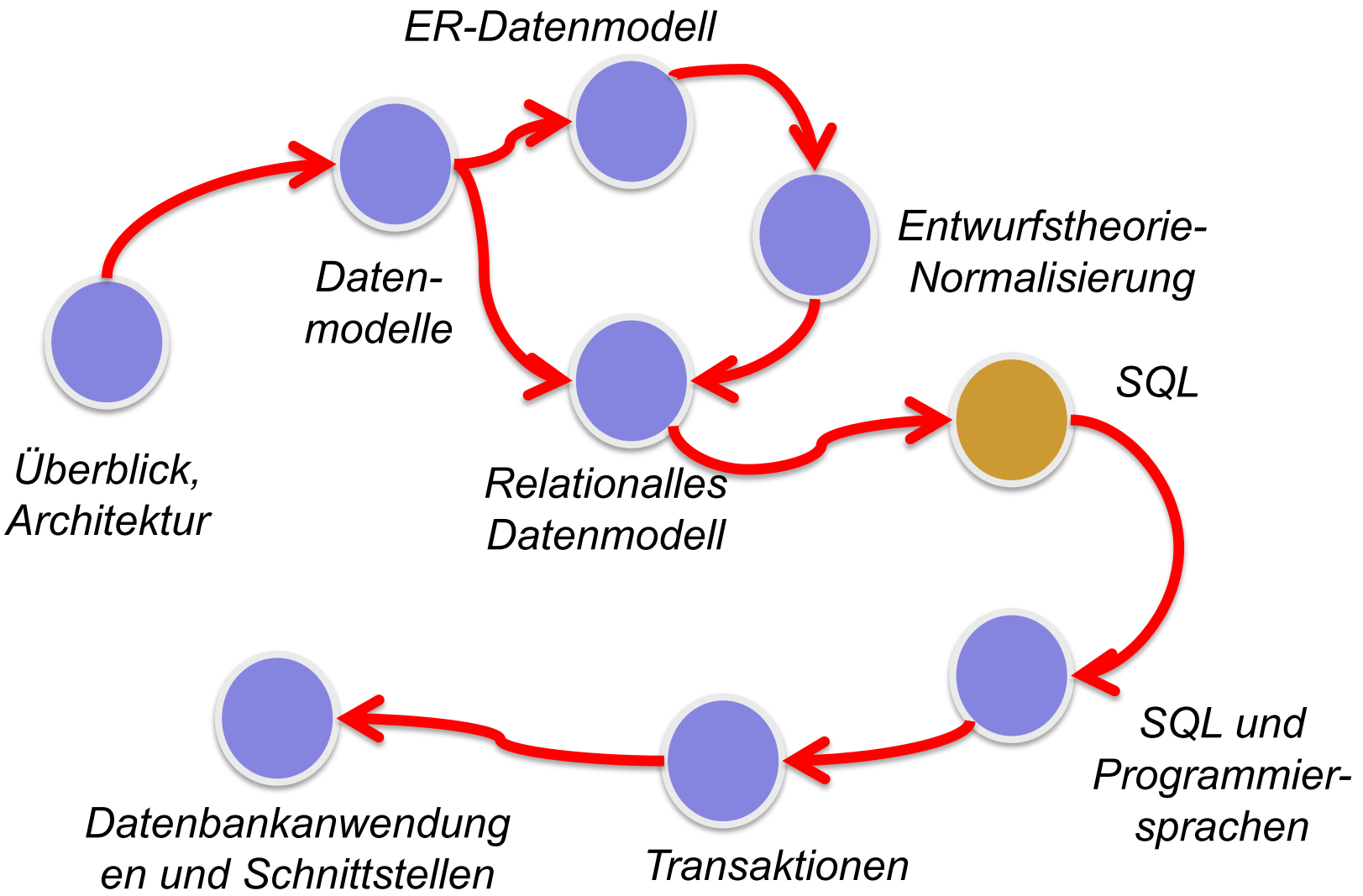
Fritz Laux
Ilia Petrov
Reutlingen
University

Kapitel 3.1: SQL

1. *Datenbanksprachen*
2. *SQL DDL (Data Definition) Data Dictionary*
3. *SQL DML (Manipulation)*

Fritz Laux
Ilia Petrov
Reutlingen
University

Struktur der Vorlesung



Fritz Laux
Ilia Petrov
Reutlingen
University

Datenbanksprachen

nicht so wichtig

Sprachtyp	*„eingebettet“*	*eigenständig*
4GL *Deskriptiv* *Objektorient.*	*SQL CLI (Call Level Interface)* *embedded **SQL*** *JDBC, JDO*	***SQL**,* *ODMG ODL,OQL*
3GL *Prozedural*	*ODBC, Codasyl*	*Relat. Algebra*
5GL *Graphisch*		*QBE (Access)*

Data Definition Language (DDL)

- Definition: Formale Sprache um die Artefakte einer Datenbank zu definieren und zu verwalten.
- Artefakte sind Datenbankobjekte, ihre Beziehungen, Integritätsbedingungen und Zugriffsberechtigungen

Data Manipulation Language (DML)

- Definition: Formale Sprache um die Daten einer Datenbank zu pflegen (einfügen, ändern, löschen).

Fritz Laux
Ilia Petrov
Reutlingen
University

Die Geschichte von SQL

- *1974 entwickelte D. Chamberlin (IBM San Jose Laboratory) die Sprache 'Structured English Query Language' (SEQUEL), welche später zu SQL umbenannt wurde*
  - IBM baute den Datenbankprototyp *System R mit* SQL als DB-Sprache (1977)
  - 1979 erschien die erste kommerzielle relationale Datenbank von ORACLE.
- *1987 publizierten ANSI und ISO den ersten SQL Standard, 1989, überarbeitet*
  - Basic DDL, DML
- *1992 erschien SQL/92, bekannt als SQL2*
  - Integritätsbedingungen, explizite Joins, Modulsprache
- *1999 wurde SQL:1999 veröffentlicht*
  - Objektrelationale Erweiterungen.
- *SQL:2003, SQL 2008*
  - Sequenzgenerator, Kollektionen und XML Erweiterung
  - Instead of, truncate
- *SQL:2011 Aktueller Standard*
- ***SQL ist das "Datenbank-Esperanto" (Chris Date)***
  - **Wird von nahezu allen Datenbanksystemen unterstützt**

Fritz Laux
Ilia Petrov
Reutlingen
University

Syntaxübersicht:

- *DDL* `Datendefinitionssprache`
  - CREATE SCHEMA | TABLE | INDEX | VIEW <s/t/i/v-name>;
- *DCL*
  - GRANT <privileges> ; | REVOKE <privileges> ;
- *DML* `Datenmanipulationssprache`
  - INSERT INTO | UPDATE | DELETE FROM <table_name>;
    SELECT <c-list> [INTO <params>] FROM <table_list>;
    COMMIT | ROLLBACK WORK;
  - embedded SQL:
    DECLARE <c-name> CURSOR FOR <SQL-query>;
    CLOSE | OPEN <cursor>;
    FETCH <c-name> INTO <params>;
- *ML* `Modulsprache`
  - MODUL <m> LANGUAGE <lang> SCHEMA <sch-name>
    DECLARE <...> CURSOR FOR
    PROCEDURE <proc-name> <params> <SQL-statement>;

Fritz Laux
Ilia Petrov
Reutlingen
University

SQL

- *SQL ist eine deklarative Datenbanksprache,*
  - keine vollständige Programmiersprache

- *SQL ist eine tabellenorientierte Sprache*

- *Der SQL-Standard besteht aus 3 Sprachgruppen:*
  - DDL zur Definition der Datenbank-Struktur.
  - DML zur Datenpflege und zum Datenretrieval.
    - ML (module language) zur Definition von SQL Routinen
  - DCL zur Rechtevergabe und Sicherheitseinstellungen

