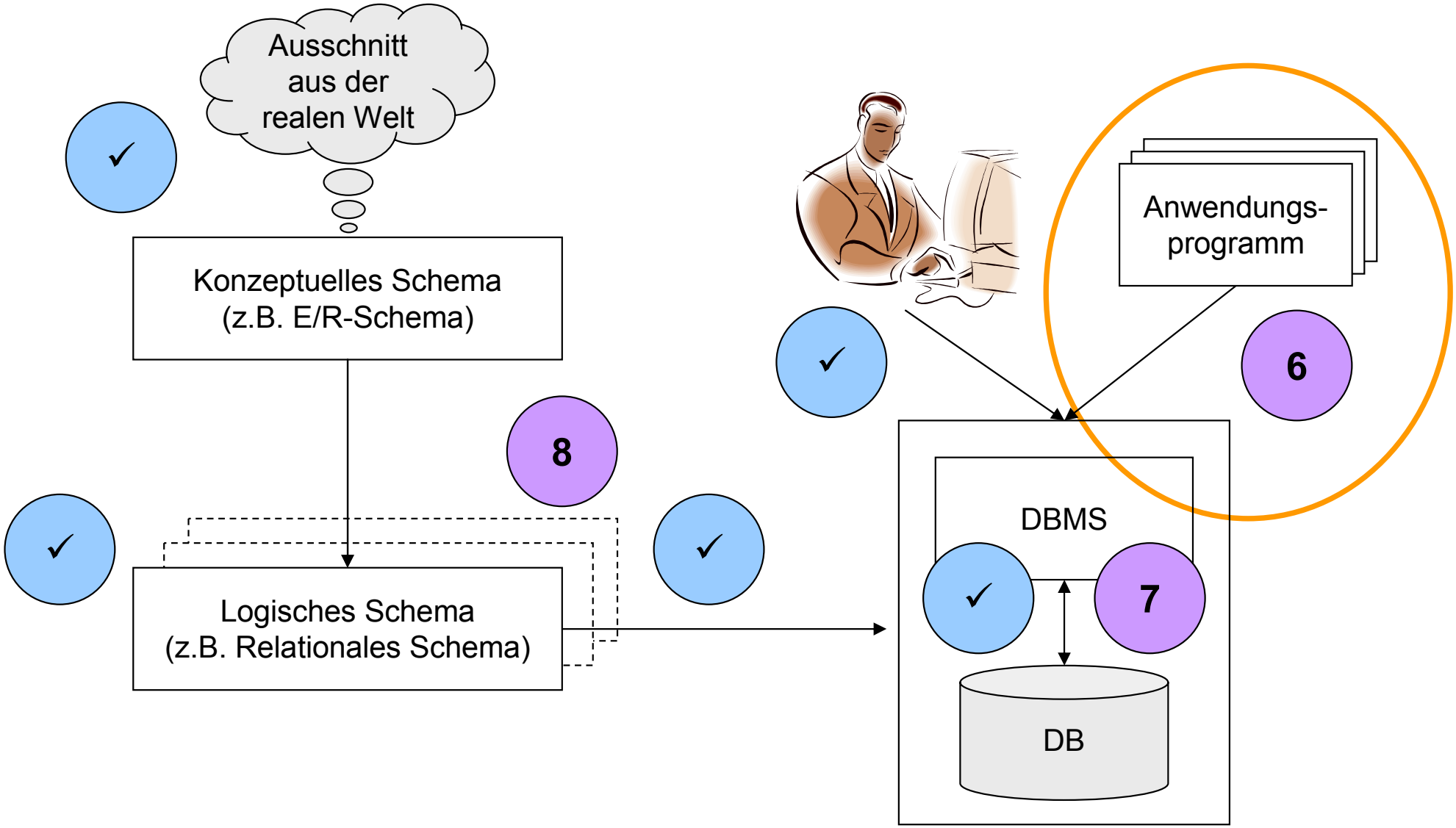
Aufbau Vorlesung Datenbanken II – Kapitel 6, 7 und 8



- 6.1 SQL - Structured Query Language / weiterführende Konzepte
  - 6.1.1 Vergabe von Zugriffsrechten: grant, revoke, roles
  - 6.1.2 Der Systemkatalog
  - 6.1.3 embedded SQL / PL/SQL / SQLJ
- 6.2 Stored procedures & Trigger
  - 6.2.1 Stored procedures
  - 6.2.2 Trigger
- 6.3 Architekturkonzepte: objektorientierte Sprache / relationales DBMS
  - 6.3.1 ODBC & Data Access Objects (DAO)
  - 6.3.2 JDBC & Data Access Beans (DAB)

6.1.1 Vergabe von Zugriffsrechten: grant, revoke, roles

- In der Regel sollen nicht alle Informationen eines Datenbanksystems allen Benutzergruppen gleichermaßen zur Verfügung stehen. Mit Hilfe von SQL können unterschiedliche Zugriffsrechte für relationale Datenbanken vergeben werden.

