*“ statt *item*-Liste: Auswahl aller Spalten des durch **from** definierten Kreuzprodukts.

SQL/99: Allgemeine select-Syntax (1)

- Allgemeine Form der **select**-Anweisung:

  select [**distinct**] $item_1$ [[**as**] A_1], ..., $item_k$ [[**as**] A_k]
  from $table\text{-}expr_1$ [[**as**] R_1], ..., $table\text{-}expr_l$ [[**as**] R_l]
  [**where** *cond-expr*]
  [**group by** col_1, ..., col_m]
  [**having** *cond-expr*]

- Einschränkungen und Default-Regeln:
  - Weglassen der **where**-Klausel ist äquivalent zu „true“.
  - Bei Angabe einer **group by**-Klausel dürfen in der **select**- und **having**-Klausel Attribute, die von col_1,...,col_m verschieden sind, nur als Argumente von Aggregatfunktionen auftreten.
  - Bei Angabe von Aggregatfunktionen ohne **group by**-Klausel wird implizit eine **group by**-Klausel mit leerer Attributliste angenommen, d.h., die ganze Ergebnistabelle wird als eine Gruppe betrachtet.

Einfügen von Tupeln in Tabellen (1)

- Allgemeine Syntax zum Einfügen von Tupeln in eine Tabelle:

  insert **into** *tablename* [(A_1, ..., A_n)]
  table-expr
- Fehlt Attributliste A_1, ..., A_n, so müssen Anzahl und Domänen der Attribute von *table-expr* mit denen von *tablename* übereinstimmen.
- Ist Attributliste A_1, ..., A_n angegeben, so werden für die nicht aus *table-expr* gewonnenen Attribute Default-Werte vergeben.

Einfügen von Tupeln in Tabellen (2)

Beispiel für Einfügen eines einzelnen Tupels:
Neue Direktverbindung zwischen Frankfurt und New York:

```
insert   into FLUG
values   ("LH6204", "FRA", "JFK", "747", "MDMDFSS",
          1000, 1200, 6188)
```

Beispiel für Einbringen einer Menge:
Befüllen einer neuen Relation FRA_START mit allen von Frankfurt abgehenden Flügen.

```
insert   into FRA_START
(select  *
from     FLUG
where    von = "FRA")
```

Ändern von Tupeln

- Allgemeine Syntax zum Ändern von Tupeln in einer Tabelle:

  update *tablename*
  set $A_1 = e_1, ..., A_n = e_n$
  [**where** *cond-expr*]
- wobei $A_1, ..., A_n$ Attributnamen und $e_1, ..., e_n$ skalare Ausdrücke sind.
- Wird die **where**-Klausel nicht angegeben, werden alle Tupel in der Tabelle geändert.
- Beispiel: Ausmustern eines Flugzeugtyps und Ersatz durch einen neuen Typ:

  update FLUG
  set ftypId = "320"
  where ftypId = "737"

Löschen von Tupeln

- Allgemeine Syntax zum Löschen von Tupeln in einer Tabelle:

  delete **from** *tablename*
  [**where** *cond-expr*]
- Wird die **where**-Klausel nicht angegeben, werden alle Tupel in der Tabelle gelöscht (die Tabelle selbst bleibt jedoch erhalten).
- Beispiel: Löschen aller Buchungen vor dem 1.1.2000:

  delete **from** BUCHUNG
  where datum < 01-JAN-00

Problem und Lösung

- Interaktive Eingabe von SQL ist eher Ausnahme, i.d.R. erfolgt Datenbasis-Zugriff unter Programmkontrolle.
- Beispiel: WWW-Shop möchte HTML-Seiten mit Produktinformationen dynamisch aus Datenbasis generieren ⇒ Programmierung von CGI-Skripten, die ihrerseits SQL-Anfragen absetzen.
- Auch hier deskriptive Anfrage wünschenswert.
- Problem: Fehlanpassung („**impedance mismatch**“) zwischen Datenmodell des DBMS und der Anwendung, da Programmiersprachen i.d.R. einen Typ „Relation“ (allgemeiner: einen polymorphen Typ „Menge“) nicht anbieten.
- Konsequenz: Bearbeitung von Relationen muss in der Wirtssprache tupelweise („satzorientiert“) erfolgen. Daher Einführung von **Iteratoren**, die sukzessives Traversieren einer Relation erlauben.