- *Die Syntax besteht aus „englischen Sätzen“:*

```
CREATE TABLE Staff(staffNo VARCHAR(5),
                   Name VARCHAR(15),
                   salary DECIMAL(7,2));

INSERT INTO Staff VALUES ('SG16', ‘G. Brown', 8300);

SELECT staffNo, Name, salary
      FROM Staff
      WHERE salary > 10000;
```

Fritz Laux
Ilia Petrov
Reutlingen
University

CREATE TABLE

Syntax: *(erweiterte BNF)*

CREATE TABLE TableName

({ **colName dataType** [NOT NULL] [UNIQUE]
[DEFAULT value] [CHECK condition] [,...] }
[**PRIMARY KEY** (listOfColumns),]
{ [UNIQUE (listOfColumns),] […,] }
{ [[**FOREIGN KEY** (listOfFKColumns)
REFERENCES ParentTableName [(listOfCKColumns)],
[ON UPDATE referentialAction]
[ON DELETE referentialAction]]] [,…] }
{ [CHECK searchCondition)] [,…] } **)**

Was passiert im Falle eines Updates, des Löschens eines Primärschlüssels mit den bereits referenzierten Tupeln

Semantik:

- Erstellt eine Tabelle mit Spalten der angegebenen Datentypen.
- NOT NULL garantiert, dass ein Wert eingegeben werden muss.
- DEFAULT legt einen Standardwert fest.
- PRIMARY KEY impliziert NOT NULL.
- FOREIGN KEY definiert einen Fremdschlüssel mit Referenzverhalten SET NULL, CASCADE, RESTRICT, SET DEFAULT

recherchieren

Fritz Laux
Ilia Petrov
Reutlingen
University

Beispiel: Relationen Branch und Staff (Wiederholung)

Attributes

Branch

tuple

Relation

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Cardinality

Degree

Primary key

Foreign key

Staff

Relation

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Fritz Laux
Ilia Petrov
Reutlingen
University

Beispiel: CREATE TABLE

```
CREATE TABLE PropertyForRent (
        propertyNo      integer,
        rooms           smallint    DEFAULT 4 NOT NULL,
        rent            decimal(6,2) DEFAULT 600.00 NOT NULL,
        type            varchar(12)
                                CHECK (typ in ('flat‘,‘appt‘,‘studio‘)),
        staffNo         integer NULL,
        branchNo        char(4) CHECK (branchNo > ‘B000‘ ),
        PRIMARY KEY (propertyNo),
        FOREIGN KEY (staffNo) REFERENCES Staff
                ON DELETE SET NULL ON UPDATE CASCADE

);
```

Try it

Fritz Laux
Ilia Petrov
Reutlingen
University

NULL Werte

Fritz Laux
Ilia Petrov
Reutlingen
University

NULL

- *Nullmarken können verschiedene Bedeutung haben:*
  - Attribut trifft bei einem Tupel nicht zu (Provision bei Angestellten mit festem Gehalt, Entbindungen bei männlichen Patienten)
  - Wert existiert, ist aber unbekannt (unbekanntes Gehalt)
  - Wert existiert nicht (Name des Ehegatten)
  - Wert ist nicht definiert (Maximalwert einer leeren Menge)
  - Wert ist ungültig (Alter eines Angestellten ist 95 Jahre)
  - Wert wurde nicht angegeben (Fragebögen)
  - Eingefügte Werte bei outer-join und outer-union

    `--> Viele verschiedene Fälle führen zu NULL`
- *Unterscheidung zwischen prinzipiellen Eigenschaften von Nullmarken und deren Realisierung in SQL*

Fritz Laux
Ilia Petrov
Reutlingen University

NULL-Werte und Three-Valued Logic

- *Prinzip: NULL ist kein Wert mit dem man rechnen kann!*
- *Arithmetische Operationen (+, -, *) mit NULL ergeben NULL*
  - Beispiel: X ist NULL → x+3 oder x*0 ergeben NULL
- *Vergleichsoperationen (=, <, >, >=, <=, <>) mit NULL ergeben Ergeben* ***UNKNOWN***
  - SQL hat eine dreiwertige Logik, die nicht nur aus TRUE und FALSE besteht, sondern auch aus einem dritten Wert UNKNOWN.
  - Beispiel: X ist NULL → x < 3 ergibt UNKNOWN
- *Boolesche Operationen {AND, OR, NOT} → siehe nächste Folie*
- *WHERE Bedingungen*
  - Nur wenn die Bedingung TRUE ergibt → Tupel im Ergebnis
  - Falls FALSE oder UNKNOWN →Tupel nicht im Ergebnis
  - SELECT * FROM t **WHERE b <> NULL;**
  - SELECT * FROM t **WHERE b IS NOT NULL;**
  - SELECT * FROM t **WHERE b IS NULL;**
- GROUP BY: NULL wird als Wert betrachtet → eigenständige Gruppe

Fritz Laux
Ilia Petrov
Reutlingen
University

NULL-Werte und Three-Valued Logic

	X	**Y**	x **AND** y	x **OR** y	**NOT** x
1	TRUE	TRUE	TRUE	TRUE	FALSE
2	TRUE	UNKOWN	UNKNOWN	TRUE	FALSE
3	TRUE	FALSE	FALSE	TRUE	FALSE
4	UNKOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
5	UNKOWN	UNKOWN	UNKNOWN	UNKNOWN	UNKNOWN
6	UNKOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
7	FALSE	TRUE	FALSE	TRUE	TRUE
8	FALSE	UNKOWN	FALSE	UNKNOWN	TRUE
9	FALSE	FALSE	FALSE	FALSE	TRUE

X	**Y**	x **>** y | x **=** y | ...	x **+** y | x * y
NULL	TRUE	UNKOWN	NULL
NULL	FALSE	UNKOWN	NULL

Fritz Laux
Ilia Petrov
Reutlingen University

Beispiele: NULL-Werte und Three-Valued Logic (3VL)

Tabelle t

A	B	C
11	NULL	1
12	NULL	2
13	3	NULL
13	1	3

- *SELECT **b+1** FROM t;*
  - Werden NULL-Werte von B als 0 betrachtet?
- *SELECT * FROM t WHERE c **>=** 1;*
  - SELECT * FROM t WHERE c+a **>=** 1;
- *SELECT * FROM t WHERE b=3 **OR** c>1;*
- *SELECT * FROM t WHERE b=3 **AND** c>1;*
- *SELECT * FROM t WHERE a <= 9 **OR** a > 9;*
  - Ergebnis: alle Tupel der Tabelle t?
- *SELECT * FROM t WHERE b <= 9 **OR** b > 9;*
  - Ergebnis: alle Tupel der Tabelle t?
- *SELECT SUM(c), b FROM t **GROUP BY** b;*
- ***Interessant**: Konzipieren Sie SQL Anfragen, um die Wahrheitstabelle der 3VL zu überprüfen!*

Fritz Laux
Ilia Petrov
Reutlingen
University

NULL

- *Felder, die mit NOT NULL gekennzeichnet sind dürfen keine NULL-Marken enthalten*
- *Felder, die als PRIMARY KEY dienen müssen NOT NULL sein*
- *Defaults können an Stelle von NULL-Marken definiert werden*
  - ☞ Beachte: Kapitel NULL und Aggregatenfunktionen!