- Bei der Vergabe von Zugriffsrechten auf Datenbanken müssen grundsätzlich drei Aspekte berücksichtigt werden:

> **WER** wird autorisiert (Subjekt),
> für **WELCHE DATEN** wird das Subjekt autorisiert,
> für **WELCHE OPERATIONEN** darf der Zugriff auf die Daten erfolgen.

- Die Verwaltung der erteilten Zugriffsrechte erfolgt ausschließlich über das DBMS.

6.1.1 Vergabe von Rechten: Die GRANT-Anweisung (SQL-92) (1)

Syntax: **GRANT** <Recht,>
ON <Objekt>
TO <User,> **[WITH GRANT OPTION]**;

- Jeder Creator eines Datenbankobjektes erhält automatisch alle Rechte, die für dieses Objekt sinnvoll sind.

  Beispiel
  Der Creator einer Tabelle B erhält automatisch SELECT, INSERT, UPDATE, DELETE und REFERENCES Rechte auf B.

- Außerdem hat der Creator das Recht, alle diese Rechte vollständig oder eingeschränkt an andere Benutzer(gruppen) weiter zu vergeben.
- Es ist wichtig, für jeden Benutzer zu speichern, zu welchem Zeitpunkt und von welchen Benutzern er Rechte erhalten hat (vgl. auch Folie 9).

6.1.1 Vergabe von Rechten: Die GRANT-Anweisung (SQL-92) (2)

- **<Recht,>** In der Liste der Rechte kann das Schlüsselwort **all** (Langform **all privileges**) als Platzhalter für alle zu vergebenden Rechte stehen, oder es können die folgenden Schlüsselwörter verwendet werden:

  - **select** (Leserecht),
  - **insert [(column,)]**,
  - **update [(column,)]**,
  - **usage** (zur Recht-Vergabe im Zusammenhang mit Domains),
  - **references [(column,)]** (zur Recht-Vergabe im Zusammenhang mit der Referenzierung einer speziellen Tabelle im Rahmen eines Foreign Key-Constraints).

6.1.1 Vergabe von Rechten: Die GRANT-Anweisung (SQL-92) (3)

- **<Objekt>** Ein Zugriffsobjekt ist in der Regel eine
  - **Table**, aber auch eine
  - **View**, oder ein
  - **Domain**.

Ist das Objekt eine Tabelle oder eine View so reicht es, den Namen des Objektes zu nennen; die Angabe des Schlüsselwortes TABLE ist optional.

Ist das Objekt ein Domain, so lautet die Syntax.

```
... ON DOMAIN <domain-Name>
```

- **<User,>** Diese Liste besteht in der Regel aus einem oder mehreren **Benutzerkennungen**, oder aus dem Schlüsselwort **PUBLIC**, mit dem alle Benutzerkennungen für das System zu jedem Zeitpunkt gemeint sind.

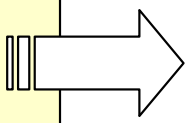
- **WITH GRANT OPTION**

  Mit dieser Option wird das **Recht auf die Erteilung von Rechten** bzgl. der Rechteliste und des Objektes der GRANT-Anweisung vergeben.

Beispiel

```
create view MeinAuftrag as
   select  *
   from    Auftrag
   where   Kunde = user;

grant select, insert
on  MeinAuftrag
to  public;
```



Welche Wirkung hat die Definition der View MeinAuftrag und die darauf basierende Rechtevergabe?

6.1.1 Vergabe von Rechten: Die REVOKE-Anweisung (SQL-92) (1)

- Alle mit der GRANT-Anweisung vergebenen Rechte können mit der REVOKE-Anweisung wieder zurück genommen werden:

```
Syntax: REVOKE [GRANT OPTION FOR]  <Recht,>
        ON         <Objekt>
        FROM       <User,>  {RESTRICT | CASCADE};
```

- In Analogie zur RESTRICT- bzw. CASCADE-Klausel im Zusammenhang mit DELETE- und UPDATE-Operationen regelt diese Klausel innerhalb der REVOKE-Anweisung den Umgang mit weitergereichten Rechten:

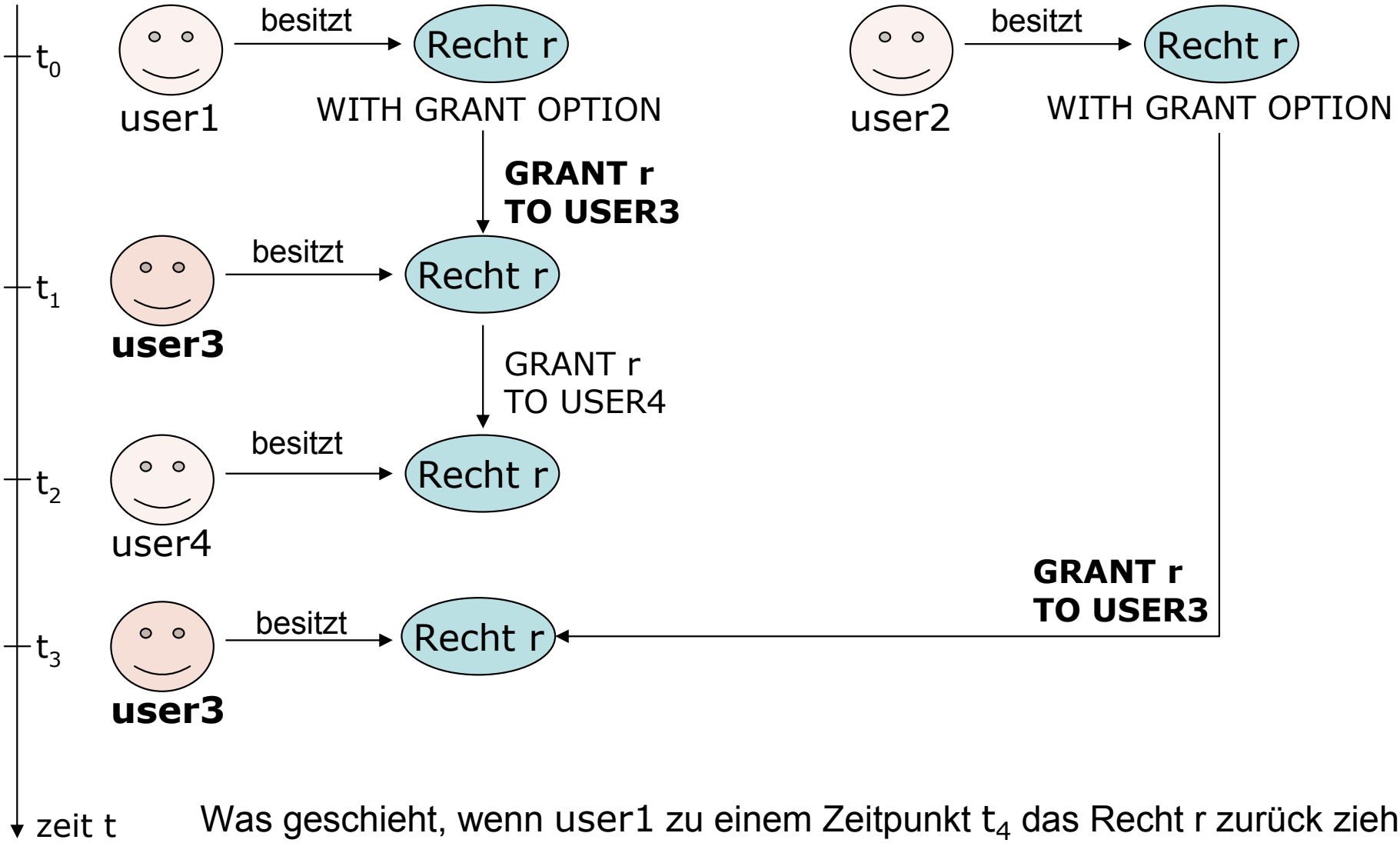
  RESTRICT: Die REVOKE-Anweisung wird abgebrochen, wenn das Recht von dem User, dem es entzogen werden soll, an Dritte weitergegeben wurde.

  CASCADE: Die REVOKE-Anweisung setzt sich automatisch fort über alle entsprechenden Rechtevergaben durch den User an Dritte.

  GRANT OPTION FOR: Bei Nutzung dieser Schlüsselworte im Rahmen der REVOKE-Anweisung werden dem User nicht die Rechte an sich, sondern nur das Recht auf die Erteilung dieser Rechte entzogen.

6.1.1 Vergabe von Rechten: Die REVOKE-Anweisung (SQL-92) (2)

Beispiel



Was geschieht, wenn user1 zu einem Zeitpunkt t_4 das Recht r zurück zieht?

6.1.1 Gruppierung von Rechten zu Rollen: Vergabe von **ROLE**s (SQL-99)

- Um personen- und benutzergruppen-unabhängig Rechte zu vergeben ist es seit SQL-99 möglich, Rollen zu generieren, für die man mit Hilfe der GRANT- und REVOKE-Anweisungen Rechte verwalten kann:

```
Syntax: CREATE ROLE <Rollen-Name>;
```

- Ein bestimmter User wird mit Hilfe der GRANT-Anweisung einer (oder mehreren) Rolle(n) zugewiesen:

```
Syntax: GRANT <Rollen-Name>
        TO    <User>;
```

- Einer Rolle kann unabhängig davon, ob und wieviele Benutzer(gruppen) ihr gerade angehören, eine Menge von Rechten zugewiesen werden.
- Die Rechte einer bestimmten Person bestehen also aus den persönlichen Rechten und allen Rechten derjenigen Rollen, denen diese Person zugeordnet wurde.
- Zur Gewährleistung der Sicherheit von Datenbanken gibt es noch zahlreiche weiterführende Konzepte (vgl. hierzu die Literatur).

