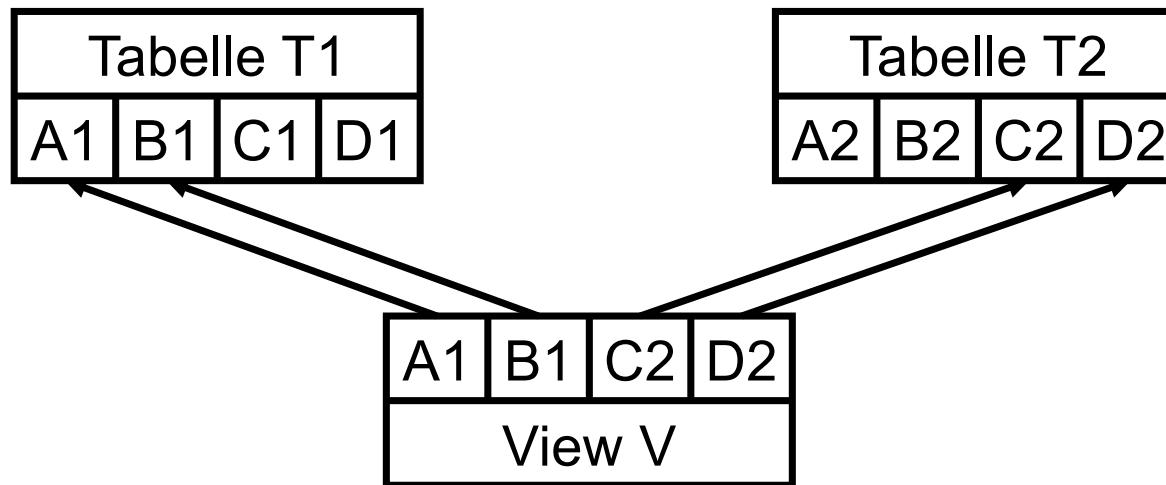


Übersicht

- SQL-Anfragen 2
 - Views
 - Indexierung
 - Large Objects
 - Join-Typen
 - Data Dictionary
 - Hierarchische Anfragen
 - Trigger

Views (Sichten, virtuelle Tabellen)

- Zusammenstellung von Daten einer gewünschten Sichtweise
 - Datenschutz
 - Ausblenden unwichtiger Informationen (Filter)
- Views enthalten keine Daten, sondern Zeiger



Views

Definition und Ausgabe

```
CREATE VIEW Gehalt_der_Chefs
AS
SELECT pnr, name, gehalt
FROM pers p1
WHERE EXISTS (SELECT *
               FROM pers p2
               WHERE p2.vnr=p1.pnr);
```

- Ausgabe

```
SELECT *
FROM Gehalt_der_Chefs;
```

Views

Definition

- Read only-Views
- Viewspalten müssen definiert werden bei
 - SQL-Funktionen
 - Arithmetischen Operationen

```
CREATE VIEW Gehaelter_Abteilungen(anr, abtGehalt)
AS
SELECT anr, SUM(gehalt)
FROM pers
GROUP BY anr
WITH READ ONLY;
```

- View löschen

```
DROP VIEW Gehaelter_Abteilungen;
```

Views

Eigenschaften

- View-Änderungen bewirken Änderungen in Originaltabellen
 - Aktualisierung mit Prüfoption (WITH CHECK OPTION)
- Änderungen nur möglich, wenn
 - Sicht bezieht sich auf eine einzige Tabelle
 - Spalten enthalten keine Funktionen oder arithm. Operationen
 - Keine Verwendung von GROUP-BY
 - Kein read-only definiert
- Vorteile
 - Lösung komplexer Probleme in Teilschritten
 - Kapselung von Logik
 - Übersichtlichkeit, Abstraktion
 - Zugriffsschutz

Views

Aktualisieren mit Prüfoption

- Einbringen eines Datensatzes in ein View, der dort nicht vorhanden sein sollte
- Der Datensatz kann nicht mehr ausgegeben werden

```
CREATE VIEW Frauen
AS
SELECT persnr, pname, geschlecht
  FROM persdat
 WHERE geschlecht = 'W';

INSERT INTO Frauen
  VALUES (45, 'Wedefeld', 'M');
```

Views

Aktualisieren mit Prüfoption

- Prüfoption zur Überprüfung, ob Datensatz in Basistabelle eingebracht wird
- Syntax:

WITH CHECK OPTION

```
CREATE VIEW Frauen
AS
SELECT persnr, pname, geschlecht
  FROM persdat
 WHERE geschlecht = 'W'
 WITH CHECK OPTION;
```

```
INSERT INTO Frauen
  VALUES (45, 'Wedefeld', 'M');
```

Views

Beispiele

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, aname, ort})

Pers							
<u>pnr</u>	name	jahrg	eindat	gehalt	beruf	<u>anr</u>	<u>vnr</u>
406	Coy	1950	01.03.86	80.000	Kaufmann	K55	123
123	Mueller	1958	01.09.80	68.000	Programmierer	K51	
829	Schmidt	1960	01.06.90	74.000	Kaufmann	K53	123
874	Abel		01.05.94	62.000	Softw.Entwickler	K55	829
503	Junghans	1975		55.000	Programmierer	K51	123
...

Abt		
<u>anr</u>	aname	ort
K51	Entwicklung	Erlangen
K53	Buchh	Nürnberg
K55	Personal	Nürnberg
...

Auf die Tabelle PERS soll eine Sicht erzeugt werden, die nur Programmierer mit weniger als 60.000 € Verdienst enthält.

Views

Beispiele

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, aname, ort})

Pers							
<u>pnr</u>	name	jahrg	eindat	gehalt	beruf	<u>anr</u>	<u>vnr</u>
406	Coy	1950	01.03.86	80.000	Kaufmann	K55	123
123	Mueller	1958	01.09.80	68.000	Programmierer	K51	
829	Schmidt	1960	01.06.90	74.000	Kaufmann	K53	123
874	Abel		01.05.94	62.000	Softw.Entwickler	K55	829
503	Junghans	1975		55.000	Programmierer	K51	123
...

Abt		
<u>anr</u>	aname	ort
K51	Entwicklung	Erlangen
K53	Buchh	Nürnberg
K55	Personal	Nürnberg
...

Lösche die Sicht ARME_PROGRAMMIERER.

Datenbank-Anfragen

Beispiel

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, aname, ort})

Pers							
<u>pnr</u>	name	jahrg	eindat	gehalt	beruf	<u>anr</u>	<u>vnr</u>
406	Coy	1950	01.03.86	80.000	Kaufmann	K55	123
123	Mueller	1958	01.09.80	68.000	Programmierer	K51	
829	Schmidt	1960	01.06.90	74.000	Kaufmann	K53	123
874	Abel		01.05.94	62.000	Softw.Entwickler	K55	829
503	Junghans	1975		55.000	Programmierer	K51	123
...