Fritz Laux
Ilia Petrov
Reutlingen
University

Datentypen

Fritz Laux
Ilia Petrov
Reutlingen
University

SQL:2003 Data Types

- *Standard (build in) Datentypen (SQL:1999, SQL:2003)*

Data type	*Declarations*
Boolean	*boolean*
Character	*char, varchar, nchar, nvarchar*
Exact numeric	*numeric, decimal, integer, smallint, bigint, tinyint*
Approximate numeric	*float, real, double precision*
Datetime	*date, time, timestamp*
Temporal	*Interval, datetime*
XML type	*XML*
Large objects (LOBs)	*clob, nclob, blob*

Fritz Laux
Ilia Petrov
Reutlingen University

DOMAINs

- *Persistente Definition (Katalog)*
  - ☞ Datentyp, (optionaler) Default, (optionale) Constraints, (optionale) Ordnung
  - ☞
- *Kann an Stelle der Datentyp Definition in Spaltendef. Verwendet werden*

```
CREATE DOMAIN money AS DECIMAL (7,2);

CREATE DOMAIN shirt_size AS CHAR (1)
                    DEFAULT 'M'
                    CONSTRAINT valid_sizes
                    CHECK (value IN ('S','M','L','X')
);

CREATE TABLE shirts (
   style CHAR(5), size  shirt_size, list_price  money
);
```

schauen ob das wichtig ist

Fritz Laux
Ilia Petrov
Reutlingen
University

Veränderungen an Datenbankobjekt-Definitionen

Fritz Laux
Ilia Petrov
Reutlingen
University

ALTER – und DROP TABLE

- *Syntax:*

  ALTER TABLE *Tablename* {ADD | DROP | ALTER} Col-*Option*

- *Beispiel:*

  ALTER TABLE Staff add gender char(1);
  - ☞ Fügt Spalte ‘gender’ der Tabelle *Staff hinzu*

  ALTER TABLE Staff
  ADD gender SET DEFAULT 'F';
  - ☞ Bestimmt ‘F’ (female) als Standardwert für *gender*

- *Syntax:*

  DROP TABLE *TableName* [RESTRICT | CASCADE]

- *Beispiel:*

  DROP TABLE PropertyForRent*;*
  - ☞ Löscht Tabelle ‘*PropertyForRent* ‘und alle Einträge.

Fritz Laux
Ilia Petrov
Reutlingen
University

ALTER TABLE

- *Bestehende Relationen können geändert werden, indem ein neues Attribut eingefügt wird*

  ALTER TABLE base-table ADD column data-type;

  ALTER TABLE staff ADD (gender CHAR (1));

- *Eingefügtes Attribut darf nicht NOT NULL sein.*
  - Definition im Katalog wird erweitert, bei nächstem Zugriff auf ein Tupel wird NULL eingefügt, bevor Benutzer es sieht. Bei der nächsten Schreiboperation wird das erweiterte Tupel geschrieben, wenn ein nicht-NULL Wert eingegeben wird

ALTER TABLE staff ADD (gender CHAR (1) DEFAULT 'F' NOT NULL);

Fritz Laux
Ilia Petrov
Reutlingen
University

Views

Definition: *View (Tabellensicht)*

Eine virtuelle Tabelle, die durch eine SQL-Abfrage definiert wurde.

Definition: Basistabelle (base table)

Eine mit CREATE TABLE erstellte Tabelle.

Eigenschaften eines Views

- Ein View basiert auf einer oder mehreren Basistabellen
- Ein View reflektiert jederzeit die aktuellen Daten der Basistabellen.
- Ein View kann Abfragen vereinfachen und benutzeroptimierten Datenzugriff ermöglichen
- Änderungsoperationen (Update, Insert) auf einem View sind möglich wenn alle Konsistenzbedingungen und Zugriffsrechte der betroffenen Basistabellen erfüllt sind → siehe später

Syntax:

CREATE VIEW ViewName [(newColumnName [,...])]
AS sub-select

Fritz Laux
Ilia Petrov
Reutlingen
University

Beispiel: CREATE VIEW

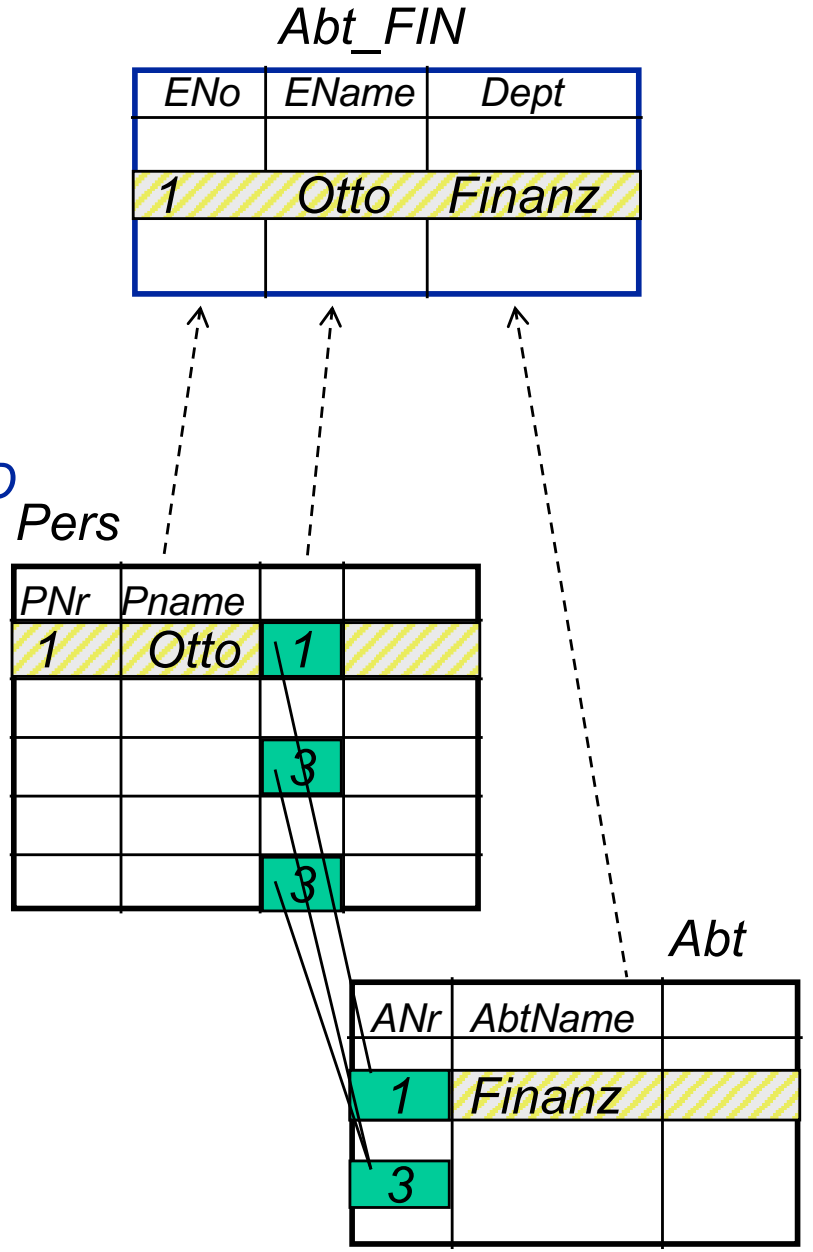
- *SELECT ENo from Abt_FIN;*

- *CREATE VIEW Abt_FIN (ENo, EName, Dept) AS*

  SELECT PNr, PName, AbtName
  FROM Pers, Abt
  WHERE Pers.AbtNr = Abt.Anr AND
  Abt.AbtName = 'Finanz';

- *CREATE TABLE Pers*
  (PNr int primary key, PName char(20),
  AbtNr int references Abt, ...);

- *CREATE TABLE Abt*
  (ANr int primary key, AbtName char(20), ...);



Fritz Laux
Ilia Petrov
Reutlingen
University

CREATE INDEX

Syntax:

CREATE [UNIQUE] INDEX index-name
ON base-table (column [order] [, column [order] ...)
[CLUSTER] ;

Semantik:

- Erstellt einen Index für die angegebenen Spalten der Tabelle.
- Der Index beschleunigt Suchvorgänge, welche die indizierte/n Spalte/n enthalten.
- CREATE INDEX ist nicht Teil des SQL-Standards, wird aber von allen Datenbankherstellern angeboten
- Viele DB-Systeme verwenden einen eindeutigen (unique) Index, um die Eindeutigkeit von Spaltenwerten sicherzustellen

- Order kann ASC (default) oder DESC sein
- CLUSTER besagt, daß es ein „clustering Index“ ist (höchstens einer pro Relation, physisches Gruppieren)
- UNIQUE besagt, daß keine Duplikate erlaubt sind (Wahrung der Schlüsseleindeutigkeit)
- DROP INDEX index-name; löscht einen Index

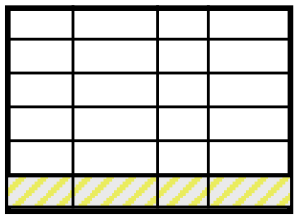
Fritz Laux
Ilia Petrov
Reutlingen
University

SQL-Syntax (DML)

- *Syntax*

- *INSERT INTO <table>*
  VALUES (<val11>,
  <val2>,...)