6.1.2 Der Systemkatalog (1)

- Alle Metadaten zu einem Objekt einer Datenbank, also alle Daten zu den erzeugten Schemaobjekten wie Tabellen, Views, Indexe etc., werden von einem DBMS in speziellen Systemtabellen gespeichert, dem so genannten **Systemkatalog** (man spricht auch von *Data Dictionary*).

- Die Inhalte der Tabellen des Systemkatalogs können mit den SQL-SELECT-Anweisungen abgefragt werden.

- Die einzelnen Tabellen des Systemkatalogs stehen miteinander in Beziehung, sodass dem Systemkatalog ein eigenes relationales Datenmodell zugrunde liegt, das automatisch mit Daten gefüllt wird, sobald eine Datenbankstruktur angelegt wird.

- Die **Bezeichner für die Systemtabellen** variieren von System zu System.
  Bei vielen DBMS beginnen die Systemtabellen mit dem Präfix SYS, z.B. bei Informix und MS SQL-Server, und haben ansonsten sprechende Bezeichner, z.B. SYSTABLES, SYSCOLUMNS, SYSVIEWS,SYSREFERENCES etc.

6.1.2 Der Systemkatalog (2)

- **Namenskonventionen der Systemtabellen bei Oracle**
  Die Systemtabellen sind bei Oracle nach folgendem Schema aufgebaut:

<Präfix>_<Schemaobjekt>

- Das **Präfix** schränkt die anzuzeigenden Objekte wie folgt ein:

USER_	Alle Schemaobjekte des aktuellen Users, also des entsprechenden Schemas**.**
ALL_	Alle Schemaobjekte, auf die der User Zugriffsrechte hat.
DBA_	Alle Schemaobjekte; nur ein Datenbankadministrator (der User DBA) darf hierauf zugreifen

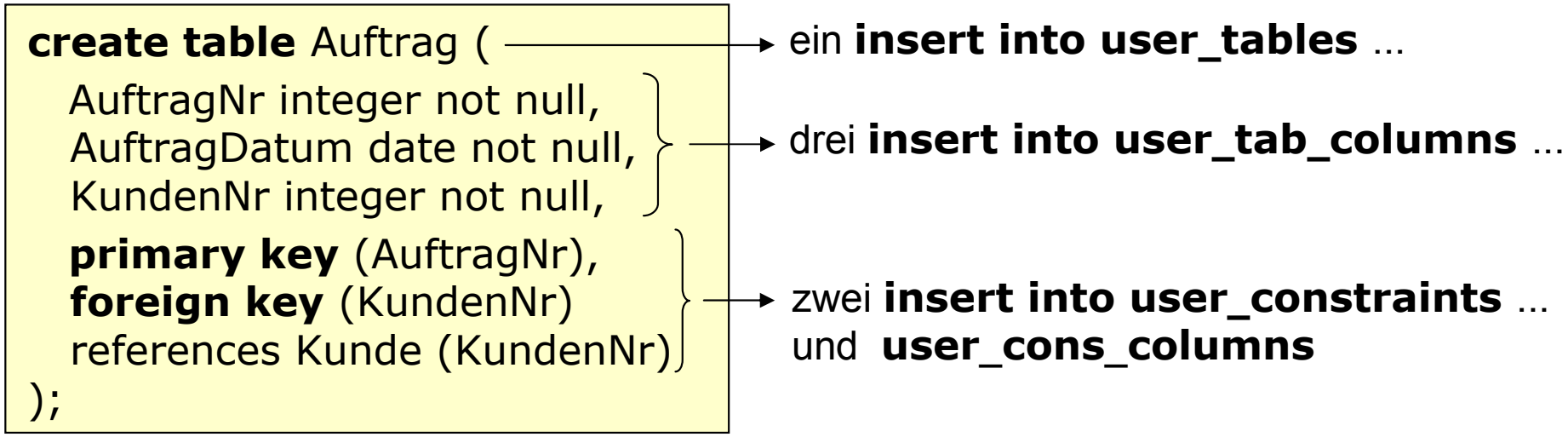
- Eine Auswahl von wichtigen **Schemaobjekten** *)**.**

CONS_COLUMNS **CONSTRAINTS** **INDEXES** **TABLES**
TABLESPACES **TAB_COLUMNS** **TRIGGERS** **VIEWS** **...**

*) Die Tabellenstruktur der Schemaobjekte kann durch entsprechendes DESC <Tablename> erfragt werden.

