

Datenbanken I (T2INF2004)

Foliensatz 7: SQL 3/3 (komplexere SQL Statements)

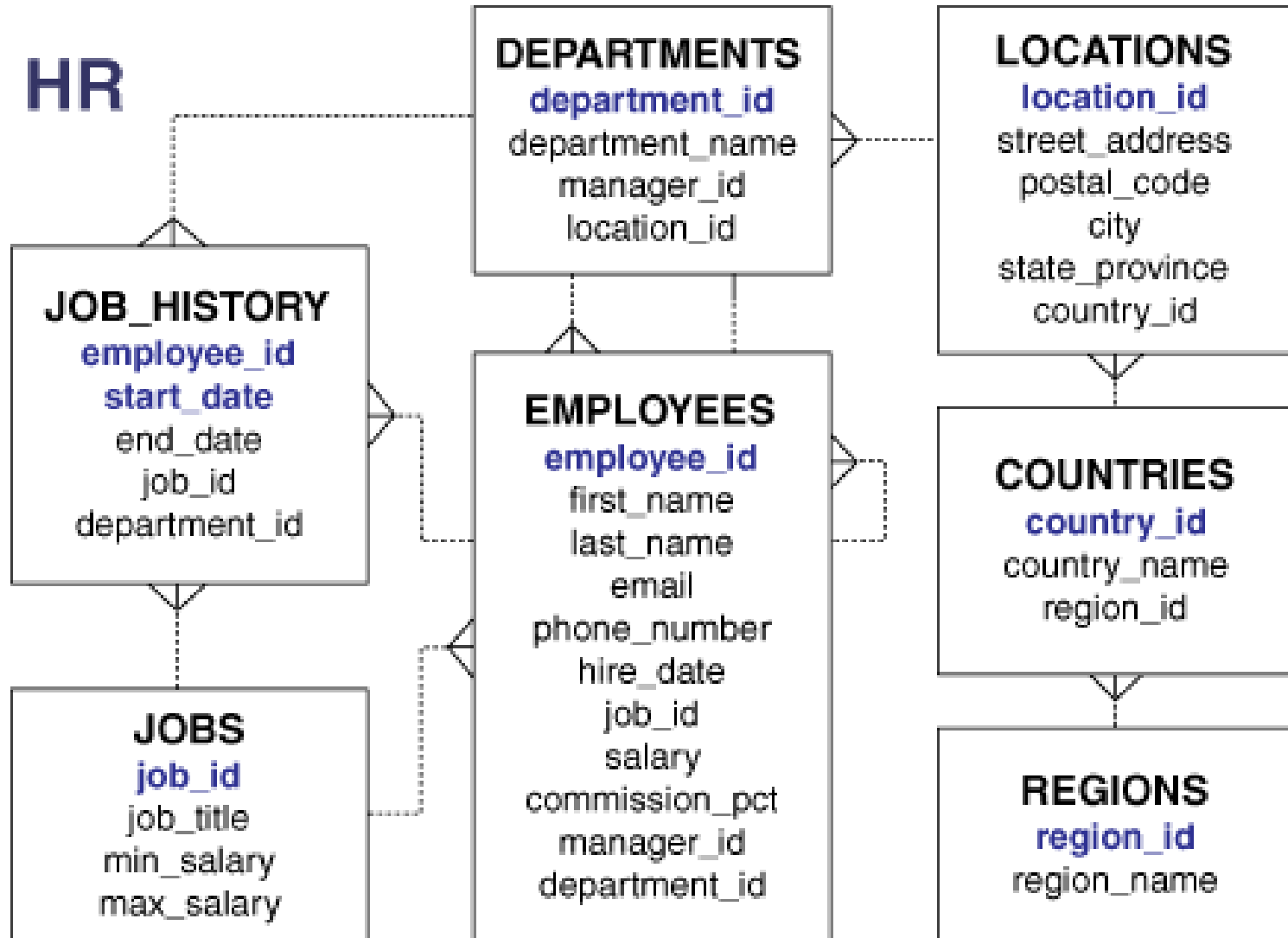
Uli Seelbach, DHBW Mannheim, 2023

Bildquellen in diesem Foliensatz:

- **Oracle-Dokumentation**

Das Oracle HR-Schema

Übersicht





SQL

Komplexität von Abfragen

- Bisherige Abfragen beschränkten sich auf „Basismittel“ von SQL
 - SELECT: Einfache Spaltenauswahl
 - FROM: Angabe von Tabellen und Join-Ausdrücken
 - WHERE: Aufzählung simpler Bedingungen mit Operatoren, z.B. AND, verknüpft
- GROUP BY und HAVING wurde eingesetzt
- Komplexere Abfragen werden mit der bisherigen Vorgehensweise (1 Aufgabenstellung = 1 Statement mit der uns bekannten Syntax) problematisch... weil...:
 - Erfahrung zum Bauen eines „Mega-Statements“ einfach fehlt
 - Wenn wir es doch schaffen: Unübersichtlich
 - Ggf. schwer wartbar, da Teile nicht wiederverwendet werden
 - Ggf. sogar langsam, da theoretisch korrekt aber nicht performant umgesetzt
 - „Debugging“? Fehlanzeige.



SQL

Anforderungen an komplexe Abfragen

- Zusätzliche Sprachkonstrukte, um Probleme intuitiver zu lösen
- Zusätzliche Sprachkonstrukte, um Probleme performanter zu lösen
- Strukturierung / Kapselung
 - Aufspaltung in Teilprobleme
 - Eventuell Wiederverwenden bereits umgesetzter Teilprobleme in anderen Abfragen
 - Entwicklertests und Quellcodedokumentation verbessern
- Einsatz von Kommentaren
 - „-- Einzeiliger Kommentar“
 - „/* Mehrzeiliger Kommentar */“

Professoren			
PERSNR	NAME	RANG	RAUM
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten		
MATRNR	NAME	SEMESTER
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Assistenten			
PERSNR	NAME	FACHGEBIET	BOSS
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

Vorlesungen			
VORLNR	TITEL	SWS	GELESEN_VON
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

hören	
MATRNR	VORLNR
25403	5022
26120	5001
27550	4052
27550	5001
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5001
29555	5022

voraussetzen	
VORGÄNGER	NACHFOLGER
5001	5041
5001	5043
5001	5049
5041	5052
5041	5216
5043	5052
5052	5259

prüfen			
MATRNR	VORLNR	PERSNR	NOTE
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Komplexe Abfragen mit SQL

Beispiel

- Welche von C4-Professoren (aber nur die mit Assistenten) gelesene Vorlesung hat überdurchschnittlich viele SWS?

```
SELECT v.titel vorlesungstitel, v.sws
FROM professoren p
JOIN vorlesungen v on p.persnr = v.gelesen_von
JOIN assistenten a on p.persnr = a.boss
WHERE p.rang = 'C4'
GROUP BY v.titel, v.sws
HAVING v.sws > avg(v.sws)
```





Komplexe Abfragen mit SQL

Beispiel, mögliche Lösung

- Welche von C4-Professoren (aber nur die mit Assistenten) gelesene Vorlesung hat überdurchschnittlich viele SWS?

```
SELECT v.titel vorlesungstitel
FROM professoren p
JOIN vorlesungen v on p.persnr = v.gelesen_von
JOIN assistenten a on p.persnr = a.boss
CROSS JOIN vorlesungen v_alle
WHERE p.rang = 'C4'
GROUP BY v.titel, v.sws
HAVING v.sws > avg(v_alle.sws)
```



komplexe Abfragen mit SQL

Beispiel, mögliche Herangehensweise

- Welche von C4-Professoren (aber nur die mit Assistenten) gelesene Vorlesung hat überdurchschnittlich viele SWS?
- Lassen Sie uns das Problem in mehrere Teilprobleme zerlegen:
 - A: Welche C4-Professoren haben Assistenten?
 - B: Wie hoch ist die durchschnittliche Anzahl SWS?
 - C: Welche Vorlesungen erfüllen die Kriterien, dass sie von einem Prof aus A gelesen werden und mehr als B SWS haben?
- Wie man das umsetzt, hängt in der Praxis von mehreren Faktoren ab
 - Erfahrung des Entwicklers
 - Werden Teilprobleme mehrmals benötigt?
 - Performance-Tuning
 - Vom DBMS zur Verfügung gestellte Sprachkonstrukte
 - Zeit
 - ...