- *INSERT INTO <table>*
  SELECT <col1>, <col2>, ...
  FROM <source-tab>
  [WHERE <condition>]

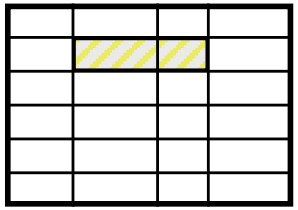


- *Beispiele:*

- *INSERT INTO Pers*
  VALUES (007, 'Bond
  James', 5, ...);

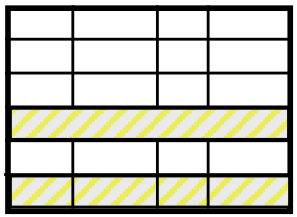
- *INSERT INTO Pers*
  SELECT Nr, Name, ANr
  FROM old_Pers
  WHERE
  Einstelldat > '01-JUN-12';

- *UPDATE <table>*
  SET <col1> = <val1>,
  <col2> = <val2>, ...
  [WHERE <condition>]



- *UPDATE Pers*
  SET PName = 'Banks
  Gordon', AbtNr = 6;

- *DELETE FROM <table>*
  [WHERE <condition>]



- *DELETE FROM Pers*
  WHERE AbtNr = 9;

Fritz Laux
Ilia Petrov
Reutlingen
University

SQL Abfragen

Fritz Laux
Ilia Petrov
Reutlingen
University

Abfrage: SELECT Statement

- *Syntax:*

```
SELECT [DISTINCT | ALL]
        {* | [columnExpression [AS newName]] [,...] }
        FROM        TableName [alias] [, ...]
        [WHERE  condition]
        [GROUP BY       columnList]
        [HAVING condition]
        [ORDER BY       columnList]
```

- *Semantik:*

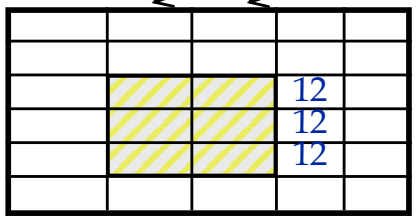
SELECT	*Spalten, die im Ergebnis erscheinen.*
FROM	*Tabelle/n, die für die Abfrage verwendet werden.*
WHERE	*Spezifiziert die Zeilen.*
GROUP BY	*Gruppiert Zeilen mit dem gleichen Spaltenwert.*
HAVING	*Filtert die Gruppierung.*
ORDER BY	*Sortiert das Ergebnis.*

Fritz Laux
Ilia Petrov
Reutlingen
University

Beispiel: Abfrage

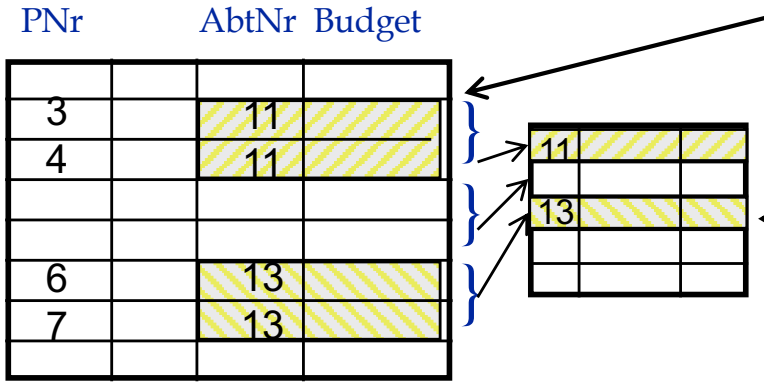
- *SELECT <col1>, <col2>, ...*
  FROM <table>
  [WHERE <condition>]
  [ORDER BY <colx>,…]

- *SELECT PNr, PName,*
  FROM Pers
  WHERE AbtNr = 12
  ORDER BY PName;



- *SELECT <col1>, <col2>, ...*
  FROM <table>
  [WHERE <condition>]
  [GROUP BY <colx>, <coly>, ...
  [HAVING <grp-cond>]]

- *SELECT AbtNr,*
  count(*), sum(Budget)
  FROM Pers
  WHERE PNr <= 10
  GROUP BY AbtNr
  HAVING count(*) >= 2;



Fritz Laux
Ilia Petrov
Reutlingen
University

GROUP BY

- *Table expressions produzieren virtuelle Tabellen*
- *GROUP BY ordnet die Tupel nach einem Gruppierungskriterium und kann auch Werte nach dieser Klassifikation aggregieren*
- *GROUP BY grouping-column [, grouping-column ...]*

```
SELECT      movie_type,      AVG (current_rental_price)
 FROM       movie_titles
GROUP BY  movie_type
```

Fritz Laux
Ilia Petrov
Reutlingen University

Beispiel GROUP BY

```
SELECT      movie_type, AVG (rental_price)
    FROM        movie_titles
    GROUP BY    movie_type
```

TITLE	MOVIE_TYPE	RENTAL_PRICE
Lethal Weapon	Action	2.99
Outlaw	Western	2.99
Kelly‘s Heroes	War	2.99
Shaft‘s Big Score	Action	1.99
Unforgiven	Western	3.99
Shaft	Action	2.99

MOVIE_TYPE	AVG(RENTAL_PRICE)
Action	2.66
War	2.99
Western	3.49

- **Auswertungsreihenfolge?**

```
SELECT      movie_type,
            AVG (rental_price)
FROM        movie_titles
WHERE       movie_studio IN
            (Paramount, Universal)
GROUP BY    movie_type
```

Fritz Laux
Ilia Petrov
Reutlingen
University

HAVING

- *HAVING ist ein zusätzlicher Filter*
- *HAVING wirkt auf die Relation der vorhergehenden Klausel*

HAVING search-condition

- *HAVING bezieht sich auf die Werte der Grouping Clause*

HAVING movie_type = ‘Western‘

OR movie_type = ‘War’

Fritz Laux
Ilia Petrov
Reutlingen
University

Verdischtungsfunktionen

Fritz Laux
Ilia Petrov
Reutlingen
University

Aggregatfunktion und Gruppierung

↳ *Aggregatfunktionen avg, max, min, count, sum*

```
SELECT AVG(Semester)
FROM Studenten;
```

```
SELECT gelesenVon, AVG(SWS)
FROM Vorlesungen
GROUP BY gelesenVon;
```

```
SELECT gelesenVon, Name, AVG(SWS)
FROM Vorlesungen, Professoren
WHERE gelesenVon = PersNr
GROUP BY gelesenVon, Name
HAVING AVG(SWS) >= 3;
```

Fritz Laux
Ilia Petrov
Reutlingen
University

Aggregierung: Set Functions

- *Berechnungsfunktionen auf Tabellen*
- *COUNT zählt die Zeilen in einer Tabelle*

  COUNT (*) *zählt alle Zeilen*

  COUNT (<attr>) *zählt die Werte eines Attributs (Vorsicht: NULL-Werte → Nächste Folie)*

- *Kann noch weiter qualifiziert werden durch ALL, DISTINCT, Attributnamen oder Prädikate*

```
SELECT COUNT(*)
FROM MOVIES_STARS
WHERE ACTOR_LAST_NAME = 'Moore';
```

```
SELECT COUNT (DISTINCT MOVIE_TITLE)
FROM MOVIES_STARS ;
```