Abt		
<u>anr</u>	aname	ort
K51	Entwicklung	Erlangen
K53	Buchh	Nürnberg
K55	Personal	Nürnberg
...

Welcher Vorgesetzte hat die meisten Mitarbeiter?

Datenbank-Anfragen

Beispiel

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, aname, ort})

Pers							
<u>pnr</u>	name	jahrg	eindat	gehalt	beruf	<u>anr</u>	<u>vnr</u>
406	Coy	1950	01.03.86	80.000	Kaufmann	K55	123
123	Mueller	1958	01.09.80	68.000	Programmierer	K51	
829	Schmidt	1960	01.06.90	74.000	Kaufmann	K53	123
874	Abel		01.05.94	62.000	Softw.Entwickler	K55	829
503	Junghans	1975		55.000	Programmierer	K51	123
...

Abt		
<u>anr</u>	aname	ort
K51	Entwicklung	Erlangen
K53	Buchh	Nürnberg
K55	Personal	Nürnberg
...

Welcher Mitarbeiter liegt pro Abteilung beim Gehalt am nächsten am Durchschnittsgehalt in der jeweiligen Abteilung?

Result Limits

- Möglicher Lösungsweg der letzten Anfrage
 - Absteigende Sortierung der Ergebnistupel nach Anzahl Vorgesetzte
 - Abbruch nach erstem Element
- Oracle
 - ```
SELECT * FROM pers
 ORDER BY gehalt DESC
 FETCH FIRST ROW ONLY;
```
- Nachteile
  - Nur eine Ausgabe, auch wenn mehrere Tupel das Maximum erzielen
  - Unterschiedliche Implementierung in verschiedenen DBMS
- Realisierung in anderen Datenbanksystemen
  - MySQL: `SELECT * FROM pers LIMIT 5`
  - MS SQL Server: `SELECT TOP 5 * FROM pers`

# Lösung des Beispiels mit WITH-Klausel

- With-Klausel
  - View, der auf eine Anweisung begrenzt ist
  - Wird nicht dauerhaft gespeichert,
  - Wird direkt vor eigentlichem Select geschrieben
  - Auch mehrere Views können so definiert werden

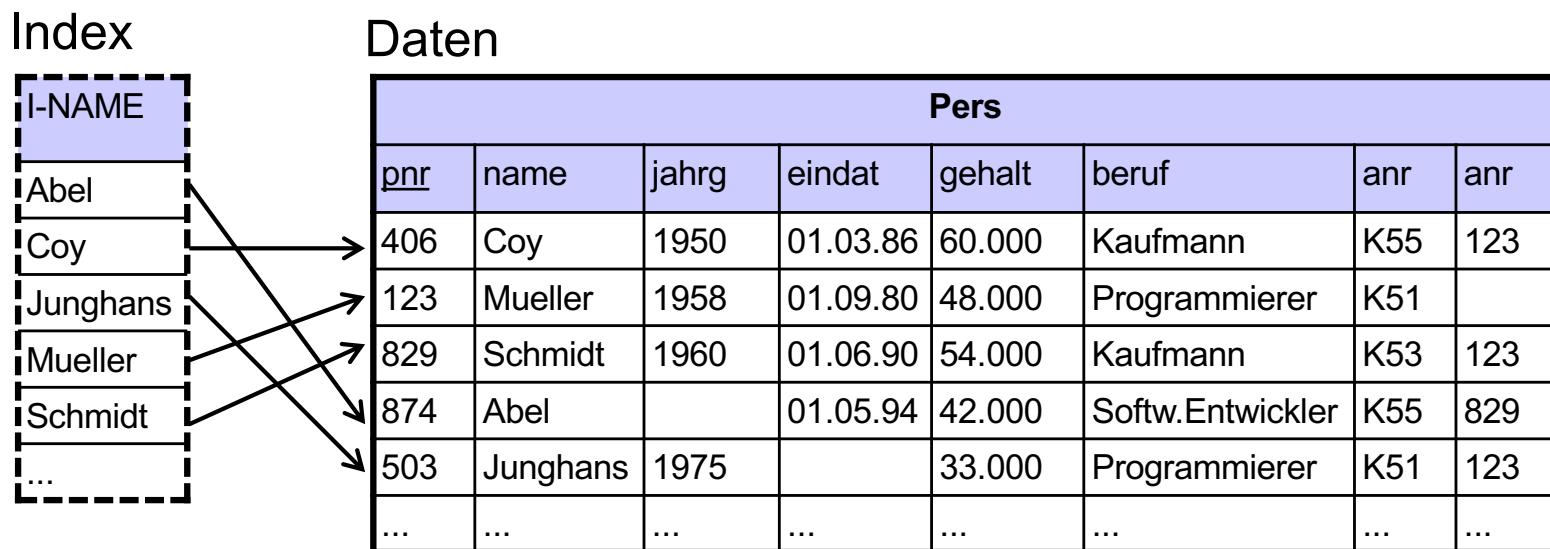
```
WITH Anzahl AS
 (SELECT vnr, COUNT(*) AS anz
 FROM Pers
 WHERE vnr IS NOT NULL
 GROUP BY vnr)
SELECT vnr
FROM Anzahl a
WHERE a.anz = (SELECT MAX(anz) FROM Anzahl);
```