Komplexe Abfragen mit SQL

Werkzeugkasten

- Unterabfragen (Subquery)
 - Korrelierte und unkorrelierte
 - Single Row
 - Multi Row
 - Mengenverarbeitende Sprachkonstrukte
- Views
- Common table expressions / subquery factoring (SQL WITH-Clause)
- (Analytische Funktionen) [Analytische Funktionen - erfolgreich eingesetzt](#)
- (User defined functions könnten hier ggf. auch genannt werden, werden aber erst in einer späteren Vorlesung näher erläutert)
- (OLAP in SQL3: Werkzeuge zur Vereinfachung komplexer GROUP-BY-Funktionen und Bereitstellung zusätzlicher Aggregationstupel (ROLLUP, CUBE, GROUPING SETS) für Berichte manchmal relevant, aber Bestandteil von Datenbanken II)



Unterabfragen

Einleitung

- Unterabfragen sind uns schon begegnet, ohne dass wir es als solche wahrgenommen haben:

- Mengenoperationen

```
SELECT * FROM a
UNION
SELECT * FROM b
```

- Views

```
SELECT * FROM a
INNER JOIN some_view b on a.id=b.id
```



Unterabfragen

Einleitung

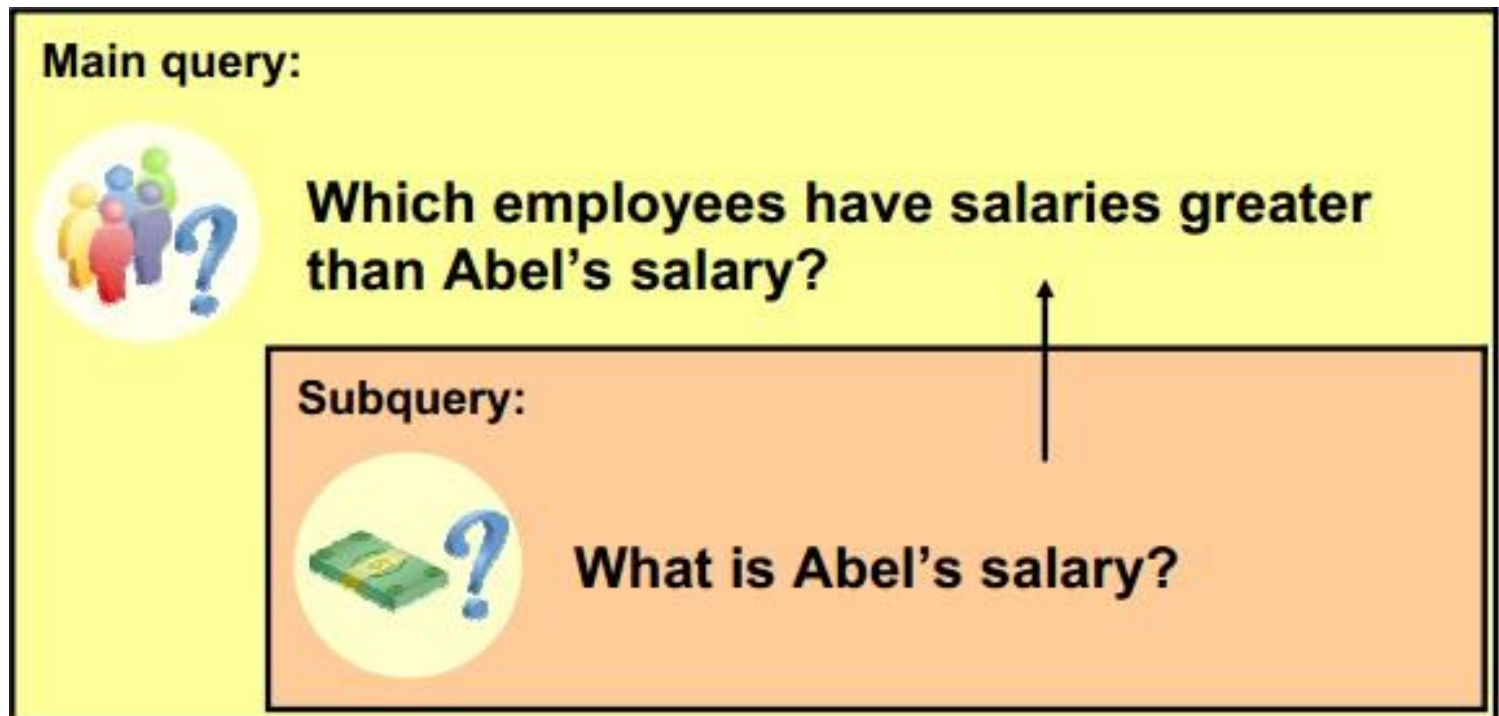
- Stellen Sie sich folgende Fragestellung vor:
 - Wer bekommt mehr Gehalt als „Abel“?
- Naiver Ansatz mit Joins

```
SELECT e.LAST_NAME
FROM employees e_abel
JOIN employees e
ON e.SALARY > e_abel.SALARY
WHERE e_abel.last_name = 'Abel'
```

Unterabfragen

Einleitung

- Stellen Sie sich folgende Fragestellung vor:
 - Wer bekommt mehr Gehalt als „Abel“?
- Etwas strukturierteres Herangehen mit explizit definierten Teilproblemen:





Unterabfragen

Funktionsweise

- Stellen Sie sich folgende Fragestellung vor:
 - Wer bekommt mehr Gehalt als „Abel“?
- Lösung mit Unterabfrage:

```
SELECT LAST_NAME
FROM EMPLOYEES
WHERE SALARY > (SELECT SALARY                11000
                 FROM EMPLOYEES
                 WHERE last_name = 'Abel')
```

- Die in Klammern gesetzte Unterabfrage könnte (da 1x1-Matrix) hinter last_name sogar in die Liste projizierter Spalten mit aufgenommen werden



Unterabfragen

Syntax

- Unterabfragen können daraufhin kategorisiert werden, was sie als Ergebnis zurückliefern → dies grenzt das Einsatzgebiet selbiger ein:
 - 1x1-Matrix (prinzipiell überall verwendbar)
 - Matrix generell (entweder als Tabelle oder im Rahmen von Vergleichen)

```
SELECT  select_list
FROM    table
WHERE   expr operator
        (SELECT      select_list
         FROM         table);
```

```
SELECT  select_list
FROM    (SELECT      select_list
         FROM         table)
WHERE   expr
```

```
SELECT  (SELECT      column
         FROM         table)
FROM    table
WHERE   expr
```



Unterabfragen

Scalar subquery vs. table subquery

A *scalarSubquery*, sometimes called an expression subquery, is a subquery that evaluates to a single row with a single column.

You can place a *scalarSubquery* anywhere an *expression* is permitted. A *scalarSubquery* turns a *selectExpression* result into a scalar value because it returns only a single row and column value.

A *tableSubquery* is a subquery that returns multiple rows. Unlike a *scalarSubquery*, a *tableSubquery* is allowed only:

- As a *tableExpression* in a **FROM clause**
- With EXISTS, IN, or quantified comparisons

When used as a *tableExpression* in a **FROM clause**, or with EXISTS, it can return multiple columns.

When used with IN or quantified comparisons, it must return a single column.

Unterabfragen

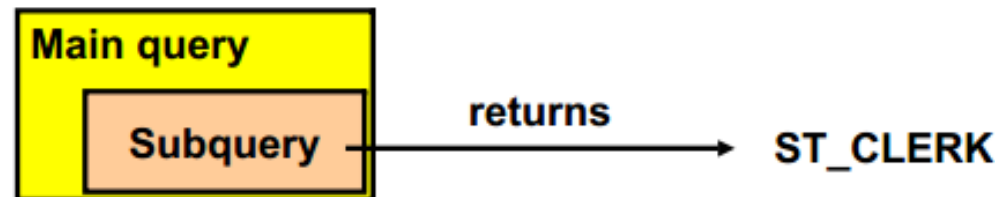
Arten

- Das vorherige Beispiel war ein Single-Row Subquery (“scalar subquery”)
 - Resultat bestand aus einer Zeile
 - Vergleichsoperator „>“ war hier also möglich
- Multi-Row Subqueries liefern mehrere Zeilen zurück und können dementsprechend nur mit Operatoren verwendet werden, die auf Mengen ausgerichtet sind

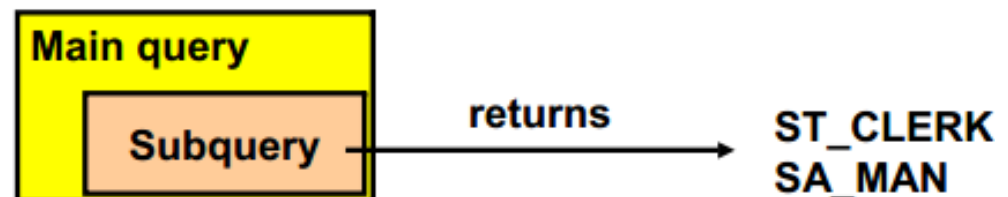


Warum macht ein „>“ keinen Sinn, wenn in der Unterabfrage 2 oder mehr Zeilen sind?