6.1.2 Der Systemkatalog (3)

- Beispiel



- Zahlreiche **Re-Engineering-Tools** nutzen den Systemkatalog, um im reverse-Verfahren ein entsprechendes (physisches) relationales und ein konzeptionelles Datenmodell der Datenbank zu generieren.

Der Systemkatalog ist insbesondere dann wichtig,
wenn mangels Dokumentation zum Datenmodell
keine schriftlichen Informationen über die Datenstruktur vorliegen!

6.1.2 Der Systemkatalog (4)

- Welche Informationen liefert diese SELECT-Anweisung?

```
select *
from  all_tab_columns
order by table_name;
```

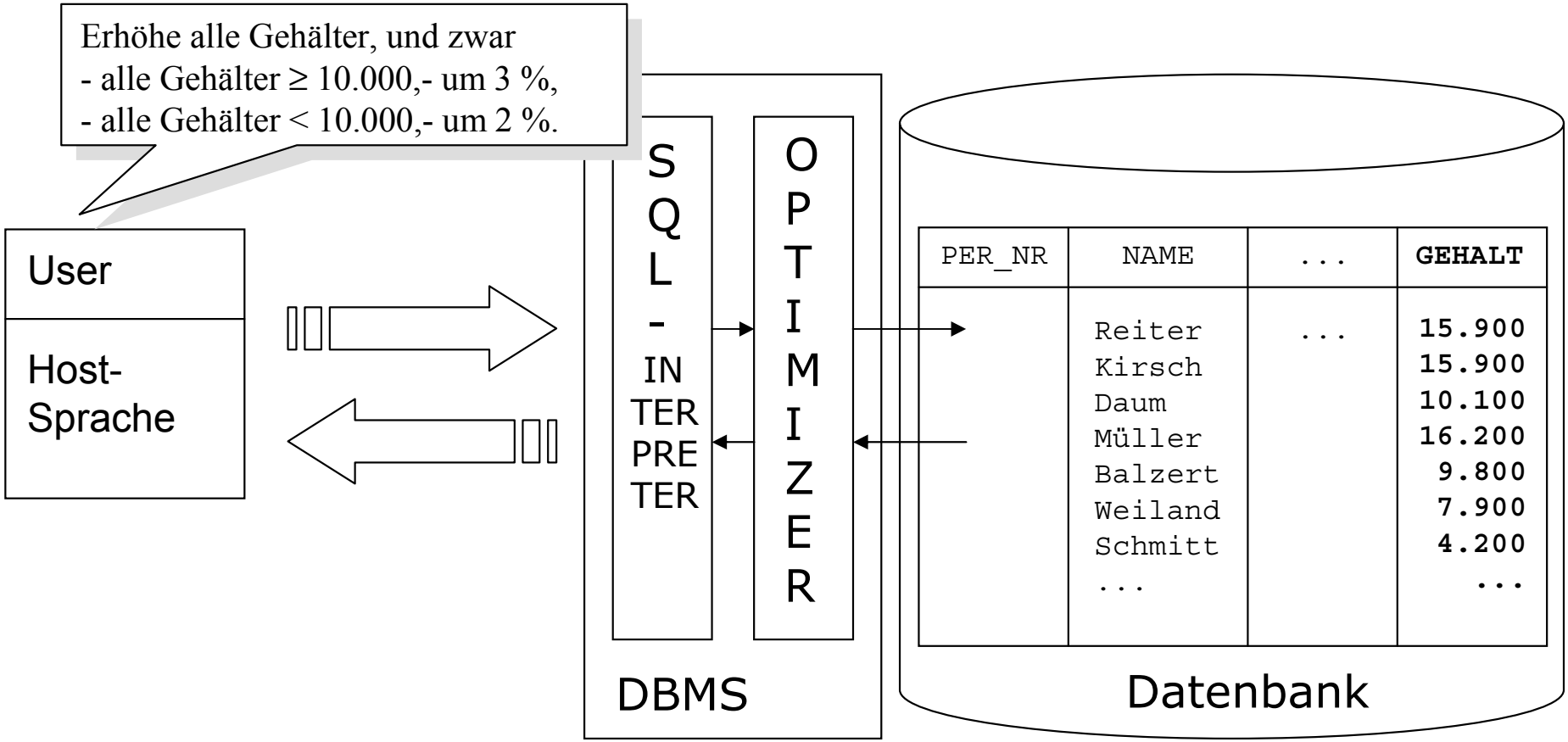
- Neben den **statischen Daten zur Datenbankstruktur** werden im Systemkatalog auch **dynamisch statistische Daten zu den Datenbankinhalten** gespeichert, wie z.B. Anzahl der Zeilen pro Tabelle Maximale und minimale Ausprägung der Spalten jeder Tabelle, etc.