# Übersicht

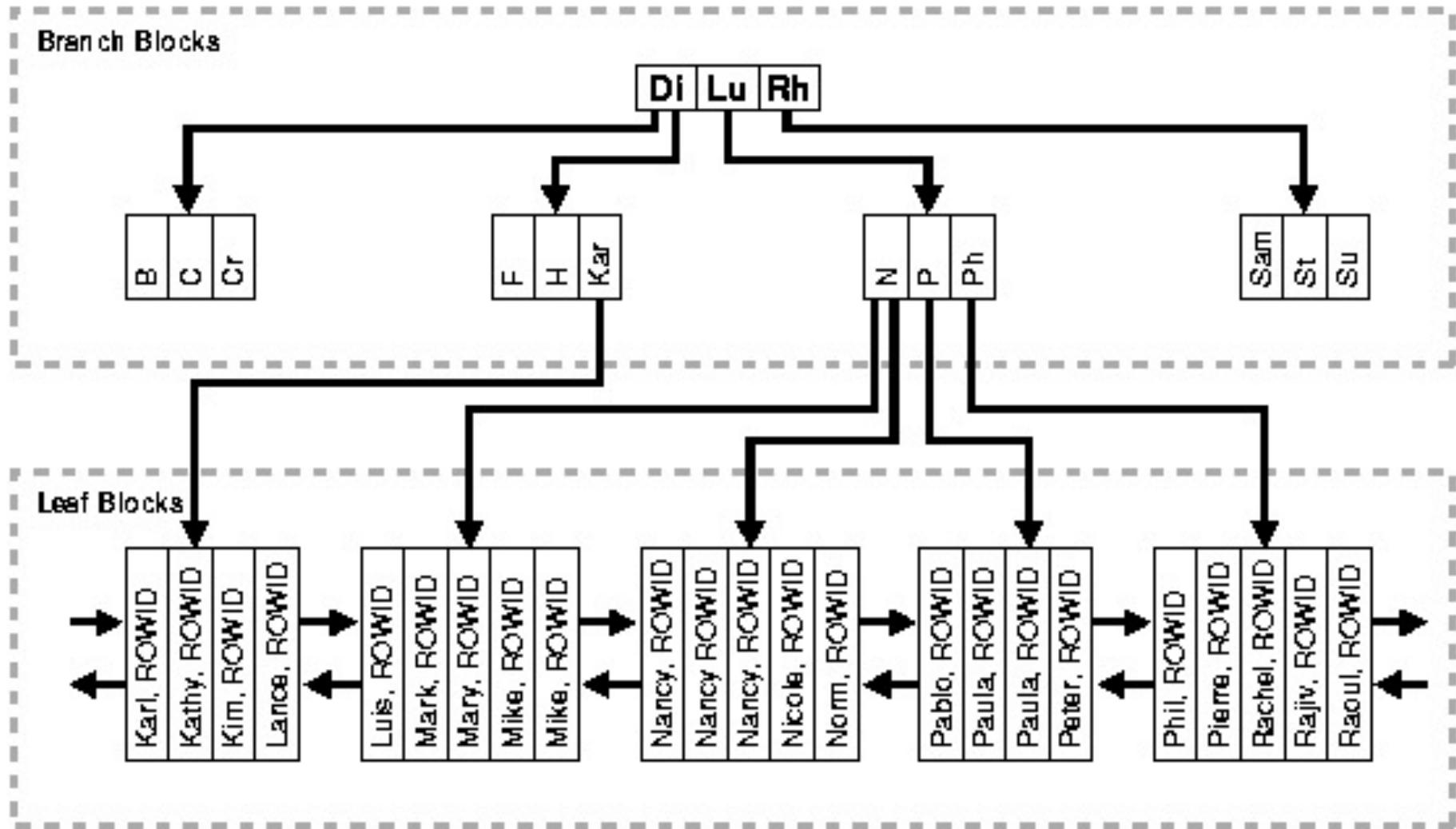
- SQL-Anfragen 2
  - Views
  - Indexierung
  - Large Objects
  - Join-Typen
  - Data Dictionary
  - Hierarchische Anfragen
  - Trigger

# Indexierung

- Indexierung = Erstellen eines sortierten Schlüssels für Spalten einer Tabelle
  - Anfrage
    - `SELECT *  
FROM pers  
WHERE name = 'Mueller';`



# Interne Struktur von Indizes in Oracle



Quelle: Oracle9i Database Online Documentation

# Indexierung

## Performance-Auswirkungen

- Schnellere Suchmethoden, z.B. binäres Suchen
- Keine Auswirkung auf das Ergebnis
  - ORDER-BY muß weiterhin angegeben werden
- Anpassen des Index notwendig nach
  - INSERT
  - UPDATE
  - DELETE
- Indexierung
  - beschleunigt Suchen
  - verlangsamt Aktualisieren

# Indexierung

## Syntax

- Anlegen eines Index
  - `CREATE [UNIQUE] INDEX indexname  
ON tabellenname  
(spaltenname [ASC/DESC]  
, spaltenname [ASC/DESC], ...);`
- Primärindex: Index auf Primärschlüssel  
Sekundärindex: Index auf Nicht-Primärschlüssel-Attribute
- UNIQUE INDEX
  - Es wird verhindert, dass in eine Spalte doppelte Werte eingegeben werden
  - Verwendung bei Spalten mit Primärschlüssel
- Löschen eines Index
  - `DROP INDEX indexname;`

# Index

## Beispiel

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, aname, ort})

| Pers |          |       |          |        |                  |     |     |
|------|----------|-------|----------|--------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt | beruf            | anr | vnr |
| 406  | Coy      | 1950  | 01.03.86 | 80.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1958  | 01.09.80 | 68.000 | Programmierer    | K51 |     |
| 829  | Schmidt  | 1960  | 01.06.90 | 74.000 | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.94 | 62.000 | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1975  |          | 55.000 | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...    | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | aname       | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchh       | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Für die Tabelle PERS soll ein Index auf dem Attribut NAME angelegt werden.

# Indexierung

- Ein Index bringt nicht nur Vorteile
  - Daher wichtig: Auswahl der Attribut mit Index
- In den meisten Datenbanksystemen wird auf dem Primärschlüssel automatisch ein Index definiert
- Auf welchem Attribut sollte im Beispiel ein Index definiert werden?
  - ```
SELECT gehalt, beruf, eindat
      FROM pers
     WHERE name = 'Mueller';
```
- Antwort
 - Die Spalten gehalt, beruf und eindat werden nur ausgegeben
 - Ein Index auf diesen Spalten bringt daher keinen Vorteil
 - Da nach name gesucht wird, ist für dieses Attribut ein Index sinnvoll
 - Name ändert sich auch nicht allzu häufig

Indexierung

Zusammengesetzter und funktionsbasierter Index

- Mehrspaltiger oder zusammengesetzter Index
 - Index über mehrere Spalten
 - Optimierte Anfrage:

```
SELECT pnr
FROM pers
WHERE name = 'Mueller'
AND beruf = 'Programmierer';
```
- Funktionsbasierter Index
 - Optimierte Anfrage:

```
SELECT ort
FROM abt
WHERE UPPER(aname) = 'PERSONAL';
```
 - Funktionsbasierte Indizes sollte nur selten eingesetzt werden

Übersicht

- SQL-Anfragen 2
 - Views
 - Indexierung
 - Large Objects
 - Join-Typen
 - Data Dictionary
 - Hierarchische Anfragen
 - Trigger

Large Objects (LOBs)

- Internal LOBs
 - BLOB – binary large object
 - CLOB – character large object
 - NCLOB – Texte in nationalen Zeichensatz
- Externe LOBs
 - BFILE: Verweis auf eine externe Datei, die außerhalb der Datenbank gespeichert ist
 - Meist read only
- Wichtiger Unterschied
 - Internal LOBs verwenden Copy Semantics
 - Externe LOBs verwenden Reference Semantics
 - Interne LOB fallen unter Transaktionsmanagement