Single-row subquery



Multiple-row subquery





Unterabfragen

Operatoren für Single-Row Subqueries

- Die bereits bekannten:

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to



Unterabfragen

Operatoren für Multi-Row Subqueries

- Ein bekannter und 3 neue:

Any

- WHERE $x > \text{ANY}(\text{subquery})$
- X muss größer als irgendein Element der Unterabfrage sein
- Gleichbedeutend: WHERE $x > \text{SOME}(\text{subquery})$

All

- WHERE $x > \text{ALL}(\text{subquery})$
- X muss größer als jedes Element der Unterabfrage sein

In

- WHERE $x \text{ IN } (\text{subquery})$
- X muss Element (i.S.v. „Datensatz“) der Unterabfrage sein
- Möglich ist bspw. auch „WHERE (a,b,c) in ((1,2,3),(2,3,4),(3,7,1))“

Exists

- WHERE EXISTS (subquery)
- Wahr, wenn subquery mindestens eine Zeile zurückliefert



Unterabfragen

Operator ANY

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ANY
      (SELECT salary
       FROM   employees
       WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
124	Mourgos	ST_MAN	5800
141	Rajs	ST_CLERK	3600
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

■ ■ ■

10 rows selected.



Unterabfragen

Operator ALL

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ALL
      (SELECT salary
       FROM employees
       WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG';
```

9000, 6000, 4200

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500



Unterabfragen

Operator IN

- Bereits bekannt mit expliziter Angabe
- Alle sind identisch:

```
SELECT last_name from employees  
WHERE employee_id = ANY(100,102);
```

```
SELECT last_name from employees  
WHERE employee_id IN (100, 102);
```

```
SELECT last_name from employees  
WHERE employee_id IN (SELECT 100 FROM dual UNION  
                      SELECT 102 FROM dual);
```

```
SELECT last_name, EMPLOYEE_ID from employees  
WHERE employee_id IN (VALUES (100), (102));
```



Unterabfragen

Erstellen Sie folgende SQL-Abfrage für das HR-Schema

- Selektieren Sie alle Mitarbeiter, die keine Führungskräfte sind (deren EMPLOYEE_ID also nicht die MANAGER_ID irgendeines Mitarbeiters ist)



Unterabfragen

Achtung bei NULL-Werten! Besonders bei “NOT IN”

- NULL bedeutet „unbekannt“

```
SELECT *  
  FROM EMPLOYEES  
 WHERE EMPLOYEE_ID NOT IN (SELECT MANAGER_ID  
                           FROM EMPLOYEES);
```





Unterabfragen

Erstellen Sie folgende SQL-Abfrage für das HR-Schema

- Selektieren Sie alle Mitarbeiter, die keine Führungskräfte sind

```
SELECT *  
  FROM EMPLOYEES  
 WHERE EMPLOYEE_ID NOT IN (SELECT MANAGER_ID  
                           FROM EMPLOYEES  
                           WHERE MANAGER_ID IS NOT NULL);
```

```
SELECT *  
  FROM EMPLOYEES  
EXCEPT  
SELECT *  
  FROM EMPLOYEES  
 WHERE EMPLOYEE_ID = SOME (SELECT MANAGER_ID FROM EMPLOYEES);
```

```
SELECT *  
  FROM EMPLOYEES  
 WHERE EMPLOYEE_ID NOT IN (SELECT COALESCE(MANAGER_ID, -1)  
                           FROM EMPLOYEES);
```




Unterabfragen

Operator EXISTS

```
SELECT c.country_id, c.country_name
  FROM countries c
 WHERE NOT EXISTS (SELECT *
                   FROM locations l
                   WHERE l.country_id = c.country_id);
```

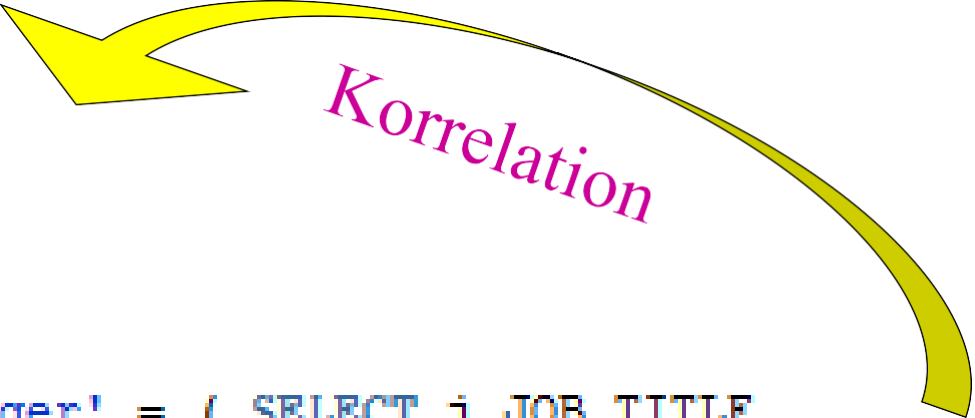
```
INSERT INTO hr.countries
(country_id, country_name, region_id)
SELECT *
FROM (VALUES ('DE', 'Germany', 1)) T
WHERE NOT EXISTS (SELECT * FROM hr.countries c
WHERE c.country_id = 'DE');
```

Unterabfragen

Arten

- Standardmäßig wird eine Unterabfrage vor der äußeren Abfrage ausgewertet und ausgeführt, das geht aber nicht immer
- Neben der Anzahl zurück gelieferter Zeilen kann eine Kategorisierung auch noch auf Basis der Korrelation zur äußeren Abfrage vorgenommen werden:
 - Unkorrelierte Unterabfragen sind unabhängig von der äußeren Abfrage, wie im Beispiel von Abel und werden nur 1x ausgewertet/ausgeführt
 - Korrelierte Unterabfragen basieren auf Attribute der äußeren Abfrage, dadurch meist sehr imperformant, da sie für jedes Tupel ausgeführt werden müssen:

```
SELECT
  e.LAST_NAME
FROM
  employees e
WHERE
  'Sales Manager' = ( SELECT j.JOB_TITLE
                     FROM jobs j
                     WHERE j.JOB_ID=e.job_id);
```



Korrelation



Unterabfragen

Arten (vgl. Ausführungsplan)

- Korrelierte Unterabfragen lassen sich häufig in nichtkorrelierte Unterabfragen oder in Joins umwandeln.
- Suche im HR-Schema nach dem Kollegen, der am wenigsten verdient:
 - Ziemlich schlechte Performance:

```
SELECT e.employee_id, e.salary
FROM employees e
WHERE NOT EXISTS ( SELECT 1
                    FROM employees e_sub
                    WHERE e_sub.salary < e.salary)
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		2	24	8 (25)	00:00:01
1	MERGE JOIN ANTI		2	24	8 (25)	00:00:01
2	SORT JOIN		107	856	4 (25)	00:00:01
3	TABLE ACCESS FULL	EMPLOYEES	107	856	3 (0)	00:00:01
* 4	SORT UNIQUE		107	428	4 (25)	00:00:01
5	TABLE ACCESS FULL	EMPLOYEES	107	428	3 (0)	00:00:01



Unterabfragen

Arten (vgl. Ausführungsplan)

- Korrelierte Unterabfragen lassen sich häufig in nichtkorrelierte Unterabfragen oder in Joins umwandeln.
- Suche im HR-Schema nach dem Kollegen, der am wenigsten verdient:
 - Bessere Performance ohne korrelierte Unterabfrage

```
SELECT e.employee_id, e.salary
FROM employees e
WHERE e.salary = (SELECT min(e_sub.salary) FROM employees)
```

28

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		2	16	6 (0)	00:00:01
* 1	TABLE ACCESS FULL	EMPLOYEES	2	16	3 (0)	00:00:01
2	SORT AGGREGATE		1	4		
3	TABLE ACCESS FULL	EMPLOYEES	107	428	3 (0)	00:00:01



