

4. Anfragen in SQL, Datenbankprogrammierung

tupelorientierter Relationenkalkül

Dreischrittverfahren:

1. Tupelerzeugung

FROM

Semantik: Erzeugung aller Tupel $t_1 \in R_1, \dots, t_n \in R_n$

Syntax: Bindung der Tupel an Tupelvariable

2. Auswahl der gesuchten Tupel

WHERE

Semantik: Auswahl der Tupelkombinationen, die eine Formel erfüllen

Syntax: Setzen der Tupelvariablen mit entsprechend passenden
Attributen und Vergleichen

3. Konstruktion der Ausgabe (Projektion)

SELECT

Semantik: Auswahl der entsprechenden Komponenten in
entsprechender Kombination

Syntax: $\mu_{i,1} \cdot A_1, \dots, \mu_{i,m} \cdot A_m$

Damit allgemeiner Ausdruck:

$\pi_{R_{i,1} \cdot A_1, \dots, R_{i,m} \cdot A_m} (\sigma_{\Phi} (R_1 \times \dots \times R_m))$

$\text{map}_{\pi} (\text{filter}_{\Phi} (\text{map}_{\times} (R_1, \dots, R_m)))$

SELECT DISTINCT $\mu_1 \cdot R_1, \dots, \mu_m \cdot R_m$

FROM R_1, \dots, R_m

WHERE Φ

Monorelationale Anfragen:

```
SELECT      Projektionsliste  
FROM  Tabelle  
WHERE Selektionsbedingung
```

Varianten: SELECT Projektionsliste FROM Tabelle
 SELECT * FROM Tabelle WHERE Selektionsbedingung

Duplikatelimination

```
SELECT DISTINCT  Projektionsliste  
FROM              Tabelle  
WHERE             Selektionsbedingung
```

Ordnen der Resultate

```
SELECT Projektionsliste  
FROM  Tabelle  
WHERE Selektionsbedingung  
ORDER BY  Attribut ASC|DESC
```

Nullwert ist der minimale Wert !

Multirelationale Anfragen

```
SELECT Projektionsliste  
FROM  Tabellenliste (mit Variablen)  
WHERE Selektionsprädikat
```

☒ Welche externen Mitarbeiter haben Urlaubsanspruch, Resturlaub oder Sonderurlaub?

Mitarbeiter		
PersNr	string	KEY
Name	string	
Vorname	string	
Extern	bool	
Adresse	string	
TNr	string	
Urlaubsanspruch	int	
Resturlaub	int	
Sonderurlaub	int	
Institut	string	

```
SELECT Name, Vorname
FROM Mitarbeiter
WHERE Extern = „TRUE“
AND (Urlaubsanspruch IS NOT NULL
OR Resturlaub IS NOT NULL
OR Sonderurlaub IS NOT NULL);
```

⊗ Nenne die Summe aller Urlaubstage jeweils aller Mitarbeiter.

Mitarbeiter		
PersNr	string	KEY
Name	string	
Vorname	string	
Extern	bool	
Adresse	string	
TNr	string	
Urlaubsanspruch	int	
Resturlaub	int	
Sonderurlaub	int	
Institut	string	

```
SELECT PersNr, Name,  
        Urlaubsanspruch +  
        Resturlaub +  
        Sonderurlaub  
FROM Mitarbeiter  
ORDER BY 3 DESC, Name ASC;
```

⊠ Nenne Nummer und Titel der Kurse, deren maximale Teilnehmerzahlen zwischen 10 und 20 liegen.

Kurs		
KursNr	string	KEY
Titel	string	
Inhalt	string	
max Teilnehmer	int	
Endtermin	date	
Leitung	string	

```
SELECT KursNr, Titel
FROM Kurs
WHERE maxTeilnehmer > 9
AND maxTeilnehmer < 21;
```

Komplexe Select-Anweisungen

☒ Welche Voraussetzung hat der Kurs „Programmieren in C“?