Eingebettetes SQL: Vorgehen in Java

- Java Database Connectivity (JDBC) Standard definiert Java-Klassen, die Verbindungen zu Datenbanken, SQL-Anweisungen und deren Resultate kapseln.
- Typischer Code:

```
// drucke FlugNr fuer alle Verbindungen zwischen
// Flughaefen meinStart und meinZiel
import java.sql.*;
Connection conn =
   DriverManager.getConnection("jdbc:db2:FLUGDATENBASIS");
Statement stmt = Connection.createStatement();
ResultSet res =
   stmt.executeQuery("select FlugNr from DIREKTFLUG" +
                     " where von = \"" + meinStart + "\"" +
                     " and nach = \"" + meinZiel + "\"");
while (res.next()) {
   System.out.println(res.getString(1));
}
```

Kapitel 3: Das relationale Modell

viele Folien: © Prof. Lockemann, IPD, Uni Karlsruhe

Schemavereinbarung (1)

- In SQL sind DDL und DML integriert.
- Daher im Grundsatz Schema genauso dynamisch wie die Datenbasis selbst.

Schemavereinbarung (2)

- Vor dem Einfügen der ersten Tupel muss zunächst die entsprechende Relation vereinbart werden, z.B.

```
create  table FLUG (
                flugNr          CHAR(6),
                von             CHAR(3),
                nach            CHAR(3),
                ftypId          CHAR(3),
                wochentage      CHAR(7),
                abflugszeit     TIME,
                ankunftszeit    TIME,
                entfernung      INT)
```

- Weitere DDL-Konstrukte:
  - **alter table**: Erweitern um Attribute,
  - **drop table**: Beseitigen von Relationen.

Sichten (1)

- Oftmals ist es für Anwendungen einfacher, nicht mit dem Original-Datenbestand, sondern einem speziell zugeschnittenen Ausschnitt zu arbeiten.
- Beispiel: Dispatcher in Frankfurt greift häufig auf in Frankfurt startende Flüge zu, daher Anlegen einer separaten Relation FRA_START als Untermenge von FLUG sinnvoll.

Sichten (2)

- 1. Möglichkeit: Deklaration einer separaten Tabelle und Befüllen mit der gewünschten Information:

```
create   table FRA_START (
                flugNr          CHAR(6),
                von             CHAR(3),
                nach            CHAR(3),
                ftypId          CHAR(3),
                wochentage      CHAR(7),
                abflugszeit     TIME,
                ankunftszeit    TIME,
                entfernung      INT);
insert   into FRA_START
(select  *
from     FLUG
where    von = "FRA")
```


Sichten (3)

- Nachteile der Relation FRA_START :
  - Inhalt ist Schnappschuss des Datenbasiszustandes zum Zeitpunkt der Einfügeoperation.
  - Nachträgliche Veränderungen an FLUG spiegelt FRA_START nicht automatisch wider, daher regelmäßiges manuelles Aufdatieren erforderlich.
  - Redundante Datenhaltung verbraucht zusätzlichen Speicherplatz.

Sichten (4)

- Bessere Alternative: Vereinbarung von FRA_START als **Sicht** (Sichtrelation, engl.: view), deren Inhalt sich durch Berechnung aus den aktuell gespeicherten Relationen (sog. **Basisrelationen**) ermitteln lässt.

```
create   view FRA_START as
(select  *
from     FLUG
where    von = "FRA")
```


Sichten (5)

- Effekt einer Sichtvereinbarung:
  - DBMS registriert die Sichtdefinition, aber berechnet zugehörige Tupelmenge nicht sofort.
  - Der Name der Sicht kann in Anfragen wie eine Basisrelation verwendet werden.
  - Bei Zugriff auf die Sicht wird der Sichtname automatisch durch die Sichtdefinition ersetzt und die modifizierte Anfrage ausgewertet.
- Konsequenz: Sicht stellt **virtuelle** Relation dar, die bei Bedarf aus den **materialisierten** Basisrelationen erzeugt wird.

