HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Datenbanksysteme 2
Einführung

Frühjahrssemester 2010

Prof. S. Keller

Datenbanken?

- Wer hat in der Zwischenzeit – oder schon immer – mit gearbeitet?
- Was ist der Unterschied Datenbank und DBMS? Was heisst Dbs?
  - "Merkformel": DBS = DBMS + n * DB
    - Datenbanksystem
    - Datenbankmanagementsystem: anwendungsunabhängige Dienste
    - Datenbank: anwendungsspezifische Informationen
- Wichtigste Aufgaben und Funktionen eines DBMS:
  - Schemaverwaltung
  - Anfragebearbeitung
  - Transaktionsverwaltung
  - Zugriffsschutz

Repetitorium: Inhalt Dbs1

- Begriffe
- Datenmodellierung
- Relationales Modell
- DB-Design
- SQL-DML (Joins, Subqueries)
- Views
- SQL als DCL
- Transaktionen
- DB-Programmierung
- Interne Ebene und DB-Optimierung
- Back Up und Recovery

Begriffe

Tabelle oder Relation?

- Praxis : Betonung auf opt. Darstellung in Tabellenform
- Wissenschaft: Betonung der mathematischen Grundlage als Relation

Unterschiede zwischen beiden Bezeichnungen:

Tabelle:
2-dim. Zeilen/Sp.
geordnet
doppelte Zeilen

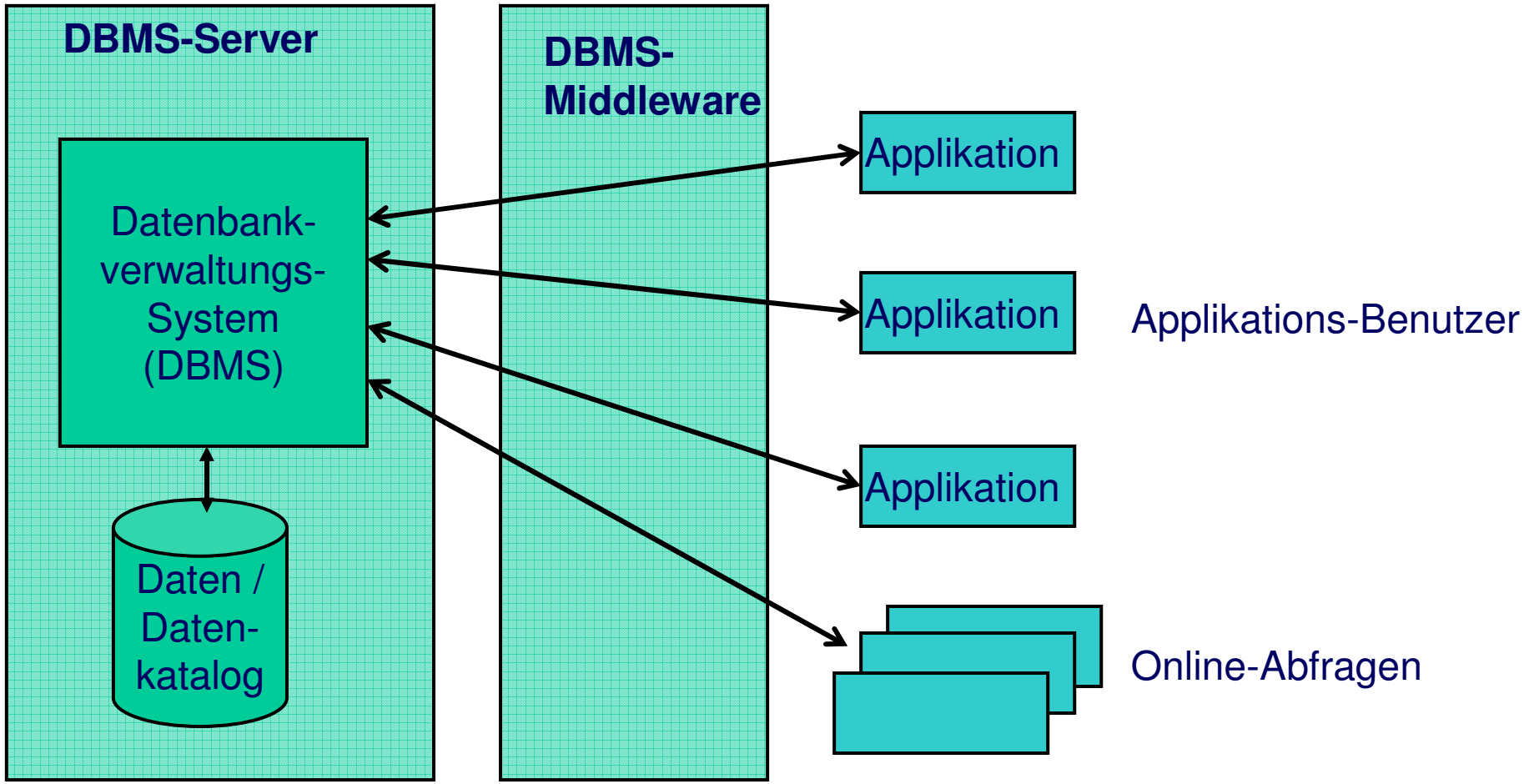
Relation:
Menge von Tupeln
ungeordnet
duplikatfrei

aber:

- Relationen lassen sich in Tabellenform visualisieren, wenn man eine bestimmte, Anordnung wählt.
- Tabellen lassen sich als Relationen formalisieren, wenn man keine Duplikate, zulässt und von der Anordnung absieht.

UML-Klassen-diagramm	**Entity Relation-ship Model**	**Relationales Datenmodell**	**Daten-bank**	**Access**
Objekt, Instanz	Entität	Tupel	Zeile	Datensatz
Objektmenge, Instanz-menge, Klasse	Entitäts-menge	Relation	Tabelle	Datenblatt
Attribut	Attribut (Property)	Attribut	Feld / Spalte / Attribut / Column	Feldbe-zeichner
Datentyp	Wertebereich	Wertebereich	Datentyp	Felddaten-typ

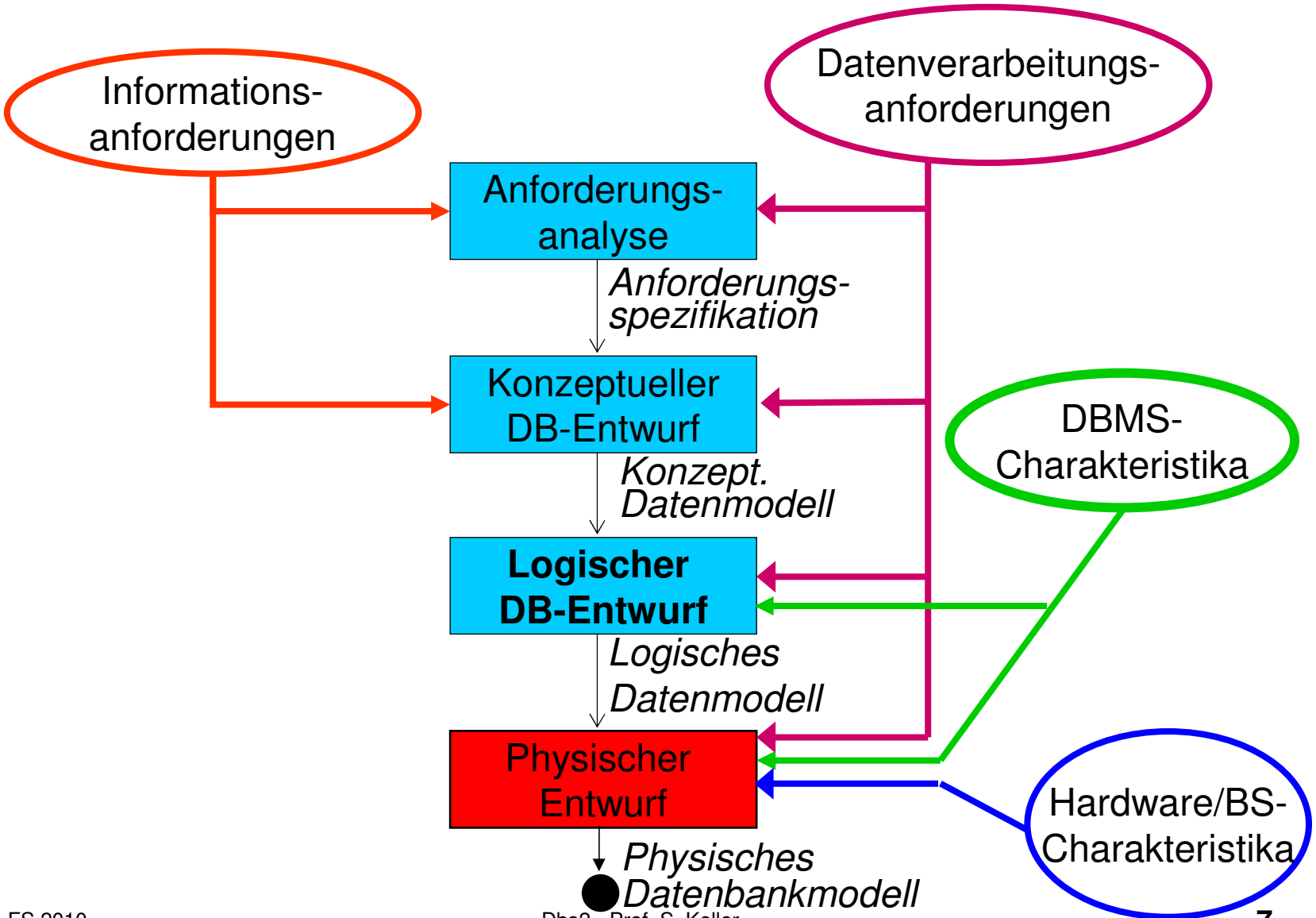
Architekturen: Client-Server



Weitere Architekturen

- Weitere Architekturen (nach Heuer & Saake)
  - Verteilte DB
  - Integrierte DB
  - Föderierte DB
- Aktuelle Begriffe: Kennen Sie…
  - In Memory-DB, Pure Java-DB, Webdatenbanken, Persistenzframeworks

DB-Entwurfsprozess

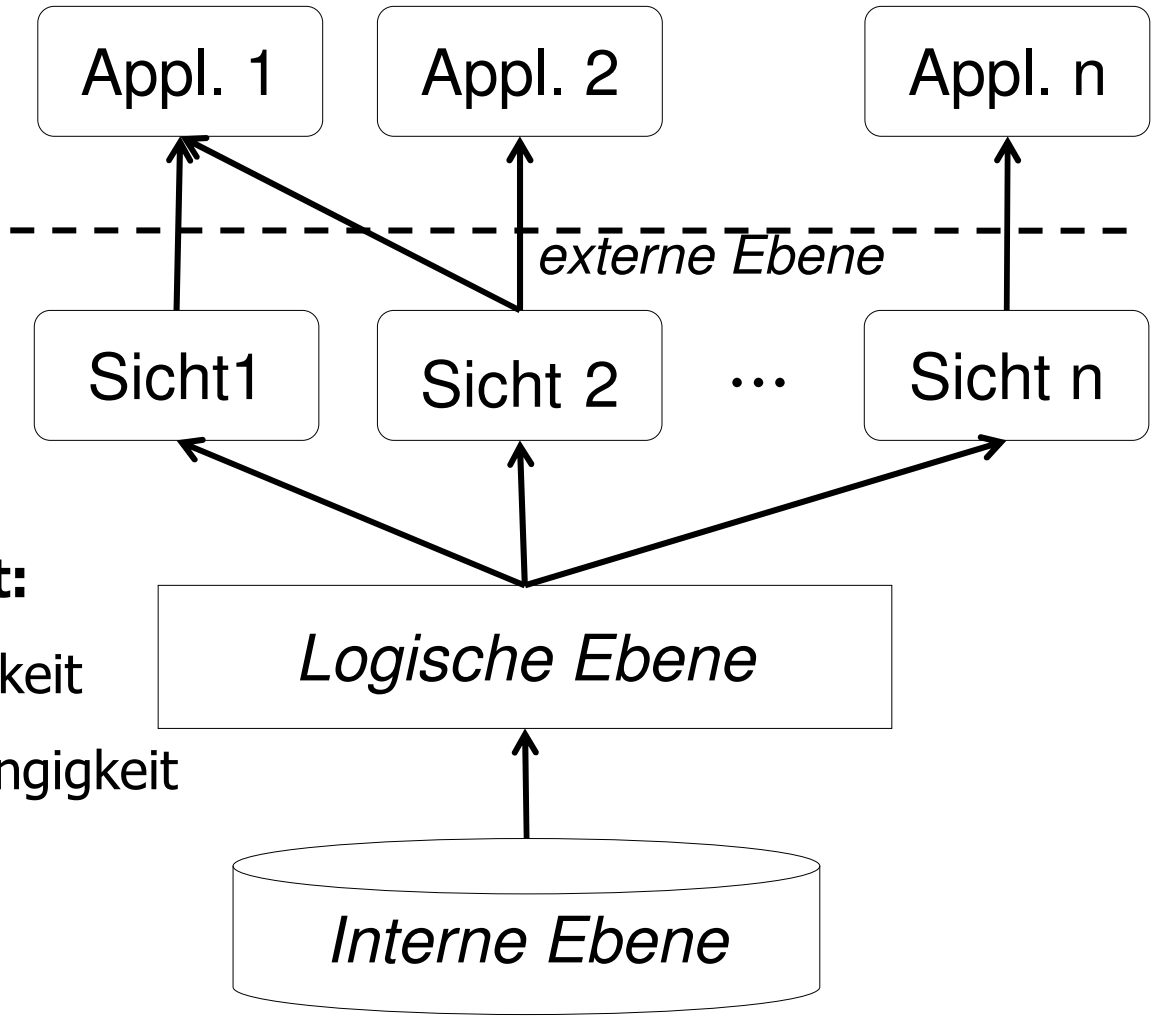


Datenbankparadigmen

- Paradigma/Modell (UML 2: Meta-Modell)
  - legt die Objekt-/Daten-Strukturierungs-Konzepte fest

- Konkrete Systeme unterstützen eines der folgenden Paradigmen:
  - Hierarchisches Datenbank-Paradigma (-Modell)
  - Netzwerk-Paradigma (-Modell)
  - Relationen-Paradigma (-Modell)
  - Postrelationale Paradigmen (-Modelle)
    - Objektrelationales Paradigma (-Modell)
    - Objektorientiertes Paradigma (-Modell)

3-Ebenen-Modell

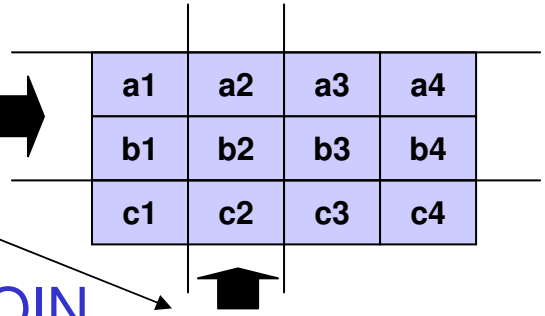


Datenunabhängigkeit:

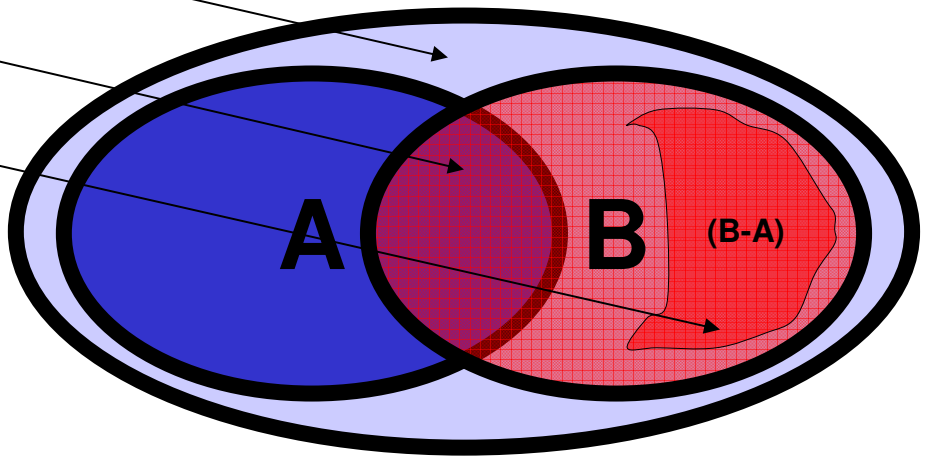
- physische Unabhängigkeit
- logische Datenunabhängigkeit

Repetitorium: Operationen und SQL

- Selektion => WHERE-Klausel
- Projektion => "SELECT *Skalar*"
- Produkt (kartesisches Produkt)
  - Join => WHERE, INNER/OUTER JOIN
- Union => (SELECT ...) UNION (SELECT ...)
- Intersection
- Difference



[... soweit wir diese hier behandeln]



HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
INFORMATIK

Repetitorium: SQL-DML-Syntax

Vor allem: `SELECT, UPDATE, INSERT, DELETE`

Syntax (vereinfacht):

```
SelectStmt  =
  SELECT [ DISTINCT ]
    ( '*' | ColumnName [ AS ] ColumnAliasName
      { ',' ColumnName [ AS ] ColumnAliasName  } )
    FROM TableName [ TableAliasName ]
    [ WHERE-Predicate ]
  [ ORDER BY ColumnName [ ASC | DESC ]
    { ',' ColumnName [ ASC | DESC ] }
  ]
```

Bemerkungen:

- Anstelle Table können auch Views stehen (zu Views später)
- Bemerkungen ff. nächste Folie(n)

R.: SQL-DML-Syntax (ff.)

- '*' steht für alle Spaltennamen einer Tabelle.
- 'ORDER BY' steuert die Sortierreihenfolge; es dürfen nur Spalten aufgeführt sein, die im SELECT-Teil spezifiziert wurden.
- Mit ColumnAliasName können die Attributnamen der Resultattabelle verändert werden
- TableAliasNamen sind notwendig bei Equi-Joins
- SQL unterscheidet

Repetitorium: Sichten

- Verschiedene Ziele von Sichten:
  - zur Gewährleistung der Datenunabhängigkeit
  - für den Datenschutz
  - für die Vereinfachung von Anfragen
- Änderbarkeit von Sichten
  - (einschränkende) Bedingungen von SQL:
    - nur eine Basisrelation
    - Schlüssel muss vorhanden sein
    - keine Aggregatfunktionen, Gruppierung und Duplikateliminierung
- Bemerkung:
  - Sichten werden von OO oft unterschätzt!

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
INFORMATIK

Repetitorium: "Gute Praktiken"

Beispiel:

```
SELECT MIN( Bezeichnung ) AS [Projektbez],
       COUNT( Name ) AS [Mitarbeitername],
       SUM( Zeitanteil ) AS [Zeitsumme]
FROM Angestellter, Projektzuteilung, Projekt
WHERE Angestellter.Persnr = Projektzuteilung.Persnr
  AND Projekt.Projnr = Projektzuteilung.Projnr
GROUP BY Projekt.Projnr;
```

Schreibregeln zur besseren Lesbarkeit und Portabilität:

- Gross-/Kleinschreibung verwenden, z.B. `FROM Projektzuteilung`
- Leerstellen bei Klammern und Vergleichsoperator, z.B. `SUM( Zeitanteil )` (Ausnahme: `COUNT(*)` ?)
- keine krypt. Abk. (:->) => selbstdokumentierende Namen, z.B. `Mitarbeitername` (Ausnahmen: `MA, viele Tabellen` ?)
- qualifizierte Namen verwenden, z.B. `Projekt.Projnr`

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Einige Unterschiede PostgreSQL - Oracle

Datenbanksysteme 2

Frühjahrssemester 2010

Prof. S. Keller

Oracle vs. PostgreSQL

- PL/pgSQL ist ähnlich mit PL/SQL. Beide haben eine Blockstruktur, sind imperativ und Variablen müssen deklariert werden. Zuweisungen, Schleifen und Auswahlanweisungen sind ähnlich. Hauptunterschiede sind:
- Ora kennt Vorgabewerte für Parameter, im Ggs.zu PostgreSQL.
- In PostgreSQL können dafür Fn. überladen werden (als Lösung für Vorgabewerte)
- Im Ggs . zu PL/pgSQL verlangt PL/SQL Cursor; man kann Anfrage nicht mehr einfach in die FOR-Anweisung schreiben.
- mehrfache ' Apostrophe (PostgreSQL) durch einfache ersetzen
- Anstelle von Packages Schemas verwenden, um Ihre Funktionen in Gruppen zu organisieren.
- PL/pgSQL kennt nur Funktionen; Ora Fns. und Prozeduren
- Datentypen…

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
INFORMATIK

Oracle vs. PostgreSQL (SQL-99)

Feature	PostgreSQL	Oracle
LOBs	nur BLOBs	Ja
Felder	Ja	Ja
Distinct Types	Nein	Ja
Benutzerdefinierte Basistypen	Ja	Ja
Strukturierte Typen	Eingeschränkt	Ja
Funktionen	Ja	Ja
Methoden	Nein	Ja
Vererbung	Ja (auch Mehrfachvererbung)	Ja
OIDs	Ja	Ja
Direkte Navigation	Ja	Ja
Typbibliotheken	keine standardisierte Schnittstelle aber prinzipiell möglich	Ja (Cartridges)

Tabelle 1: Unterstützung von SQL-99

Quelle: Seminararbeit Wirtschaftsinformatik, Universität Mannheim, Juni 2003 (Oracle 9, Postgres 7.3, http://www.wifo.uni-mannheim.de)

ORDBMS: Bedeutung?

- Marktanteil:
  - Oracle Anzahl verkaufte Lizenzen 39,4% Marktanteil, gefolgt von IBM (33,6%) und Microsoft (11,1%) (IDC 2003). PostgreSQL: Nicht einfach zu bestimmen, da FOSS! Teil von Linux-Distributionen wie RedHat oder SuSE.
  - Vergleich PostgreSQL vs. ORACLE

- Einer der Hauptvorteile (nochmals):
  - Multivalue-Types/Enumerators (DOMAIN…),
  - Collections (SET/LIST/ARRAY)
  - und UDT/ADT (CREATE TYPE…)

- Siehe auch CERN, 2006 (http://dcdbappl1.cern.ch...)

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

DB-Integrität, insbes. Konsistenzbedingungen

Datenbanksysteme 2

Prof. S. Keller

Überblick

Datenintegrität insbes. Konsistenzbedingungen in DB

- Datenintegrität
- Begriffe und Konzepte
- Klassifikation der Konsistenzbedingungen
- Konsistenzbedingungen in SQL
- Konsistenzbedingungen in Oracle

Was kennen Sie/wir bisher?

ACID

- **Atomicity** ("Alles oder Nichts"-Regel):
  - Commit: alle Änderungen werden vollständig in der Datenbank gespeichert.
  - Rollback: nichts wird verändert.
- **Consistency** (Konsistenzerhaltung):
  - Die Transaktion bringt die DB von einem konsistenten Zustand DB' in einen anderen konsistenten Zustand DB".
- **Isolation** (Isolierter Ablauf, Independence)
  - Eine Transaktion sieht keine Zwischenresultate einer anderen parallel ablaufenden Transaktion.
- **Durability** (Dauerhaftigkeit, Persistenz):
  - Änderungen einer Transaktion gehen auch bei nachfolgenden Fehlersituationen (z.B. SW- und HW-Fehler, Disk Crash etc.) nie verloren gehen.

Anforderungen an die Datenintegrität

- Explizite:
  - … folgen
- Implizite:
  - Schlüssel
  - Kardinalitäten
  - Jedes Entity eines Untertyps muss in Obertyp enthalten sein
  - Domäne für jedes Attribut
  - Referentielle Integrität

Konsistenzbedingungen: Einleitung

Überblick

- Klassifizierung / Kriterien
- Konsistenzbedingungen in SQL
- Referentielle Integrität
- Triggers und Stored Procedures

Nochmals: Die Konsistenzbedingungen definieren Regeln, denen die Daten genügen müssen.

Konsistenzbedingungen: Beispiel

Beispiele aus der Angestellten-Projekt Datenbank:

K1: Der Wertebereich der gespeicherten Telefonnummern ist zwischen 100 und 500.

– Diese K. bedingt, dass beim Einfügen und Ändern von Daten die Werte der Attribute auf die Einhaltung des Wertebereiches überprüft werden müssen.

- Eine kompliziertere Konsistenzbedingung:

  K2: Die Tabelle 'Abteilung' wird um ein Attribut 'Salärsumme' erweitert. Die Salärsumme einer Abteilung ist gleich der Summe der Saläre der Angestellten dieser Abteilung.

- Noch komplexer ist die folgende Konsistenzbedingung:

  K3: Jeder Angestellte muss in mindestens einem Projekt mitarbeiten. Mindestens 60% seiner Arbeitszeit muss projektbezogen sein.

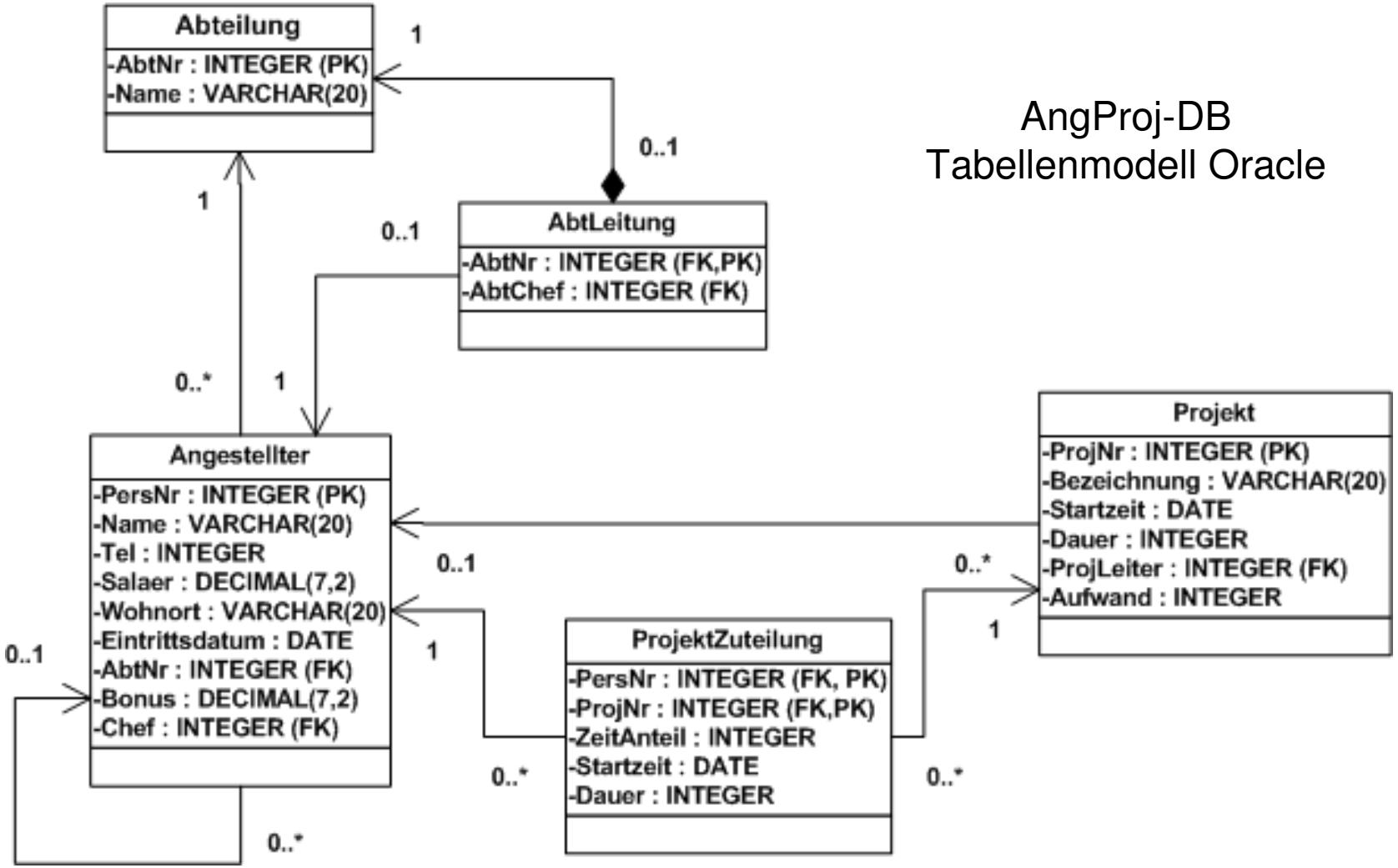
  – Die Überprüfung dieser K. bedingt das Traversieren der Datenbank von der 'Angestellten'- zur 'ProjektZuteilungs'- und 'Projekt'-Tabelle.

Konsistenzbedingungen: Beispiel (ff.)

Diskussion

- Die Konsistenzbedingung K1 kann sofort nach einer elementaren Änderungsoperation überprüft werden.
- K2 wird nach einer Saläränderung temporär verletzt sein:
  - Zuerst werden die Saläre einzelner Mitarbeiter erhöht. Dann wird die neue Salärsumme pro Abteilung berechnet und der Wert des Attributes 'SalaerSumme' entsprechend gesetzt. Zwischen dem Verändern der Angestellten-Saläre und dem Schreiben der neuen Salärsumme ist die Bedingung K2 temporär verletzt.
- Die Bedingung K3 kann sogar erst überprüft werden, wenn die Datenbank aufgebaut ist:
  - Zuerst werden Angestellte und Projekte eingefügt, erst dann können die Projektzuordnungen definiert werden.

AngProj-DB Tabellenmodell Oracle



AngProj-DB
Tabellenmodell Oracle

Klassifikationen / Kriterien

Primäre- und sekundäre Konsistenzbedingung:

- Beispiel:
  - K1 ist eine primäre,
  - K2 ist eine sekundäre Konsistenzbedingung.
- D.h. Klassifikation nach Kriterium "Zeitpunkt der Überprüfung":
  - Eine K. heisst primär, wenn sie nach jeder elementaren Datenbankoperation (insert, delete, modify) auf einem Tupel einer Tabelle (oder View) erfüllt sein muss.
  - Eine K. heisst sekundär, wenn für einen Übergang von einem konsistenten Zustand zu einem neuen konsistenten Zustand mehrere elementare Datenbankoperationen notwendig sind. Zwischen den einzelnen Operationen sind Inkonsistenzen möglich, die Konsistenz wird erst nach Ausführung aller Operationen überprüft.

Klassifikationen / Kriterien (II)

Starke und schwache Konsistenzbedingungen

- Beispiel:
  - K1 und K2 sind starke Konsistenzbedingungen
  - Die Bedingung K3 ist eine schwache K.:
    - Wenn z.B. ein Mitarbeiter neu eingestellt wird, dann ist er ev. noch keinem Projekt zugeordnet. Oder: Firma hat zuwenig Projekte um alle Mitarbeiter gemäss der Regel K3 auszulasten.
- Fehlerreaktion beim Verletzen der Konsistenzbedingung:
  - Starke (oder strenge) Konsistenzbedingungen müssen immer eingehalten werden
  - Schwache K. sind im Normalfall einzuhalten, können aber in Ausnahmefällen oder für eine gewisse Zeit umgangen werden.
    - Wenn eine Transaktion eine schwache Bedingung verletzt, dann nicht abgebrochen (nur Warnung). Ausserdem muss es möglich sein, schwache K. auf Benutzerwunsch hin zu überprüfen. (Beispiel: Der Benutzer verlangt eine Liste der Angestellten, welche K3 nicht erfüllen)

Klassifikationen / Kriterien (III)

Wirkungsbereich einer Konsistenzbedingung

- Kriterium zur Klassifizierung ist die Überprüfung der einzubeziehenden Datenobjekte:
  - Einzelnes Attribut: Wertebereich und Datentyp – Überprüfung
  - Einzelnes Tupel: Zusammenhänge zwischen den Attributwerten.
  - Einzelne Tabelle: Überprüfen von Eindeutigkeitsbedingungen (z.B. Eindeutigkeit eines Schlüsselwertes).
  - Mehrere Tabellen: Überprüfen von referentiellen Integritätsbedingungen

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Klassifikationen / Kriterien: Zusammenfassung

Man kann zwischen folgenden Arten von Konsistenzbedingungen unterscheiden:

- Solchen, die sich auf ein einzelnes Objekt beziehen. Diese werden weiter untergliedert in Bedingungen, die jedes Objekt einer Klasse erfüllen muss ("harte" Konsistenzbedingungen), und Bedingungen, deren Verletzung zwar an sich möglich, aber selten ist ("weiche" Konsistenzbedingungen).
- Solchen, welche die Eindeutigkeit von Attributkombinationen aller Objekte einer Klasse fordern.
- Solchen, welche die Existenz eines Attributwertes in einer Instanz einer anderen Klasse fordern.
- Kompliziertere Bedingungen, die sich auf Objektmengen beziehen und mittels Sichten formuliert werden.

Zusammenfassung

- Transaktion
  - Operationen auf Transaktionsebene
  - ACID
- Transaktionen in SQL
  - Datenbank-Verbindungen
- Klassifizierung der Konsistenzbedingungen
  - Primäre- und Sekundäre Konsistenzbedingungen:
  - Starke und schwache Konsistenzbedingungen
  - Wirkungsbereich einer Konsistenzbedingung
- Konsistenzbedingungen in SQL
  - Bedingung für einzelne Attributwerte
  - Bedingung für Attributwerte aller Tupel einer Tabelle:
  - Bedingungen für mehrere Tabellen: Referentielle Integrität
  - Referentielle Integrität in SQL

Konsistenzbedingungen in Oracle

Datenbanksysteme 2

Prof. S. Keller

Zusammenfassung

- Transaktion
  - Operationen auf Transaktionsebene
  - ACID
- Transaktionen in SQL
  - Datenbank-Verbindungen
  - Formulierung von Transaktionen in SQL
- Klassifizierung der Konsistenzbedingungen
  - Primäre- und Sekundäre Konsistenzbedingungen:
  - Starke und schwache Konsistenzbedingungen
  - Wirkungsbereich einer Konsistenzbedingung
- Konsistenzbedingungen in SQL
  - Bedingung für einzelne Attributwerte
  - Bedingung für Attributwerte aller Tupel einer Tabelle:
  - Bedingungen für mehrere Tabellen: Referentielle Integrität
  - Referentielle Integrität in SQL

Constraints in Oracle (Source: Ausg. 2000)

- Name a constraint or the Oracle Server will generate a name by using the SYS_C*n* format.
- Create a constraint:
  - At the same time as the table is created
  - After the table has been created
- Define a constraint at the column or table level.
- Constraints:
  - PRIMARY KEY
  - UNIQUE
  - NOT NULL
  - CHECK
  - FOREIGN KEY ... ON DELETE CASCADE

Defining Constraints

```
CREATE TABLE [schema.]table
             (column datatype [DEFAULT expr]
              [column_constraint],
             …
              [table_constraint]);
```

```
CREATE TABLE emp(
             (empno   NUMBER(4),
              ename   VARCHAR2(10),
              …
              deptno   NUMBER(7,2) NOT NULL,
              CONSTRAINT emp_empno_pk
                           PRIMARY KEY (EMPNO));
```

Example

```
SQL> CREATE TABLE dept(
  2       deptno    NUMBER(2),
  3       dname     VARCHAR2(14),
  4       loc       VARCHAR2(13),
  5       CONSTRAINT dept_dname_uk UNIQUE (dname),
  6       CONSTRAINT dept_deptno_pk PRIMARY KEY (deptno));
```

defined at table level

```
SQL> CREATE TABLE emp(
  2       empno     NUMBER(4),
  3       ename     VARCHAR2(10) NOT NULL,
  4       job       VARCHAR2(9),
  5       mgr       NUMBER(4),
  6       hiredate  DATE,
  7       sal       NUMBER(7,2),
  8       comm      NUMBER(7,2),
  9       deptno    NUMBER(7,2) NOT NULL,
 10       CONSTRAINT emp_deptno_fk FOREIGN KEY (deptno)
 11                 REFERENCES dept (deptno));
```

defined at column level

Adding a Constraint

```
ALTER TABLE table
ADD [CONSTRAINT constraint] type (column);
```

- Add or drop, but not modify, a constraint
- Enable or disable constraints
- Add a NOT NULL constraint by using the MODIFY clause

– Example

- Add a FOREIGN KEY constraint to the EMP table indicating that a manager must already exist as a valid

```
SQL> ALTER TABLE     emp
  2  ADD CONSTRAINT emp_mgr_fk
  3            FOREIGN KEY(mgr) REFERENCES emp(empno);
Table altered.
```

Dropping a Constraint

– Remove the manager constraint from the EMP table.

```
SQL> ALTER TABLE           emp
  2  DROP CONSTRAINTemp_mgr_fk;
Table altered.
```

- Remove the PRIMARY KEY constraint on the DEPT table and drop the associated FOREIGN KEY constraint on the EMP.DEPTNO column.

```
SQL> ALTER TABLE           dept
  2  DROP PRIMARY KEY CASCADE;
Table altered.
```

Disabling and Enabling Constraints

- Execute the DISABLE clause of the ALTER TABLE statement to deactivate an integrity constraint.
- Apply the CASCADE option to disable dependent integrity constraints.

```
SQL> ALTER TABLE          emp
  2  DISABLE CONSTRAINT   emp_empno_pk CASCADE;
Table altered.
```

Activate an integrity constraint currently disabled in the table definition by using the ENABLE clause.

```
SQL> ALTER TABLE          emp
  2  ENABLE CONSTRAINT    emp_empno_pk;
Table altered.
```

A UNIQUE or PRIMARY KEY index is automatically created if you enable a UNIQUE or PRIMARY KEY constraint.

Viewing Constraints

- Query the USER_CONSTRAINTS table to view all constraint definitions and names.

```
SQL>  SELECT  constraint_name, constraint_type,
  2           search_condition
  3   FROM    user_constraints
  4   WHERE   table_name = 'EMP';
```

```
CONSTRAINT_NAME           C SEARCH_CONDITION
------------------------- - -------------------------
SYS_C00674                C EMPNO IS NOT NULL
SYS_C00675                C DEPTNO IS NOT NULL
EMP_EMPNO_PK              P
...
```

Viewing the Columns Associated with Constraints

- View the columns associated with the constraint names in the USER_CONS_COLUMNS view

```
SQL> SELECT   constraint_name, column_name
  2  FROM     user_cons_columns
  3  WHERE    table_name = 'EMP';
```

```
CONSTRAINT_NAME              COLUMN_NAME
---------------------------- --------------------
EMP_DEPTNO_FK                DEPTNO
EMP_EMPNO_PK                 EMPNO
EMP_MGR_FK                   MGR
SYS_C00674                   EMPNO
SYS_C00675                   DEPTNO
```

Database Objects

Object	Description
Table	Basic unit of storage; composed of rows and columns
View	Logically represents subsets of data from one or more tables
Sequence	Generates primary key values
Index	Improves the performance of some queries
Synonym	Alternative name for an object

What Is a Sequence?

- Automatically generates unique numbers
- Is a sharable object
- Is typically used to create a primary key value
- Speeds up the efficiency of accessing sequence values when cached in memory
- Example:
  - Create a sequence named DEPT_DEPTNO to be used for the primary key of the DEPT table.

```
SQL> CREATE SEQUENCE dept_deptno
  2       INCREMENT BY 1
  3       START WITH 91
  4       MAXVALUE 100
  5       NOCACHE
  6       NOCYCLE;
Sequence created.
```

Confirming Sequences

– Verify your sequence values in the USER_SEQUENCES data dictionary table.

```
SQL> SELECT   sequence_name, min_value, max_value,
  2           increment_by, last_number
  3  FROM     user_sequences;
```

– The LAST_NUMBER column displays the next available sequence number.

NEXTVAL and CURRVAL Pseudocolumns

– NEXTVAL returns the next available sequence value.

It returns a unique value every time it is referenced, even for different users.

- Example Insert a new department named “MARKETING” in San Diego.

```
SQL> INSERT INTO     dept(deptno, dname, loc)
  2  VALUES          (dept_deptno.NEXTVAL,
  3                  'MARKETING', 'SAN DIEGO');
1 row created.
```

– CURRVAL obtains the current sequence value.

NEXTVAL must be issued for that sequence before

```
SQL> SELECT   dept_deptno.CURRVAL
  2  FROM     SYS.dual;
```

Using a Sequence

– Insert a new department named “MARKETING” in San Diego.

```
SQL> INSERT INTO     dept(deptno, dname, loc)
  2  VALUES          (dept_deptno.NEXTVAL,
  3                  'MARKETING', 'SAN DIEGO');
1 row created.
```

– View the current value for the DEPT_DEPTNO sequence.

```
SQL> SELECT   dept_deptno.CURRVAL
  2  FROM     SYS.dual;
```

Using a Sequence

- Caching sequence values in memory allows faster access to those values.
- Gaps in sequence values can occur when:
  - A rollback occurs
  - The system crashes
  - A sequence is used in another table
- View the next available sequence, if it was created with NOCACHE, by querying the USER_SEQUENCES table.

Modifying a Sequence

- Change the increment value, maximum value, minimum value, cycle option, or cache option.

```
SQL> ALTER SEQUENCE dept_deptno
  2          INCREMENT BY 1
  3          MAXVALUE 999999
  4          NOCACHE
  5          NOCYCLE;
Sequence altered.
```

Guidelines for Modifying a Sequence

- You must be the owner or have the ALTER privilege for the sequence.
- Only future sequence numbers are affected.
- The sequence must be dropped and re-created to restart the sequence at a different number.
- Some validation is performed.

Removing a Sequence

- Remove a sequence from the data dictionary by using the DROP SEQUENCE statement.
- Once removed, the sequence can no longer be referenced.

```
SQL> DROP SEQUENCE dept_deptno;
Sequence dropped.
```

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Stored Procedures und Triggers

Datenbanksysteme 2

Inhalt

- Einführung Stored Procedures und Triggers
- PL/SQL
- Stored Procedures
- Funktionen
- Exception Handling
- Cursors
- Triggers
- Packages

Was sind Stored Procedures?

Programmiersprache für, bzw. in Datenbanken.

Stored Procedures (SP)

- sind Programme 'gespeichert' (stored) bei / nahe den Daten
- können in versch. Programmiersprachen implementiert sein: SQL, Java, C, C++
- können SQL Statements ausführen
- können in Triggern genutzt werden

Stored Procedures - für was?

- Validierung
  - (vgl. voriges Kapitel)

- Security
  - Verfeinerter Zugriffsschutz mit SP
  - Sicherheitsüberprüfungen mit Triggers
  - Auditing von Zugriffen mit Trigger

SP implementiert in SQL

SQL mit prozeduralen Elementen

Genormt in SQL:1999/SQL:2003 Teil SQL/PSM

SQL als imperative Programmiersprache

(R)DBMS bieten Erweiterungen (Dialekte) an

- Microsoft SQL Server: Transact-SQL
- DB2: SQL/PL
- PostgreSQL: PL/pgSQL
- Oracle: PL/SQL

Realisierung in Oracle: PL/SQL

Oracle‘s PL/SQL

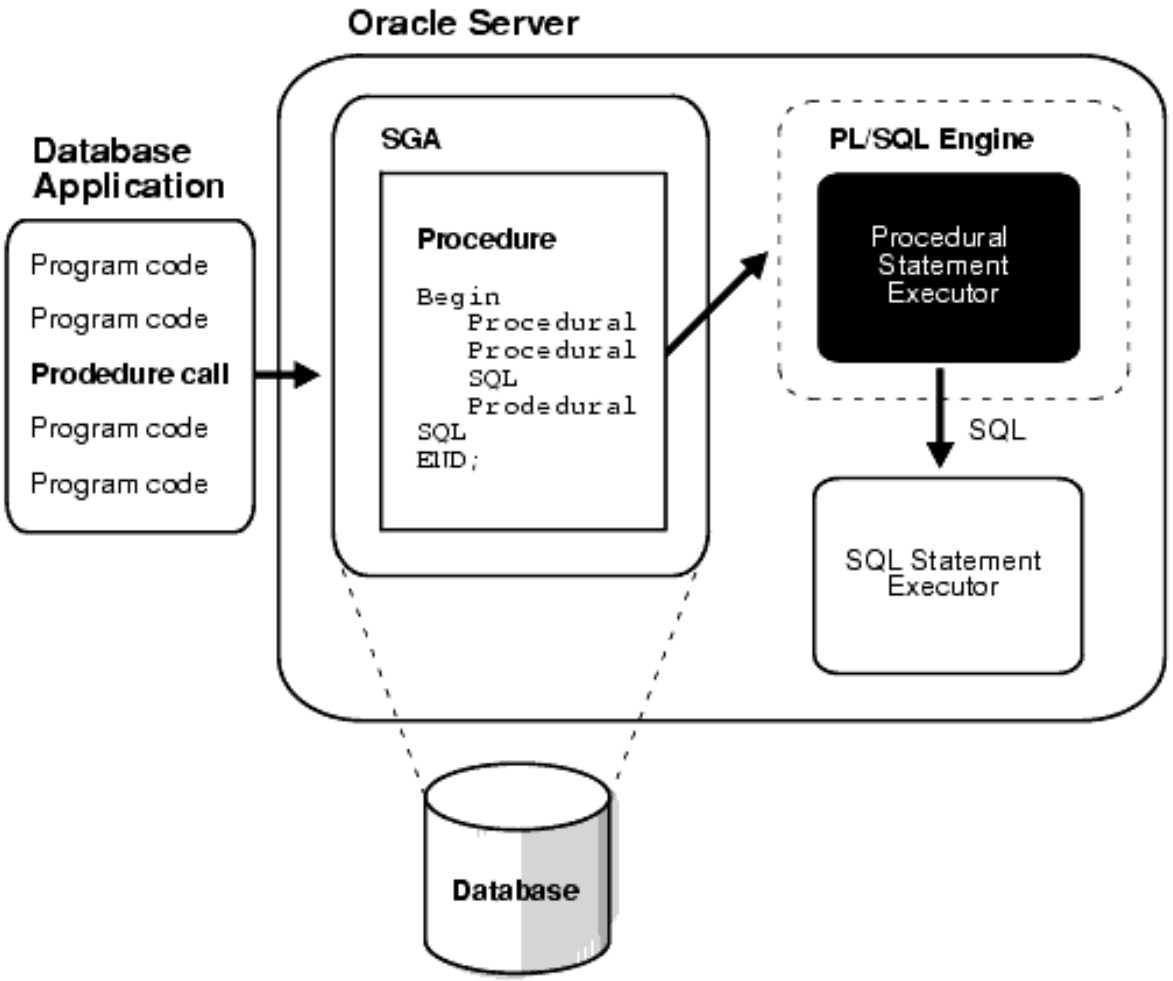
- blockorientierte Programmiersprache, proprietär (Block ~= Modul)
- Sprachelemente für Definition von Variablen und für Steuerung des Kontrollflusses
- Sprachelemente für Zugriff auf DB-Objekte

```
DECLARE
  variable_decl
BEGIN
   program_code
EXCEPTION
   exception_handlers
END;
```

Einsatz

- Server: Stored Procedures und Triggers
- Oracle Client Tools: anonymer PL/SQL Block

Orcacle‘s PL/SQL Engine



Beispiel: PL/SQL-Block

```
SQL>
  1  DECLARE
  2     Emp_name            VARCHAR2(10);
  3     Emp_number          INTEGER;
  4     Empno_out_of_range EXCEPTION;
  5  BEGIN
  6     Emp_number := 10001;
  7     IF Emp_number > 9999 OR Emp_number < 1000 THEN
  8        RAISE Empno_out_of_range;
  9     ELSE
 10        SELECT Name INTO Emp_name FROM Angestellter
 11           WHERE PersNr = Emp_number;
 12        DBMS_OUTPUT.PUT_LINE('Employee name is ' || Emp_name);
 13     END IF;
 14  EXCEPTION
 15     WHEN Empno_out_of_range THEN
 16        DBMS_OUTPUT.PUT_LINE('Employee number ' || Emp_number ||
 17         ' is out of range.');
 18  END;
 19  /
Employee number 10001 is out of range.
```

Beispiel: Verschachtelter PL/SQL-Block

```
DECLARE
  x integer;
  y varchar2(20);
BEGIN
  x := 10;
  y := 'outer block';
  DBMS_OUTPUT.PUT_LINE(y || '  ' || x);
  DECLARE
    y varchar2(20);
  BEGIN
    x := 20;
    y := 'inner block';
    DBMS_OUTPUT.PUT_LINE(y || '  ' || x);
  END;
  DBMS_OUTPUT.PUT_LINE(y || '  ' || x);
END;
```

Beispiel: SELECT ... INTO

```
DECLARE
  AngNr Angestellter.PersNr%TYPE;
BEGIN /*lokaler, anonymer Block */
    SELECT Angestellter.PersNr INTO AngNr
    FROM Angestellter
    WHERE Angestellter.Name = 'Marxer, Markus';
EXCEPTION /*Exception-Handler des lokalen Blocks*/
     WHEN NO_DATA_FOUND THEN
       /*System Exception:
       SELECT INTO liefert keinen Wert*/
       RAISE;
     WHEN TOO_MANY_ROWS THEN
     /*SELECT INTO* liefert mehr als einen Wert*/
     RAISE;
END;
```

SELECT ... INTO muss genau ein Tupel liefern
Anz. Tupel > 1: Exception TOO_MANY_ROWS
Anz. Tupel = 0: Exception NO_DATA_FOUND

Beispiel: INSERT INTO

```
DECLARE
   PNr Projekt.ProjNr%TYPE;
   AngNr Angestellter.PersNr%TYPE;
   ProzAnt DECIMAL,
BEGIN
   ...
   INSERT INTO ProjektZuteilung
   VALUES (AngNr, PNr, ProzAnt, NULL, NULL);
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
      /*Projektzuteilung existiert bereits*/
END;
```

Prüft eine Bedingung!

Fehler: Verstoss gegen Eindeutigkeit eines Schlüssels:
`Exception DUP_VAL_ON_INDEX`

Beispiel UPDATE

```
DECLARE
  AngName Angestellter%Name;
  SalIncr NUMBER;
BEGIN
...
 UPDATE Angestellter A
     SET A.Salaer = A.Salaer + SalIncr
     WHERE A.PersNr = AngName;
 IF SQL%NOTFOUND THEN
     /* kein Angestellter mit AngName*/
 ELSE
   dbms_output.put_line
     ( SQL%ROWCOUNT || ' Tupels aktualisiert');
 END IF;
```

UPDATE, DELETE

Anzahl bearbeitete Tupels können mit SQL%NOTFOUND bzw. SQL%ROWCOUNT abgefragt werden.

Stored Procedures

Für Zugriffs- und Modifikationsoperationen (Datenkapselung)

Sind Datenbankobjekte (wie Tabellen etc.)

Können aus anderen SP, Triggern, PL/SQL-Blöcken oder von Client-Applikationen aufgerufen werden

Recht für Ausführung:

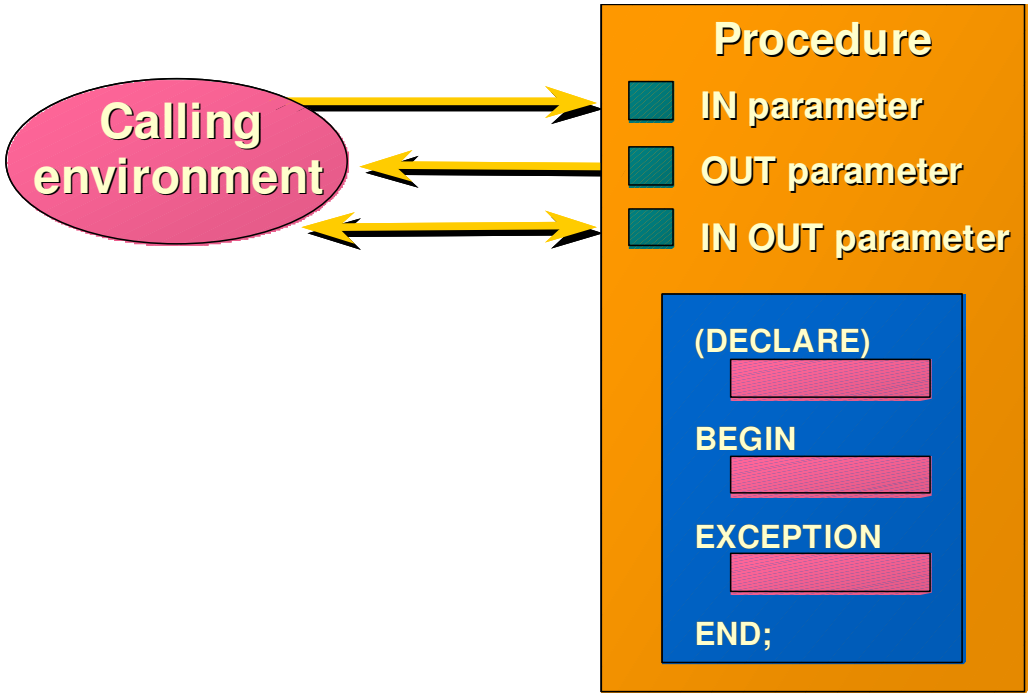
```
GRANT EXECUTE ON ProjektZuteilen TO UserXY;
```

Vorteile:

- Erhöhte Datensicherheit, da alle wichtigen Überprüfungen zentral formuliert und ausgeführt werden
- Verbesserter Datenschutz: Zugriff auf Tabellen nur via Prozeduren
- Verbesserte Leistungsfähigkeit im Client-Server-Betrieb

Prozedur-Parameter

```
CREATE [OR REPLACE] PROCEDURE procedure_name
  ( argument1 [ IN | OUT | INOUT ] datatype1 [ := value ],
    argument2 [ IN | OUT | INOUT ] [NOCOPY] datatype [ := value ],
   . . .)
IS [AS]
PL/SQL-Block;
```



Beispiel PROCEDURE ProjektZuteilen

```
1.  CREATE OR REPLACE
2.      PROCEDURE ProjektZuteilen (
3.          ProjName Projekt.Bezeichnung%TYPE,
4.          AngName Angestellter.Name%TYPE,
5.          ProzAnt DECIMAL,
6.          ErrCode OUT DECIMAL)
7.  AS
8.         PNr Projekt.ProjNr%TYPE;  /* lokale Variable */
9.         AngNr Angestellter.PersNr%TYPE;
10. BEGIN /*Body der Prozedur, vgl. folgende Folien... */
      …
46. EXCEPTION /* Procedure*/
47.        WHEN OTHERS THEN /*alle Exceptions abfangen*/
48.             ROLLBACK;
49. END ProjektZuteilen;  /*Ende Body*/
```

Beispiel PROCEDURE ProjektZuteilen

```
10. BEGIN /*Body der Prozedur*/
11.    ErrCode := 10; /*Def. für eigener unbekannter Fehler*/
12.    BEGIN /*lokaler, anonymer Block*/
13.      SELECT Angestellter.PersNr INTO  AngNr
14.      FROM Angestellter
15.      WHERE Angestellter.Name=AngName;
16.    EXCEPTION /*Exception-Handler des lokalen Blocks*/
17.      WHEN NO_DATA_FOUND THEN
18.        /*System Exception: SELECT INTO liefert keinen Wert*/
19.        ErrCode := -1; /*Angestellter AngName existiert nicht
20.        RAISE;
21.      WHEN TOO_MANY_ROWS THEN
22.        /*SELECT INTO liefert mehr als einen Wert*/
23.        ErrCode := -2; /*AngName ist kein eindeutiges Krit. *
24.        RAISE;
25.   END; /*Ende Body*/
26.
```

Beispiel PROCEDURE ProjektZuteilen

```
38.   BEGIN
39.     INSERT INTO ProjektZuteilung
40.             VALUES (AngNr, PNr, ProzAnt, NULL, NULL);
41.     ErrCode := 1; /* ok */
42.   EXCEPTION
43.     WHEN DUP_VAL_ON_INDEX THEN
44.       ErrCode := -5; /*Projektzuteilung existiert bereits*/
45.   END;
```

Aufruf PL/SQL (aus SQLPlus)

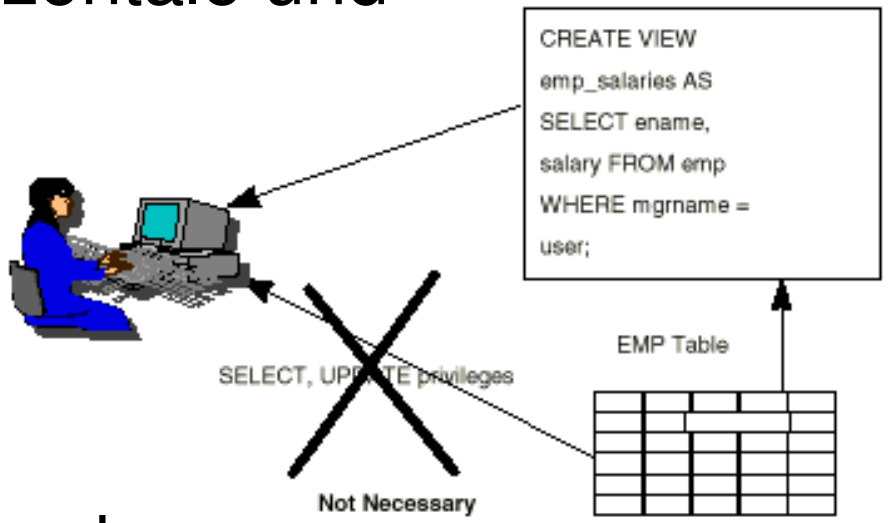
```
DECLARE
 errcode decimal;
BEGIN
  ProjektZuteilen( 'Mars', 'Rey, Herbert', 80, errCode );
  dbms_output.put_line( errcode );
END;
```

Beispiel PROCEDURE SalaerErhoehung

```
CREATE OR REPLACE
  PROCEDURE SalaerErhoehung
          ( AngNr IN NUMBER, SalIncr IN NUMBER )
   AS
          FalscheAngNr EXCEPTION;
          UngueltigeErhoehung EXCEPTION;
   BEGIN
     IF ( SalIncr < 0 OR SalIncr > 2000 ) THEN
          RAISE UngueltigeErhoehung;
     END IF;
     UPDATE Angestellter A
          SET A.Salaer = A.Salaer + SalIncr
          WHERE A.PersNr = AngNr;
     IF SQL%NOTFOUND THEN
          RAISE FalscheAngNr;
     END IF;
END SalaerErhoehung;
```

Zur Erinnerung: Zugriffsschutz mit Views

- Benutzer (Applikation) greift über die View auf Daten zu
- Views erlauben horizontale und vertikale Filterung der Daten
- Benutzer benötigt keine Privilegien für den Zugriff auf die darunterliegende Tabelle



Zur Erinnerung: Zugriffsschutz mit Views

Mit Views können vertrauliche Daten geschützt werden: Zuerst wird ein Sicht mit den öffentlichen Attributen definiert (vertikale Filterung), dann wird diese mit der Grant-Anweisung allen Benutzern zugänglich gemacht. Der Zugriff auf die unterliegende Tabelle wird verwehrt. Vertrauliche Daten, die in der Sicht nicht enthalten sind, bleiben so geschützt.

```
CREATE VIEW AngPublic (Persnr, Name, Tel,
Wohnort) AS
SELECT Persnr, Name, Tel, Wohnort
FROM Angestellter;

GRANT SELECT ON AngPublic TO PUBLIC;
```

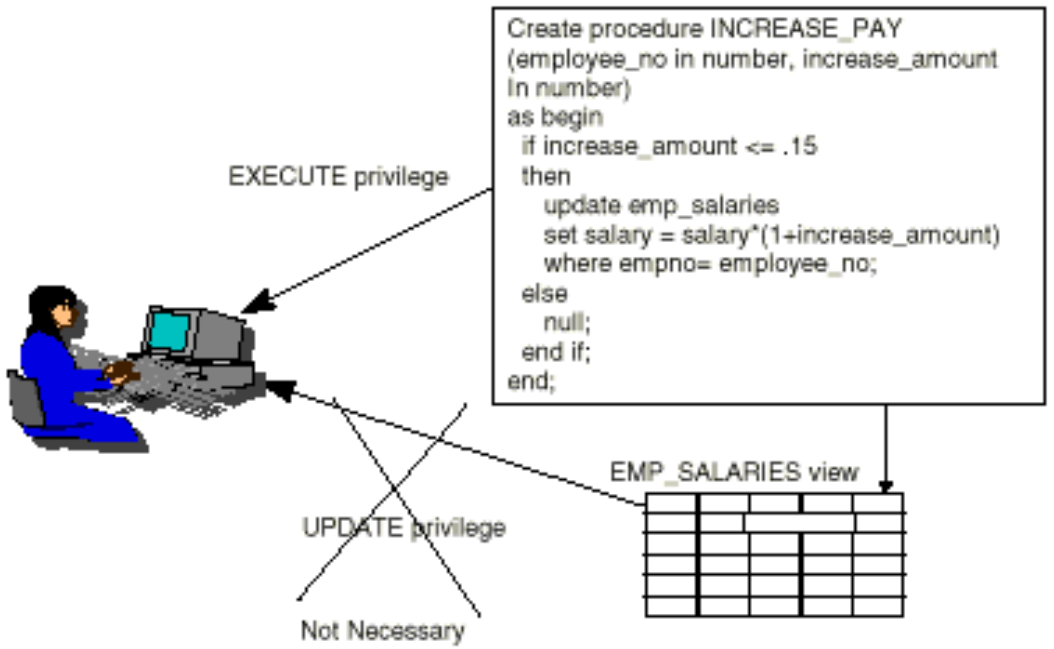
Zugriffsschutz mit Stored Procedures

- Benutzer (Applikation) modifiziert über den Aufruf von SP die Daten
- SP kapseln die Daten und können Teile der Geschäftslogik implementieren
- Benutzer benötigt keine Privilegien für die Modifikation der darunterliegende Tabelle

Definer-Rights Procedures laufen mit den Rechten ihres Erstellers.

Procedures sollten vom Owner der Tabellen erstellt werden.

PS/SQL erkennt Privilegien nicht, die über Rollen gewährt werden!