Large Objects (LOBs)

- Definition von Tabellen mit internal LOBs

```
CREATE TABLE Person
( name      varchar2(30),
  pic       blob(16M));
```

- Zugriff auf LOBs

```
SELECT name, length(pic)  FROM Person
WHERE name = 'Müller';

SELECT p1.name as name1, p2.name as name 2
FROM Person p1, Person p2
WHERE p1.name < p2.name
AND p1.pic = p2.pic
```

Large Objects

- Einschränkungen bei LOBs
 - Operatoren <, <=, >, >= sind nicht definiert
 - LOBs können kein Primärschlüssel sein
 - Kein Index auf LOBs
 - Kein GROUP BY, ORDER BY auf LOBs
- Lesen und Schreiben von Daten aus / in LOBs
 - Siehe nächstes Kapitel unter JDBC

Übersicht

- 4.3 SQL-Anfragen 2
 - Views
 - Indexierung
 - Large Objects
 - Join-Typen
 - Data Dictionary
 - Hierarchische Anfragen
 - Trigger

An SQL query walks into a bar and sees two tables.
He walks up to them and asks 'Can I join you?'

Inner- und Outer-Joins

NAME	ANAME
Coy	Entwick
Mueller	Entwick
Schmidt	
Abel	Buchh
Junghans	Personal
...	...

Gebe alle Mitarbeiter und ihr Abteilungsname aus:

```
SELECT name, aname  
FROM pers, abt  
WHERE pers.anr = abt.anr
```

}

Alle Mitarbeiter, denen eine
Abteilung zugeordnet ist

UNION

```
SELECT name, ''  
FROM pers  
WHERE anr IS NULL
```

}

Alle Mitarbeiter, denen keine
Abteilung zugeordnet ist

Join-Typen in SQL99

- Cross-Join
 - Entspricht kartesischem Produkt ohne Bedingungen
 - Wird in der Praxis selten verwendet
- Join mit ON
 - Explizite Festlegung der Spalten für eine Join-Bedingung durch ON-Klausel, falls Spalten unterschiedliche Namen haben
- Natural-Join
 - Alle Spalten mit gleichen Namen und Typ werden in Join-Bedingung eingebunden
- USING
 - Festlegung der Join-Bedingung durch USING-Klausel
 - Beispiel: `pers JOIN abt USING (anr)`

Join-Typen

- Inner-Join
 - Enthält nur Daten aus den Tabellen, aus denen eine gemeinsame Übereinstimmung existiert
- Outer-Join
 - Beinhaltet auch Daten, die keine verwandten Daten aus einer anderen Tabelle existiert
 - Drei Typen:
 - LEFT OUTER JOIN
 - RIGHT OUTER JOIN
 - FULL OUTER JOIN
- Syntax Join-Statement
 - FROM QuellTabelle1 [Join-Typ] Tabelle [ON (Join Bedingung)]

Outer-Joins

Inner-Join:

```
SELECT name, aname  
FROM pers INNER JOIN abt ON pers.anr = abt.anr;
```

Left-Outer-Join (Left Join):

```
SELECT name, aname  
FROM pers LEFT OUTER JOIN abt ON pers.anr = abt.anr;
```

Right-Outer-Join (Right Join):

```
SELECT name, aname  
FROM pers RIGHT OUTER JOIN abt ON pers.anr = abt.anr;
```

Full-Outer-Join (Full Join):

```
SELECT name, aname  
FROM pers FULL OUTER JOIN abt ON pers.anr = abt.anr;
```

Inner-Join

Die folgenden Joins sind äquivalent

```
SELECT p.name, a.aname  
FROM pers p INNER JOIN abt a  
ON p.anr = a.anr  
WHERE a.aname = 'Personal';
```

```
SELECT p.name, a.aname  
FROM pers p INNER JOIN abt a  
ON p.anr = a.anr  
AND a.aname = 'Personal';
```



Weitere Darstellung für Joins

Syntax von Oracle vor Version 9i, **bitte nicht mehr verwenden !**

Left-Outer-Join

```
SELECT x.name, y.aname  
FROM pers x, abt y  
WHERE x.anr(+) = y.anr;
```

Right-Outer-Join

```
SELECT x.name, y.aname  
FROM pers x, abt y  
WHERE x.anr = y.anr(+);
```

Full-Outer-Join

```
SELECT x.name, y.aname  
FROM pers x, abt y  
WHERE x.anr(+) = y.anr(+);
```

Übersicht

- SQL-Anfragen 2
 - Views
 - Indexierung
 - Large Objects
 - Join-Typen
 - Data Dictionary
 - Hierarchische Anfragen
 - Trigger

Das Data Dictionary von Oracle

- Aufgaben
 - Speicherung aller Informationen, die zur Verwaltung der Objekte in der Datenbank benötigt werden
 - Tabellen- und Spaltennamen werden in Großbuchstaben konvertiert
- Views für Abfrage von Informationen
 - "Landkarten"-Views
 - Tabellen, Views, Spalten
 - Constraints
 - Indizes
 - Zugriffsrechte
 - etc.

Dictionary Views in Oracle

- "User"-Views
 - Präfix "USER"
 - Alle Informationen, die dem Account gehören, der Abfrage ausführt
- "ALL"-Views
 - Präfix "ALL"
 - USER-Datensätze zuzüglich Informationen über Objekte, für die öffentliche oder private Berechtigungen vergeben wurden
- "DBA"-Views
 - Präfix "DBA"
 - Alle Datenbankobjekte ohne Rücksicht auf Benutzer

Einige Beispiele für Dictionary Views

- **USER_TABLES**
 - Tabellen eines Benutzers
- **USER_TAB_COLS**
 - Attribute einer Tabellen eines Benutzers
- **USER_TRIGGERS**
 - Trigger eines Benutzers
- **USER_TAB_PRIVS**
 - Zugriffsrechte auf eine Tabelle eines Benutzers
- **USER_VIEWS**
 - Views eines Benutzers

Das Data Dictionary von Oracle

Beispiel

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, aname, ort})

Pers							
<u>pnr</u>	name	jahrg	eindat	gehalt	beruf	<u>anr</u>	<u>vnr</u>
406	Coy	1950	01.03.86	80.000	Kaufmann	K55	123
123	Mueller	1958	01.09.80	68.000	Programmierer	K51	
829	Schmidt	1960	01.06.90	74.000	Kaufmann	K53	123
874	Abel		01.05.94	62.000	Softw.Entwickler	K55	829
503	Junghans	1975		55.000	Programmierer	K51	123
...