Sichten (6)

- Beispiel für Anfrageauswertung über Sichten:
  - Sichtvereinbarung:

```
create   view FRA_START as
(select  *
from     FLUG
where    von = "FRA")
```

  - Anfrage „Bestimme von Frankfurt ausgehende Fernflüge“:

```
select   flugNr
from     FRA_START
where    entfernung > 6000
```


Sichten (7)

- Sichten können durch beliebige SQL-Anfragen definiert werden.
- Beispiel: Sicht mit in Frankfurt eingesetzten Flugzeugtypen und ihrer Einsatzhäufigkeit:

```
create     view FRA_START_ZAHL (ftypId, anzahlFlüge) as
select     ftypId, count(flugNr)
from       FRA_START
group by   ftypId
```

- Bemerkungen:
  - Attributliste (ftypId, anzahlFlüge) gibt explizite Benennungen der Spalten vor, da für **count**(flugNr) ansonsten Defaultbezeichnung generiert wird.
  - Definition von Sichten auf Sichten (FRA_START) ist zulässig.

Sicht-Änderungen (1)

- Änderungen von Sichten sind i.d.R. problematisch, da sie in entsprechende Änderungen der Basisrelationen überführt werden müssen und dies nicht immer möglich ist.
- Beispiel: Änderung eines Flugzeugtyps in FRA_START:

```
update   FRA_START
set      ftypId = "320"
where    ftypId = "737"
```

- Zugehörige Änderung von FLUG:

```
update   FLUG
set      ftypId = "320"
where    ftypId = "737"
and      von = "FRA"
```

Unproblematisch!

Sicht-Änderungen (2)

- Weiteres Beispiel: Änderung eines Startflughafens:

```
update   FRA_START
set      von = "MUC"
where    flugNr = "LH4616"
```

- Zugehörige Änderung von FLUG:

```
update   FLUG
set      von = "MUC"
where    flugNr = "LH4616"
and      von = "FRA"
```

Sicht-Änderungen (3)

Inhalt von FLUG **vor** Ausführung von

update FRA_START
set von = "MUC"
where flugNr = "LH4616"

flugNr	von	nach	ftypId	wochentage	abflzt.	ankzt.	entfernung	
LH400	FRA	JFK	747	MDMDFSS	1035	1245	6188	
LH676	FRA	ALY	319	-D--FSS	1330	1825	2741	
LH208	FRA	DUS	321	MDMDFSS	0910	0955	182	
LH4616	FRA	LHR	AB6	MDMDFSS	1330	1410	654	
LH710	FRA	NRT	744	MDMDFSS	1350	0745	9360	
LH778	FRA	SIN	744	MDMDFSS	2215	1605	10264	} Sicht
LH454	FRA	SFO	744	MDMDFSS	0955	1205	9130	
LH3724	FRA	VIE	320	MDMDFSS	1705	1820	597	
LH4080	FRA	CDG	314	MDMDFSS	1710	1820	478	
LH500	FRA	GIG	340	-D-D-S-	2235	0530	9585	
LH401	**JFK**	**FRA**	**747**	**MDMDFSS**	**1630**	**0550**	**6188**	
LH677	**ALY**	**FRA**	**319**	**M-M--SS**	**0745**	**1100**	**2741**	
LH458	**MUC**	**SFO**	**744**	**MDMDFSS**	**1220**	**1445**	**9130**	
...								

Sicht-Änderungen (3)

Inhalt von FLUG **nach** Ausführung von

update FRA_START
set von = "MUC"
where flugNr = "LH4616"

Änderung propagiert korrekt nach FLUG, zeigt aber in der Sicht eine löschende Wirkung!

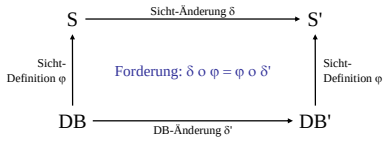
flugNr	von	nach	ftypId	wochentage	abflzt.	ankzt.	entfernung	
LH400	FRA	JFK	747	MDMDFSS	1035	1245	6188	
LH676	FRA	ALY	319	-D--FSS	1330	1825	2741	
LH208	FRA	DUS	321	MDMDFSS	0910	0955	182	
LH710	FRA	NRT	744	MDMDFSS	1350	0745	9360	
LH778	FRA	SIN	744	MDMDFSS	2215	1605	10264	} Sicht
LH454	FRA	SFO	744	MDMDFSS	0955	1205	9130	
LH3724	FRA	VIE	320	MDMDFSS	1705	1820	597	
LH4080	FRA	CDG	314	MDMDFSS	1710	1820	478	
LH500	FRA	GIG	340	-D-D-S-	2235	0530	9585	
LH4616	**MUC**	**LHR**	**AB6**	**MDMDFSS**	**1330**	**1410**	**654**	
LH401	**JFK**	**FRA**	**747**	**MDMDFSS**	**1630**	**0550**	**6188**	
LH677	**ALY**	**FRA**	**319**	**M-M--SS**	**0745**	**1100**	**2741**	
LH458	**MUC**	**SFO**	**744**	**MDMDFSS**	**1220**	**1445**	**9130**	
...								