Zugriffsrechte mit Stored Procedures

Modifikationsoperationen mit Stored Procedures

Lesezugriff über Views

Beispiele:

- alle dürfen ihre Daten ansehen

```
GRANT SELECT ON Angestellter_V TO PUBLIC;
```

- Abteilungsleiter dürfen die Daten ihrer Mitarbeiter einsehen

```
GRANT SELECT ON AbtAng_V TO  AbtLeiter_R;
```

- PersonalChefs dürfen Salärerhöhungen ausführen:

```
GRANT EXEC ON SalaerErhoehung TO
PersonalChef_R;
```

Funktionen

```
CREATE OR REPLACE FUNCTION AbteilungSalaer(AbtName VARCHAR2)
RETURN NUMBER
IS
 AbtSalaer NUMBER (10,2) := 0;
BEGIN
 select sum(A.salaer) INTO AbtSalaer
 from Angestellter A, Abteilung Abt
 where A.AbtNr=Abt.AbtNr
 and Abt.Name=AbtName;
 RETURN AbtSalaer;
END AbteilungSalaer;
```

Hinweis:
'IS' oder 'AS'

Aufruf von Funktionen

Verwendung in einer SQL-Abfrage:

```
select Name, AbteilungSalaer(Name)  from Abteilung
where AbteilungSalaer(Name) > 40000
```

Aufruf der Funktion AbteilungSalaer() aus einem PL/SQL-Block:

```
SQL> run
  1  DECLARE
  2   TotSalaer Number;
  3  BEGIN
  4   TotSalaer := AbteilungSalaer( 'Entwicklung' );
  5   dbms_output.put_line( 'Salaer Entwicklung '||TotSalaer );
  6* END;

Salaer Entwicklung 63496
```

Aufruf von Funktionen über DUAL

Aufruf einer benutzerdefinierten Funktion:

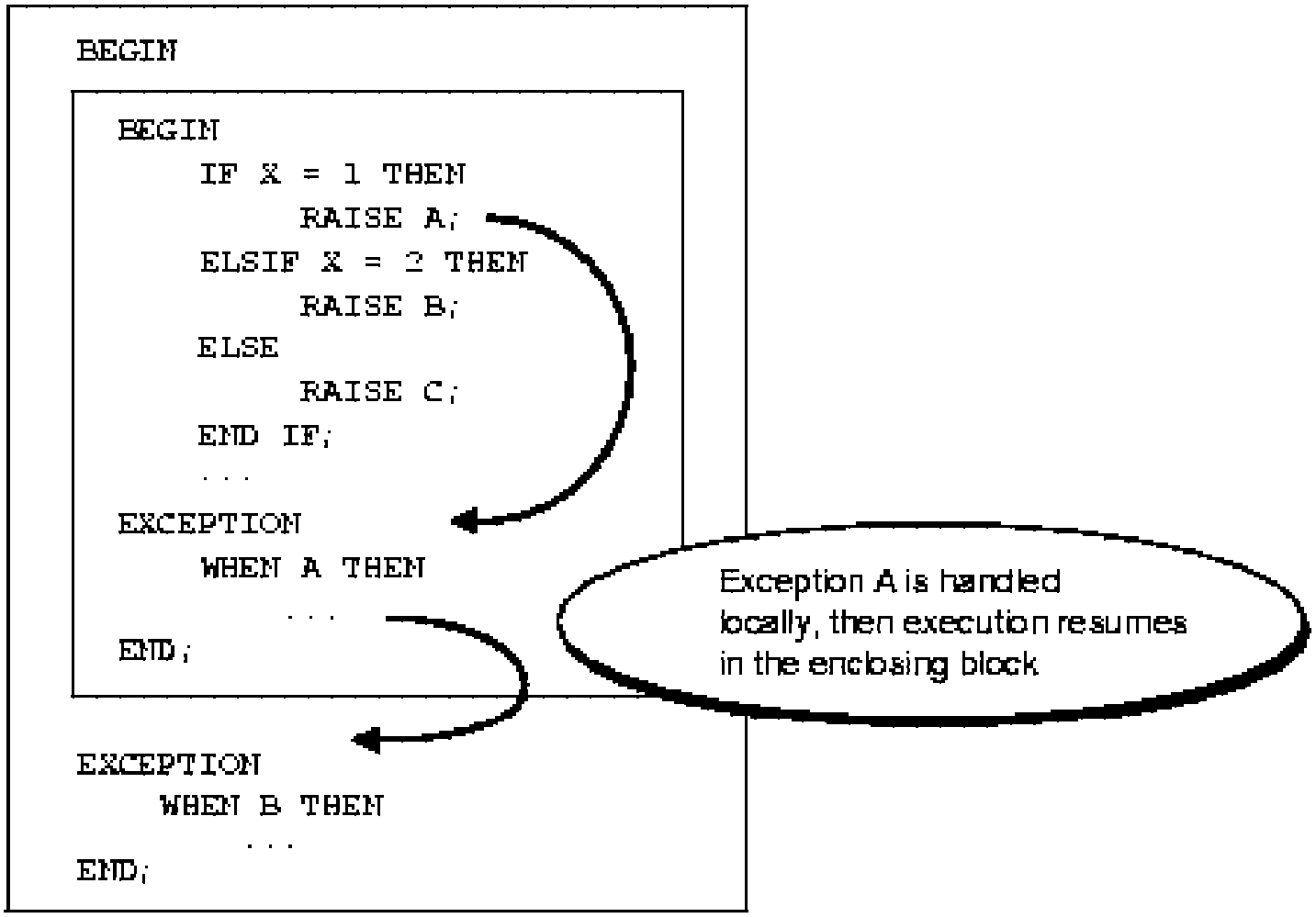
```
SQL> select AbteilungSalaer('Entwicklung') from DUAL;

ABTEILUNGSALAER('ENTWICKLUNG')
------------------------------
                         63496
```

Über ein SELECT auf Tabelle DUAL lassen sich auch alle Systemfunktionen aufrufen:

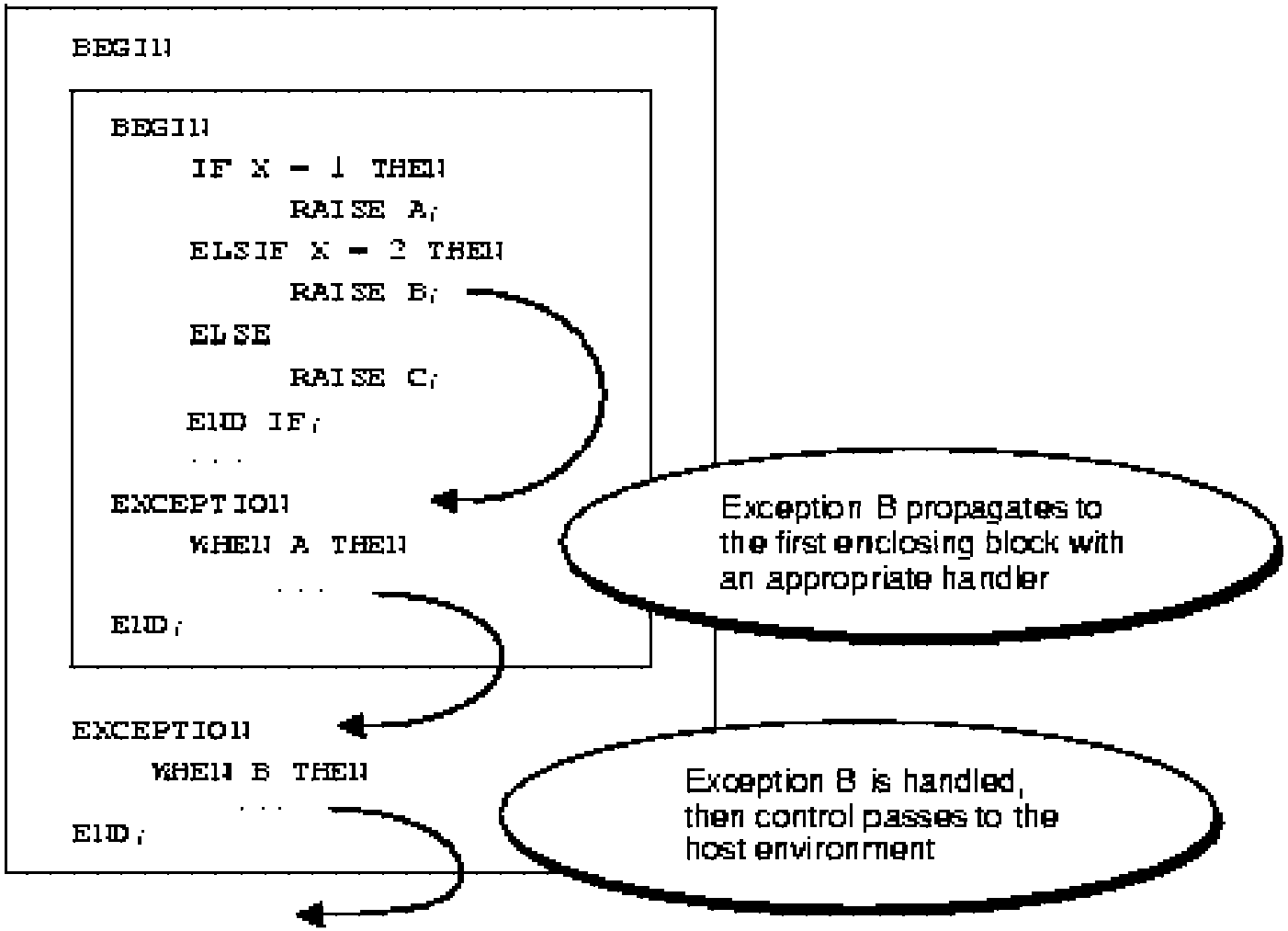
```
SQL> select sysdate from DUAL;

SYSDATE
---------
03-SEP-01
```

PL/SQL: Exceptions

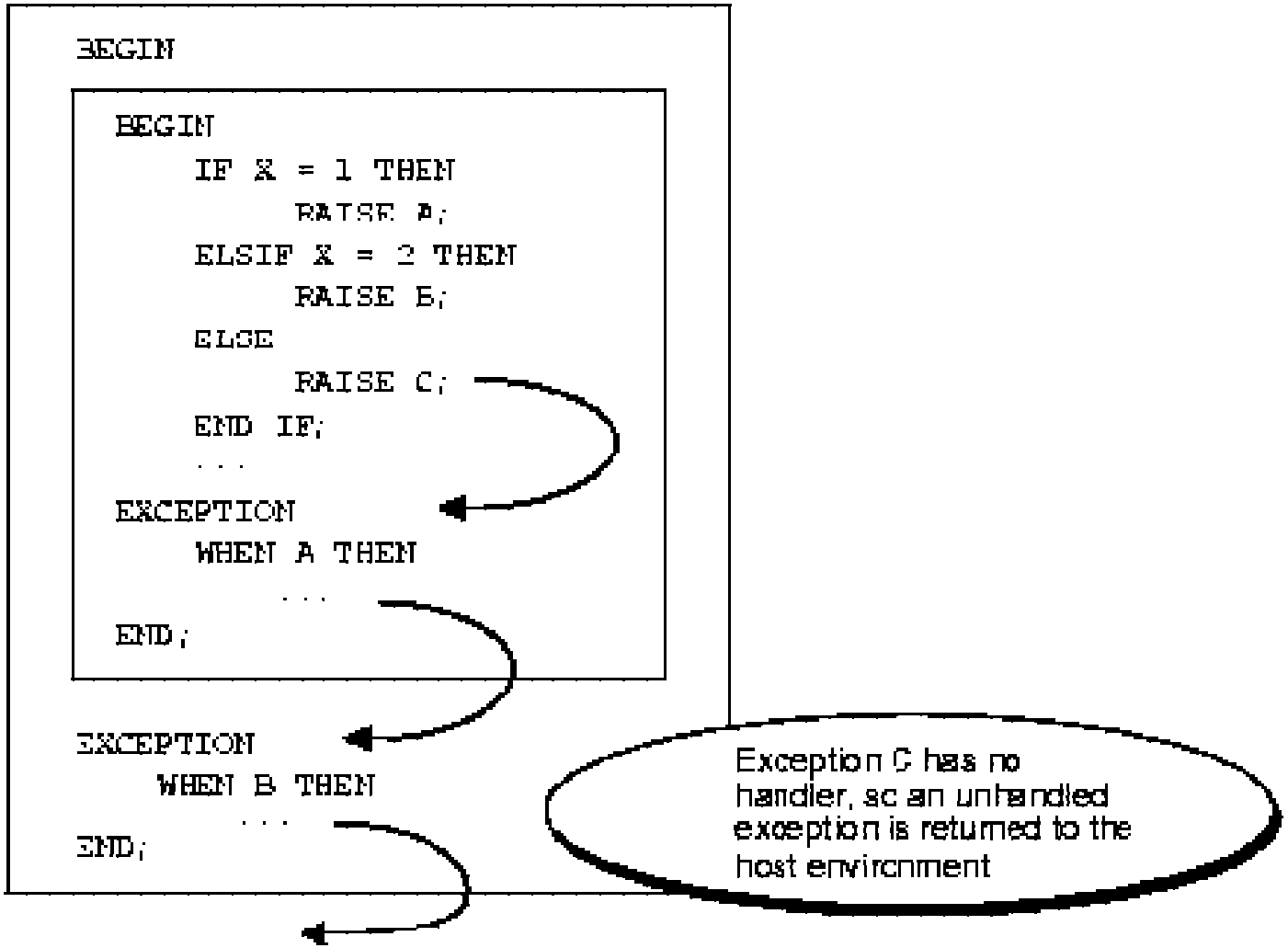
PL/SQL Exception Propagation 1



PL/SQL Exception Propagation 2



PL/SQL Exception Propagation 3



PL/SQL Exceptions: Benannte Benutzer-Ausnahmen

```
create or replace procedure BenannteBenutzerException
is
  Ausnahme1 exception;
  pragma exception_init(Ausnahme1, -20000);

begin
  raise Ausnahme1;
exception
when Ausnahme1 then
  DBMS_OUTPUT.PUT_LINE
   ('Exception Handler Ausnahme1 ErrNr' || SQLCODE ||
' ErrMsg ' || SQLERRM );
  raise Ausnahme1;
when others then
  DBMS_OUTPUT.PUT_LINE ('unbekannte exception');
end;
```

PL/SQL Exceptions: Benannte Benutzer-Ausnahmen

```
Client (anonymer PL/SQL-Block)

declare
     Ausnahme1 Exception;
     pragma exception_init(Ausnahme1, -20000);
begin
     BenannteBenutzerException;
exception
when Ausnahme1 then
     DBMS_OUTPUT.PUT_LINE
          ('--- Exception -20000 abgefangen');
end;
```

PL/SQL Exceptions: Benannte System-Ausnahmen

```
create or replace procedure BenannteSystemexception is
  i integer := 10;
begin
  i := i / 0;
end;
```

Client (anonymer PL/SQL-Block)

```
begin
  BenannteSystemException;
exception
  when ZERO_DIVIDE then
     DBMS_OUTPUT.PUT_LINE  ('-- ZERO_DIVIDE abgefangen: ErrNr' ||
                             sqlcode || ' ErrMsg ' || sqlerrm);
end;
```

```
Generierte Ausgabe:
-- ZERO_DIVIDE abgefangen: ErrNr-1476 ErrMsg ORA-01476: divisor is
equal to zero
```

PL/SQL Exceptions: Unbenannte Benutzer-Ausnahmen

```
create or replace procedure UnbenannteBenutzerException is
begin
  RAISE_APPLICATION_ERROR
         (-20001, 'Unbenannte Benutzer Exception');
end;
```

Client (anonymer PL/SQL-Block)

```
begin
   UnbenannteBenutzerException;
exception
when others then
   DBMS_OUTPUT.PUT_LINE ('-- Exception abgefangen: ErrNr ' ||
                         sqlcode || ' ErrMsg ' || sqlerrm);
end;
```

Generierte Ausgabe:

```
-- Exception abgefangen: ErrNr -20001 ErrMsg ORA-20001: Unbenannte
BenutzerException
```

Beispiele benannter Systemexceptions

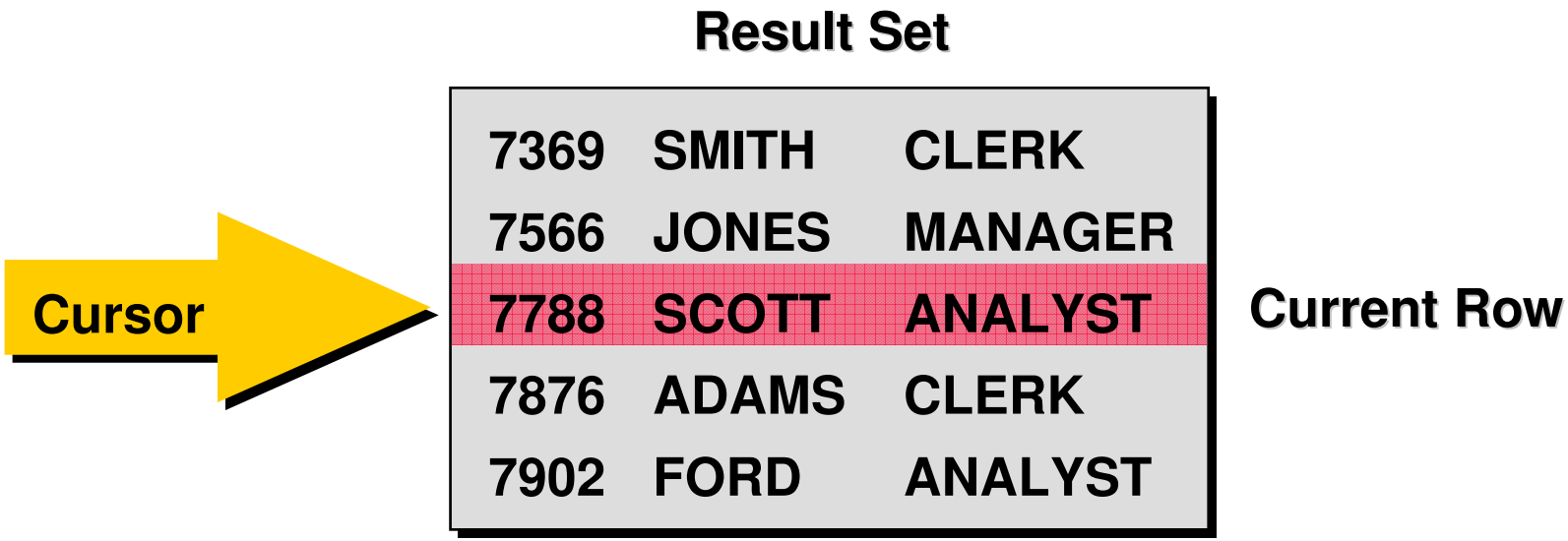
Name der Ausnahme	SQLCode	Beschreibung
DUP_VAL_ON_INDEX	-1	Insert- oder Update- würde mehrfach vorkommende Werte für eine Spalte erzeugen, für die ein eindeutiger Index existiert
NO_DATA_FOUND	-100	SELECT INTO liefert kein Tupel
TOO_MANY_ROWS	-1422	SELECT INTO liefert mehr als ein Tupel
LOGON_DENIED	-1017	Connect weist ungültiger Username und/oder Passwort auf
VALUE_ERROR	-6502	Fehler bei einer Konversion von Daten (z.B. Umwandlung eines Strings in einen numerischen Wert oder Umwandlung eines numerischen Wertes in einen zu kurze String-Variable)
ZERO_DIVIDE	-1476	Division durch Null
CURSOR_ALREADY_OPEN	-6511	Versuch einen bereits geöffneten Cursor zu öffnen.

Cursors

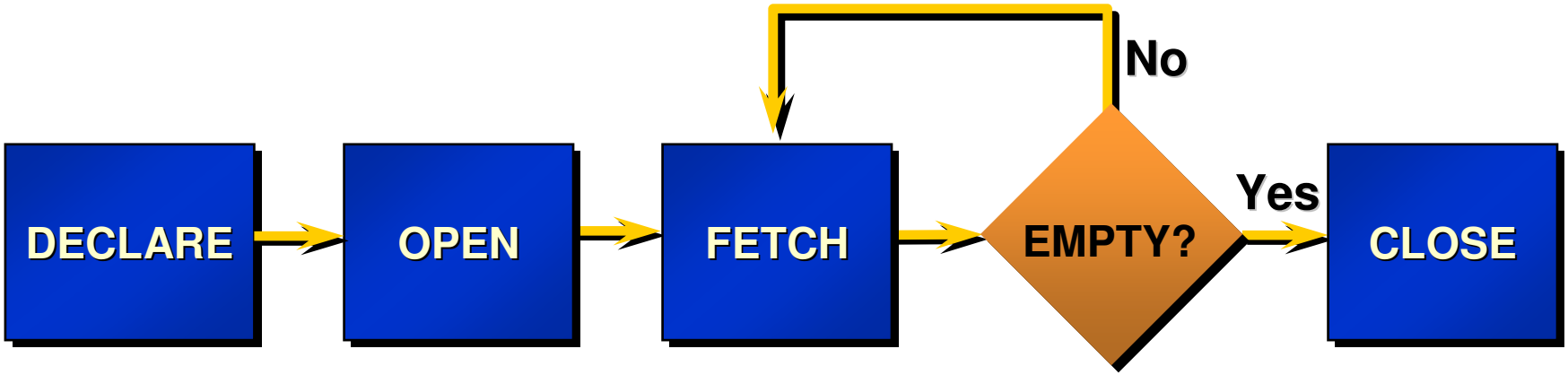
Cursor

Select-Statement liefert in der Regel eine Menge von Tupels (=Result Set)

Cursor erlaubt den sequentiellen Zugriff auf die einzelnen Tuples des Result Set

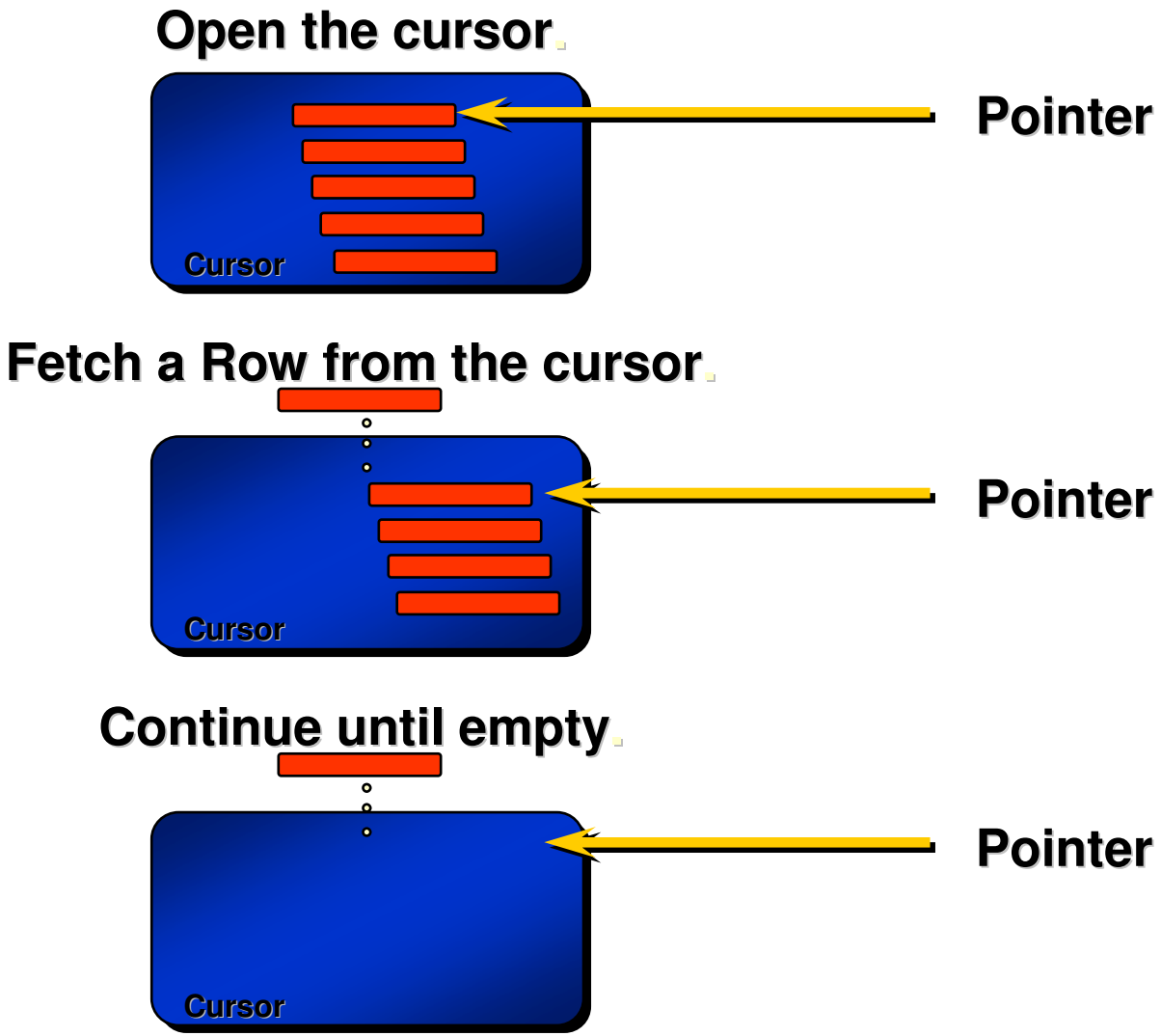


Cursor-Verarbeitung



- Create a named SQL area
- Identify the active set
- Load the current row into variables
- Test for existing rows
- Return to FETCH if rows found

Cursor Prinzip



Cursor-Verarbeitung

1. Deklaration des Cursors

```
DECLARE
  CurrAbtNr integer := 1;
CURSOR AngCursor IS
   SELECT Salaer, PersNr FROM Angestellter
     WHERE Angestellter.AbtNr=CurrAbtNr;
  SalSumme NUMBER (8, 2) := 0;
  AngSalaer Angestellter.Salaer%TYPE;
  AngPersNr Angestellter.PersNr%TYPE;
```

Cursor-Verarbeitung

2. Öffnen des Cursors, Verarbeiten der Tuples

```
BEGIN
  OPEN AngCursor; /*SQL-Abfrage starten und Resultat in Puffer
speichern*/
  LOOP /*Iteration ueber Resulatmenge*/
    FETCH AngCursor INTO AngSalaer, AngPersNr;
    EXIT WHEN AngCursor%NOTFOUND OR AngCursor%ROWCOUNT>10
    SalSumme := SalSumme + AngSalaer;
    dbms_output.put_line('Angstellter PersNr: ' || AngPersNr ||
' Salaer: ' || AngSalaer);
  END LOOP;
  CLOSE AngCursor;
  dbms_output.put_line('Salaersumme: ' || SalSumme);
END;
```

Cursor mit Parametern

Parameter müssen beim Oeffnen des Cursors mit aktuellen Werten versehen werden

```
DECLARE
  CurrAbtNr integer;
  CURSOR AngCursor (AbtId IN Abteilung.AbtNr%TYPE) IS
     SELECT Salaer, PersNr FROM Angestellter
     WHERE Angestellter.AbtNr=AbtId;
    SalSumme NUMBER (8, 2) := 0;
  AngRec    AngCursor%ROWTYPE;
```

ROWTYPE liefert den Typ des Tupels eines Cursors oder einer Tabelle

Verwenden eines Cursors mit Parametern

```
BEGIN
  FOR CurrAbtNr IN 1..2
  LOOP
    OPEN AngCursor (CurrAbtNr);
      LOOP /*Iteration ueber Resulatmenge*/
        FETCH AngCursor INTO AngRec;
        EXIT WHEN AngCursor%NOTFOUND;
        SalSumme := SalSumme + AngRec.Salaer;
     END LOOP;
    CLOSE AngCursor;
    dbms_output.put_line('Salaersumme: ' || SalSumme);
  END LOOP;
END;
```

Cursor for Update

```
CURSOR AngCursor IS
    SELECT Salaer, Chef, PersNr FROM Angestellter
    WHERE Angestellter.AbtNr=CurrAbtNr
    FOR UPDATE;
```

```
OPEN AngCursor;
LOOP
  FETCH AngCursor INTO AngRec;
  EXIT WHEN AngCursor%NOTFOUND;
  UPDATE Angestellter SET Salaer = MinSalaer
  WHERE CURRENT OF AngCursor;
END LOOP;
CLOSE AngCursor;
```

Positioniert Cursor auf das aktuelle Tupel, aktuelles Tupel wird gesperrt.

Aendert das durch den Cursor referenzierte Tupel

Cursor-Attribute

Attribute	Type	Description
%ISOPEN	Boolean	Evaluates to TRUE if the cursor is open.
%NOTFOUND	Boolean	Evaluates to TRUE if the most recent fetch does not return a row.
%FOUND	Boolean	Evaluates to TRUE if the most recent fetch returns a row; complement of %NOTFOUND
%ROWCOUNT	Number	Evaluates to the total number of rows returned so far.

Cursor: Beispiel

```
CREATE OR REPLACE
   FUNCTION BerechneSalSumme (PersNr Angestellter.PersNr%TYPE)
RETURN NUMERIC IS
CURSOR AngCursor IS
   SELECT PersNr, Name, Salaer FROM Angestellter
     WHERE Angestellter.Chef = PersNr;
  SalSumme NUMBER (10, 2) := 0;
  AngRec AngCursor%ROWTYPE;
BEGIN
  OPEN AngCursor; /*Abfrage, dann Resultat in Puffer speichern*/
  LOOP /*Iteration ueber Resulatmenge*/
    FETCH AngCursor INTO AngRec;
    EXIT WHEN AngCursor%NOTFOUND;
    SalSumme := SalSumme + AngRec.Salaer;
    SalSumme := SalSumme + BerechneSalSumme (AngRec.PersNr);
  END LOOP;
  CLOSE AngCursor;
  RETURN SalSumme;
END BerechneSalSumme;
```

Trigger

Trigger

- Sicherstellung komplexer Konsistenzbedingungen
- Berechnen der Werte von abgeleiteten Attribute (derived attribute values)
- Zugriffsschutz und Auditing
- Sammeln von Statistik- und Logdaten
- Triggers
  - sind keine DB-Objekte,
  - sind immer einer Tabelle zugeordnet,
  - haben keine Parameter
  - können nicht direkt aufgerufen werden
  - werden vom DBMS **automatisch** beim Eintreten eines Trigger-Events aufgerufen

Eigenschaften von Triggern

Auslösendes Ereignis (Trigger Event)

- INSERT
- DELETE auf einer Tabelle,
- UPDATES von Attributen.

Trigger Time:

- Before-Triggers Überprüfen Vorbedingungen
- After-Triggers : Sichern Nachbedingungen

Trigger Restriction: bedingte Ausführung

Trigger-Typen:

- Row-Triggers: Aufruf pro betroffenes Tupel
- Statement-Triggers: nur ein Aufruf pro SQL-Anweisung

Eigenschaften von Triggern

Trigger Action:

- Block mit SQL-Anweisungen und Deklarationen für Variablen und Cursors etc.

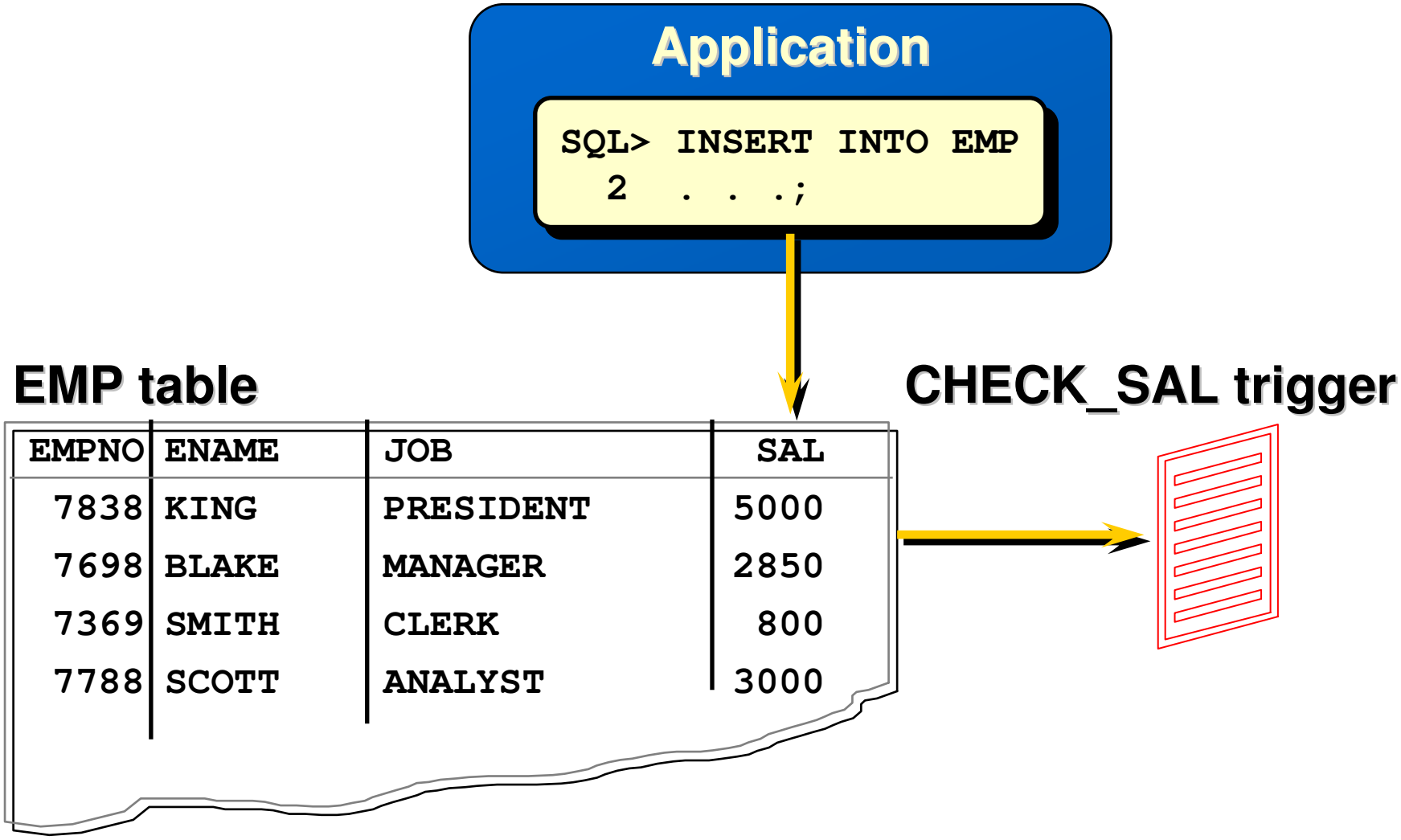
Implizite Parameter (Row-Trigger)

- :OLD: Zugriff auf das 'alte' Tupel.
- :NEW Zugriff auf das 'neue' Tupel.

Rechte:

- Triggers haben bei der Ausführung die Rechte ihres Owners
- Erzeugen von Triggers bedingt das Systemprivileg CREATE TRIGGER.

Database Trigger: Beispiel



Sicherheitsüberprüfungen mit Triggers

Der Trigger `CheckInsertAng` überprüft, dass Änderungen an Angestelltendaten nur während der Arbeitszeit an Wochentagen ausgeführt werden können:

```
CREATE OR REPLACE TRIGGER CheckInsertAng
  BEFORE INSERT OR UPDATE OR DELETE ON Angestellter
BEGIN
  IF  (TO_CHAR (sysdate,'DY') IN ('SAT','SUN') ) OR
      (TO_CHAR(sysdate,'HH24') NOT BETWEEN '08' AND '18')
  THEN
       RAISE_APPLICATION_ERROR
         (-20500, 'Aenderungen von Ang. nur während
Arbeitszeit erlaubt!');
  END IF;
END;
```

Sicherstellung der referentiellen Integrität

Implementation der ‘on update cascade‘ Semantik

```
CREATE OR REPLACE TRIGGER cascade_updates
   AFTER UPDATE OF AbtNr ON Abteilung
   FOR EACH ROW
   BEGIN
     IF :OLD.AbtNr != :NEW.AbtNr THEN
       UPDATE Angestellter A
       SET    A.AbtNr = :new.AbtNr
       WHERE  A.AbtNr = :old.AbtNr;
       UPDATE AbtLeitung AL
       SET    AL.AbtNr = :new.AbtNr
       WHERE  AL.AbtNr = :old.AbtNr;
     END IF;
  END;
```

Sicherstellen von Vorbedingungen

```
CREATE OR REPLACE TRIGGER check_SalaerAenderung
  BEFORE UPDATE OF Salaer ON Angestellter
  FOR EACH ROW
  WHEN  ( (:new.Salaer < :old.Salaer * 1.1 ) OR
          (:new.salaer > :old.salaer * 1.1 )  )
  BEGIN
     RAISE_APPLICATION_ERROR (-20508,
        'Salaer nicht um mehr als 10% ändern!');
END;
```

Abgeleitete Attribute

Die Tabelle Firma enthält berechnete Werte (z.B. Salärsumme)

Dieser abgeleitete Wert muss nach jeder Änderung der Basisdaten (z.B. Salär des Angestellten) neu berechnet werden

```
CREATE TABLE Firma (
  SalaerSumme NUMBER(10,2)
);
```

Initialisiere die abgeleiteten Attribute:

```
INSERT INTO Firma
  ( SELECT Sum(Salaer) FROM Angestellter )
;
```

Abgeleitete Attribute

```
CREATE OR REPLACE PROCEDURE increment_salaer
    ( v_salaer IN Firma.SalaerSumme%TYPE )
IS
  BEGIN
    UPDATE Firma
    SET SalaerSumme = NVL (SalaerSumme ,0) + v_salaer;
END increment_salaer;

CREATE OR REPLACE TRIGGER berechne_Salaersumme
  AFTER INSERT OR UPDATE OF Salaer OR DELETE ON Angestellter
  FOR EACH ROW
  BEGIN
   IF DELETING THEN increment_salaer   ( -1 * :old.salaer);
   ELSIF UPDATING THEN
           increment_salaer(:new.Salaer-:old.Salaer);
   ELSE /*inserting*/ increment_salaer (:new.salaer);
   END IF;
END;
```

Auditing mit Triggers

Der Trigger `LogSalaerAenderung` wird immer aufgerufen, wenn mit

```
update Angestellter SET Salaer = ....
```

ein Salär verändert wird.

Der Trigger schreibt einen Auditsatz in die Tabelle `AngAudit`:

```
CREATE OR REPLACE TRIGGER LogSalaerAenderung
AFTER UPDATE OF Salaer ON Angestellter
FOR EACH ROW
WHEN (:new.Salaer != :old.Salaer)
BEGIN
  INSERT INTO AngAudit VALUES
   (user, :new.PersNr, SYSDATE, :new.Salaer,
                                'Neues Salaer');
END;
```

Sicherheitsüberprüfungen mit Triggers

Der Trigger `emp_permit_changes` überprüft, dass Änderungen an Angestelltendaten nur während der Arbeitszeit an Wochentagen ausgeführt werden können:

```
CREATE TRIGGER emp_permit_changes
BEFORE INSERT OR DELETE OR UPDATE ON Angestellter
DECLARE
   dummy INTEGER;
   not_on_weekends EXCEPTION;
   non_working_hours EXCEPTION;
BEGIN
   /* check for weekends */
   IF (TO_CHAR(sysdate, 'DY') = 'SAT' OR
      TO_CHAR(sysdate, 'DY') = 'SUN') THEN
      RAISE not_on_weekends;
   END IF;
```

Sicherheitsüberprüfungen mit Triggers

```
/* Check for work hours (8am to 6pm) */
   IF (TO_CHAR(sysdate, 'HH24') < 8 OR
      TO_CHAR(sysdate, 'HH24') > 18) THEN
      RAISE non_working_hours;
   END IF;
EXCEPTION
   WHEN not_on_weekends THEN
      raise_application_error(-20324,'May not change '
         ||'employee table during the weekend');
   WHEN non_working_hours THEN
      raise_application_error(-20326,'May not change '
      ||'emp table during non-working hours');
END;
```

Ausführung von Triggers

1. Führe alle BEFORE statement Triggers aus
2. Für jedes betroffenes Tupel gilt:
   - Führe alle BEFORE row Triggers aus
   - Bearbeite Tupel (Lock und Update, Locks werden gehalten)
   - Führe alle AFTER row Triggers aus
3. Überprüfe allfällige deferred integrity constraints
4. Führe alle AFTER statement Triggers aus

Spezielle Triggers (Oracle)

Instead-of-Triggers

- Modifikationsoperationen auf Views werden zur darunterliegenden Tabelle weitergeleitet

Logon- / Logoff-Trigger

- Werden beim erfolgreichen Anmelden oder vor dem Abmelden gefeuert
- Trigger hat zugriff auf userid und username

DDL Triggers

- Before / After Create / Alter / Drop Databaseobject

Instead-of-Triggers

Modifikationsoperationen auf Views werden zur darunterliegenden Tabelle weitergeleitet

Können für die Operationen Insert, Update und Delete auf Tables und Views definiert werden

Werden anstelle der generischen SQL-Operation ausgeführt

Anwendung: Implementierung von veränderbaren Views, z.B. Für externe Sichten für Applikationen um die DB-Schema Komplexität zu verringern

Instead-of-Triggers: Beispiel

```
CREATE OR REPLACE VIEW AbtLeiterInfo
                        (AbtName, ALPersNr, ALName)
 AS
    SELECT abt.name, ang.persnr,ang.name
    FROM Abteilung abt
     inner join  abtleitung al on abt.abtnr=al.abtnr
    inner join angestellter ang on ang.persnr=al.abtchef
;
/
```

View ist beschränkt änderbar, Insert- und Delete-Operationen sind nicht definiert.

➔ Implementiere instead-of Triggers für Insert, Delete und Update

Instead-of-Triggers
Beispiel INSTEAD OF INSERT

```
CREATE OR REPLACE TRIGGER AbtLeiterInfo_Insert
INSTEAD OF INSERT ON AbtLeiterInfo
REFERENCING NEW AS n
FOR EACH ROW
DECLARE
   rowcnt number;
   abtnr  number := 1001;
BEGIN
   SELECT COUNT(*) INTO rowcnt FROM Abteilung Abt
   WHERE Abt.name= :n.AbtName;
   IF rowcnt = 0  THEN
         select abtNr_SEQ.nextval into abtnr from dual;
         INSERT INTO Abteilung (abtnr, name)
         VALUES(abtnr, :n.AbtName);
   ELSE
         SELECT Abt.abtnr INTO abtnr FROM Abteilung Abt
         WHERE Abt.name= :n.AbtName;
   END IF;
   // ... ff.
```

Instead-of-Triggers
Beispiel INSTEAD OF INSERT ff.

```
    SELECT COUNT(*) INTO rowcnt FROM Angestellter
    WHERE persnr= :n.ALpersnr;
    IF rowcnt = 0  THEN
        INSERT INTO Angestellter (persnr,name, abtnr, salaer)
                        VALUES (:n.ALpersnr, :n.ALName, abtnr, 10000);
        INSERT INTO AbtLeitung values(abtnr, :n.ALpersnr);
    ELSE
       UPDATE Angestellter SET Name= :n.ALName
          WHERE persnr= :n.alpersnr;
    END IF;
END;
```

Packages

Packages: Überblick

- Gruppieren logisch zusammenhängende PL/SQL-Typen und -Unterprogramme
- Haben zwei Teile:
  - Specification
  - Body
- Können nicht aufgerufen, parameterized, oder verschachtelt werden

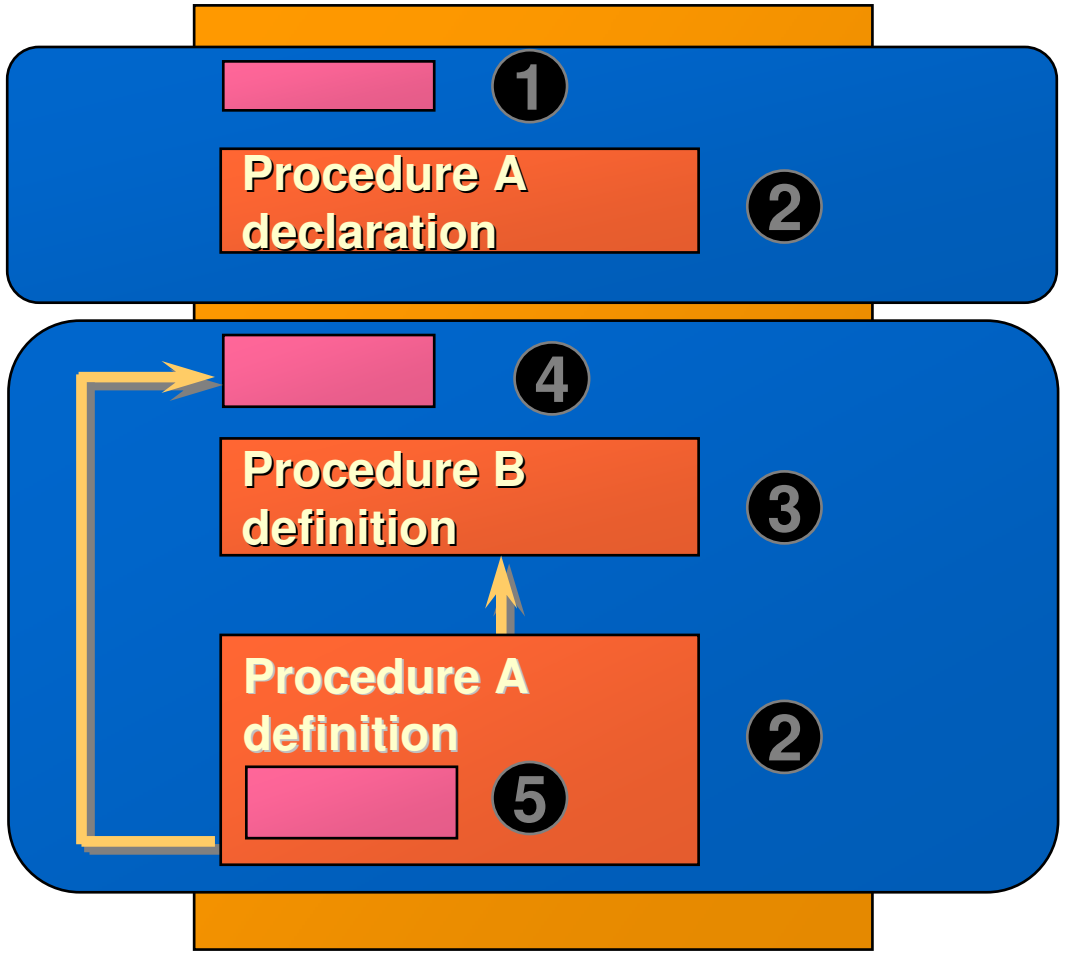
Packages: Vorteile

- Modularity
- Easier application design
- Information hiding
- Added functionality
- Better performance

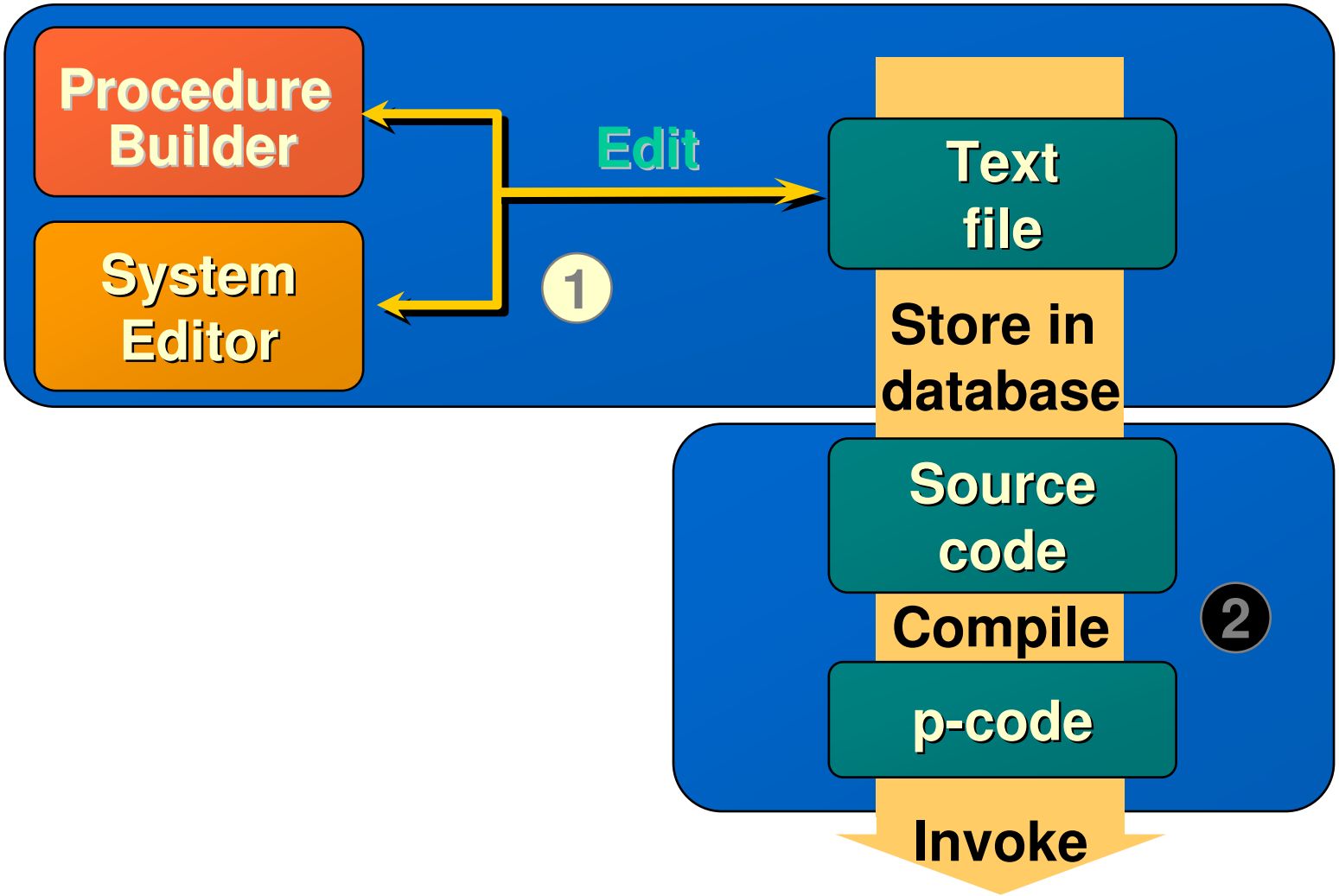
Developing a Package

Package specification

Package body



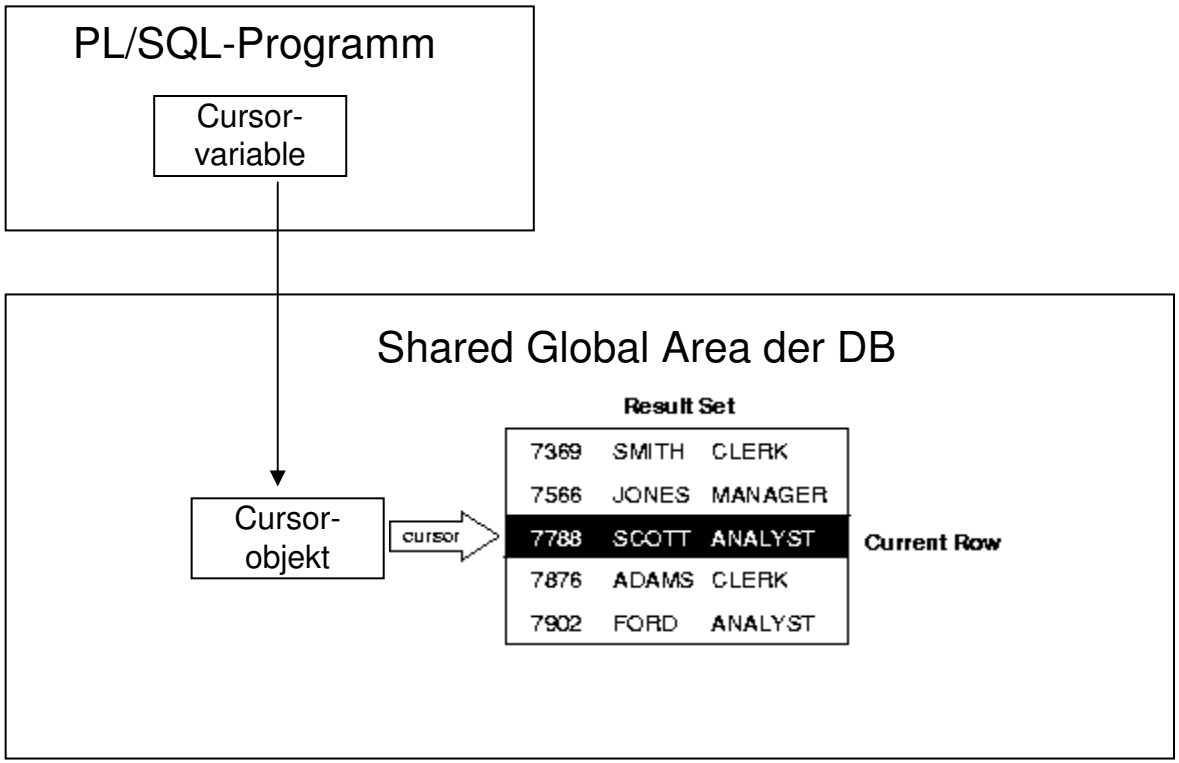
Developing a Package



Developing a Package

- Saving the text of the CREATE PACKAGE statement in two different text files facilitates later modifications to the package.
- A package specification can exist without a package body, but a package body cannot exist without a package specification.
- If you have incorporated a standalone procedure into a package, you should drop your standalone procedure.

Cursor-Variable



Package: Beispiel

```
CREATE OR REPLACE PACKAGE AngestelltenVerwaltung  AS
        MinSalaer CONSTANT NUMBER := 1000.00;
   FUNCTION AngEinstellen (Name VARCHAR2,
     AbtName VARCHAR2, WOrt VARCHAR2, Sal NUMBER := MinSalaer) RETURN
   NUMBER;

    PROCEDURE AngEntlassen (AngNr IN NUMBER);
    PROCEDURE SalaerErhoehung (AngNr IN NUMBER, SalIncr IN NUMBER);
    PROCEDURE Bonus (AngNr IN NUMBER, Betrag IN NUMBER);

    TYPE AngCurTyp IS REF CURSOR RETURN angestellter%ROWTYPE;

    TYPE AngRecTyp IS RECORD (
          Name Angestellter.Name%TYPE,
          AbtName Abteilung.Name%TYPE,
          Salaer Angestellter.Salaer%TYPE
          );
    TYPE AngCursorTyp IS REF CURSOR RETURN AngRecTyp;

    PROCEDURE GetAngOfAbt ( AbtNo IN NUMBER, AngCursorRef IN OUT
   AngCursorTyp);
END AngestelltenVerwaltung;
```

Package: Beispiel

```
CREATE OR REPLACE PACKAGE BODY AngestelltenVerwaltung AS
...
 PROCEDURE GetAngOfAbt  ( AbtNo IN NUMBER,
                          AngCursorRef IN OUT AngCursorTyp)
   IS
   BEGIN
      OPEN AngCursorRef FOR
         SELECT ang.name, abt.name, ang.salaer
         FROM angestellter ang, abteilung abt
         WHERE ang.abtnr=abt.abtnr
         AND  ang.abtnr = AbtNo;
   END GetAngOfAbt;

END AngestelltenVerwaltung;
```

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Objektrelationale Datenbanksysteme

Teil 1

Prof. S. Keller

Dank an Prof. H. Huser

Warum objektrelationale Datenbanken?

- Welche Schwächen von relationalen Datenbanken haben Sie bisher entdeckt? Hier einige Fragen…
- Man denke an das Angestellte-Projekte-Datenmodell:
  - Wie würde man mehrere TelNummern modellieren, die zu einem Angestellten gehören?
    - Es gibt verschiedene Lösungen: TelNummer als Zeichenkette, als Typ, als Referenz...
  - Wie kann man Person - Angestellte direkter modellieren?
  - Kann man Identität unabhängig von den Werten des Tupels haben?
- Wie würde man ein Koordinaten-Paar behandeln?
  - Können auch Operationen dazu abgelegt werden?

Warum objektrelationale Datenbanken?

- Schwächen von relationalen Datenbanken
  - „Impedance Mismatch“:
    - Unterschiedliche Typensysteme zwischen OO-Programmiersprachen und der relationalen DB-Welt
    - Mengen-Ansatz versus Objekt-/Pfad-Ansatz
    - Beziehungen Ganzes-Objekt zu seinen Teilen umgekehrt
    - etc. (siehe später OO-DBMS)
  - Relationales Modell ungeeignet für die Darstellung von komplexeren Datenstrukturen
  - Relationales Modell unterstützt keine benutzerdefinierten Datentypen und Operationen in der Datenbank
  - Referenzen / Identifiers

OO-Ansätze für Datenbanksysteme

- Objektorientierte Datenbanksysteme
  - OO-Programmiersprache + DB-Konzepte
  - Neue Technologie, Migration von bestehenden Datenbanken schlecht möglich

- Objektrelationale Datenbanksysteme
  - Relationale DBMS + OO-Konzepte
  - Evolutionärer Ansatz, kompatibel mit bestehenden Datenbanken

Objektrelationales Modell

- Funktionale Erweiterung des relationalen Modelles um OO-Strukturen
- Strukturelle Objektorientierung:
  - Standardisierung als SQL:99 im Rahmen des SQL-3-Projektes
  - Komplex strukturierte benutzerdefinierte Typen, die von Typkonstruktoren erzeugt werden können
  - Objektidentifikatoren (OID) und Referenzen für die (persistente) Referenzierung von Objekten.
  - Spezialisierung auf Objekt- und Klassenebene

Objektrelationale Erweiterungen in Oracle

- Objektrelationale Konzepte (seit Oracle 8i/9i)
  - Benutzerdefinierte Objekttypen mit Methoden
  - Objekttabellen
  - OIDs und Objekt-Referenzen
  - Aggregationen (Collections)
  - Object Views: Verknüpfung von Objekttypen mit relationalen Tabellen
  - Objekttyp-Vererbung

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Objekttypen

Objekttypen (Object Types)

- Ein benutzerdefinierter Objekttyp
  - *Abstract Data Type* (ADT),
    oft auch *User Defined Type* (UDT) genannt
- wird definiert durch
  - den Namen des Typs
  - die Attribute (Name, Typ)
  - die Methoden
- Ein Objekttyp wird verwendet für
  - die Definition von Objekttabellen
  - die Definition von Attributen in relationalen oder in Objekttabellen

Deklaration von Objekttypen

```
CREATE OR REPLACE TYPE NameTyp AS OBJECT (
    NStr            VARCHAR2(20)
);
CREATE OR REPLACE TYPE AdresseTyp AS OBJECT (
    WOHNORT         VARCHAR2(20),
    PLZ             VARCHAR2(10),
    Land            CHAR(2)
);
CREATE OR REPLACE TYPE PersonenTyp AS OBJECT (
    NName           NameTyp,
    VName           VARCHAR2(20),
    GebDatum        DATE,
    Adr             AdresseTyp
);
```

Achtung: In Typ-Deklarationen können keine Constraints und keine Defaultwerte definiert werden!

Typen in relationalen Tabellen: Column Objects

- Typen können auch für die Definition von (komplexen) Attributen in relationalen Tabellen verwendet werden. Die Werte dieser Attribute werden als Column-Objects bezeichnet.
- Achtung: Tupels der relationalen Tabellen haben keine OID's. Es können daher keine Referenzen auf solche Tupels definiert werden!

Beispiel

```
CREATE TABLE Angestellter (
  PersNr INTEGER PRIMARY KEY,
  PersInfo PersonenTyp,
  Salaer NUMBER NOT NULL
);
```

Operationen auf Tabellen mit Column-Objects

Beispiel: Einfügen eines Tupels mit Column-Object (Konstruktor)

```
INSERT INTO Angestellter
VALUES (100, PersonenTyp(
    'Graf',
    'Michael',
    AdresseTyp ('Thalwil', '8810', 'CH'),
    To_Date('22.01.78', 'DD.MM.YY')),
    8888);
```

Beispiel: Abfragen auf Tabellen mit Column-Objects

```
SELECT * FROM Angestellter;
SELECT persnr,a.persinfo.nname,a.persinfo.GibAlter()
    FROM Angestellter a;
```

Konstruktoren

```
INSERT INTO Person VALUES (
    NameTyp ( 'Mueller' ),
    'Hans',
    TO_DATE( '12.01.78', 'DD.MM.YY' ),
    AdresseTyp ( 'Rapperswil', '8640', 'CH' )
);
INSERT INTO Person (NName, VName, GebDatum) VALUES (
    NameTyp ( 'Meier' ),
    'Vreni',
    TO_DATE( '09.02.62', 'DD.MM.YY' )
);
```

- Zu jedem Typ wird ein Default-Konstruktor generiert
- Es gibt (noch) keine anderen Konstruktoren
- Beim Erzeugen von Tupeln müssen die Werte von Attributen mit Objekttypen mit dem Konstruktor generiert werden.

SQL-Erweiterungen: SELECT

Liste der Objekte mit komplexen Werten:

```
SELECT * FROM person;
```

Zugriff auf die objektwertigen Attribute mit dem ‚Punkt'-Operator:

```
SELECT t.NName, t.ADR.PLZ FROM Person t;
```

Gleichheitsoperator:

```
SELECT * FROM Person
WHERE adr = AdresseTyp('Thalwil', '8810', 'CH');
```

Achtung: Der „ORDER BY"- und die Vergleichs-Operatoren sind auf den benutzerdefinierten Typen *nicht* definiert (siehe MAP- und ORDER Operator)

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Vererbung: Objekttabellen

Vererbung

- Polymorphismus, Überschreiben von Methoden und Dynamische Bindung können angewendet werden
- FINAL / NOT FINAL: Vererbung kann gezielt verboten bzw. erlaubt werden
  - bei Typ- und Methodendeklaration spezifizieren
  - Defaults: FINAL bei Typen, NOT FINAL bei Methoden
- INSTANTIABLE / NOT INSTANTIABLE: Zur Definition von abstrakten Typen und Methoden
  - bei Typ- und Methodendeklaration spezifizieren
  - Defaults: INSTANTIABLE bei Typen und Methoden
- UNDER-Klausel bei Typdeklaration
- OVERRIDING-Klausel bei Methodensignatur

Objekttabellen (Object Tables)

```
CREATE TABLE Person OF PersonenTyp (
     NName    NOT NULL,
     Adr      DEFAULT AdresseTyp( NULL, NULL, 'CH' ),
     GebDatum NOT NULL,
     CONSTRAINT Name_UNIQUE
         UNIQUE( NName.NStr, VName, Adr.PLZ )
);
```

- Objekttabellen sind Instanzen von Objekttypen
- Bei der Deklaration der Objekttabellen können Constraints und Defaults angegeben werden
- Jedes Tupel (= Row-Object) hat einen OID.

Beispiel

```
CREATE OR REPLACE TYPE PersonenTyp AS OBJECT (
  name VARCHAR(40),
  adresse VARCHAR(100),
  geb_dat DATE,
  MEMBER PROCEDURE druckeAdresse
) NOT FINAL;
CREATE OR REPLACE TYPE StudentTyp UNDER PersonenTyp(
  matrikelnr CHAR(7),
  stud_adresse VARCHAR(100),
  OVERRIDING MEMBER PROCEDURE druckeAdresse
);

CREATE TABLE Person OF PersonenTyp (...);
  /* kann Instanzen von Person und Student aufnehmen
  */
```

Anfragen / Queries

- Um in einer Objekttabelle auf Attribute von Subtypen zugreifen zu können, stellt Oracle die Funktion TREAT (... AS type) zur Verfügung (nur im SELECT-Teil einer SQL-Query erlaubt):
- Beispiel:
  - SELECT TREAT(VALUE(p) AS PersonenTyp).VName
  - FROM person p;
- Um in einer SQL-Query den Typ eines Objektes zu ermitteln stellt Oracle das Prädikat IS OF (type) zur Verfügung.
- Beispiel:
  - SELECT TREAT(VALUE(p) AS PersonenTyp).VName
  - FROM person p WHERE VALUE(p) IS OF (PersonenTyp);
- Dieses Query wird nur für Objekte des Typs PersonenTyp ausgewertet

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Methoden

Methoden für Objekttypen

- Zu Objekttypen können Methoden definiert werden
- Einschränkungen: Methoden nur für Lesezugriff
- Kapselung von Attributen wird nicht unterstützt, d.h. Attribute sind immer global sichtbar

Beispiel: Typ-Deklaration (inkl. Deklaration von gibAlter):

```
CREATE OR REPLACE TYPE PersonenTyp AS OBJECT (
      NName       VARCHAR2(20),
      VName       VARCHAR2(20),
      Adr         AdresseTyp,
      GebDatum    DATE,
      MEMBER FUNCTION gibAlter RETURN NUMBER,
      ORDER MEMBER FUNCTION
            PersonenOrdnung (pers IN PersonenTyp)
            RETURN INTEGER
);
/
```

Implementation und Verwendung der Methoden

Implementation in der separat übersetzbaren Typdefinition

```
CREATE OR REPLACE TYPE BODY PersonenTyp AS
    MEMBER FUNCTION gibAlter RETURN NUMBER IS
        BEGIN
            RETURN (SYSDATE - SELF.GebDatum) / 365;
        END;
    -- ...
END;
```

Methoden können in PL-SQL oder in Java (auf dem Server) oder als externe Funktionen in C etc. (ausserhalb Server) implementiert werden

Aufruf der Methoden aus SQL:

```
SELECT p.NName,p.GibAlter()
FROM Person p;
```

Methoden können auch aus PL-SQL oder Java (auf dem Server) aufgerufen werden.

Vergleichsmethoden: MAP und ORDER

- Für den Test auf Gleichheit/Ungleichheit von Attributwerten mit benutzerdefiniertem Typ ist entweder eine MAP- oder eine ORDER-Methode nötig.
- MAP-Methode liefert einen Basistyp mit einer definierten Sortierordnung

```
MAP MEMBER FUNCTION VergleichsWert RETURN NUMBER;
```

- ORDER-Methode vergleicht zwei Objekte und liefert `-1 (bzw. neg. Zahl)` für `<`, `+1` für `>` und `0` für `==`

```
ORDER MEMBER FUNCTION
  PersonenOrdnung (pers IN PersonenTyp)
  RETURN INTEGER;
```

- Verwendung:

```
SELECT * FROM Person p ORDER BY VALUE(p);
```

D.h. `ORDER...` wertet hier die Vergleichsmethode aus.

Vergleichsmethoden (ff.)

- Bei der Definition eines Objektyps kann man entweder eine MAP oder eine ORDER-Methoden definieren (aber nicht beides)
- Zum Vergleich:
  - Oracle compares two objects of a type that lacks a comparison method by comparing corresponding attributes:
    - If all the attributes are non-null and equal, the objects are considered equal.
    - If there is an attribute for which the two objects have unequal non-null values, the objects are considered unequal.
    - Otherwise, the objects are considered unequal.
  - Because the system can perform scalar value comparisons very efficiently, coupled with the fact that calling a user-defined function is slower than calling a kernel implemented function, sorting objects using the ORDER method is relatively slow compared to sorting the mapped scalar values returned by the MAP function.

Beispiel: Implementation der ORDER-Methode

```
CREATE OR REPLACE TYPE BODY PersonenTyp AS
  ...
ORDER MEMBER FUNCTION PersonenOrdnung (pers IN PersonenTyp)
RETURN INTEGER
     IS
       SelfPerson VARCHAR2(80) := SELF.NName || SELF.VName;
       persPerson VARCHAR2(80) := pers.NName || pers.VName;
       BEGIN
         IF selfPerson < persPerson THEN RETURN -1;
           ELSIF selfPerson > persPerson THEN RETURN  1;
           ELSE  RETURN 0;
         END IF;
         -- Schreibfaule könnten auch schreiben:
         -- RETURN ( selfPerson - persPerson );
       END;
END;
/
```

Beispiel einer MAP-Methode

MAP: Bildet Objektwerte auf Werte von Basisdatentypen (Zahlen, Zeichenketten, Datum) ab (benötigt kein „other“ Object): Syntax:
`MAP MEMBER FUNCTION Funktioname RETURN Typ`

```
CREATE TYPE Complex AS OBJECT (
   rpart REAL;
   ipart REAL;
   MAP MEMBER FUNCTION convert RETURN REAL,
... );
CREATE TYPE Complex AS OBJECT (
   rpart REAL;
   ipart REAL;
   ORDER MEMBER FUNCTION match(c Complex)
     RETURN INTEGER,
... );
CREATE TYPE BODY Complex AS MAP MEMBER FUNCTION convert
   RETURN REAL IS
   BEGIN
        RETURN SQRT(rpart*rpart + ipart*ipart);
   END convert;
... END;
```

Vererbung: Einschränkungen

- Oberster Typ einer Hierarchie muss zumindest *ein* Attribut definieren
- “Leere Typen” (keine Attribute, keine Methoden) sind nicht erlaubt
- PL/SQL stellt nur das Schlüsselwort SELF, nicht aber SUPER zur Verfügung
- Spezialisierung von Methoden nur durch vollständiges überschreiben möglich
- Keine SUPER-Aufrufe (d.h. Ausführen der Implementierung einer Methode beim Basistyp) möglich
- Änderungen der Objekttabellen sind nur möglich über eine Änderung des Typs (keine Typ-Evolution)

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Objekt-Referenzen

Object Identifiers (OIDs)

- Objekttabellen: Jedes ‚Row-Object' wird mit einem systemgenerierten 16 byte-OID systemweit eindeutig identifiziert (der Default).
- Alternativ können die klassischen Primärschlüssel als OIDs verwendet werden. Diese OIDs sind aber *nicht* systemweit eindeutig, benötigen dafür weniger Platz!

Objekttabelle mit systemgenerierten OID's:

```
CREATE TABLE Person1 OF PersonenTyp (
     NName          NOT NULL
) OBJECT IDENTIFIER IS SYSTEM GENERATED;
```

Objekttabelle mit Primärschlüssel als OID's:

```
CREATE TABLE Person2 OF PersonenTyp (
  CONSTRAINT Person2_PK PRIMARY KEY( NName.NStr )
) OBJECT IDENTIFIER IS PRIMARY KEY;
```

Objektreferenzen (REF/DEREF)

- Objektreferenzen speichern OID des referenzierten Objektes (genauer Row-Object in einer Object-Table)
- Navigation über Dereferenzierung der Objektreferenz
- Objektreferenzen ersetzen Fremdschlüssel-beziehungen in den relationalen Tabellen
- Objektreferenzen sind typisiert. Sie können Tupels desselben Typs referenzieren. Diese Tupels können in unterschiedlichen Tabellen sein.

Beispiel

Ziel sei die Modellierung einer Ganzes-Teile-Struktur am Beispiel eines Velohändlers, der Velos und seine Einzelteile, bzw. Ersatzteile verwalten will.

Beispiel: Object-Types mit Referenzen

```
CREATE TYPE TeilTyp AS OBJECT (
  TName VARCHAR2(20)
);

CREATE TYPE TeileStrukturTyp AS OBJECT (
  OberTeil  REF TeilTyp,
  UnterTeil REF TeilTyp,
  Menge integer
);
```

`OberTeil` und `UnterTeil` sind Referenzen auf Instanzen vom Typ `TeilTyp`.

Diese allgemeinen Referenzen haben den Nachteil, dass die referentielle Integrität nicht überprüft wird!