Fritz Laux
Ilia Petrov
Reutlingen
University

MAX, MIN, SUM, AVG

- *MAX wählt den höchsten Wert aus*
- *MIN wählt den Minimalwert aus*
- *SUM summiert alle Instanzen*
- *AVG bildet den arithmetischen Durchschnitt*
  - Vorsicht! Umgang mit NULL!
  - COUNT (*) zählt alle Zeilen, inkl. die mit NULL
  - SUM ignoriert NULL in der Summe
  - AVG zählt beim Durchschnitt nur die Tupel, die nicht NULL im summierten Attribut haben
  - AVG ≠ SUM / COUNT

Fritz Laux
Ilia Petrov
Reutlingen
University

Zeichenketten in SQL

Fritz Laux
Ilia Petrov
Reutlingen
University

‚Ähnlichkeitssuchen' in SQL

- *Keyword LIKE*

%	→	Wildcard für n Zeichen
_	→	Wildcard für ein Zeichen
ESCAPE '$'	→	Wildcard-Ersatz

- *Beispiel:* *Strings ABCDE und ABCD*

LIKE 'ABC%'	findet ABCDE, ABCD und ABC
LIKE 'ABC_'	findet nur ABCD
LIKE '%10$%' ESCAPE '$'	findet '10%' und '810%'

Fritz Laux
Ilia Petrov
Reutlingen
University

SQL Query mit LIKE

```
SELECT title
FROM movie_titles
WHERE title LIKE 'Bev%' ;
```

Beverly Hills Cop

Beverly Hills Cop II

```
SELECT title
FROM movie_titles
WHERE title LIKE '%Bev%' ;
```

Beverly Hills Cop

Beverly Hills Cop II

Down and Out in Beverly Hills

Fritz Laux
Ilia Petrov
Reutlingen
University

Viele weitere Funktionen zur Bearbeitung von Zeichenkennten

- *SUBSTRING*
- *CONCAT*
- *LEFT*
- *LEN*
- *LOWER*
- *REPLACE*
- *RIGHT*
- *TRIM*
- …

Fritz Laux
Ilia Petrov
Reutlingen
University

SQL Verbundoperationen (Join)

Fritz Laux
Ilia Petrov
Reutlingen
University

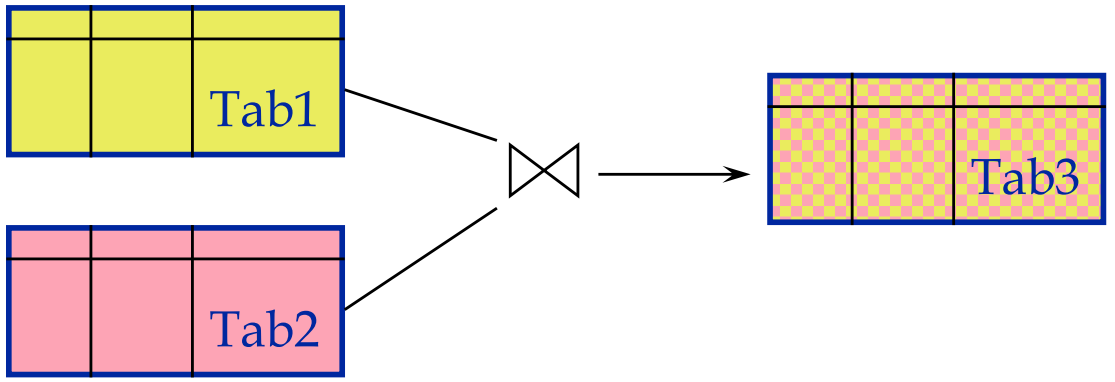
SQL-Syntax JOIN (Verbund) - Wiederholung

```
SELECT <col1>,<col2>, ...
FROM <tab1>, <tab2>
WHERE <condition>;
```

```
SELECT PNr, PName, AbtName, ...
FROM Pers, Abt
WHERE Pers.AbtNr = Abt.ANr;
```

```
SELECT PNr, PName, AbtName, ...
FROM Pers JOIN Abt ON (Pers.AbtNr = Abt.Anr);
```

$f: Tab_1 \bowtie Tab_2 \rightarrow Tab_3$



Verbund:

Bei der Selektion über mehrere Tabellen wird zuerst das *Kartesische Produkt* gebildet, dann werden die Tupel, welche die WHERE-Bedingung erfüllen, ausgewählt.

Eine Selektionsbedingung, welche nur die Zeilen mit korrespondierenden Attributen auswählt und diese Spalten nur einmal aufführt (tab1.attribx = tab2.attribx), heißt **'Natürlicher Verbund'.**

Fritz Laux
Ilia Petrov
Reutlingen University

Example: Join Operation - Wiederholung

- ***List names of all clients who have viewed a property along with any comments supplied.***
- SELECT c.clientNo, fName, lName, propertyNo, comment
  FROM Client c, Viewing v
  WHERE c.clientNo = v.clientNo;
  - ☞ Nur jene Zeilen aus beiden Tabellen mit gleichen Werten für die clientNo-Spalte (c.clientNo = v.clientNo) werden berücksichtigt.

Client

clientNo	fName	lName	…
CR56	Aline	Stewart	…
CR74	Mike	Ritchie	…
CR62	Mary	Tregear	…

Viewing

clientNo	propertNo	…	comment
CR56	PG36	…	
CR56	PA14		too small
…	…	…	…
CR62	PA14	…	no dining room

clientNo	fName	lName	propertyNo	comment
CR56	Aline	Stewart	PG36	
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR62	Mary	Tregear	PA14	no dining room
CR76	John	Kay	PG4	too remote

Fritz Laux
Ilia Petrov
Reutlingen
University

SQL:1992 Join Syntax

- *SQL:1992 (und neuere Versionen) bieten eine spezielle Syntax für Verbundoperationen wobei FROM … WHERE durch FROM … JOIN … ersetzt werden*
  - ☞ FROM <Client> NATURAL JOIN <Viewing>
    - ⇨Für den natürlichen Verbund
  - ☞ FROM <Client> c JOIN <Viewing> v ON c.clientNo = v.clientNo
    - ⇨Für den expliziten Verbund
  - ☞ FROM <Client> JOIN <Viewing> USING (clientNo)
    - ⇨Für Verbundoperationen mit gleichnamigen Spalten

Fritz Laux
Ilia Petrov
Reutlingen
University

JOIN

STUDENTS		
LNAME	FNAME	NICK
James	Robert	Bob
Long	Robert	Bobby

ENROLLMENTS		
LAST	FIRST	COURSE
James	Robert	CS101
Smith	Diane	CS200

SQL-89 inner joins (natural inner join)

```
SELECT  lname, nick, course
FROM    students, enrollments
WHERE   students.lname = enrollments.last AND
                  students.fname = enrollments.first;
```

LNAME	NICK	COURSE
James	Bob	CS101

Fritz Laux
Ilia Petrov
Reutlingen
University

SQL-92 JOINs - CROSS JOIN

- *SQL-92 führt verschiedene zusätzliche Joins ein: cross join (kartesisches Produkt), unionjoin, outerjoin*