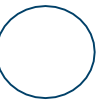
Unterabfragen

Inline-Views

- Anstelle der Angabe einer Tabelle/View kann auch eine Unterabfrage verwendet werden:
- Welche Regionen (RegionID, Name, AnzLaender) haben mehr als 5 Länder?

```
SELECT *  
  FROM (SELECT region_id    regionid  
           , region_name  name  
           , COUNT(*)     anzlaender  
         FROM hr.regions r  
        NATURAL JOIN hr.countries c  
        GROUP BY region_id  
           , r.region_name) T  
WHERE anzlaender > 5
```

Achtung: Derby
braucht zwingend
einen "correlation
name"



Unterabfragen

Einschub: Erweiterung HR-Schema (siehe courses.sql in moodle)

```
CREATE TABLE courses(
  course_id    varchar(6) PRIMARY KEY,
  course_name  varchar(50) NOT NULL,
  course_topic varchar(50) NOT NULL
);

INSERT INTO courses VALUES
  ('ORA_1' , 'Oracle Basics', 'Oracle' )
, ('ORCL' , 'Oracle Administration', 'Oracle' )
, ('ORA_2' , 'Oracle Advanced', 'Oracle' )
, ('MS_XL1', 'MS Excel Basics', 'MS Office' )
, ('MS_WD1', 'MS Word Basics', 'MS Office' )
, ('HAPPY' , 'How to Find Happiness in Life', 'Others')
;

CREATE TABLE employee_education (
  employee_id DECIMAL(6) REFERENCES employees,
  course_id   VARCHAR(6) REFERENCES courses,
  PRIMARY KEY (employee_id,course_id)
);

INSERT INTO employee_education
SELECT e.employee_id, c.course_id
  FROM employees e
 CROSS JOIN courses c
 ORDER BY RANDOM()
  FETCH FIRST 200 ROWS only
;
```



Unterabfragen

Inline-Views

- Anstelle der Angabe einer Tabelle/View kann auch eine Unterabfrage verwendet werden:
- Wieviel % aller Mitarbeiter nahmen an bereits stattgefundenen Schulungen teil?

```
SELECT k.course_id
      , k.anzprokurs / double(g.gesamtanz) anteil
FROM (SELECT ee.course_id, COUNT(*) AS anzprokurs
      FROM employee_education ee
      GROUP BY ee.course_id) k
      , (SELECT COUNT(*) gesamtananz FROM employees) g;
```



Unterabfragen

Unsere Ausgangsfragestellung

Welche von C4-Professoren (aber nur die mit Assistenten) gelesene Vorlesung hat überdurchschnittlich viele SWS?

```
SELECT v.titel vorlesungstitel
FROM professoren p
INNER JOIN vorlesungen v
    on p.persnr = v.gelesen_von
WHERE p.rang = 'C4'
    and p.PERSNR in (select boss from assistenten)
    and v.sws > (select avg(sws)
                from vorlesungen)
```




Unterabfragen

Übung

- Welche Angestellte (Vorname+Nachname) sind Vorgesetzte? Vermeiden Sie Joins
- Jedes Department (Name) hat ein maximales Einkommen. Ermitteln Sie eine solche Liste und erweitern Sie diese Abfrage, um nur noch den Durchschnitt der Maximaleinkommen aller Departments anzuzeigen.
- Welche „Shipping Clerk“s haben ein Einkommen, das höher ist als jedes der Einkommen von „Purchasing Clerk“s? Lösen Sie die Aufgabe auf 2 Weisen, je mit einer single-row und einer multi-row-subquery!
- Welche „Sales Representative“s haben ein Einkommen, das höher ist als jedes der Einkommen von allen erst später angestellten „Accountant“s? Lösen dies mit einer korrelierten Unterabfrage beliebiger Art.
- Welche Mitarbeiter (ID) nahmen sowohl an der Schulung ‚How to Find Happiness in Life‘ als auch an der Schulung ‚MS Excel Basics ‘ teil?



Unterabfragen

Lösung (1)

- Welche Angestellte (Vorname+Nachname) sind Vorgesetzte? Vermeiden Sie Joins

```
SELECT e.FIRST_NAME, e.LAST_NAME
FROM EMPLOYEES e
WHERE e.EMPLOYEE_ID IN (
    SELECT MANAGER_ID
    FROM EMPLOYEES
)
```

- Jedes Department (Name) hat ein maximales Einkommen. Ermitteln Sie eine solche Liste und erweitern Sie diese Abfrage, um nur noch den Durchschnitt der Maximaleinkommen aller Departments anzuzeigen.

```
SELECT AVG(max_sal)
FROM
(
    SELECT d.DEPARTMENT_NAME, MAX(e.SALARY) max_sal
    FROM employees e
    INNER JOIN DEPARTMENTS d ON d.DEPARTMENT_ID=e.DEPARTMENT_ID
    GROUP BY d.DEPARTMENT_NAME
)
```



Unterabfragen

Lösung (2)

- Welche „Shipping Clerk“s haben ein Einkommen, das höher ist als jedes der Einkommen von „Purchasing Clerk“s? Lösen Sie die Aufgabe auf 2 Weisen, je mit einer single-row und einer multi-row-subquery!

```
SELECT e.EMPLOYEE_ID
FROM employees e NATURAL JOIN jobs j
WHERE j.JOB_TITLE='Shipping Clerk'
AND e.SALARY > ALL (SELECT ex.SALARY
                    FROM employees ex NATURAL JOIN jobs jx
                    WHERE jx.JOB_TITLE='Purchasing Clerk');
```

```
SELECT e.EMPLOYEE_ID
FROM employees e NATURAL JOIN jobs j
WHERE j.JOB_TITLE='Shipping Clerk'
AND e.SALARY > (SELECT max(ex.SALARY)
                FROM employees ex NATURAL JOIN jobs jx
                WHERE jx.JOB_TITLE='Purchasing Clerk');
```



Unterabfragen

Lösung (3)

- Welche „Sales Representative“s haben ein Einkommen, das höher ist als jedes der Einkommen von allen erst später angestellten „Accountant“s? Lösen dies mit einer korrelierten Unterabfrage beliebiger Art.

```
SELECT e.EMPLOYEE_ID
FROM employees e NATURAL JOIN jobs j
WHERE j.JOB_TITLE='Sales Representative'
AND e.SALARY > (SELECT max(ex.SALARY)
                FROM employees ex NATURAL JOIN jobs jx
                WHERE jx.JOB_TITLE='Accountant'
                AND e.HIRE_DATE<ex.HIRE_DATE);
```

- Welche Mitarbeiter (ID) nahmen sowohl an der Schulung ‚How to Find Happiness in Life‘ als auch an der Schulung ‚MS Excel Basics ‘ teil?

```
SELECT ee.employee_id FROM
employee_education ee
JOIN courses c ON c.course_id = ee.course_id
AND c.course_name = 'How to Find Happiness in Life'
```

INTERSECT

```
SELECT ee.employee_id FROM
employee_education ee
WHERE ee.course_id = (SELECT course_id FROM courses c
                     WHERE c.course_name = 'MS Excel Basics')
```



Unterabfragen

Beispiel – „Allquantor“?

- Eine Art **Existenzquantor** gibt es in SQL (wir erinnern uns):

```
SELECT l.location_id
```

```
FROM locations l
```

```
WHERE NOT EXISTS (SELECT *
```

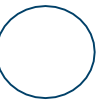
```
FROM departments d
```

```
WHERE d.location_id = l.location_id)
```

- Ein **Allquantor** ist jedoch nicht vorhanden...

- Schauen Sie sich das HR-Schema an:

Welche Mitarbeiter haben alle Schulungen zum Thema „Oracle“ besucht?

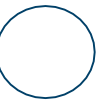


Unterabfragen

Divisionsoperator aus der rel. Algebra, leider nicht SQL

Welche Mitarbeiter haben alle Schulungen zum Thema „Oracle“ besucht?

```
-- Pseudo-SQL
SELECT e.employee_id
      FROM EMPLOYEE_EDUCATION e
DIVIDE COURSES c
      USING (COURSE_ID)
WHERE c.course_topic = 'Oracle'
```



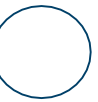
Unterabfragen

Lösung – Divisionsoperator aus der rel. Algebra

$$R \div S := \pi_{R'}(R) - \pi_{R'}((\pi_{R'}(R) \times S) - R)$$

```
SELECT DISTINCT EMPLOYEE_ID
  FROM EMPLOYEE_EDUCATION
EXCEPT
SELECT EMPLOYEE_ID
  FROM (SELECT EMPLOYEE_ID, COURSE_ID
        FROM (SELECT DISTINCT EMPLOYEE_ID
              FROM EMPLOYEE_EDUCATION) e
        CROSS JOIN courses
        WHERE course_topic = 'Oracle'
      )
EXCEPT
SELECT EMPLOYEE_ID, COURSE_ID
  FROM EMPLOYEE_EDUCATION
) T;
```





Unterabfragen

Lösung – „Allquantor“?