Kurs		
KursNr	string	KEY
Titel	string	
Inhalt	string	
maxTeilnehmer	int	
Endtermin	date	
Leitung	string	

```
SELECT k2.KursNr, k2.Titel
FROM Kurs k1, Kurs k2, Voraussetzung
WHERE k1.Titel = "Programmieren in C"
AND k1.KursNr = Voraussetzung.KursNr
AND Voraussetzung.VoraussNr = k2.KursNr;
```

Voraussetzung		
KursNr	string	KEY
VoraussNr	string	KEY

⊗ Welcher Kurs wird an mehreren Orten durchgeführt?

Kurs		
KursNr	string	KEY
Titel	string	
Inhalt	string	
maxTeilnehmer	int	
Endtermin	date	
Leitung	string	

Kursdurchführung		
KursNr	string	KEY
Datum	date	KEY
Anfang	time	KEY
Ende	time	KEY
Ort	string	

```
SELECT Kurs.KursNr, Kurs.Titel
FROM Kurs,
      Kursdurchführung kd1,
      Kursdurchführung kd2
WHERE Kurs.Nr = kd1.KursNr
      AND Kurs.Nr = kd2.KursNr
      AND kd1.Ort < > kd2.Ort;
```

⊗ Gib eine Liste aller genommenen „normalen“ Urlaubstage aller Mitarbeiter von dem Institut aus, das Herr Schmidt leitet.

INSTITUT		
Bezeichnung	String	KEY
Adresse	String	
PersNrLeiter	String	

Mitarbeiter		
PersNr	string	KEY
Name	string	
Vorname	string	
Extern	bool	
Adresse	string	
TNr	string	
Urlaubsanspruch	int	
Resturlaub	int	
Sonderurlaub	int	
Institut	string	

Urlaubsliste		
PersNr	string	KEY
Anfang	date	KEY
Ende	date	
genommen	bool	
Sonderurlaub	bool	
AnzahlTage	int	

```

SELECT  m2.PersNr, m2.Name
        Urlaubsliste.Anfang
        Urlaubsliste.Ende
        Urlaubsliste.AnzahlTage
FROM    Institut,
        Mitarbeiter m1,
        Mitarbeiter m2,
        Urlaubsliste
WHERE   m1.Name = "Schmidt"
        AND m1.PersNr = Institut.PersNrLeiter
        AND Institut.Bezeichnung = m2.Institut
        AND Urlaubsliste.PersNr = m2.PersNr
        AND Urlaubsliste.Sonderurlaub = "FALSE"
        AND Urlaubsliste.genommen = "TRUE";

```

⊗ Welcher Mitarbeiter leitet einen Kurs, an dem der Leiter seines Instituts teilnimmt?

INSTITUT		
Bezeichnung	String	KEY
Adresse	String	
PersNrLeiter	String	

Mitarbeiter		
PersNr	string	KEY
Name	string	
Vorname	string	
Extern	bool	
Adresse	string	
TNr	string	
Urlaubsanspruch	int	
Resturlaub	int	
Sonderurlaub	int	
Institut	string	

Kursteilnahme		
KursNr	string	KEY
PersNr	string	KEY

Kurs		
KursNr	string	KEY
Titel	string	
Inhalt	string	
max Teilnehmer	int	
Endtermin	date	
Leitung	string	

```

SELECT  m2.PersNr, m2.Name
FROM    Mitarbeiter m1,
        Mitarbeiter m2,
        Institut, Kurs,
        Kursteilnahme
WHERE   m2.Institut = Institut.Bezeichnung
        AND Institut.PersNrLeiter = m1.PersNr
        AND Kursteilnahme.PersNr =
        AND Kursteilnahme.KursNr = Kurs.KursNr
        AND Kurs.Leitung = m2.PersNr;