- *CROSS JOIN*

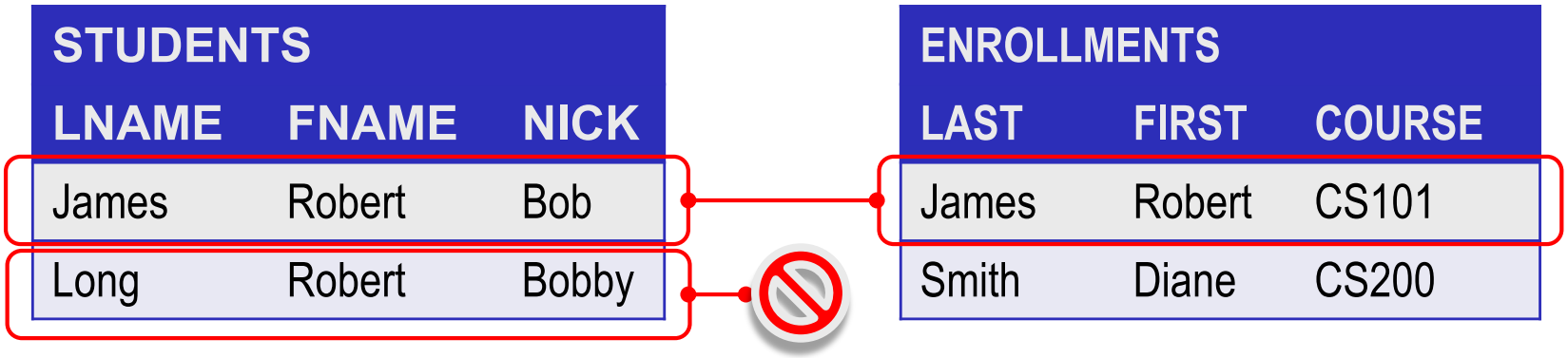
```
SELECT    *
FROM              (students   CROSS JOIN enrollments );
```

LNAME	FNAME	NICK	LAST	FIRST	COURSE
James	Robert	Bob	James	Robert	CS101
James	Robert	Bob	Smith	Diane	CS200
Long	Robert	Bobby	James	Robert	CS101
Long	Robert	Bobby	Smith	Diane	CS200

Fritz Laux
Ilia Petrov
Reutlingen
University

LEFT OUTER JOIN

*SELECT **
FROM *students **LEFT OUTER JOIN** enrollments*
ON lname = last AND fname = first ;



STUDENTS

LNAME	FNAME	NICK
James	Robert	Bob
Long	Robert	Bobby

ENROLLMENTS

LAST	FIRST	COURSE
James	Robert	CS101
Smith	Diane	CS200

RESULT

LNAME	*FNAME*	*NICK*	*LAST*	*FIRST*	*COURSE*
James	*Robert*	*Bob*	*James*	*Robert*	*CS101*
Long	*Robert*	*Bobby*	***NULL***	***NULL***	***NULL***

Fritz Laux
Ilia Petrov
Reutlingen
University

RIGHT OUTER JOIN

*SELECT **
FROM *students **RIGHT OUTER JOIN** enrollments*
ON lname = last AND fname = first ;

STUDENTS

LNAME	FNAME	NICK
James	Robert	Bob
Long	Robert	Bobby

ENROLLMENTS

LAST	FIRST	COURSE
James	Robert	CS101
Smith	Diane	CS200

RESULT

LNAME	*FNAME*	*NICK*	*LAST*	*FIRST*	*COURSE*
James	*Robert*	*Bob*	*James*	*Robert*	*CS101*
NULL	***NULL***	***NULL***	*Smith*	*Dianne*	*CS200*

Fritz Laux
Ilia Petrov
Reutlingen
University

NATURAL FULL OUTER JOIN

```
SELECT *
FROM    students NATURAL FULL OUTER JOIN v_enrollments;
```

STUDENTS

LNAME	FNAME	NICK
James	Robert	Bob
Long	Robert	Bobby

V_ENROLLMENTS

LNAME	FNAME	COURSE
James	RobertCS101	
Smith	Diane	CS200

LNAME	FNAME	NICK	COURSE
James	Robert	Bob	CS101
Long	Robert	Bobby	NULL
Smith	Diane	NULL	CS200

Fritz Laux
Ilia Petrov
Reutlingen
University

Point und Range Queries

- *Point Queries:*
  - ☞ SELECT * FROM movies WHERE ID = 123;

- *Range Queries beziehen sich auf Vergleiche mit Wertbereichen in der WHERE Klausel*

- *2 Arten von Range Queries:*

Movie(title, year, length, inColor, studioName, producerC#)

- ***select*** * ***from*** *movie*
  where *Year > = 1998* ***and*** *Year < = 2013;*
- ***select*** * ***from*** *movie*
  where *Year* ***between*** *1998* ***and*** *2013;*
- ***select*** * ***From*** *movie*
  where *Year* ***in*** *(1998, 2000, 2010, 2013);*

Fritz Laux
Ilia Petrov
Reutlingen
University

SQL Sichten

Fritz Laux
Ilia Petrov
Reutlingen
University

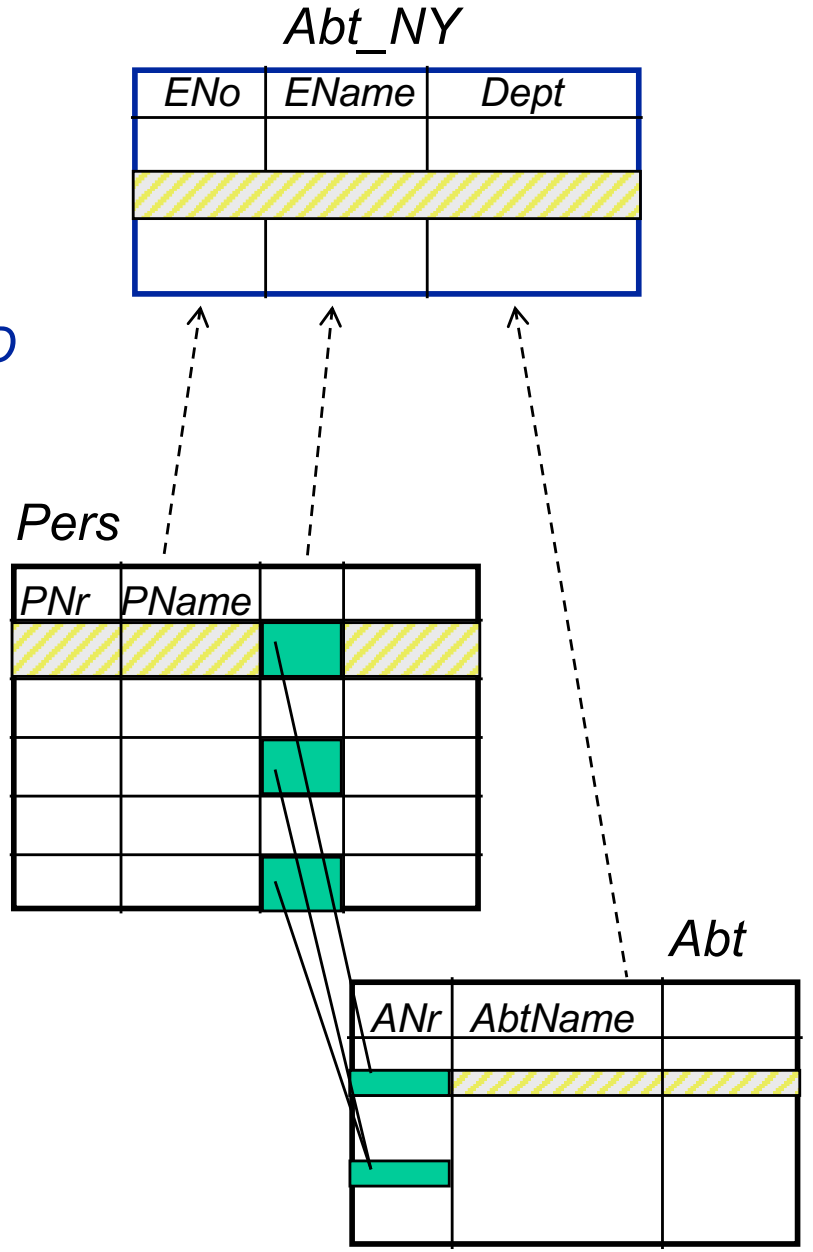
VIEWs (Sichten) - Wiederholung

- *CREATE VIEW Abt_NY (ENo, EName, Dept) AS*

  SELECT PNr, PName, AbtName
  FROM Pers, Abt
  WHERE Pers.AbtNr = Abt.Anr AND
  Abt.AbtName = 'New York';