/ mit einigen Umformungen aus der Aussagenlogik kommt man auf solch eine „absolut intuitive“ Form: */*

```
SELECT e.employee_id
FROM employees e
WHERE NOT EXISTS (SELECT 1
                  FROM courses c
                  WHERE NOT EXISTS (SELECT 1
                                    FROM employee_education ee
                                    WHERE ee.employee_id = e.employee_id
                                          AND ee.course_id = c.course_id)
                  AND c.course_topic = 'Oracle')
```

$$\forall x \in R: P(x) \Leftrightarrow \neg (\exists x \in R: \neg P(x))$$





Unterabfragen

Lösung – „Allquantor“?

/* Allquantifizierung auch immer mit COUNT(*) möglich! */

```
SELECT employee_id
  FROM employee_education ee
NATURAL JOIN courses c
  WHERE c.course_topic = 'Oracle'
  GROUP BY employee_id
  HAVING COUNT(distinct c.course_id) = (SELECT COUNT(*)
                                         FROM courses
                                         WHERE course_topic = 'Oracle');
```



Views

Allgemeines

- Bereits in letzter Vorlesung eingeführt
- Gespeicherte, wiederverwendbare Abfragen
- Vorteile
 - Wiederverwendbar
 - Gewährleistet gut Datenunabhängigkeit
 - Materialisiert bieten sie ggf. signifikante Performancegewinne
- Nachteile
 - Anlegen benötigt Rechte
 - Anlegen benötigt Ressourcen (Dokumentation → Zeit ...)
 - Pflege kann sehr aufwendig werden



Views

Syntax, Wiederholung

- Können nach dem Anlegen (CREATE VIEW name_of_view AS ...) in Abfragen wie normale Tabellen verwendet werden
- Bsp.:

```
SELECT id  
FROM personen_mit_e_als_Anfangsbuchstabe_v  
WHERE alter > 20
```



Views

Unsere Ausgangsfragestellung

- Welche von C4-Professoren (aber nur die mit Assistenten) gelesene Vorlesung hat überdurchschnittlich viele SWS?

```
SELECT v.titel vorlesungstitel
FROM professoren p
INNER JOIN vorlesungen v on p.persnr=v.gelesenvon
INNER JOIN professoren_mit_assistenten_v pa
      ON pa.profId=p.persNr
WHERE p.rang='C4'
AND v.sws > (select avg_sws from stat_vorlesungen_v)
```



Common table expressions (CTEs)

Motivation

- Anlegen einer View eventuell nicht zielführend
 - Ggf. gar nicht möglich
 - View würde ohnehin nur in einer Abfrage benötigt werden
 - View ist so einfach, dass es sich nicht lohnt
 - Es wird gewünscht, dass in der Abfrage zu sehen ist, was „hinter“ einem SQL-Statement steckt
- Verwenden von Subqueries eventuell nicht zielführend
 - Unübersichtlich bei sehr großen Abfragen
 - Ggf. Redundanz und somit schlechte Wartbarkeit
- Mittel der Wahl:
 - Common table expressions (MS SQL, PostgreSQL, ...)
 - subquery factoring (so nennt es Oracle)
 - In SQL3 ('99) eingeführt, Erfahrung aus der Praxis zeigt jedoch: aktuell noch selten verwendet, da fast nicht bekannt



Common table expressions (CTEs)

Syntax (einfach gehalten)

WITH

```
subquery1_name AS (subquery1_select)
[, subquery2_name AS (subquery2_select)]
...
[, subqueryN_name AS (subqueryN_select)]
```

SELECT ... FROM ... ;



Common table expressions (CTEs)

Motivation, Beispiel

Mit Views, 2 DDL-Kommandos:

- `CREATE VIEW view_x AS (SELECT ... FROM tab_a WHERE ...);`
`CREATE VIEW view_y AS (SELECT ... FROM view_x WHERE ...);`
`SELECT * FROM view_y;`

Mit Unterabfragen, alles in einer Abfrage:

- `SELECT * FROM (SELECT ... FROM (SELECT ... FROM tab_a WHERE ...) WHERE ...);`

Mit CTEs, alles in einer Abfrage:

- `WITH CTE_x AS (SELECT ... FROM tab_a WHERE ...),`
`CTE_y AS (SELECT ... FROM CTE_x WHERE ...)`
`SELECT * FROM CTE_y;`



Common table expressions (CTEs)

Beispiel

- Welche Abteilungen sind überdurchschnittlich teuer?

WITH dept_costs AS

(SELECT department_name, SUM(salary) dept_total

FROM employees e, departments d

WHERE e.department_id = d.department_id

GROUP BY department_name)

, avg_cost AS

(SELECT SUM(dept_total)/COUNT(*) AVG

FROM dept_costs)

SELECT *

FROM dept_costs

WHERE dept_total > (SELECT AVG FROM avg_cost)

ORDER BY department_name;

DEPARTMENT_NAME	DEPT_TOTAL
Sales	304500
Shipping	156400



Common table expressions (CTEs)

Anmerkungen

- Bessere Performance macht sich erst bei komplexeren Abfragen bemerkbar
 - Bei simpleren Abfragen sind CTEs zwar ggf. weniger performant, aber dafür besser les- und wartbar
- Wird gern eingesetzt für rekursive Abfragen
 - War auch einer der Hauptgründe für die Aufnahme in den Standard
 - Ist die einzige (standardisierte und performante) Möglichkeit, Rekursion in Abfragen zu gewährleisten, wenn es mit Views nicht geht und das DBS keine sonstigen Mittel anbietet (wie Oracles CONNECT BY-Clause)
- Nicht in jedem DBS verfügbar, da nicht jedes dem SQL-Standard folgt, seit 2018 aber fast flächendeckend unterstützt



Common table expressions (CTEs)

Unsere Ausgangsfragestellung

- Welche von C4-Professoren (aber nur die mit Assistenten) gelesene Vorlesung hat überdurchschnittlich viele SWS?

WITH

avg_sws AS (SELECT AVG(sws) avg_sws FROM vorlesungen)

, bosse AS (SELECT boss FROM assistenten)

SELECT v.titel vorlesungstitel

FROM professoren p

INNER JOIN vorlesungen v ON p.persnr=v.gelesenvon

WHERE p.rang ='C4'

AND p.PERSNR IN (SELECT boss FROM bosse)

⁴⁶ AND v.sws > (SELECT avg_sws FROM avg_sws)



Common table expressions (CTEs)

Übung

- Welche in 1997 eingestellten Mitarbeiter (ID) erhalten mehr Gehalt als der Median aller bis inkl. 1997 eingestellten Kollegen? Blenden Sie die Ergebnisse von Departments innerhalb der Region „Europe“ aus.
- Jedes Department (Name) hat ein maximales Einkommen. Ermitteln Sie eine solche Liste und erweitern Sie diese Abfrage, um nur noch den Durchschnitt der Maximaleinkommen aller Departments anzuzeigen.