```

Sichtbarkeitsregeln und Attributnamenkonflikte

Lösung durch Variablennamen bzw. Aliasnamen

SELECT Projektionsliste

FROM T1 [AS] X1, T2 [AS] X2, ..., TN [AS] XN

WHERE Selektionsbedingung

Einfache und quantifizierte Selektionsbedingungen

Aufbau wie in Aussagenlogik über AND, OR, NOT

Vergleichsoperatoren <, >, >=, <=, <>, =

Spezielle Probleme beim Nullwerttest

Prädikat	x IS NULL	x IS NOT NULL	NOT x IS NULL	NOT x IS NOT NULL
x ist gleich dem Nullwert	true	false	false	true
x ist nicht der Nullwert	false	true	true	false
x=(x1,...,xn), kein xi ist gleich dem Nullwert	false	true	true	false
x=(x1,...,xn), mindestens ein xi, aber nicht alle sind gleich dem Nullwert	false	false	true	true
x=(x1,...,xn), alle Werte xi sind Nullwerte	true	false	false	true

BETWEEN ... AND ... zur Prüfung von Intervallzugehörigkeit

wie

a <= x AND x <= b

⊗ Gib die Summe und den Durchschnitt aller Resturlaubstage pro Institut aus.

Mitarbeiter		
PersNr	string	KEY
Name	string	
Vorname	string	
Extern	bool	
Adresse	string	
TNr	string	
Urlaubsanspruch	int	
Resturlaub	int	
Sonderurlaub	int	
Institut	string	

```
SELECT      Institut,  
            SUM (Resturlaub),  
            AVG (Resturlaub)  
FROM        Mitarbeiter  
GROUP BY   Institut;
```

⊠ Nenne alle Kurse, die keine anderen Kurse voraussetzen.

Kurs		
KursNr	string	KEY
Titel	string	
Inhalt	string	
max Teilnehmer	int	
Endtermin	date	
Leitung	string	

Voraussetzung		
KursNr	string	KEY
VoraussNr	string	KEY

```
SELECT KursNr, Titel
FROM Kurs k
WHERE NOT EXISTS
    (SELECT *
     FROM Voraussetzung
     WHERE Voraussetzung.KursNr = k.KursNr);
```

Weitere SQL-Anweisungen

SQL kann nur Existenz von Tupeln prüfen, deshalb für Aussagen P
über Mengen M Umweg über Umwandlung
 $(\forall x \in M : P(x)) \Leftrightarrow (\neg \exists x \in M : \neg P(x))$

*Finde alle Studententupel, bei denen der Fremdschlüssel auf einen
nicht existierenden Fachbereich verweist.*

Inkorrekt:

```
SELECT * FROM Student s
      WHERE FORALL Fachbereich fb
             fb.nummer <> s.fachbereich
```

Korrekt:

```
SELECT * FROM Student s
      WHERE NOT EXISTS
             (SELECT * FROM Fachbereiche fb
              WHERE fb.nummer = s.fachbereich)
```

Bei EXISTS wird Menge konstruiert, deshalb nicht nur über ein Attribut.

Bestimme alle die Studenten, so daß es für alle Vorlesungen eine Belegung durch diese Studenten gibt.

```
SELECT * FROM Student s  
WHERE NOT EXISTS
```

```
(SELECT * FROM Vorlesung v  
WHERE NOT EXISTS
```

```
(SELECT * FROM Belegung b  
WHERE  
b.Student = s.Nummer AND  
b.Veranstaltung = v.Nummer ))
```

Verbindung von Subqueries durch **SOME/ALL-Operatoren**

=SOME

=ALL

<SOME

<ALL

nur anwendbar für einspaltige Subqueries

x =SOME q : es gibt einen Wert y in der Query q mit x = y

x <SOME q : es gibt einen Wert y in der Query q mit x < y

x =ALL q : für alle Werte y in der Query q mit x = y

x <ALL q : für alle Werte y in der Query q mit x < y

ist q leer, dann gilt die Aussage

=SOME ist äquivalent zum IN-Operator; statt SOME kann auch ANY

Finde alle Veranstaltungen, die nur von Studenten des veranstaltenden Fachbereiches belegt worden sind.

```
SELECT v.Titel, v.Nummer
FROM Veranstaltung v, Lehrkräfte l
WHERE (v.Veranstalter = l.Nummer) AND
      (l.FBNummer =ALL
        (SELECT s.FBNummer
         FROM Belegung b , Student s
         WHERE (s.Nummer = b.Student) AND
              (b.Veranstaltung = v.Nummer))
```