Beispiel: Definition der Object-Tables

```
CREATE TABLE Teil OF TeilTyp (
  TName PRIMARY KEY
);

CREATE TABLE TeileStruktur OF TeileStrukturTyp (
 SCOPE FOR (OberTeil) IS Teil,
 SCOPE FOR (UnterTeil) IS Teil
);

INSERT INTO TEIL VALUES ('Velo');
INSERT INTO TEIL VALUES ('Rahmen');
INSERT INTO TEIL VALUES ('Rahmenstange');
INSERT INTO TEIL VALUES ('Rahmenzylinder');
```

SCOPE FOR schränkt den Wertebereich der Referenzen auf Tupels in der Tabelle Teil ein (Optimierung)

Beispiel: Einfügen von Tupels mit Referenzen

```
INSERT INTO TeileStruktur VALUES (
  ( SELECT REF(t) FROM Teil t
    WHERE t.TName = 'Velo' ),
  ( SELECT REF(t) FROM Teil t
    WHERE t.TName = 'Rahmen' ),  1
);
INSERT INTO TeileStruktur VALUES (
  ( SELECT REF(t) FROM Teil t
    WHERE t.TName = 'Rahmen' ),
  ( SELECT REF(t) FROM Teil t
    WHERE t.TName = 'Rahmenstange' ),  1
);
INSERT INTO TeileStruktur VALUES (
  ( SELECT REF(t) FROM Teil t
    WHERE t.TName = 'Rahmen' ),
  ( SELECT REF(t) FROM Teil t
    WHERE t.TName = 'Rahmenzylinder'), 1
);
```

REF(t): liefert Referenz auf Tupel t

Dereferenzierung von Referenzen

`REF(t)` : berechnet die Referenz (=OID) eines Row-Objects
`DEREF(ref)` : dereferenziert eine Referenz `ref` und liefert Zugriff auf das Row-Object
`myRef.NameId`: dereferenziert eine Referenz `myRef` und liefert Zugriff auf das Element `NameId` des Row-Objects

Beispiel:

```
SELECT DEREF(t.Oberteil) from Teilestruktur t;
```

Beispiel:

```
SELECT t.Oberteil.TName, t.UnterTeil.TName
FROM TeileStruktur t;
```

Man beachte: Die Dereferenzierung mit dem '.'-Operator ersetzt den klassischen Join (!):

```
SELECT t.Oberteil.TName FROM TeileStruktur t
WHERE t.OberTeil IS NOT DANGLING;
```

Dangling References

- Oracle überprüft bei den Referenzen die Einhaltung der referentiellen Integrität nicht.
- Allerdings wird für die Überprüfung von “dangling“ Referenzen das Pädikat
  `IS[NOT]DANGLING` unterstützt.
- Bei der Definition einer Tabelle können aber normale referentielle Bedingungen über Referenzen definiert werden

Beispiele

```
-- Erzeuge dangling-Referenzen
DELETE FROM TEIL WHERE TNAME = 'Velo';

-- Gib die Unterteile aus ohne Oberteile
SELECT t.UnterTeil.TName FROM TeileStruktur t
WHERE t.OberTeil IS DANGLING
```

Constraints

- REF‘s können mit folg. Constraints versehen werden:
  - SCOPE, REFERENCES, NOT NULL
  - Bsp.: Die Ref‘s Oberteil und Unterteil werden als Fremdschlüssel deklariert (keine Dangling Refs mehr möglich)

```
CREATE TABLE TeileStruktur OF TeileStrukturTyp
(
  CONSTRAINT fk_OT
        FOREIGN KEY (OberTeil)  REFERENCES Teil
        ON DELETE CASCADE,
  CONSTRAINT fk_UT
        FOREIGN KEY (UnterTeil) REFERENCES Teil
        ON DELETE CASCADE
);
```

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Kollektionen / Collections

VARRAYs und NESTED TABLEs

Collection Types: VARRAYS

- Geordnete Menge von Datenelemente, max. Anzahl Elemente
- Normalerweise „in line“ im Tupel gespeichert, ab einer gewissen Grösse wird der VARRAY als BLOB ausserhalb des Tupels gespeichert

```
CREATE OR REPLACE TYPE TelNummerTyp AS OBJECT (
    Ort VARCHAR2(20),
    Nummer VARCHAR2(12)
);
/

CREATE OR REPLACE TYPE TelNummerListeTyp AS
    VARRAY(10) OF TelNummerTyp;
/
```

VARRAYS: Beispiel

Beispiel: TelListe als VARRAY:

```
CREATE OR REPLACE TYPE PersonenTyp1 AS OBJECT (
     Nname        VARCHAR2(20),
     Vname        VARCHAR2(20),
     Adr          AdresseTyp,
     GebDatum     DATE,
     TelListe     TelNummerListeTyp
);/
```

Definiere eine Tabelle `Person4` mit Tupeln vom Typ `PersonenTyp1`.
Das VARRAY-Attribut `TelListe` wird mit 0 Elementen initialisiert:

```
CREATE TABLE Person4 OF PersonenTyp1 (
     NName        NOT NULL,
     Adr          DEFAULT AdresseTyp (NULL, NULL, 'CH'),
     GebDatum     NOT NULL,
     TelListe     DEFAULT TelNummerListeTyp ()  -- leere L.
);/
```

VARRAY: DML

```
INSERT INTO Person4 VALUES (
    'Mueller', 'Hans',
    AdresseTyp ('Rapperswil', '8640', 'CH'),
    To_Date('12.01.1978', 'DD.MM.YYYY'),
    TelNummerListeTyp ( TelNummerTyp('G', '01 766 23 21'),
                        TelNummerTyp('P', '01 233 44 23'))
);

SELECT * FROM person4;

SELECT p.VNAME || ' ' || p.NName, t.*
    FROM person4 p, TABLE (p.TelListe) t;

SELECT p.VNAME || ' ' || p.NName, t.Nummer
    FROM person4 p, TABLE (p.TelListe) t
    WHERE t.Ort='G';
```

TABLE(..):
Unnesting-Operator, wandelt VARRAY in relationale Darstellung um

VARRAY: DML

- Update, Delete, Index? Geht in Oracle nicht (als sep. Tabelle implementieren oder Stored Procedure)

```
SQL> create table employees ( ename varchar2(15), addr address);
Table created.

SQL> insert into employees values (
   2 'Howard Rogers',
   3 Address ('15 Acacia Avenue','Blogsville','NSW 2156'));
1 row created.

SQL> select * from employees;
ENAME ADDR
--------------- ----------------------------------------------------
Howard Rogers   ADDRESS('15 Acacia Avenue', 'Blogsville', 'NSW 2156')

SQL> select addr from employees;
ADDR
--------------------------------------------------------------------
ADDRESS('15 Acacia Avenue', 'Blogsville', 'NSW 2156')

SQL> select addr(1) from employees;
select addr(1) from employees
*
ERROR at line 1:
ORA-00904: "ADDR": invalid identifier  // kein Zugriff auf Teile von VARRAYs

SQL> update employees set addr=
2 Address('16 Acacia Avenue','Blogsville','NSW 2156')
3 where ename='Howard Rogers';
1 row updated.  // Man beachte: Das Feld als Ganzes muss geupdated werden
```

PostgreSQL: Mit Array-Operator [] zugreifen

Nested Tables

- Tabellentyp: Typ einer Tabelle mit Object-Rows
- Attribute einer Tabelle können einen Tabellentyp aufweisen, die Attributwerte werden in einer separaten Tabelle gespeichert.

Beispiel: TelListe als Nested Table

Definiere Tabellentyp TelNummerTabTyp:

```
CREATE TYPE TelNummerTabTyp AS TABLE OF TelNummerTyp;
```

Nested Tables: Beispiel

Definiere Typ mit Nested Table Attribut TelListe:

```
CREATE OR REPLACE TYPE PersonenTyp2 AS OBJECT (
     Nname          VARCHAR2(20),
     Vname          VARCHAR2(20),
     Adr            AdresseTyp,
     GebDatum       DATE,
     TelListe       TelNummerTabTyp
);
```

Definiere Tabelle `Person5`. Das Nested Table Attribut `TelListe` wird in der separaten Tabelle `TelListe_Table` gespeichert:

```
CREATE TABLE Person5 OF PersonenTyp2 (
     NName          NOT NULL,
     Adr            DEFAULT AdresseTyp (NULL, NULL, 'CH'),
     GebDatum       NOT NULL,
     TelListe       DEFAULT TelNummerTabTyp () -- leere Tab.
) NESTED TABLE TelListe STORE AS TelListe_Table;
[vgl. IOT und andere Syntax im Teil 2]
```

Nested Tables: Speicherstrukturen

DATA1	DATA2	DATA3	DATA4	NT_DATA
...	...	...	...	A
...	...	...	...	B
...	...	...	...	C
...	...	...	...	D
...	...	...	...	E

Storage Table

NESTED_TABLE_ID	Values
B	B21
B	B22
C	C33
A	A11
E	E51
B	B25
E	E52
A	A12
E	E54
B	B23
C	C32
A	A13
D	D41
B	B24
E	E53

The figur shows how the storage table works. The storage table contains each value for each nested table in a nested table column. Each value occupies one row in the storage table. The storage table uses the NESTED_TABLE_ID to track the nested table for each , all of the value. So, in the figur, values that belong to nested table A are identified, all of the values that belong to nested table B are identified, etc.

Nested Tables: DML

Füge Tupel mit zwei Tel.-Nummern ein:

```
INSERT INTO Person5 VALUES (
   'Mueller', 'Hans',
   AdresseTyp ('Rapperswil', '8640', 'CH'),
   To_Date('12.01.1978', 'DD.MM.YYYY'),
   TelNummerTabTyp ( TelNummerTyp('G', '01 766 23 21'),
                     TelNummerTyp('P', '01 233 44 23') )
);
```

Füge einen weiteren Telefon-Eintrag dazu:

```
INSERT INTO TABLE (
    SELECT p.TelListe FROM Person5 p
        WHERE p.NName = 'Mueller')
    VALUES ('Handy', '079 265 45 09')
);
```

Nested Tables: DML

Lösche TelListe:

```
UPDATE Person5 SET TelListe = NULL
WHERE NName = 'Graf';
```

Erzeuge neue Tabelle mit einem Eintrag:

```
UPDATE Person5 SET TelListe =
    TelNummerTabTyp (
        TelNummerTyp('P', '01 233 54 33') )
    WHERE NName = 'Graf';
```

Relationale Abfragen mit dem TABLE-Operator

```
SELECT p.nname, q.*
    FROM person5 p, TABLE(p.telliste) q;

SELECT p.nname,q.nummer
    FROM person5 p, TABLE(p.telliste) q
    WHERE q.ort='Handy';
```

Nested Tables: Beispiel Methode

```
CREATE OR REPLACE TYPE BODY PersonenTyp2 AS
MEMBER FUNCTION summiereTelNummern
      (ArgOrt VARCHAR2 DEFAULT NULL)
  RETURN NUMBER
  IS
      i              INTEGER;
      Total          NUMBER := 0;
  BEGIN
    FOR i in 1..SELF.TelListe.COUNT LOOP
        IF (ArgOrt IS NULL ) OR
           (SELF.TelListe(i).Ort=ArgOrt) THEN
          Total := Total+1;
        END IF;
      END LOOP;
      RETURN Total;
   END;
END;
```

COUNT liefert Anzahl Einträge in der Nested Table

Index-Operator erlaubt Zugriff auf die Elemente in der Nested Table

Vergleich VARRAY – NESTED TABLE

- VARRAY ist sortiert, NESTED TABLE ist nicht sortiert
- VARRAY ist in Grösse limitiert, NESTED TABLE nicht
  - Wenn Grösse im voraus bekannt, VARRAY nutzt Speicherplatz besser
- Auf NESTED TABLES können Queries ausgeführt werden
- VARRAY‘s bekommt man immer als Ganzes
- VARRAY‘s können keine LOB‘s enthalten
- VARRAY‘s sind normalerweise „in line“ gespeichert

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Objektrelationale Datenbanksysteme

Teil 2

Prof. S. Keller

Dank an Prof. H. Huser

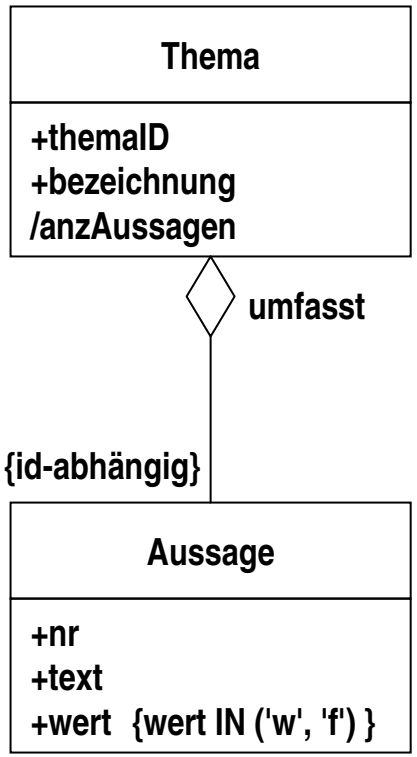
HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Beispiel Relational - Objektorientiert

Realisierung gem. relationalem Paradigma

UML-Diagramm

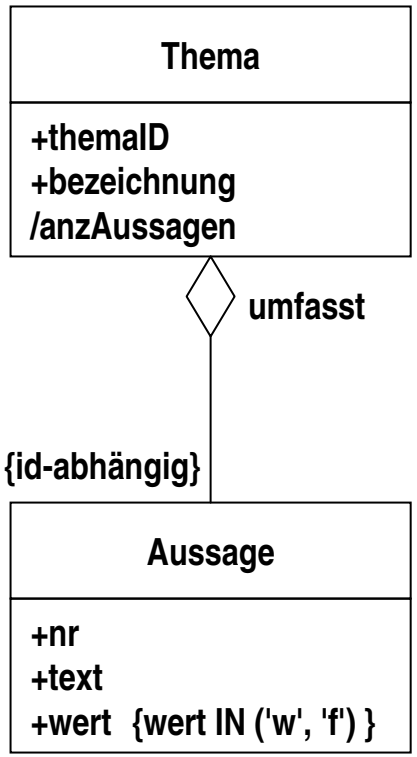


relationale Umsetzung

```
CREATE TABLE Themen (
  themaId CHAR(5) PRIMARY KEY,
  bezeichnung VARCHAR(30) NOT NULL
);
CREATE TABLE Aussagen (
  themaId CHAR(5) NOT NULL
    REFERENCES Themen(themaId)
    ON DELETE CASCADE,
  nr NUMBER(2) NOT NULL,
  text VARCHAR(256) NOT NULL,
  wert CHAR(1) NOT NULL
    CHECK (wert IN ('w','f')),
    PRIMARY KEY (themaId, nr)
);
```

Realisierung gem. OO-Paradigma

UML-Diagramm



objektorientierte Umsetzung (bidirektional)

```
CLASS Thema (
  themaID String,
  bezeichnung String,
  aussagen {Aussage}
);
CLASS Aussage (
  thema Thema,
  nr Integer,
  text String,
  wert Char
);
// Constraints:
// - not null?
// - wert IN ('w','f') fehlen
```

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Beispiel zum Vergleich relational vs. objekt-relational

Beispiel Bestellung

- Entitätstypen
  - Kunde
  - Bestellung
  - BestellPosition
  - Teile
- Variante relational
- Variante objekt-relational

Beispiel Bestellung - Nested Table I

Objekttyp für die NESTED TABLE

```
CREATE OR REPLACE TYPE Bestellpos_objtyp
    AS OBJECT (
        BestellposNr    NUMBER,
        Teil            REF Teil_objtyp,
        Anzahl          NUMBER
);
```

Objekttabelle für NESTED TABLE

```
CREATE OR REPLACE TYPE BestellposList_ntabtyp
    AS TABLE OF Bestellpos_objtyp;
```

Beispiel Bestellung - Nested Table II

Definiere Typ mit Nested Table-Attribut **`BestellposList`** :

```
CREATE OR REPLACE TYPE Bestellung_objtyp AS OBJECT (
  BestNr                NUMBER,
  …
  BestellposList        BestellposList_ntabtyp,
  …
  MEMBER FUNCTION
      sumBestellpos RETURN NUMBER
)
```

Definieren der Tabelle: Das Nested Table-Attribut `BestellposList` wird in der separaten Tabelle `TelListe_Table` gespeichert:

```
CREATE TABLE Bestellung_tab OF Bestellung_objtyp (
  …
) NESTED TABLE BestellposList STORE AS Bestellpos_ntab (   //1
     ( PRIMARY KEY( NESTED_TABLE_ID, BestellposNr ) )      //2
      ORGANIZATION INDEX COMPRESS )                        //3
RETURN AS LOCATOR;                                         //4
```

Beispiel Bestellung - Nested Table III

Linie 1: Das Nested Table Attribut `BestellposList` wird in der separaten Tabelle `Bestellpos_ntab` gespeichert.

Linie 2: - dient als Key für den Index Organized Table (`IOT`)
- erzwingt Eindeutigkeit der Spalte BestellposNr der `NESTED TABLE` innerhalb jeder Zeile der Parent Tabelle. (garantiert also, dass eine BestellPos nur einmal in einer Bestellung sein kann.)

Linie 3: Tabelle ist INDEX – organisiert (`IOT`). Dies ermöglicht das Clustering von Zeilen welche zu selben Zeile in der Parent Tabelle gehören.

Linie 4: Gibt an, dass die `NESTED TABLE` als `LOCATOR` zurückgegeben wird. Default wäre `VALUE`, was bedeutet, dass immer die ganze `NESTED TABLE` zurückgegeben wird, anstelle nur eines `LOCATOR's` . Dies ist nicht sehr effizient. Wenn man hingegen `LOCATOR` als Rückgabewert angibt, bekommt man nur die Werte der aktuellen Collection. Über das Interface `UTL.COLL.IS_LOCATOR` kann herausgefunden werden, ob man einen `LOCATOR` bekommen hat.

Beispiel Bestellung – Methode `sumBestellpos` I

```
MEMBER FUNCTION sumBestellpos
    RETURN NUMBER
IS
   i           INTEGER;
   einTeil     Teil_objtyp;
   Total       NUMBER := 0;

BEGIN
   IF ( UTL_COLL.IS_LOCATOR(BestellposList) ) -- check for locator
      THEN
         SELECT SUM( L.Anzahl * L.Teil.Preis ) INTO Total
         FROM   TABLE( CAST(BestellposList AS BestellposList_ntabtyp) ) L;
   ELSE
      FOR i in 1..SELF.BestellposList.COUNT LOOP
         UTL_REF.SELECT_OBJECT(BestellposList(i).Teil,einTeil);
         Total := Total + SELF.BestellposList(i).Anzahl * einTeil.Preis;
      END LOOP;
   END IF;
   RETURN Total;
END;
/
```

Beispiel Bestellung – Methode `sumBestellpos` II

- SELF gibt es für jede Funktion
  - Ermöglicht den Zugriff auf die Elemente des aufrufenden Objektes
- COUNT
  - Gibt die Anzahl Elemente in der Collection (mit `SELF.X.COUNT` wird der Bereich eingeschränkt)
- UTL_REF ist ein Package von Oracle zum Handeln von REF's
  - Ist nötig weil Oracle kein implizites Dereferenzieren unterstützt
  - `SELECT_OBJECT` gibt das eigentliche Objekt zurück
- CAST
  - Wird gebraucht um dem SQL-Compiler den Typ der Collection bekannt zu geben.

MAP MEMBER Fn

```
MAP MEMBER FUNCTION vergleicheAng RETURN
   VARCHAR2 is
  BEGIN
    RETURN Name || SALAER;
  END;
```

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Diskussion OR-DBMS

Weiterentwicklung der OR-DBMS

- Bessere Unterstützung der Norm allgemein
  - Aktuelle Lösungsmöglichkeiten: ...
- Spezielle Typen: Vererbung
  - Aktuelle Lösungsmöglichkeiten: ...
- Speziell Operationen: Stored Procedures, Triggers

  Aktuelle Lösungsmöglichkeiten: ...
  - Stored Procedures
  - Triggers
- Updateable Views

  Aktuelle Lösungsmöglichkeiten: ...

Vergleich R – OR – OO

- Relationenmodell (Entity Relationship-M.):
  - alle Info. über Entitäten und Beziehungen (Def. in Tabellen)
  - nur einfache (= vordefinierte) Datentypen (1. Normalform)
  - Definition von Datenstruktur in Tabellen, Definition von Verhalten in separaten Funktionen, Packages
- Objektrelationales Modell:
  - Informationen in Entitäten/Objekten (Def. in Objekttabellen)
  - Definition von Struktur und Verhalten in komplexen Objekttypen
  - teilw. Rückwärtskompatibilität mit Relationenmodell

Weitere: OO- und weitere Modelle

ER – OR – OO (ff.)

- Objektorientiertes Modell:
  - alle Informationen in Objekten (Def. in Klassen)
  - Definition von Struktur und Verhalten nahe beisammen
  - komplexe (= benutzerdefinierte) Datenstrukturen

Weitere Datenbankmodelle für die Realisierung

- Netzwerkmodell und hierarchisches Modell
- Semistrukturierte Datenmodelle (XML)
- Mehrdimensionale Datenmodelle
  - Data Ware Houses (und OLAP)

Bewertung

- OR-DB sind besser als ER-DB...
- SQL:99 ist nicht mehr relational!
- Implementationen
  - könnten aber noch besser sein:
    noch viele Einschränkungen bei den unterstützten SQL:99- und weiteren Konzepten
  - Die meisten bekannten Produkte unterstützen OR-DB, also Oracle, PostgreSQL, DB2, (MySQL hingegen kennt keine objektrelationalen Erweiterungen)

Zusammenfassung

- Objektrelationale Konzepte
  - Objekttypen
  - Methoden
  - Objekttabellen
  - Referenzen (REF)
  - Kollektionen/Aggregationen (Collections)
  - Nicht oder nur oberflächlich behandelt
    - Object Views: (Sichten) Verknüpfung von Objekttypen mit relationalen Tabellen
    - Schemas

Ausblick

- Eigene ADT/UDTypes
- Ausgewählte ADT-Anwendungen: SQL MM/Spatial
- DB-Programmierung (ff.): JDBC und OR-DBMS
- Objektorientierte DBMS

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Datenbank-Programmierung: JDBC ff.

Datenbanksysteme 2

Prof. Stefan Keller

Inhaltsüberblick

1. Rückblick und Refresher (diese Folien)
2. JDBC und O/R-Datentypen
3. Stored Procedures in Java (Oracle) und Python (PostgreSQL)
4. (Ev.) Data Access Patterns

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

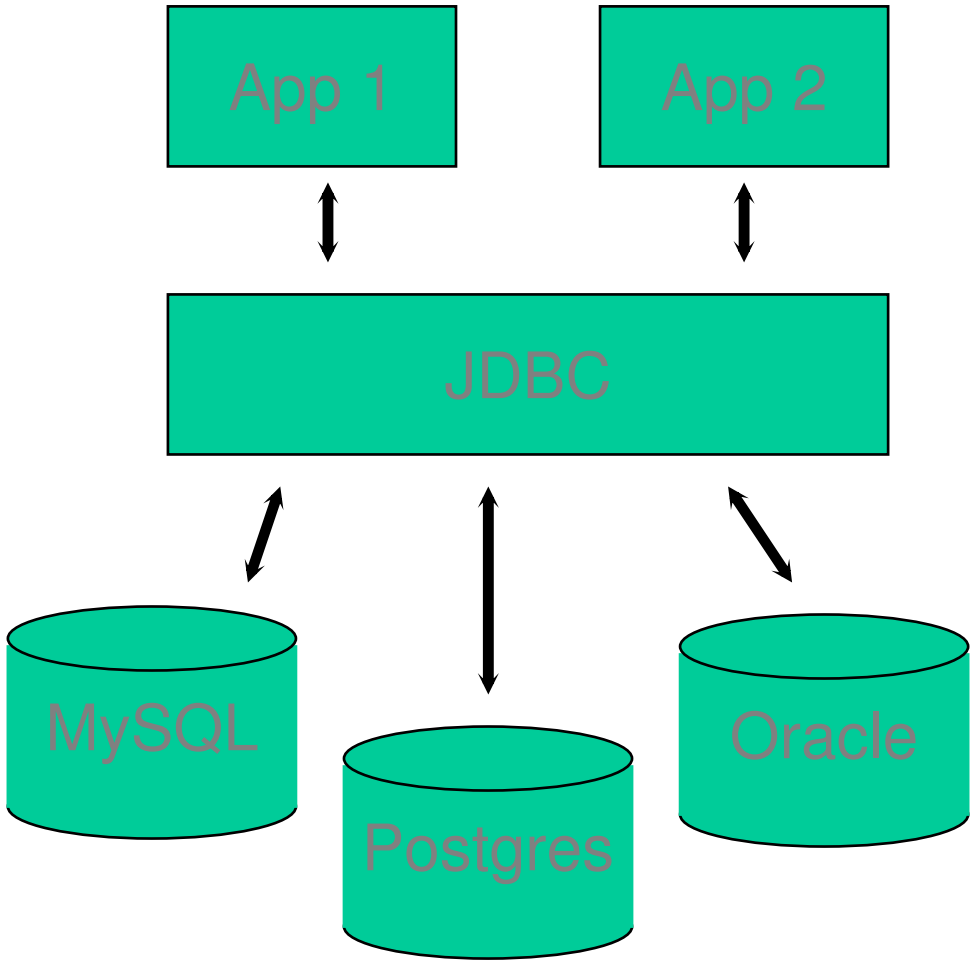
JDBC - Refresher

Was Sie bereits kennen (sollten):
Siehe JDBC-Einführung in Dbs1

JDBC-Architektur
How To (Hello World)
JDBC 2-Erweiterungen
RowSets
JNDI
Connection Pooling

JDBC-Architektur

- JDBC definiert eine Menge von Schnittstellen
- Hersteller implementieren diese für ihr DBMS-Produkt
- Bekannt als 'JDBC driver'
- Es ist einfach, das Produkt zu wechseln, ohne eine Zeile Code anzupassen: Ein einfacher Eintrag in Konfig.-Datei genügt
- Entwicklung kann auf Spielwiesen wie MS Access oder gar Dateien und Tabellenkalkulations-Programmen beginnen. Später wechselt man einfach zu mächtigeren Produkten



How-To…

Hauptklassen:

- DriverManager
- Connection
- Statement
- ResultSet

Man beschaffe sich ein relationales DBMS

- ORACLE, MS SQL Server, MS Access
- PostgreSQL, MySQL
- http://java-source.net/open-source/database-engines

Man beschaffe sich einen passenden Treiber

- http://developers.sun.com/product/jdbc/drivers

1: Load the driver

```
Connection con = null;
try {
  Class.forName("org.gjt.mm.mysql.Driver");
  con = DriverManager.getConnection
        ("jdbc:mysql://hazel/studs","mpc","");
  Statement stmt = con.createStatement();
  ResultSet rs = stmt.executeQuery("SELECT name FROM student");
  while (rs.next())
     System.out.println(rs.getString(1));
} catch (SQLException e) {
  System.out.println("SQLException: " + e.getMessage());
} catch (Exception e) {
  System.err.println("Unable to load driver.");
  e.printStackTrace();
}
```

2: Get a connection

```
Connection con = null;
try {
  Class.forName("org.gjt.mm.mysql.Driver").newInstance();
  con = DriverManager.getConnection
        ("jdbc:mysql://hazel/studs","mpc","");
  Statement stmt = con.createStatement();
  ResultSet rs = stmt.executeQuery("SELECT name FROM student");
  while (rs.next())
     System.out.println(rs.getString(1));
} catch (SQLException e) {
  System.out.println("SQLException: " + e.getMessage());
} catch (Exception e) {
  System.err.println("Unable to load driver.");
  e.printStackTrace();
}
```

3: Execute a query

```
Connection con = null;
try {
  Class.forName("org.gjt.mm.mysql.Driver").newInstance();
  con = DriverManager.getConnection
        ("jdbc:mysql://hazel/studs","mpc","");
  Statement stmt = con.createStatement();
  ResultSet rs = stmt.executeQuery("SELECT name FROM student");
  while (rs.next())
     System.out.println(rs.getString(1));
} catch (SQLException e) {
  System.out.println("SQLException: " + e.getMessage());
} catch (Exception e) {
  System.err.println("Unable to load driver.");
  e.printStackTrace();
}
```

4: Do something with the results

```
Connection con = null;
try {
  Class.forName("org.gjt.mm.mysql.Driver").newInstance();
  con = DriverManager.getConnection
        ("jdbc:mysql://hazel/studs","mpc","");
  Statement stmt = con.createStatement();
  ResultSet rs = stmt.executeQuery("SELECT name FROM student");
  while (rs.next())
     System.out.println(rs.getString(1));
} catch (SQLException e) {
  System.out.println("SQLException: " + e.getMessage());
} catch (Exception e) {
  System.err.println("Unable to load driver.");
  e.printStackTrace();
}
```

Tidy up a precious resource

```
    …
    rs.close();
    stmt.close();
} finally {
    if (con != null)
        con.close();
}
```

JDBC 2+ - Erweiterungen

- Scrollbare und änderbare Resultsets
- Batch Updates
- JNDI und DataSource
- Connection Pooling
- Rowsets
- O/R-Unterstützung
- und mehr…

JDBC 2+ (1 von 4)

- OR Unterstützung
- Scrollbare und änderbare Resultsets
- Batch Updates
- JNDI und DataSource
- Connection Pooling
- Rowsets
- etc.

Rückwärtskompatibel zu JDBC 1

JDBC 2+ (2 von 4)

- JNDI und DataSource
  - JNDI:
    - Abstraction layer for directory services.
    - Setzt LDAP, RMI (-Server) oder Filesystem ein.
    - ‚Bindet' Konfig an einen Namen (property, XML)
  - DataSource

```
Context initialContext = new InitialContext();
DataSource datasource =
  (DataSource)initialContext.lookup("jdbc/ora");
Connection con =
  datasource.getConnection("borg","");
```

JDBC 2+ (3 von 4)

- Connection Pooling
  - ‚Versteckt' vor Programmierer: getConnection(…)
- Rowsets
  - wie JavaBeans, mit Rowset change events
- Distributed Transactions
  - mehrere DBs gleichzeitig bearbeiten mit einer Transaktion
  - Kein eigenes commit() oder rollback() zugelassen
  - Benötigt Transaction Monitor

JDBC 2+ (4 von 4)

- JDBC 4.0
  - Annotations and generic DataSet.
    - Simplify execution of SQL queries (single result sets) and SQL statements that return a row count or nothing
  - XML Datatype (SQL 2003)
  - Autogenerated Keys: java.sql.RowId
  - Service provider mechanism for driver loading
    - if the JDBC driver vendors package their drivers as services, the DriverManager code would implicitly load the driver by searching for it in the classpath.

RowSet

- Ein RowSet ist ein Enterprise Java Bean
  - Kann in einem Grafik-Editor konfiguriert werden
  - Andere Komponenten können über Event Mechanismus Änderungen erfahren
- Ein Row Set kapselt ein ResultSet.
  - Stellt dieselben Methoden zur Verfügung
  - Tabellendaten mit oder ohne permanenter Verbindung zur Datenquelle
- Zwei Arten:
  - Diconnected
  - Connected

JNDI

- JNDI = Java Naming and Directory Interface
- Verbindet Namen mit Objekten
  - Bsp: DNS: 152.96.37.80 www.hsr.ch
- Diverse Naming und Directory Services Implementationen verfügbar:
  - File System
  - NIS (Network Information System) SUN
  - LDAP (Lightweight Directory Access Protocol)
  - RMI Registry (Remote Method Invocation)
  - COS (Common Object Naming Service, CORBA)
- Diese werden mit JNDI "gekapselt"

JNDI Context

- Objekt, welcher eine Sammlung von Bindings mit unterschiedlichen Namen zusammenfasst.

```
/          Initial Context des Unix Filesystem
/usr       Subcontext
```

- Context kann Attribute haben (Directory Service) → DirContext

JNDI und JDBC

- Datenquellen-Objekte werden vom Provider über einen Namen angeboten (gebunden).
- Benutzer bezieht über lookup ein vorkonfiguriertes Objekt. Z.B. `DataSource`
  - Objekt hat bereits Verbindung zu DBMS
  - Benutzer braucht kein Username und Passwort
  - Macht Connection Pooling transparent für Benutzer

JNDI Beispiel

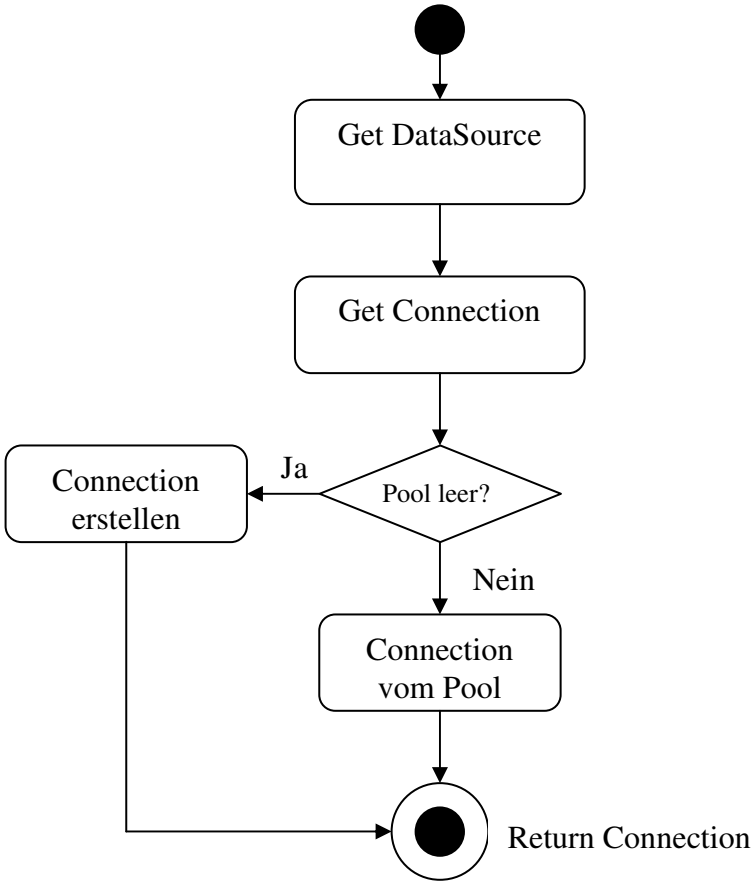
```
// Erzeugen des InitialContext
  Properties prop = new Properties();
  prop.put (Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jndi.fscontext.RefFSContextFactory");
  prop.put (Context.PROVIDER_URL, "file:/tmp ");
  Context ctx = new InitialContext(prop);

// Erzeugen des zu bindenden Objekts
  String myString = new String("Test");

// Binden an den zusammengesetzten Namen test/myObject
  ctx.bind("test/myObject", myString);

// Verwendes des unter dem Namen "file:/tmp/test/myString" bekannten
// Objekts
  String str = (String)ctx.lookup("test/myObject");
```

Connection Pooling 1

- Auf- und Abbau von Verbindungen ist Zeitintensiv
- Physische Verbindungen werden von logischen gekapselt
- Verbindungen werden wiederverwendbar
- Verbindungen werden von einem Pool angefordert und nach Gebrauch zurückgegeben.

Connection Pooling 2

- Anwender erhält über JNDI Name ein DataSource Objekt.
- Bezieht Connection aus DataSource
- Verwendet Connection
- Mit close wird die Connection an den Pool zurückgegeben



ConnectionPoolDataSource

- Abgeleitet von DataSource
- Wird mit DataSource Objekt verbunden:
  - ConnectionPoolDataSource Objekt erzeugen und mit JNDI an einen Namen binde. ("jdbc/pool/cpds")
  - DataSource Objekt erzeugen und mit Hilfe der Methode `setDataSourceName("jdbc/pool/cpds")` mit ConnectionPoolDataSource Objekt verbinden.
  - DataSource Objekt mit JNDI an einen Namen binden. ("jdbc/ds")
  - Name des DataSource Objekt veröffentlichen.
  - Benutzer verwendet DataSource Objekt, erhält aber seine Connections aus einem Connection Pool.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Datenbank-Programmierung: JDBC und O/R-Datentypen

Datenbanksysteme 2

Prof. Stefan Keller

Inhalt

1. Refresher
2. JDBC-Unterstützung der O/R-Datentypen
3. Stored Procedures in Java (Oracle) und Python (PostgreSQL)
4. (Ev.) Data Access Patterns

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

JDBC 2.0-Erweiterungen

JDBC 2.0 Erweiterungen

- OR-Unterstützung
- Scrollbare und änderbare Resultsets
- Batch Updates
- Rowsets
- JNDI und DataSource
- Connection Pooling
- und mehr.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

JDBC-Unterstützung der O/R-Datentypen

Unterstützung der O/R-Datentypen

- JDBC 2 bietet mit den Interfaces
  - Array (VARRAY / Nested Table)
  - Blob / Clob (Bild / Text)
  - Struct (Benutzerdef. Typen)
  - Ref (Referenzen)

  die Möglichkeit auf alle Arten der objektrelationalen Abbildungen lesend und schreibend zuzugreifen.

Using SQL3 Datatypes

SQL3 type	getXXX method	setXXX method	updateXXX method
BLOB	`getBlob`	`setBlob`	updateBlob
CLOB	`getClob`	`setClob`	updateClob
ARRAY	`getArray`	`setArray`	updateArray
Structured type	`getObject`	`setObject`	updateObject
REF (structured type)	`getRef`	`setRef`	updateRef

Zugriff auf VARRAY

- Collection mit `VARRAY`
  - Interface `Array`
  - Methode `getArray()`

  Beispiel:

```
// VARRAY auslesen
Array a = rs.getArray("list");
// Array in ein String[] wandeln
String[] liste = (String[])
  a.getArray();
```

Zugriff auf NESTED TABLE

- Collection mit `NESTED TABLE`
  - gleiches Vorgehen wie bei `VARRAY`
  - Methode `getResultSet()`
    - liefert Inhalt als `ResultSet`

    1. Spalte Index
    2. Spalte Inhalt der Tabelle als `Object`

```
Array a = rs.getArray("Kinder");
ResultSet rsk = a.getResultSet();
while (rsk.next()){
  Struct st = (Struct)rsk.getObject(2);
  Object kindAtt[] = st.getAttributes();
  for (int i = 0; i < kindAtt.length; ++ i){
        System.out.println(kindAtt[i].toString());
  }
}
```

LOBs ("large objects") (1)

- CLOB, BLOB
  - Interface `Clob, Blob`
  - Methoden `getByte()`, `setBinaryStream()`

  Beispiel schreiben:

```
FileInputStream file = new
  FileInputStream("bild.jpg");
PreparedStatement p;
// das Blob mittels BinaryStream des
// FileStreams schreiben
p.setBinaryStream(2, file, MAX);
```

CLOB-/BLOB-Daten lesen (2)

```
// CLOB lesen
Clob c = rs.getClob("Daten");
// String aus CLOB holen
String ss = c.getSubString(1,(int)c.length());
```

```
// BLOB lesen
Blob b = rs.getBlob("Daten");
// BLOB in byte[] wandeln
byte bb[] = b.getBytes(1, (int)b.length());
```

CLOB-/BLOB-Daten schreiben (3)

```
// CLOB schreiben
String filename = "in_text.txt";
FileInputStream file = new
    FileInputStream(filename);

String s = "insert into texte values(?,?)";
PreparedStatement p = con.prepareStatement(s);

p.setString(1, filename);
p.setAsciiStream(2, file, MAX);
p.execute();
```

```
// BLOB schreiben
p.setBinaryStream(2, file, MAX);
```

CLOB Daten anhängen (4)

```
// CLOB Daten anhängen
ResultSet rs = s.executeQuery("SELECT * from texte
WHERE Filename='text2.txt' FOR UPDATE");
String filename;
while (rs.next())
{
  filename = rs.getString(1);
  CLOB c = ((OracleResultSet)rs).getCLOB(2);
  c.putString(c.length()+1,„xyz_string");
}
```

Oracle Extension für den Datentypen CLOB verwenden, da Java Clob keine `putString()` Methode besitzt.

Zugriff auf eigene Typen mit Struct

- Benutzerdefinierter Typ
  - Interface `Struct`
  - Methoden `getAttributes()`

  Beispiel:

```
// Adresse in Struct einlesen
Struct st =
  (Struct)rs.getObject("Adr");
// Alle Attribute auslesen
Object adr[] = st.getAttributes();
```

Zugriff auf eigene Typen mit Klasse (1)

- Benutzerdefinierter Typ mit Klasse (hier: Kind)
  - Interface `SQLData`
  - Klasse registrieren

  Beispiel der Registrierung (mit User ‚info'):

```
java.util.Map map = con.getTypeMap();
conn.setTypeMap(map);
map.put("INFO.KindTyp",
        Class.forName("Kind"));
```

  - `Struct.getSQLTypeName() //gib Name des Typs`

Zugriff auf eigene Typen mit Klasse (2)

```
public class Kind implements SQLData
{
  private String vorname;
  private Date gebDatum;
  private int kinderZulage;

  private String sqlType;

  public Kind()
  {}

  public String getSQLTypeName() throws SQLException
  {
    return sqlType;
  }
```

Zugriff auf eigene Typen mit Klasse (3)

```
public void readSQL(SQLInput stream, String typeName)
    throws SQLException
{
  sqlType = typeName;
  vorname = stream.readString();
  gebDatum = stream.readDate();
  kinderZulage = stream.readInt();
}

public void writeSQL(SQLOutput stream)
    throws SQLException
{
  stream.writeString(vorname);
  stream.writeDate(gebDatum);
  stream.writeInt(kinderZulage);
}
```

Zugriff auf eigene Typen mit Klasse (4)

- Benutzerdefinierter Typ mit Klasse
  - Beispiel, lesen aus einer `NESTED TABLE`:

```
Array a = rs.getArray("Kinder");
// ResultSet auf das Array anlegen
ResultSet rsk = a.getResultSet();

while (rsk.next())
{
  // Kind aus dem ResultSet lesen
  Kind k = (Kind)rsk.getObject(2);
```

Zugriff auf Referenz

- Referenz Typ
  - Interface `Ref`
  - Methode `getStruct()`
  - `oracle.sql.REF` verwenden
  - Beispiel:

```
// Referenz auf Abteilung einlesen
oracle.sql.REF
re=(oracle.sql.REF)rs.getRef("Abt");
// Daten der referenzierten Abteilung laden
Struct st = re.getSTRUCT();
// Attribute der Abteilung lesen
Object abt[] = st.getAttributes();
```

Zusammenfassung (Aufgabe): Vergleich von Datentypen

Vergleich der Typen von OO-Programmiersprachen und einer relationalen Datenbank

DB-Typen (ORACLE)	*Java*	*JDBC*
BLOB		
BFILE		
CHAR		
VARCHAR2		
LONG		
NCHAR		
NVARCHAR		
CLOB		
NCLOB		
DATE		
NUMBER		
OBJECT		
REF		
ROWID		
TABLE		
VARRAY		

Quellen

- Oracle Online-Dokumentation - JDBC Developer's Guide and Reference
  - Overview of JDBC 2.0 Support
  - Overview of Oracle Extensions
  - Working with LOBs and BFILEs
  - Working with Oracle Object Types
  - Working with Oracle Object References
  - Working with Oracle Collections

- Java Tutorial
  - Lesson: New Features in the JDBC 2.0 API
  - http://java.sun.com/developer/Books/JDBCTutorial/

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Stored Procedures in Java (Oracle) und Python (PostgreSQL)

Datenbanksysteme 2

Prof. Stefan Keller

Inhalt

1. Refresher
2. JDBC-Unterstützung der O/R-Datentypen
3. Stored Procedures in Java (Oracle) und Python (PostgreSQL)
4. (Ev.) Data Access Patterns

Stored Procedures

- Business Logik auf einem zentralen Server
- Performance
  - Verbesserte Leistungsfähigkeit im Client-Server Betrieb
  - Operationen finden da statt, wo die Daten sich befinden
- Security
  - Erhöhte Datensicherheit, da alle wichtigen Überprüfungen zentral formuliert und ausgeführt werden
  - Verbesserter Datenschutz: Zugriff auf Tabellen nur via Prozedures
- Nachteil: Sprache ist herstellerspezifisch
  - Oracle: PL/SQL
  - MS SQL Server: Transact SQL (T/SQL)
  - PostgreSQL: pl/pgsql

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

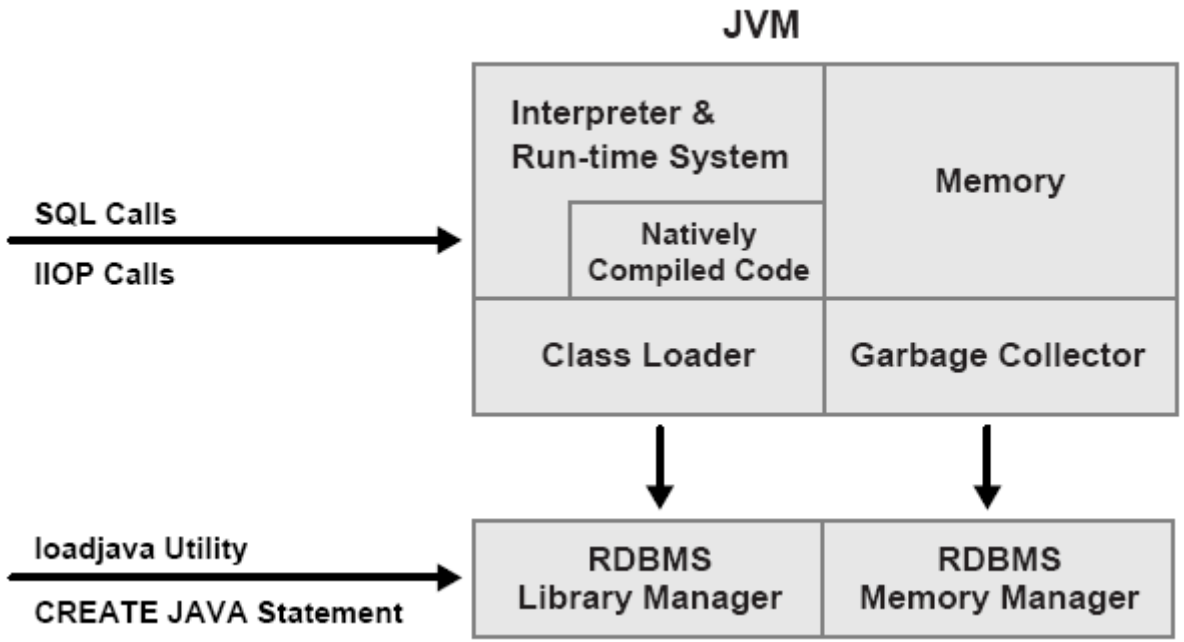
INFORMATIK

Stored Procedures in Java (Oracle)

Stored Procedures in Java auf Oracle

- Oracle Server enthält JVM
  - Procedures,
  - Functions,
  - Member Functions von Object Types und
  - Triggers können in Java geschrieben werden
- Zur Entwicklung kann beliebige IDE verwendet werden
- Alle Sprachmittel von Java 2 vorhanden
  - java.lang, java.io, java.net, java.math, and java.util
  - Ausnahmen:
    - Keine main() Methoden
    - Keine GUI-Komponenten
    - Multithreading

Stored Procedures in Java



Oracle – Java in the Database

Entwicklungsprozess

1. Schreibe eine Prozedur in Java
   - Verwende beliebige Entwicklungsumgebung
2. Prozedur in die Datenbank laden
   - als .java oder .class File
3. Prozedur als Stored Procedure publizieren (im Oracle Data Dictionary)
   - Definition der Call Specification
4. Verwende die Stored Procedure
   - aus PL/SQL (Stored Procedure, Trigger)
   - aus Client-SW (via JDBC, SQLJ)

Schritt 1: Procedure schreiben

```
public class POManager {
   public static void changeQuantity (int newQty, int
   orderNo, int stockNo)
        throws SQLException
   {
      String sql = "UPDATE LineItems SET Quantity = ? " +
        "WHERE PONo = ? AND StockNo = ?";
      try {
        Connection conn = new
   OracleDriver().defaultConnection();
        PreparedStatement pstmt =
   conn.prepareStatement(sql);
        pstmt.setInt(1, newQty);
        …
        pstmt.executeUpdate();
        pstmt.close();
      } catch (SQLException e)
   {System.err.println(e.getMessage());}
   }
}
```

Schritt 1: Procedure schreiben

- Kommunikation mit den Tabellen findet über JDBC statt
  - `Connection conn =`
    `new OracleDriver().defaultConnection();`
  - Connection und Transaktion die bereits besteht wird verwendet
- OUT / INOUT Parameter → Array mit einem Element
  - `p OUT NUMBER → float[] p;  p[0] = 10,9;`
- Namespaces
  - `CLASSPATH → Resolver,`

Schritt 2: Procedure in die Datenbank laden

- CREATE JAVA CLASS in SQL*Plus
  - Source Code muss in Tabelle (BLOB, BFILE) gespeichert sein
- loadjava
  - Command-line Tool

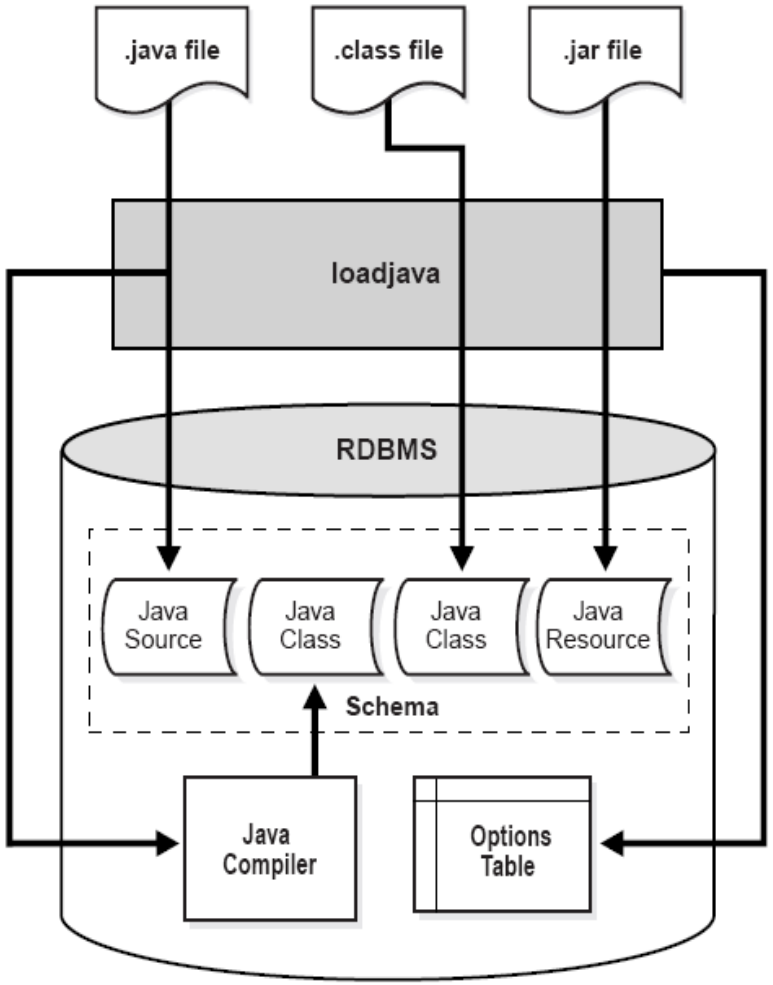
```
loadjava -u scott/tiger@pinfplw11:1521:oraclepe -v -r -t
   POManager.java
initialization complete
loading : POManager
creating : POManager
resolver : resolver ( ("*" scott) ("*" public) ("*" -))
resolving: POManager
```

-v enables verbose mode

-r compiles uploaded Java source files and resolves external references

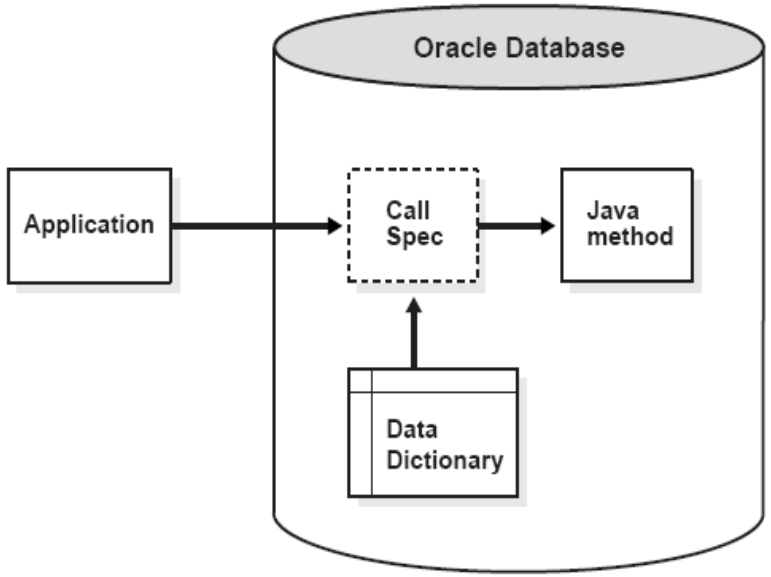
-t connect to the database using the client-side JDBC Thin driver.

Schritt 2: Procedure in die Datenbank laden



Schritt 3: Procedure publizieren

- Registrierung im Data Dictionary
- Nur öffentliche Methoden müssen publiziert werden
- Call Specifications (Call Specs) definieren Mapping zwischen Java und SQL
  - Methoden Namen
  - Parameter Typen
  - Return Typen



Schritt 3: Procedure publizieren

- The methods in the Java class POManager are logically related, so you group their call specs in a PL/SQL package. First, you create the package spec, as follows:

```
CREATE OR REPLACE PACKAGE po_mgr AS
   …
   PROCEDURE change_quantity (new_qty NUMBER,
     order_no NUMBER, stock_no NUMBER);
   PROCEDURE total_orders;
   …
END po_mgr;
```

Schritt 3: Procedure publizieren

- Then, you create the package body by writing call specs for the Java methods:

```
CREATE OR REPLACE PACKAGE BODY
  po_mgr AS
   PROCEDURE change_quantity
     (new_qty NUMBER, order_no
    NUMBER, stock_no NUMBER)
   AS LANGUAGE JAVA
   NAME
```

Schritt 4: Procedure verwenden

- Aufruf geschieht über SQL DML, SQL Call oder aus einem PL/SQL-Block

```
-- Output zum Client leiten
SET SERVEROUTPUT ON
CALL dbms_java.set_output(2000);

-- Aufruf der Prozeduren
BEGIN
   po_mgr.change_quantity (20, 1001, 12);
   po_mgr.change_quantity (30, 1002, 43);
   po_mgr.total_orders;
END;
```

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Stored Procedures in Python (PostgreSQL)

Stored Procedures in Python (PostgreSQL)

- Currently four procedural languages available in standard PostgreSQL distribution: PL/pgSQL, PL/Tcl, PL/Perl, and

PL/Python!

- Based on libpq, the common C API to PostgreSQL
- Python Binaries can be downloaded from www.python.org
- Install: Run in psql
  ```
  CREATE PROCEDURAL LANGUAGE 'plpythonu' HANDLER
  plpython_call_handler;
  ```
- PL/Python is an “untrusted” language (does not offer any way of restricting what users can do in it). Thus the ‚u‘ in „plpythonu“.

Example

- Simple finding if a file exists

```
CREATE OR REPLACE FUNCTION fnfileexists(IN afilename text) RETURNS
   boolean AS
$$
    import os
    return os.path.exists(afilename)
$$
LANGUAGE 'plpythonu' VOLATILE;


-- testing the function --
SELECT fnfileexists(E'C:\\test.htm')


fnfileexists
------------
t
```

Datenbank-Programmierschnittstellen: Data Access Patterns

Datenbanksysteme 2

Prof. Stefan Keller

Inhalt

1. Refresher
2. JDBC-Unterstützung der O/R-Datentypen
3. Stored Procedures in Java (Oracle) und Python (PostgreSQL)
4. Data Access Patterns

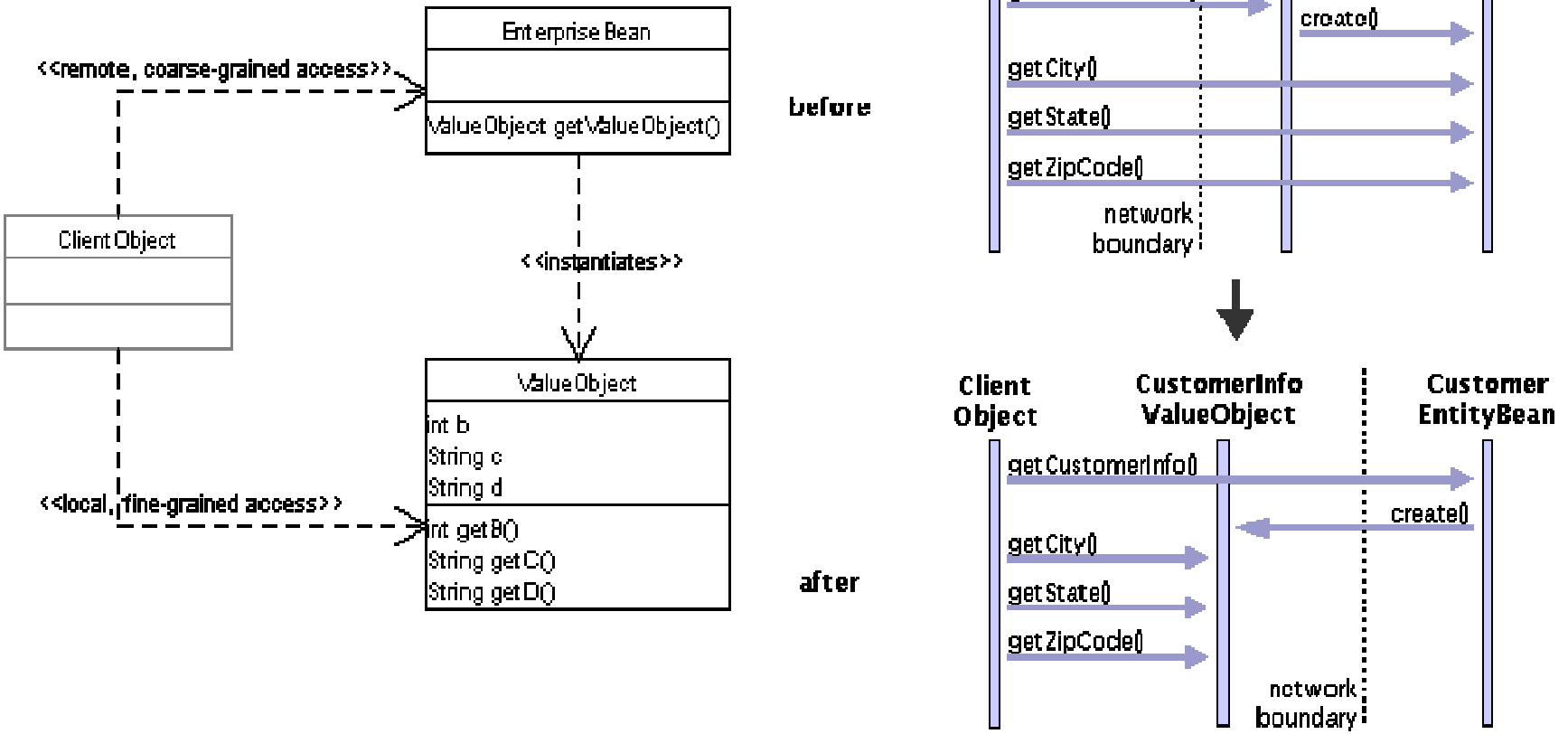
Patterns für Data Access

Welche Patterns kennen Sie für Datenzugriff?

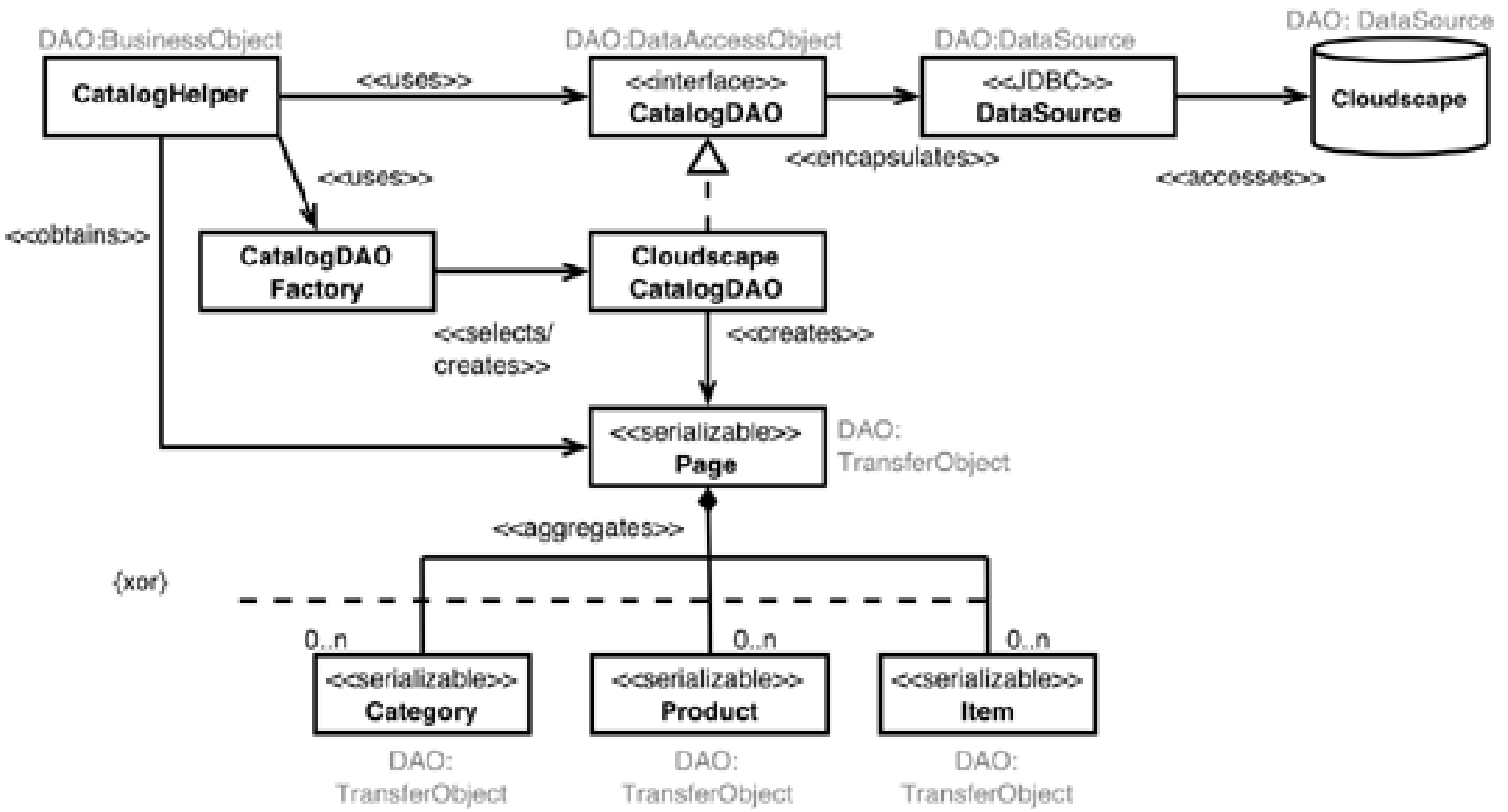
Patterns für Data Access

1. Value Object
2. Data Access Object
3. Page by Page Iterator
4. Active Record Pattern
5. SQL Antipatterns

1. Value Object

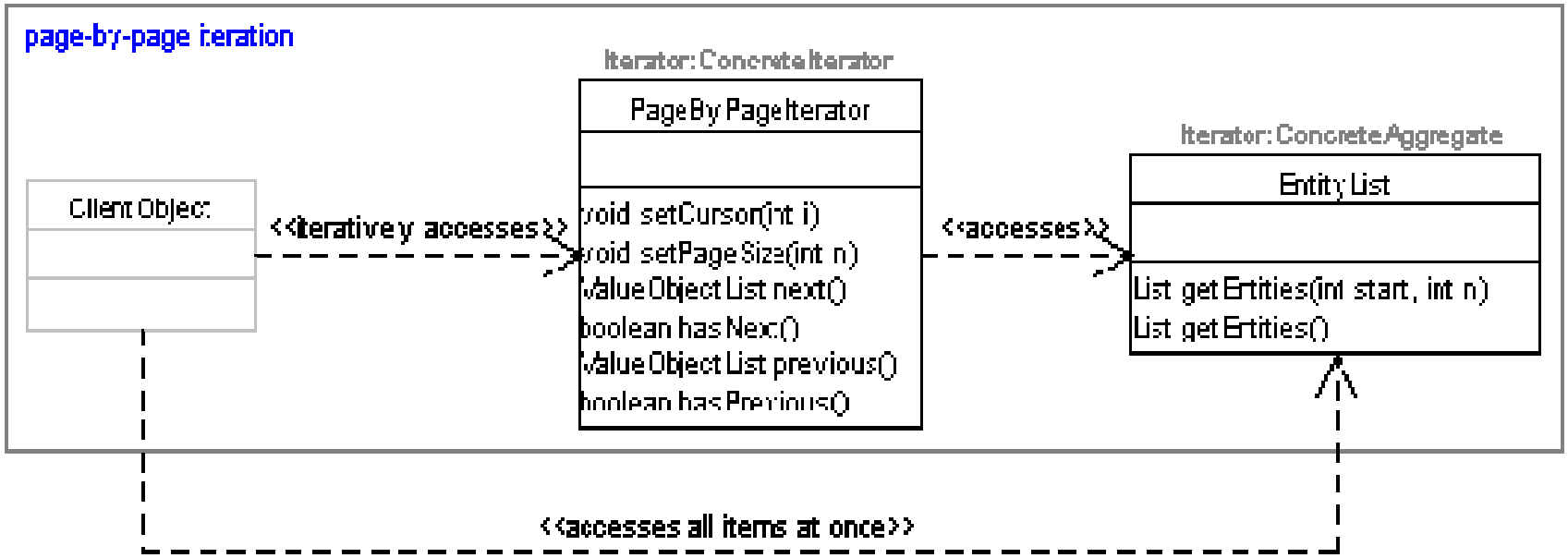


2. Data Access Object (DAO)



*) http://java.sun.com/blueprints/patterns/DAO.html

3. Page by Page Iterator



4. Active Record Pattern

- Ziel: Erzeugung/Änderung von Objekten entkoppeln vom Zeitpunkt, wann es in die DB (als Zeile) gespeichert wird
- In Persistence Frameworks verwendet (O/R-Mapping). Aus Martin Fowlers Buch “Patterns of Enterprise Application Architecture”
- Lösung: Datenzugriff: Tabelle/View wird zur Klasse ‘gewrapped’. Object/Instanz entspricht einer Zeile in der Tabelle.
- Beispiel:

```
part = new Part();
part.name = "Sample part“;
part.price = 123.45;
part.save();
```

entspricht dann:

```
INSERT INTO parts (name, price) VALUES ('Sample
part', 123.45);
```

5. SQL Antipatterns

- Buch von Bill Karwin, Verlag „The Pragmatic Programmers“
- Multi-value Attribute
  - Logical DB Design Antipattern
  - Lösung: Comma Separated List oder ARRAY
  - Antipattern: Nicht Formattieren!
- Entity-Attribute-Values
  - Logical DB Design Antipattern
  - Lösung: Assoziatives Array
  - Antipattern: Keine generische Tabelle
- Using NULL as an ordinary value
  - Query Antipattern
  - Lösung: NULL ist ein separater Wert ausserhalb 0 oder „“

Vergleich von Datentypen

Vergleich der Typen von OO-Programmiersprachen und einer relationalen Datenbank

DB-Typen (ORACLE)	*Java*	*JDBC*
BLOB		
BFILE		
CHAR		
VARCHAR2		
LONG		
NCHAR		
NVARCHAR		
CLOB		
NCLOB		
DATE		
NUMBER		
OBJECT		
REF		
ROWID		
TABLE		
VARRAY		

Vergleich von Datentypen (2)

Vergleich der Typen von OO-Programmiersprachen und einer relationalen Datenbank

DB-Typen (ORACLE)	***Java***	***JDBC***
BLOB	- (byte-Array; getBinaryStream(), getBytes())	java.sql.Blob
BFILE	-	-
CHAR	char, byte, double, float	-
VARCHAR2	char-Array, String, (getAsciiStream())	-
LONG	int, long, short	-
NCHAR	-	-
NVARCHAR	-	-
CLOB	(String)	java.sql.Clob
NCLOB	-	-
DATE	(String; Date /Gregorian Calendar...)	java.sql.Date, .. Time, Timestamp
NUMBER	byte, double, float, int, long, short, Byte, Double, Float, Integer, Long, Short, String	-
OBJECT	(CLASS)	java.sql.Struct
REF	(Referenztyp, String)	java.sql.Ref
ROWID	(String)	-
TABLE	(CLASS)	java.sql.Array
VARRAY	(List)	java.sql.Array

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Objektorientierte Datenbanksysteme

Datenbanksysteme 2 – Teil 1 ODMG

Prof. Stefan Keller

Inhalt

- Einführung in ODBMS
- ODMG-Standard
- ODMG-Language Bindings C++ und Java
- ODBMS-Architekturen

Dank an Prof. H. Huser

Charakterisierung relationaler DB-Systeme I

- Einsatzgebiet Data Management:
  - kommerzielle Datenverarbeitung
  - einfache Datenstrukturen
  - viele Datenelemente
  - relativ einfache Operationen
  - RDBMS optimiert für diese Anwendungen
- Schwachstellen:
  - Struktur:
    - satzorientiert (flache Strukturen)
    - wertorientiert (Identität gegeben durch Werte des Primärschlüssels)
  - Operationen:
    - generische, an der Struktur orientierte Operationen
    - keine anwendungsspezifische Operationen.