Abt		
<u>anr</u>	aname	ort
K51	Entwicklung	Erlangen
K53	Buchh	Nürnberg
K55	Personal	Nürnberg
...

Welche Constraints sind für Tabelle pers definiert?

Übersicht

- SQL-Anfragen 2
 - Views
 - Indexierung
 - Large Objects
 - Join-Typen
 - Data Dictionary
 - Hierarchische Anfragen
 - Trigger

Hierarchische Anfragen in Datenbanken

- Beispiele für rekursive Datenmodelle
 - Familienbeziehungen (z.B. Vater, Sohn) oder Vorgesetztenbeziehung
 - Graphen, Bäume, rekursive Listen
 - Stücklisten in der Konstruktion und Fertigung
 - Fahrplanverbindungen
- Direkte Rekursion
 - Tabelle besitzt Fremdschlüsselbeziehung zu sich selbst
- Indirekte Rekursion
 - Rekursion durch zyklische Fremdschlüsselbeziehungen

Hierarchische Anfragen in Datenbanken

- Abfrage des Chefs von Herrn Abel
 - ```
SELECT p2.name
 FROM pers p1, pers p2
 WHERE p1.vnr = p2.pnr
 AND p1.name = 'Abel';
```
- Abfrage der Chefs von Herrn Abel (Stufe 2)
  - ```
SELECT p2.name, p3.name
  FROM pers p1, pers p2, pers p3
 WHERE p1.vnr = p2.pnr
   AND p2.vnr = p3.pnr
   AND p1.name = 'Abel';
```
- Abfrage der Chefs einer Person über beliebige Stufen
 - Rekursive Abfrage

Hierarchische Anfragen durch Sichten in SQL99

- Wiederverwendung mehrfach verwendeteter Teilanfragen
 - Syntax: `WITH Viewname [Cols] AS (<query>)`
- Entspricht temporäre Sicht für Dauer der Anfrageausführung
- Hierarchische Anfragen
 - Aufspannung von Graphen durch Referenzen
- Aufbau hierarchische Anfragen
 - Initialisierung (initial subquery)
 - Rekursion (rekursive subquery)
 - UNION ALL (rekursive Ausführung)

Hierarchische Abfragen durch Sichten in SQL99

- Rekursion durch rekursive Verwendung einer Sicht
- Verwendung von UNION ALL

```
– WITH RECURSIVE Chefs (pnr)
      (SELECT vnr
       FROM pers p1
       WHERE p1.name = 'Abel')
UNION ALL
      (SELECT p.pnr
       FROM Chefs c, pers p
       WHERE c.vnr = p.pnr)
SELECT p.name
       FROM Chefs c, pers p
       WHERE c.pnr = p.pnr;
```

The diagram illustrates the hierarchical structure of the recursive query. It is divided into three main sections by curly braces on the right:

- Initialisierung (Sicht)**: This section contains the first part of the query, which initializes the recursive view `Chefs` with the root node ('Abel').
- Weitere Rekursionsstufen**: This section contains the recursive part of the query, which defines the view `Chefs` as itself plus all children (subordinates).
- Bildung Endergebnis**: This section contains the final `SELECT` statement that retrieves the names of all nodes from the `Chefs` view.

Hierarchische Anfragen in Oracle

- Ausgangspunkt der Rekursion durch
 - START WITH
- Verbindung von Eltern-Objekten zu Kindern durch
 - CONNECT BY ... PRIOR
- Beispielanfrage: Ermittle alle Chefs von Herrn Abel
 - ```
SELECT name
 FROM pers
 CONNECT BY pnr = PRIOR vnr
START WITH name = 'Abel';
```
- Position von PRIOR bestimmt die Richtung der Rekursion
- Ohne START WITH wird Rekursion von allen Tupeln gestartet

# Hierarchische Anfragen in Oracle

- Regeln
  - Position von PRIOR relativ zu CONNECT-BY-Ausdruck bestimmt, welcher Ausdruck die Root und welcher die Zweige des Baums definiert
  - Eine WHERE-Klausel schließt einzelne Elemente des Baums aus, jedoch nicht deren Nachkommen
  - Eine CONNECT BY-Klausel schließt auch deren Nachkommen aus
  - Reihenfolge beim Einsatz von CONNECT BY:  
SELECT  
FROM  
WHERE  
CONNECT BY  
START WITH  
ORDER BY

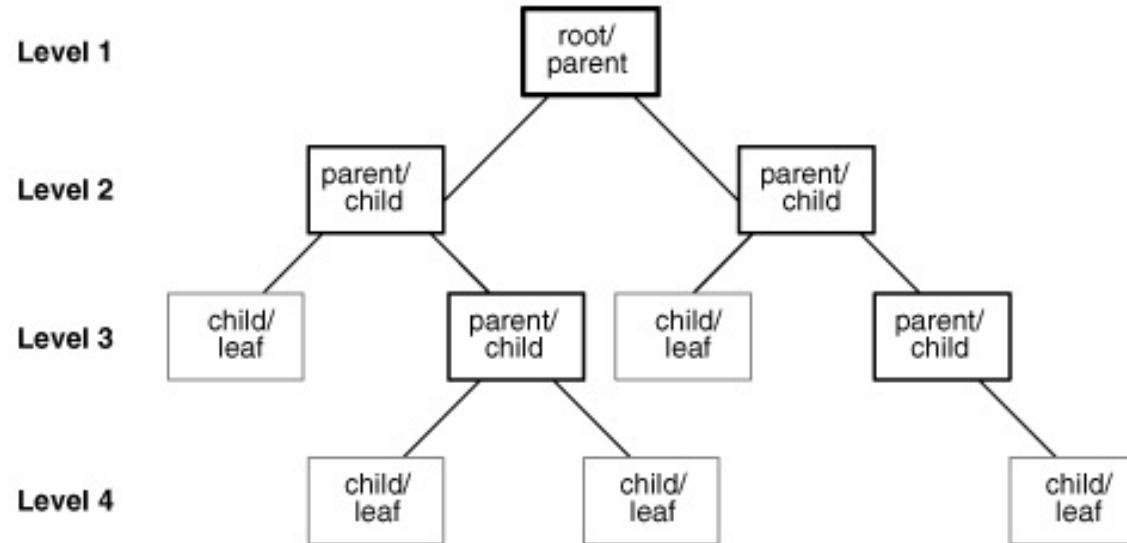
# Zyklen in hierarchischen Anfragen

- Zyklen
  - Falls Müller Vorgesetzter von Schmidt, Schmidt Vorgesetzter von Abel und Abel Vorgesetzter von Müller wäre, entsteht ein Zyklus
  - Abbruch in Anfrage mit Fehlermeldung "CONNECT BY loop in user data"
  - Fehler kann durch NOCYCLE verhindert werden

```
SELECT name
FROM pers
CONNECT BY NOCYCLE pnr = PRIOR vnr
START WITH name = 'Abel';
```