SOME entspricht einer einstufigen Existenzquantifizierung auf Attributvergleichsebene

```
SELECT *  
    FROM Rel1 r1  
    WHERE r1.attr1 =ANY (SELECT r2.attr2 FROM Rel2 r2);
```

Ein Tupel aus Rel1 qualifiziert sich anhand des Attributes attr1 , wenn es in Rel2 ein Tupel gibt, das in attr2 den gleichen Wert hat.

\$ALL entspricht einer einstufigen Allquantifizierung auf der Attributvergleichsebene:

```
SELECT *  
    FROM Rel1 r1  
    WHERE r1.attr1 =ALL (SELECT r2.attr2 FROM Rel2 r2 );
```

Ein Tupel aus Rel1 qualifiziert sich anhand des Attributs attr1 , wenn in Rel2 alle Tupel in attr2 den gleichen Wert haben.

Geschachtelte Quantifizierungen sind i.a. mit SOME/ALL nur extrem umständlich ausdrückbar, wenn dies überhaupt möglich ist.

Operationen des Relationenkalküls in SQL

Projektion	SELECT DISTINCT ... FROM R
Selektion	SELECT * FROM R WHERE Φ
Vereinigung	SELECT DISTINCT ... UNION SELECT DISTINCT ...
Differenz	SELECT DISTINCT * FROM R EXCEPT SELECT DISTINCT * FROM S
Verbund	SELECT * FROM R NATURAL JOIN S
Kartesisches Produkt	SELECT * FROM R CROSS JOIN S
Theta-Verbund	SELECT * FROM R JOIN S ON R.B=S.C
Äußerer Verbund	SELECT * FROM R LEFT OUTER JOIN S ON ... auch RIGHT OUTER JOIN, FULL OUTER JOIN

Anmerkung: Verbund, Theta-Verbund, äußerer Verbund sind SQL92-Intermediates!

Bestimme alle Namen von Professoren, die Datenbanken lesen.

$\Pi_{\text{PERSON.Name}} (\sigma_{\text{VORLES.Kurs}='Datenbanken'} (\text{PERSON} \bowtie_{\text{PNum}} \text{PROFESSOR} \bowtie_{\text{PNum}} \text{VORLES}))$

```
SELECT DISTINCT Name
FROM PERSON, PROFESSOR, VORLES
WHERE PERSON.PNum = PROFESSOR.PNum
AND PERSON.PNum = VORLES.PNum
AND VORLES.Kurs = 'Datenbanken'
```

PERSON = ({Name, Adresse, PNum}, {PNum}, \emptyset)

STUDENT = ({PNum, SNum, Hauptf, Nebenf, Betreuer}, {SNum}, {{PNum} \bowtie {SNum}})

PROFESSOR = ({PNum, Spezialis}, {PNum}, \emptyset)

VORLES = ({Kurs, Raum, Zeit, Semester, Lesender.Pnum}, {Raum, Zeit, Semester}, ...)

TEILNAHME = ({Kurs, Semester, Lesender.PNum, SNum, Note}, {SNum, Kurs, Semester}, ...)

Bestimme für die Datenbankprofessoren ihre Vorlesungen.

$(\sigma_{\text{Spezialis}='Datenbanken'} (\text{VORLES} \bowtie_{\text{PNum}} \text{PROFESSOR}))$

```
SELECT DISTINCT *
FROM VORLES, PROFESSOR
WHERE PROFESSOR.Spezialis = 'Datenbanken'
AND PROFESSOR.PNum = VORLESUNG.PNum
GROUP BY PROFESSOR.PNum
```

Bestimme alle Namen von Professoren, die selbst eine Vorlesung Datenbanken hörten und eine solche Vorlesung lesen.