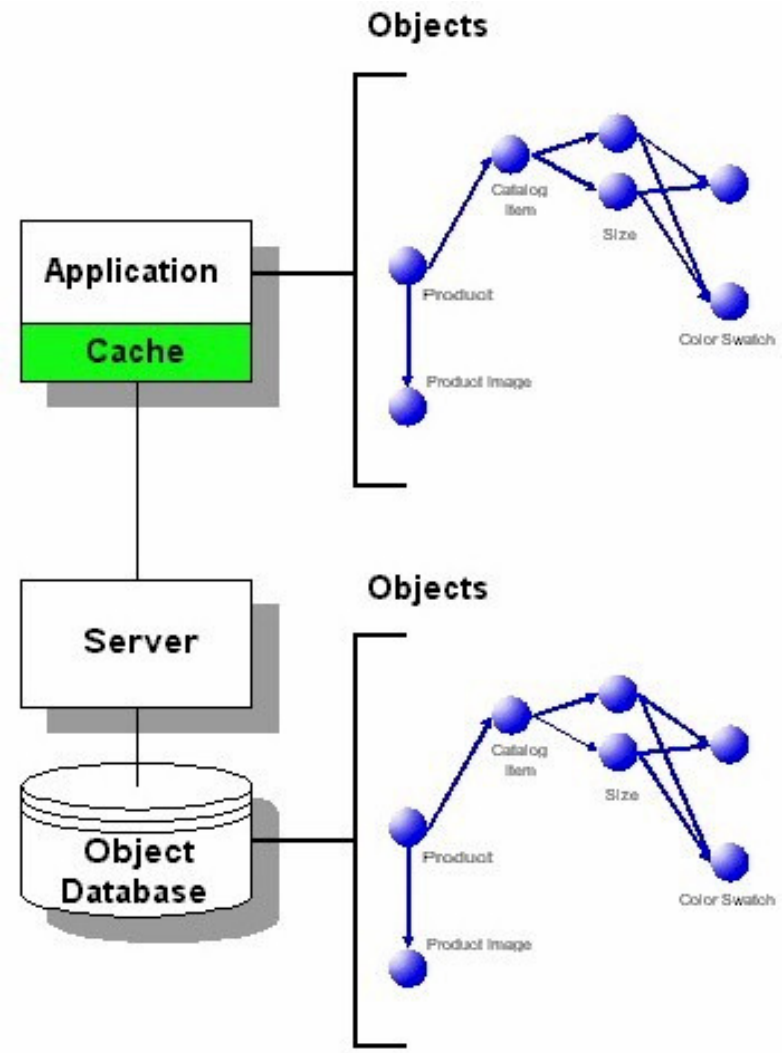
Charakterisierung relationaler DB-Systeme II

- Schwachstellen ff.:
  - Konsistenzbedingungen:
    - Deskriptiv: eingeschränkte Möglichkeiten
    - Prozedural: Trigger, Stored Procedures
  - Typen:
    - nur Basistypen, keine benutzerdefinierbaren strukturierten Typen (gilt nicht für OR-Erweiterungen)
  - Fehlende Einheit von Code und Daten
  - Datenmanipulation
    - SQL mächtig als Abfragesprache, schwach für komplexere Datenverarbeitungsprobleme.
  - Komplizierte Einbettung von Programmiersprachen

Neue Anwendungsgebiete für DBMS

- Entwurfsdatenbanken für CAD oder VLSI-Design
- Software-Engineering Datenbanken als Repository für CASE-Tools, CAD
- Datenbanken für die Speicherung von Multimedia- und Hypertext-Dokumenten (XML)
- Echtzeitdatenbanken für die Leittechnik
- Datenbanken für geografische Daten (Geoinformationssysteme)
- Spezialdatenbanken im wissenschaftlichen oder medizinischen Bereich

ODBMS - komplexe Objekte



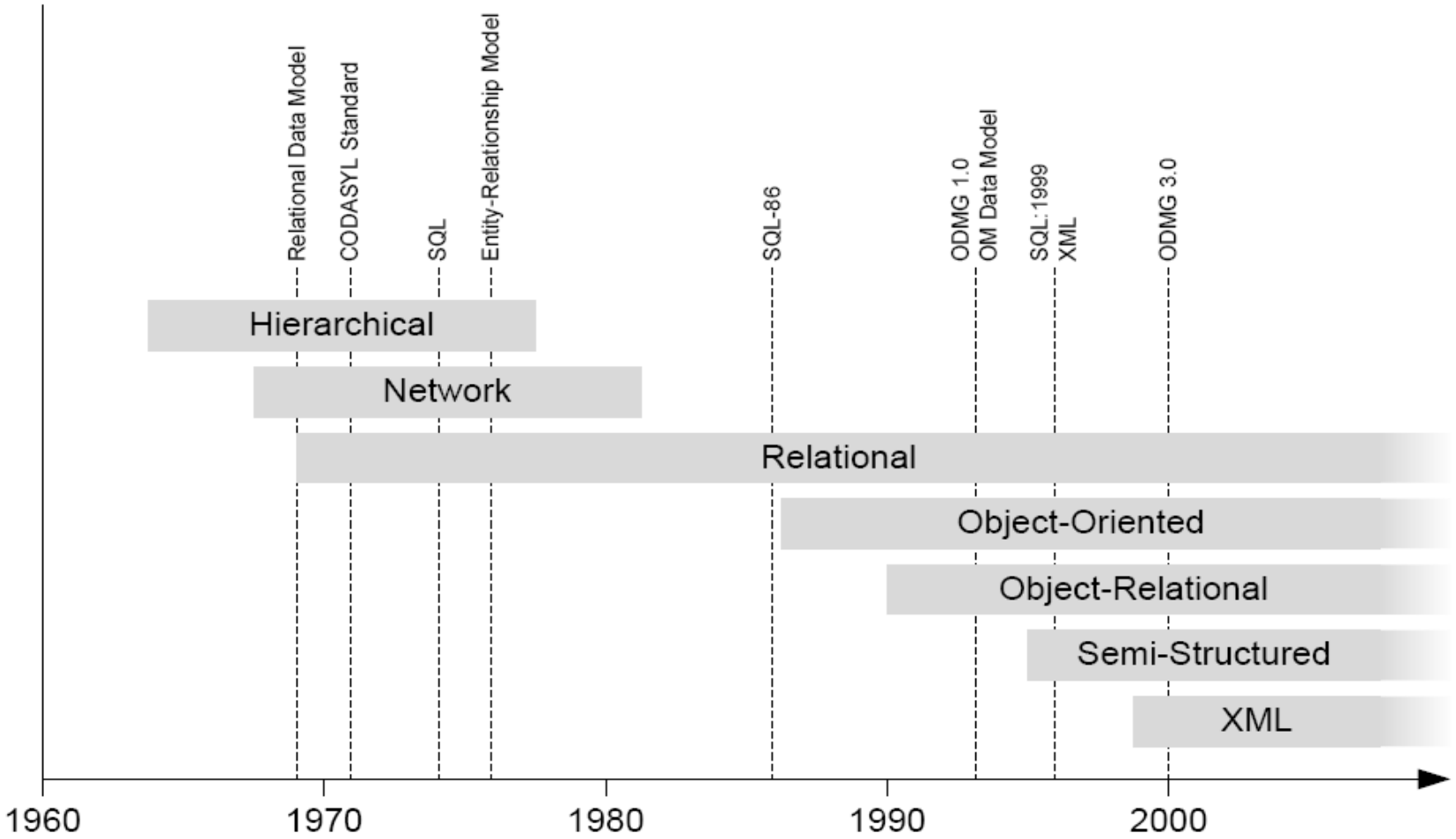
Folgerungen

- Das DBMS muss viel mehr über die Struktur und die Semantik der Daten kennen.
- Mehr Semantik bedeutet:
  - Das DBMS kann eine bessere Konsistenz der Daten garantieren
  - Der Zugriff auf die Daten ist schneller, da das DBMS ein ganzes Objekt als Einheit laden oder speichern kann
  - Das Manipulieren der Daten wird einfacher, da komplexe Operationen auf ganzen Objekten möglich sind

Objektorientierte DBMS (ODBMS)

- Objektrelationale DBMS = RDBMS + OO-Konzepte
- Objektorientiertes Datenbanksystem:
  - Datenbanksystem mit objektorientiertem Datenmodell
  - OO-Programmiersprachen erweitert um DB-Konzepte
  - Revolutionärer Ansatz
- Einsatz:
  - Nischen, wo bisher keine käuflichen DBMS eingesetzt werden konnten
  - Nicht-Standard-DBMS
- Marktanteil (kommerzielle Produkte):
  - ca. 10x kleiner als RDBMS (Bereich 100 Millionen $)

Historische Entwicklung der Datenbanken



Quelle: M. Grossniklaus, ETHZ, CAS 'Object-Oriented Databases', Sept. 2007

Beispiele für ODBMS-Produkte

- Versant (C++- und Java), Versant, www.versant.com:
  - Windows und Unix-Plattformen (C++)
  - Nicht-persistente Version für Java; Persistente Version für Java-ODMG; Persistente Version für C++-ODMG
- ObjectivityDB (C++), Objectivity, www.objectivity.com
- Caché, intersystems, www.intersystems.com
- Db4o (Java, .NET), db4objects.com

ODBMS-Manifesto: Anforderungen

Object-oriented systems

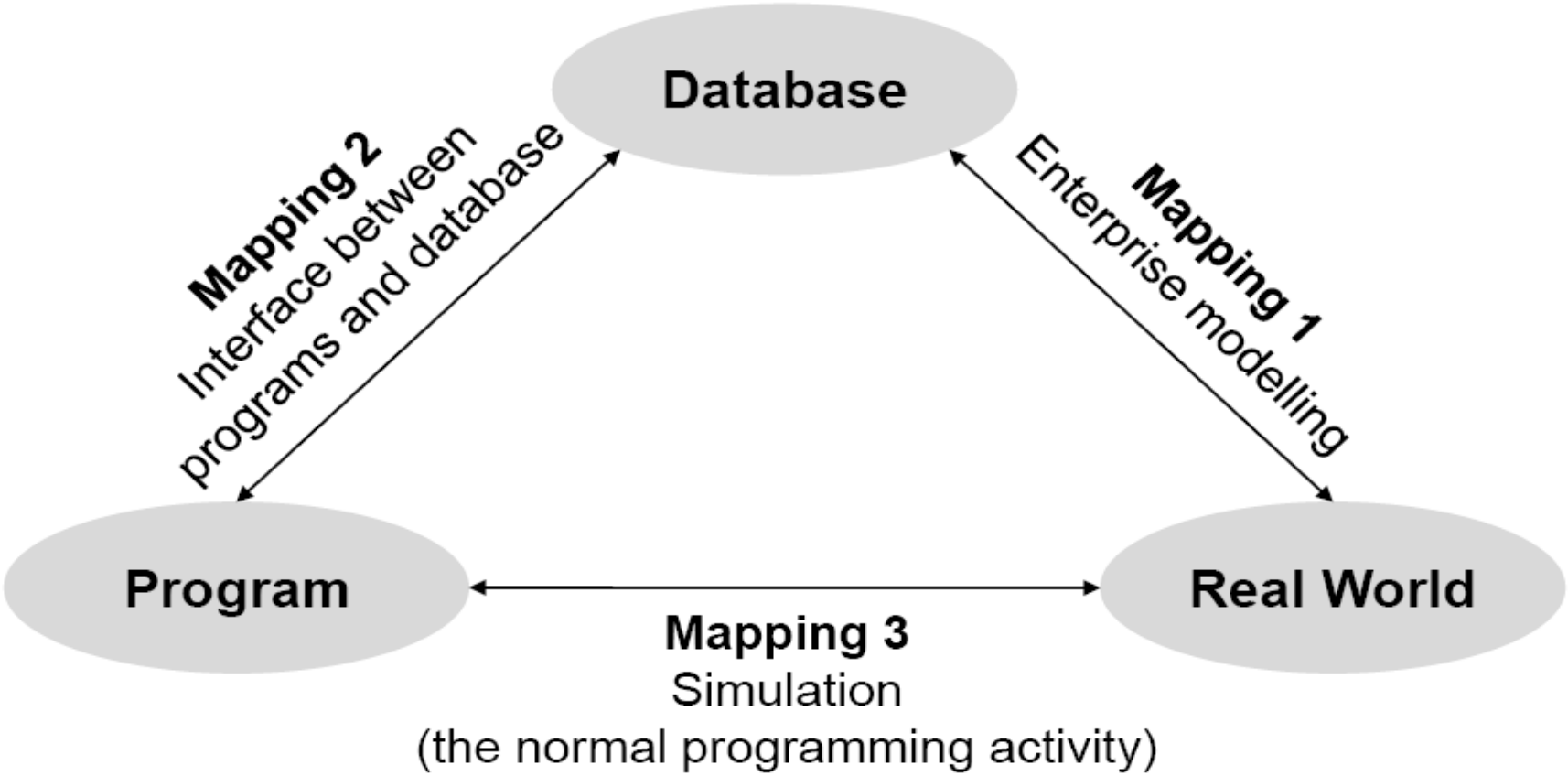
1. Complex objects
2. Object identity
3. Encapsulation
4. Types and Classes
5. Type and Class Hierarchies
6. Overriding, overloading and late binding
7. Computational completeness
8. Extensibility

Database management systems

9. Persistence
10. Efficiency
11. Concurrency
12. Reliability
13. Declarative query language

Atkinson, Dittrich et. al. (1992)

Abbildungen (Mappings) zwischen den Modellen



Quelle: M. Grossniklaus, ETHZ, CAS 'Object-Oriented Databases', Sept. 2007

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

ODMG-Standard

Der ODMG-Standard

- Object Database Management Group (ODMG) (seit 2005 Rechte bei der OMG)
  - Die ODMG war ein firmenorientiertes Gremium
  - Hauptanliegen war Source-Code-Portabilität, nicht Produktvereinheitlichung

- Komponenten des ODMG-Standards V.3.0, 2000
  - Object Model OM
  - Object Definition Language ODL
  - Object Interchange Format OIF
  - Object Query Language OQL
  - Bindings für C++ / Java / Smalltalk

ODMG Objekt-Modell

- Typ
  - Bestimmt Eigenschaften und Verhalten von Objekten
  - Besteht aus Spezifikation (*Attribute*, *Beziehungen* und *Operationen)* sowie aus einer oder mehreren Implementierungen in einer konkreten Programmiersprache
- Typ-Spezifikation
  - Schnittstellendefinition: spezifiziert das abstrakte Verhalten von Objekttypen
  - Klassendefinition: spezifiziert Verhalten und den abstrakten Zustand
  - Literaldefinition: spezifiziert den abstrakten Zustand von Literaltypen (Literale sind Werte)
  - implementiert die Operationen, insbesondere auch solche für die Erzeugung und Vernichtung von Objekten.

ODMG Objekt-Modell

- Klasse
  - implementiert die Operationen (inkl. Konstruktoren, Destruktoren) sowie den konkreten Zustand
- Attribute
  - haben einen Typ, der mit *Basistypen* (und deren Literalen) wie long, float, string, date, time und *Konstruktoren* wie struct<>, set<>, list<>, array<> zusammengesetzt wird.
  - Beziehungen
  - Binäre Relationships zwischen jeweils 2 Typen, Kardinalität 1:n oder n:m, uni- oder bi-direktional
- Vererbung
  - Bei Schnittstellen Mehrfachvererbung

ODMG Objekt-Modell

- Objekte
  - besitzen *Attribute*, *Beziehungen* und *Methoden* sowie einen lebenslangen *Identifier* OID. Sie sind Instanz einer Klasse
- Lebenszyklus eines Objektes
  - entweder *transient* oder *persistent*.
  - persistenzfähige Klasse: Instanzen können transient oder persistent sein
- OID
  - ist eine Art künstlicher Schlüssel für persistente Objekte, wird durch das ODBMS erzeugt. Neben dem OID dürfen Objekte auch einen *Namen* haben. Gewisse Attribute können die Funktion eines externen Primärschlüssels (*Key*) wahrnehmen.
- Extent
  - Die Menge aller Objekte einer Klasse.

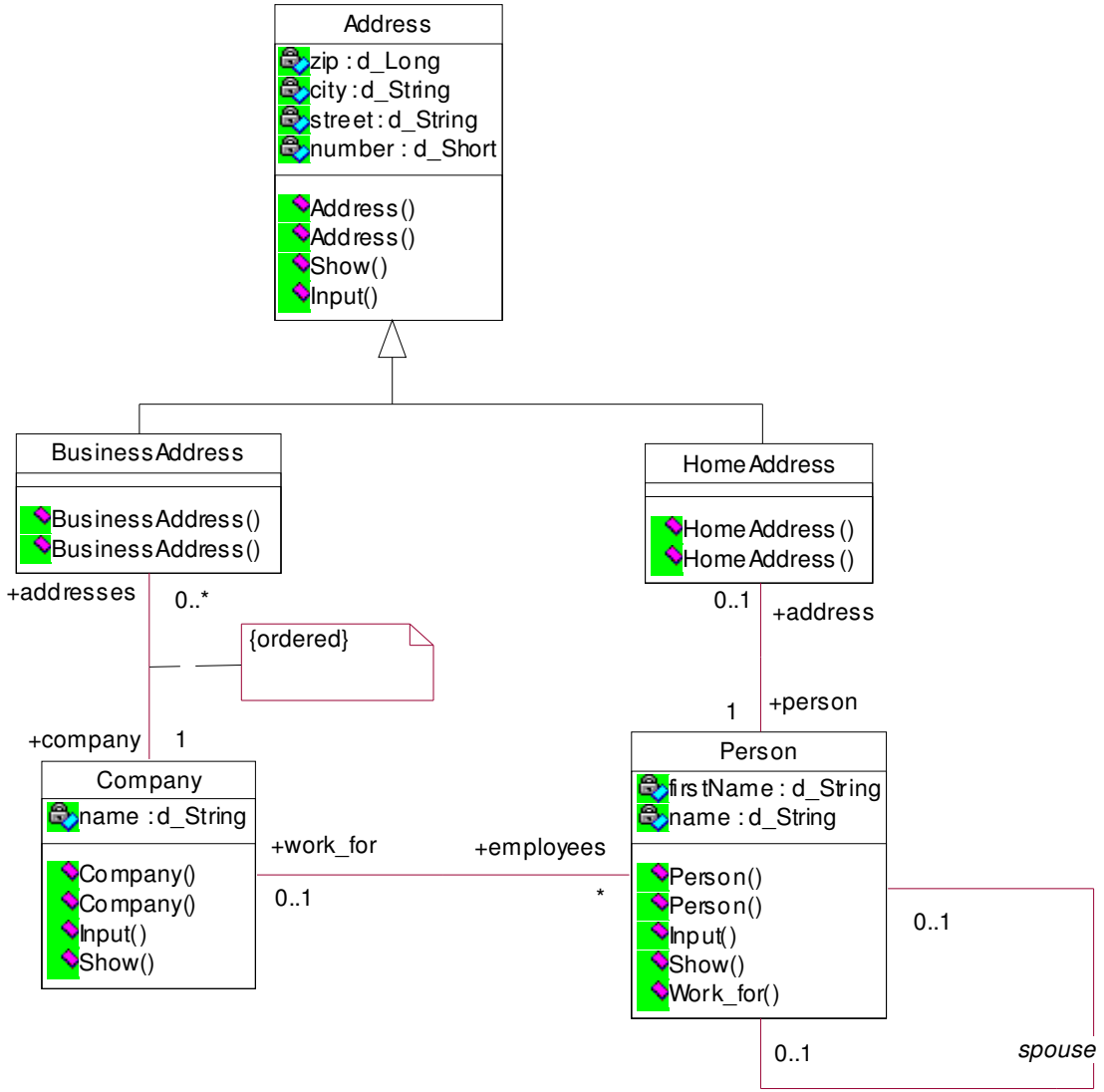
ODMG Objekt-Modell

- Transaktionen
  - Operationen auf den Objekten müssen im Rahmen einer Transaktion ausgeführt werden. Transaktionen genügen der der ACID-Regel.
- Sperren
  - s-locks und x-locks
  - Upgrade locks für Lesen mit späterem Schreiben
  - Sperrprotokoll:
    - Implizit: Sperren werden automatisch beim Lesen bzw. Modifizieren eines Objektes gesetzt
    - Explizit: mit der Operation `lock` bzw `try_lock` (nicht blockierend)
    - Bei Transaktions-Ende werden alle Sperren automatisch freigegeben

Object Definition Language (ODL)

- Programmiersprachen-unabhängige Spezifikation eines Datenmodells mit den Vorgaben des Object Models
- Aus der ODL-Beschreibung wird mit (herstellerspezifischen) Generatoren das Schema der OODB sowie die Definitionen in der Ziel-Programmiersprache generiert.
- ODL ist ein Superset von IDL
- Sprachneutrale ODL hat sich bis heute nicht durchgesetzt.

Beispiel-Schema Firmenadressen



ODL-Beispiel Firmenadressen

```
class Person ( extent people )
{ attribute string name;
  attribute struct Address {
    unsigned short number,
    string street,
  City city} address;

  relationship Person spouse inverse Person::spouse;
  relationship list<Person> children inverse Person::parents;
  relationship list<Person> parents inverse Person::children;

  void birth (in String name);
  boolean marriage (in string person_name)
    raises (no_such_person);
  unsigned short ancestors(out set<Person> all_ancestors)
    raises (no_such_person);
};
```

Object Query Language (OQL)

- Deskriptive Abfragesprache: ‘Was‘ statt ‘Wie‘
  - stand alone: interaktiv
  - eingebettet: C++-OQL, Java-OQL
- Syntax von QQL basiert auf SQL
- Erweiterungen
  - Unterstützung von Collections, Assoziationen, strukturierte Typen
- Einschränkungen
  - keine Update-Operationen, reiner Lesezugriff, keine Views

OQL: Select

- Resultattyp einer Select-Abfrage
  - Objekt, Collections mit Objekten
  - Literale, Collections mit Literalen
  - SQL: Menge von Tupels
- Select Abfragen operieren auf einer Collection
  - Extent: Menge der Objekte eines Typs
  - beliebige Menge von Objekten
  - einzelnes Objekt

OQL - Beispiele

Annahme: Objekt `Person p` existiert

```
p.spouse.address.city.name
```

Adresse der Person p

```
select c.address
  from Persons p, p.children c
```

Adresse der Kinder aller gespeicherten Personen

```
select c.address
  from Persons p, p.children c
  where p.address.street = "Main Street"
  and count(p.children) >= 2
  and c.address.city != p.address.city
```

Adresse der Kinder der Personen, die in der Main Street wohnen und mehr als 2 Kinder haben.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

ODMG Language Bindings

ODMG-Language Bindings

- Definiert die Einbettung folgender Konstrukte in die Programmiersprache:
  - Object Definition Language (ODL)
  - Object Query Language (OQL)
  - Object Manipulation Language (OML) für das Erzeugen und Löschen von Objekten sowie für den Zugriff auf Attribute, Methoden, Typinformationen etc.
- Definiert für C++, Smalltalk und Java

ODMG-Language Bindings: Grundsätze

- Einheitliches Typensystem für die Programmiersprache und für die Datenbank. Instanzen dieser Typen können persistent oder transient sein.
- Die Einbettung definiert eine kleine Menge von Erweiterungen zur Programmiersprache. Diese Erweiterungen sind orthogonal zur bestehenden Funktionalität.
- Ausdrücke der OML können mit Ausdrücken der Programmiersprache beliebig kombiniert werden.

ODMG Java-Binding

- Persistenzkonzepte in Java
  - Objekt-Serialisierung
  - JDBC
  - SQLJ-Embedded SQL
  - ODMG: „Transparente Persistenz“

  - Vorteile / Nachteile / Einsatzgebiete der einzelnen Verfahren

ODMG Java Binding-Prinzipien

- Einheitliches Typsystem, d.h ODBMS unterstützt bzw. erweitert Java-Typen
- keine Java-Spracherweiterungen
- Persistenzfähige Klasse
  - Im Prinzip kann jede Java-Klasse persistenz-fähig gemacht werden (nicht immer sinnvoll!)
    - ODL Preprocessor oder
    - Preprocessor (.java), Postprocessor (.class)
  - Instanzen einer persistenz-fähigen Klasse können persistent oder transient sein
  - Mit ‚transient' markierte Attribute einer persistenz-fähigen Klasse, werden nicht gespeichert

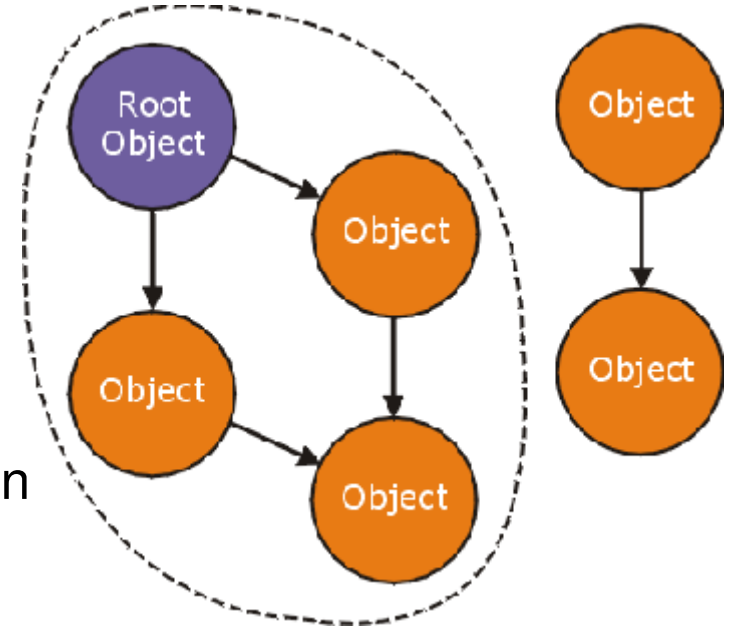
Persistence by Reachability

Root Object: persistentes Objekt, das über einen Namen geladen werden kann (Binding)

Alle Objekte, die von einem persistenten Objekt aus über Referenzen erreicht werden können, sind automatisch persistent.

Das Laden von Objekten aus der DB in den Speicher geschieht automatisch beim Navigieren

Alle DB-Operationen werden innerhalb einer Transaktion ausgeführt. Beim Commit werden alle persistenten Objekte vom Cache in den DB-Speicher zurückgeschrieben.



Java ODL

- Definition des DB-Objekt-Modells mit Java Interfaces, Classes und Collections
- Typen und Methoden
  - Java Basistypen
  - ODMG Collections mit Java Collection Interfaces
  - Java Methoden
- Einschränkungen
  - Bidirektionale Beziehungen, Extents und Keys werden im Java-Binding nicht unterstützt.

Beispiel einer persistenzfähigen Klasse (FastObjects)

```
public class Company {
        private String name;
        private ListOfObject addresses;
        private ListOfObject employees;


        public Company() {
                addresses = new ListOfObject();
                employees = new ListOfObject();
                name = "unknown";
        }

        public Company(String name) {
                        addresses = new ListOfObject();
                        employees = new ListOfObject();
                        this.name = name;
        }
...
}
```

Bem: Diese Klasse muss bei FastObjects gemäss dem "Default-ODMG-Approach" in einem separaten File (Property File) als persistent für den Postprozessor markiert werden!

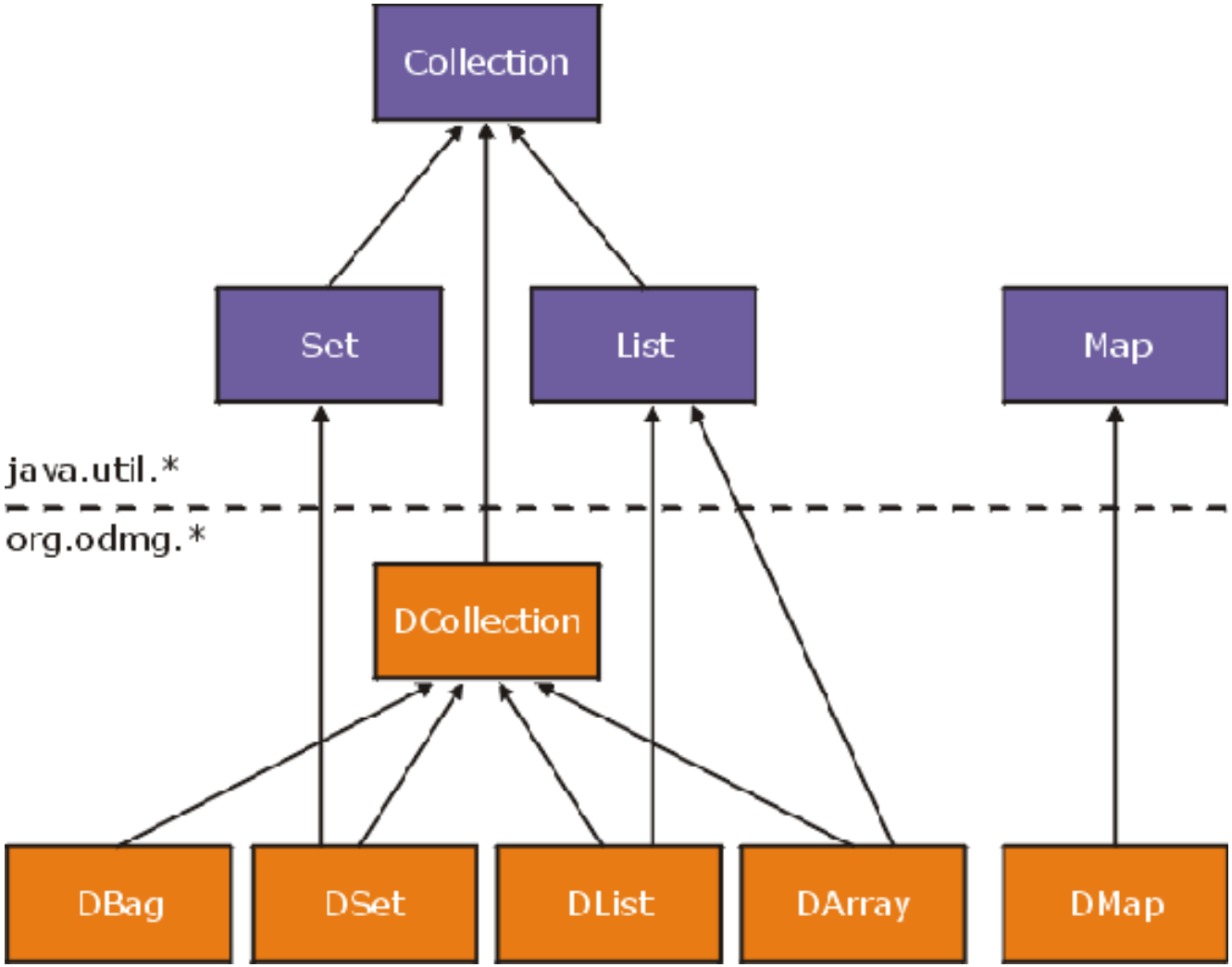
Eigener OGMG-DList als Typ.

ODMG Collection Interfaces

Collection	Order	Duplicates	Typ. Impl
DBag	No	Yes	Hash table / Tree
DSet	No	No	Hash table / Tree
DList	Yes	Yes	List
DArray	Yes	Yes	VArray

- Die ODMG Collections implementieren das Interface DCollection mit Methoden für Element-Abfragen (existsElement, `select()`, `selectElement()`, `query()`)

ODMG Collection Interfaces



Java OML

- Persistenz:
  - Alle Objekte, die über ein persistentes Root-Objekt erreichbar sind, werden persistent
  - alle Objekte, die nicht mehr von einem persistenten Objekt referenziert werden, werden automatisch aus der DB gelöscht ???
- Zugriff auf Objekte
  - Navigation über Referenzen, Einstieg über ein Root-Objekt
  - Iteration über Extent
  - Iteration über Result Set einer Query
- DB-Klassen
  - `class Transaction`
  - `class Database`
  - `class OQLQuery`

Java OML

- Klasse für die Datenbank-Verwaltung

```
public class Database {
  public static Database open( String name, int mode )
                                    throws ODMGException;
  public void close() throws ODMGException;
  public void bind( Object o, String name );
  public void unbind( String name )
                    throws ObjectNameNotFoundException;
  public Object lookup( String name )
                    throws ObjectNameNotFoundException;
  public void deletePersistent(java.lang.Object object);
}
```

Beispiel: Öffnen der DB

```
public void openDB() {
   String dburl= "sod://sa:@localhost/DemoDB";
   try {
        System.out.println("Database: " + dburl);
        // Get implementation
        org.odmg.Implementation Impl =
        de.tneumann.sod2.SOD2.getImplementation();
        // Create a new DB
        org.odmg.Database DB = Impl.newDatabase();
        DB.create(dburl, org.odmg.Database.OPEN_READ_WRITE);
        ...
   }
   catch(DatabaseNotFoundException e)
      { ... }
   catch(DatabaseOpenException e)
      { ... }
   catch(ODMGException e)
      { ... }
   catch(ODMGRuntimeException e)
      { ... }
}
```

Transaktionen

- Operationen auf persistenten Objekten müssen immer im Kontext einer Transaktion ablaufen.
- Der ODMG-Standard schreibt nicht vor, ob das Locking von Objekten implizit oder explizit zu erfolgen hat.
- Beim Ende der Transaktion, werden alle Sperren freigegeben.
- Bei explizitem Locking kann das 2PL-Protokoll verwendet werden (Reihenfolge beachten wegen Deadlocks)

Transaktionen mit Java-OML

- Klasse für die Transaktionsverwaltung:

```
public class Transaction {
  public void begin();
  public void commit();
  public void abort();
  public void checkpoint();
  public void join();
  public void leave();
  public static Transaction current();
  public boolean isOpen();
  public void lock( Object obj, int lockMode)
                    throws LockNotGrantedException;
  public boolean tryLock(Object obj,int lockMode);
}
```

Transaktionen (ff.)

- Bemerkungen
  - `Transaction.lock`: setzt blockierende Schreib- oder Lese- Sperre auf einem Objekt.
  - `Transaction.trylock`: verlangt Sperre, falls das Objekt bereits eine inkompatible Sperre hat, terminiert die Methode (Result false)
  - `Transaction.checkpoint`: Aenderungen werden in der DB gespeichert, Ressourcen werden gehalten, Transaktion läuft weiter.
- Default gem. ODMG ist "implicit locking"

Beispiel Java-OML (Firmenadressen)

```
private void newPerson() {
    Transaction txn= new Transaction(db);
    txn.begin();
    ObjectServices.current().awake(persons);
    try {
      // Attribute der neuen Person bestimmen
      Person p = null;
      boolean found = false;
      Iterator it = persons.iterator();
      while(it.hasNext()) {
         p = (Person)it.next();
         if (p.getName().equals(name)) { //name unique
            found=true; break; }
      }
      if (!found) {
        p = new Person(name); persons.add(p);
        HomeAddress adr = new HomeAddress(...);
        p.setAddress(adr);
      }
      txn.commit();
```

Objekte aus einer vorgängigen Transaktion müssen wieder aktiviert werden!!

Navigationen über Collections mit Java-Standard- Iteratoren und -Enumeratoren

Erzeugen und Modifizieren von Objekten

Aenderung in DB speichern, Locks freigeben

Beispiel Java-OML

```
class FooPure {
   attribute long id;
   attribute bag<long> prims;
};
class BarPure {
   attribute list<FooPure> fooList;
};
// CollectionsPure.java (Fragment)
   ... // Create some data for BarPure / bplist
   BarPure bar= buildBar();
   DB.bind(bar, "bar");
   org.odmg.DList bplist= impl.newDList();
   list.add(new Integer(1));
   list.add(new Integer(1));
   DB.bind(bplist, "list");
   ... Trans.commit();
```

Java OQL

- Klasse für das Durchführen von OQL-Queries:

```
public class OQLQuery {
  public OQLQuery();
  public OQLQuery( String query )
                              throws InvalidQueryException;
  public void create( String query )
                              throws InvalidQueryException;
  public void bind( Object param )
                throws QueryParameterCountInvalidException;
  public void execute() throws QueryException
}
```

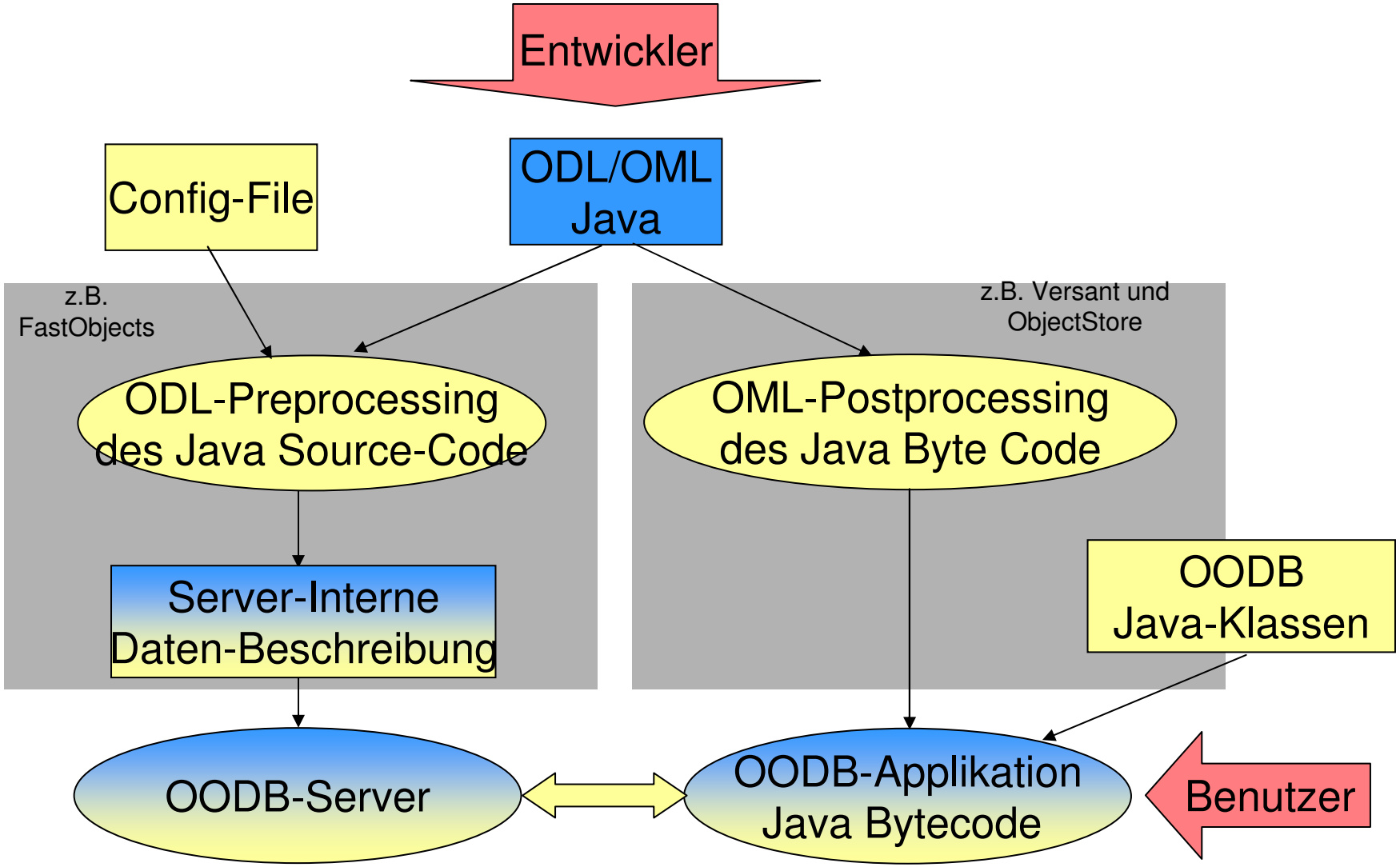
Beispiel Java-OQL

```
// Query data
 String queryStr= "select * from Angestellte a";
 System.out.println( "Query: '" + queryStr + "'" );

 OQLQuery query= impl.newOQLQuery();
 query.create( queryStr );

 DBag extent= (DBag)query.execute(db);
 System.out.println( "- size(): " + extent.size() );
 for (Iterator iter= extent.iterator();
        iter.hasNext(); ) {
     Person p= (Person) iter.next();
     System.out.print( p.Name() + " " );
 }
```

C++-Sprachanbindung

- C++-ODL: Object Definition Language (ODL)
  - Definition des Schemas
- C++-OQL: Object Query Language (OQL)
  - eingebette Abfragen (embedded SQL)
- C++-OML: Object Manipulation Language (OML)
  - Erzeugen / Löschen von Objekten
  - Navigation über Beziehungen
  - Zugriff auf Attribute, Methoden, Typinformationen etc.

Applikationsgenerierung Java



Sprach-/Datenbank-Anbindung und Applikationen

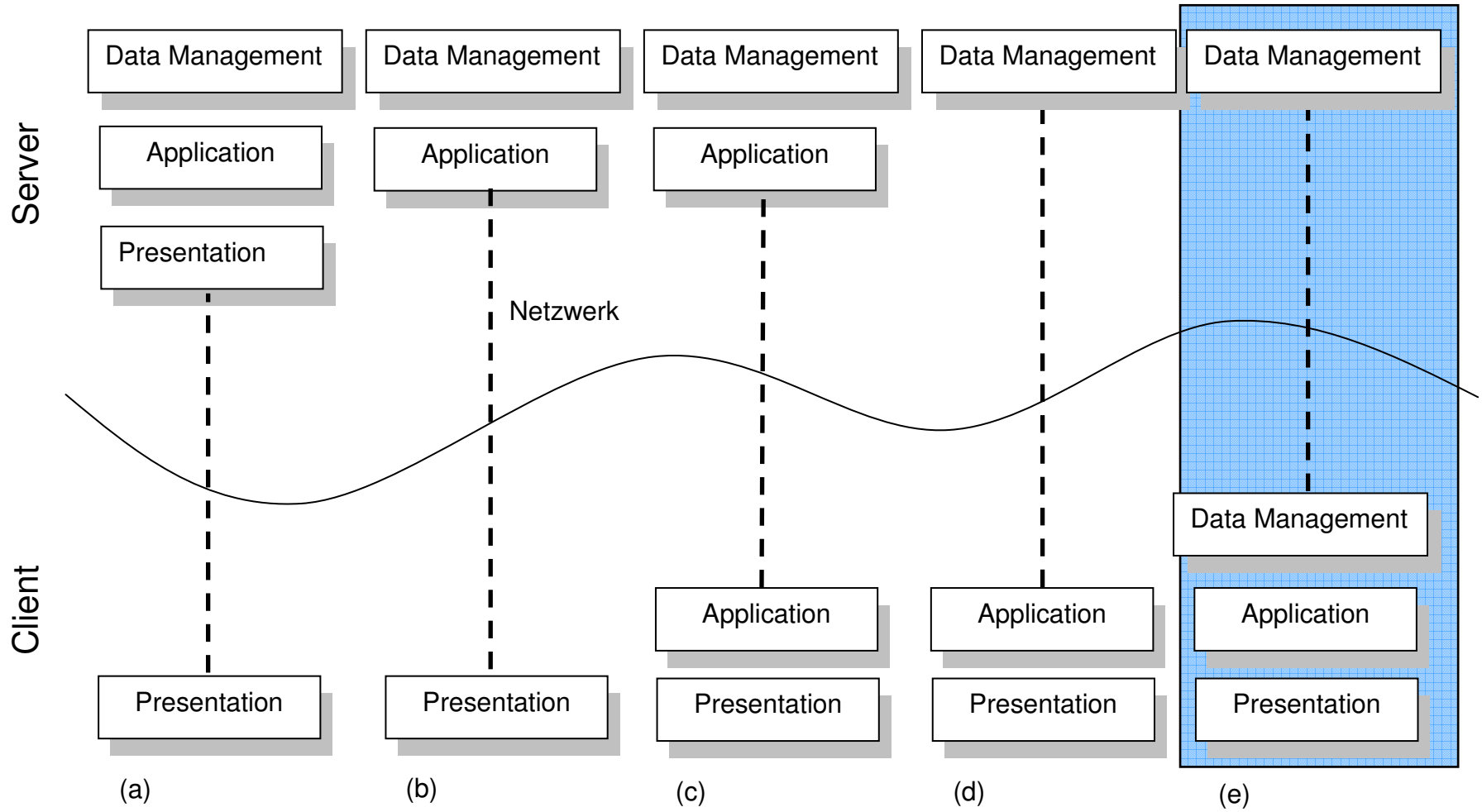
0. ev. Entwerfe das konzeptionelle Domain Model (UML) und generiere 1 und 2
1. (ev.) Spezifiziere das Logische Schema mit ODL
2. (ev.) Generiere den Source Code (Stubs) mit Preprocessor
3. Implementiere die Applikation
4. Compiliere den Code (mit DB-Libraries)
5. (ev.) Postprocess des Codes (Java: byte code engineering)

- Man beachte die Varianten
  - Man kann die ersten zwei Schritte auslassen (bei Postproc.)
  - Vorteile der Varianten mit Schritt 0 und Schritt 1?
  - Wie ist das bei Model- oder Data Driven Approaches?

ODBMS-Architekturen

- 2-tier C/S Architekturen
- Fat Client
  - Applikationscode
  - DB-Code
    - Implementation der persistenten Klassen
    - Queries !
- DB-Server
  - Datenspeicherung
  - Locking

ODBMS-Architekturen: Fat Client

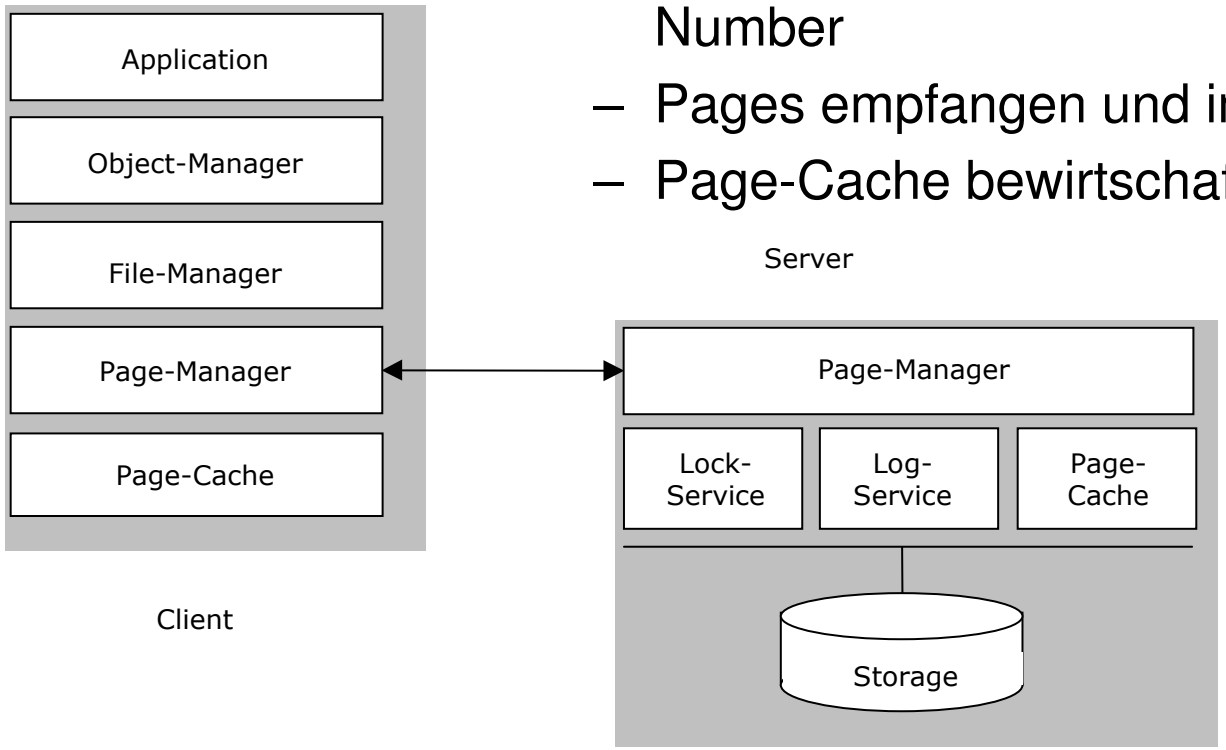


ODBMS Server-Architekturen

- zwei grundsätzlich verschiedenen Architekturen:
  - *Page-Server* "verstehen" nichts von Objekten. Sie liefern lediglich I/O Pages (2^n KB) an die Clients aus und sind zusätzlich für die Atomarität, Isolation und Dauerhaftigkeit von Transaktionen, basierend auf I/O-Pages, zuständig.
  - *Object-Server* liefern aufgrund einer Client-Anfrage einzelne Objekte aus. Daneben sind sie für die Einhaltung der ACID-Transaktionseigenschaften und allenfalls für die Durchführung von OQL-Abfragen und Objektmethoden verantwortlich.

Page-Server (1/2)

- Verbindungs- und Transaktionskontrolle
- Sperren setzen und freigeben auf angeforderte Pages
- Ausliefern von Pages aufgrund einer Page-Number
- Pages empfangen und im Logfile protokollieren
- Page-Cache bewirtschaften



Produkte

O_2

Objectivity/DB

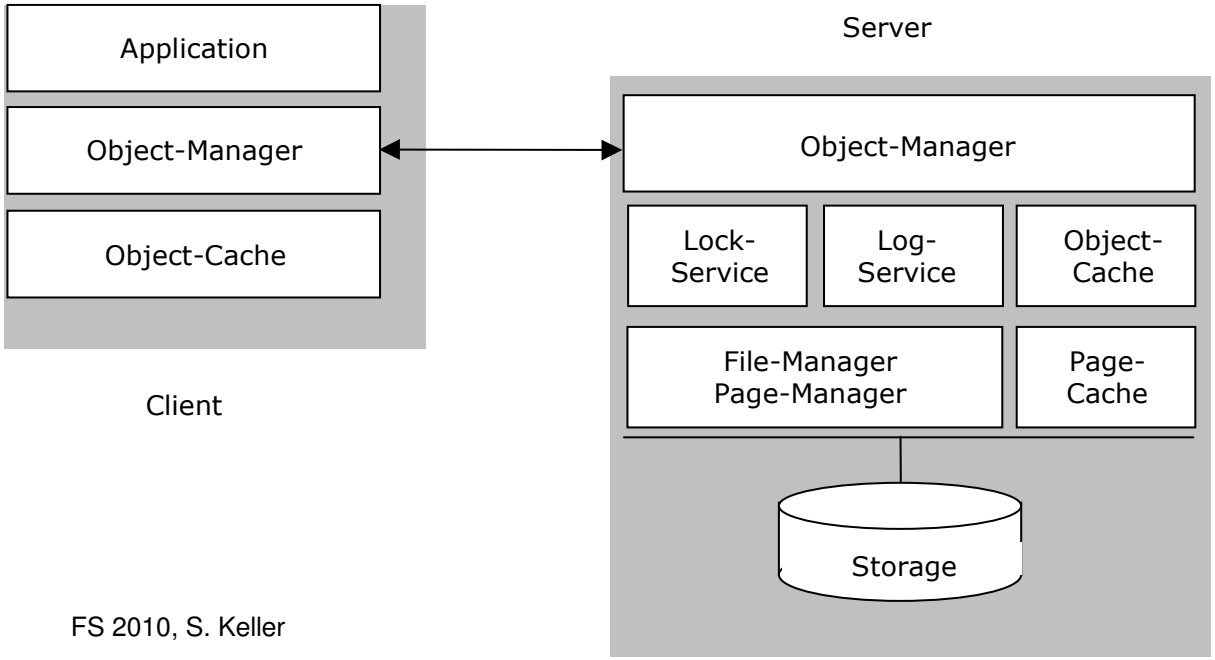
ObjectStore

Page-Server (2/2)

- Eigenschaften
  - Prozessorabhängiges Page-Format für Daten und Code
  - effizienter Transfer von vielen kleinen Objekten
  - Page Level Locking
  - Fat Clients, I/O bounded Server
  - Es werden mehr Objekte als notwendig transferiert
  - Ausführung von OQL und Methoden im Client
- Einsatz für
  - grosse Datenmengen
  - Kleiner Concurrent Use gemeinsamer Daten
  - Applikationen mit rechenintensiven Aufgaben

Object-Server (1/2)

- Verbindungs- und Transaktionskontrolle
- Sperren auf Objekten verwalten
- Übersetzen von OID in physische Adressen
- Objekte empfangen und im Logfile protokollieren
- Page- und Object-Cache bewirtschaften
- OQL Abfragen und Objektmethoden durchführen



Produkte

Versant
POET
GemStone
Matisse
ITASCA

Object-Server (2/2)

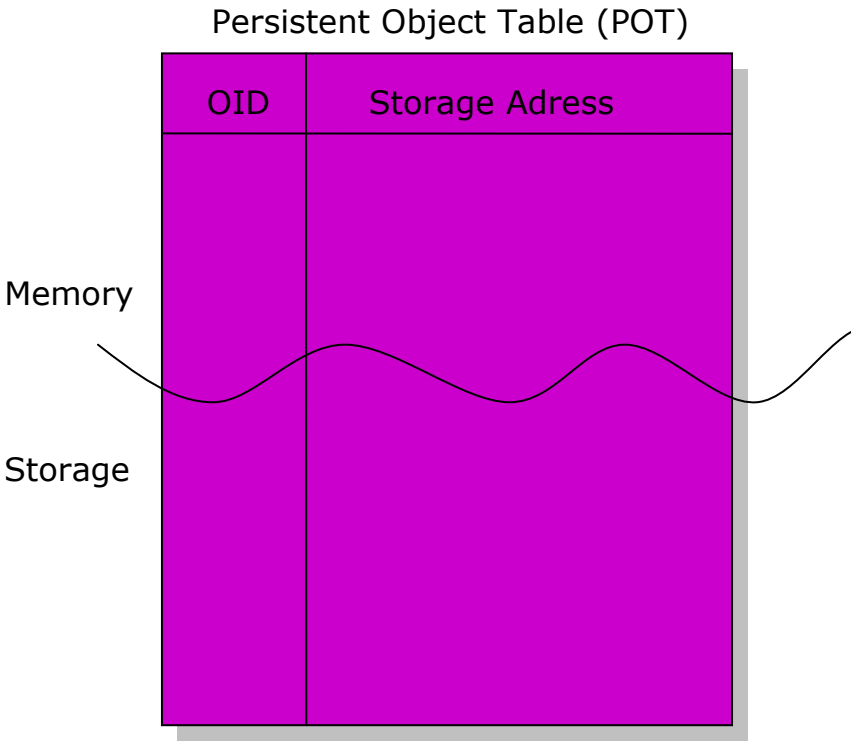
- Eigenschaften
  - Abstraktes Datenformat
  - häufiges Kopieren zwischen Speichermedium und Applikation
  - Object Level Locking
  - Thin Clients, I/O- und CPU-bounded Server
  - inkonsistente Daten zwischen Applikationscache und DB
  - Ausführung von Abfragen beim Server
- Einsatz für
  - kleine und mittlere Datenmengen
  - Hoher Concurrent Use gemeinsamer Daten

Object Identifiers OID

- Jedes Objekt besitzt eine eindeutige Identifikation, den *Object Identifier* (OID).
- Der OID muss systemweit eindeutig, unveränderlich, orts- und zustandsunabhängig sein (Logischer Objekt Identifier LOID).
- Einmal vergebene OID's dürfen auch nach dem Löschen des Objektes nie mehr wiederverwendet werden dürfen, damit eine bereits beim Client abgelegte Objektreferenz nicht plötzlich auf ein neues Objekt zeigt.

Logische OID

- Kein Bezug zu Speicherort
- Serverseitige Übersetzungstabelle POT
  - Memory-residente Hashtabelle
  - Recovery
  - Concurrency Control
  - Grösse ~ Anzahl Objekte in der DB
- ODMG
  - Logische OID mit Eindeutigkeit innerhalb des Storage-Domain (Datenbank)
  - Keine Angabe zu Format und Wertebereich (i.a. 8 Byte)



Physische OID

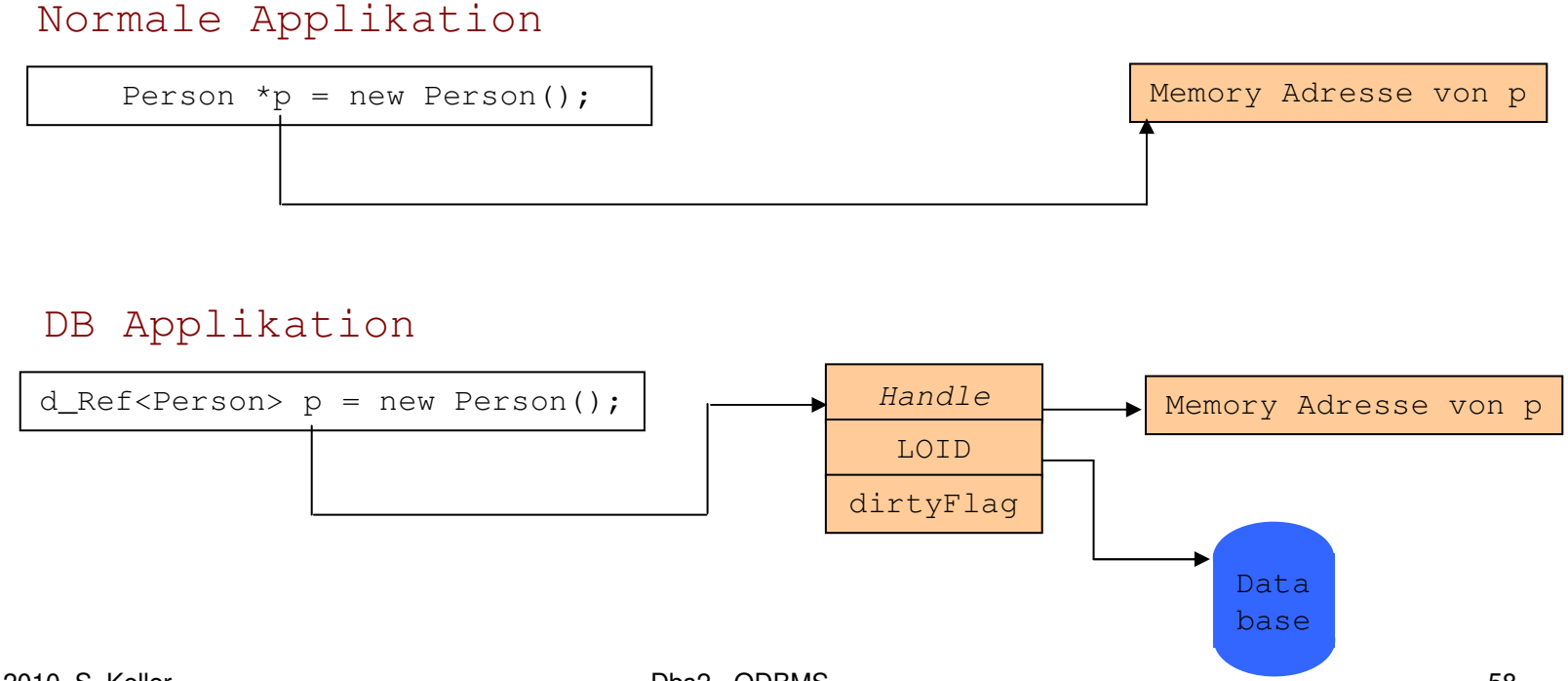
- OID = Physische Speicheradresse, z.B. VolumeID + PageID + PageOffset

- Vorteile
  - Keine serverseitige Übersetzungstabelle notwendig
  - schnelle Auslieferung von Objekten
  - Gut geeignet für Page-Server Architektur

- Nachteile
  - Wiederverwendung nicht ausgeschlossen, z.B. nach Reorganisations-Routinen der Datenbank.
  - Verschiebung eines Objektes innerhalb der Datenbank bedingt erhöhten Verwaltungsaufwand. Beispiel O_2: Am alten Ort wird die Adresse des neuen hinterlegt.

Dereferenzierung von OID

- Ein objektorientiertes Applikationsprogramm arbeitet nicht direkt mit OIDs, sondern mit Pointern (Referenzen).
- Damit die Derefenzierung eines Pointers auf ein Objekt zeigt, müssen entweder die Derefenzierungsoperatoren überschrieben oder der Binärcode (Bytecode) modifiziert werden.



Dereferenzierung von OID

- Die Handles sind in der Applikation in einer Tabelle organisiert, der Resident Object Table ROT.
- Beim Laden eines benannten Objektes mit den ODMG Lookup-Funktionen wird mindestens ein Handle für das Objekt erzeugt.
- Sobald die Applikation Zugriff auf Attribute verlangt, wird das effektive Objekt von der Datenbank geladen.
- Enthält ein zu ladendes Objekt Referenzen auf weitere Objekte, werden diese in Pointer auf Handles übersetzt und letztere in der ROT eingetragen.
- Zum Commit- oder Abort-Zeitpunkt werden i.a. alle Objekte aus dem Applikations-Cache gelöscht, nicht jedoch die Handles. Auf Verlangen kann auch die ROT geleert werden.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Zusammenfassung und Ausblick

OQL-Beispiel Courses

```
// ODL:
class Person {
  attribute string firstname;
  attribute string lastname;
};

class Student extends Person {
  attribute short begin_year;
  relationship set<Course *> courses inverse students;
};

class Course {
  attribute string title;
  attribute string description;
  relationship set<Student *> students inverse courses;
  relationship Teacher *teacher inverse courses;
};

// OQL: Courses
select c.title from Course c where c.students[?].lastname = "Mulder";

select c.title from Course c where c.students[?] =
   (select one s from Student s where s.lastname = "Mulder" and
                                      s.firstname = "Suzan");
```

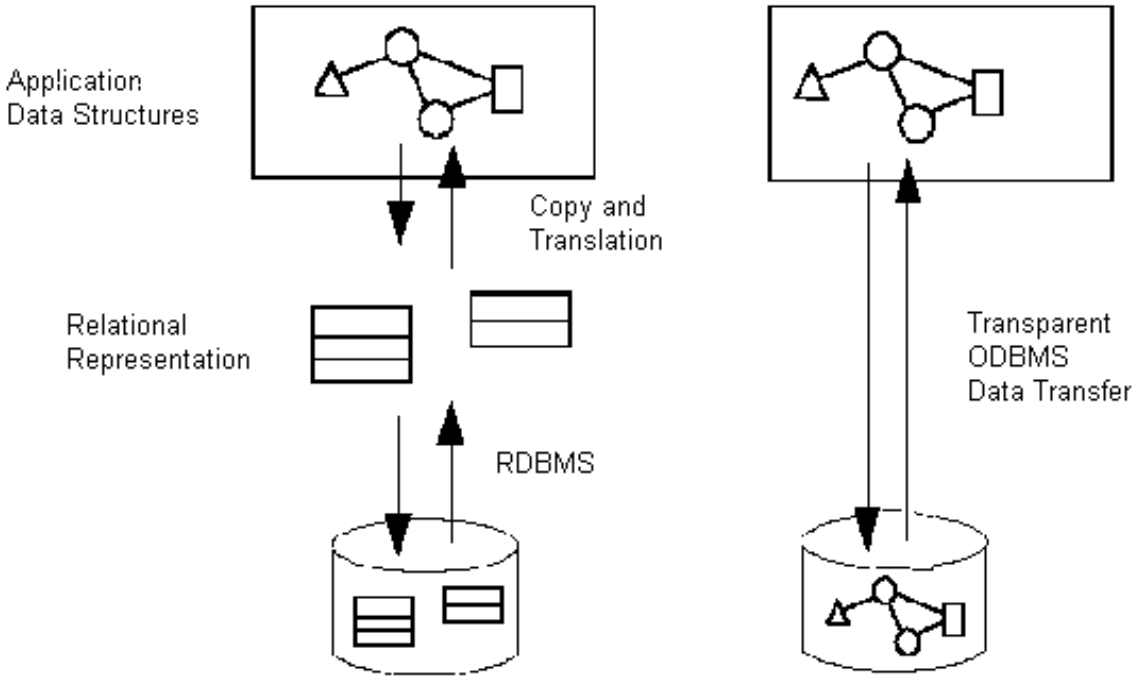
Warum nicht z.B. POJOS serialisieren?

- Nicht vollständig
  - Klassendefinition wird nicht mit den Objekten gespeichert
  - Probleme mit Schema-Evolution und Versionerung
- Nicht orthogonal
  - Serialisierbare Klassen müssen Interface implementieren
- Nicht persistent
  - Object identity geht verloren
- Nicht skalierbar
  - Objekt-Graphen als Ganzes (de-)serialisiert
- Keine Transaktionen
- Weder wiederherstellbar noch ‘concurrent’

Gründe für OODB

- Es wird nur ein einziges Datenmodell benötigt, Kein „Impedance Missmatch“ (mehr)
  - Objektbeziehungen (spaces) und nicht Mengenorientiert
  - Vererbung und Kardinalitäten, Uniqueness und Identity
  - Datentypen
- Potential für Wiederverwendung: Vererbung von Views
- Technologische Gründe:
  - Neue Konzepte, z.B. Objektversionierung, Datentypen Sets, Lists, Maps
  - Potential in Mobile, Embedded und Personal DBMS
- Organisatorische Gründe
  - Erlernbarkeit
  - einfachere Modell-Abbildungen

Vergleich RDBMS-ODBMS: DB-Architekturen



RDBMS

ODBMS

Das Fahrrad-Problem!