- Diese statistischen Daten werden insbesondere zur Ermittlung des optimalen Anfrageweges durch den Optimizer benutzt.

- Statistische Daten werden i.d.R. wegen der erheblichen Belastung des Systems nicht automatisch nach jedem Schreibvorgang, sondern z.B. im Rahmen der Nachtverarbeitung durch explizit gestartete Prozesse aktualisiert (z.B. UPDATE STATISTICS).

6.1.3 Embedded SQL / PL/SQL / SQLJ

- Die DSL SQL entspricht als mengen-orientierte DB-Sprache anderen Sprachkonzepten als imperative, höhere Programmiersprachen (Host-Sprachen), die Deklarationsblöcke und Kontrollstrukturen (Sequenzen, bedingte Anweisungen und Schleifen) enthalten.



6.1.3 Embedded SQL

- Man spricht von **Embedded SQL - ESQL (eingebettetes SQL)**, wenn Techniken zur Verfügung stehen, SQL-Anweisungen in die Syntax imperativer Programmiersprachen zu integrieren.

- Die Einbettung von SQL in eine (datenbank-unabhängige) höhere Programmiersprache wie C, Pascal, C++, Java etc. erfolgt durch Integration von SQL-Anweisungen in die Syntax der entsprechenden Sprache im Quellcode.

- Im Zusammenhang mit ESQL unterscheidet man
  - **statisches Embedding** und
  - **dynamisches Embedding.**

- Ein Block von SQL-Anweisungen wird durch jeweils durch die Schlüsselworte **exec sql** eingeleitet und endet mit dem Terminatorsymbol der entsprechenden Host-Sprache.

6.1.3 Embedded SQL – statisches Embedding

- Ein Pre-Compiler wandelt die jeweiligen SQL-Blöcke in Prozeduraufrufe der entsprechenden Host-Sprache um, so dass das Programm anschließend durch den gewöhnlichen Compiler für die Host-Sprache übersetzt wird.

```
void main( ) {
   ...
   exec sql update eteil
            set e_mng=e_mng-10
            where e_nr=10122;
   ...
}
```

- Beim statischen Embedding kann bereits beim Compilieren die Syntax der SQL-Abfrage geprüft werden. Dieses Konzept erfordert die Installation DBMS-spezifischer Libraries für den Compiler.

6.1.3 Embedded SQL – Verwendung von Hostvariablen im SQL-Statement (1)

- ESQL-Statements können Host-Variablen referenzieren. Diesen muss ein **Doppelpunkt** vorangestellt sein, um sie von SQL-Spaltennamen unterscheiden zu können. Solche Host-Variablen müssen in einem Block deklariert sein, der durch **begin declare section** eingeleitet und durch **end declare section** abgeschlossen wird:

```
void main( ) {
   ...
   exec sql begin declare section;
        dcl e_nr     int;
        dcl sqlstate char(5);          } *)
   exec sql end declare section;
   e_nr = 10122;
   exec sql update eteil
            set e_mng=e_mng-10
            where e_nr=:e_nr;
            if sqlsate = '00000'
               ... ;
   ...
}
```

*) Die **Datentypen** für die Host-Variablen können dem SQL-Typsystem entsprechen.
Der Pre-Compiler generiert dann aus ihnen die entsprechenden Typangaben der verwendeten Host-Sprache.

6.1.3 Embedded SQL – Verwendung von Hostvariablen im SQL-Statement (2)

```
void main( ) {
   ...
         if    sqlsate = '00000'
               ...
         else ...;
   ...
}
```

- Die Host-Variable **sqlstate** sollte in jedem ESQL-Programm enthalten sein. Diese Variable enthält einen Statuscode nach jedem exec-SQL-Statement:

  '00000': erfolgreiche Ausführung des Statements

  '02000': ausgeführt, aber keine Daten gefunden, die die Anfrage erfüllen, etc.

- Host- und SQL-Datentypen müssen kompatibel sein. Host-Variable und SQL-Spalten können den gleichen Namen haben.

6.1.3 Embedded SQL – dynamisches Embedding

- Die Konstruktion von SQL-Anweisungen zur Laufzeit ist möglich, da SQL-Anfragen der Host-Sprache Konstruktion von SQL-Anweisungen als manipulierbare Zeichenketten übergeben werden können.

```
exec sql begin declare section;
    dcl Anfrage String char(256) varying;
    dcl AnfrageObjekt  statement;
exec sql end declare section;
int anz = 10;
AnfrageString = 'update eteil set e_mgn=e_mng-' || anz ||
                ' where e_nr=10122';
...
// interne Optimierung
exec sql prepare AnfrageObjekt from :AnfrageString;

// Ausführung des optimierten Ablaufplans
exec sql execute AnfrageObjekt;
```

- Die beiden aufeinanderfolgenden Anweisungen **prepare** und **execute** können zu der Anweisung **execute immediate** zusammengefasst werden.
- Welches sind die Vor- und Nachteile des dynamischen Embedding gegenüber dem statischen Embedding?