Common table expressions (CTEs)

Lösung (1)

- Welche in 1997 eingestellten Mitarbeiter (ID) erhalten mehr Gehalt als der Median aller bis inkl. 1997 eingestellten Kollegen? Blenden Sie die Ergebnisse von Departments innerhalb der Region „Europe“ aus.

```
WITH median_vor_1998 AS
  (SELECT median(e.salary) m
   FROM employees e
   WHERE e.hire_date < '01.01.1998')
SELECT median_vor_1998.m median
      , e.employee_id
      , salary
FROM median_vor_1998
CROSS JOIN employees e
JOIN departments d ON e.department_id = d.department_id
JOIN locations l ON l.location_id = d.location_id
JOIN countries c ON c.country_id = l.country_id
JOIN regions r ON r.region_id = c.region_id
WHERE e.hire_date BETWEEN '01.01.1997' AND '31.12.1997'
      AND r.region_name != 'Europe'
      AND e.salary > median_vor_1998.m
```



Common table expressions (CTEs)

Lösung (1)

- Jedes Department (Name) hat ein maximales Einkommen. Ermitteln Sie eine solche Liste und erweitern Sie diese Abfrage, um nur noch den Durchschnitt der Maximaleinkommen aller Departments anzuzeigen.

```
WITH uebersicht AS
  (SELECT d.department_name
        ,MAX(e.salary) max_sal
    FROM employees e
    INNER JOIN departments d
        ON d.department_id = e.department_id
    GROUP BY d.department_name)
SELECT AVG(max_sal)
FROM uebersicht
```



ANALYTISCHE FUNKTIONEN

Motivation

- Folgendes Problem ist nun zu lösen: „Für jeden Mitarbeiter des HR-Schemas soll der Abt.-Name, sein Gehalt sowie die Gehaltsumme seiner Abteilung angezeigt werden“

DEPARTMENT_ID	DEPARTMENT_NAME	SALARY	SUM_SAL
10	Administration	4400	4400
20	Marketing	13000	19000
20	Marketing	6000	19000
30	Purchasing	11000	24900
30	Purchasing	3100	24900
30	Purchasing	2900	24900
30	Purchasing	2800	24900
30	Purchasing	2600	24900
30	Purchasing	2500	24900

?

Entwickeln Sie eine solche Abfrage!

Bearbeitungszeit:
5 Minuten



ANALYTISCHE FUNKTIONEN

Motivation (Lösung der letzten Frage)

- Durchaus umsetzbar, z. B. eine inline-View:

```
SELECT
    e.DEPARTMENT_ID, d.DEPARTMENT_NAME, e.SALARY, emp_gruppiert.sum_sal
FROM EMPLOYEES e
INNER JOIN DEPARTMENTS d ON e.DEPARTMENT_ID=d.DEPARTMENT_ID
INNER JOIN (SELECT
                emp_sub.DEPARTMENT_ID, SUM(emp_sub.SALARY) sum_sal
            FROM EMPLOYEES emp_sub
            GROUP BY emp_sub.DEPARTMENT_ID
        ) emp_gruppiert ON e.DEPARTMENT_ID=emp_gruppiert.DEPARTMENT_ID
ORDER BY e.DEPARTMENT_ID;
```

- Großer Nachteil: Die Tabelle emp wird hier 2x durchlaufen. 1x für die Anzeige jedes Mitarbeiters und 1x für die Generierung der inline-View
 - Hohe I/O-Kosten
 - Eigentlich unnötig



ANALYTISCHE FUNKTIONEN

Alternative Lösung mit analytischen Funktionen

- Abfrage sieht viel einfacher aus:

```
SELECT
    e.DEPARTMENT_ID, d.DEPARTMENT_NAME, e.SALARY,
    sum(e.salary) over (partition by e.DEPARTMENT_ID) sum_sal
FROM EMPLOYEES e
INNER JOIN DEPARTMENTS d ON e.DEPARTMENT_ID=d.DEPARTMENT_ID
ORDER BY e.DEPARTMENT_ID;
```

- Tabelle wird nun noch 1x durchlaufen
 - Beweis: siehe Execution-Plan (nicht prüfungsrelevant)



ANALYTISCHE FUNKTIONEN

Hintergrund

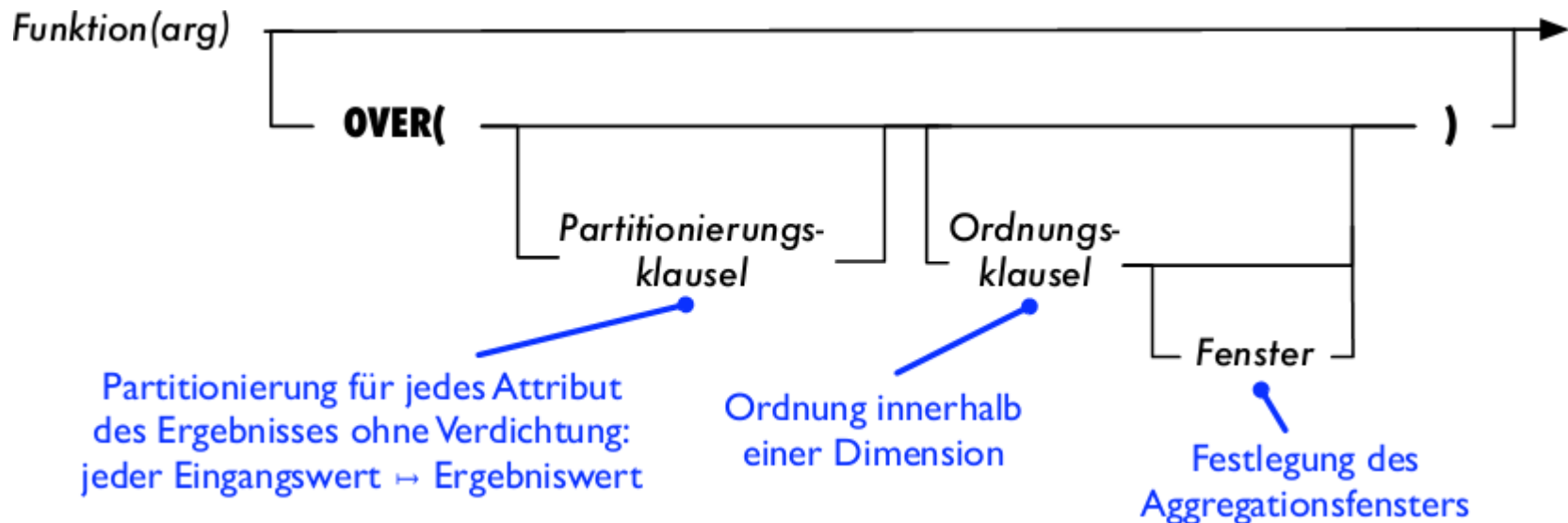
- Aggregatfunktionen lassen sich nur auf das kleinstmögliche Aggregat anwenden
- OLAP-Funktionen mit SQL3 eingeführt (neuester Standard)
 - Können Aggregationen bilden, ohne die Ergebnismenge weiter zu verdichten
 - Einsatz oft in DWHs
- Möglichkeiten zur
 - Aggregation, Kumulierung, Ranking
 - Attributlokale Partitionierung und auch Bildung dynamischer Fenster



Analytische Funktionen

Syntax

- Kernkonstrukt: OVER()





Analytische Funktionen

Beispiel über alle Tupel

R		
Jahr	Quartal	Umsatz
2004	1	10
2004	2	20
2004	3	10
2004	4	20
2005	1	30

```
SELECT Jahr, Quartal,  
       Umsatz,  
       SUM(Umsatz) OVER ()  
FROM R
```

R			
Jahr	Quartal	Umsatz	Gesamt- umsatz
2004	1	10	90
2004	2	20	90
2004	3	10	90
2004	4	20	90
2005	1	30	90



ANALYTISCHE FUNKTIONEN

Beispiel mit attributlokaler Partition

R		
Jahr	Quartal	Umsatz
2004	1	10
2004	2	20
2004	3	10
2004	4	20
2005	1	30

```
SELECT Jahr, Quartal,  
       Umsatz,  
       SUM(Umsatz) OVER (  
         PARTITION BY Jahr  
       )  
FROM R
```

R			
Jahr	Quartal	Umsatz	Jahresumsatz
2004	1	10	60
2004	2	20	60
2004	3	10	60
2004	4	20	60
2005	1	30	30



ANALYTISCHE FUNKTIONEN

Beispiel mit attributlokaler Partition mit

```
SELECT deptno
,   ename
,   hiredate
,   sal
,   COUNT(*) OVER (PARTITION BY deptno
                    ORDER BY sal
                    RANGE BETWEEN 1000 PRECEDING
                           AND 1000 FOLLOWING) count_sal
FROM emp
```

DEPTNO	ENAME	HIREDATE	SAL	COUNT_SAL
10	MILLER	23-JAN-82	1300	1
10	CLARK	09-JUN-81	2450	1
10	KING	17-NOV-81	5000	1
20	SMITH	17-DEC-80	800	2
20	ADAMS	12-JAN-83	1100	2
20	JONES	02-APR-81	2975	3
20	SCOTT	09-DEC-82	3000	3
20	FORD	03-DEC-81	3000	3
30	JAMES	03-DEC-81	950	5
30	WARD	22-FEB-81	1250	5
30	MARTIN	28-SEP-81	1250	5
30	TURNER	08-SEP-81	1500	5
30	ALLEN	20-FEB-81	1600	5
30	BLAKE	01-MAY-81	2850	1