Sicht-Änderungen (4)

- Weitere Beispiele für Änderungen:
  - **insert** **into** FRA_START
    values ("LH6204", "FRA", "JFK", "747", "MDMDFSS", 1000, 1200, 6188)
    (OK, wird in FLUG übernommen.)
  - **insert** **into** FRA_START
    values ("LH1694", "MUC", "TXL", "MDMDF--", "AB6", 1045, 1150, 421)
    (Fraglich, da Tupel in der Sicht keine Wirkung zeigt.)
  - **update** FRA_START_ZAHL
    set anzahlFlüge = anzahlFlüge + 1
    where ftypId = "744"
    (Unmöglich, da Einsatzhäufigkeit berechnete Größe ist.)

Sicht-Änderungen (5)

- Semantisches Zulässigkeitskriterium für Sicht-Änderung δ, die Sicht S in S' überführt: Ist DB die Datenbasis, aus der S berechnet wurde, so muss es eine (eindeutig bestimmbare) Datenbasis DB' geben, die zur Sicht S' führt.



Kann i.d.R. nicht vom DBMS überprüft werden, daher Erlass von strikten syntaktischen Einschränkungen.

Sicht-Änderungen (6)

- Beispielhafter Satz von Regeln für änderbare Sichten:
  1. Die Sicht muss durch eine einzelne **select**-Anweisung definiert sein (d.h., kein **union**, etc.).
  2. Die **select**-Klausel dieser Anweisung darf nur Attributnamen, und jeden nur einmal, enthalten (keine Aggregatfunktionen, berechnete oder konstante Ausdrücke) und darf nicht **distinct** spezifizieren.
  3. Die **from**-Klausel darf nur einen einzigen Relationsnamen enthalten; dieser muss eine Basisrelation oder eine änderbare Sicht bezeichnen.
  4. Falls die **where**-Klausel geschachtelte Anfragen beinhaltet, darf in deren **from**-Klauseln der Relationsname aus (3) nicht auftauchen.

Intuition: Die Sicht muss durch Auswahl von Zeilen oder Spalten aus einer Basisrelation oder änderbaren Sicht entstehen und die Auswahlbedingung „tupel-lokal“ sein.

Konsistenzbedingungen (1)

SQL gestattet es, die wichtigsten Konsistenzbedingungen als Bestandteil der Vereinbarung einer Relation zu formulieren:

```
create table FLUG (
    primary key      flugNr,                          ← Primärschlüssel
    unique           (von, nach, abflugszeit),        ← weiterer Schlüssel
    flugNr           CHAR(6),
    von              CHAR(3) not null                 ← NULL-Werte nicht zulässig
Fremdschlüsselbed. → foreign key references FLUGHAFEN (flughCode),
    nach             CHAR(3) not null
                     foreign key references FLUGHAFEN (flughCode),
    ftypId           CHAR(3)
                     foreign key references FLUGZEUGTYP (ftypId)
Kompensationsaktionen → on update cascade on delete set null,
    wochentage       CHAR(7) default "MDMDFSS",       ← Default-Wert
    abflugszeit      TIME not null,
    ankunftszeit     TIME not null,
    entfernung       INT check (entfernung >= 0))     ← Konsistenzbed.
```


Konsistenzbedingungen (2)

- Für Konsistenzbedingungen, die mehrere Relationen überdecken, gibt es das Konstrukt der **assertion**.
- Beispiel: Für jeden Flug darf Anzahl der verkauften Tickets nicht Kapazität des eingesetzten Flugzeugtyps überschreiten.

```
create assertion KEINE_ÜBERBUCHUNG
check (not exists
        (select *
         from   FLUG F, FLUGZEUGTYP T
         where F.ftypId = T.ftypId
         and    T.first + T.business + T.economy <any
                (select    count(ticketNr)
                 from      BUCHUNG B
                 where     B.flugNr = F.flugNr
                 group by datum)))
```