Π PERSON. Name (σ VORLES.Kurs='Datenbanken'
 (VORLES_{Lesender.PNum} \bowtie PROFESSOR_{PNum} \bowtie PERSON_{PNum}
 (STUDENT_{PNum} \bowtie TEILNAHME_{SNum,Kurs})))
 SELECT DISTINCT y.Name
 FROM PERSON y, PROFESSOR p, VORLES v, TEILNAHME t, STUDENT s
 WHERE y.PNum = p.PNum
 AND y.PNum = v.PNum
 AND v.Kurs = 'Datenbanken'
 AND t.SNum = s.SNum
 AND s.PNum = y.PNum
 AND t.Kurs = 'Datenbanken'

Wird zusätzlich die Semantik benutzt (nur Studenten nehmen an Vorlesungen teil, nur Professoren lesen

Vorlesungen), dann kann man die Anfrage vereinfachen:

Π PERSON. Name (σ VORLES.Kurs='Datenbanken'
 (VORLES_{Lesender.PNum} \bowtie PERSON_{PNum} \bowtie STUDENT_{PNum}
 (STUDENT_{PNum} \bowtie TEILNAHME_{SNum,Kurs})))
 FROM PERSON y, VORLES v, TEILNAHME t, STUDENT s
 WHERE y.PNum = v.PNum
 AND v.Kurs = 'Datenbanken'
 AND t.SNum = s.SNum
 AND s.PNum = y.PNum AND t.Kurs = 'Datenbanken'

Name, Adresse aller Studenten ohne Fehlleistung.

```
SELECT Name, Adresse
FROM PERSON, STUDENT
WHERE PERSON.PNum = STUDENT.PNum
AND NOT EXISTS (SELECT *
FROM TEILNAHME
WHERE TEILNAHME.SNum = STUDENT.SNum
AND TEILNAHME.Note = 5)
ORDER BY Name)
```

Dazu ist folgende Anfrage äquivalent:

```
SELECT Name, Adresse
FROM PERSON
WHERE PNum IN (SELECT PNum
FROM STUDENT
WHERE NOT EXISTS (SELECT *
FROM TEILNAHME
WHERE
TEILNAHME.SNum = STUDENBT.SNum
AND TEILNAHME.Note = 5))
```


Embedded SQL und Wirtssprachen (z.B. C, PL/1, COBOL, ADA,...)

Kein Aufruf von C-Funktionen, -Prozeduren aus SQL

Werteübermittlung durch Variable:

```
EXEC SQL BEGIN DECLARE SECTION;
```

```
char *name;
```

C-Variable, die DBMS liest

```
int semester;
```

evt. Typkonvertierungen

```
int nummer;
```

```
EXEC SQL END DECLARE SECTION;
```

```
EXEC SQL SELECT name, semester
```

SQL-Attribute

```
INTO :name, :semester
```

Hostvariable

```
FROM Student
```

```
WHERE nummer = :nummer;
```

Tabellendeklaration in Embedded SQL (C-Compiler kann dann statisch prüfen)

```
EXEC SQL DECLARE Student TABLE
```