Partition

Window

Current
Row



ANALYTISCHE FUNKTIONEN

Beispiel mit attributlokaler Partition mit

DEPTNO	ENAME	HIREDATE	SAL	COUNT_SAL
10	MILLER	23-JAN-82	1300	1
10	CLARK	09-JUN-81	2450	1
10	KING	17-NOV-81	5000	1
20	SMITH	17-DEC-80	800	2
20	ADAMS	12-JAN-83	1100	2
20	JONES	02-APR-81	2975	3
20	SCOTT	09-DEC-82	3000	3
20	FORD	03-DEC-81	3000	3
30	JAMES	03-DEC-81	950	5
30	WARD	22-FEB-81	1250	5
30	MARTIN	28-SEP-81	1250	5
30	BLAKE	01-MAY-81	2850	1

Partition

Window

Current Row



ANALYTISCHE FUNKTIONEN

attributlokale Partition mit Fenster

- Wenn wir ein ORDER BY verwenden, wird bei einigen Funktionen immer vom Anfang der Partition bis zum aktuellen Datensatz gezählt (kumuliert), z. B. bei SUM():

```
SELECT
```

```
    e.DEPARTMENT_ID, d.DEPARTMENT_NAME, e.SALARY,
```

```
    sum(e.salary) over (order by e.DEPARTMENT_ID, salary) sum_sal
```

```
FROM EMPLOYEES e
```

```
INNER JOIN DEPARTMENTS d ON e.DEPARTMENT_ID=d.DEPARTMENT_ID
```

```
ORDER BY e.DEPARTMENT_ID, salary;
```

DEPARTMENT_ID	DEPARTMENT_NAME	SALARY	SUM_SAL
10	Administration	4400	4400
20	Marketing	6000	10400
20	Marketing	13000	23400
30	Purchasing	2500	25900
30	Purchasing	2600	28500
30	Purchasing	2800	31300



ANALYTISCHE FUNKTIONEN

attributlokale Partition mit Fenster

- Einige Funktionen wie „RANK()“ oder „DENSE_RANK()“ benötigen zwingend eine Sortierung

SELECT

e.DEPARTMENT_ID, d.DEPARTMENT_NAME, e.SALARY,

RANK() over (partition by d.DEPARTMENT_ID order by -salary) rank_sal

FROM EMPLOYEES e

INNER JOIN DEPARTMENTS d ON e.DEPARTMENT_ID=d.DEPARTMENT_ID

ORDER BY e.DEPARTMENT_ID;

DEPARTMENT_ID	DEPARTMENT_NAME	SALARY	RANK_SAL
60	IT	9000	1
60	IT	4200	5
60	IT	4800	3
60	IT	4800	3
60	IT	6000	2
70	Public Relations	10000	1
80	Sales	10500	7

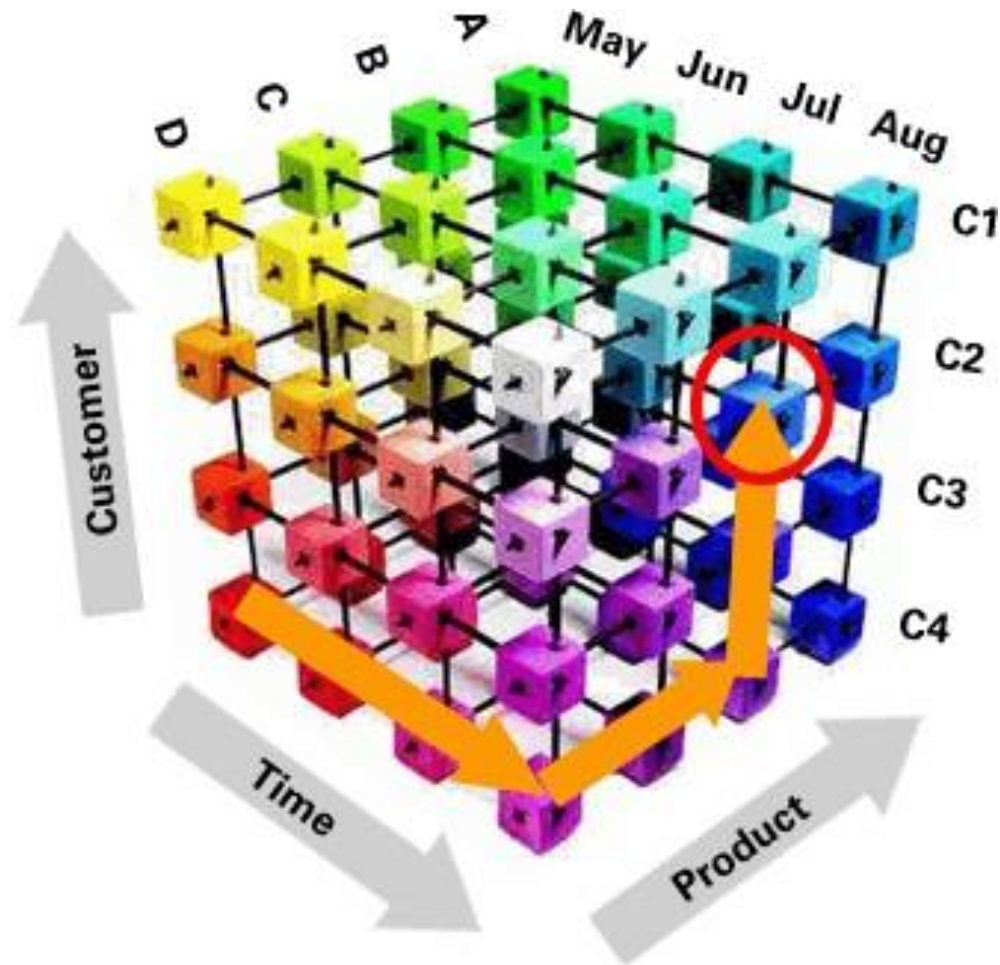
– Mehr unter

http://docs.oracle.com/cd/E11882_01/server.112/e26088/functions004.htm#S_QLRF51199



OLAP-Cubes

Motivation





OLAP-Cubes

Motivation

CHANNEL_DESC	CALENDAR	CO	SALES\$
Internet	2000-09	GB	16,569
Internet	2000-09	US	124,224
Internet	2000-09		140,793
Internet	2000-10	GB	14,539
Internet	2000-10	US	137,054
Internet	2000-10		151,593
Internet			292,387
Direct Sales	2000-09	GB	85,223
Direct Sales	2000-09	US	638,201
Direct Sales	2000-09		723,424
Direct Sales	2000-10	GB	91,925
Direct Sales	2000-10	US	682,297
Direct Sales	2000-10		774,222
Direct Sales			1,497,646
			1,790,032



OLAP-Cubes

Übung HR-Schema

	REGION_NAME	COUNTRY_NAME	GEHALTSSUMME
1	Europe	Germany	10000
2	Europe	United Kingdom	311000
3	Europe		321000
4	Americas	Canada	19000
5	Americas	United States of America	344416
6	Americas		363416
7			684416

- „NULL“ bedeutet hier „insgesamt“ und NICHT „fehlendes Land“ oder „fehlende Region“



OLAP-Cubes

Lösung HR-Schema

```
SELECT e.region_name
       ,e.country_name
       ,SUM(e.salary) gehaltssumme
FROM hr.emp_details_view e GROUP
BY ROLLUP(e.region_name,
          e.country_name);
```