Kapitel 3: Das relationale Modell

viele Folien: © Prof. Lockemann, IPD, Uni Karlsruhe

Transaktionsprozeduren (1)

- In SQL gibt es ausschließlich Transaktionsprozeduren:
  - Jede einzelne SQL-Anweisung wird als Dienstprimitiv betrachtet und robust (resistent, isoliert, persistent) ausgeführt.
  - Folgen von SQL-Anweisungen können zu Transaktionsprozeduren zusammengefasst werden.
  - Im Prinzip werden drei Anweisungen benötigt:
    - begin_of_transaction (BOT)
    - end_of_transaction (EOT)
    - abort
  - SQL kennt jedoch nur Anweisungen **commit** (= end_of_transaction) und **rollback** (= abort). begin_of_transaction wird vor erster Anweisung und nach jedem **commit** oder **rollback** impliziert.

Transaktionsprozeduren (2)

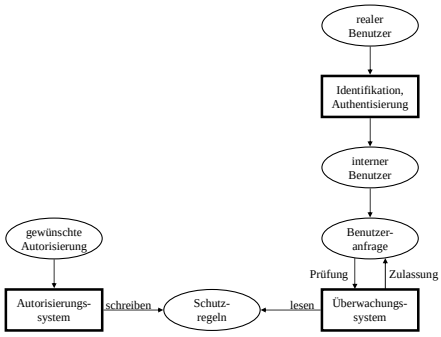
- Beispiel für Transaktionsprozedur: Simultanes Einfügen in BUCHUNG und TICKET wg. Fremdschlüsselbedingung:

```
insert  into BUCHUNG
values  ("LH4616", "7219654721", "01A", 11-NOV-00);
insert  into TICKET
values  ("7219654721", "Kramer_Mr_R");
commit;
```


Sicherheit durch technischen Schutz

- Aufgabenkomplexe des technischen Schutzes:
  - **Autorisierung** = Festlegung der Schutzregeln, d.h. Vergabe von Zugriffsrechten an Nutzer.
  - **Identifikation und Authentisierung** = Unverfälschbare Erkennung der Identität eines Nutzers bei Anmeldung.
  - **Zugriffsüberwachung** = Sicherstellung der Einhaltung der Schutzregeln während der Transaktionen.

Schutz: Systemarchitektur



Schutz durch Sichten

- Einfache Lösung für die Zugriffsüberwachung: Sichten.
- Voraussetzung: Jeder Nutzer kann auf die Datenbasis nur über Sichten zugreifen.
- Beispiel: Lufthansa-Disponent darf nur Flüge bearbeiten, deren Flugnummern mit „LH“ beginnen:

```
create   view LH_FLUG as
select   *
from     FLUG
where    flugNr like "LH%"
```

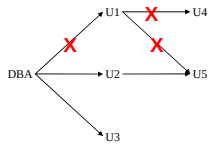
- Da LH_FLUG die Regeln für änderbare Sichten erfüllt, ist auch Schreibzugriff möglich.

Schutz durch Privilegien

- Differenziertere Zugriffsüberwachung durch Vergabe von Privilegien mittels **grant**-Anweisung bzw. Rücknahme mittels **revoke**-Anweisung.
- Beispiele:
  - **grant select on** FLUG **to public**
  - **grant update** (ftypId) **on** FLUG **to** Dispatcher
  - **grant all privileges on** FLUG **to** Systemverwalter
  - **grant delete on** BUCHUNG **to** Jahresabschluss
  - **grant select** (ticketNr, platzCode) **update** (platzCode) **on** BUCHUNG **to** Reisebüro, Flugsteig
- Ermächtigung zur Rechteweitergabe:
  - **grant update on** BUCHUNG **to** Filialleiter **with grant option**

Entzug von Privilegien

- Rechteentzug:
  revoke update on BUCHUNG **from** Filialleiter
- Kann kaskadieren!



Schutz: Schemaebene

- Regeln für Schemata:
  - Ein Schema hat einen Besitzer.
  - Er allein kann an ihm Vereinbarungen vornehmen.
  - Er besitzt zudem an allen Relationen dieses Schemas automatisch sämtliche Privilegien. Diese kann er weitergeben.