Gründe gegen OODB

- Technische Gründe:
  - OQL bei vielen Produkten auf einfacherem Niveau im Vergleich zu SQL
  - Mangel an kommerziellen Datentypen und entsprechenden Abfragemöglichkeiten, z.B. Datum und Zeit
  - Mangelnde Internationalisierung, z.B. Sortierordnung
  - Teilweise wenig ausgereift und fehlende Funktionen
  - wenig Interoperabilität
  - Leistung bei grossen Datenmengen?
- "Umfeld"-Gründe:
  - Kleinfirmen-Problematik
  - Fehlende Drittprodukte (GUI- und Application-Builder)
  - Einführungs- und Einarbeitungsaufwand

Weitere Anforderungen an ODBMS

- Datenbankadministrations-Utilities
  - Datenorganisation
  - Monitoring- und DBA-Tools
- Browser für Definitionen und abgeleiteten Daten
- Schema- und Objekt-Evolution
- Constraints
- Tuning (Locking, Clustering, Indices, Caches)
- Autorisierung
- Mehrere Sprachen-Anbindungen

Ausblick: "ODMG 4.0"

- ODMG-Gruppe wurde 2001 aufgelöst
- OMG erwarb Rechte an ODMG 3.0 2003
- OMG Object Database Technology Working Group (ODBTWG) 2005 gegründet aufgrund wieder-erwachtem Interesse
- Noch keine spruchreife Resultate zu erwarten

Ausblick: Markt und Bedeutung ODBMS

- Marktbedeutung ODBMS
  - Umsatz
    - 1997: 200 Mio US$, 2004: 14.9 Mia US$.
  - Anteil: Oracle 1998 40-30%. Aufkommend: FOSS-DB MySQL, PostgreSQL and Berkeley DB, db4objects
- Produktreife ODBMS
  - Recovery, Datenreplikation und Performance ist vergleichbar mit relationalen Datenbanken.
  - Serverbasierte Funktionen (Queries) im Fluss
  - Integrationsfähigkeit mit RDB's, Webservices, Entwicklungsumgebungen besonders wichtig.
- Akzeptanz
  - im Nischenbereich Mobile, Embedded und Personal DBMS

Quellen: Literatur und Weblinks

- Literatur:
  - Andreas Geppert: Objektrelationale und objektorientierte Datenbankkonzepte und -systeme, dpunkt.verlag, 2002, ISBN 3-89864-124-4
  - "The Object Database Standard: ODMG 3.0"; R.G.G. Cattel; Morgan Kaufmann, 2000.

- Weblinks:
  - www.odbms.org - Portal on Object Databases
  - www.odmg.org - ODMG Publikationen
  - www.odbmsfacts.com - Facts und Marktanalysen

ODL-Beispiel AngProj

```
class Angestellter (extent Angestellte)
{
  attribute long   PersNr;          // INTEGER
  attribute string Name;            // STRING(20)
  attribute long   Tel;
  attribute float  Salaer;          // DECIMAL (7,2);
  attribute string Wohnort;
  attribute string Eintrittsdatum; // DATE;
  attribute float  Bonus;
  relationship Abteilung abteilung
    inverse Abteilung::angestellte;
};

class Abteilung (extent Abteilungen)
{
  attribute long AbtNr;
  attribute string Name;
  relationship list<Angestellter> angestellte
    inverse Angestellter::abteilung;
};
```

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Objektorientierte Datenbanksysteme

Datenbanksysteme 2 – Teil 2 db4o

Prof. Stefan Keller

Inhalt

- Grundlegendes
- db4o
  - Objekte verwalten, speichern und wieder abrufen
  - Drei Anfragesprachen
    - Query by Example
    - Native Queries
    - Simple Object DB Access (SODA => interne Basis)
  - Einfache und komplexe Objekte
  - Aktivierung, Transaktionen
  - Callbacks
  - Konfiguration und Tuning
  - Replikation und Schema-Evolution

Grundlegendes

- Orthogonal persistence

- Persistent strategies
  - Persistence by reachability
  - Persistence by instantiation
  - Persistence by inheritance

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

db4o - open source object database

Fa. db4objects (USA)

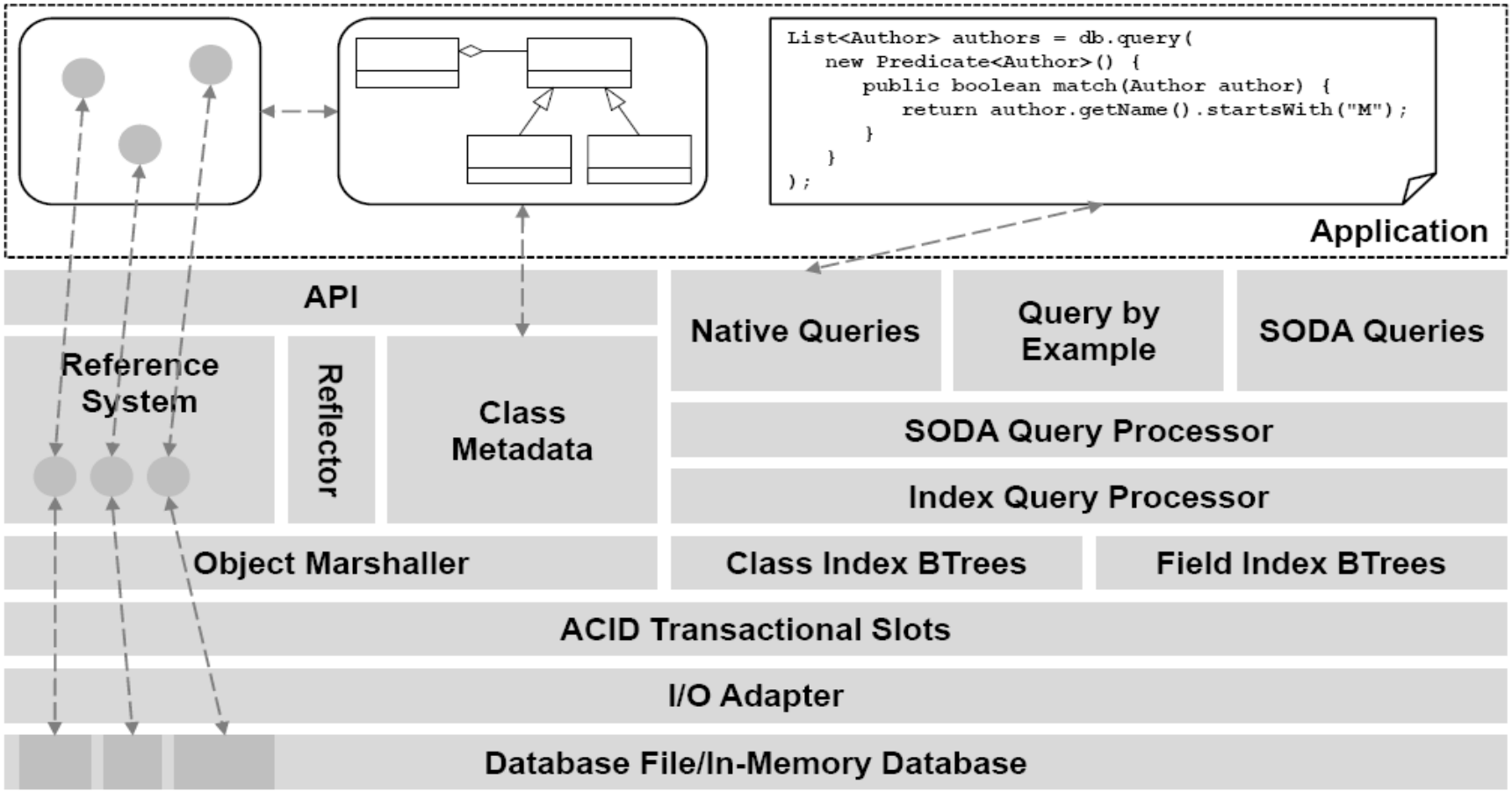
www.db4o.com

Quelle zu folgenden Folien:
"Object Oriented Databases - Complete and up to date lecture series", Michael Grossniklaus, Moira Norrie, ETH Zürich, November 2007

db4o

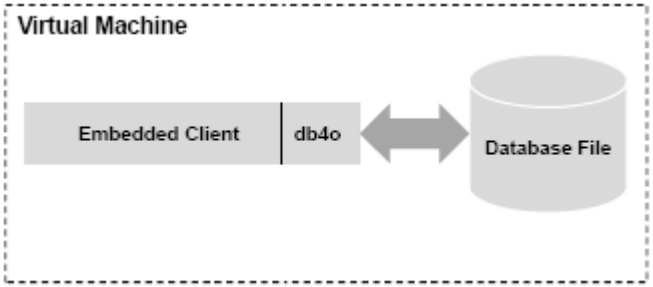
- Open source Native OODBMS für Java und .NET
- Merkmale
  - Kein O/R-Mapping
  - Keine Anpassungen an Klassen für Objektpersistenz
  - Eine Zeile, um ein Objekt jeder Komplexität zu speichern
  - Local und Client/Server Modus
  - ACID-Transaktionsmodell
  - Object Caching, Integration mit Garbage Collector
  - Autom. Verwaltung und Versioning des Datenbank-Schemas
  - Java und .NET Language Binding
  - Kleiner “Memory foot-print” (eine Library, 500Kb)

db4o-Architektur

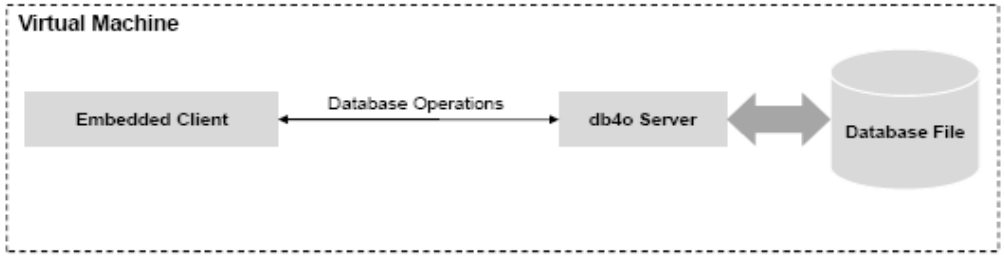


Architektur / Modi

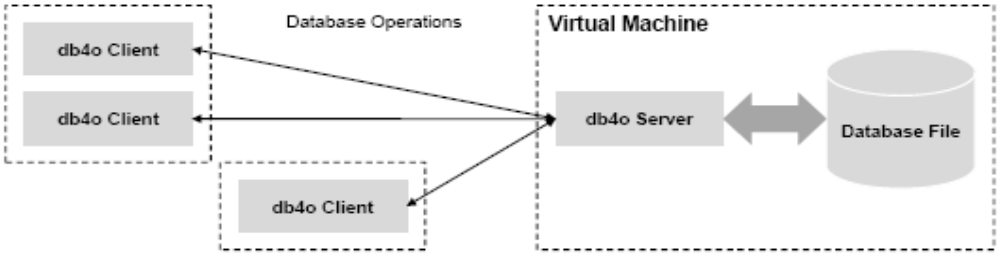
- Local Mode



- Client/Server Mode: Embedded



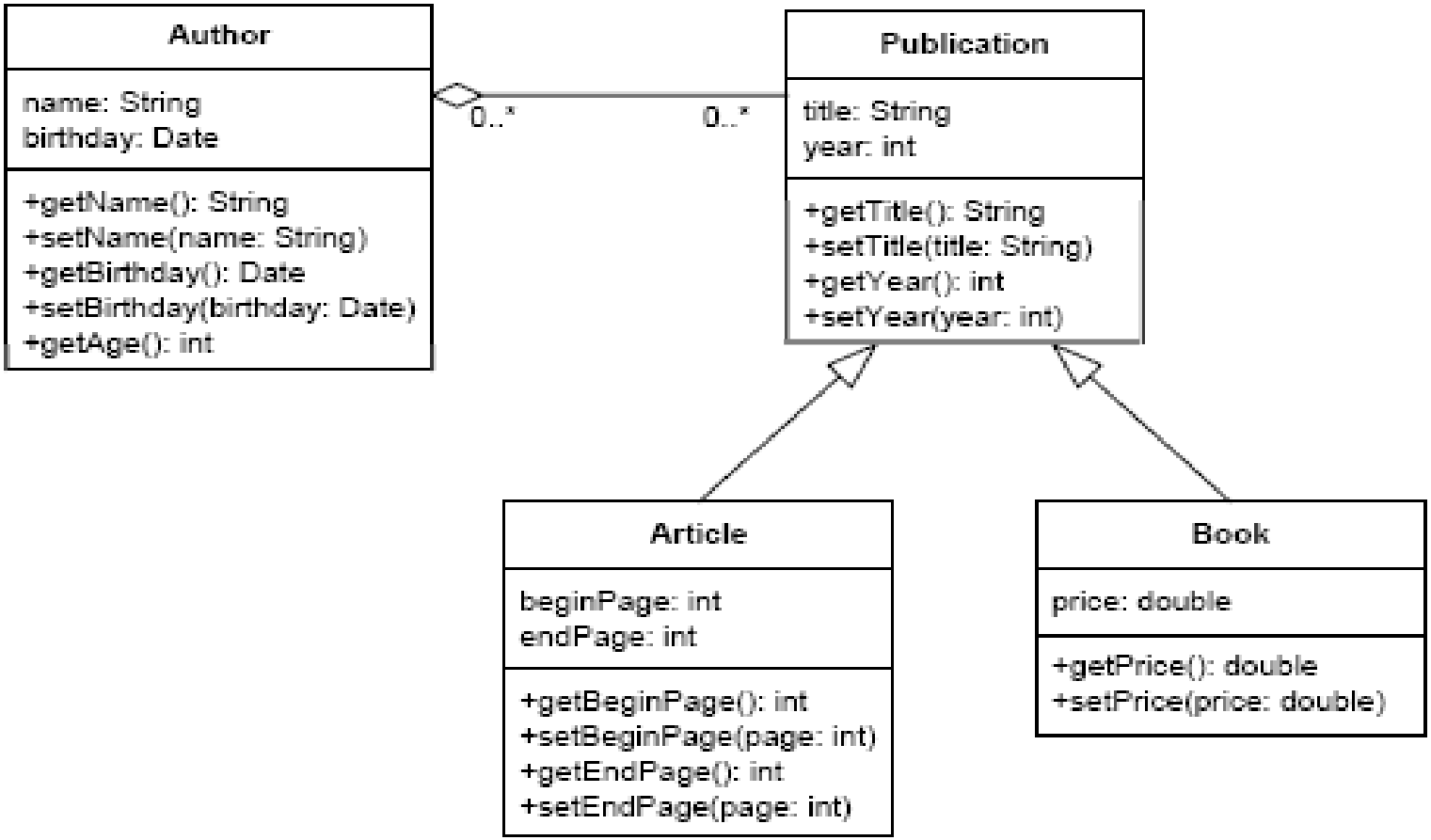
- Client/Server Mode: Networking



Architektur / Modi

- Local (‚embedded') Mode:
  - Standalone Database
  - öffnet und greift auf Datenbank-Datei direkt zu
  - 1 Benutzer, 1 Prozess oder 1 Thread aufs Mal
- Client/Server Mode:
  - Mehrere Clients und ein zentraler Server
  - Networking Mode
    - TCP/IP connection to server
    - Methode Db4o.openServer(filename, port)
    - Methode Db4o.openClient(host, port, user, pass)
  - Embedded Mode: gut für Multi-Threaded Apps.
    - Kein Netzwerk, Server auf Port 0
    - Client ObjectServer.openClient()

Beispiel: Klassen-Hierarchie



Objekte verwalten, speichern und wieder abrufen

```
public class Author {
  private String name;
  private Date birthday;
  private Set<Publication> pubs;
  public Author(String name) {
    this.name = name;
    this.pubs =
      new HashSet<Publication>();
  }
  public String getName() {
    return this.name;
  }
  public void setName(String name) {
    this.name = name;
  }
  ...
}
```

```
public class Publication {
  private String title;
  private int year;
  private List<Author> authors;
  public Publication(String title) {
    this.title= title;
    this.authors =
      new ArrayList<Author>();
  }
  public String getTitle() {
    return this.title;
  }
  public void setTitle(String title) {
    this.title = title;
  }
  ...
}
```

Object Container

- Repräsentiert db4o-DB
  - Local Modus (Datei), Client/server modus
- Verwaltet eine Transaktion
  - alle Operationen sind in einer Transaktion
  - Transaktion startet implizit wenn Objekt Container geöffnet wird
  - Nach commit/rolback beginnt autom. nächste T.
- Verwaltet Referenzen zu gespeicherten und instantiierten (transienten) Objekten
  - inkl. Objekt-Identitäten
- Lebenszyklus
  - Object Container bleibt offen solange Programm läuft
  - Wenn Object Container geschlossen wird, werden Referenzen zu Objekten im Memory weggeworfen

Objekte speichern

- mit store-Methode der Klasse ObjectContainer
- Speichert ganzes Objekt (deep copy)
- Persistence by reachability

```
// create a publication
Publication pub =
  new Publication("Route Finding and db4o");
// create authors
Author keller = new Author("Stefan Keller");
Author mueller = new Author("Lothar Müller");
// assign authors to publication
pub.addAuthor(keller);
pub.addAuthor(mueller);
// store complex object
database.store(pub)
```

Objekte wieder abrufen

db4o kennt drei Query-Sprachen

- Query by Example
  - Einfacher Ansatz mit Prototyp-Objekten
  - erlaubt nur exakte ‘Matches’
- Native Queries
  - Ausgedrückt in Programmiersprache (Java, .NET)
  - typsicher
  - transformed to SODA and optimised
  - (Zukunft: LINQ - Language INtegrated Query. LINQ - Language INtegrated Quer)
- Simple Object Data Access (SODA)
  - Query API auf Basis von Query-Graphen
  - Wendet Graph-Traversierung und Constraints an

Query by Example: Beispiel

```
ObjectContainer database = Db4o.openFile("pub.db");

// get author "Stefan Keller"
Author prototype = new Author("Stefan Keller");
ObjectSet<Author> authors = database.queryByExample(prototype);
for (Author author: authors) {
  System.out.println(author.getName());
}

// get all publications
ObjectSet<Publication> publications =
  database.queryByExample(Publication.class);
for (Publication publication: publications) {
  System.out.println(publication.getTitle());
}
```

Native Queries: Beispiel

```
ObjectContainer database = Db4o.openFile("pub.db");

// find all publications after 2001
ObjectSet<Publication> publications = database.query(
  new Predicate<Publication>() {
    public boolean match(Publication publication) {
      return publication.getYear() > 2001;
    }
  }
);
for (Publication publication: publications) {
  System.out.println(publication.getTitle());
}
```

SODA Query

- Expressed using Query objects
  - descend: adds or traverses a node in the query tree
  - constrain: adds a constraint to a node in the query tree
  - sortBy: sorts the result set
    - orderAscending and orderDescending
  - execute: executes the query
- Interface Constraint
  - greater and smaller: Vergleiche (Modus)
  - identity, equal, like: Ausdrucks-Auswertung
  - and, or und not Operatoren
  - startsWith, endsWith: String-Vergleiche
  - contains: Tested Collection membership

SODA Query: Beispiel

```
ObjectContainer database = Db4o.openFile("pub.db");

// find all publications after 2001
Query query = database.query();
query.constrain(Publication.class);
query.descend("year").constrain(Integer.valueOf(2001)).greater();
ObjectSet<Publication> publications = query.execute();
for (Publication publication : publications) {
  System.out.println(publication.getTitle());
}

// find all publications of author "Stefan Keller"
Query query = database.query();
query.constrain(Publication.class);
Author prototype = new Author("Stefan Keller");
query.descend("authors").constrain(prototype).contains();
ObjectSet<Publication> publications = query.execute();
for (Publication publication : publications) {
  System.out.println(publication.getTitle());
}
```

Objekte updaten

- Vorgehen
  - Objekt von der DB holen
  - Objekt ändern
  - Objekt mit store-Methode in DB zurückspeichern
- Hintergrund
  - db4o benutzt IDs, um in-Memory-Objekt mit gespeichertem Objekt zu verknüpfen
  - IDs sind (Java) Weak References bis DB schliesst
  - Objekt-Query, create und set garantieren eine neue, aktuelle Referenz

Objekte updaten: Beispiel

```
EmbeddedConfiguration config = Db4oEmbedded.newConfiguration();
ObjectContainer database = Db4o.openFile(config, "pub.db");

Author keller =
  (Author) database.queryByExample(
    new Author("Stefan Keller")).next();
Calendar calendar = Calendar.getInstance();
calendar.set(1961, Calendar.NOVEMBER, 10);
keller.setBirthday(calendar.getTime());
database.store(keller);
```

next() liefert hier das erste (und hoffentlich einzige) Objekt vom Iterator auf das ObjetSet, das von get() zurückgegeben wurde.

Objekte löschen

- Ähnlich wie Objekte updaten
  - neue Referenz verlangen
  - entweder durch Query oder create/store
- Methode delete der Klasse ObjektContainer

```
ObjectContainer database = Db4o.openFile("pub.db");

// retrieving author "Lothar Müller"
Author mueller =
  (Author) database.queryByExample(
     new Author("Lothar Müller")).next();

// deleting author "Lothar Müller"
database.delete(mueller);
```

Einfache Objekte

- Speichern mit store-Methode
  - Objekt-Graph wird traversiert und alle referenzierten Objekte werden persistiert
  - Persistence by reachability!
- Updaten mit store-Methode
  - Default für Update-Tiefe ist 1 (shallow copy)
  - Nur Basistypen und Strings werden aktualisiert
  - Objekt-Graph wird wegen Performance nicht traversiert
- Löschen mit delete
  - Delete kaskadiert per Default nicht
  - Referenzierte Obj. muss man händisch löschen
  - Kaskadierendes Delete kann pro Klasse konfiguriert werden

Updating Simple Structured Objects: Bsp.

```
ObjectContainer database = Db4o.openFile("pub.db");

// retrieving author "Lothar Müller"
Author mueller =
  (Author) database.queryByExample(
    new Author("Lothar Müller")).next();
// updating all publications
for (Publication publications: mueller.getPublications()) {
  publication.setYear(2007);
}
// storing author "Lothar Müller" has no effect on publications
database.store(mueller);
// storing updated publications
for (Publication publications: mueller.getPublications()) {
  database.store(publication);
}
```

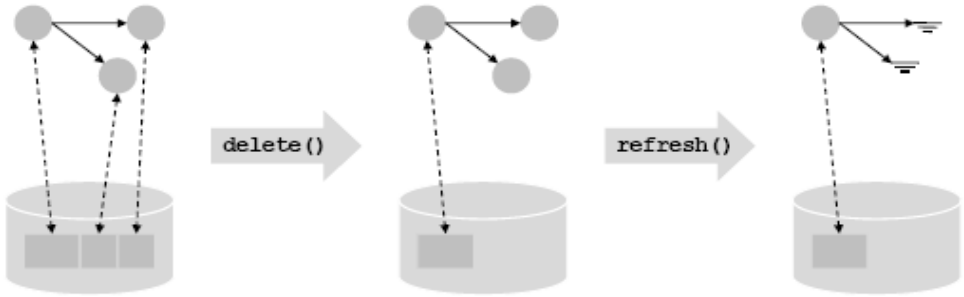
Deleting Simple Structured Objects

- Ähnlich wie kaskadierende Updates
  - Pro Klasse konfiguriert
  - Methode objectClass von Configuration
  - Methode cascadeOnDelete von ObjectClass

```
ObjectContainer database = Db4o.openFile("pub.db");

// configuration of cascading deletes for Author objects
Db4o.configure().objectClass(Author.class).cascadeOnDelete(true);
// retrieving author "Lothar Müller"
Author mueller =
  (Author) database.queryByExample(
     new Author("Lothar Müller")).next();
// deleting author "Lothar Müller"
database.delete(mueller);
```

Deleting Simple Structured Objects

- Löschen erzeugt Differenz zu In-Memory-Objekt und gespeichertem Objekt
  - cache und Disk (Datei) werden inkonsitent
  - Methode refresh von ExtObjectContainer schafft Abhilfe („Synchronisation“)
  - Frischt Memory mit „Commited“-Objekt von der Disk (Datei) auf

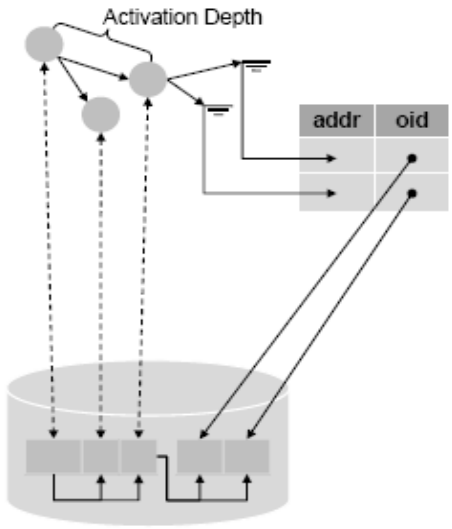


Komplexe Objekte: Objekt-Hierarchien

- db4o verwaltet komplexe Objektstrukturen automatisch
  - einfache und zusammengesetzte (composite) Hierarchien
  - Inverse Assoziationen
  - Vererbung und Interfaces
  - Mehrwertige Attribute, Arrays und Collections
- Kaskadierende Operationen sind möglich
- db4o bietet „database-aware“ Collections: *com.db4o.collections.ArrayList4* und *ArrayMap4* (ab v7.0)
- Macht Implementation von db4o abhängig

Aktivierung

- kontrolliert Instantiierung von Objekt-Attributwerten
  - Attributwerte nur bis zu gew. Tiefe ins Memory geladen
  - Tiefe: Länge der Referenzkette zwischen zwei Objekten
  - Attribute jenseits der Aktivierungstiefe sind null (für Basistypen), bzw. Defaultwerte.
  - Default ist 5
- Aktivierung erfolgt,
  - bei next auf ObjectSet einer Query
  - bei activate (Klasse ObjectContainer)
  - bei Zugriff auf Element einer db4o-Collection
  - bei Aktivierung einer (ganzen) Java Collection
- Verwaltung
  - nicht geladene Attributwerte über Weak References
  - inaktive Objekte über Mapping Table



Transaktionen

- db4o kennt ACID Transaktions-Modell
  - read committed isolation level
  - d.h. Inkonsistenzen können immer noch vorkommen
- Transaktions-Log
  - Kein Datenverlust bei Absturz
  - Autom. Recovery danach
- db4o Core ist Thread-safe
- Jede Operation auf ObjectContainer ist eine Transaktion
  - T. startet implizit bei Öffnen des Containers
  - T. auto. commit bei Schliessen des Containers
  - Explizites commit() und rollback()

Transaktionen: Beispiel

```
ObjectContainer database = Db4o.openFile("pub.db");

// retrieving author "Lothar Müller"
Author mueller =
  (Author) database.queryByExample(
     new Author("Lothar Müller")).next();
// creating author "Dominik Wild"
Author wild = new Author("Dominik Wild");
// creating new publication
Publication article =
  new Publication("Internet Technologien");
article.addAuthor(wild);
article.addAuthor(mueller);
// storing publication
database.store(article);
// committing database
database.commit();
```

Transaktionen: Rollback

- Trotz rollback sind Inkonsistenzen möglich
  - rollback stellt „nur“ konsistenten Zustand der DB sicher; Cache bleibt unberührt
  - In-Memory-Objekte müssen erneuert werden

```
ObjectContainer database = Db4o.openFile("pub.db");
// retrieving publication
Publication article =
  (Publication) database.queryByExample(
     new Publication("Internet Technologien")).next();
// updating publication
Author keller = new Author("Stefan Keller");
article.addAuthor(keller);
database.store(article);
// aborting transaction
database.rollback();
// refreshing article to remove author from in-memory representation
database.ext().refresh(article, Integer.MAX_VALUE);
```

Callbacks

- Methoden, aufgerufen von Events (triggers)

Anwendungsfälle:

- Integritätsprüfung / Check vor Speichern
  - Attributwerte prüfen mit canNew() and canUpdate()
- Updates loggen oder verhindern:
  - canUpdate() und onUpdate()
- Transiente Variablen setzen/wiederherstellen
  - Graphische Elemente oder Netzverbindungen

Callbacks

- db4o-Events
  - activate / deactivate
  - new, update, delete
- Methoden called before and after event
  - Methoden beginnend mit can… werden vor dem Event aufgerufen
  - Methoden beginnend mit on… werden nach dem Event aufgerufen
- Interface ObjectCallbacks

Konfiguration und Tuning

- Schnittstelle
  - Globale Konfiguration mit Db4o.configure()
  - Aktuelle globale Settings werden geclont wenn ein neuer Objekt Container/Server geöffnet wird
- Index - optimiert Query-Auswertung
  - B-Tree auf einzelne Attribute
  - Db4o.configure().objectClass(...).objectField(...).indexed(true);
  - werden autom. erzeugt beim Öffnen, bzw. gelöscht beim Schliessen eines Objekt-Containers
- Externe Werkzeuge
  - Performance Tuning: Defragmentieren
  - DB-Diagnose: Statistik
  - Logger

Replikation und Schema-Evolution

- Replikation (seit v.5), sep. Packages
  - Daten ändern auf Master/Publisher
  - Änderungen werden an (read-only) Clients/Subscriber weitergeleitet
  - uni- oder bidirectional
  - Brücke zu RDBMS
  - db4o->db4o, db4o->Hibernate, Hibernate->db4o

- Schema-Evolution
  - Änderungen an Klassen und Vererbung
  - einfacher handhabbar als in RDBMS

Literatur

- db4o Tutorial:
  http://www.db4o.com/about/productinformation/resources/
- db4o Referenzen:
  http://developer.db4o.com/Resources/
- Bücher:
  - J. Paterson, S. Edlich, H. und R. Hörning: The Definitive Guide to db4o, APress, 2006
  - P. Römer & L. Visengeriyeva: db4o. schnell + kompakt, Entwickler.Press, 2006

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Objektpersistenz: Einführung

Datenbanksysteme 2
Prof. Stefan Keller

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Inhalt

- Persistenz von Objekten
- Objektpersistenz-Standards
- O/R-Mapping: Prinzipien und Produkte/Projekte
- Diskussion

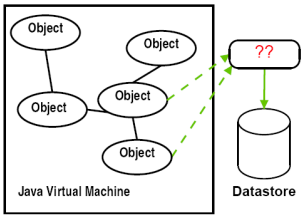
HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Warum persistente (Programm-)Objekte?

- Problem:
  - Objekte einer objektorientiereten Programmiersprache (wie z.B. Java) sind transient (flüchtig), also bei Programmende verloren
- Die Verwaltung persistenter Daten ein Standardproblem der Software-Entwicklung
  - Diskussionen, ob CRUD-Operationen händisch codiert oder automatisiert werden sollten
  - Konsens: automatisiert!
  - Frage ist Wie?

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Gewünschte Eigenschaften der Persistenz



- Transparenz:
  - Benutzer arbeiten in gleicher Weise mit transienten und persistenten Objekten.
  - Persistenz erfordert keine Sonderbehandlung bei Programmierung
- Interoperabilität:
  - Persistente Objekte können auch in anderen als der Erstellungsumgebung verwendet werden. Das Festschreiben ist vom Persistenzsystem unabhängig. Laufzeitumgebung und persistenter Speicher sind austauschbar.
- Leistung/Skalierbarkeit:
  - Auffinden von persistenten Objekten erfolgt ohne vollständiges Durchsuchen des Objektpools
- Mehrbenutzer/Sharing, Konflikterkennung, Verteilung
- Anfragesprachen

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Welche Persistenztechniken für (Java-)Objekte gibt es?

1. Objektserialisierung
2. Objektdatenbanksysteme (ODMBS)
3. Objekt-Relationales Mapping (O/R-Mapping)

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

1. Objektserialisierung

- Schreibt Objekte aus dem Speicher direkt in Streams
- Standard in Java integriert

+ völlig transparent
+ einfacher Einsatz
- kein inkrementelles Laden
- kein Sharing (Transaktionsmanagement)
- schlechte Interoperabilität (SourceCode und Sprache sind fix)
- keine Anfragesprache (Query)

- Verbesserungsansatz: Persistent programming languages z.B. Persistent Java (PJama ☺), Uni. Glasgow

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
INFORMATIK

2. Objektorientierte Datenbanksysteme

+ objektorientiert, daher gute Transparenz
+ einfaches bzw. praktisch kein Mapping
+ gute IDE-Integration inkl. Refactoring
- schlechte Interoperabilität, da immer noch kein Objektmodell- und Query-Standard (vgl. Folien ODMG-Standard)
- begrenzte Datenmengen
- schlechte Leistungen bei grossen Datenmengen
- mangelnde Tool-Unterstützung

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
INFORMATIK

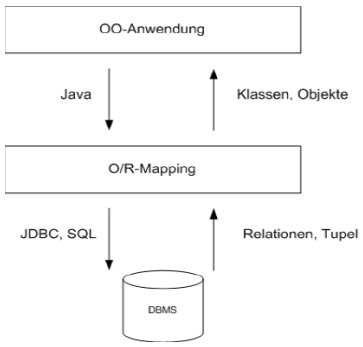
3. O/R-Mapping

Grundlagen und Standards

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

O/R-Mapping Allgemein

- Brücke vom objektorientierten ins relationale Paradigma



HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

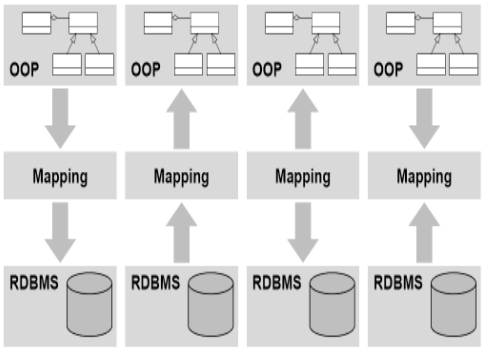
Wie kommen Objekte, Query und Mapping-Definitionen zusammen?

- Top down
  - Beginne mit Business-Modell in Java, erzeuge Mapping-Meta-Daten, erzeuge (generiere) DB-Schema
- Bottom up
  - Existierendes DB-Schema. Reverse-Engineering (händisch/automatisch), um Mapping-Meta-Daten zu erzeugen. Erzeuge (generiere) Java-Business-Modell
- Middle out
  - Mapping-Meta-Daten reichen aus, um sowohl Java, als auch DB-Schema zu generieren.
- Meet in the middle
  - Existierendes Java-Modell und DB-Schema. Wir schreiben Mapping-Meta-Daten. Schwierigster Fall und in der Regel nicht ohne Änderung am Business-Modell und/oder DB-Schema möglich
- Plus: Meta Model / Model driven

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Generierungs-Möglichkeiten / O/R-Mapping-Varianten

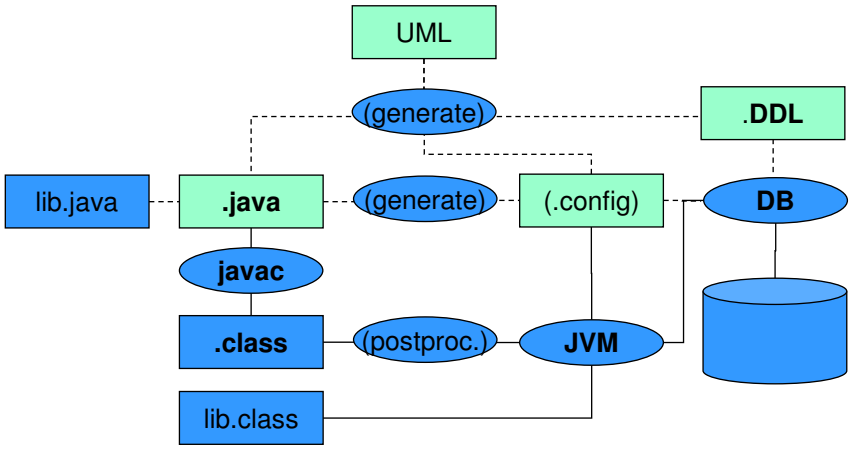
1. Forward Engineering (Code first, ‚Top-Down')
2. Bottom Up / Reverse Engineering (DB first)
3. Middle-out/Inside-out (Mapping first)
4. Meet-in-the-middle / Outside-in
5. Meta Model (‚model-driven')



Quelle: Fowler (2003): „Patterns Of Enterprise Application Architecture"

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Generierungs-Möglichkeiten / O/R-Mapping-Varianten



HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Realisierung des O/R-Mappings

- API: Factories, DB-Managers, Query-Klassen
- Data Type Mapping: Defaults für primitive Typen
- Primärschlüssel: Generierung und zusammengesetzte Schlüssel
- Beziehungen zwischen Entitäten:
  - Navigation,
  - Cascading,
  - Lazy-Loading
- Vererbung: Strategien
- Abfragesprache(n)
- Callbacks und Listeners
- Lebenszyklus
- Transaktionen: Exceptions und Rollback

Vgl. auch Modul „Enterprise Computing“, Wolfgang Giersche

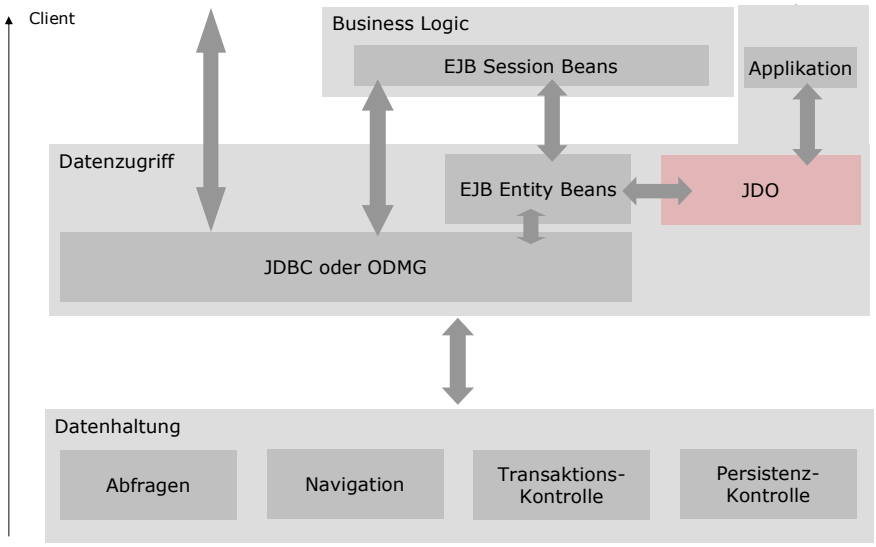
HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

O/R-Mapping Standards: Überblick

- Manuell => JDBC
- Java Data Objects (JDO)
- Java Persistence API (JPA/EJB3)

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

JDBC, EJB und JDO im J2EE-Umfeld



HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Manuelles OR-Mapping

… via JDBC
oder anderen DB-APIs

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
INFORMATIK

Manuelles OR-Mapping

Objekte werden in RDBMS meist mittels JDBC gespeichert

+ Optimale Leistung
+ Interoperabilität zwischen verschieden RDBMS
- RDBMS beschränkt
- SQL-Kenntnisse vorausgesetzt
- Keine Transparenz, komplexerer Code
- Low level programming, grosser Arbeitsaufwand

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
INFORMATIK

Java Data Objects (JDO)

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Java Data Objects (JDO)

- "…what Sun intended to be the Java object persistence standard"
- Realisierung
  - Angelehnt an ODMG Standard
  - Byte Code Enhancer
  - Projekt DataNucleus ist Referenz-Implementation
- Geschichte:
  - 2005: JDO 2.0 (JSR 243): Extension, 332 pages
  - 2008: praktisch von Hibernate „überrollt" und durch JPA „abgelöst" (viel Politik!)

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Unterschiede JDO zu ODMG

+ Java Standard-Collections (keine DSet, DList, ...)
+ Gleichzeitiger Gebrauch verschiedener DB-Systeme
+ Unterstützt auch ODBMS!
– Isolation Levels und explizites Locking fehlen
– Proprietär bezügl. Programmiersprache: Java
– Schwindende Benutzerbasis

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
INFORMATIK

Java Persistence API (JPA)

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
INFORMATIK

J2EE Entity Beans

- Container Managed Persistence CMP
- EJB 3.0 mit Teil Java Persistence API (JPA)
- EJB3 spec: 818 pages!

+ Völlige Transparenz
+ gehört zu J2EE Standard, daher garantierte Interoperabilität (?)
(-) Allgemein grosser Konfigurationsaufwand für J2EE
(-) zusätzlicher Aufwand erbringt etwas schlechtere Leistungen als bei JDBC

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Java Persistence API (JPA)

- Formell Teil der EJB3-Spec. bzw. J2EE. From java,sun.com:
  The Java Persistence API provides a POJO persistence model for object-relational mapping. The Java Persistence API was developed by the EJB 3.0 software expert group as part of JSR 220, but its use is not limited to EJB software components. It can also be used directly by web applications and application clients, and even outside the Java EE platform, for example, in Java SE applications. See JSR 220.
- Bemerkungen:
  - J2SE 5.0 bringt Annotations: Alternative zu XML
  - Ideen von Hibernate, Oracle TopLink, and JDO sowie früherer EJB container-managed persistence (CMP)
  - Design-Prinzipien:
    - Intuitive Defaults, dann Annotations, dann erst XML.
    - Zu jeder Annotation gibt es eine Alternative in persistence.xml

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

JPA Building Blocks

- Annotations beschreiben Bindung an die Datenbank
- Packages/Klassen
  - javax.persistence.EntityManager
    - EntityManager.persist (Object pojo)
    - EntityManager.createQuery (String query)
  - javax.persistence.Query
- Persistence Context (META-INF/persistence.xml)

```
<persistence>
  <persistence-unit name="greenbill">
  <jta-data-source>java:/MySqlXaGreenBillDs</jta-data-source<
  <properties><property name="…">…</property></properties>
 </persistence-unit>
</persistence>
```

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

POJOs annotieren: Beispiel

Currency.java

```
@Entity
@Table (name="currencies")
public class Currency {
  private Long id;
  private String currencyCode;

  @Id @GeneratedValue
  public Long getId() { return id };
  public void setId ( Long id ) { this.id = id };

  @Column (name="currency_code", nullable="false" )
  public String getCurrencyCode() {
     return currencyCode;
  }
  public void setCurrencyCode (…) {…}
}
```


HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Unterschiede JDO und JPA

- Vorteile:
  - OO!
  - Klassen mit echter Vererbung
  - Feine Zustands-Kontrolle (Live Cycle)
- Einschränkungen JDO
  - Class must habe default constructor (can be private)
- Diskussion
  - Konfig. separat ist umständlich (mangels Tools) aber nötig
  - Schritt 3 Postprocessing wäre mit Java-Interface-Implementationen nicht nötig (generieren?)

- Vorteile:
  - Kein separates XML
  - Bem.: Best Practices Trennung von Schema und Code (= sep. XML-Konfig.)
- Einschränkungen JPA
  - Class can't be final
  - Class can't have final methods
  - Class must have a non-private default constructor.
  - Class must have a field that stores the "identity„
- Diskussion
  - Immer sep. „Id“
  - Vererbung umständlich

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Vergleich einiger Persistence Standards

	JDO	Serializable	JDBC	OR Mapping	OODMBS	EJB/CMP (inkl. JPA)
Paradigmus	O.O	O.O	SQL	O.O	O.O	O.O
Transparenz	+	+	-	+	+	+
Interop.	++	+	+	-	o	++
Leistung	o	-	+	o	o	o
Funktionalität	+	--	o	+	+	++
Caching	auto.	kein	kein	ja	auto.	auto.
Mapping	auto.	auto.	explicit	auto.	auto.	file
Datastore	universal	Streams	RDBMS	RDBMS	OODBMS	? [a]

[a] von Application Server abhängig

Quelle: JDO

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Language INtegrated Query (LINQ)

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Language INtegrated Query (LINQ)

- Idee:
  - Abstrahierte Anfragesprache von Collections (Tabellen, Objekte, XML)
  - typsicher und in Sprache „eingebettet“
- Syntax angelehnt an SQL-Anfragesprache
  - Sog. „standard query operators“ (analog Relationale Algebra)
  - Erzeugt und manipuliert Syntaxbäume
- Mapping zu Objekten, SQL und XML (XLink- und XQuery)
  - LINQ to SQL: LINQ-Abfrage wird zu SQL-Statement umgewandelt und gemappt (gem. O/R-Mapping)

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Implementation

- Spracherweiterungen:
  - Übergabe und Rückgabe von Programmvariablen an Query => Variablen leben nach Ende Block weiter => ‘Closures’
  - Ergänzen von Methoden an bestehende Klassen zur Laufzeit (Lambda-Ausdrücke)
- Java: noch nicht möglich => Java 7?
- Teil von Microsofts .NET-Framework seit Version 3.5
  - Befehle sprechen mit Hilfe der Erweiterungen direkt .NET-Objekte an (Erweiterungen von IEnumerable).
  - LINQ-Framework enthält u.a. SQLMetal für Codegenerierung von Wrapper-Klassen für C# mit DLINQ (MS SQL Server)
- Noch nicht abgeschlossen… z.B. rekursive Anfragen

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
INFORMATIK

Beispiel

Alle Produkte, deren Produktbezeichnung mit einem A beginnt sortiert nach ID:

```
var query = from product in this.Products
            where product.Name.StartsWith("A")
            orderby product.ID
            select product;

foreach ( var product in query )
{
    Console.WriteLine ( product.Name );
}
```

Alternative mit Erweiterungsmethoden (Lambda-Ausdrücke)

```
var query = this.Products
              .Where(p => p.Name.StartsWith("A"))
              .OrderBy(p => p.ID);

foreach ( var product in query )
{
    Console.WriteLine ( product.Name );
}
```


HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
INFORMATIK

O/R-Mapping: Produkte/Projekte

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

O/R-Mapping: Produkte/Projekte

- Kommerzielle Produkte:
  - EJB/JPA: TopLink (Oracle); CocoBase (by Thought Inc.)
  - JDO+JPA: Xcalia Inter(by Progress)
- Open Source Projekte mit JPA:
  - EclipseLink, Teil von GlassFish (Sun), „verbandelt“ mit TopLink, Referenzimplementation JPA 2.0
  - Hibernate (Java) und NHibernate (.NET)
  - Apache iBATIS (Java und .NET)
- Open Source Projekte mit JDO/ODMG:
  - DataNucleus (ex. JPOX): bewährt, mit OO-Anbindung
  - Apache DB ObJectRelationalBridge (OJB)
  - Apache OpenJPA (ex. Kodo) (by bea/Oracle);
- Weitere:
  - Apache Cayenne u.a. mit GUI Modeler
- Siehe “What do you prefer for enterprise Java persistence?” 2006 (auf http://java.net/pub/pq/122)

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

O/R-Mapping: Produkte/Projekte

- Allen (Open Source) Projekten gemeinsam:
  - DB Support:
    - MySQL, PostgreSQL, Oracle, HSQLDB, Apache Derby, etc.
  - O/R mapping Definition (auch) in XML
  - native and generated primary keys

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
INFORMATIK

OR-Mapping: Diskussion

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
INFORMATIK

OR-Mapping Frameworks

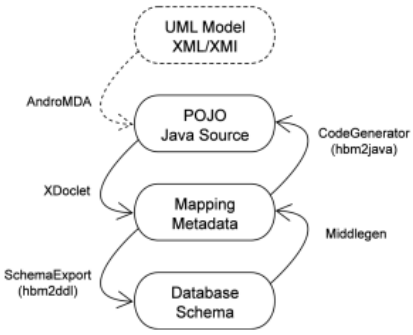
- OO-Persistenz in RDBMS wird von spezieller Software übernommen, z.B. Hibernate

\+ gute Transparenz

\+ weniger Arbeitsaufwand

\+ mittlerweile gute Leistungen

\- Interoperabilität (KGN) oder auf DB beschränkt



HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

O/R-Mapping: Probleme

- Query-Sprache ‚bindet' Domain von Persistenzschicht
- Basis-Datentypen-Unterschiede:
  - Gemeinsame JDBC 3.0-Datentypen, einiger wichtiger Datenbanken: PostgreSQL, Oracle, MS-Access, MS SQL-Server, IBM DB2, MySQL):
    - SMALLINT, INTEGER, REAL, DOUBLE, NUMERIC, CHAR, VARCHAR, LONGVARCHAR, VARBINARY, LONGVARBINARY und TIMESTAMP.
- Darunterliegende Unterschiede scheinen durch:
  - Begrenzte Tabellen- und Feldnamen-Längen! gemeinsamer Nenner: nur 30 Zeichen: PostgreSQL: 512, Oracle: 30, MS-Access: 64, MS SQL-Server: 128, DB2: 128, MySQL: 64.
- Erweiterbarkeit: schlechte bis keine; z.B. für benutzerdef. Datentypen

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

O/R-Mapping: Standards

- Gegeben: RDBMS – mehr oder weniger abstrahiert
- Standard-Abbildung ist möglich
- Bedarf nach Konfiguration bleibt (z.B. für DB spezifische Typen und Funktionalitäten)
- Tools und Standards erst am Kommen und ihrem Standard voraus,
- Noch nicht ausgereift, z.B. Benutzerdef. Datentypen

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Erfahrungen

- Erfahrungen DataNucleus (JDO/JPA)
  - JDO bietet bessere OO-Kompatibilität
  - Best of the open source offerings
  - Kein lazy-loading von grossen Recordsets
  - Two-man-show
  - Performance im Vergleich mit JDBC und OODBMS?
- Erfahrungen Hibernate:
  - Ok, falls RDBMS gesetzt und Standard nicht im Vordergrund
  - LINQ bei .NET gesetzt
- Fazit:
  - Standards bieten mehrere Implementationen
  - ODMG und JDO sind tot, lang lebe ODMG und JDO!
  - Vieles noch im Fluss

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Quellen

- Bernd Müller: Java-Persistence-API mit Hibernate Standardisierte Persistenz
- JDO: http://www.fh-wedel.de/mitarbeiter/iw/lehrveranstaltungen-in-frueheren-semestern-ab-ss-2007/ss-2007/objektorientierte-datenbanken/

- Addison-Wesley, ISBN 978-3-8273-2537-2
- Christian Bauer / Gavin King: Java Persistence with Hibernate, Manning 2006, ISBN 1-932394-88-5
- David Jordan / Craig Russell: Java Data Objects, O'Reilly 2003, ISBN 0-596-00276-9
- Chris Richardson: POJOs in Action, Manning 2006, ISBN 1932394583 (Vortrag des Autors über den Buchinhalt)
- Robin Roos: Java Data Objects, Addison-Wesley 2002, ISBN 0-321-12380-8

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

O/R-Mapping:
JPA mit EclipseLink (1.Teil)

Datenbanksysteme 2

Prof. Stefan Keller

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Inhalt Java Persistence API (JPA)

- Teil 1 – Kleine Tour! **(diese Folien)**
- Teil 2 – JPA 2 im Detail
- Quellen:
  - „Pro JPA 2 - Mastering the Java Persistence API“, M. Keith and M. Schincariol, Verlag apress, 2009.
  - "Java Persistence API (JPA) with EclipseLink – Tutorial“, Lars Vogel, 2009, www.vogella.de/articles/JavaPersistenceAPI/article.html

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Java Persistence API (JPA)

- Java Persistence API, JPA 2
- Bietet Persistenz für POJOs mittels O/R Mapping
- Formell Teil der EJB3-Spec. bzw. J2EE (seit 5), JSR 220
- Nicht an EJB Software gebunden, d.h. funktioniert sowohl in J2EE als auch in J2SE Umgebungen

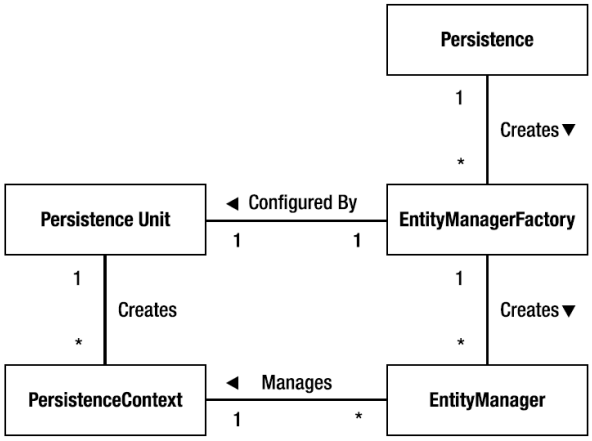
HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

JPA Entwurfsprinzipien

- Beeinflusst von Hibernate, Oracle TopLink, and JDO sowie früherer EJB Container Managed Persistence (CMP)
- Annotations (als Alternative zu XML seit Java 5)
- Configuration by Exception: Intuitive Defaults, dann Annotations, dann erst XML.
- Zu jeder Annotation gibt es eine Alternative in persistence.xml
- Persistence Provider implementieren die JPA Spezifikation, z.B. EclipseLink

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

JPA Architektur



HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Kleine Tour!

- Installation EclipseLink
  - www.eclipse.org/eclipselink/downloads/
    - eclipselink.jar und persistence.jar
  - Eintrag eines JPA Providers in Classpath
    - Falls mehrere im Classpath, kann dies in persistence.xml angegeben werden:
      `<provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>`
- Installation und Start eines DB-Servers
  - Z.B. PostgreSQL, H2 oder Derby

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Tour: Beispiel

- Klasse Employee
  - man beachte @Entity und @Id

```
@Entity
public class Employee {
    @Id private int id;
    private String name;
    private long salary;
    public Employee() {}
    public Employee(int id) { this.id = id; }
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public long getSalary() { return salary; }
    public void setSalary (long salary) { this.salary = salary; }
}
```


HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Tour: Entity Manager

- EmployeeService ist eine Persistence Unit

```
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory("EmployeeService");
```

- EntityManager:

```
EntityManager em = emf.createEntityManager();
```

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Tour: Entity speichern

- Entity persistieren:

```
Employee emp = new Employee(158);
em.persist(emp);
```

- Als Methode:

```
public Employee createEmployee(int id, String name, long salary) {
    Employee emp = new Employee(id);
    emp.setName(name);
    emp.setSalary(salary);
    em.persist(emp);
    return emp;
}
```


HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Tour: Entity finden und löschen

- Finden:

```
public Employee findEmployee(int id) {
    return em.find(Employee.class, id);
}
```

- Löschen:

```
public void removeEmployee(int id) {
    Employee emp = em.find(Employee.class, id);
    if (emp != null) {
        em.remove(emp);
    }
}
```

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Tour: Update und Transaktion

- Update

```
public Employee raiseEmployeeSalary(int id, long raise) {
    Employee emp = em.find(Employee.class, id);
    if (emp != null) {
        emp.setSalary(emp.getSalary() + raise);
    }
    return emp;
}
```

- Transaktion

```
em.getTransaction().begin();
createEmployee(158, "John Doe", 45000);
em.getTransaction().commit();
```


HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Tour: Query

- Java Persistence Query Language (JPQL)
- Query gegen das logische Entity-Modell
  - nicht gegen das physische relationale Schema
- Beispiel: Alle Employees…

```
public List<Employee> findAllEmployees() {
    TypedQuery<Employee> query =
        em.createQuery("SELECT e FROM Employee e",
                       Employee.class);
    return query.getResultList();
}
```

- Es gibt neben JPQL auch das „Criteria API“
  - Mehr dazu später

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
INFORMATIK

2. Teil

- Anfragesprachen: Query Language und Criteria API
- Data Type Mapping: Defaults für primitive Typen
- Primärschlüssel: Generierung und zusammengesetzte Schlüssel
- Beziehungen zwischen Entitäten:
  - Navigation,
  - Cascading,
  - Lazy-Loading
- Vererbung: Strategien
- Lebenszyklus
- Callbacks und Listeners
- Transaktionen

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

O/R-Mapping: JPA mit EclipseLink (2.Teil)

Datenbanksysteme 2

Prof. Stefan Keller

Inhalt Java Persistence API (JPA)

- Teil 1 – Kleine Tour!
- **Teil 2 – JPA 2 im Detail (diese Folien)**
  - Anfragesprachen: Query Language und Criteria API
- Was wir bisher wissen…
  - Object-Relational Mapping
  - O-R mapping, ORM: „technique of bridging the gap between object model and relational model.“

Anfragesprachen: Query Language

- Java Persistence Query Language (JPQL)
- Query gegen das logische Entity-Modell
  - nicht gegen das physische relationale Schema
- „JPQL is not SQL!
  - Its is a language for querying entities“
  - Anstelle Tabellen und Kolonnen Entities und Objekte
  - JPQL kann in SQL-Dialekte übersetzt werden
  - JPQL lehnt sich an SQL an wo sinnvoll

JPQL

- Simples Beispiel:

  SELECT e FROM Employee e

  – Gibt Null oder mehrere Employee-Ojekte zurück
  – Man beachte das Alias e

- Navigation mit dem Dot-Operator („.“) ff.

  SELECT e.department FROM Employee e

  – Gibt die Department-Objekte zurück. Employee hat many-to-one Relationship zu Department

JPQL

- Filter/Selektion

  SELECT e FROM Employee e
  WHERE e.department.name = 'NA42'
  AND e.address.state IN ('NY','CA')

  – Gibt Employee-Ojekte zurück
  – „entity expressions“ in der Where-Klausel

- Navigation und Projektion:

  SELECT e.name, e.salary FROM Employee e

  – Gibt Namen (String) der Employees zurück

JPQL

- Der Resultattyp eines JPSQL kann keine Collection sein – wie bei SQL, z.B. e.phones wäre ein Set von Collections
- Joins

```
SELECT p.number FROM Employee e, Phone p
WHERE e = p.employee
AND e.department.name = 'NA42' AND p.type = 'Cell‘
```

  - Umgeschrieben in echten JOIN:

```
SELECT p.number FROM Employee e
JOIN e.phones p
WHERE e.department.name = 'NA42' AND p.type = 'Cell'
```

JPQL

- Zwei Arten von Query Parameter
- 1. Positional Parameter Notation

  ```
  SELECT e FROM Employee e
  WHERE e.department = ?1 AND e.salary > ?2
  ```

- 2. Named Parameter Notation

  ```
  SELECT e FROM Employee e
  WHERE e.department = :dept AND e.salary >
  :base
  ```

JPQL

- Two approaches to define a JPQL query:
  - Dynamic Query
    - specified at runtime; nothing more than strings, therefore may be defined on the fly.
  - Named query
    - configured in persistence unit metadata (annotation or XML) and referenced by name. static and unchangeable; more efficient to execute

JPSQL: Dynamic Query

```
@Stateless
public class QueryServiceBean implements QueryService {
    @PersistenceContext(unitName="DynamicQueries")
    EntityManager em;
    public long queryEmpSalary(String deptName, String empName) {
        String query = "SELECT e.salary " +
                       "FROM Employee e " +
                       "WHERE e.department.name = '" + deptName +
                       "' AND " +
                       " e.name = '" + empName + "'";
        return em.createQuery(query, Long.class).getSingleResult();
    }
}
```

JPSQL: Named Query

```
@NamedQuery(name="findSalaryForNameAndDepartment",
    query="SELECT e.salary " +
          "FROM Employee e " +
          "WHERE e.department.name = :deptName AND " +
          " e.name = :empName")
```

Platziert am ehesten in der Nähe der Resultattyps, hier also Employee.