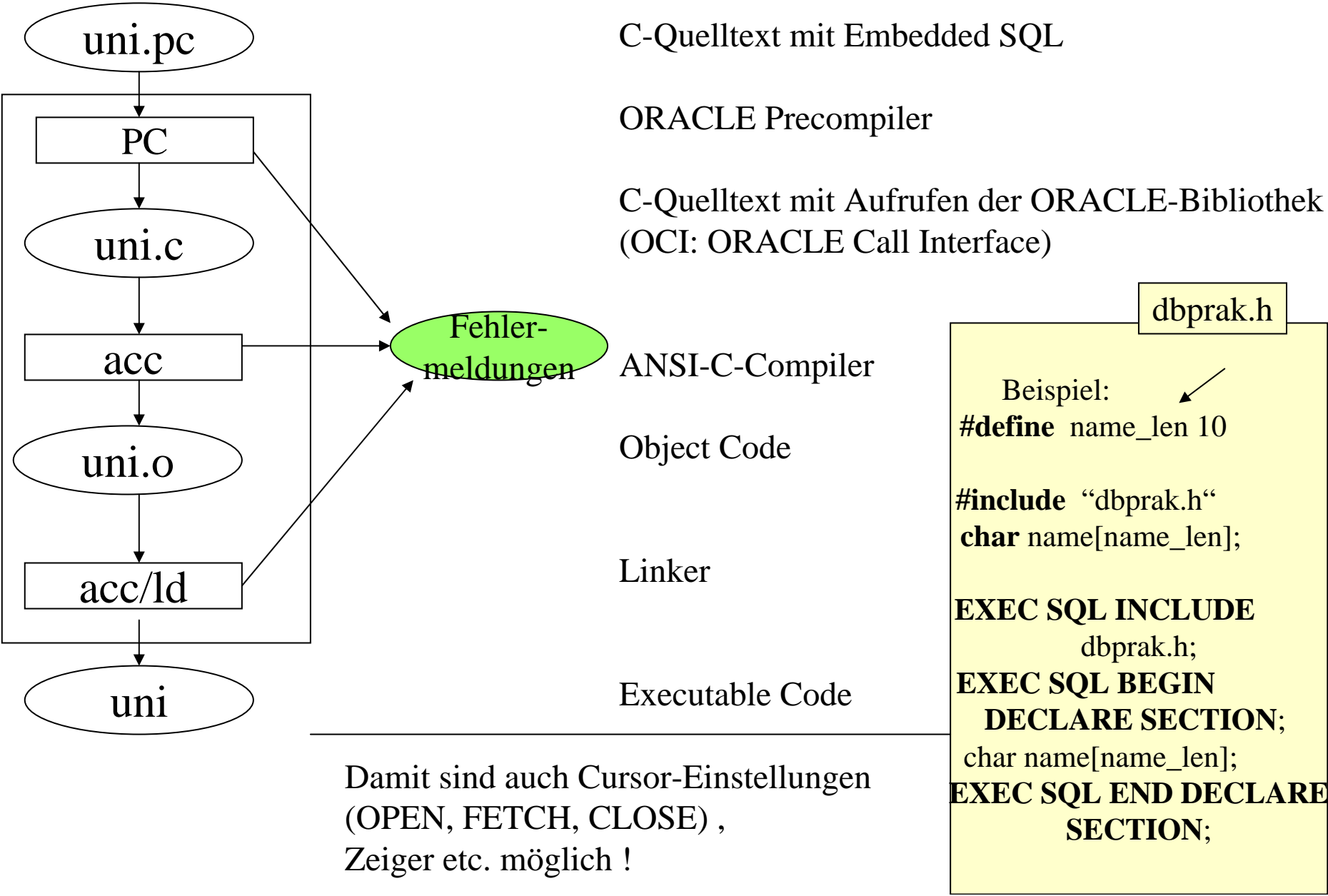
```
(nummer NUMBER(5) NOT NULL,
```

```
name VARCHAR(20) NOT NULL,
```

```
semester NUMBER(2));
```

Fehlerbehandlung mit WHENEVER

Compile und Link-Zyklus am Beispiel von ORACLE



Anmerkungen:

- SQL erlaubt i.a. Multimengen (ohne DISTINCT)
- Tupelvariable sind Alias-Konstrukte (anstelle der Tabellennamen)
- Falls keine echte Projektion, dann *
- Kombination von Anfragen über INTERSECT | UNION | MINUS (EXCEPT)
- Genestete Anfragen durch Kopplung im WHERE-Teil
 - Kopplung über IN (Tupel-element der inneren Anfrage) oder
 - EXISTS (eine Menge ist nicht leer) oder
 - NOT EXISTS (eine Menge ist leer)
- Ordnung durch ORDER BY Attributliste
- Gruppierung durch GROUP BY Attributliste
- Arithmetische Aggregationsoperationen: AVG, SUM, MAX, MIN, COUNT
- Textverarbeitung durch Vergleichsoperationen z.B. LIKE '%Daten%'