6.1.3 Embedded SQL – das Cursor-Konzept (1)

- Jedes interaktive SQL-Statement (DDL und DML) kann in ESQL verwendet werden. Darüber hinaus gibt es in ESQL spezielle Konzepte, die die zeilenweise Verarbeitung von Ergebnistabellen einer SQL-Abfrage ermöglichen.

- I.d.R. können solche Tabellen von Host-Sprachen nicht in einem einzelnen Zugriff verwaltet werden. Vielmehr besteht das Bedürfnis, einzelne Zeilen einer Tabelle entsprechend einzelner Records in einer Datei zu lesen.

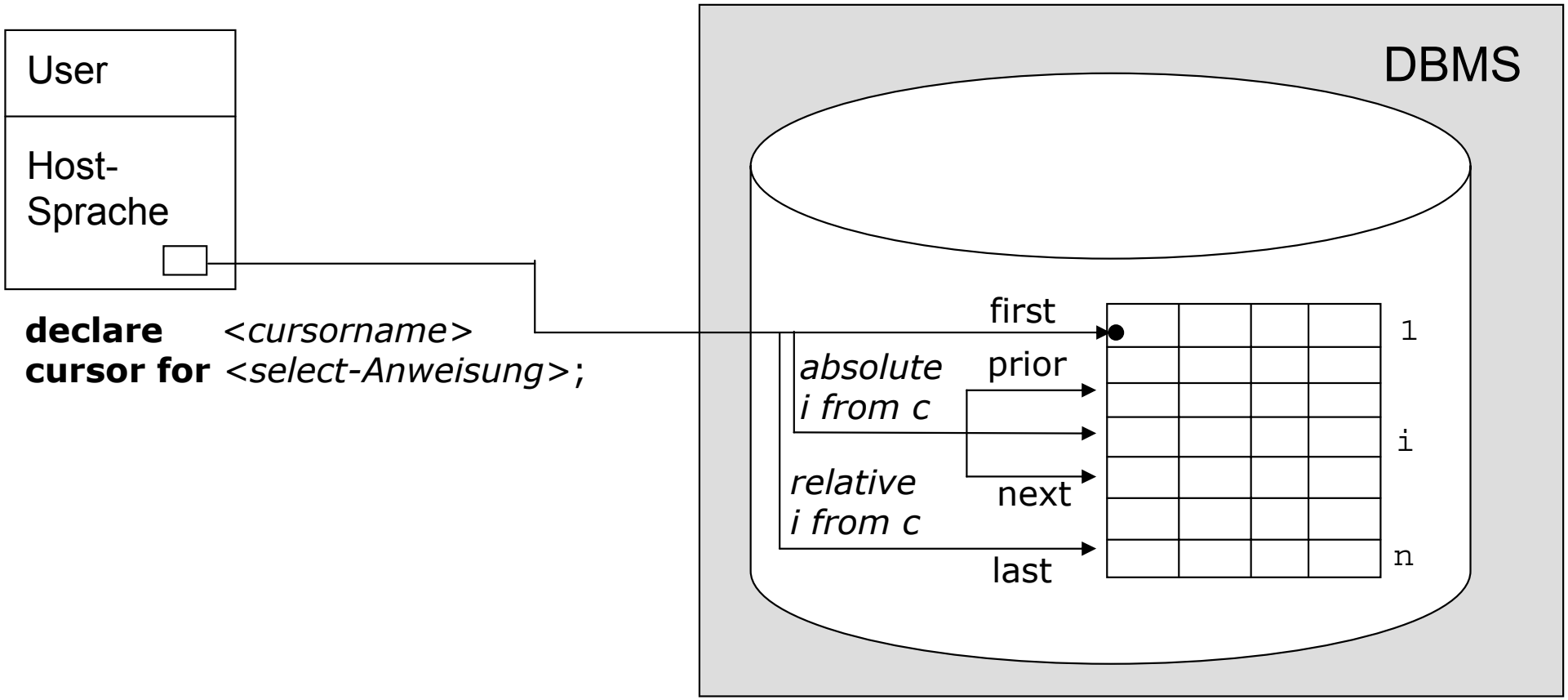
- Hierfür wird - in Analogie zum Dateicursor - ein Cursor definiert, der die zeilenweise Verarbeitung einer Ergebnistabelle ermöglicht.

- **Cursor zum Lesen deklarieren**:

```
exec sql declare [scroll] <cursorname> cursor
                                for <select-Anweisung>;
```

  Die Option scroll ermöglicht beliebiges Navigieren in der fetch-Anweisung (s.u.).