Anfragesprachen: Criteria API

- Method Chaining, Java Generics

- Einfaches Beispiel als JPQL:

```
SELECT e FROM Employee e
WHERE e.name = 'John Smith'
```

- … als Criteria API:

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Employee> c =
    cb.createQuery(Employee.class);
Root<Employee> emp = c.from(Employee.class);
c.select(emp).where(cb.equal(emp.get("name"),
"John Smith"));
```

Criteria API

Analogien wo möglich:

Klausel-Methoden:

JP QL Clause	Criteria API Interface	Method
SELECT	CriteriaQuery	`select()`
	Subquery	`select()`
FROM	AbstractQuery	`from()`
WHERE	AbstractQuery	`where()`
ORDER BY	CriteriaQuery	`orderBy()`
GROUP BY	AbstractQuery	`groupBy()`
HAVING	AbstractQuery	`having()`

JPQL to CriteriaBuilder Predicate Mapping:

JP QL Operator	CriteriaBuilder Method
AND	`and()`
OR	`or()`
NOT	`not()`
=	`equal()`
<>	`notEqual()`
>	`greaterThan(), gt()`
>=	`greaterThanOrEqualTo(`
...	...

Criteria API

- Bisher: String Based-API; Besser:
  - JPA 2.0 Metamodel API, insbes. Canonical Metam.
    - enthält Klassen (typischerweise generiert, eine pro persistente Klasse), mit statischen Deklarationen der Metamodellobjekte.
  - Beispiel:

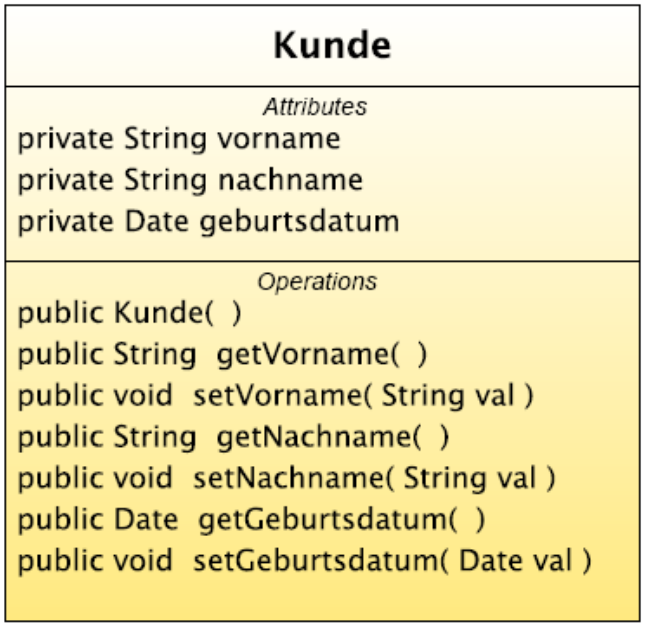
```
CriteriaQuery<Object> c = cb.createQuery();
Root<Employee> emp = c.from(Employee.class);
MapJoin<Employee,String,Phone> phone =
    emp.join(Employee_.phones);
c.multiselect(emp.get(Employee_.name),
phone.key(), phone.value());
```

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Mapping einfacher Klassen mit Annotationen

Beispiel Kunde



- POJO = JavaBean
- also Default-Konstruktor
- Getter und Setter
- Problem: Java-Identität und DB-Identität sind etwas anderes

Mapping einfacher Klassen mit Annotationen

- Vorteil
  - Alles an einer Stelle

- Nachteile:
  - geht nur mit Java 5 aufwärts
  - mehr Java-Code
  - man sieht es der Klasse an
  - Bindet Code an DB
  - Favorisiert Forward Engineering und schliesst die anderen praktisch aus

Unterschied zu XML-Mapping-Datei

- Angabe, welche Klasse persistent ist.
  - Durch die @Entity-Annotation, nicht durch Mapping
- Mindestangabe des Primärschlüssels.

  => @Id-Annotation
- Defaults:
  - alle Properties der Klasse, die nicht mit @Transient annotiert sind
  - Tabelle wie Klasse
  - Spalten wie Properties

Beispiel Kunde

```
@Entity
public class Kunde implements Serializable {
  private Integer id;
  private String vorname;
  private String nachname;
  private Date geburtsdatum;
  public Kunde () {}
  @Id
  @GeneratedValue ( strategy = GenerationType.IDENTITY )
  public Integer getId () { return this.id; }
  public void setId ( Integer id) { this.id = id; }
  public String getVorname () { return this.vorname ; }
  public void setVorname ( String vorname ) {
    this.vorname = vorname;
  ...
```

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Objekt-Relationale Abbildung mit JPA

Assoziationen

- Assoziationen mit XML-Mapping
  - Assoziationen werden wie ganz normale Properties behandelt
  - d.h. Instanzvariable plus Getter und Setter
  - entsprechend ihrer Kardinalität (Objekt oder Collection)
  - die Collection-Art in Java muss zur Collection-Art im Mapping passen
- Assoziationen mit Annotationen
  - es müssen nur die Properties für die Assoziationen mit den entsprechenden Annotationen versehen werden
  - Nachteil: es kann dazu kommen, dass dies relativ unübersichtlich wird
  - Fremdschlüssel-Referenzen
    - @OneToOne, @ManyToOne, @OneToMany, @ManyToMany

Vererbung

- „Einfache Klassen“ => für Persistenz: default eine Tabelle pro Klasse
- In Realität wegen Vererbung etwas komplizierter
- Vererbung kann auf drei verschiedene Weisen gemapped werden:

```
(Inheritance
  (strategy=InheritanceType
     SINGLE_TABLE
     TABLE_PER_CLASS
     JOINED))
```

Vererbung (als Annotation)

- Superklasse:

```
@MappedSuperclass
public abstract class Konto implements
  Serializable {
private Integer id;
private BigDecimal kontostand ;
...
```

Weitere O/R-Abbildungen mit JPA

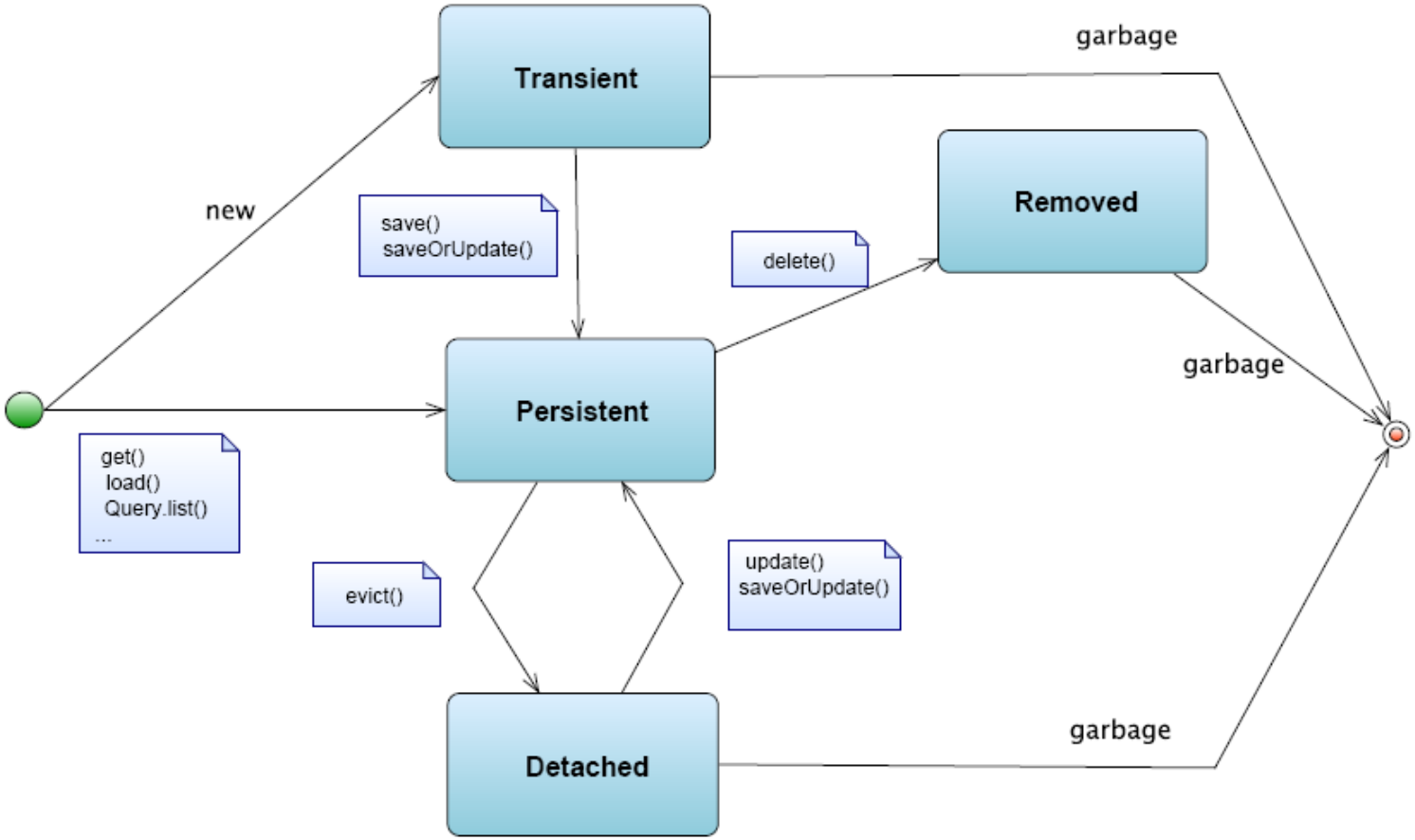
- Beliebig viele Tabellen können zusammen auf eine Klasse abgebildet werden
  - @SecondaryTable
- Klassen können zusammen auf eine Tabelle abgebildet werden
  - @Embedded

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Lebenszyklus

Lebenszyklus



Lebenszyklus

- Typischerweise realisiert mit Callbacks und Listeners

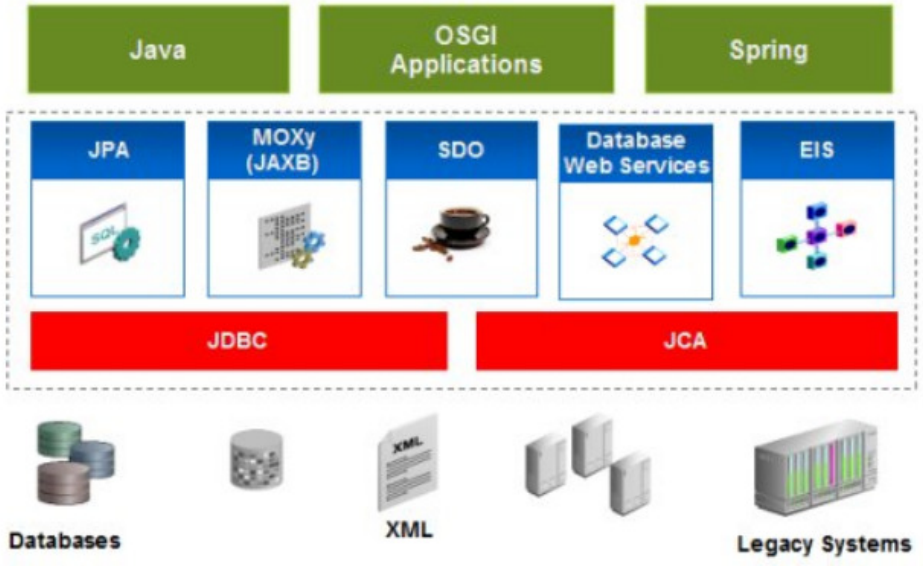
- Transient
  - Nach Aufruf des new-Operators ist ein Objekt zunächst transient. Es ist nicht mit einer Tabellenzeile verbunden. Ein transientes Objekt kann durch den Aufruf eines Persistenzmanagers persistent werden oder indem es von einem persisteten Objekt referenziert wird.
- Persistent
  - Ein persistentes Objekt hat eine Datenbankidentität (PK). Jedes persistente Objekt hat einen

Weitere Themen

- Data Type Mapping:
  - Defaults für primitive Typen
- Primärschlüssel:
  - Generierung und zusammengesetzte Schlüssel
- Beziehungen zwischen Entitäten:
  - Cascading, Lazy-Loading
- Transaktionen
- Tools

EclipseLink (aus Übungen)

- JPA-Referenz; bewährt - trotz Version 1.x
- Mehr als ORM zu RDMS,
  - auch XML und JCA als Datenspeicher nutzbar
  - Unterstützt neben JPA auch JAXB und SDO



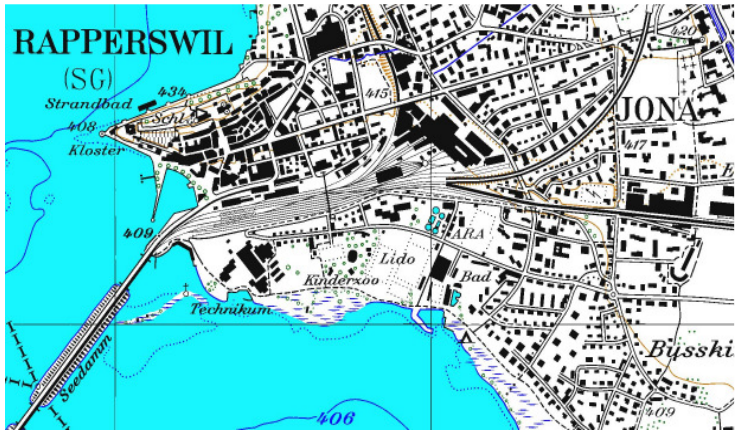
Quelle: Persistenz mit EclipseLink, Stefan Scheidt, JAXenter, 2009

Zusammenfassung

- „Manifesto“:
  - Objekte, nicht Tabellen.
  - ORM ist für Wissende; nicht um Mapping-Probleme zu verdrängen oder verstecken
  - Unobtrusive, not transparent: Eine Applikation will die Kontrolle von Objekt-Lebenszyklen behalten.
  - Die meisten Applikationen finden bereits eine RDBMS vor, sie erzeugen keine eigene, neue.
- …

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK



Quelle: (c) L+T, LK 25 Nr. 1112 Kombi.

GIS (1)

Datenbanksysteme 2

Prof. Stefan Keller

Frühjahrssemester 2010

Inhaltsüberblick

- Vision Multimedia-Datenbanken
- Geometrie-Datentypen
- Geo-Informationssysteme (GIS)
- Besonderheiten von Geodaten

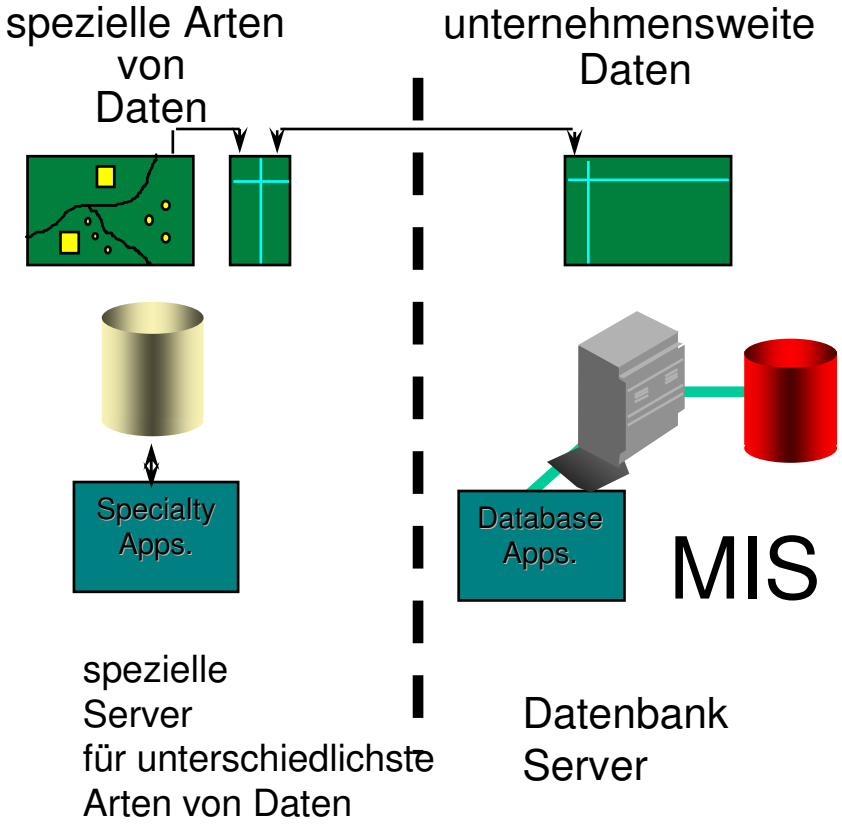
Multimedia-Datenbanken

- Medien-Daten und -Operationen werden in vielen Anwendungen eingesetzt
- ORDBMS sind für die Definition von Medienobjekten bereit
- Packages erleichtern die Verwaltung (Installation, Upgrade, Entfernen) und die Wiederverwendung (ein Package kann ein anderes benutzen)
- Genau das Richtige für SQL:99!
- Oracle's Vision dazu…

Oracle's Vision: Das Problem…

Über 90% multimedialer Daten werden nicht (zentral) innerhalb von DBs und proprietär gespeichert…

- eigene Server für die unterschiedlichen Datenarten
- die Daten sind isoliert
- hohe Administrations- und Wartungskosten
- hohe Schulungskosten
- komplexe Anwendungsentwicklung



Nutzen und Vorteile einer zentralen Multimedia-DB (laut Oracle)

Nutzen

- Integration in ein Multimedia- oder Internetkonzept
- Konsolidierung in einer einzigen Datenbank
- Verwaltung grosser Datenmengen
- Performanz
- Sicherheit und Zugriffsschutz
- ‘Plattformunabhängigkeit’

Vorteile

- Einheitliches Content Management
- Skalierbare, indizierte Suche von Medieninhalten
- Ein Server zu installieren und zu warten
- Zugriff von jedem Client auf alle Inhalte

Die Multimedia-Norm SQL/MM

- gehört zum SQL-Standard, ist aber eigenständig
  - SQL: ISO/IEC 9075, SQL/MM: ISO/IEC 13249
- besteht aus mehreren Teilen
  - Teil 1: SQL/MM Framework (Nov. 2002)
  - Teil 2: SQL/MM Full Text (Okt. 2000)
  - Teil 3: SQL/MM Spatial (Dez. 1999)
  - Teil 5: SQL/MM Still Image (Mai 2001)
- Heute von Bedeutung sind Standards/Normen von
  - Open Geospatial Consortium (OGC)=> v.a. Firmen
  - ISO 19100 Geoinformation=> Normenverein

Geo-Informationen?

- 80% unserer Informationen - und damit unserer Informationssysteme mit den darauf aufbauenden Entscheidungen - haben einen Raumbezug!

- Viele Daten haben einen Raumbezug:
  - Points of Interest
  - Verkaufsstandorte, Immobilien
  - Personen, Kunden

Geo-Informationssysteme (GIS)

- Bekannte GIS
  - Navigationssysteme (‘GPS’)
  - Google Maps/Earth, OpenStreetMap…
- Kennen Sie weitere?
  - …
- Hinweis: GIS = raumbezogenes System
  Raumbezug?
  - Koordinaten
  - Geonamen, Adresse ()
  - Streckennummern

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Geometrie-Datentypen

Geometrie-Datentypen

- Geometrie oder raumbezogenes Attribut
- Semantik eines Geometrieattributs (ADT) wird durch Datentyp bestimmt:
  - Punkt
  - Linie (genauer: Polylinie, LineString)
  - Fläche (‘Polygon’)
- Schon beim Punkt-Datentyp: lat/lon oder lon/lat? Evtl. Z-Koordinaten (=2.5D)
- Linien noch anspruchsvoller: Eine geordnete Folge von Stützpunkten, die durch gerade Linien oder Bögen miteinander verbunden sind…

Aufgabe

- Definieren Sie den Geometrietyp zweidimensionale Linie…

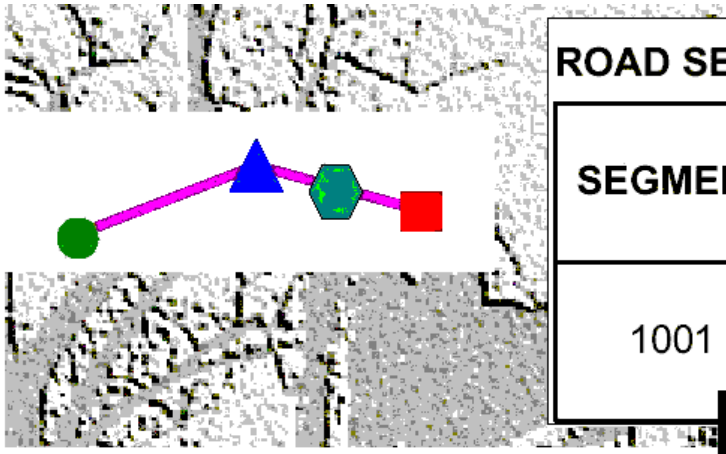
Aufgabe - Lösung Linie

- Linienzug!
- Relationale Lösung:
  - vordefinierte Anzahl von Spalten des Typs NUMBER und eine oder mehrere Zeilen pro Objekt
- Objekt-relationale Lösung:
  - ADT mit vordefinierten Datentypen und Methoden

Realisierung Geometrie-Datentyp

- Sequentielle Dateien
- Normalisierte Tabellen
- Binary Large Objekts (BLOB)
- und (schliesslich) ADT/UDT (gemäss SQL:99)
  - Binary Large Objekts (CLOB/BLOB) gekapselt über
    - Konstruktoren
    - Funktionen
    - REF-Typen

Normalisierte Tabellen



ROAD SEGMENT

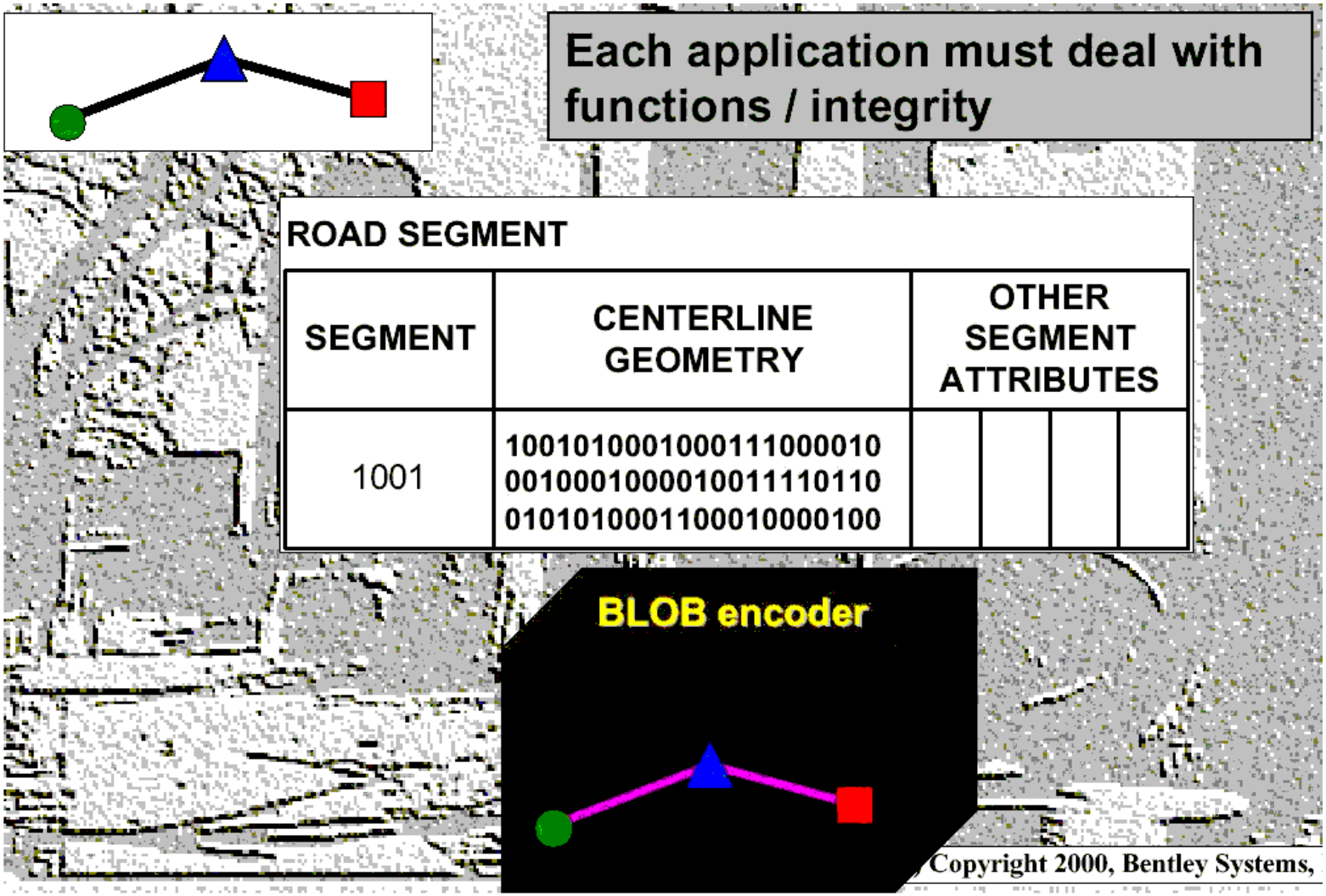
SEGMENT	CENTERLINE GEOMETRY	OTHER SEGMENT ATTRIBUTES			
1001					

Each application must deal with functions / integrity

VERTEX

LINE	VERTEX SEQUENCE	POINT	POINT	X	Y	Z
	1					
	2					
	3					

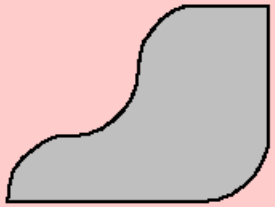
BLOBs


Each application must deal with functions / integrity
ROAD SEGMENT
SEGMENT
CENTERLINE GEOMETRY
OTHER SEGMENT ATTRIBUTES
1001
1001010001000111000010
0010001000010011110110
0101010001100010000100
BLOB encoder
Copyright 2000, Bentley Systems,

ADT mit SQL:99, bzw. SQL/MM

Beispiel:

`CITY (NAME, POPULATION, CITY PARKS, LOCATION)`

NAME varchar(30)	POPULATION integer	CITY PARKS varchar(30) array[10]	LOCATION st_geometry
My Little Town	1042	['Lincoln Park']	
The Big City	1450000	['City Park', 'Grant Park', 'Castle Park', 'Forest Park']	
A Big Town	4900	[ ]	

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Geo-Informationssysteme (GIS)

Überblick

- Geometriedatentypen
  - Datentypen (vordefinierte ADT's)
  - Metadaten: Koordinatensysteme etc.
  - Methoden (Funktionen): u.a. Spatial Join
- Geo-Informationssysteme
  - Karten und Pläne
  - Raumbezug - der neue Beziehungstyp
  - Was sind CAD, GIS…? Eigenschaften
  - Von Geodaten zur Grafik

Ziele

- Eigenheiten und Bedeutung der Geo-Informationssysteme (Geodaten) für die Informationstechnologie kennenlernen
- Geo-Datenbanken als konkrete Anwendung
  - von Multimedia-Daten
    - realisiert durch objekt-relationalen Erweiterungen (ADTs)
  - benutzerdef. (mehrdim.) Index
    - Generalized Search Tree (GIST): Template-Struktur! Ein Balanced tree ähnlich B-tree mit <key, pointer>-Paaren als UDT mit vier Methoden, die überschrieben werden

Wozu Karten und Geodaten?

Zur Orientierung und Navigation:

- Wo liegt der nächste Bankomat? Wie komme ich nach Rappi?
- Wieviel Lärm verursacht Flughafen X; wieviel Land besitzt B?

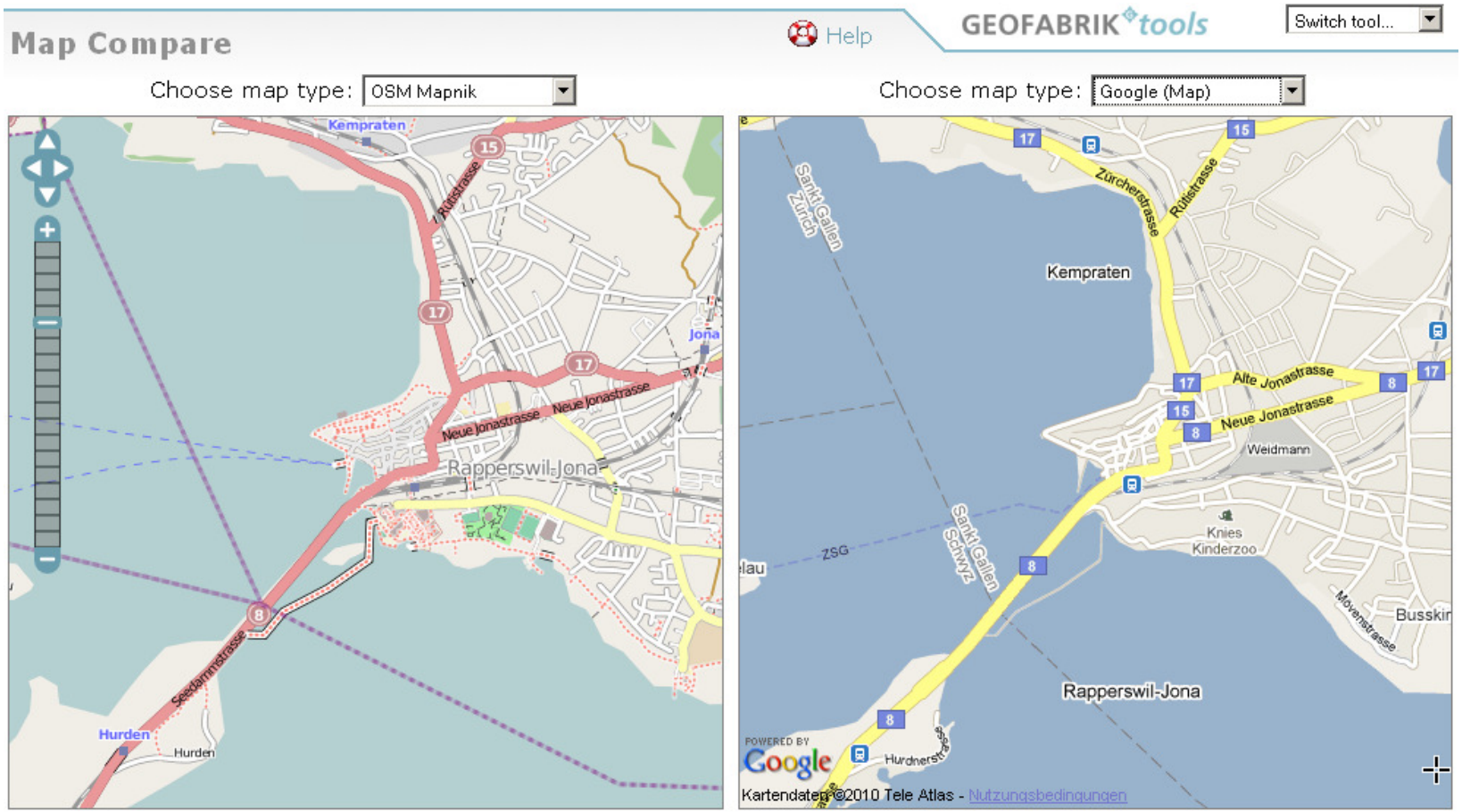
Zur Planung und Entscheidungsfindung:

- Welche Freunde wohnen im Umkreis von 50 km von hier?
- Wieviele Einwohner sind vom Lärm einer Flughafen-Landebahn betroffen?

Zur Verwaltung

- Wo gibt es Strassen oder Wasserleitungen
- Welche sind älter als 50 Jahre?

Google Maps vs. OpenStreetMap

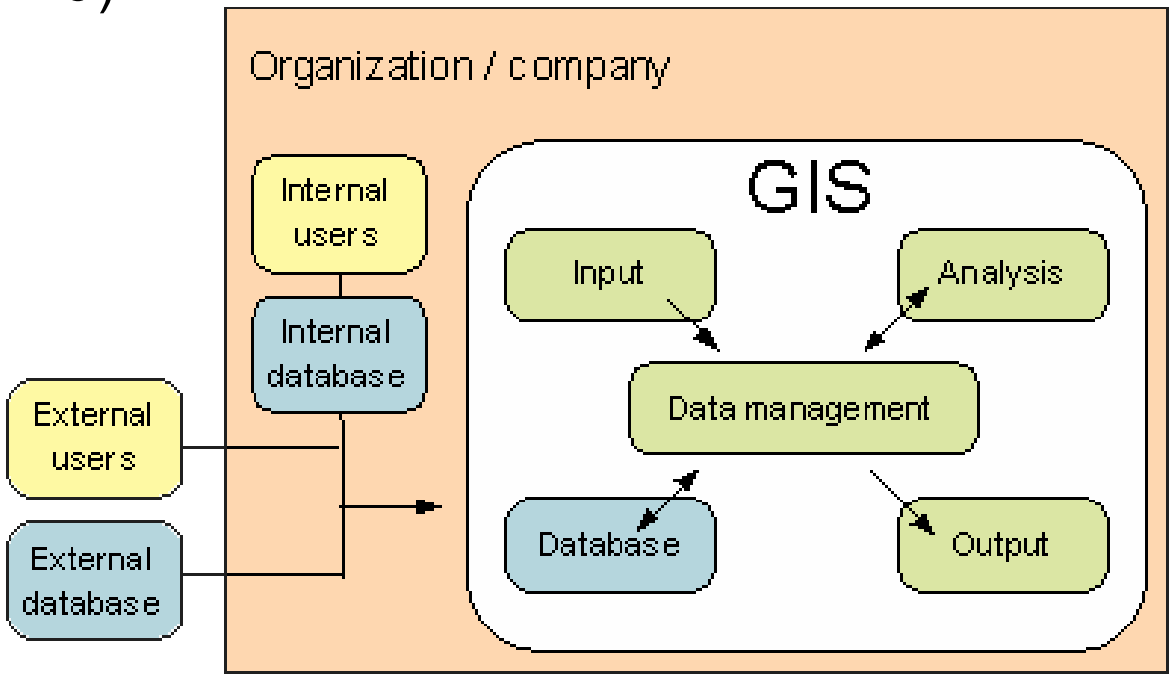


Was ist GIS?

Def. Geographisches Informationssystem (GIS): Informationssystem zur Erfassung, Verwaltung, Verarbeitung, Analyse und Visualisierung von Geodaten. (Goodchild)

Abk.: LIS, UIS, NIS, KIS, etc...

Aufteilung in Basissystem ("Werkzeugkasten") und Fachschale

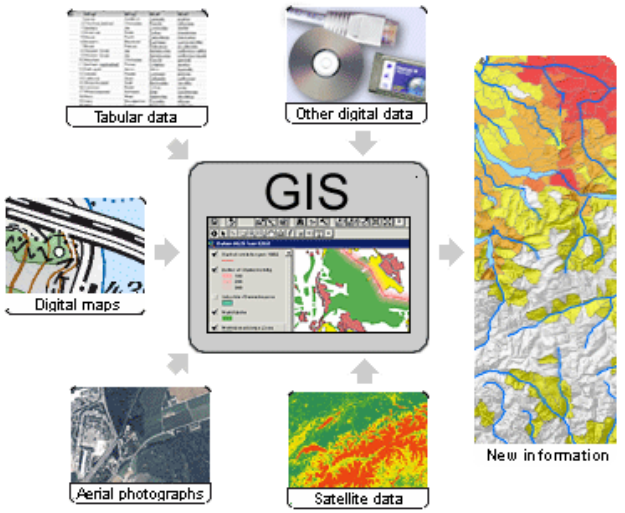


Quelle: www.gitta.info

Was ist GIS?

besonders (gegenüber Informationssystem)?

- Ausgabe (Präsentation, Visualisierung) als Karte, Grafik, Tabelle
- Datentypen:
  - Koordinate (ev. Höhe);
  - Linie und Fläche;
  - georeferenzierte Rasterdaten
- grosse Datenmengen
- grosser Erfassungsaufwand
- komplexe Konsistenzbedingungen



Quelle: www.gitta.info

Anwendung/Nutzen von GIS

Stadtplan, Wahlabstimmungskarte, Wetterkarte, Tourismus (Bahnen), Karten für Spezialisten (oft der öffentlichen Verwaltung): Grundbuch-Kataster, Leitungen, Natur- und Umweltschutz, etc.

Diese herzustellen lohnt sich nur, wenn deren zugrundeliegende Geodaten langfristig und konsistent verwaltet werden.

Langfristig bedeutet für Geodaten: 10 bis 50 Jahre!

Diese lassen sich zudem dank Koordinaten mit anderen Informationen kombinieren (räuml. Join).

Kartendarstellung als Archiv, Visualisierung (Simulation) und als Analysewerkzeug.

Karten und Pläne

Seit über 1000 Jahren bedienen wir uns nicht nur der Schrift sondern auch Karten und Pläne:

- "Ein Bild sagt mehr als 1000 Worte…"

Karten und Pläne vermitteln ein abstrahiertes (=modelliertes) Bild unserer Welt:

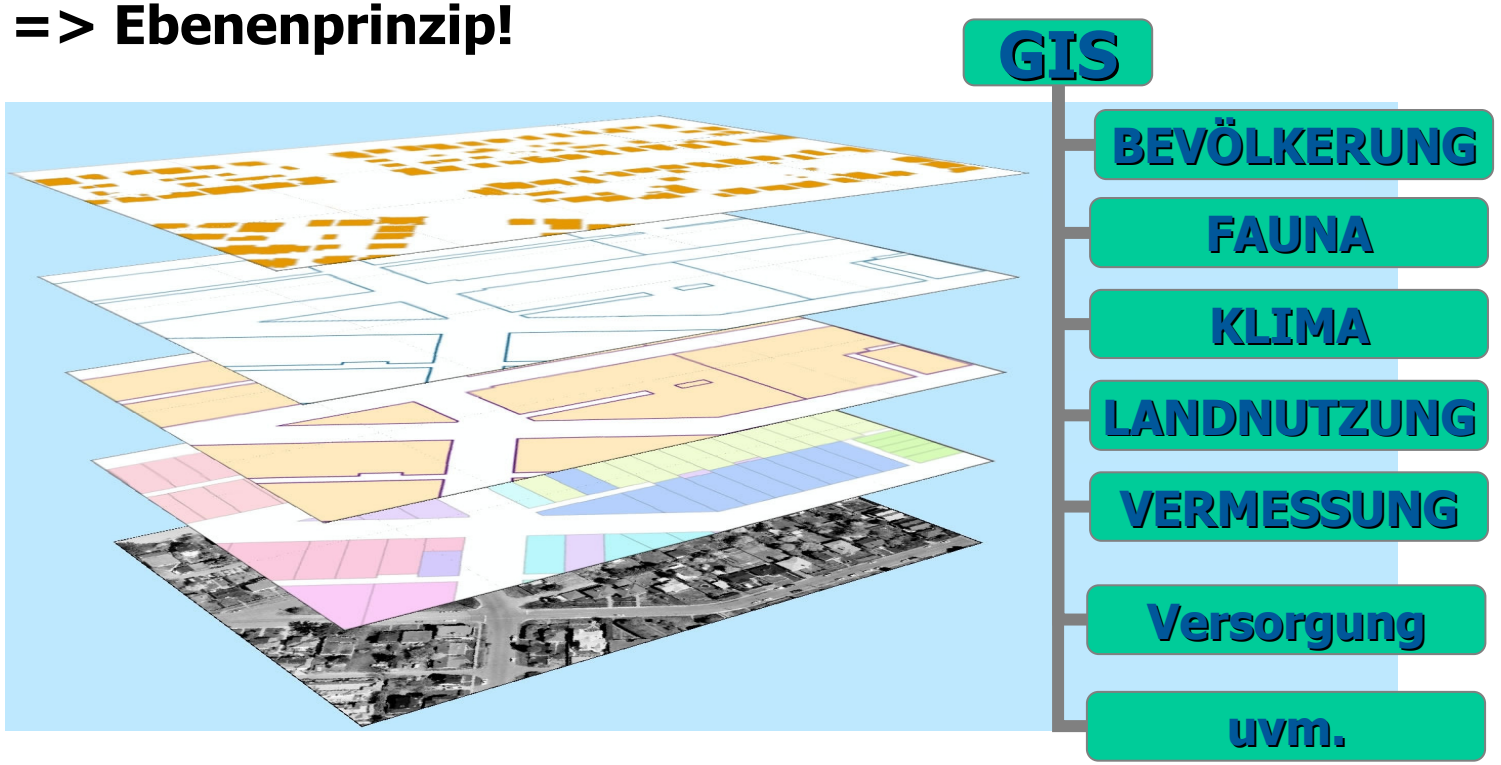
- sichtbare Objekte wie Gebäude, Strassen, Böschungen oder Restaurants
- darauf aufbauend Verwaltungseinheiten, Stromleitungen, Mobilfunkzellen, etc.

(englisch für beides: Map, bzw. Mapping)

Ebenenprinzip

Koordinaten ermöglichen Lagevergleich (vertikal).

Stärke von GIS: Kombination (unabhängiger) Daten und daraus neue Informationen abzuleiten
=> Ebenenprinzip!



Raumbezug - der andere Beziehungstyp

Der *Raumbezug*, ist der gemeinsame Nenner aller raumbezogenen Themen.

Raumbezug heisst: Für jedes Objekt gibt es ein *Koordinatensystem* (Referenzsystem).

Koordinaten stellen einen *grundsätzlich neuen Beziehungstyp* dar, der das Kombinieren von Themen erlaubt (räuml. Join)

Neue Zusammenhänge erkennen durch Visualisierung (Simulation) und Analyse

Geogr. Modellierung u. Repräsentation

Zwei (Modell-)Sichten auf geogr. Phänomene:

- Diskrete Entitäten-/Objekte-Sicht
  - Räumlich begrenzt, z.B. ein Gebäude
- Kontinuierliche Felder-Sicht
  - Z.B. Luftdruck, Vegetationsgebiete

Zwei Repräsentations-/Codierungsarten

- Vektordaten
- Rasterdaten

GIS-"Datenwelten"

- Raster: Raster-, Grid oder Zelldaten
  - zusammengehörige Menge von (Farb-)Pixeln, die einen Raumbezug haben (sie sind sog. "geo-referenziert"). Typischerweise eingescannt oder von CCD-Sensoren (Flugbilder); z.B. GeoTIFF-Format
- Vektor:
  - Geometrische Datentypen Punktkoordinaten, Linien und Flächen.
- Vergleich:
  - Rasterdaten: schneller in der Darstellung, billiger in der Herstellung als Vektordaten;
  - Vektordaten: einzeln ansprechbar, gruppierbar, skalierbar und berechenbarer.

Implementation Geoobjekte

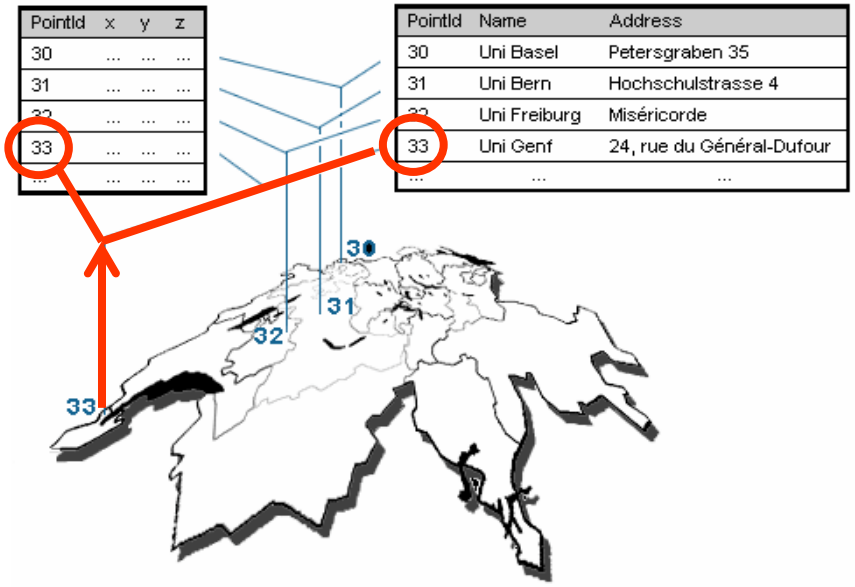
Implementation Geoobjekte mit Vektordaten:

- 'Geometrie'
- Sachdaten (Attributdaten, Thematische Daten)

Typische Einschränkung Geodaten-Tabelle:

- Tabelle mit genau einem Geometrie-Attribut und beliebig vielen weiteren nicht-geom. Attr.

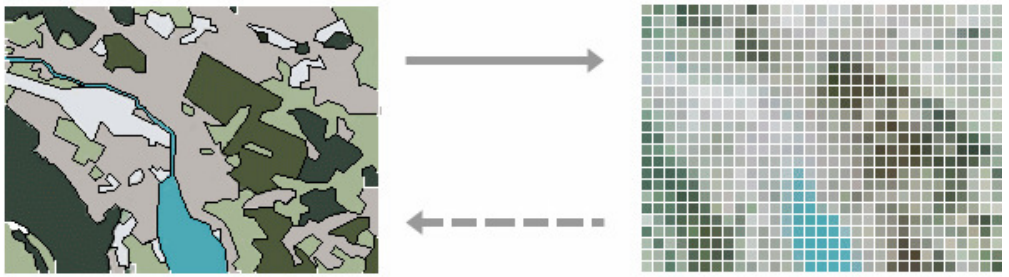
Synonym: Feature Class



Etwas vereinfacht aber häufig verwendet (Quelle: www.gitta.info)

Datentypen: Man unterscheidet...

- Geometriedaten: Vektor- und Rasterdaten



Vektordaten ('gerendert')

Rasterdaten (pro Pixel ein (Farb-)Wert)

Luftbild (Quelle: www.gitta.info)

- Vektordaten:
  - Geometrische Datentypen (Koordinaten) und topologische Datentypen
  - Topologie = Nachbarschaft, z.B. wichtig für Kürzestwege-Berechnung, Navigation

Datentypen: Vektorgeometrie

- GEOMETRY (abstrakter Obertyp)
  - POINT, LINESTRING, POLYGON,
  - MULTIPOINT, MULTILINESTRING, MULTIPOLYGON
- etc.
  - GEOMETRYCOLLECTION, CURVESTRING, CURVEPOLYGON, COMPOUNDCURVE
- Gemäss Standard von ISO 19100, OpenGIS Consortium (und ISO SQL/MM)

GIS-Methoden/Funktionen (1)

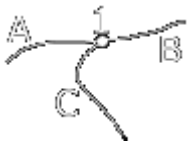
Analyse, Geoprocessing

Geographische Lage zusammen mit topologischen Nachbarschaftsbeziehungen erlauben versch. Ab-/ Anfragen:

1. Benachbart?
2. Enthalten?
3. Verbunden?

(1)

(2)



(3)

GIS-Methoden/Funktionen (2)

Auszug:

- Räumliche Anfragen: Punkt in Polygon, Alle Punkte im Umkreis, etc.
- Flächenverschneidung mit Booleschen Operatoren im Rastermodell und im Vektormodell
- Zoom/Pan ist Standard; aber Achtung: Zoom ist nicht gleich Skalieren!
- Geometrischer Puffer

GIS-Methoden/Funktionen(3): Verschnitt

Verschnitt-Methoden von 2 Objekten mit Geometrieattributen…

(Quelle: A. Lienhard)

Synonym:

Overlay

	Punkte	Linien	Flächen
Punkte	Nächste	Nächste	Innerhalb
Linien	Nächste	Teil von	Innerhalb
Flächen	---	---	Innerhalb

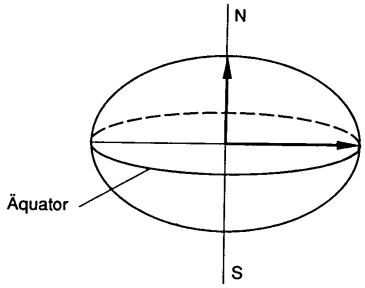
Besonderheiten von Geodaten

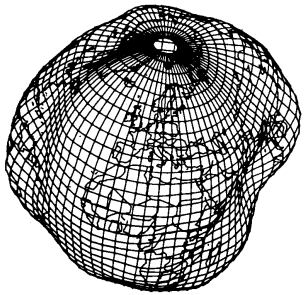
- (Konsistenz und grosse Datenmengen)
- Koordinatensysteme
- Metadaten
- Grafik

Raumbezug etwas Genauer

Koordinatensysteme oder Referenzsysteme:

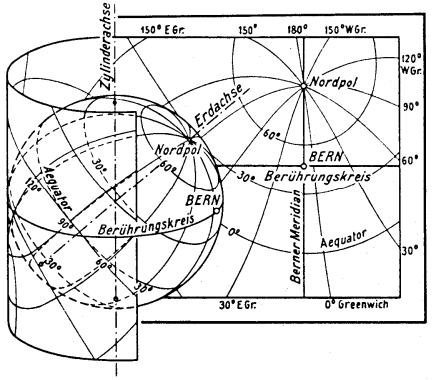
- Die Erdoberfläche ist eine Kugel (Ptolemäus / Galilei);
- besser ist eine Annäherung mit Ellipsoid...
- und mit Schwerefelder geht es noch genauer:





Koordinatensysteme (1)

Karten und Bildschirme sind aber "eben": Eine Projektion bildet das Ellipsoid auf eine Fläche ab, d.h. auf kartesische Koordinaten: x und y, die rechtwinklig aufeinander stehen.



Meist bezeichnet man Koordinatensystem und Projektion mit einem Wort, z.B. CHLV03 für Schweizer Landes.-K. (1903) oder WGS84 für weltweites GPS.

Koordinatensysteme (2)

Bekannte Koordinatensysteme sind

- WGS84: ein geografisches K. mit Länge, Breite:

```
WGS84Coord = COORD  !! Datentyp (ADT) Koordinaten(-punkt)
    -90:00:00 ..  90:00:00          [Angle_DMS] {WGS84/1},
      0:00:00 .. 359:59:59 CIRCULAR [Angle_DMS] {WGS84/2};
```

- CHLV03: das Landesvermessungs-Koordinatensys. der Schweiz: Nord- + West-Werte x/y und z = Höhe):

```
CHCoord: COORD  !! Datentyp (ADT) Koordinaten(-punkt)
    480000    .. 850000.00 [m] {LandesSys/1}, !! Achse 1
     60000    .. 320000.00 [m] {LandesSys/2}, !! Achse 2
      -200.00 ..   9000.00 [m] {LandesSysH/1};!! Höhe
```

Für uns genügt reine Text-Angabe zum Attribut; die Umrechnung erfolgt mit Spezial-SW, im GIS oder in DB.

Metadaten allgemein

Metadaten: bedeutet vieles…

Definition: (normierte) Daten über Daten:

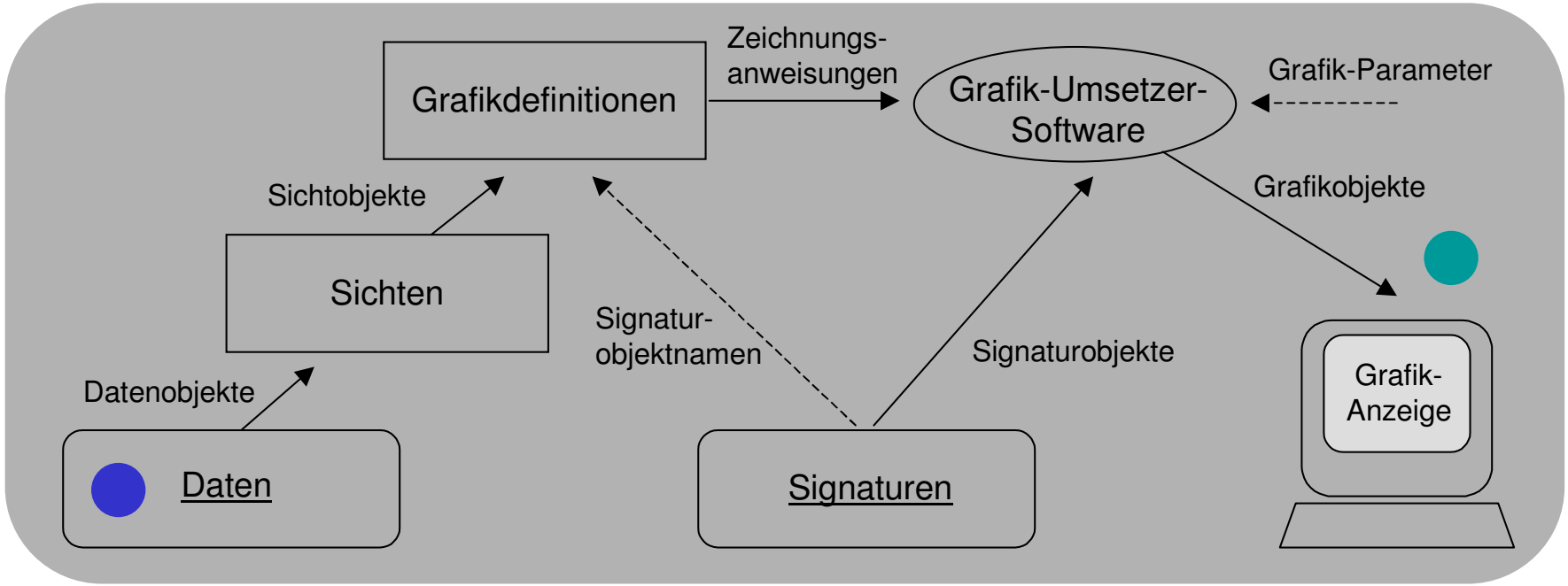
- Bei Sprachen (z.B. Java):
  - Informationen über Klassen, etc.
- Bei Informationssystemen:
  - (Daten-)Schema
  - weitere Eigenschaften von Datentypen (geordnet)
  - und deren Masseinheiten (Meter, Temp., Winkel...)
  - Beispiele:

```
Skipiste: (blau, rot, schwarz: FINAL) ORDERED;
Richtungswinkel: 0.0 .. 359.9 CIRCULAR [Altgrad];
```

Geo-Metadaten

- Bei Geo-Informationssystemen:
  - Weitere Eigenschaften der (komplexen) Datentypen: Koordinatensysteme (siehe gleich anschliessend)!
  - Beschreibung der Geodaten inkl. Verweis auf Schema als Metadaten (!): übliche Produktangaben
    - zur Qualität (Herkunft)
    - zum Preis
    - und zur Bezugsquelle.

Von (Vektor-)Geodaten zur Grafik (1)



"CAD" (die meisten GIS noch heute)

"GIS" heute und morgen: Geo-Visualisierung

Von Geodaten zur Grafik (2)

Beispiel Autobahn
Geodaten-Objekt:

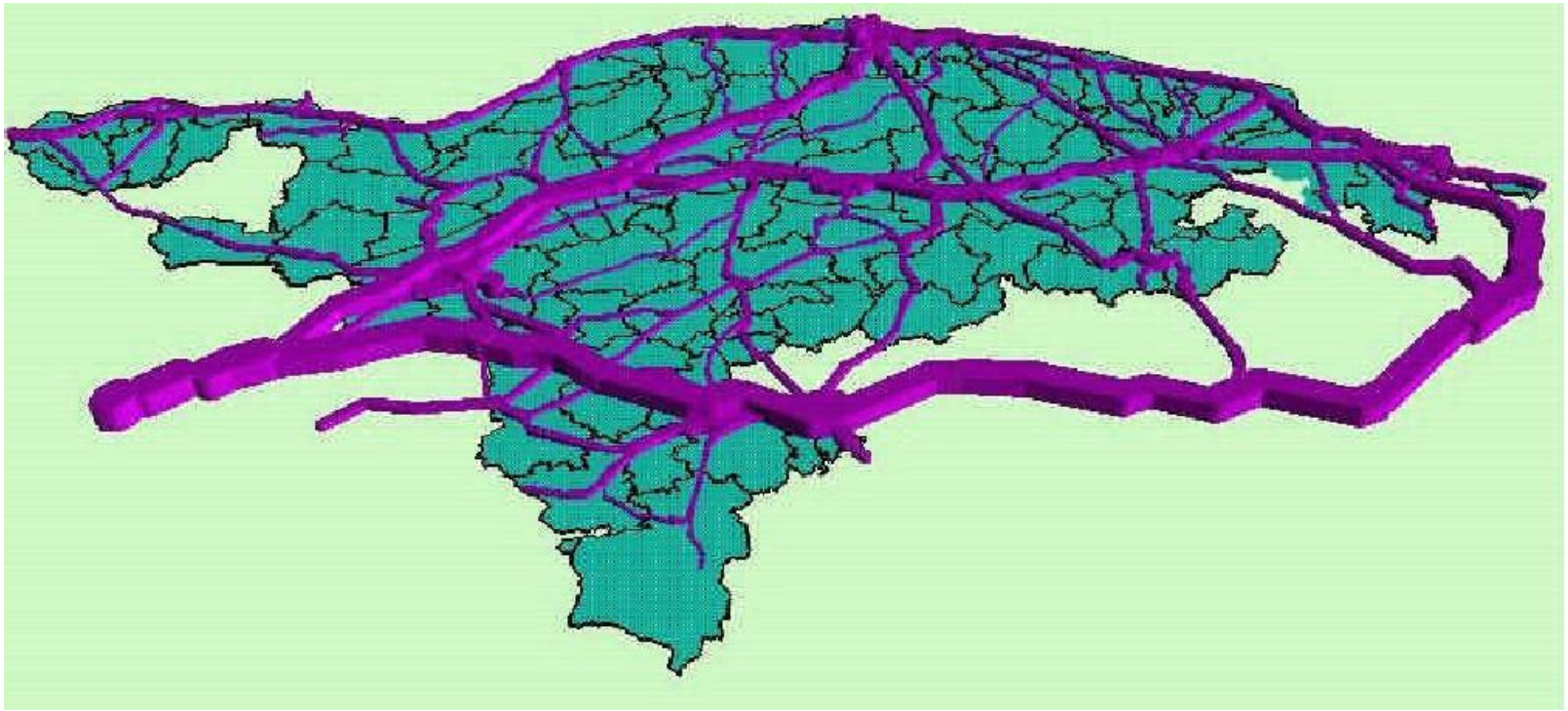
- Instanz der Klasse Autobahn
  - Linien-Geometrie: x/y, x/y, x/y, …
  - Bezeichnung: N1

Grafikobjekte:

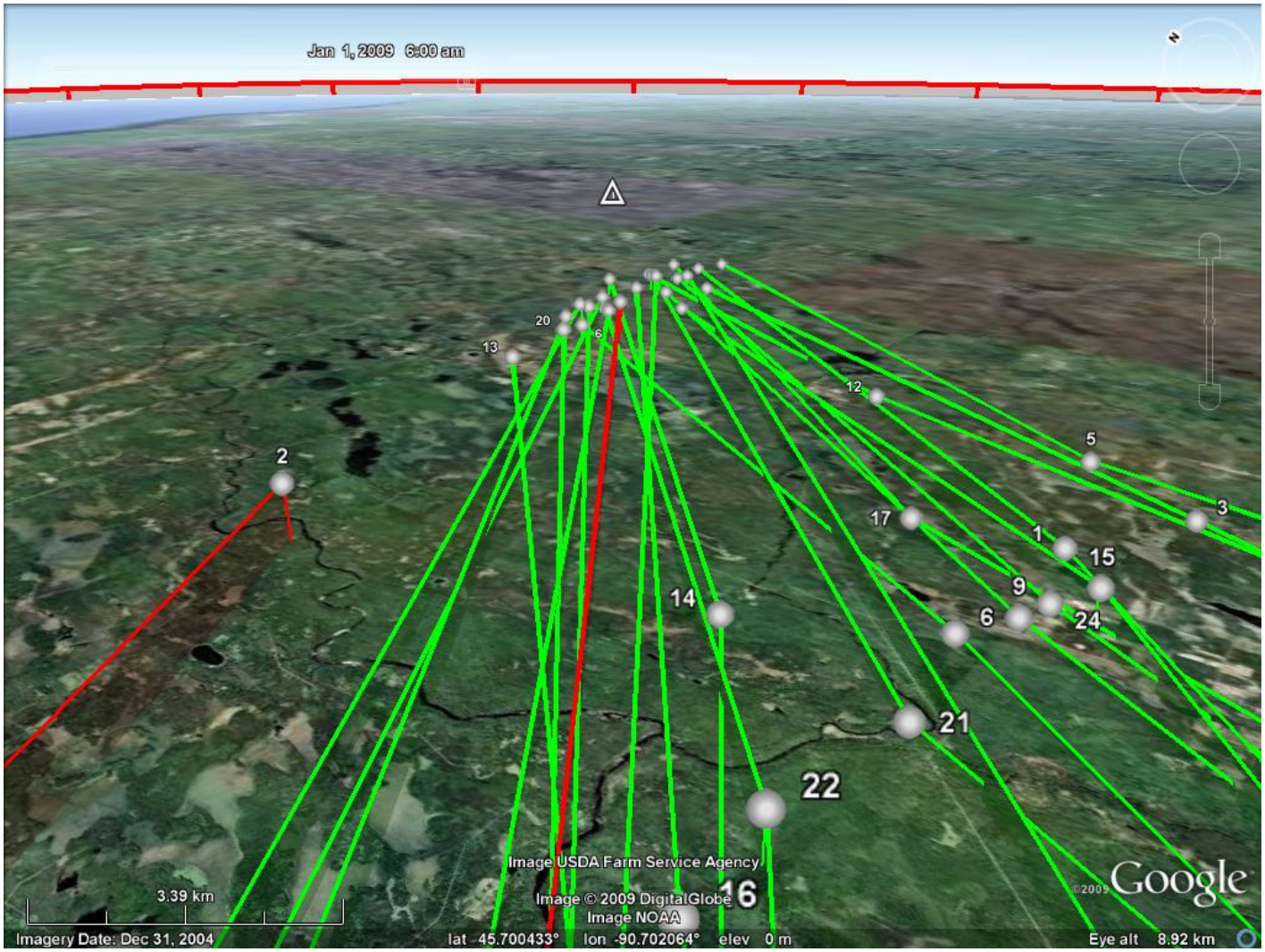
- Graf. Linienelement
  - x/y
  - Breite: 3mm
  - Farbe: Rot
- Beschriftungselement
  - Text: "N1"
  - Koordinate: x/y
  - Schriftart Helvetica
  - darüberliegend

Analyse und Visualisierung

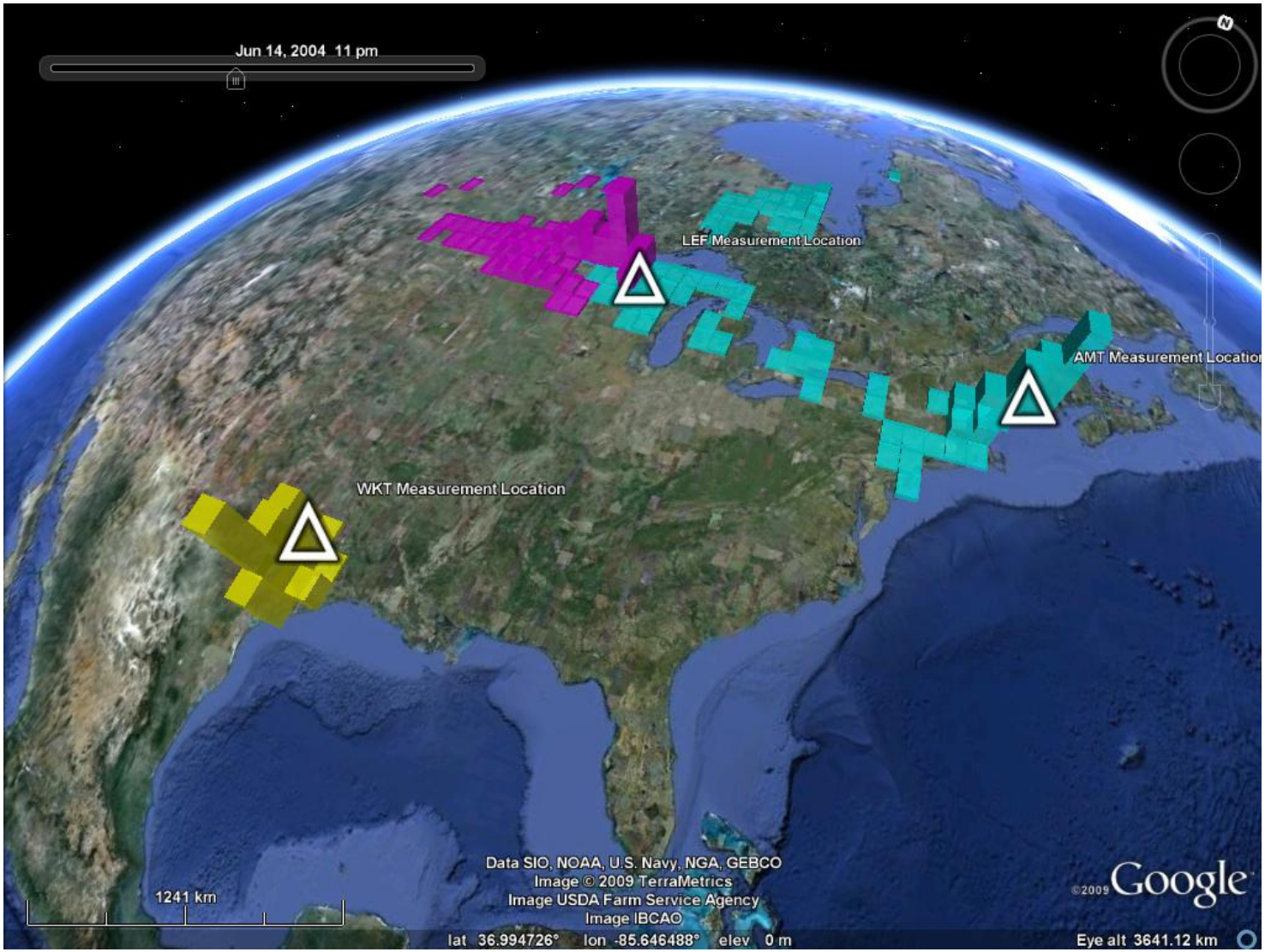
Visualisierung von Fahrzeugabgasvolumen entlang der Verkehrswege im Kanton Thurgau. (Quelle: Thomas Klink, NDK GIS/HSR)



Visualisierung GE: Particle Tracks

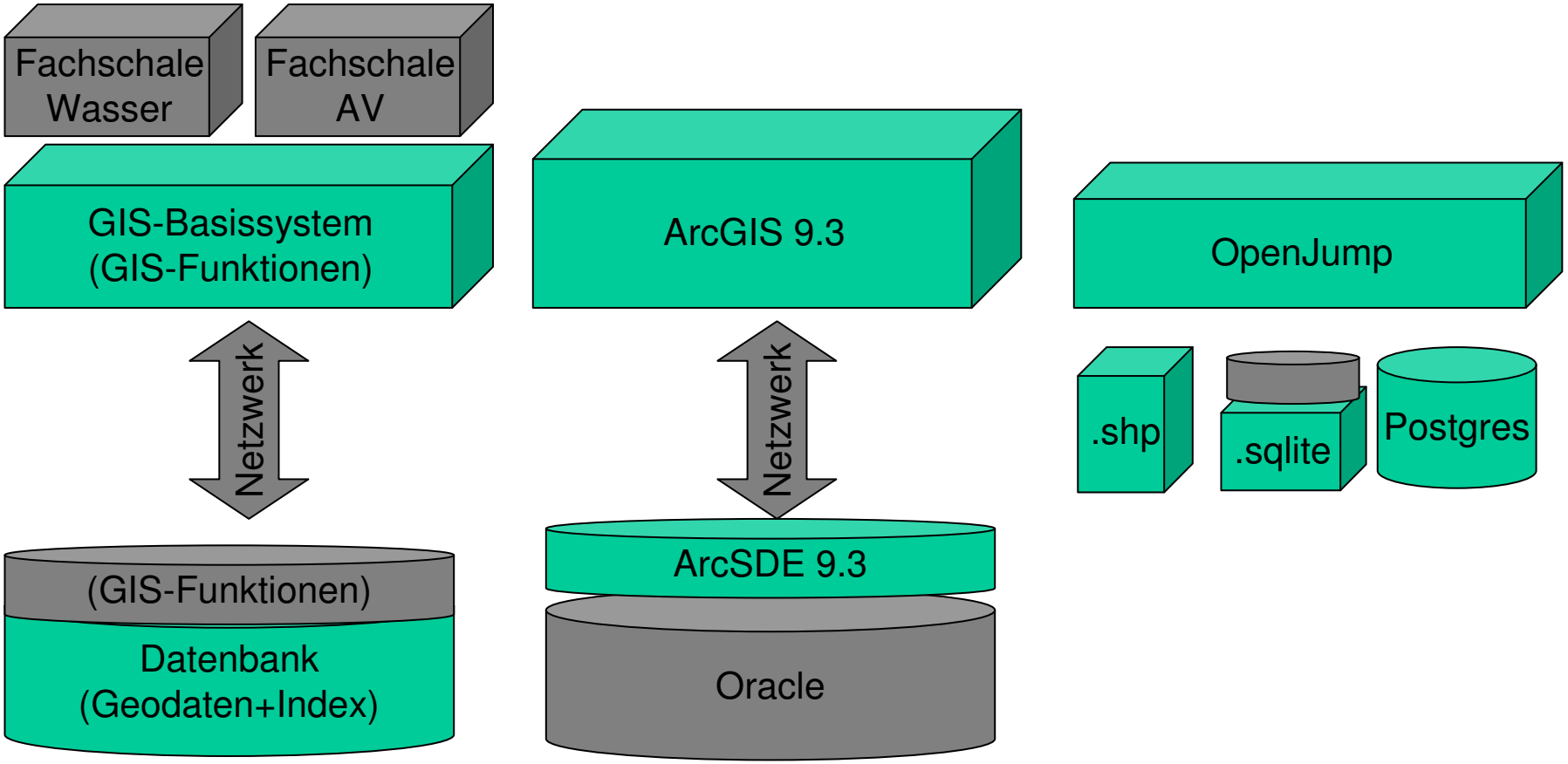


GE: Sensitivity to Surface Fluxes



Systemarchitekturen von (Desktop-)GIS

Theorie | Praxis mit ESRI | In den Übungen



Verteilte GIS, Web-GIS

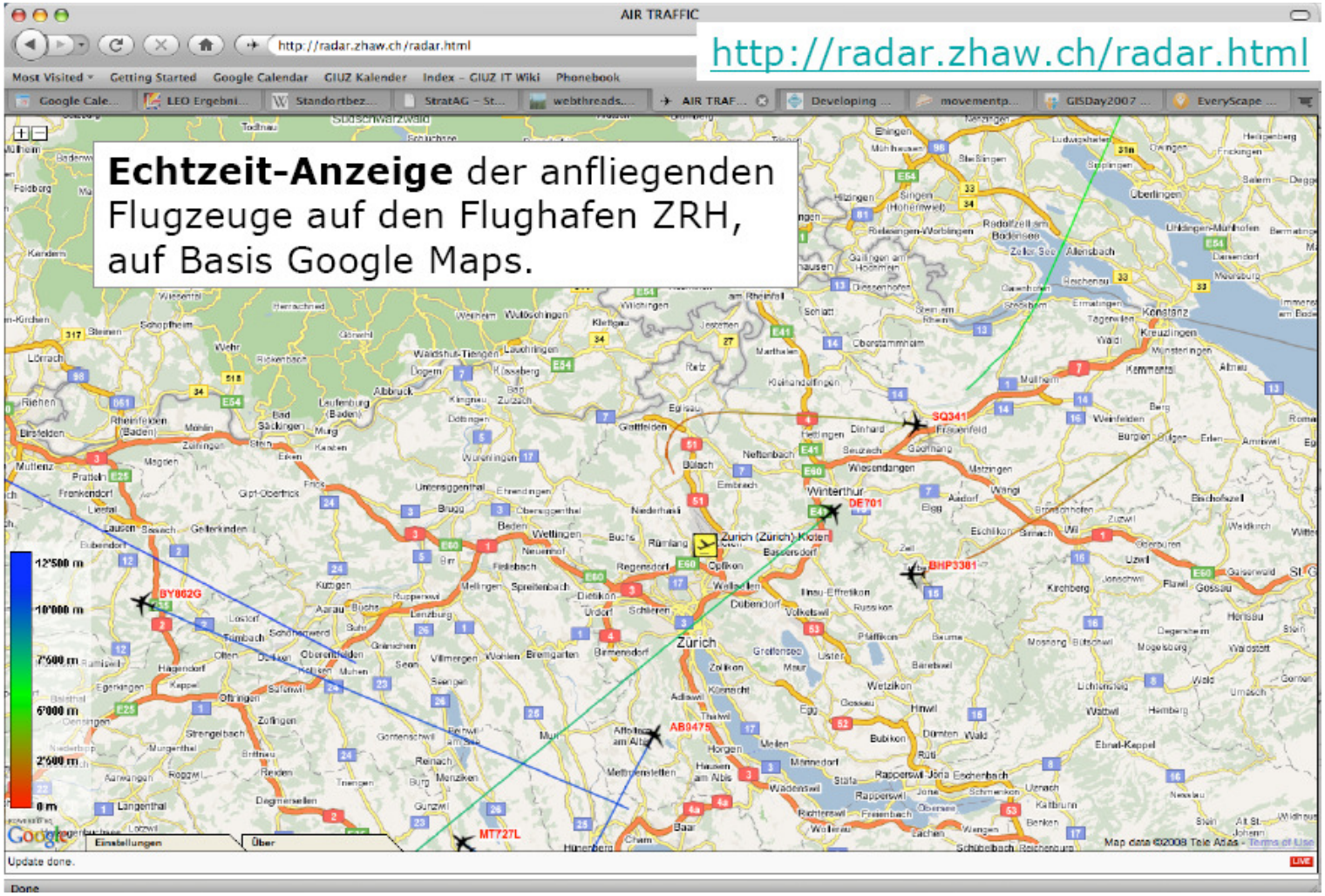
Was braucht es, um GIS in „verteilt“ zu betreiben statt nur isoliert auf dem Desktop wie bis anhin?

- “verteilt” = Daten- und Software-Ressourcen laufen auf unterschiedlichen Rechnern und GIS.

Web-GIS: GIS-Funktionalität übers Web.

- Client/Server-Architektur
- Typisch im amtlichen Bereich:
  - Map Server: Kartendarstellung zu Themen; Zoom/Pan; Layers ein-/ausschalten; Abfragemöglichkeiten (meist nur Suchabfragen)
  - Daten-/Vertriebsserver: "Geoshop"; Geodaten, oft kostenpflichtig
  - Metadaten Server: Suche nach Geodaten.
- Interessant: ‘Mashups’ mit frei zugänglichen Services und Daten auf dem Web (vgl. später).

Mashup-Beispiele

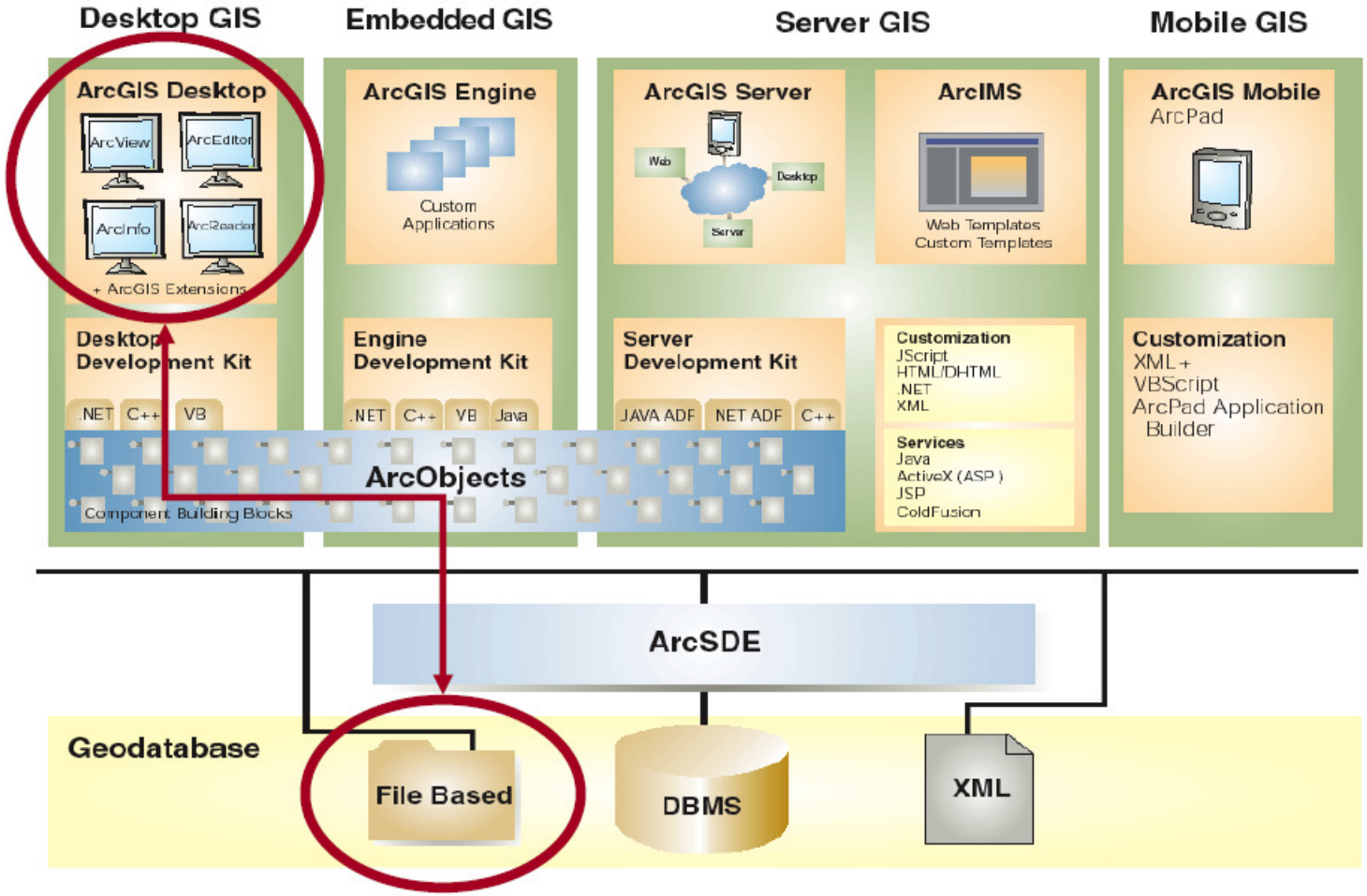


http://radar.zhaw.ch/radar.html

ArcGIS, Fa. ESRI

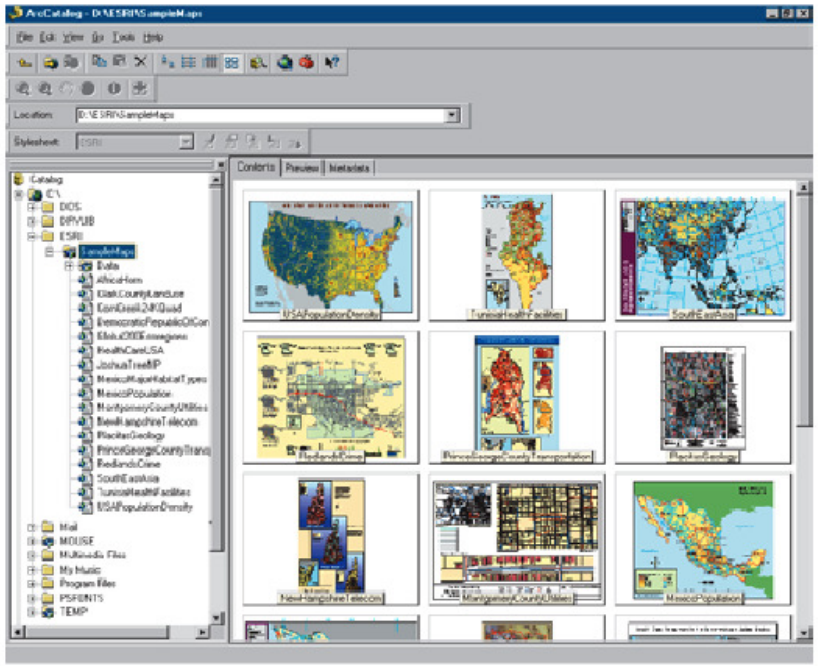
Die Nr.1

ArcGIS 9 Komponenten und Architektur

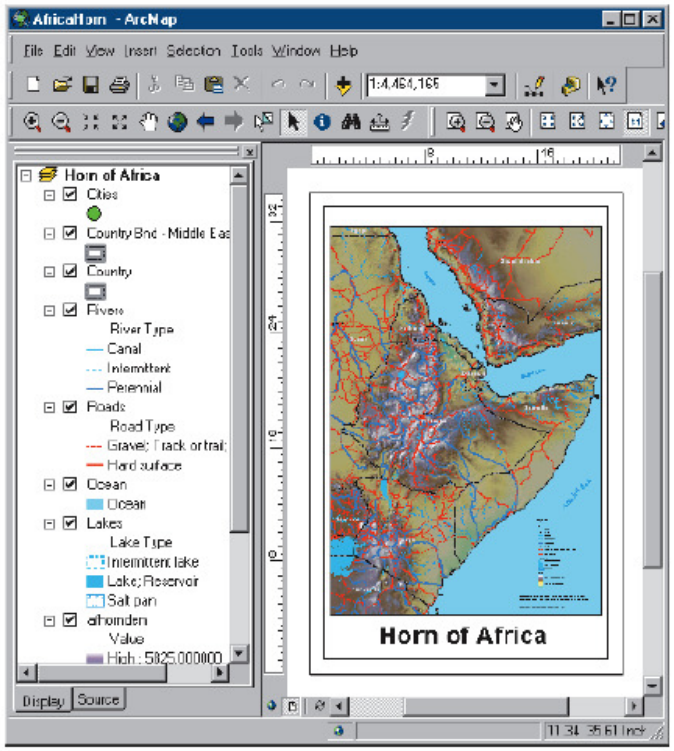


ArcGIS

Hauptkomponenten ArcCatalog und ArcMap:



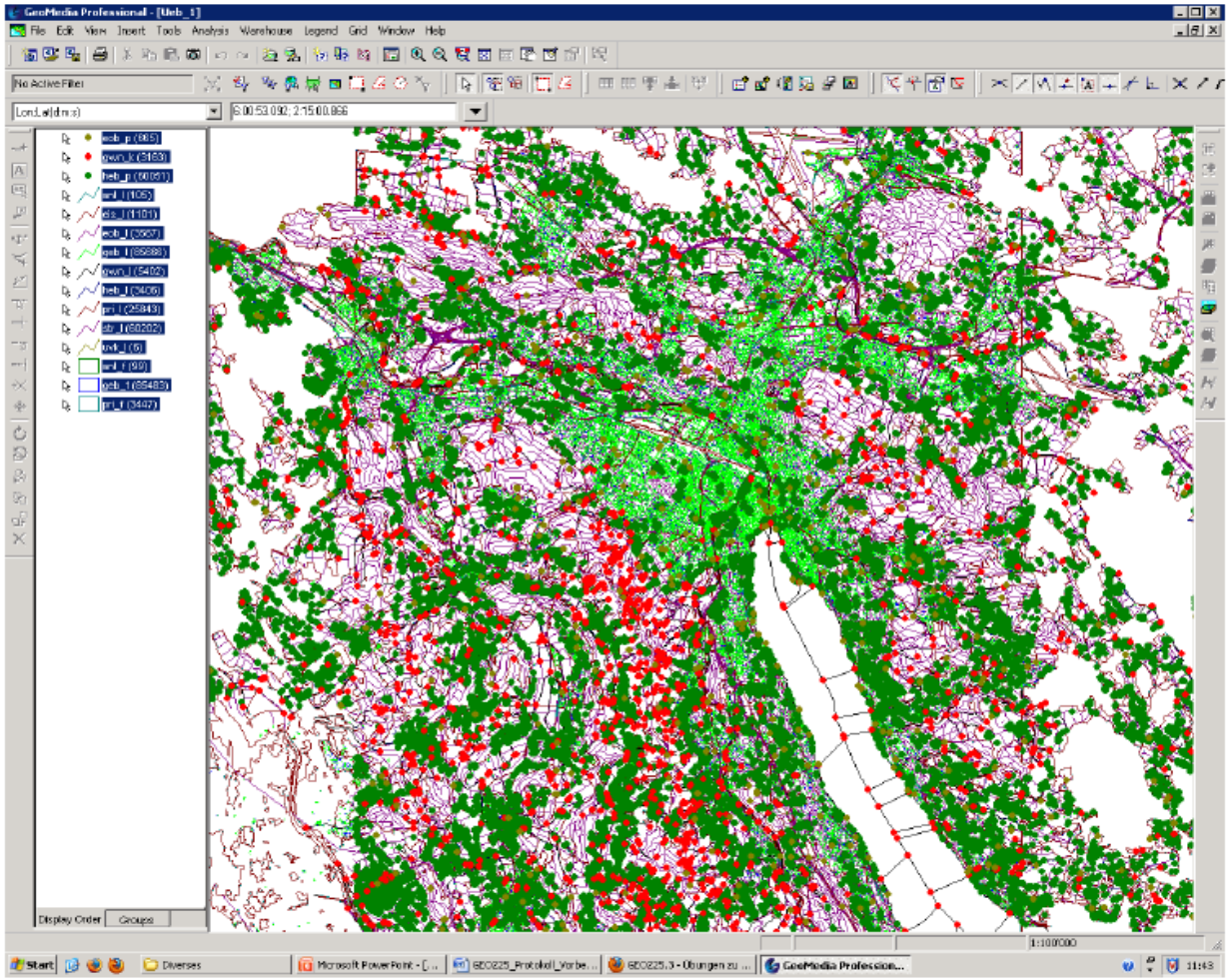
ArcCatalog is the application for managing spatial data holdings and database designs as well as for recording, viewing, and managing metadata.



ArcMap is used for all mapping and editing tasks as well as map-based analysis.

GeoMedia, Intergraph

Die Nr.2



HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

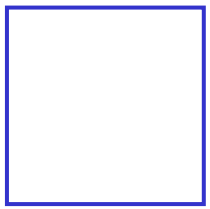
GIS (2)

PostgreSQL (PostGIS) und Ausblick

PostGIS: Einfach

- PostGIS Polygon

  POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))



- Oracle Polygon