# Pseudospalten bei hierarchischen Anfragen

- Level
  - Tiefe, Level 1 für Wurzel
- CONNECT\_BY\_ISLEAF
  - Anzeige ob Knoten Blatt ist
- CONNECT\_BY\_ISCYCLE
  - Anzeige Zirkelbezug



Quelle: docs.oracle.com

# Hierarchische Anfragen in Oracle

- Sortierung
  - ORDER SIBLINGS BY name
  - Sortierung der Child-Knoten
- Beispiel

```
SELECT last_name, employee_id, manager_id, LEVEL
FROM employees
START WITH employee_id = 100
CONNECT BY PRIOR employee_id = manager_id
ORDER SIBLINGS BY last_name;
```

```
SELECT last_name, employee_id, manager_id, LEVEL
FROM employees
START WITH employee_id = 100
CONNECT BY PRIOR employee_id = manager_id
ORDER SIBLINGS BY last_name;
```

| LAST_NAME | EMPLOYEE_ID | MANAGER_ID | LEVEL |
|-----------|-------------|------------|-------|
| King      | 100         |            | 1     |
| Cambrault | 148         | 100        | 2     |
| Bates     | 172         | 148        | 3     |
| Bloom     | 169         | 148        | 3     |
| Fox       | 170         | 148        | 3     |
| Kumar     | 173         | 148        | 3     |
| Ozer      | 168         | 148        | 3     |
| Smith     | 171         | 148        | 3     |
| De Haan   | 102         | 100        | 2     |
| Hunold    | 103         | 102        | 3     |
| Austin    | 105         | 103        | 4     |
| Ernst     | 104         | 103        | 4     |
| Lorentz   | 107         | 103        | 4     |
| Pataballa | 106         | 103        | 4     |
| Errazuriz | 147         | 100        | 2     |
| Ande      | 166         | 147        | 3     |
| Banda     | 167         | 147        | 3     |
| ...       |             |            |       |

Quelle: docs.oracle.com

# Hierarchische Anfragen

## Beispiel 20

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, aname, ort)

| Pers |          |       |          |        |                  |     |     |
|------|----------|-------|----------|--------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt | beruf            | anr | vnr |
| 406  | Coy      | 1950  | 01.03.86 | 80.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1958  | 01.09.80 | 68.000 | Programmierer    | K51 |     |
| 829  | Schmidt  | 1960  | 01.06.90 | 74.000 | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.94 | 62.000 | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1975  |          | 55.000 | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...    | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | aname       | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchh       | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Ermittle alle Chefs von Herrn Abel, die mehr als 70.000 verdienen

# Hierarchische Anfragen

## Beispiel

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, aname, ort})

| Pers       |          |       |          |        |                  |            |            |
|------------|----------|-------|----------|--------|------------------|------------|------------|
| <u>pnr</u> | name     | jahrg | eindat   | gehalt | beruf            | <u>anr</u> | <u>vnr</u> |
| 406        | Coy      | 1950  | 01.03.86 | 80.000 | Kaufmann         | K55        | 123        |
| 123        | Mueller  | 1958  | 01.09.80 | 68.000 | Programmierer    | K51        |            |
| 829        | Schmidt  | 1960  | 01.06.90 | 74.000 | Kaufmann         | K53        | 123        |
| 874        | Abel     |       | 01.05.94 | 62.000 | Softw.Entwickler | K55        | 829        |
| 503        | Junghans | 1975  |          | 55.000 | Programmierer    | K51        | 123        |
| ...        | ...      | ...   | ...      | ...    | ...              | ...        | ...        |

| Abt        |             |          |
|------------|-------------|----------|
| <u>anr</u> | aname       | ort      |
| K51        | Entwicklung | Erlangen |
| K53        | Buchh       | Nürnberg |
| K55        | Personal    | Nürnberg |
| ...        | ...         | ...      |

Wer verdient mehr als einer seiner Chefs (rekursiv)?

# Übersicht

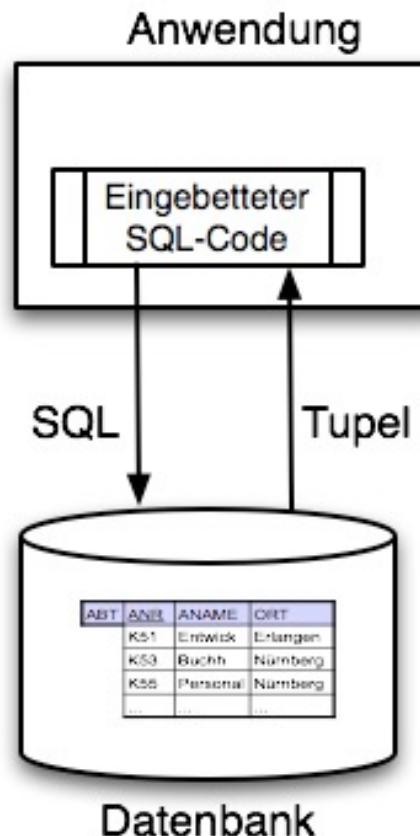
- SQL-Anfragen 2
  - Views
  - Indexierung
  - Large Objects
  - Join-Typen
  - Data Dictionary
  - Rekursion
  - Trigger

# Stored Procedures

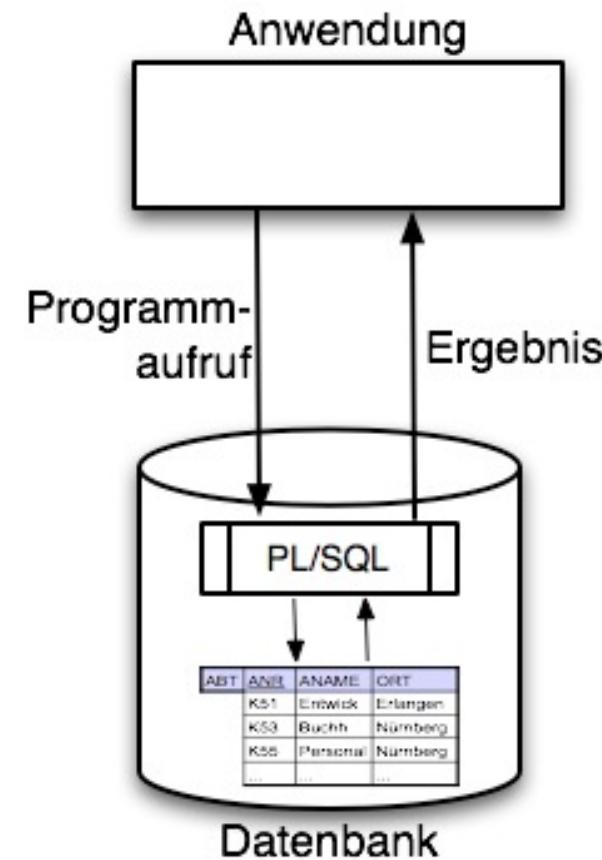
- Stored Procedures (gespeicherte Prozeduren)
  - Speichern und Ausführen von Anwendungslogik direkt im Datenbanksystem
- Vorteile
  - Performance, da weniger Datentransfer
  - Vermeidung von Code-Redundanzen
  - Erhöhung der Datensicherheit
  - Einhaltung von komplexen Integritätsregeln
- Prozeduren und Funktionen in Oracle
  - CREATE PROCEDURE bzw. CREATE FUNCTION
  - EXECUTE