- *CREATE TABLE Pers*
  (PNr int primary key, PName char(20),
  AbtNr int references Abt, ...);

- *CREATE TABLE Abt*
  (ANr int primary key, AbtName char(20), ...);



Fritz Laux
Ilia Petrov
Reutlingen
University

VIEWs (Sichten) - Wiederholung

- *Sicht (View): benannte, abgeleitete, virtuelle Relation*
- *Sichten können von Basisrelationen und anderen Sichten abgeleitet werden*
- *Korrespondenz zum externen Schema bei ANSI/SPARC, d.h. View agiert als Filter (Unterschied: Benutzer sieht ein externes Schema kann aber viele Sichten und Tabellen sehen)*
- *Sichten werden im Schema definiert und Intension wird im Katalog gespeichert*

  CREATE VIEW view [(column-commalist)] AS query-exp

  [WITH [CASCADED | LOCAL] CHECK OPTION]

  DROP VIEW view

Fritz Laux
Ilia Petrov
Reutlingen
University

Views

- *Views werden durch eine Query definiert ➔ Sichtnamen und Queries sind austauschbar*
- *Sichten werden i.a. nicht permanent gespeichert (werden on-the-fly erstellt)*
- *Wenn Sichten gespeichert werden (materialized views) muss Konsistenz gewahrt werden!!!*
- *Selects auf Views unproblematisch (wirkt wie Verundung der Selektionsprädikate der Query und der Sichtdefinition)*
- *Abbildungsprozeß für Sichten kann mehrstufig sein*
- *Abbildungsmächtigkeit ist eingeschränkt (Prim. Schlussel, keine Schachtelung von GROUP BY)*

Fritz Laux
Ilia Petrov
Reutlingen
University

Updatable Views

- *Updates auf Sichten können problematisch sein*
- *Änderungsoperationen auf Sichten erfordern, daß jedem Tupel der Sicht genau ein Tupel der Basisrelation zugeordnet werden kann*
  - Sichten auf Basisrelation sind nur aktualisierbar, wenn der Primärschlüssel in der Sicht enthalten ist
  - müssen alle nicht-NULL Attribute in Sichtdefinition enthalten sein
  - müssen alle nicht-NULL Attribute zumindest DEFAULT-Werte zugewiesen bekommen
  - eingefügte Tupel müssen das die Sicht definierende Prädikat erfüllen (CHECK-Option)
  - Sichten, die über Aggregatfunktionen und GROUP BY definiert sind, sind nicht aktualisierbar
  - Sichten, die über mehr als eine Relation definiert werden sind (i.a.) nicht aktualisierbar

Fritz Laux
Ilia Petrov
Reutlingen
University

SQL Geschachtelte Anfrage

Fritz Laux
Ilia Petrov
Reutlingen
University

Tupel/Tabellen-Variablen

- *Es ist möglich Variablen zu definieren, die durch alle Tupel der Tabelle durchgehen.*
  - ☞ Ein Tupel pro Zeitpunkt (analog zu FOREACH)

- *Dafür wird die FROM-Klausel genutzt*
  - ☞ FROM Staff **s**;

```
SELECT s.name
FROM staff s
WHERE s.salary > 8000;
```

- *Der Tabellenname darf als Tupelvariable genutzt werden*

```
SELECT staff.salary
FROM staff, branch
WHERE staff.branchFK = branch.branchID;
```

Fritz Laux
Ilia Petrov
Reutlingen
University

Geschachtelte Anfrage – Wiederholung

- *Welche Prüfungen sind besser als durchschnittlich verlaufen?*

- *SELECT **
  FROM prüfen
  WHERE Note < ~~avg (Note~~);

*In SQL **NICHT** möglich!*

- *Unteranfrage in der WHERE-Klausel*

```
SELECT *
FROM prüfen
WHERE Note < (
        SELECT avg (Note) FROM prüfen
);
```

Fritz Laux
Ilia Petrov
Reutlingen
University

Korrelierte und Unkorrelierte Unteranfragen

↳ *Korrelierte Unteranfrage*

☞ Wie häufig wird die unteranfrage ausgewertet?

```
select p.Name
from Professoren p
where not exists ( select *
                   from Vorlesungen v
                   where v.gelesenVon = p.PersNr );
```

Korrelation

↳ *Unkorrelierte Unteranfrage: wird nur einmal ausgewertet (evtl. performanter)*

```
select p.Name
from Professoren p
where p.PersNr not in ( select v.gelesenVon
                        from Vorlesungen v);
```

Fritz Laux
Ilia Petrov
Reutlingen University

Beispiel: Korrelierte Unteranfragen

Fremdschlüssel

Professoren			
PersNr	**Name**	**Rang**	**Raum**
1	**Otto**		**023**
2	**Mayer**		**027**
3	**Müller**		**030**

```
select p.Name
from Professoren p
where not exists (
    select *
    from Vorlesungen v
   where v.gelesenVon = p.PersNr
);
```

Vorlesungen			
VorlNr	**Titel**	**SWS**	**gelesen Von**
V42	**ABC**	**2**	**1**
V47	**DEF**	**2**	**3**
V43	**GHI**	**4**	**2**
V57	**JKL**	**4**	**1**
V64	**MNO**	**5**	**1**

Fritz Laux
Ilia Petrov
Reutlingen
University

Geschachtelte Anfrage (Forts.)

- *Unteranfrage in der SELECT-Klausel*
- *Für jedes Ergebnistupel wird die Unteranfrage ausgeführt*
- *Man beachte, dass die Unteranfrage korreliert ist (greift auf Tupel/Attribute der umschließenden Anfrage zu)*
- ***SELECT** PersNr, Name, (**SELECT** sum (SWS) as Lehrlast*
  ***FROM** Vorlesungen*
  ***WHERE** gelesenVon=p.PersNr)*
  ***FROM** Professoren p;*

  - Professoren (*PersNr Int*, Name CHAR(30), Rang CHAR(2), Raum Int);
  - Vorlesungen (VorlNr Int, Titel CHAR(30), SWS Int, *gelesenVon Int REFERENCES Professoren*);

Fritz Laux
Ilia Petrov
Reutlingen
University

Entschachtelung der Unteranfragen

↳ *Liste alle Studenten auf, die älter sind als der jungste Professor*

↳ *korrelierte Formulierung*

```
SELECT s.*
FROM Studenten s
WHERE EXISTS
        (SELECT p.*
        FROM Professoren p
        WHERE p.GebDatum < s.GebDatum);
```

↳ *Äquivalente unkorrelierte Formulierung*

```
SELECT s.*
FROM Studenten s
WHERE s.GebDatum >
        (SELECT min(p.GebDatum)
         FROM Professoren p);
```

☞ Vorteil: Unteranfrageergebnis kann materialisiert werden

☞ Unteranfrage braucht nur einmal ausgewertet zu werden

Fritz Laux
Ilia Petrov
Reutlingen
University

Existenzquantoren: EXISTS und NOT EXISTS

- *EXISTS und NOT EXISTS werden nur bei Subqueries benutzt.*
  - ☞ NOT EXISTS ist die Negation von EXISTS.