Alternative Anfragesprache: **Query-By-Example QBE**

Werteorientierter Kalkül

Tabellengerüst, Variablen (evt. als Ausgabenvariable (P.), Insert (I.),

Delete (D.), Update (U.)), Vergleichsoperatoren

Zeilen in \forall -Interpretation

Query-By-Example QBE als Alternative

Generiere eine Namensliste der Vorlesungsteilnehmer für die Vorlesung(en) 'Datenbanken' im Wintersemester 1994.

PERSON	Name	Adresse	PNum
	P.x		y

TEILNEHMER	Kurs	Semester	Lesender.PNum	SNum	Note
	'Datenbanken'	'WS 94/95'		z	

STUDENT	PNum	SNum	Hauptf	Nebenf	Betreuer
	y	z			

Namen, Adresse aller Studenten mit mindestes einmal ohne Fehlleistung

PERSON	Name	Adresse	PNum
	P.	P.	y

TEILNEHMER	Kurs	Semester	Lesender.PNum	SNum	Note
				z	5

STUDENT	PNum	SNum	Hauptf	Nebenf	Betreuer
	y	z			

Spezifikation von Integritätsbedingungen in SQL

Beispiel: Implizite Integritätsbedingung

Mitarbeiter		
PersNr	string	KEY
Name	string	
Vorname	string	
Extern	bool	
Adresse	string	
TNr	string	
Urlaubsanspruch	int	
Resturlaub	int	
Sonderurlaub	int	
Institut	string	

INSTITUT		
Bezeichnung	String	KEY
Adresse	String	
PersNrLeiter	String	

$\forall i \in \text{Institut}:$
 $(\exists m \in \text{Mitarbeiter}:$
 $(i.\text{PersNrLeiter} = m.\text{PersNr}))$

Datenbanksystem: SYBASE

```
CREATE TRIGGER InstitutLeiter1
ON Institut FOR INSERT, UPDATE AS
SELECT COUNT (PersNrLeiter)
      FROM Institut
      WHERE PersNrLeiter NOT IN
            SELECT PersNr
            FROM Mitarbeiter
> O THEN ROLLBACK TRANSACTION;
```

```
CREATE TRIGGER InstitutLeiter2
ON Mitarbeiter FOR UPDATE, DELETE AS
SELECT COUNT (PersNrLeiter)
      FROM Institut
      WHERE PersNrLeiter NOT IN
            SELECT PersNr
            FROM Mitarbeiter
> O THEN ROLLBACK TRANSACTION;
```

Anwendung zur Definition von Sichten in SQL

Externe bzw. Sicherheitssichten

zusammengesetzt aus drei Einschränkungstypen

Strukturelle Einschränkungen: Teilmenge der Spalten (Projektion)

CREATE VIEW Sichtename(B1,...,BN) AS SELECT A1,...,AN FROM Tabelle

mit Umbenennung

Modifikationsoperationen auf Sicht werden übertragen, wobei Default- bzw.

Nullbelegung der restlichen Attribute (soweit erlaubt, sonst zurück)

Prädikative Einschränkungen: Teilmenge der Tupel (Selektion)

CREATE VIEW Sichtename AS SELECT * FROM Tabelle WHERE Prädikat

Modifikationsbefehle werden um das Prädikat expandiert

Vorsicht bei Einfügen, da damit auch weitere Tupel ‚durchgestellt‘ werden

CREATE VIEW ... AS SELECT ... WITH CHECK OPTION

evt. auch WITH CHECK OPTION [LOCAL | CASCADED]

Operationale Einschränkungen: Einschränkung der benutzbaren SQL-Operationen

Nicht änderbare Sichten bei: SELECT DISTINCT,

SELECT mit Ausdrücken oder gleicher Attributname mehrfach

Benutzung einer nicht änderbaren Sicht oder mehrere Tabellen (Joins)