VS

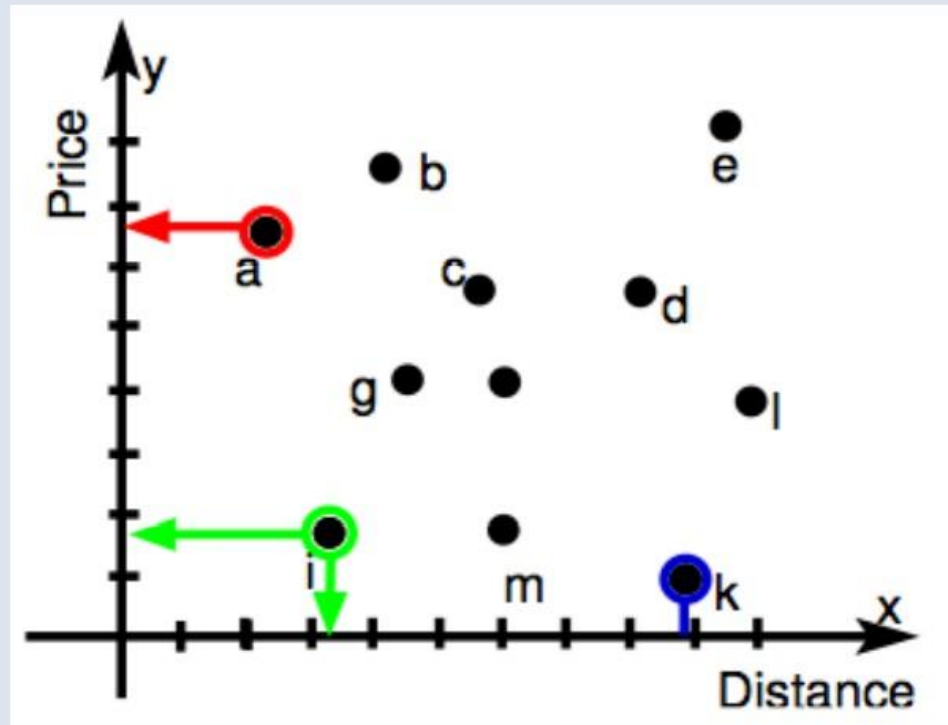
- SELECT e.region_name
 - ,e.country_name
 - ,SUM(e.salary) gehaltssummeFROM hr.emp_details_view e GROUP BY e.region_name
 - ,e.country_name
- UNION
- SELECT e.region_name
 - ,NULL
 - ,SUM(e.salary) gehaltssummeFROM hr.emp_details_view e GROUP BY e.region_name
- UNION
- SELECT NULL
 - ,NULL
 - ,SUM(e.salary) gehaltssummeFROM hr.emp_details_view e



Weitere Operatoren und Forschungsgebiete?

Beispiel: Pareto-Optimierung

- Example: Hotels near to the beach with a low price.



<http://www.slideshare.net>

[_ victormier/skyline-queries](#)

- The interesting points in the dataset are $\{a, i, k\}$. **a** has the least distance to the beach, **k** has the lowest price, **i** has neither the shortest distance nor the minimum price. **i** has a less distance value than **k** and a lower price value than **a**. **i** is hence not dominated by either one of them.



Weitere Operatoren und Forschungsgebiete?

Skyline- / Pareto-Efficiency-Operator

Wie kommt man an solch komplexe und neue Funktionen heran?

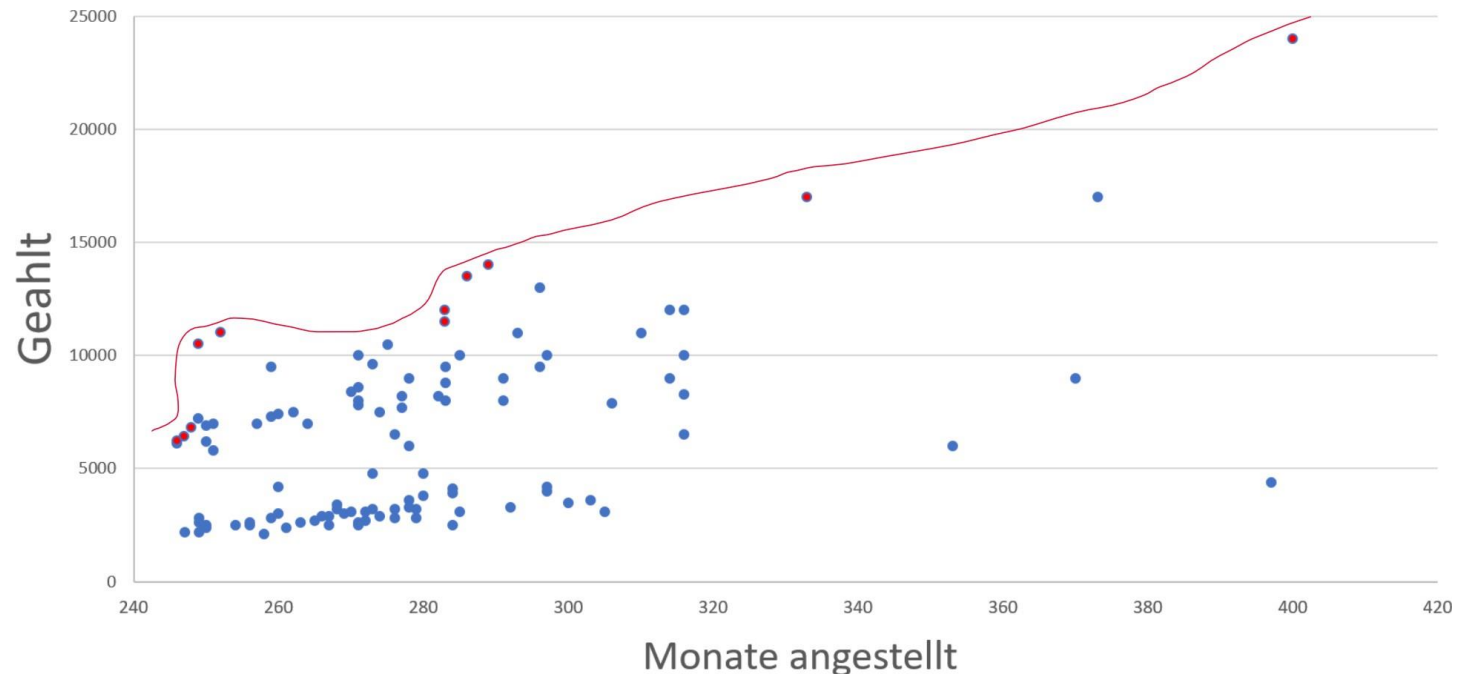
- On-Top of SQL: Preference SQL
 - <http://www.preferencesql.com/>
- SQL selbst erweitern mit der gewünschten Funktion
 - `SELECT * FROM Hotels`
`SKYLINE BY price MIN, distance MIN`
- Eine Funktion schreiben, die eine Tabelle von IDs zurückgibt
 - `SELECT * FROM Hotels WHERE HOTEL_ID in (`
`SELECT * FROM`
`TABLE(SKYLINE('Hotels','PRICE[MIN],DISTANCE[MIN]'))`
`)`
- Mehr zu Pareto-Optimierung
 - [Ausführliches Video:](https://www.youtube.com/watch?v=DDOsDqDFGvk)
<https://www.youtube.com/watch?v=DDOsDqDFGvk>



Weitere Operatoren und Forschungsgebiete?

Skyline- / Pareto-Efficiency-Operator - Übung

- Nicht relevant für die Klausur... aber: versuchen Sie aber trotzdem mit Ihrem Wissen folgende Aufgabe mit einer „gewöhnlichen“ SQL-Abfrage zu lösen:



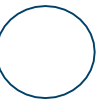
- 67 – Wer neu im Unternehmen ist und viel verdient, hat etwas richtig gemacht! Welche Mitarbeiter bilden die „Skyline“, werden also von keinem anderen Mitarbeiter bzgl. Betriebszugehörigkeit und Einkommen dominiert? (es sind 11)



Weitere Operatoren und Forschungsgebiete?

Skyline- / Pareto-Efficiency-Operator - Lösung

```
SELECT
    e.employee_id
    , round(months_between(sysdate,e.hire_date))
    , salary monate_angestellt
FROM
    employees e
WHERE NOT EXISTS (
    SELECT 1 FROM employees e_dominant
    WHERE    e_dominant.hire_date>=e.hire_date
            AND e_dominant.salary> e.salary
    OR      e_dominant.hire_date> e.hire_date
            AND e_dominant.salary>=e.salary
)
```

Weitere Operatoren und Forschungsgebiete?

... wo gibt es sonst noch Forschung?

- Im Bereich Data Mining
 - Sequenzanalysen / Mustererkennung in Zeitreihen
 - Mustererkennung in Bild-, Audio- und Videodaten
 - Spatio-temporal databases
 - ...
- „Datenstromsysteme“
 - Echtzeitverarbeitung von Daten
 - real time decision-support / -generation (denken Sie an Hochfrequenzhandel)
- Optimierung, Parallelisierung und Skalierung (wie immer)
- „weichere“ Themen wie „NoSQL-Datenbanken in der SWE“



Hausaufgaben

bis nach der „langen Pause“



@

Bearbeiten Sie das Aufgabenblatt

Datenbanken I - Praxis - HA SQL Fortgeschritten Aufgaben.pdf