6.1.3 Embedded SQL – das Cursor-Konzept (2)



- Die select-Anweisung definiert die Ergebnistabelle, auf der mit Hilfe der cursor-Variablen eine zeilenweise Bearbeitung möglich ist.

- **Cursor öffnen**

```
exec sql open <cursorname>;
```

- **Zeilen lesen** zum Weiterverarbeiten

```
exec sql fetch <cursername>
            into :<a1>, :<a2>;
do while sqlstate = '00000'
   <Anweisungen>
   exec sql fetch [position] <cursorname>
               into :<a1>, :<a2>;
end;
```

Als position kann optional eine der Spezifizierungen auf Folie 22 angegeben werden.

- **Cursor zum Update deklarieren**

```
exec sql declare [scroll] <cursorname> cursor
        for <select-Anweisung>
        for update of <col1, col2, ...>;
... open  ...
... fetch ...
exec sql update <Tabelle> set <update>
          where current of <cursorname>;
```

- **Cursor schließen**

```
exec sql close <cursorname>;
```

6.1.3 PL/SQL

Nachteile von ESQL in höheren (DB-unabhängigen) Programmiersprachen:

- Das Anwendungsprogramm wechselt häufig in „kleinsten Einheiten“ zwischen Host-Sprache und SQL-Anweisung.
- Der Optimizer (als „Kernstück“ des DBMS) kann nur den Bereich einzelner SQL-Anweisungen überschauen, da er die darüber hinausgehenden Ablaufstrukturen nicht kennt.

Lösungen

- Zahlreiche große relationale DB-Systeme bieten eine Erweiterung von SQL mit Konzepten prozeduraler Sprachen an. Diese werden in der Regel **PL/SQL** (**P**rocedural **L**anguage/SQL) genannt. Diese Sprache wird von den meisten DBMS auch zur Programmierung von **Stored Procedures und Triggern** verwendet (vgl. 6.2 und Praktikumsaufgabe 4).
- PL/SQL ist im SQL-99-Standard als prozedurale Spracherweiterung von SQL vorgesehen.

Eine Einführung in die Syntax von PL/SQL-Oracle enthält die Praktikumsaufgabe 4 als Anlage.

6.1.3 SQLJ

- **JDBC** als objektorientierte Schnittstelle aus Java zu relationalen Datenbanken unterstützt dynamisches SQL (vgl. 6.3). Dies hat den Nachteil, dass Laufzeitfehler, die vom DB-System zurückgegeben werden, erst zur Ausführungszeit erkannt werden und nicht abgefangen werden können.

- Mit **SQLJ** wurde ein Standard entwickelt, der die direkte Einbettung von SQL-Anweisungen in Java-Code ermöglicht.

- Die entsprechenden **Java-Spezifikationen** gliedern sich in drei Teile:
  1. Embedded SQL für Java (Part 0)
  2. Java Storede Procedures (Part 1) (vgl. auch 6.3)
  3. Java-Klassen für benutzer-definierte SQL-Datentypen (Part2)

- Die Firmen IBM (DB2/Informix), Oracle und Sybase und setzen diesen Standard bereits ein.

6.1.3 Beispiel für einen anonymen PL/SQL-Block

```
-- Deklarationsblock
declare
   cursor c is select gehalt from angestellter
               for update of gehalt;
   v_gehalt angestellter.gehalt%type;

-- Ausführungsblock
begin
   -- Cursor öffnen
   open c;

   -- erste Zeile lesen
   fetch c into v_gehalt;

   -- alle Zeilen der Ergebnistabelle lesen
   while c%found loop
    if   v_gehalt > 10000
    then update angestellter set gehalt = gehalt * 1.03
                where current of c;
    else update angestellter set gehalt = gehalt * 1.02
                where current of c;
    end if;
    fetch c into v_gehalt;
   end loop;

end;
/
```

Tabelle ANGESTELLTER

PER_NR	NAME	...	GEHALT
1001001	Reiter	...	15.900
1001002	Kirsch		15.900
1001003	Daum		10.100
1004008	Müller		16.200
1005013	Balzert		9.800
1005112	Weiland		7.900
9001011	Schmitt		4.200
...	...		...

GEHALT = 16.377

Der PL/SQL-Block soll bei Ausführung die Gehälter erhöhen, und zwar
- alle Gehälter ≥ 10.000,-- um 3 %,
- alle Gehälter < 10.000,-- um 2 %.

6.1.3 Beispiel für einen anonymen PL/SQL-Block

1. **Cursorstruktur**: Spalte GEHALT
2. **PL/SQL-Variable**: v-gehalt
3. Der Cursor wird als **Updatecursor** deklariert.
4. Beim **open** wird das select ausgeführt.
5. Die bool‘sche Variable **c%found** ist true, falls die vorhergehende fetch-Anweisung Daten gelesen hat.
6. Mit **... current of c** wird der gerade gelesene Datensatz aktualisiert.