# PL/SQL - Datenbankprogrammierung in Oracle

Datenbankoperationen bisher



Datenbankoperationen mit  
Stored Procedures und PL/SQL



# PL/SQL - Datenbankprogrammierung in Oracle

- PL/SQL: Prozedurales Sprach-Superset von SQL
- Sprache mit Programmiermöglichkeiten (Schleifen, bedingten Verzweigungen, explizite Fehlerbehandlung)
- Gruppierung in Blöcke
- Ein Block umfasst drei Bereiche
  - Deklarationen: Definition von Variablen
  - Executable-Befehle: Ausführung von Befehlen
  - Exception-Behandlung: Verarbeitung von Fehlerbedingungen
- Mengenwertige Abfragen durch Cursor (s. Folien über Embedded SQL)

# PL/SQL-Programm

## Beispiel

```
declare
 pi constant NUMBER(9,7) := 3.1415927;
 radius INTEGER(5);
 area NUMBER(14,2);
 some_variable NUMBER(14,2);
begin
 radius := 2;
 loop
 some_variable := 1/(radius-7);
 area := pi*power(radius,2);
 insert into AREAS values (radius, area);
 radius := radius+1;
 exit when area>1000;
 end loop;
exception
 when ZERO_DIVIDE
 then insert into AREAS values (radius,0);
end;
```

Deklaration von Variablen

Kontrollstrukturen und SQL-Befehle

Fehlerbehandlungen

# Triggerkonzept

- Trigger sind gespeicherte Prozeduren, welche bei einem bestimmten Ereignis ausgelöst werden
  - Automatische Durchführung bei der Ausführung einer INSERT-, UPDATE- oder DELETE-Anweisung
  - Anwendung z.B. zur Gewährleistung der Integrität
  - Aufruf von SQL bzw. PL/SQL-Prozeduren (in Oracle)
- Trigger-Elemente ECA aktiver Datenbanken

|           |                           |
|-----------|---------------------------|
| Event     | Regelauslösendes Ereignis |
| Condition | Bedingung                 |
| Action    | Auszuführende Aktionen    |

# Triggerkonzept in SQL

- Ereignis (Event)
  - Löschen von Tupel einer Tabelle (`DELETE`)
  - Hinzufügen von Tupel in eine Tabelle (`INSERT`)
  - Ändern von Attributen eines bestehenden Tupels (`UPDATE`)
  - ...
- Bedingung (Condition)
  - Die Bedingung wird überprüft, wenn das Ereignis aufgetreten ist
- Rumpf (Action)
  - Aktivität wird nur ausgeführt, wenn Bedingung erfüllt ist
  - Auszuführende SQL-Anweisungen

# Triggerkonzept in SQL

- Aktivierungszeitpunkt
  - BEFORE: Aktivierung vor der ereignisauslösenden Anweisung
  - AFTER: Aktivierung nach der ereignisauslösenden Anweisung
  - INSTEAD OF: Aktivierung anstatt der Anweisung
- Transitions-Variable
  - NEW: Zugriff auf den Attributwert *nach* der Änderung
  - OLD: Zugriff auf den Attributwert *vor* der Änderung
  - Im PL/SQL-Block ist den Schlüsselwörter ein NEW und OLD ein Doppelpunkt ":" voranzustellen (z.B. :new.name)
- Zeilen-/Anweisungsebene
  - FOR EACH ROW: Trigger auf Zeilenebene, d.h. Ausführung pro relevante Zeile
  - FOR EACH STATEMENT: Trigger auf Anweisungsebene, d.h. Ausführung pro Befehl

# Triggerkonzept in SQL

- Syntax

- CREATE TRIGGER triggername  
AFTER | BEFORE | INSTEAD OF  
INSERT | DELETE | UPDATE  
[ OF column ] ON tabellenname  
{ FOR EACH STATEMENT | FOR EACH ROW }  
WHEN bedingung  
SQL-Statement | PL/SQL-Code

*Zeitpunkt*

*Ereignis*

*Bedingung*  
*Aktionen*

- Einführung in SQL3

- Rekursive Trigger

- Trigger können selbst weitere Trigger aktivieren
  - Eine Terminierung zyklischer Trigger ist nicht gewährleistet !

# Triggerkonzept

## Beispiel

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, aname, ort})

| Pers       |          |       |          |        |                  |            |            |
|------------|----------|-------|----------|--------|------------------|------------|------------|
| <u>pnr</u> | name     | jahrg | eindat   | gehalt | beruf            | <u>anr</u> | <u>vnr</u> |
| 406        | Coy      | 1950  | 01.03.86 | 80.000 | Kaufmann         | K55        | 123        |
| 123        | Mueller  | 1958  | 01.09.80 | 68.000 | Programmierer    | K51        |            |
| 829        | Schmidt  | 1960  | 01.06.90 | 74.000 | Kaufmann         | K53        | 123        |
| 874        | Abel     |       | 01.05.94 | 62.000 | Softw.Entwickler | K55        | 829        |
| 503        | Junghans | 1975  |          | 55.000 | Programmierer    | K51        | 123        |
| ...        | ...      | ...   | ...      | ...    | ...              | ...        | ...        |

| Abt        |             |          |
|------------|-------------|----------|
| <u>anr</u> | aname       | ort      |
| K51        | Entwicklung | Erlangen |
| K53        | Buchh       | Nürnberg |
| K55        | Personal    | Nürnberg |
| ...        | ...         | ...      |

Im Attribut ORT sollen ab sofort nur noch Großbuchstaben verwendet werden.