```
MDSYS.SDO_GEOMETRY(
    2003, NULL, NULL,
    MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,1),
    MDSYS.SDO_ORDINATE_ARRAY(0,0, 0,1, 1,1,
    1,0, 0,0)
)
```

PostGIS: Schnell und Robust

- Schnell
  - Lightweight geometry implementation
  - Lightweight indexes (50% size savings)
  - Linear time R-Tree
  - Row- level locking algorithm
- Funktionne
  - > 300 Funktionen
  - OGC Simple Features for SQL, ST_Buffer(), etc.
  - Aggregates: ST_Collect(), ST_Union()
  - Extras: ST_AsGML(), ST_AsKML(), ST_AsSVG()
  - ST_BuildArea(), ST_LineMerge, ST_Transform()

Spatial Relationships

Bei raumbezogenen Daten/Geodaten interessieren nicht nur die geografische Lage…

…sondern auch Beziehungen (relationships) zwischen Objekten, die sonst nicht so einfach modelliert werden könnten

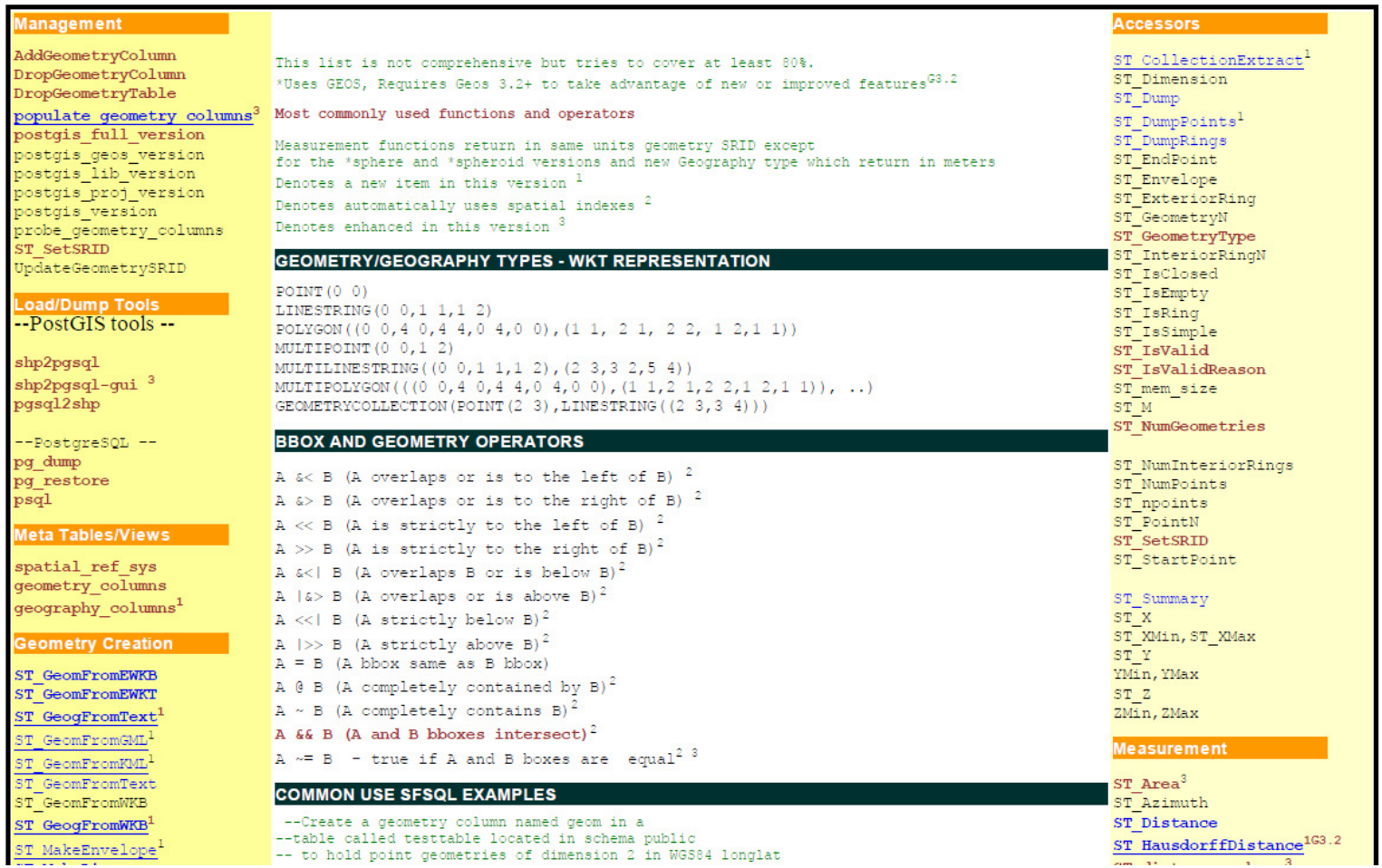
Typische Beziehungen sind

- Nähe (proximity): Distanz
- Nachbarschaft (adjacency): grenzen an, (touching), verbunden mit (connectivity)
- Containment : inside/overlapping

PostGIS Funktionen (>330) in SQL

- Coordinate transformation
- Identify
- Buffer
- Touches
- Crosses
- Within
- Overlaps
- Contains
- Area
- Length
- Point on surface

PostGIS Funktionen (>330) in SQL



Management

AddGeometryColumn
DropGeometryColumn
DropGeometryTable
populate_geometry_columns[3]
postgis_full_version
postgis_geos_version
postgis_lib_version
postgis_proj_version
postgis_version
probe_geometry_columns
ST_SetSRID
UpdateGeometrySRID

Load/Dump Tools

--PostGIS tools --

shp2pgsql
shp2pgsql-gui [3]
pgsql2shp

--PostgreSQL --
pg_dump
pg_restore
psql

Meta Tables/Views

spatial_ref_sys
geometry_columns
geography_columns[1]

Geometry Creation

ST_GeomFromEWKB
ST_GeomFromEWKT
ST_GeogFromText[1]
ST_GeomFromGML[1]
ST_GeomFromKML[1]
ST_GeomFromText
ST_GeomFromWKB
ST_GeogFromWKB[1]
ST_MakeEnvelope[1]

This list is not comprehensive but tries to cover at least 80%.
*Uses GEOS, Requires Geos 3.2+ to take advantage of new or improved features[G3.2]

Most commonly used functions and operators

Measurement functions return in same units geometry SRID except
for the *sphere and *spheroid versions and new Geography type which return in meters
Denotes a new item in this version [1]
Denotes automatically uses spatial indexes [2]
Denotes enhanced in this version [3]

GEOMETRY/GEOGRAPHY TYPES - WKT REPRESENTATION

```
POINT(0 0)
LINESTRING(0 0,1 1,1 2)
POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))
MULTIPOINT(0 0,1 2)
MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))
MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ..)
GEOMETRYCOLLECTION(POINT(2 3),LINESTRING((2 3,3 4)))
```

BBOX AND GEOMETRY OPERATORS

A &< B (A overlaps or is to the left of B) [2]
A &> B (A overlaps or is to the right of B) [2]
A << B (A is strictly to the left of B) [2]
A >> B (A is strictly to the right of B)[2]
A &<| B (A overlaps B or is below B)[2]
A |&> B (A overlaps or is above B)[2]
A <<| B (A strictly below B)[2]
A |>> B (A strictly above B)[2]
A = B (A bbox same as B bbox)
A @ B (A completely contained by B)[2]
A ~ B (A completely contains B)[2]
A && B (A and B bboxes intersect)[2]
A ~= B - true if A and B boxes are equal[2] [3]

COMMON USE SFSQL EXAMPLES

```
 --Create a geometry column named geom in a
--table called testtable located in schema public
-- to hold point geometries of dimension 2 in WGS84 longlat
```

Accessors

ST_CollectionExtract[1]
ST_Dimension
ST_Dump
ST_DumpPoints[1]
ST_DumpRings
ST_EndPoint
ST_Envelope
ST_ExteriorRing
ST_GeometryN
ST_GeometryType
ST_InteriorRingN
ST_IsClosed
ST_IsEmpty
ST_IsRing
ST_IsSimple
ST_IsValid
ST_IsValidReason
ST_mem_size
ST_M
ST_NumGeometries

ST_NumInteriorRings
ST_NumPoints
ST_npoints
ST_PointN
ST_SetSRID
ST_StartPoint

ST_Summary
ST_X
ST_XMin, ST_XMax
ST_Y
YMin, YMax
ST_Z
ZMin, ZMax

Measurement

ST_Area[3]
ST_Azimuth
ST_Distance
ST_HausdorffDistance[1G3.2]

Erstellen einer Datenbank 1 + 2

Von Kommandozeile:

- createdb [option...] [dbname] [description]

```
% createdb -E UTF-8 -T postgis beers_db
-E ... Encoding
-T ... Template Database (Postgis-Fn.)
-D ... Tablespace
```

Mit SQL:

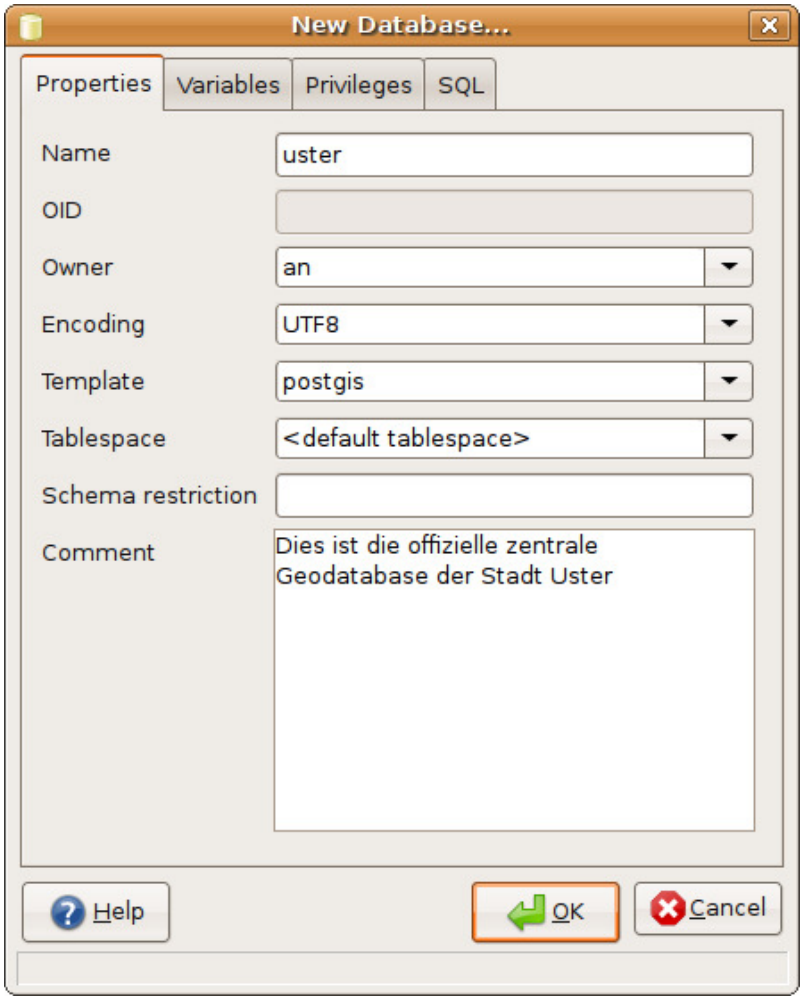
- CREATE DATABASE name

```
CREATE DATABASE beers_db OWNER postgres
ENCODING 'UTF-8,
TEMPLATE template_postgis;
```

Erstellen einer Datenbank 3

pgAdmin III

Falls Geodaten-Typen mit template_postgis



PostGIS-Systemtabellen (1)

geometry_columns – Liste der räumlichen Spalten (mit Angabe zu Tabelle und Schema), Projektionssystem (SRID) und Geometrie-Datentypen in der DB

pgAdmin III Edit Data - PostgreSQL Database Server 8.2 (localhost:5432) - pguebung - geometry_columns

File Edit View Help

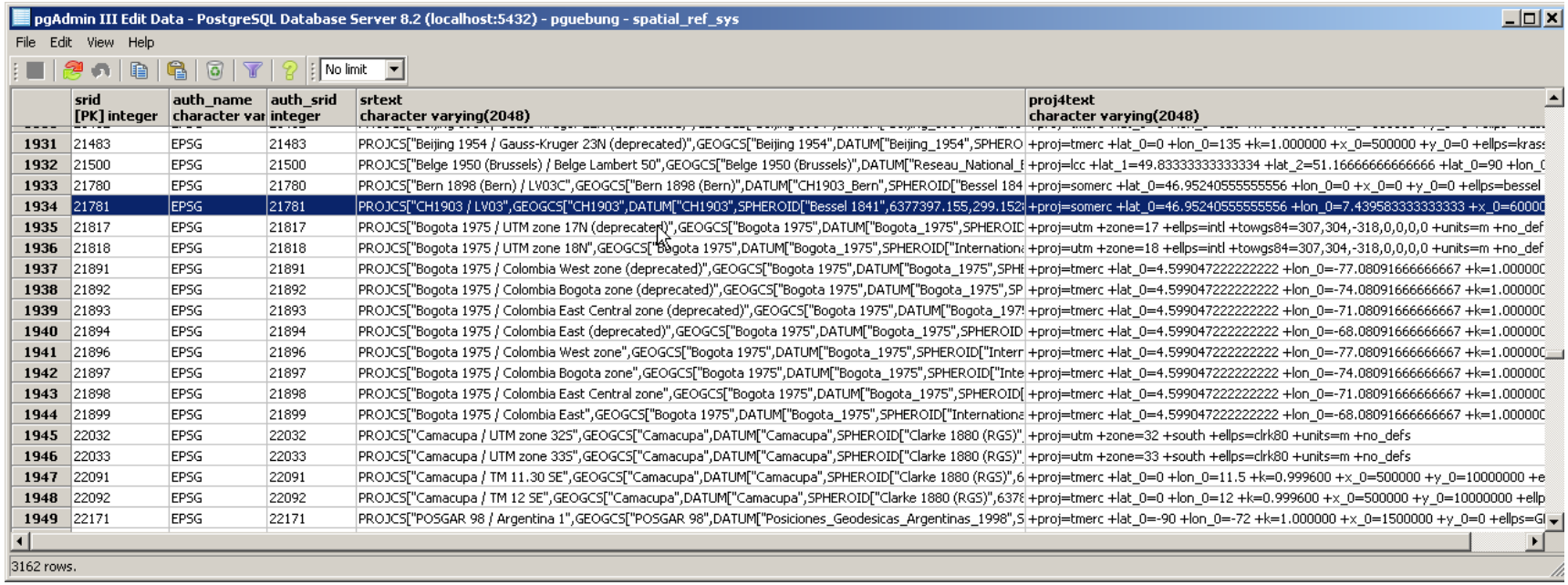
No limit

	oid	f_table_catal [PK] characte	f_table_schei [PK] characte	f_table_name [PK] character varying(256)	f_geometry_ [PK] characte	coord_dimen integer	srid integer	type character varying(30)
1	18345	''	public	fluesse	the_geom	2	21781	MULTILINESTRING
2	18289	''	public	gemeinden	the_geom	2	21781	MULTIPOLYGON
3	18328	''	public	grenzen	the_geom	2	21781	MULTILINESTRING
4	18304	''	public	seen	the_geom	2	21781	MULTIPOLYGON
5	18371	''	public	staedte_schweiz	the_geom	2	21781	POINT
*								

5 rows.

PostGIS-Systemtabellen (2)

spatial_ref_sys – Liste der Projektionen mit SRID und Parametern (für internem Prog. proj4)



Räumliche Erweiterung einer PostgreSQL DB

(entfällt wenn PostGIS-DB über Template erstellt wurde)

Kommandozeile oder pgAdmin (SQL Fenster):

1. Procedural Language Pl/pgSQL aktivieren (Linux):

```
% createlang plpgsql uster
```

2. PostGIS-Objekte, Funktionen und Operatoren laden:

```
% psql -d uster -f lwpostgis.sql
```

3. Koordinatensystem-Daten laden:

```
% psql -d uster -f spatial_ref_sys.sql
```

4. Dokumentation/Kommentare laden:

```
% psql -d uster -f postgis_comments.sql
```

PostgreSQL-Tabelle einrichten

Zuerst Tabelle mit Primärschlüssel erstellen:

```
CREATE TABLE eisenbahnen (id int4, name
text, CONSTRAINT pk_id PRIMARY KEY (id));
```

Dann räumliche Spalte hinzufügen

```
SELECT AddGeometryColumn('public',
'eisenbahnen', 'geom',
21781,'MULTILINESTRING', 2);
```

Dann räumlichen Index erstellen:

```
CREATE INDEX in_eisenbahnen_the_geom ON
 eisenbahnen USING gist(geom);
```

Optional: weitere Indizes erstellen: (z.B. id, name)

Networks

For enhanced network traversal and routing PostgreSQL/PostGIS can be extended with pgRouting software which can perform:

- Shortest path search (with 3 different algorithms)
- Traveling Salesperson Problem solution (TSP)
- Driving distance geometry calculation

http://pgrouting.postlbs.org/

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

PostgreSQL mit PostGIS in Beispielen

Quelle: Pat Browne, 2009-10

Beispiel „Rappi Beers“:

Erzeuge Tabelle “Beers”:

```
create table beers (
  id serial,
  name varchar,
  price float4);

select addgeometrycolumn(
  'public','beers','geom',4326,'POINT',2);
```

Rappi Beers: Daten einfügen

```
insert into beers(name, price, geom) values (
        'Manor', 1.15,
        GeometryFromText('POINT(8.81915 47.22640)',4326)
    );
```

Rappi Beers: Query 1

```
select * from beers order by price;

21;   "Coop Sunnehof";          1.05; "01010000..."
18;   "Manor";                  1.15; "01010000..."
20;   "Dorflade";               2.4;  "01010000..."
23;   "Rappi Bier Factory";     2.9;  "01010000..."
22;   "Lido Markt";             3.1;  "01010000..."
19;   "Biergarten";             3.5;  "01010000..."
17;   "Nelson Pub";             4.5;  "01010000..."
```

Rappi Beers: Query 2

```
select name, price,
  round(st_distance_sphere(geom, GeometryFromText(
    'POINT(8.816511 47.223064)', 4326))::numeric, 0)
      as "distance"
  from beers
  order by distance;
```

```
"Biergarten";      3.5;  235
"Dorflade";        2.4;  300
"Nelson Pub";      4.5;  381
"Manor";           1.15; 422
"Coop Sunnehof";   1.05; 706
```

Rappi Beers: Query 2

```
select name, price,
  round(st_distance_sphere(geom,
            GeometryFromText('POINT(8.816511 47.223064)',
                                4326))::numeric, 0) as
                                                "distance"
  from beers
  order by price;
```

```
"Coop Sunnehof";          1.05; 706
"Manor";                  1.15; 422
"Dorflade";               2.4;  300
"Rappi Bier Factory";     2.9;  966
"Lido Markt";             3.1;  912
```

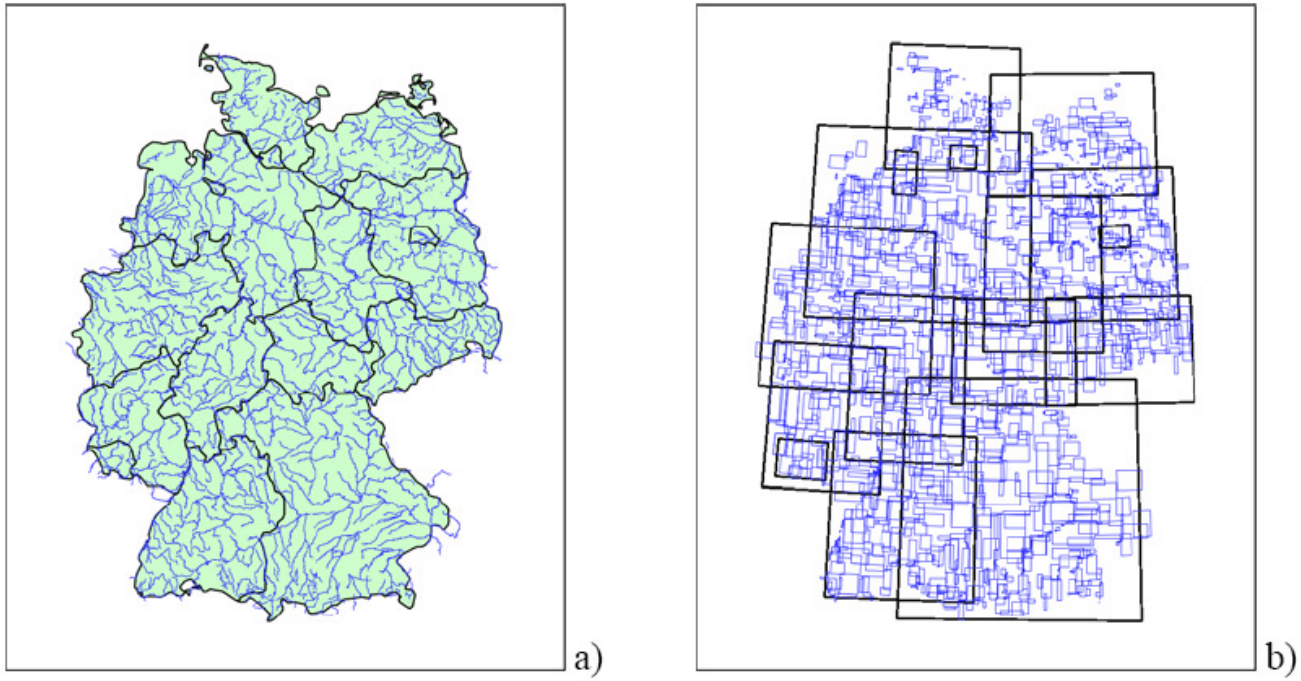
Rappi Beers: Query 3

```
select name, price,
  round((price + 0.001 * st_distance_sphere(geom,
            GeometryFromText('POINT(8.816511
                47.223064)',4326)))::numeric,2)
                   as net_price
  from beers
  order by 3;
```

```
"Manor";                  1.15; 1.57
"Coop Sunnehof";          1.05; 1.76
"Dorflade";               2.4;  2.70
"Biergarten";             3.5;  3.74
"Rappi Bier Factory";     2.9;  3.87
```

Räumliche Indizes erstellen

R-Trees (vgl. GIST) organisieren die Boundingboxen der Daten in verschachtelten räumlichen Rechtecken.



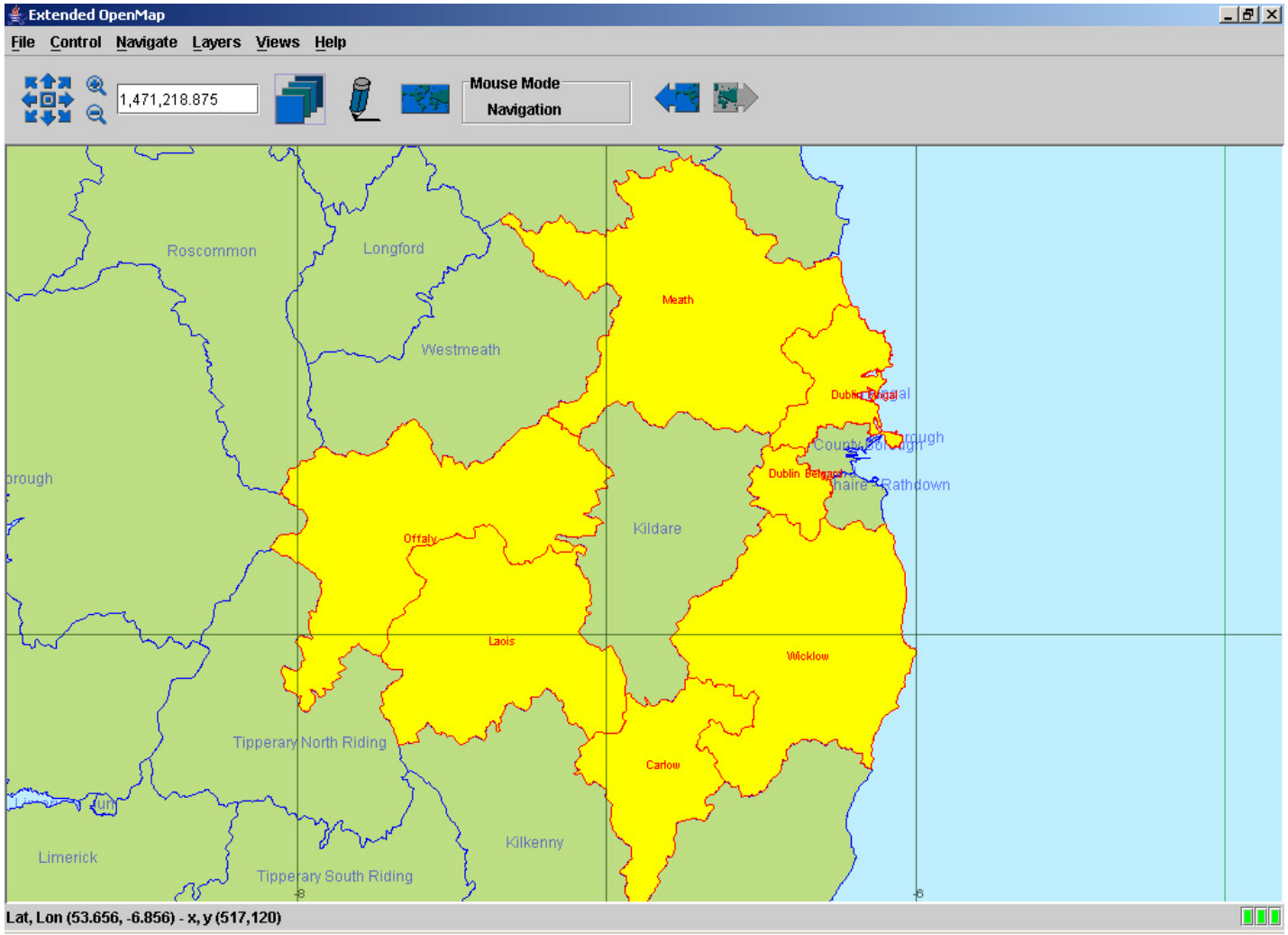
Query 1

"Display all counties that border Rapperswil".

This query can be implemented using the following SQL command:

```
select c1.name as
  name,transform(c1.the_geom,4326) as
  the_geom
from county c1,county c2
where
touches(c1.the_geom,c2.the_geom)
and
c2.name='Rapperswil';
```

Result 1

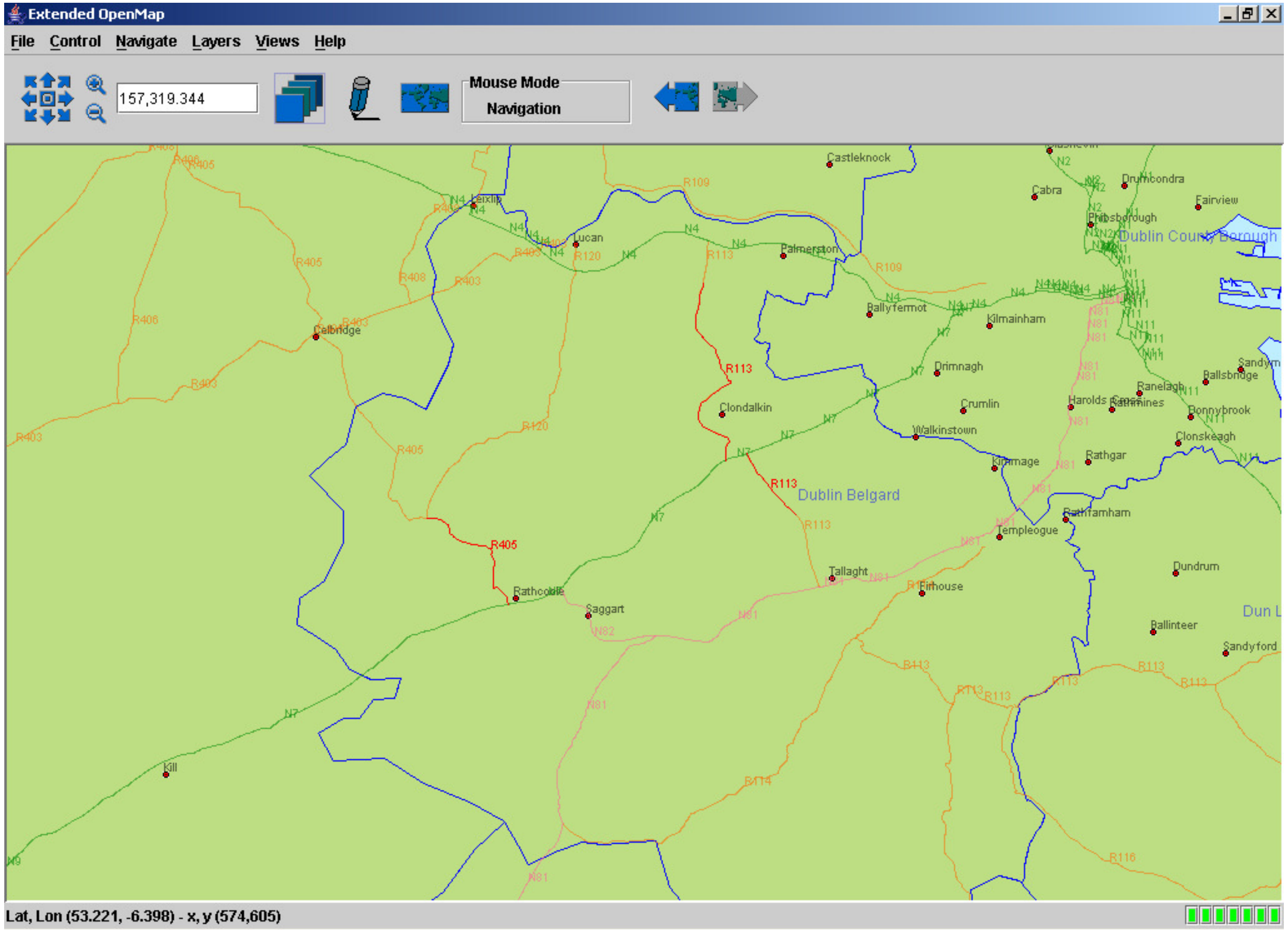

Extended OpenMap
File Control Navigate Layers Views Help
1,471,218.875
Mouse Mode
Navigation
Roscommon
Longford
Westmeath
Meath
Offaly
Kildare
Laois
Wicklow
Carlow
Kilkenny
Tipperary North Riding
Tipperary South Riding
Limerick
Rathdown
Lat, Lon (53.656, -6.856) - x, y (517,120)

Query 2

"Display all regional roads that intersect the N7 National Primary Road within the region of Dublin Belgard" This query can be implemented using the following SQL command:

```
SELECT r.class as name,transform(r.the_geom,4326) AS
   the_geom
FROM regional_road r,national_primary_road n,county c
WHERE
n.class='N7'
AND
n.the_geom && r.the_geom
AND
intersects(n.the_geom,r.the_geom)
AND
c.name='Dublin Belgard'
AND
contains(c.the_geom,intersection(r.the_geom,n.the_geo
   m));
```

Result 2


Extended OpenMap
File Control Navigate Layers Views Help
157,319.344
Mouse Mode
Navigation
Castleknock
Cabra
Drumcondra
Fairview
Leixlip
Lucan
Palmerston
Dublin County Borough
Ballyfermot
Kilmainham
Celbridge
Drimnagh
Clondalkin
Crumlin
Walkinstown
Rathgar
Dublin Belgard
Rathfarnham
Templeogue
Dundrum
Tallaght
Rathcoole
Saggart
Firhouse
Ballinteer
Sandyford
Kill
Lat, Lon (53.221, -6.398) - x, y (574,605)

Querying moving objects

Find where and when will it snow given

Clouds(X, Y, Time, humidity)

Region(X, Y, Time, temperature)

```
(SELECT          x, y, time
 FROM     Clouds
WHERE  humidity  >=  80)
    INTERSECT
 (SELECT  x, y, time
FROM     Region
WHERE  temperature  <=  32)
```

Example Query

“How many people live within 5 miles of the toxic gas leak?”

```
SELECT sum(population)
FROM census_tracts
WHERE
  distance(census_geom,’POINT(…)’) < 5
```

Image from Paul Ramsey Refractions Research

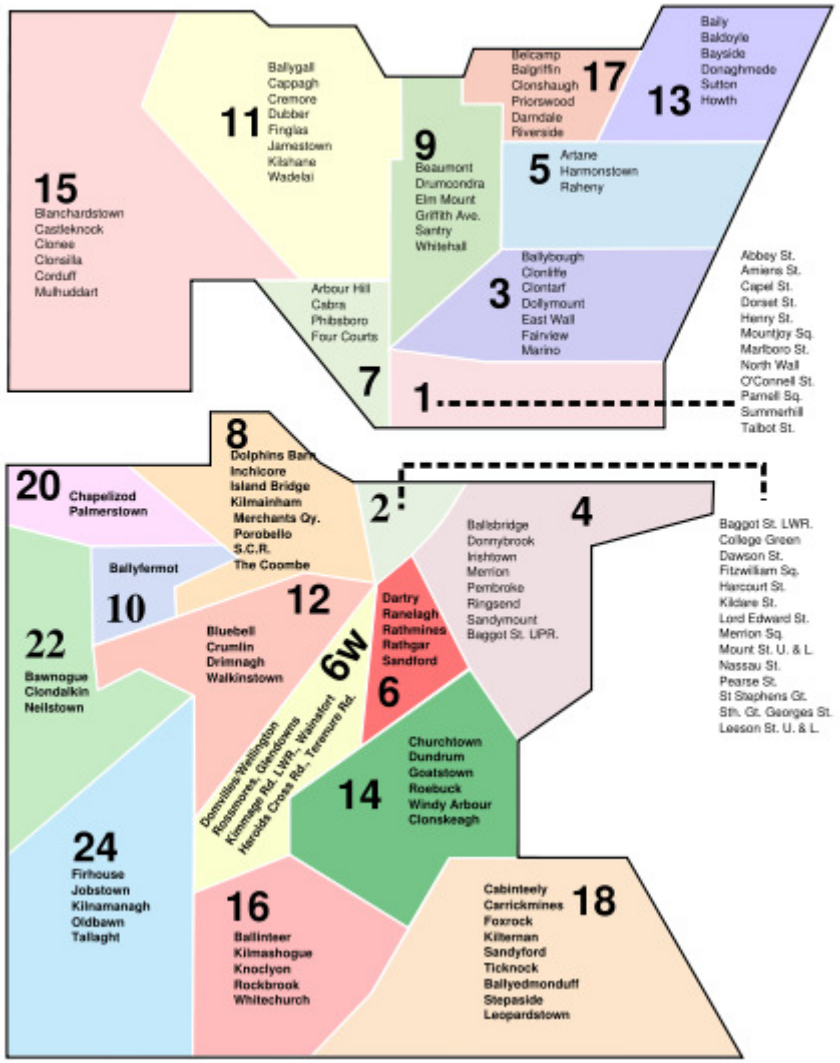
Example Query

"What is the area of all parks inside the Dublin postal district 1?"

```
SELECT sum(area(park_geom))
FROM parks, postalDistrict
WHERE
contains(postalDistrict_geom,park_geom) AND
   postalDistrict_name = 'Dublin 1'
```

Based on lecture from Paul Ramsey Refractions Research

Dublin Postal District



Example Query

What is the maximum distance a student has to travel to school?”

```
SELECT
  max(distance(
    student_location, school_location
  ))
FROM students, schools
```

Based on lecture from Paul Ramsey Refractions Research

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

SQL/MM
Schlussbemerkungen

SQL/MM Schlussbemerkungen

SQL/MM - Eine SQL-Norm für Medienobjekte:

- erst zwei Parts verabschiedet: FullText, Spatial
- Teil für Bilder ist in der Normung, noch einige Kritik
- für Video und Audio sind noch keine Parts in Sicht

Merkwürdige Uneinheitlichkeit, z.B. Contains bei FullText, Score bei StillImage: Warum nicht Generalisierung zu MM_Object o.ä.?

Frage:

- Werden die Normen umgesetzt?
- Wie schnell veralten sie durch "neuere" IT?

DBMS mit Geodaten-Erweiterung

- Oracle Spatial
  - Komplizierte Implementation: Geometrie über Punkte- und Infoarray :-<
- DB2:
  - Ok, wenig verwendet
- MS SQL Server:
  - Seit letzter Version (endlich)
- MySQL:
  - One man Show?
- PostgreSQL/PoistGIS
  - “the leader”… :->

Was ist speziell an Geo-DB?

- Geometrie-Datentypen (2D und 2D+Z)
  - Spezielle Methode
  - Eigener Index
- Komplexe Funktionen
  - Views: Verknüpfung mit Computergrafik
  - Lange Transaktionen
- Benutzerspezifische Daten
  - Spezielle Konsistenzbedingungen
  - Grosse Datenmengen
  - Teure Datenerfassung

Was ist der Beitrag von Geo-DB?

- Anwendungsbeispiel für die objekt-relationalen Erweiterungen (von SQL:99, bzw. SQL/MM): Benutzerdef. Datentyp
- Beitrag zur Datenmodellierung durch zusätzlicher
  - Beziehungstyp: der Raumbezug
  - Neuer Verbundoperator: 'Spatial Join‘

Quellen zu GIS

- Weblinks:
  - GIS-Know How: www.gis.hsr.ch/wiki/PostGIS

- Bücher:
  - Ramakrishnan & Gehrke
  - Geoinformatik, Bartelme

- Aus- und Weiterbildung:
  - Modul GIS im Master MSE
  - GIS-Kurse an der HSR
  - UNIGIS Master in GIS, Salzburg

SQL: WINDOW – Oracle Implementation

- Aggregation über ein "query window"

```
SELECT year, month,
       AVG(sales) OVER
            (PARTITION BY year
             ORDER BY year, month
             ROWS BETWEEN 1 PRECEDING
                      AND 1 FOLLOWING)
       AS moving_average
  FROM all_sales
 WHERE year > 2003
```

Gruppen auf die der SQL-Befehl angewandt wird

Sortierung innerhalb der Gruppen

window für die Berechnung je eines Ergebnistupels

Aggregate functions go analytical

- bekannt: Aggregatfunktionen in SQL
  - **`AVG()`**
  - **`COUNT()`**
  - **`MAX()`**
  - **`MIN()`**
  - **`SUM()`** und so weiter…
- Die meisten Aggregatfunktionen sind auch als analytische Funktionen anwendbar

  → sie können mit query windows benutzt werden

analytische Aggregatfunktion - Beispiel

```
SELECT
COUNT(1) OVER (PARTITION BY lecture),
COUNT(1) OVER (PARTITION BY exam),
AVG(mark) OVER (PARTITION BY exam),
SUM(ects) OVER (PARTITION BY student),
SUM(ects) OVER (PARTITION BY student
                ORDER BY semester)
FROM t_exam_participation
```

Anzahl Studierende pro Vorlesung

Anzahl Studierende pro Prüfung

ØNote pro Prüfung

Summe aller ECTS-Punkte pro Studierende

Summe aller ECTS-Punkte pro Studierendem bis zu diesem Semester

- alles in nur 1 SQL-Befehl…

Analytic functions (Auswahl) - 1

Zusätzliche, neue analytische Funktionen:

- **`VAR_POP()`, `VAR_SAMP()`, `STDDEV_POP()`, `STDDEV_SAMP()`**
  - Varianz und Standardabweichung bestimmen
- **`RANK()`, `DENSERANK()`**
  - Reihenfolge ermitteln – mit und ohne (dense) Lücken
- **`CORR()`**
  - Berechnung des Korrelationskoeffizienten

Analytic functions (Auswahl) - 2

Mehr zusätzliche analytische Funktionen:

- **`LAG(), LEAD()`**
  - gehe um n records in der Partition vor (lead) oder zurück (lag)
- **`RATIO_TO_REPORT()`**
  - Anteil des Wertes an der Gesamtsumme der Partition
- **`NTILE()`**
  - Teilt jede Partition in n gleiche Teile auf (Viertel, Zehntel, Halbe usw.)

Analytic functions - Anwendungsbeispiele

```
SELECT RANK() OVER (ORDER BY sales) ...
```

```
SELECT sales AS this_month,
       LAG(sales,3) OVER (ORDER BY month)
            AS three_months_ago ...
```

```
SELECT product_no, product_name,
       RATIO_TO_REPORT(sales) OVER
          (PARTITION BY product_line) ...
```

```
SELECT LAST_VALUE(shareprice) OVER
    (ORDER BY shareprice ROWS
 BETWEEN 38 PRECEDING AND CURRENT ROW) ...
```

Erweiterung von GROUP BY

- mehrere unterschiedliche Aggregationen in einer Abfrage

```
SELECT product, date, SUM(sales)
  FROM t_sales
 GROUP BY GROUPING SETS ((product,date),date)
```

record	product	date	sum(sales)
1	beer	20-Mar-2009	34'987.00
2	diapers	20-Mar-2009	65'023.58
3561		19-Mar-2009	900'520.60
3562		20-Mar-2009	1'236'998.21

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

DWH: Von Decision Support zu Data Warehouses

Datenbanksysteme 2

Dr. Klaus Wolfertz

04. Mai 2010

Die nächsten 5 Veranstaltungen

Analyse von Daten übergeordnetes Thema

- Analyse strukturierter Daten
  - Auswertung bekannter Muster → **DSS, OLAP**
  - Finden unbekannter Muster → **Data Mining**

Data Warehouses: Grundlage für beides

nicht Thema:

- Analyse un/semi-strukturierter Daten
  - insbesondere natürlichsprachliche Texte

    → Information Retrieval

Überblick

- **Von Decision Support zu Data Warehouses**
- OLAP – Online Analytical Processing
- Data Mining – Klassifikation
- Data Mining – Assoziationsanalyse
- Data Mining – Clustering

Analyse strukturierter Daten anhand bekannter Muster

- Datenanalyse
  - selektieren, aggregieren, sortieren von Datenwerten
- strukturierte Daten
  - Verwendung von Datenbanksystemen
  - Analyse von "elementaren" Datentypen
- anhand bekannter Muster
  - Attribute, "Dimensionen" des Resultats sind bekannt

Agenda

Von Decision Support zu Data Warehouses

- **Decision Support**
- Analytical Queries
- Datenbankintegration

Transaction Processing

Einsatz von Datenbanksystemen in erster Linie für **"Transaction Processing"**

- Tagesgeschäft
  - auf "operationellen" Datenbanksystemen
- viele, aber kurze Transaktionen
  - viele Benutzer, viele Schreiboperationen, hoher I/O-Durchsatz
- DBMSe sind optimiert gerade für "Transaction Processing"

Beispiele für Transaction Processing

Typische Anwendungen

- Auftragsbearbeitung
- Reservationssysteme
- Zahlungsverarbeitung
- Lagerverwaltung
- Personalverwaltung
- Online Shop

...

Analytical Processing

Datenanalyse ist etwas anderes!

es geht um **"Analytical Processing"**

- Transaction Processing
  - für schnelles, kostengünstiges Ausführen von genau definierten und standardisierten Geschäftsabläufen
- Analytical Processing
  - soll **Entscheidungen** im Unternehmen untermauern, die noch gefällt werden müssen

Decisions

Was macht Entscheiden so schwierig?

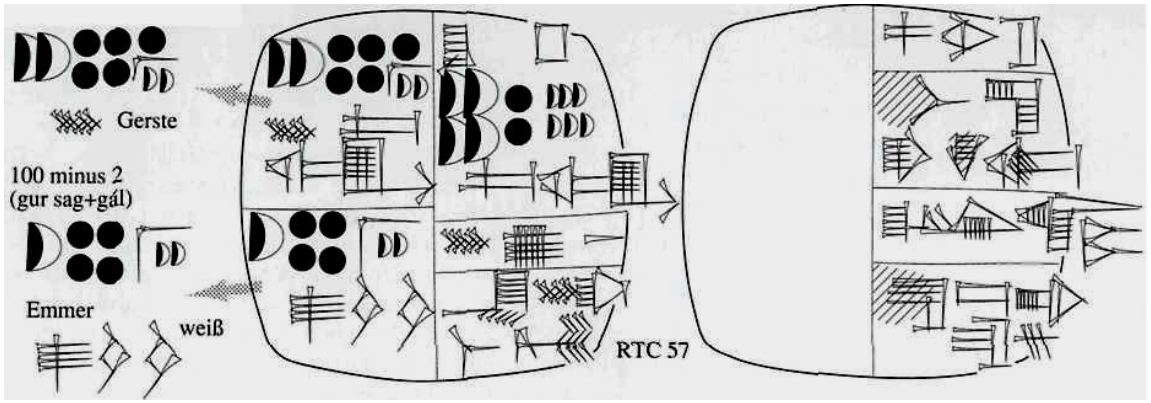
- unvollständige und unsichere Information
- Risiko, "falsch" zu entscheiden
- Unterstützung jeder Art dabei hilfreich

"Decision Support" (Entscheidungsunterstützung)

- Risiko mindern durch
  - Information vervollständigen
  - Information verlässlicher machen

Management Support

- Decision Support = Management Support
- Management ist (und war immer) interessiert an Datenanalyse
  - Buchhaltung
  - Controlling
  - Reports
  - etc. ...



Altsumerischer "Buchungsbeleg" aus:
NISSEN/ DAMEROW/ ENGLUND, Frühe Schrift und Techniken der Wirtschaftsverwaltung im alten Vorderen Orient: Informationsspeicherung und -verarbeitung vor 5000 Jahren, 2. Aufl., Bad Salzdetfurth 1991, S. 85

- notwendig: vollständige Sicht auf alle Bereiche des Unternehmens
- Heutzutage: softwaregestützt

Decision Support

Decision Support Systems – DSS

(Entscheidungsunterstützungssysteme - EUS)

- Gewinnen aktueller und vergangener Daten
  - aus verschiedenen operationellen Systemen
- Verdichten der Daten
  - Erstellen von Salden
  - Berechnung von Kennzahlen
- Ziel
  - Verstehen und Beeinflussen der Abläufe im Unternehmen

Anwendungsgebiete für DSS

- Informationsorientierte DSS
  - verteilen vorhandener Information
  - Messgrössen liefern, Kennzahlen berechnen
- Analyseorientierte DSS
  - Generierung neuer Information basierend auf vorhandener Information
- Planungsorientierte DSS
  - unterstützen Soll/Ist-Vergleiche als grundlegendes Führungsinstrument

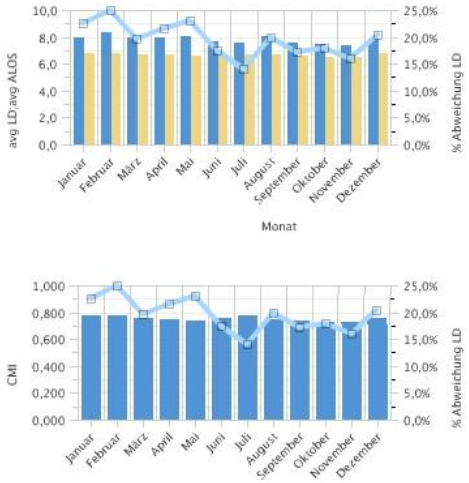
Informationsorientierte DSS

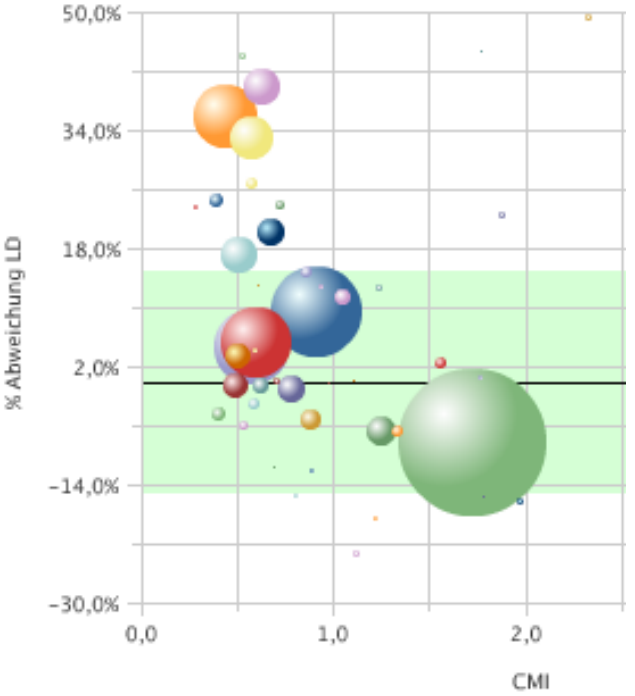
"Corporate Performance Management"

- Liefern entscheidungsrelevanter Indikatoren
- Absolutzahlen
  - Umsatz, Gewinn, Cashflow, Bilanzsumme
  - Deckungsbeitrag, Handelsspanne
- Verhältniszahlen
  - Umsatzrentabilität, Kapitalrendite (ROI)
  - Lagerumschlag, Arbeitsproduktivität
- Einzelkennzahlen → Kennzahlensystem

Informationsorientierte DSS - Visualisierung

- Darstellung: spezielle Benutzeroberflächen





– Tacho-, Ampel-Anzeige → Dashboard / Cockpit

Analyseorientierte DSS

"Marketing- und Vertriebs-Controlling"

- Umsatzberechnung und -vorausschätzung
- Beurteilen der Effizienz
  - einzelner Marketingmassnahmen
  - der gesamten Vertriebsorganisation
- Beurteilen des Ressourceneinsatzes
  - Optimal eingesetzt? Wo erhöhen, reduzieren?
  - mehr Ertrag durch andere Kombination?
- → Marktvorsprung erreichen / halten

Marketing- und Vertriebs-Controlling

- Untersuchen aller Kombinationen relevanter Einflussgrössen
  - Auftragsgrösse
  - Absatzregion
  - Vertriebskanal
  - Kundensegment
  - Sortiment
  - Zeit

- Kundenpotentiale für neue / ergänzende Angebote identifizieren
- Simulation von Marketing- und Vertriebsmassnahmen

Scorecard Systems

- Kennzahlen**hierarchie**
  - Aggregation entlang der Hierarchie
    - → Spitzenkennzahl (key performance indicator - KPI)

Beispiele:

- DuPont System of Financial Control
  - KPI: Return on Investment (ROI)
- Balanced Scorecard
  - Ursache/Wirkung in der Wertschöpfungskette
  - Kombination verschiedener Sichten:
    - Finanzen, Kunden, Lernen & Innovation, Prozesse

Kostenrechnung

- Kostenstellenrechnung
- Auftragskalkulation

Kombinieren und Umlegen von Kosten auf

- Kostenstellen und Kostenträger
- Aufträge und Auftragsgrössen
- Kunden und Kundensegmente
- Vertriebskanäle und Absatzregionen

Planungsorientierte DSS

"Management Information Systems" (MIS)

- Ist-Kennzahlen ergänzt mit Planzahlen
- "Management" benötigt Planzahlen
  - zur Beurteilung von Abweichungen
  - um Entscheidungen zu treffen
- Soll/Ist-Vergleiche
  - wo sind Entscheidungen nötig
  - wo sind Entscheidungen am wirksamsten
- Simulation verschiedener Plan-Szenarien

Ursprünge von DSS

- Operations Research (Unternehmensforschung)
  - Lösen mathematischer Optimierungsprobleme
    - Beispiel: Produktionsprogrammplanung
- Anwendung von Managementtheorien
  - zur Marktentwicklung
  - zum Kundenverhalten
- Simulationen
- Visualisierung

DSS Anwender

- "user fitness" für DSS
  - nur 10 – 20% der Benutzer beherrschen ad hoc queries und Reporting-Tools
- heterogene Anwendergemeinde
  - vom CEO bis zum Sachbearbeiter
  - sporadische Anwender und "Power User"
- Konsequenzen für Bedienoberflächen
  - nicht zu einfach, nicht zu komplex
  - benutzerspezifisch konfigurierbar
  - Browser-Schnittstelle weit verbreitet

Datenbanksysteme für Decision Support

Daten für Decision Support

- werden in Datenbanksystemen gehalten
- müssen spezifisch ausgewertet werden
  - Kennzahlenberechnung / -ableitung

→ *spezielle Analyseabfragen notwendig*

- stammen aus vielen Geschäftsbereichen
  - die jeweils eigene Datenbanken betreiben

→ *Integration von Datenbanken notwendig*

Agenda

Von Decision Support zu Data Warehouses

- Decision Support
- **Analytical Queries**
- Datenbankintegration

Analytische Datenbankanfragen

"Analytical Processing" in der relationalen Welt

- bis und mit SQL-92
  - Aggregatfunktionen (z.B. `SUM()`, `AVG()`)
  - GROUP BY
- seit SQL:1999
  - mehr Unterstützung für die Datenanalyse
  - speziell das SQL/OLAP "amendment" von 2000

die wichtigsten Möglichkeiten nachfolgend…

Erweiterung von GROUP BY

- mehrere unterschiedliche Aggregationen in einer Abfrage

```
SELECT product, date, SUM(sales)
  FROM t_sales
 GROUP BY GROUPING SETS ((product,date),date)
```

record	product	date	sum(sales)
1	beer	20-Mar-2009	34'987.00
2	diapers	20-Mar-2009	65'023.58
3561		19-Mar-2009	900'520.60
3562		20-Mar-2009	1'236'998.21

SQL: WINDOW-Konstrukt

- Aggregation über ein "query window"

```
SELECT year, month,
       AVG(sales) OVER w AS moving_average
  FROM t_sales
 WHERE year > 2003
WINDOW w AS (PARTITION BY year
             ORDER BY month
             ROWS BETWEEN 1 PRECEDING
                      AND 1 FOLLOWING)
```

Gruppen auf die der SQL-Befehl angewandt wird

Sortierung innerhalb der Gruppen

window für die Berechnung je eines Ergebnistupels

Window - Veranschaulichung

- Anzahl Ergebnistupel identisch mit / ohne Window
- Für jede Partition SQL getrennt ausgeführt
- Für jedes Tupel Window neu bestimmt
  - falls nicht spezifiziert = Partition

```
... OVER
(PARTITION BY year
 ORDER BY month
 ROWS BETWEEN
 1 PRECEDING AND
 1 FOLLOWING)
```

record	year	month	sum(sales)
100	2008	November	87'986.22
101	2008	December	123'870.00
102	2009	January	111'787.13
103	2009	February	185'542.37
104	2009	March	131'572.01
105	2009	April	109'564.28
106	2009	May	108'560.03

ORDER BY

Partition

Partition

Window

SQL: WINDOW – Oracle Implementation

```
SELECT year, month,
       AVG(sales) OVER
             (PARTITION BY year
              ORDER BY month
              ROWS BETWEEN 1 PRECEDING
                       AND 1 FOLLOWING)
       AS moving_average
  FROM t_sales
 WHERE year > 2003
```

Aggregate functions go analytical

- bekannt: Aggregatfunktionen in SQL
  - **`AVG()`**
  - **`COUNT()`**
  - **`MAX()`**
  - **`MIN()`**
  - **`SUM()`** und so weiter…
- Die meisten Aggregatfunktionen sind auch als analytische Funktionen anwendbar

  → sie können mit query windows benutzt werden

analytische Aggregatfunktion - Beispiel