Benutzung der Klausel GROUP BY oder HAVING

Explizite Vergabe von Rechten GRANT, REVOKE

Generierung von Sichten

Mitarbeiter		
PersNr	string	KEY
Name	string	
Vorname	string	
Extern	bool	
Adresse	string	
TNr	string	
Urlaubsanspruch	int	
Resturlaub	int	
Sonderurlaub	int	
Institut	string	



MitarbeiterOhneUrlaub		
PersNr	string	KEY
Name	string	
Vorname	string	
Extern	bool	
Adresse	string	
TNr	string	
Institut	string	

```
CREATE VIEW MitarbeiterOhneUrlaub
    (PersNr, Name, Vorname,
     ExterneM, Adresse, TNr, Institut)
AS SELECT PersNr, Name, Vorname, Extern,
    Adresse, TNr, Institut
FROM Mitarbeiter;
```

Mitarbeiter		
PersNr	string	KEY
Name	string	
Vorname	string	
Extern	bool	
Adresse	string	
TNr	string	
Urlaubsanspruch	int	
Resturlaub	int	
Sonderurlaub	int	
Institut	string	



ExterneMitarbeiter	
Name	string
Vorname	string
(Extern	TRUE)
Adresse	string
TNr	string

```

CREATE VIEW externeMitarbeiter
      (Name, Vorname, Adresse, TNr,)
AS SELECT Name, Vorname, Adresse, TNr,
      FROM Mitarbeiter;
WHERE Externe = „TRUE“;

```

Mitarbeiter		
PersNr	string	KEY
Name	string	
Vorname	string	
Extern	bool	
Adresse	string	
TNr	string	
Urlaubsanspruch	int	
Resturlaub	int	
Sonderurlaub	int	
Institut	string	



AgrarTechnikUrlaub	
Name	string
Vorname	string
(Extern	FALSE)
TNr	string
Resturlaub	int
(Institut	„...AgrarTechnik“)

```

CREATE VIEW AgrarTechnikUrlaub
    (Name, Vorname, TNr, Resturlaub)
AS SELECT Name, Vorname, TNr, Resturlaub
    FROM Mitarbeiter;
WHERE Extern = „FALSE“;
AND Institut =
    „Grundlagen der Agrartechnik“
ORDER BY Resturlaub;

```

Institutsleiter:

```
SELECT * FROM AgrarTechnikUrlaub
```

Mitarbeiter UserId		
PersNr	string	KEY
UserId	string	

```
CREATE VIEW indivMitarbeiter
(PersNr, Name, Vorname, Extern,
Adresse, TNr,Urlaubsanspruch,
Resturlaub, Sonderurlaub, Institut)
```

Mitarbeiter		
PersNr	string	KEY
Name	string	
Vorname	string	
Extern	bool	
Adresse	string	
TNr	string	
Urlaubsanspruch	int	
Resturlaub	int	
Sonderurlaub	int	
Institut	string	

```
AS SELECT *
FROM Mitarbeiter;
WHERE PersNr =
(SELECT PersNr
FROM MitarbeiterUserId
WHERE UserId = USER);
```

Anwendung von Anfrage für Update Operationen

```
INSERT INTO PERSON VALUES('Gerste', 'Zentrum', 0815)
```

auch mit Tabellen

```
INSERT INTO TEILNAHME SELECT x. Kurs, x.Semester, x.PNum, y.SNum, z.Note
FROM VORLESUNG x, STUDENT y, TEILNAHME t,
      PERSON, TEILNAHME z
WHERE x.Kurs LIKE '%Geschichte%' AND PERSON.PNum = y.PNum
      AND PERSON.Name LIKE 'B%' AND z.SNum = y.SNum
      AND NOT EXISTS (SELECT*
      FROM TEILNAHME w
      WHERE w.SNum = z.SNum AND w.Note < z.Note
```