# Triggerkonzept

## Beispiel

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, aname, ort})

| Pers       |          |       |          |        |                  |            |            |
|------------|----------|-------|----------|--------|------------------|------------|------------|
| <u>pnr</u> | name     | jahrg | eindat   | gehalt | beruf            | <u>anr</u> | <u>vnr</u> |
| 406        | Coy      | 1950  | 01.03.86 | 80.000 | Kaufmann         | K55        | 123        |
| 123        | Mueller  | 1958  | 01.09.80 | 68.000 | Programmierer    | K51        |            |
| 829        | Schmidt  | 1960  | 01.06.90 | 74.000 | Kaufmann         | K53        | 123        |
| 874        | Abel     |       | 01.05.94 | 62.000 | Softw.Entwickler | K55        | 829        |
| 503        | Junghans | 1975  |          | 55.000 | Programmierer    | K51        | 123        |
| ...        | ...      | ...   | ...      | ...    | ...              | ...        | ...        |

| Abt        |             |          |
|------------|-------------|----------|
| <u>anr</u> | aname       | ort      |
| K51        | Entwicklung | Erlangen |
| K53        | Buchh       | Nürnberg |
| K55        | Personal    | Nürnberg |
| ...        | ...         | ...      |

Zu jeder Gehaltserhöhung wird ein Bonus von 100 € dazugerechnet

# SQL-Anfragen

## Wiederholung

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, aname, ort})

| Pers       |          |       |          |        |                  |            |            |
|------------|----------|-------|----------|--------|------------------|------------|------------|
| <u>pnr</u> | name     | jahrg | eindat   | gehalt | beruf            | <u>anr</u> | <u>vnr</u> |
| 406        | Coy      | 1950  | 01.03.86 | 80.000 | Kaufmann         | K55        | 123        |
| 123        | Mueller  | 1958  | 01.09.80 | 68.000 | Programmierer    | K51        |            |
| 829        | Schmidt  | 1960  | 01.06.90 | 74.000 | Kaufmann         | K53        | 123        |
| 874        | Abel     |       | 01.05.94 | 62.000 | Softw.Entwickler | K55        | 829        |
| 503        | Junghans | 1975  |          | 55.000 | Programmierer    | K51        | 123        |
| ...        | ...      | ...   | ...      | ...    | ...              | ...        | ...        |

| Abt        |             |          |
|------------|-------------|----------|
| <u>anr</u> | aname       | ort      |
| K51        | Entwicklung | Erlangen |
| K53        | Buchh       | Nürnberg |
| K55        | Personal    | Nürnberg |
| ...        | ...         | ...      |

Aus welchem Jahrgang hat die Firma die meisten Mitarbeiter?

# SQL-Anfragen in Klausuren

Durch die folgenden Datenbank-Tabellen werden Behandlungsdaten von Patienten einer Arztpraxis gespeichert.

```
Patient = ({pnr, name, adresse, krankenkasse}, { })
Arztbesuch = ({anr, pnr, datum, diagnose}, { })
Leistung = ({lnr, peschreibung, kosten}, { })
Durchfuehren = ({anr, lnr}, { })
```

Anmerkung: Ein Arzt führt bei einem Krankenbesuch mehrere Behandlungen durch. Die Relation durchführen beschreibt, welche Leistungen bei welchem Arztbesuch durchgeführt wurden.

Die folgenden Tabellen verdeutlichen die Bedeutung dieser Relationen durch Belegungen mit Beispieldaten:

| Patient | <u>pnr</u> | name   | adresse      | krankenkasse |
|---------|------------|--------|--------------|--------------|
|         | 12         | Maier  | Seestr. 4    | AOK          |
|         | 14         | Müller | Mainaustr. 2 | Techniker    |
|         | ...        |        | ...          |              |

| Arztbesuch | <u>anr</u> | <u>pnr</u> | datum  | diagnose  |
|------------|------------|------------|--------|-----------|
|            | 26         | 14         | 3.6.18 | Beinbruch |
|            | 28         | 12         | 4.6.18 | Masern    |
|            | ...        | ...        |        |           |

| Leistung | <u>lnr</u> | beschreibung     | kosten |
|----------|------------|------------------|--------|
|          | 124        | Blutdruck messen | 30     |
|          | 125        | Bein eingipsen   | 100    |
|          | ...        | ...              |        |

| Durchfuehren | <u>anr</u> | <u>lnr</u> |
|--------------|------------|------------|
|              | 26         | 124        |
|              | 28         | 125        |
|              | ...        | ...        |

Stellen Sie SQL-Statements gemäß SQL-Standard für folgende Datenbank-Anfragen an. Geben Sie immer die entsprechenden Namen und Bezeichnungen aus (z.B. Leistungsbeschreibung) und nicht die intern verwendeten Schlüssel (z.B. lnr oder pnr).

- Wie vielen AOK-Patienten wurde bereits ein Bein eingegipst?
- Wie viele Patienten haben 2018 die Praxis nicht besucht?
- Stellen Sie die Anzahl der Masernfälle in 2017 pro Krankenkasse zusammen.
- Welche Leistung hat die meisten Kosten verursacht?
- Wie viele Patienten haben bereits mehrmals Masern bekommen?

# SQL-Anfragen in Klausuren - Lösung

a)

```
SELECT COUNT(DISTINCT p.pnr)
FROM patient p, arztbesuch a, durchfuehren d, leistung l
WHERE p.pnr = a.pnr
AND d.anr=a.anr
AND d.lnr = l.lnr
AND l.beschreibung = 'Bein gipsen'
AND p.kasse = 'AOK';
```

b)

```
SELECT COUNT(*)
FROM patient
WHERE pnr not in
(SELECT pnr
 FROM arztbesuch
 WHERE TO_CHAR(datum, 'YYYY') = '2018')
```

c)

```
SELECT p.kasse, count(*)
FROM patient p, arztbesuch a
WHERE a.pnr = p.pnr
AND a.diagnose = 'Masern'
AND TO_CHAR(datum, 'YYYY') = '2017'
GROUP BY p.kasse;
```

d)

```
CREATE VIEW z(lnr, k) AS
SELECT l.lnr, sum(kosten)
FROM leistung l, durchfuehren d
WHERE l.lnr=d.lnr
GROUP BY l.lnr;
```

```
SELECT beschreibung
FROM z, leistung l
WHERE z.lnr=l.lnr
AND k = (SELECT MAX(k) FROM z);
```

e)

```
SELECT COUNT(DISTINCT pnr)
FROM Arztbesuch a1, Arztbesuch a2
WHERE a1.pnr = a2.pnr
AND a1.anr != a2.anr
AND a1.diagnose = 'Masern'
AND a2.diagnose = 'Masern';
```