- *EXISTS Quantor ist wahr, wenn Subquery mindestens ein Tupel liefert.*

- *Beispiel:* ***Find all staff who work in a London branch.***

```
SELECT staffNo, fName, lName, position
FROM Staff s
WHERE EXISTS
            (SELECT * FROM Branch b
            WHERE b.branchNo = s.branchNo
            AND city = 'London');
```

Korrelation

- *Die Suchbedingung s.branchNo = b.branchNo ist notwendig, um den Bezug zu den Tupeln der äußeren Query herzustellen*

Fritz Laux
Ilia Petrov
Reutlingen University

Allquantor: Der Vergleich mit "all"

- *Vergleich mit ‚ALL':*

  Gibt dann WAHR zurück, wenn die Bedingung für **alle** Elemente der Menge erfüllt ist. (AND)

Tabelle t		
A	**B**	**C**
11	NULL	1
12	NULL	2
13	3	NULL
13	1	3

```
SELECT a FROM t
WHERE a >= ALL(SELECT a FROM t);
```

```
SELECT b FROM t WHERE b >= ALL(SELECT b FROM t);
```

```
SELECT a FROM t WHERE a <= ALL(SELECT a FROM t);
```

```
SELECT b FROM t WHERE b <= ALL(SELECT b FROM t);
```

- *Vergleich mit ‚ANY'*

  Gibt dann WAHR zurück, wenn die Bedingung für **mindestens ein** Element der Menge erfüllt ist.

  (**b > ANY (e_1, … e_n) → b>e_1 OR ... OR b>e_n**)

```
SELECT a FROM t WHERE a >= ANY(SELECT a FROM t);
SELECT b FROM t WHERE b >= ANY(SELECT b FROM t);  ☺
```

Fritz Laux
Ilia Petrov
Reutlingen University

EXISTS, IN, ANY, ALL und NULL-Werte

- *IN ähnliche Funktionalität zu =ANY*
  - **b IN (e_1, … ,e_n) → b=e_1 OR ... OR b=e_n**
  - SELECT * FROM t
    WHERE b = ANY (NULL, NULL,1,3);
  - SELECT * FROM t
    WHERE b = ANY (select b from t);
  - SELECT * FROM t
    WHERE b IN (NULL, NULL,1,3);
  - SELECT * FROM t
    WHERE b IN (select b from t);

Tabelle t

A	B	C
11	NULL	1
12	NULL	2
13	3	NULL
13	1	3

- *EXISTS: zählt Mengen-Elemente/Tupel nicht die Werte(!), sogar dann wenn diese NULL sind*
  - SELECT * FROM t
    WHERE EXISTS (select b from t where a in (11,12));
  - SELECT * FROM t WHERE EXISTS (select b from t);
- *IN und EXISTS sind sich diesbezüglich ähnlich*
  - SELECT * FROM t WHERE b IN (select b from t);
  - SELECT * FROM t t1
    WHERE EXISTS (select c from t t2 where t1.c=t2.c);

Fritz Laux
Ilia Petrov
Reutlingen
University

EXISTS, IN, ANY, ALL und NULL-Werte

- *NULL-Werte ändern das:*
  - **a NOT IN (e_1,e_2,..,e_n) → a!=e_1 AND a!=e_2 AND...AND a!=e_n**
  - SELECT * FROM t WHERE a NOT IN (1,2,3);
  - SELECT * FROM t WHERE a NOT IN (NULL,1,2,3);

- *NOT IN und NOT EXISTS unterscheiden sich!*
  - SELECT * FROM t WHERE b IN (select b from t);
  - SELECT * FROM t t1 WHERE EXISTS (select c from t t2 where t1.c=t2.c);
  - SELECT * FROM t WHERE b NOT IN (select b from t);
  - SELECT * FROM t t1 WHERE NOT EXISTS (select c from t t2 where t1.c=t2.c);

Tabelle t

A	B	C
11	NULL	1
12	NULL	2
13	3	NULL
13	1	3

Fritz Laux
Ilia Petrov
Reutlingen
University

Top-K

- *Top-K: Bestimmung der ‚ersten' K Tupel die ein Kriterium ‚am Besten' erfühlen.* ***Ordnung*** *der Ergebnisse - wichtig!*
  - ☞ Liste die 2 Professoren auf, die die Vorlesungen mit den meisten SWS lesen.
    - ⇨ Oder: Liste die 10 best-verdienenden Angestellten auf

- *Kanonischer Algorithmus*
  - ☞ Korrelierte unteranfrage
    - ⇨ (1) Für jedes Tupel - **v**
    - ⇨ (2) Wähle die Tupel, die ‚besser' sind – **bv**
      - Und wenn die ‚besseren' **bv** Tupel weniger sind als eine bestimmte Grenze K (hier **2**) gebe das Tupel **v** aus, weil es zu den K-Besten gehört.

```
SELECT DISTINCT sws, gelesenVon
FROM vorlesungen v
WHERE 2 >= (
      SELECT count( DISTINCT bv.sws) FROM vorlesungen bv
      WHERE v.sws <= bv.sws
 );
```

Fritz Laux
Ilia Petrov
Reutlingen
University

Top-K - Example

```
SELECT DISTINCT sws, gelesenVon
FROM vorlesungen v
WHERE 2 >= (
      SELECT count( DISTINCT bv.sws) FROM vorlesungen bv
      WHERE v.sws <= bv.sws
 );
```

VorlNR	TITEL	SWS	GelesenVON
5001	Grundlagen der Informatik	4	2137
5041	Theoretische Informatik	4	2125
5043	Business Intelligence	3	2126
5049	Mathematik 1	2	2125
4052	Logistik	4	2125
5052	Mathematik 2	3	2126
5216	Compiler-Bau	2	2126
5259	Software Engineering	2	2133
5022	Relationalle Datnbanksysteme	2	2134
4630	DBMS Technologien	4	2137

Fritz Laux
Ilia Petrov
Reutlingen
University

Top-K

- *Aufgrund von Algorithmus und DB-spezifischen Optimierungen nicht Standard-konformen Erweiterungen*
  - ORACLE

    SELECT sws, gelesenVon

    FROM (SELECT sws, gelesenVon FROM vorlesungen ORDER BY sws DESC)

    WHERE **rownum <=2**;

  - Microsoft SQL Server

    SELECT **TOP 2** sws, gelesenVon FROM vorlesungen;

  - MySQL, PostgreSQL

    SELECT sws, gelesenVon FROM vorlesungen **LIMIT 2**;

  - General

    SELECT sws, gelesenVon FROM vorlesungen ORDER BY sws

    STOP AFTER 2;

Fritz Laux
Ilia Petrov
Reutlingen
University

Casting der Integer zu Decimal

- Beachten Sie die CAST Funktion sowie die unkorrelierte unteranfrage in der SELECT Klausel

- SELECT **h**.VorlNr, **h**.AnzProVorl, **g**.GesamtAnz,
  cast(h.AnzProVorl as decimal(6,2))/**g**.GesamtAnz as Marktanteil

  FROM (SELECT VorlNr, count(*) as AnzProVorl
  FROM hoeren group by VorlNr) **h**,
  (SELECT count (*) as GesamtAnz FROM Studenten) **g**;

- Äquivalent zu:
  - SELECT h.VorlNr, h.AnzProVorl,
    (SELECT count (*) as GesamtAnz FROM Studenten),
    cast(h.AnzProVorl as decimal(6,2))/(SELECT count (*) as GesamtAnz FROM Studenten) as Marktanteil

    FROM (SELECT VorlNr, count(*) as AnzProVorl
    FROM hoeren group by VorlNr) h;

Fritz Laux
Ilia Petrov
Reutlingen
University

Modularisierung **mit „with“**

*WITH **h** as (select VorlNr, count(*) as AnzProVorl from hoeren*
*group by VorlNr) **,***

***g** as (select count (*) as GesamtAnz from Studenten)*

SELECT h.VorlNr, h.AnzProVorl, g.GesamtAnz,
cast (h.AnzProVorl as decimal(6,2)) / g.GesamtAnz as Marktanteil
FROM g,h

Fritz Laux
Ilia Petrov
Reutlingen
University

Das CASE-Konstrukt

↳ *Die erste qualifizierende when-Klausel wird ausgeführt*

```
select MatrNr, ( case when Note < 1.5 then  ´sehr gut ´
                      when Note < 2.5 then  ´gut ´
                      when Note < 3.5 then  ´befriedigend ´
                      when Note < 4.0 then  ´ausreichend ´
                      else  ´nicht bestanden ´
                      end) AS beschreibung
from pruefen;
```