```
SELECT
COUNT(1) OVER (PARTITION BY lecture),
COUNT(1) OVER (PARTITION BY exam),
AVG(mark) OVER (PARTITION BY exam),
SUM(ects) OVER (PARTITION BY student),
SUM(ects) OVER (PARTITION BY student
                ORDER BY semester)
FROM t_exam_participation
```

Anzahl Studierende pro Vorlesung

Anzahl Studierende pro Prüfung

ØNote pro Prüfung

Summe aller ECTS-Punkte pro Studierende

Summe aller ECTS-Punkte pro Studierendem bis zu diesem Semester

- alles in nur 1 SQL-Befehl…

Analytic functions (Auswahl) - 1

Zusätzliche, neue analytische Funktionen:

- **`VAR_POP()`, `VAR_SAMP()`, `STDDEV_POP()`, `STDDEV_SAMP()`**
  - Varianz und Standardabweichung bestimmen
- **`RANK()`, `DENSERANK()`**
  - Reihenfolge ermitteln – mit und ohne (dense) Lücken
- **`CORR()`**
  - Berechnung des Korrelationskoeffizienten

Analytic functions (Auswahl) - 2

Mehr zusätzliche analytische Funktionen:

- **`LAG(), LEAD()`**
  - gehe um n records in der Partition vor (lead) oder zurück (lag)
- **`RATIO_TO_REPORT()`**
  - Anteil des Wertes an der Gesamtsumme der Partition
- **`NTILE()`**
  - Teilt jede Partition in n gleiche Teile auf (Viertel, Zehntel, Halbe usw.)

Analytic functions - Anwendungsbeispiele

```
SELECT RANK() OVER (ORDER BY sales) ...
```

```
SELECT sales AS this_month,
       LAG(sales,3) OVER (ORDER BY month)
            AS three_months_ago ...
```

```
SELECT product_no, product_name,
       RATIO_TO_REPORT(sales) OVER
          (PARTITION BY product_line) ...
```

```
SELECT LAST_VALUE(shareprice) OVER
    (ORDER BY shareprice ROWS
 BETWEEN 38 PRECEDING AND CURRENT ROW) ...
```

Vorteile für den Anwender

Kennzahlenberechnung für DSS

- nur eine oder wenige Abfragen
  - für komplexe betriebswirtschaftliche oder statistische Berechnungen
- Berechnung vom DBMS durchgeführt
  - Keine Programmierung notwendig
    - ~~Entwicklung, Test, Wartung~~
  - Keine Programmierkenntnisse notwendig
    - für Decision Support-Standardabfragen
    - für die Spezifikation von Kennzahlenberechnungen

Performanzsteigerung

- Performantere Abfragen
  - DB-Optimizer kennt Analytic Functions
  - Parallelisierung möglich
  - Kombination mit anderen Selektionskriterien
- Sekundärspeicherzugriffe reduziert
  - DB-externe Programmlogik würde alle Daten benötigen und erst in den Hauptspeicher laden

Analytic functions – Oracle Implementation

- Für Oracle
  - vollständige Liste aller analytischen Funktionen unter:

http://download-uk.oracle.com/docs/cd/B19306_01/server.102/b14200/functions001.htm#sthref964

  - Link ist auf dem Wiki, unter dem Eintrag für die heutige Übung

- Für andere DBMS-Produkte
  - siehe Handbücher des jeweiligen Herstellers

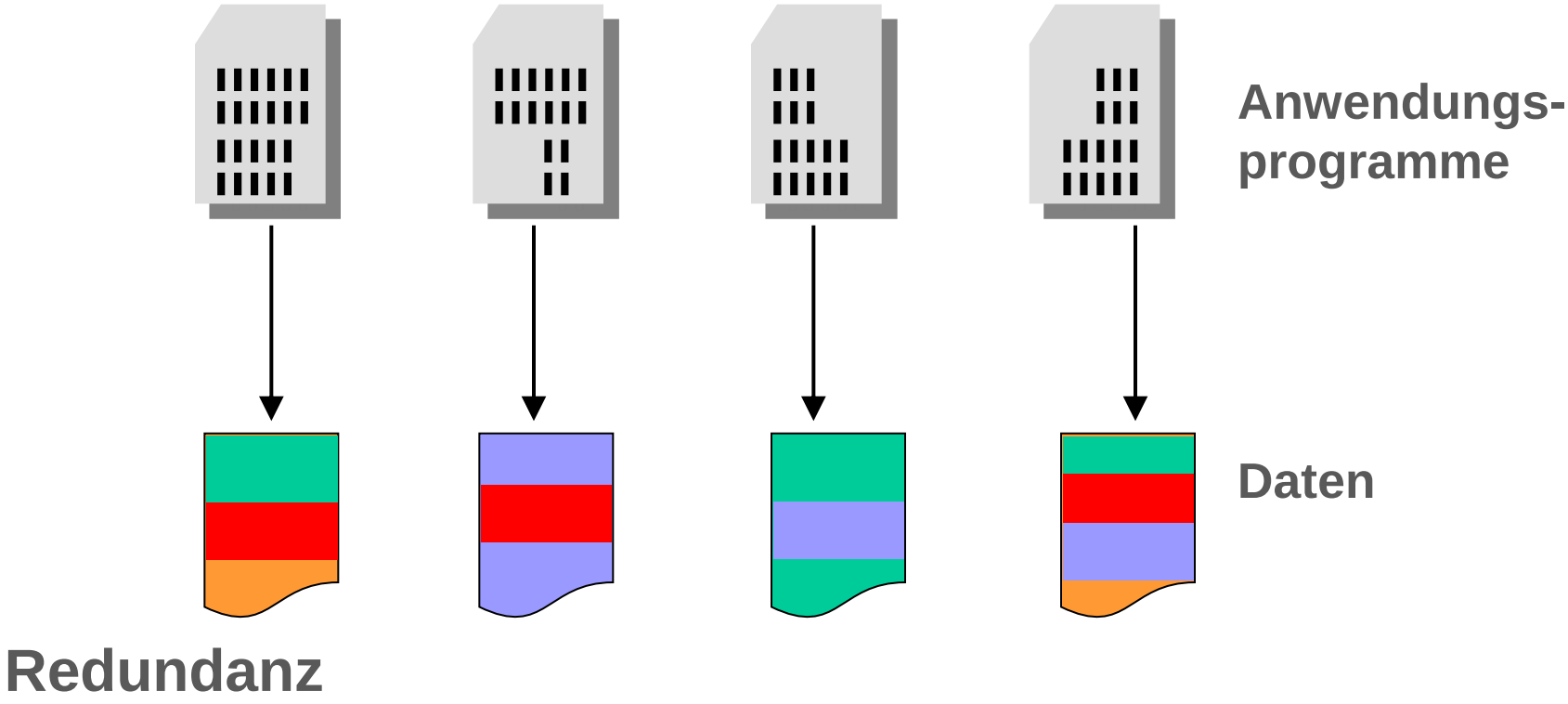
Agenda

Von Decision Support zu Data Warehouses

- Decision Support
- Analytical Queries
- **Datenbankintegration**

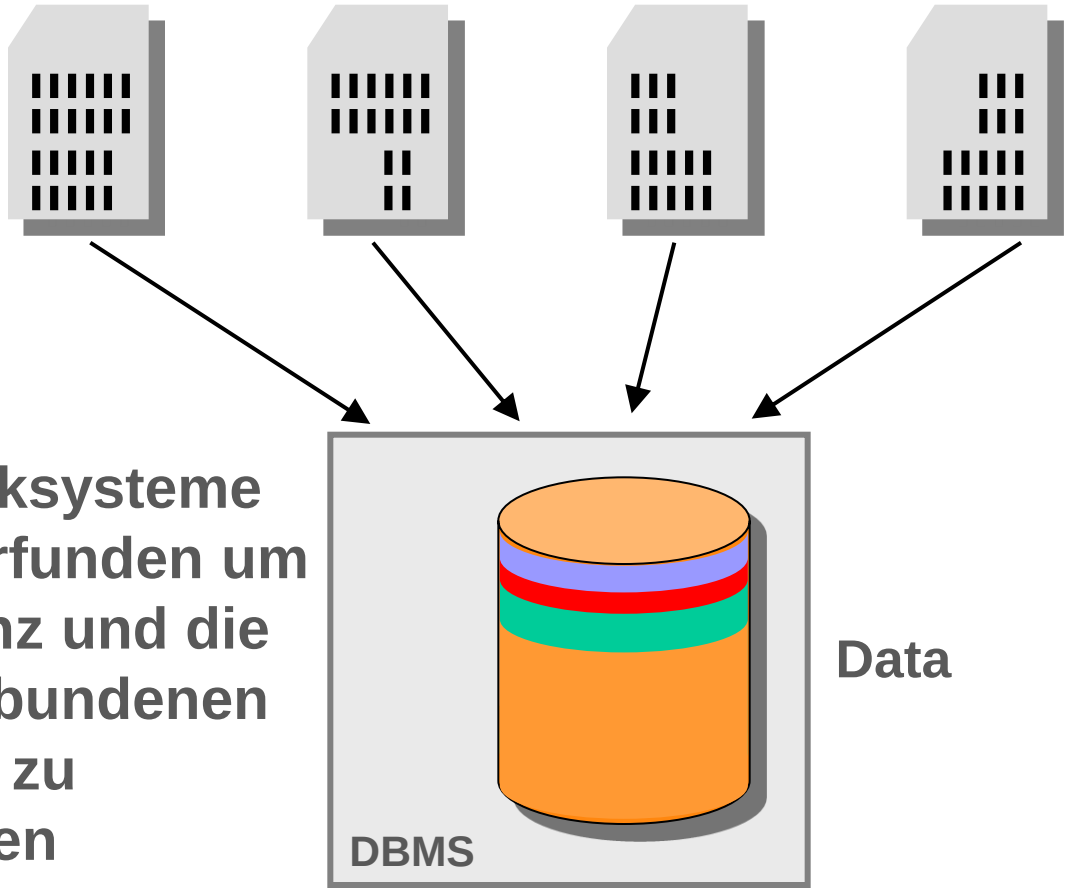
Datenredundanz

- Mehrfachspeicherung von Daten



Datenintegration

- Einmalspeicherung von Daten



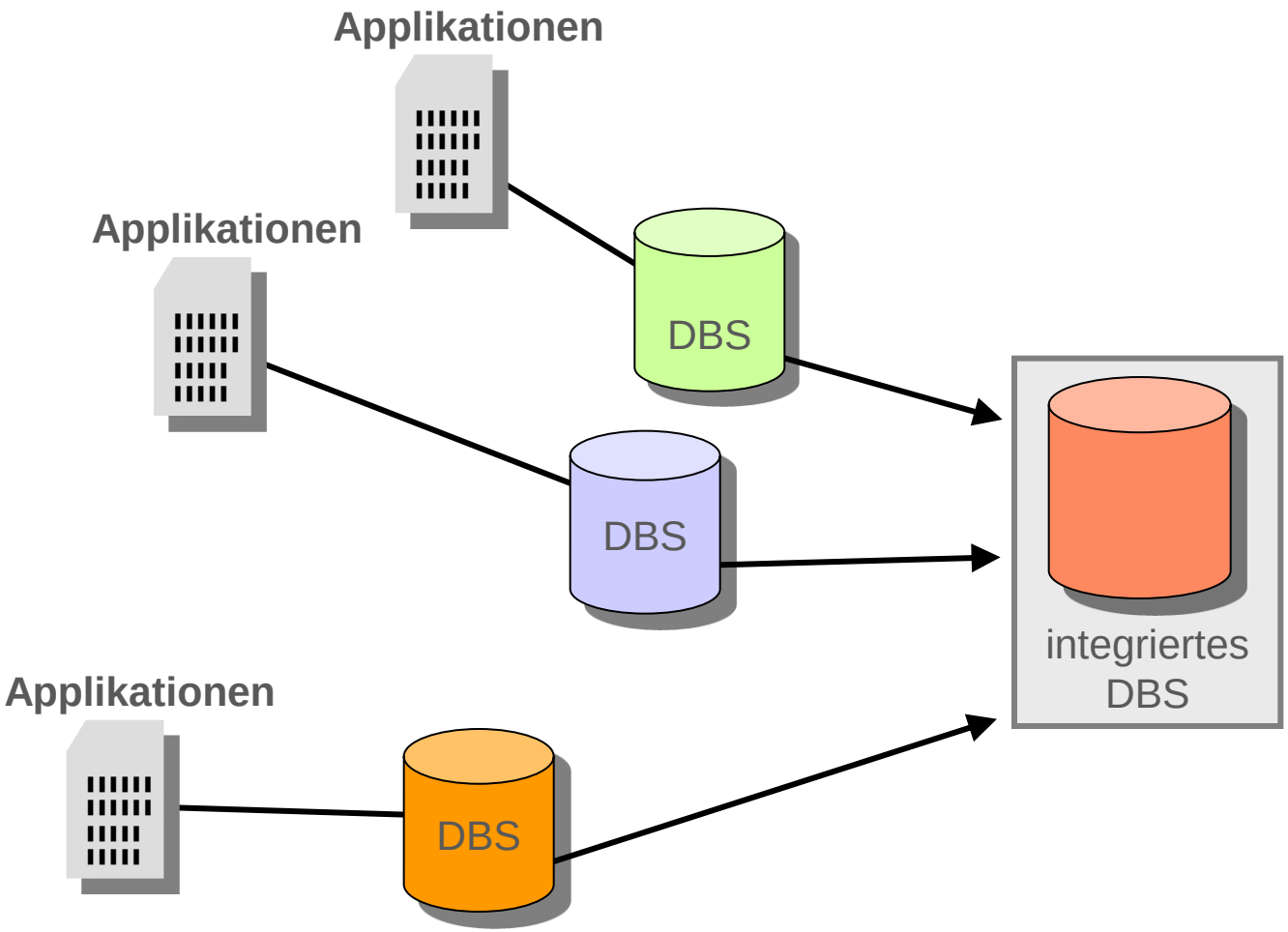
Datenbanksysteme wurden erfunden um Redundanz und die damit verbundenen Probleme zu überwinden

Erweiterter Integrationsbedarf

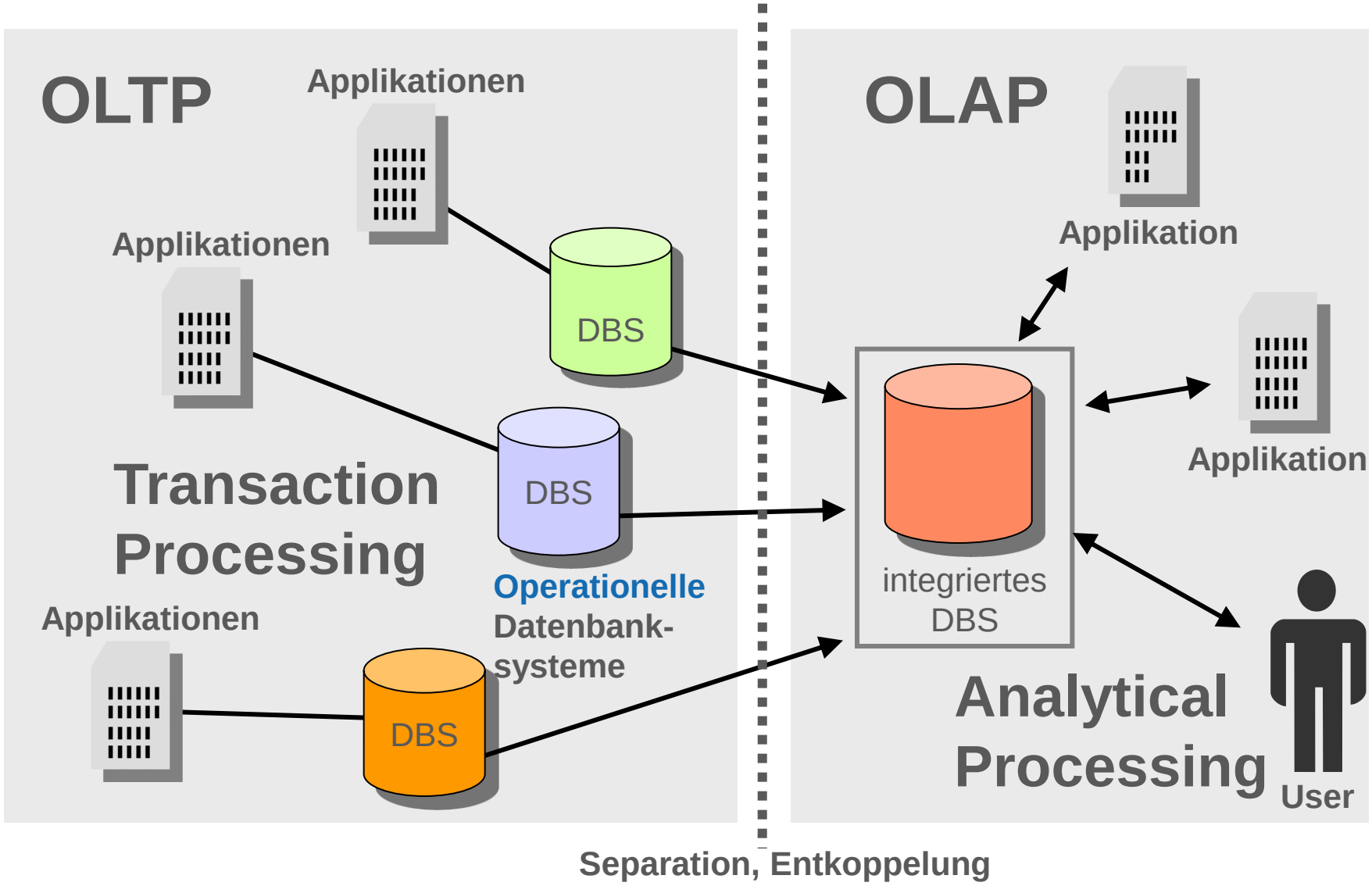
Gründe für weitere Datenintegration

- "Unternehmenssicht" auf die Datenanalyse
  - Datenabfrage in allen Unternehmensbereichen
- Abdecken von Geschäftsprozessen
  - betreffen mehrere DB-Anwendungen
- Änderungen in der Organisation
  - Umstrukturierungen
  - Firmenübernahmen
- Änderungen im Markt

Datenbankintegration



OLTP und OLAP



Online Analytical Processing

OLAP = **O**n**L**ine **A**nalytical **P**rocessing

- online
  - real time
  - im Dialog mit dem Anwender, ad hoc queries
  - Ergebnisse werden sofort generiert
    - im Unterschied zu batch processing
- analytical
  - für Analysezwecke speziell aufbereitet

Dual System Approach

- "clash of cultures"

	OLTP	OLAP
Auslastung	planbar, konstant hoch	unbestimmt, erratisch
Transaktionen	kurz, oft schreibend	lang, nur lesend
Performance-anforderungen	strikt	wechselnd
Anfragen	einfach strukturiert	komplex

- → Forderung nach Unabhängigkeit der Analyse-Datenbank

Strikte Trennung gegenüber OLTP

- OLAP getrennt vom Transaction Processing
  - darf geschäftskritische Prozesse nicht stören
- eigenes Datenbanksystem
  - Daten werden von den operationellen Datenbanksystemen bezogen

andererseits:

- Datenanalyse auf den operationellen Datenbanken hat auch Vorteile
  - kein Kopieren von den Quellsystemen
  - Daten sind stets aktuell

kaum verbreitet

Datenbankanfragen auf Views

- häufiger Gebrauch von Datenbank-Views
  - für viele Einzelanwender
  - für viele unterschiedliche Anwendungen
- Views meist vorberechnet
  - "materialized views"
  - aus Performance-Gründen
- nächster logischer Schritt
  - vorberechnete Views auslagern → separate DB
  - um die Performance im operationellen System zurückzugewinnen

Data Warehouses

- separater Datenspeicher: Data Warehouse
  - "Sekundärdatenbank"
- problematische Übersetzung
  - Warehouse = Lagerhaus
  - Data Warehouse = *Datenlager*
- Definition:
  materialisiertes Datenintegrationssystem für Analysezwecke

Schlussfolgerungen

gefordert:

- Datenhaltung für bestmögliche Unterstützung analytischer Anfragen

→ *besondere Form der Speicherung (und Abfrage)*

- Anfragen auf Datenbestände mehrerer Quelldatenbanken

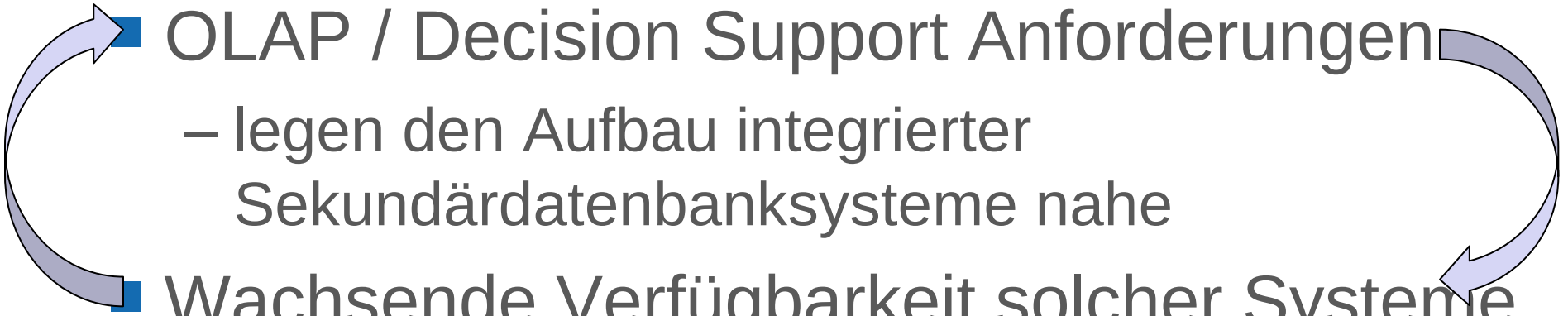
→ *integriertes Datenbanksystem*

- keine Beeinträchtigung des Transaction Processing

→ *unabhängige Sekundärdatenbank*

DSS, OLAP und DWH

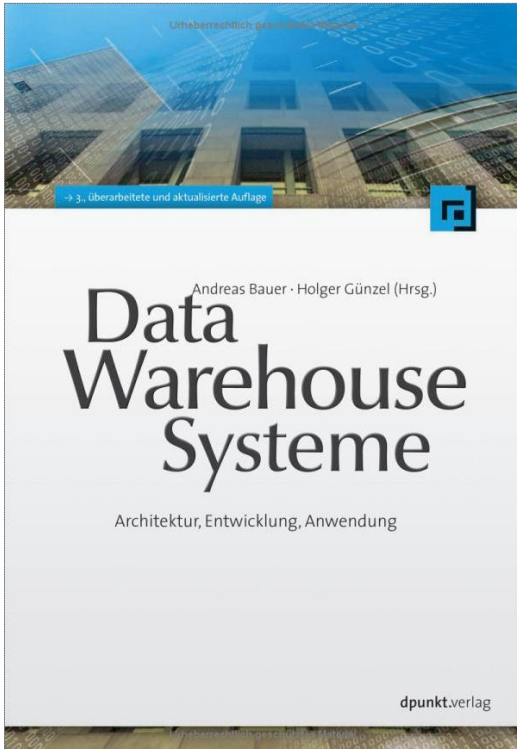
Gegenseitige Verstärkung



- OLAP / Decision Support Anforderungen
  - legen den Aufbau integrierter Sekundärdatenbanksysteme nahe
- Wachsende Verfügbarkeit solcher Systeme
  - wiederum erleichtern OLAP und Decision Support-Anwendung, erhöhen so die Nachfrage

Weiterführende Literatur

für Interessierte:



BAUER, A. / GÜNZEL, H. (Hrsg.)
Data Warehouse Systeme
Architektur, Entwicklung, Anwendung
3. Aufl., dpunkt-Verlag, Heidelberg: 2009

Ausblick

- ☑ Von Decision Support zu Data Warehouses
- ■ **OLAP – Online Analytical Processing**
- ☐ Data Mining – Klassifikation
- ☐ Data Mining – Assoziationsanalyse
- ☐ Data Mining – Clustering

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

DWH: OLAP – Online Analytical Processing

Datenbanksysteme 2

Dr. Klaus Wolfertz

11. Mai 2010

Überblick

- ☑ Von Decision Support zu Data Warehouses
- ■ **OLAP – Online Analytical Processing**
- ☐ Data Mining – Klassifikation
- ☐ Data Mining – Assoziationsanalyse
- ☐ Data Mining – Clustering

Agenda

OLAP – Online Analytical Processing

- **Multidimensionales Modell**
- Relationales OLAP - ROLAP
- Betrieb des Datawarehouse

Schlussfolgerungen

gefordert:

- Datenhaltung für bestmögliche Unterstützung analytischer Anfragen

→ *besondere Form der Speicherung (und Abfrage)*

- Anfragen auf Datenbestände mehrerer Quelldatenbanken

→ *integriertes Datenbanksystem*

- keine Beeinträchtigung des Transaction Processing

→ *unabhängige Sekundärdatenbank*

Anforderungen der Datenanalyse

- Datenauswertungen nach vielen unterschiedlichen Merkmalen denkbar
  - Verkäufe von Mineralwasser in Zürich
  - Gesamter Absatz an Getränken
  - Verkäufe am Wochenende vom 08./09. Mai
- Speichern aller Kombinationen **un**möglich
- → Vorhalten auf granularer Ebene
  - Daten erst zur Abfragezeit aggregiert und je nach Anforderung

Fakten und Dimensionen

- Aggregation
  - von Kennzahlen
  - Mengenattribute (Verkaufszahlen, Beträge, ...)
  - genannt "**Fakten**" ("Facts" / "Measures")
- Kriterien für die Aggregation
  - qualifizierende Attribute (Filiale, Kunden-Nr., Währung, Datum, ...)
  - genannt "**Dimensionen**"

Multidimensionale Ablage

- Mehrdimensional
  - Kennzahlen (Fakten) werden gespeichert für den kompletten Satz an Dimensionen

  Beispiel:

Dimensionen		Fakt	
Kunden_Nr	4711	Betrag	150'000.--
Produkt	Hypothek		
Filiale	Zürich Wiedikon		
Kundenberater	Meier		
Laufzeit	18 Monate		
Währung	CHF		
Datum	10-May-2010		

Aggregation von Kennzahlen

- Fakten können jetzt aggregiert werden
  - Verkäufe von Mineralwasser in Zürich
    - **SUM** Verkauf **WHERE** Artikel-Dimension = "Mineralwasser" **AND** Filial-Dimension = "Zürich"
  - Gesamter Absatz an Getränken
    - **SUM** Verkauf **WHERE** Artikel-Dimension = "Getränke" (schliesst "Mineralwasser" mit ein!)
  - Verkäufe am Wochenende vom 08./09. Mai
    - **SUM** Verkauf **WHERE** Zeit-Dimension = "08-May-10" **OR** "09-May-10"
- restliche Dimensionen jeweils ignoriert

Multidimensionales Datenmodell

Einführung eines speziellen Datenmodells

- Kennzahlen auf granularer Ebene gespeichert
- Auswertungskriterien in mehreren (*multi-*) Dimensionen gespeichert
  - Auswertungskriterien = Dimensionen = beschreibende Informationen zu jeder Kennzahl

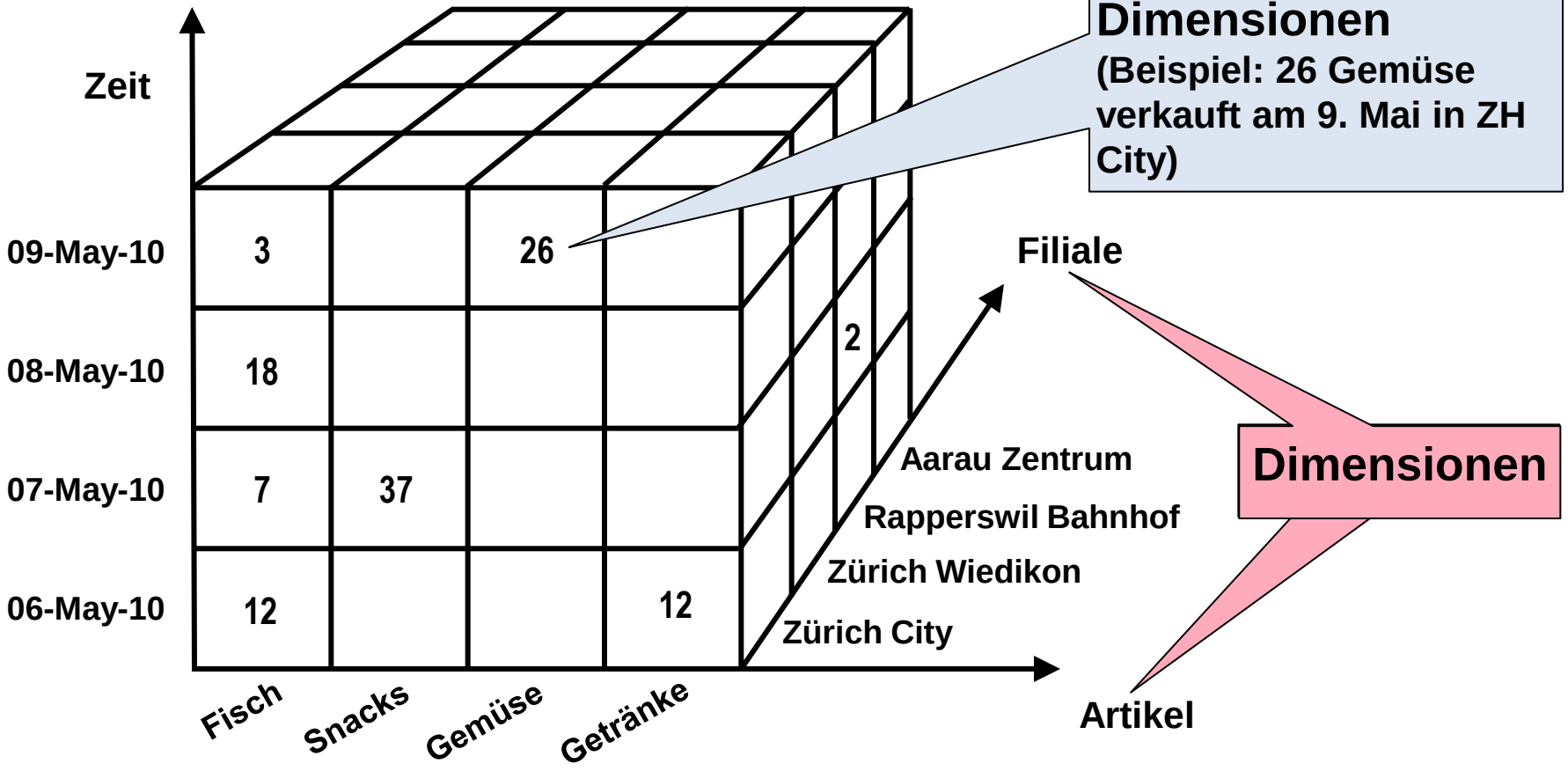
Multidimensionale Abfragen

Multidimensionales Datenmodell

- ermöglicht 1- bis n-dimensionale Abfragen
  - mit n = Zahl der angelegten Dimensionen
- ermöglicht Abfragen für jede Kombination von Dimensionen
- jede Kennzahl eindeutig beschrieben
  - durch die Werte in den Dimensionen

Daten in Würfeln

- Modell veranschaulicht durch Daten"würfel"



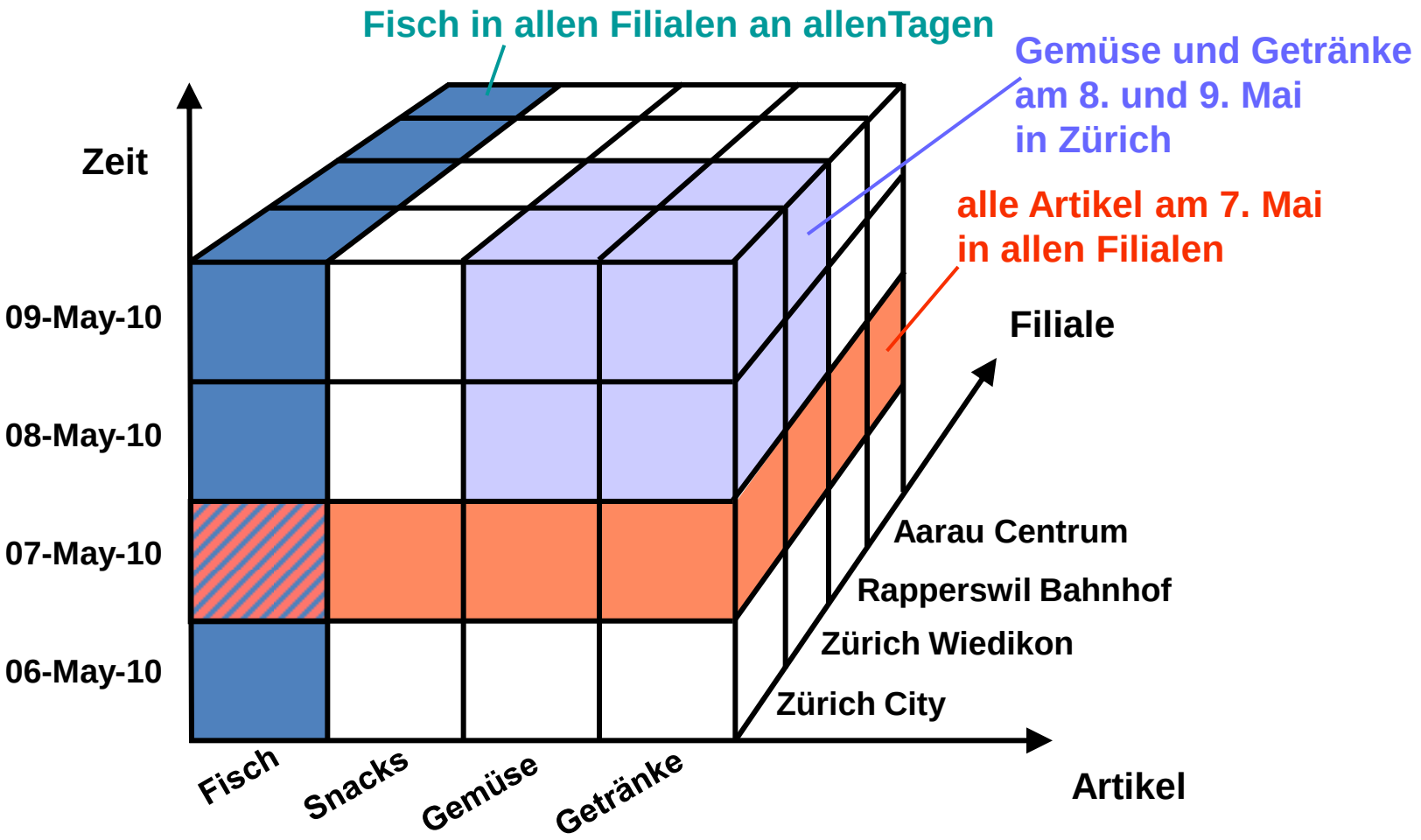
spezielle OLAP-Operationen

Auf den Würfeln werden spezielle Operationen ausgeführt:

- Slice-and-Dice
  - Erzeugen bestimmter Sichten auf Datenwürfel (Scheiben und Teilwürfel)
- Drill-down
  - Verfeinern entlang einer Würfeldimension
- Roll-up
  - Aggregieren entlang einer Würfeldimension

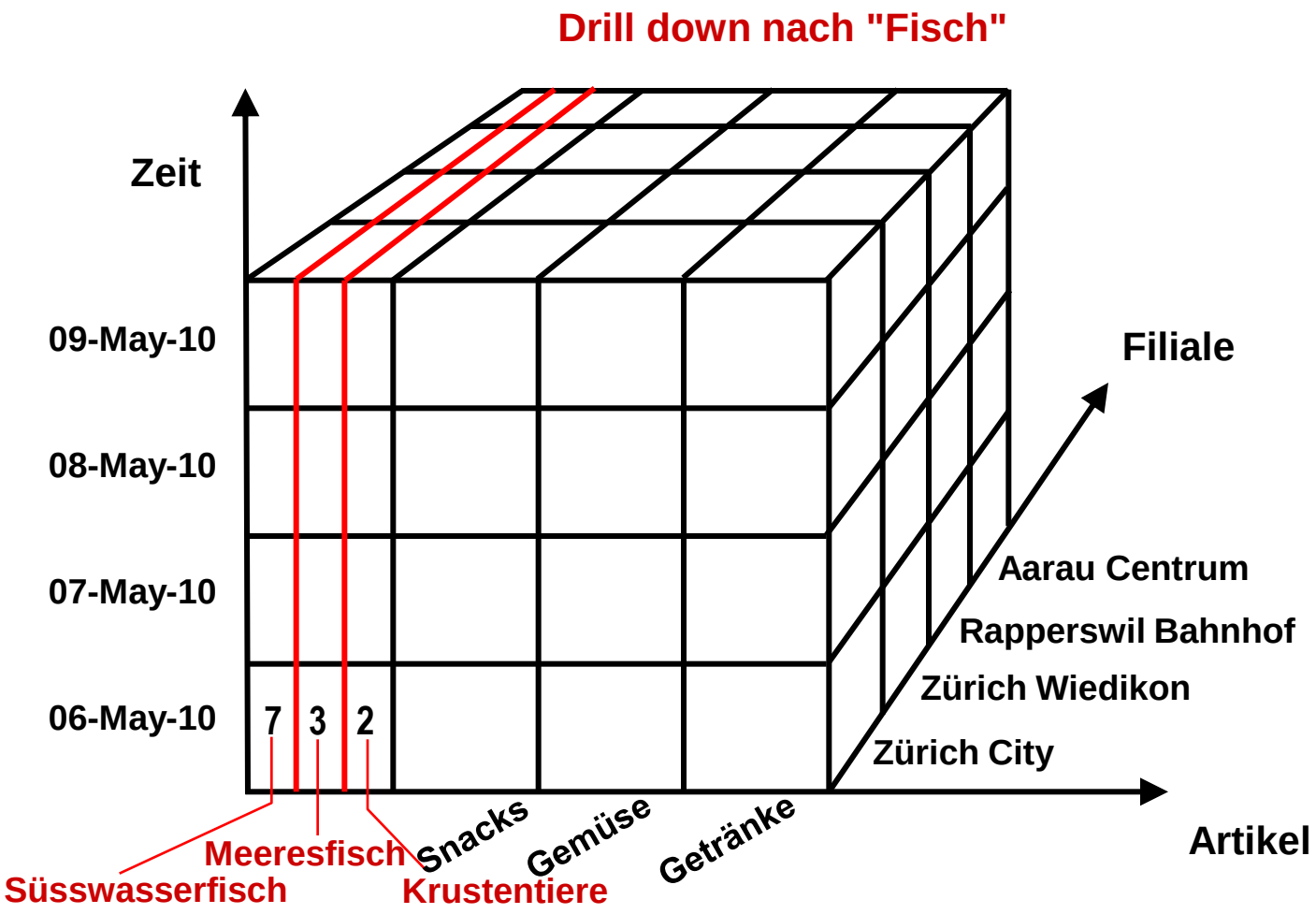
spezielle OLAP-Operationen

- Slice-and-Dice



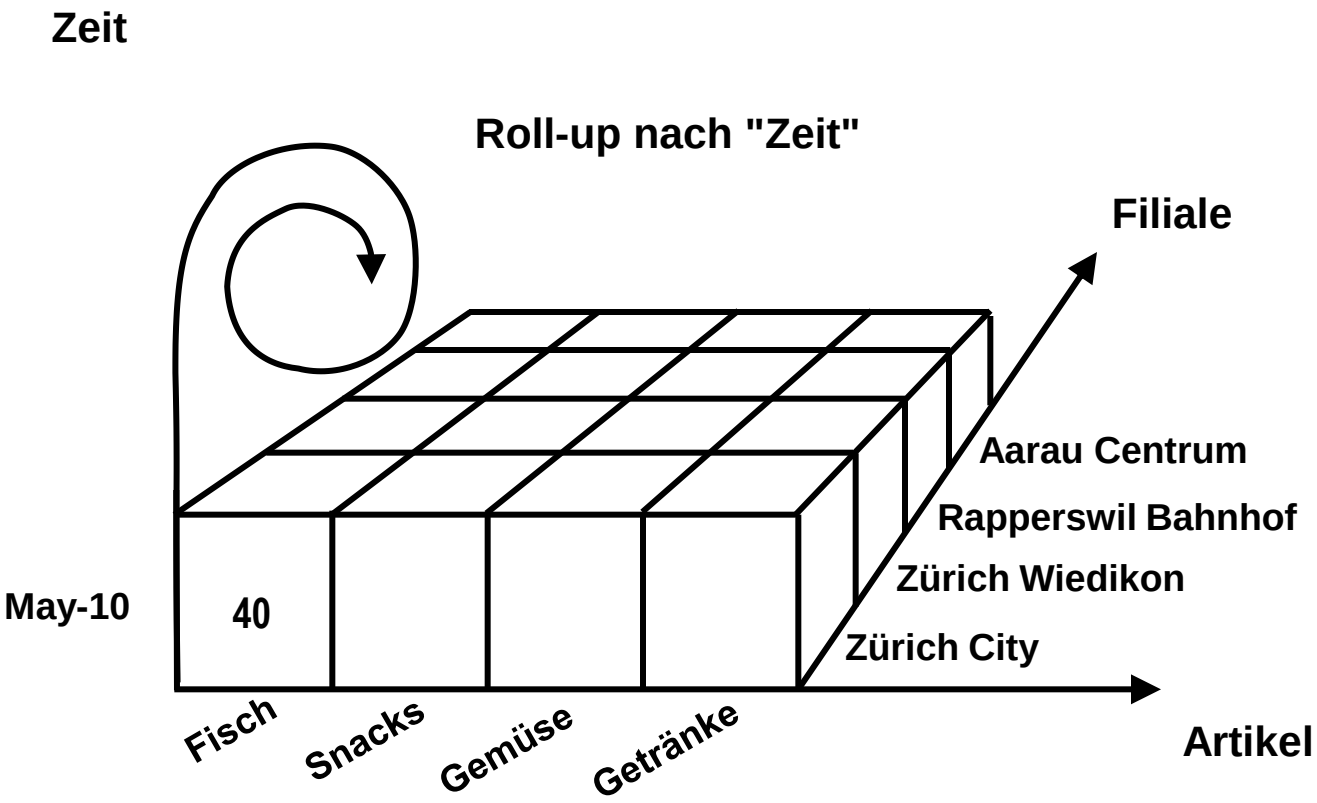
spezielle OLAP-Operationen

- Drill-down



spezielle OLAP-Operationen

- Roll-up



SQL-Support

SQL:1999 unterstützt Arbeiten mit Würfeln zu einem gewissen Grad

- CUBE
  - berechnet alle 2^n Aggregationen für n Dimensionen
- ROLLUP
  - aggregiert entlang von Dimensionen

Syntax

- CUBE

```
SELECT product, shop, SUM(sales)
  FROM t_sales
 WHERE year = 2009
 GROUP BY CUBE(product, shop)
```

- ROLLUP

```
SELECT product, shop, SUM(sales)
  FROM t_sales
 WHERE year = 2009
 GROUP BY ROLLUP(product, shop)
```

Attention: Oracle Syntax. Might be slightly different with other DBMS

Ergebnis des CUBE-Operators

- "Super aggregates"
  - zusätzliche Ergebniszeilen werden erzeugt
  - weiter verdichtete Aggregate

product	shop	sum(sales)
Fisch	ZH City	200
Fisch	Aarau	30
Snacks	ZH City	790
Snacks	Aarau	51
	ZH City	990
	Aarau	81
Fisch		230
Snacks		841
		1071

herkömmliche GROUP BY Ergebniszeilen

zusätzliche aggregierte Zeilen (super aggregates)

NULL-Wert

Kreuztabellen

- Der CUBE-Operator liefert alle Grössen für das Aufstellen einer Kreuztabelle (Kontingenztabelle / "cross tab")

  Beispiel für eine Kreuztabelle:

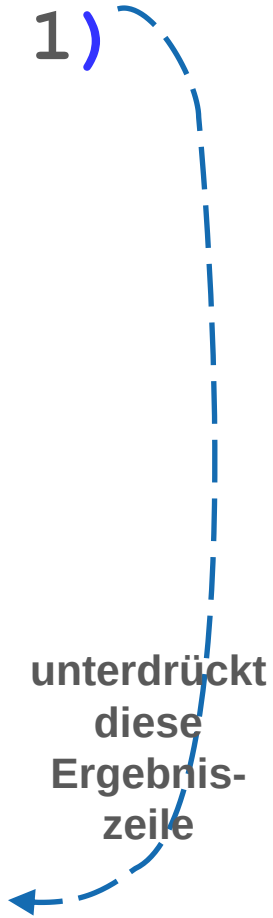
	Aarau	ZH HB	ZH City	ZH Wdk.	Rappi	total
Fisch	2085	1779	11511	1927	4264	**21566**
Snacks	1314	1341	18621	1437	4445	**27158**
Gemüse	1146	1272	2587	1307	1946	**8258**
Getränke	1443	1293	7350	1386	5588	**17060**
total	**5988**	**5685**	**40069**	**6057**	**16243**	**74042**

GROUPING-Funktion

- Zur Identifizierung bestimmter "super aggregates"

```
... HAVING NOT (GROUPING(product)= 1
           AND GROUPING(shop)= 1)
```

product	shop	sum(sales)	grouping (product)	grouping (shop)
Fish	ZH City	200	0	0
Fish	Aarau	30	0	0
Snacks	ZH City	790	0	0
Snacks	Aarau	51	0	0
	ZH City	990	1	0
	Aarau	81	1	0
Fish		230	0	1
Snacks		841	0	1
		1071	1	1



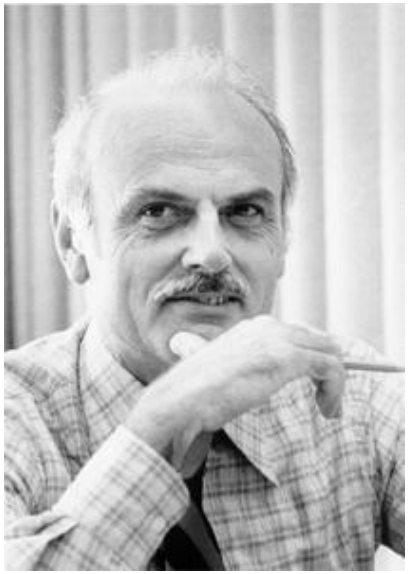
Agenda

OLAP – Online Analytical Processing

- Multidimensionales Modell
- **Relationales OLAP - ROLAP**
- Betrieb des Datawarehouse

Ursprünge von OLAP

- E. F. Codd
  - "Vater" der relationalen DB (1970)
  - die "12 Regeln" (1993): Evaluierungskriterien für die OLAP-Fähigkeit von Software (gemünzt auf "Essbase" – **E**xtended **S**pread**s**heet Data**base**)
- N. Pendse / R. Creeth
  - **FASMI** – **F**ast **a**nalysis of **s**hared **m**ultidimensional **i**nformation (1995)
  - produktunabhängig, einfach



Edgar Frank Codd
1923 - 2003

FASMI – die Definition von OLAP

- **F**ast
  - Anfragen < 5 s, komplexe max. 20 s
- **A**nalysis
  - intuitive Analysemöglichkeiten
- **S**hared
  - Benutzer > 1, Autorisierung, Authentifizierung
- **M**ultidimensional
  - Multidimensionale konzeptuelle Sicht auf Daten
- **I**nformation
  - Analyse unbeschränkt durch OLAP-System

Datenbanken für OLAP

- OLAP setzt geeignete Datenhaltung voraus

2 Implementierungsansätze:

- **MOLAP**
  - **M**ultidimensional **OLAP**
  - Speicherung in n-dimensionalen Arrays
  - so genannte multidimensionale Datenbanken
- **ROLAP**
  - **R**elational **OLAP**
  - Speicherung in relationalen Tabellen

MOLAP

Speicherung in n-dimensionalen Arrays

- Dimensionen und Fakten unterschiedlich
- Fakten werden in Arrays gespeichert
  - sequentiell
    - "Linearisation" der Daten notwendig
- jede Dimension ist ein Index
  - auf den Array
  - Kenntnis des Dimensionswertes erlaubt Bestimmung des Speicherorts im Array

ROLAP

Speicherung in relationalen Tabellen

- Dimensionen in je einer relationalen Tabelle
- Fakten in einer getrennten Tabelle
  - "Faktentabelle" (fact table)
  - jeder Eintrag in der Faktentabelle verweist auf die zugehörigen Werte in den Dimensionstabellen (Fremdschlüssel)
  - Primärschlüssel der Faktentabelle ist die Konkatenation aller Fremdschlüssel
    - jedes Faktum ist eindeutig bestimmt

Vergleich

- MOLAP
  - schnell
  - skaliert schlecht
- ROLAP
  - bewährte Technologie, Know-how verbreitet
  - ständige Transformation notwendig multidimensionales ←→ relationales Modell
  - für die meisten Anfragen Joins zwischen Faktentabelle und Dimensionen nötig

ROLAP - Beispiel

**Es gibt einen neuen Artikel:
Kaviar, Art-Nr. C007
Wir beziehen ihn von "Lakeside" und verkaufen sofort 12 Einheiten**

SUPPLIER_DMSN

Supplier_ID	Name	Place	Phone
FMX	Fish Max	Uster	044 955 57 00
LAK	Lakeside	Horgen	043 311 80 70
TRO	Trout King	Einsiedeln	055 412 13 14

ARTICLE_DMSN

ArtNr	Description	Category	Price
123	Trout	Fish	11.70
139	Egli	Fish	12.90
220	Red Snapper	Fish	17.50
589	King Prawns	Fish	26.00
CU-09	Cucumber	Vegetables	0.80
ZW701	Pretzels	Snacks	1.95
C007	Caviar	Fish	160.00

SALES_FACT

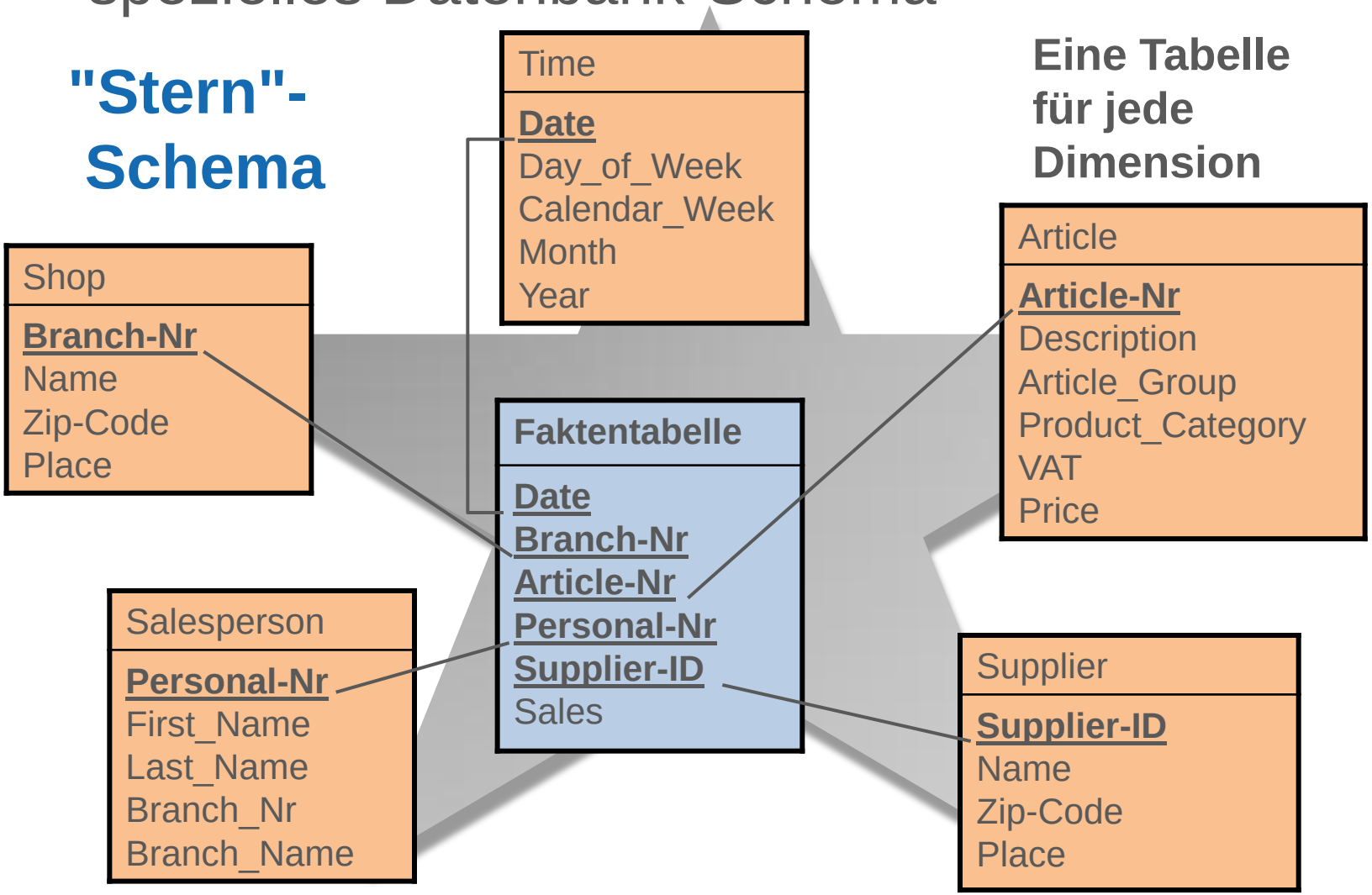
Article_ID	Supplier_ID	Sales
139	FMX	20
123	TRO	40
220	FMX	75
123	LAK	10
123	FMX	200
139	FMX	37
C007	LAK	12

Implementierung mit relationalem DBS

- spezielles Datenbank-Schema

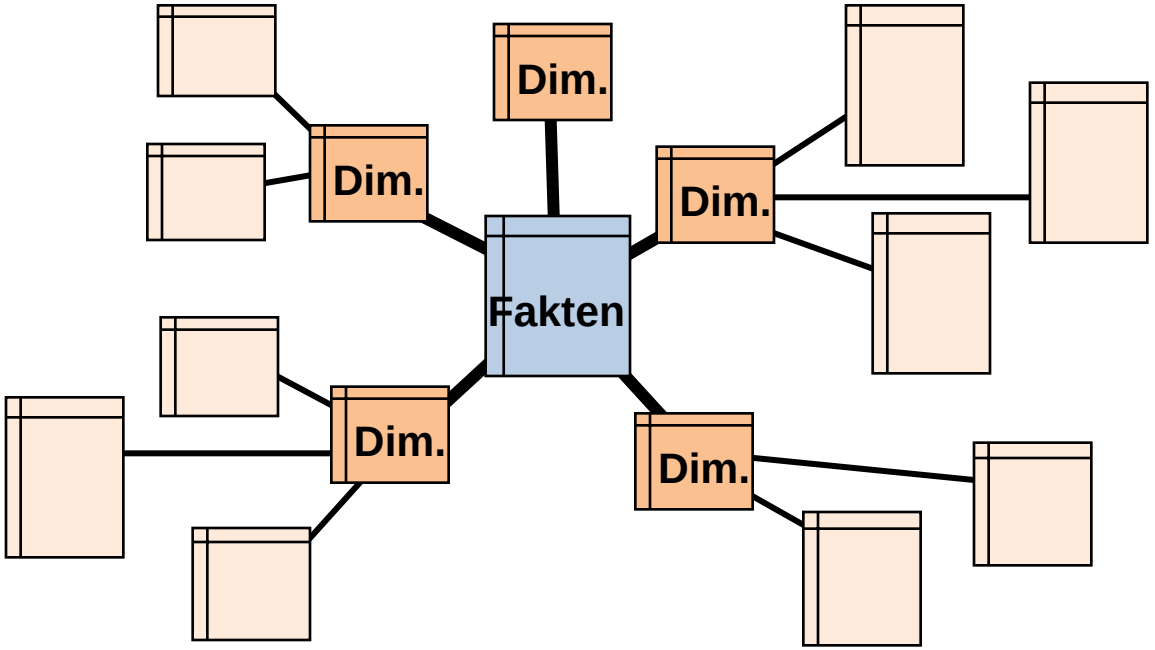
"Stern"-Schema

Eine Tabelle für jede Dimension



Normalisierung

- Stern-Schema
  - Dimensionstabellen denormalisiert → Performance
- Snowflake-Schema
  - Dimensionen normalisiert → Redundanzvermeidung



ROLAP - Abfrage

- Fakten enthalten die Kennzahlen
  - und die Fremdschlüssel der Dimensionen, wie z.B. die Artikel-Nr.
- Beschreibende Attribute in den Dimensionen
  - z.B. Preis, Mehrwertsteuersatz, Farbe, Lieferant
- Konsequenz: Join notwendig (teuer!)
  - oft über mehrere Tabellen
    - wenn beschreibende Attribute verschiedener Dimensionen verlangt, z.B. Preis, Kundensegment, Herkunftsland des Lieferanten, Kalenderwoche etc.

Dimensionshierarchien

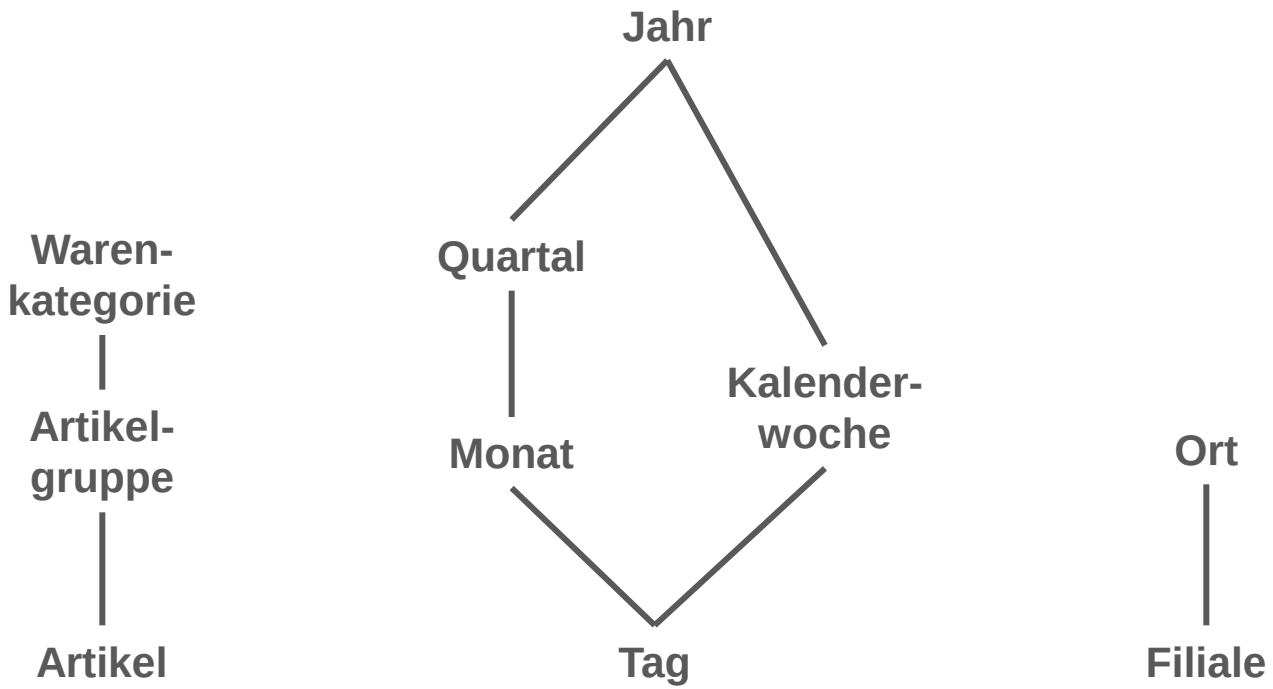
Beispiel: Gesamtverkauf an Getränken

- **`SUM`** Verkauf **`WHERE`** Artikel-Dimension = "Getränke" (schliesst "Mineralwasser" mit ein)

- Dimensionen haben typischerweise eine "Konzepthierarchie"
  - notwendig für "drill down"
  - z.B. "Valser" ist enthalten in "Mineralwasser" ist enthalten in "Getränke" ist enthalten in "Lebensmittel" auf der Artikel-Dimension

Konzepthierarchien

- sind unterschiedlich tief
- können sich verzweigen und vereinen



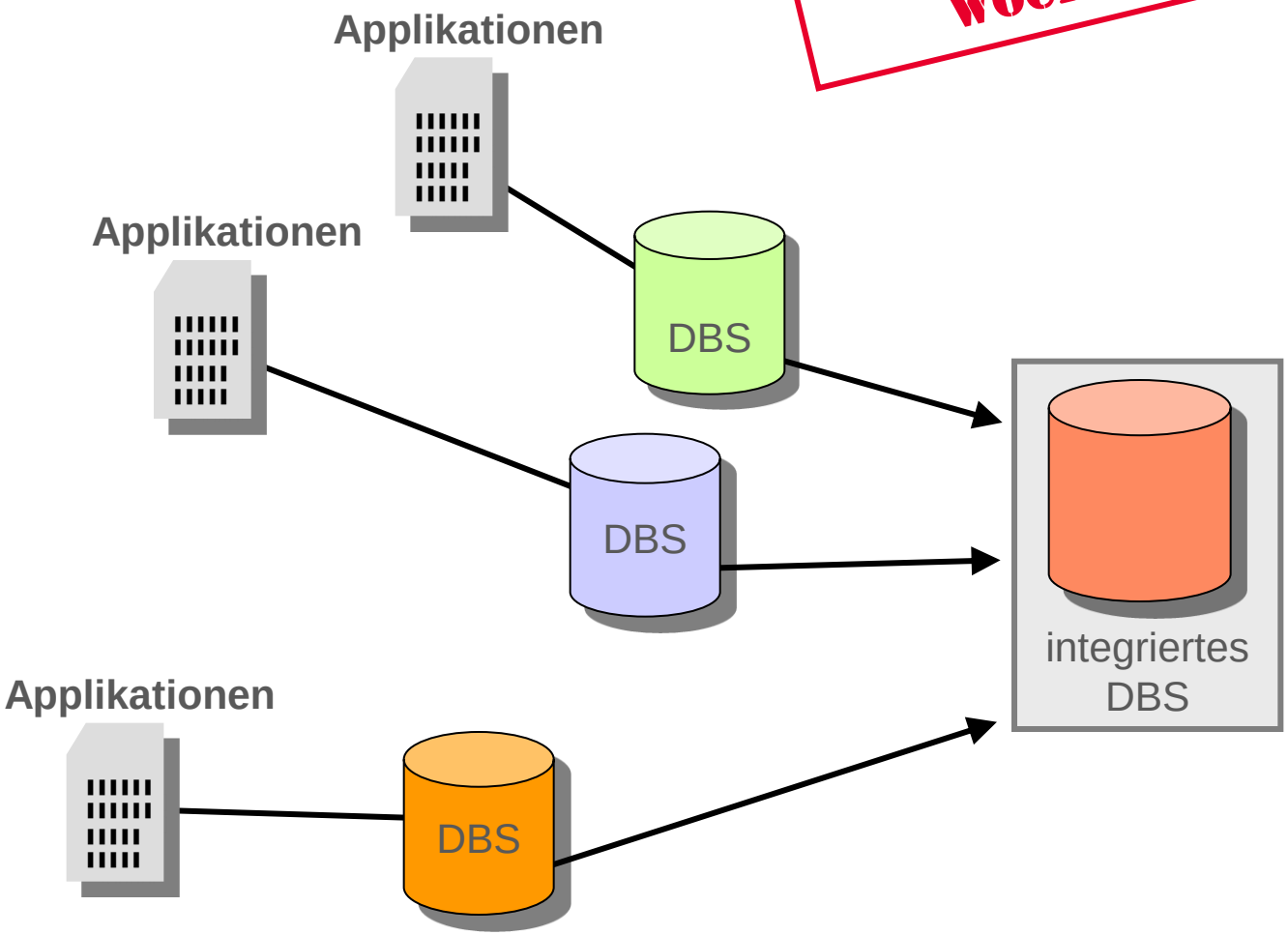
Agenda

OLAP – Online Analytical Processing

- Multidimensionales Modell
- Relationales OLAP - ROLAP
- **Betrieb des Datawarehouse**

Datenbankintegration

VON LETZTER WOCHE

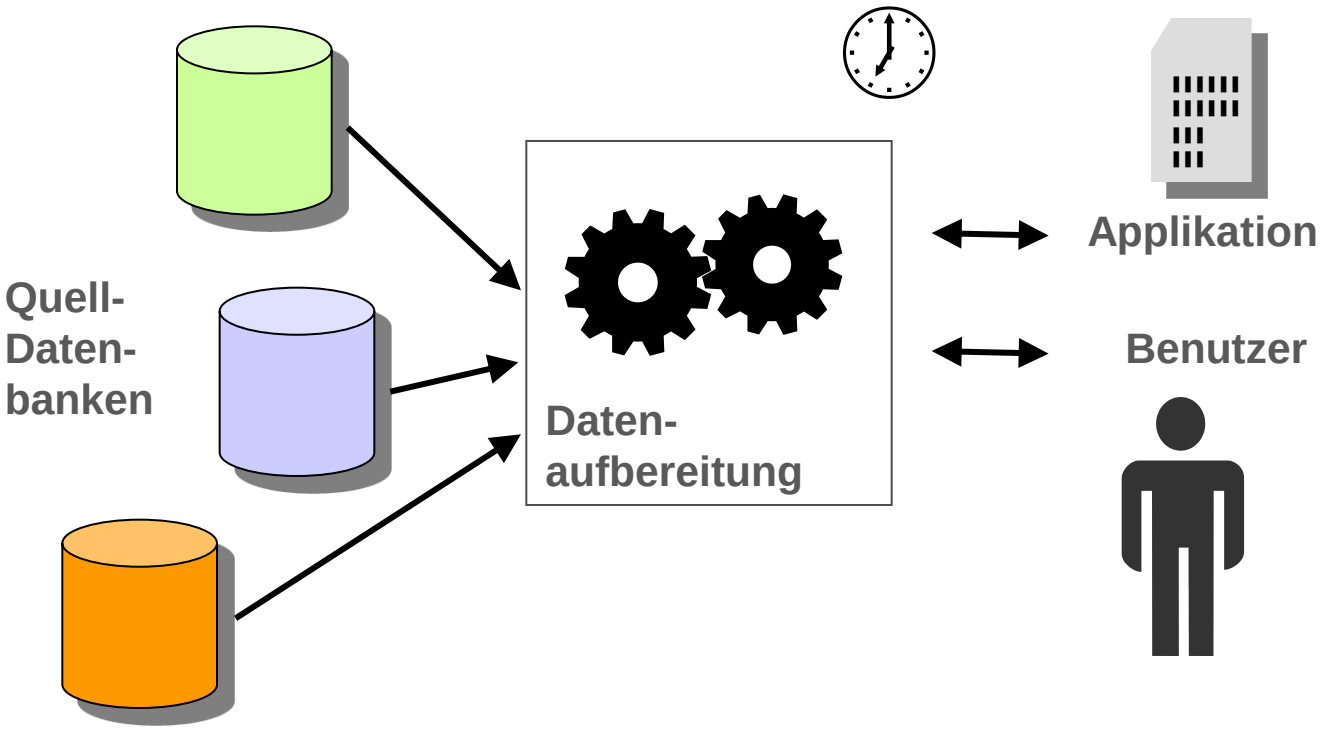


Formen der Datenbankintegration

- virtuelle Systeme
  - temporär
  - Integration zum Abfragezeitpunkt
- materialisierte Systeme
  - permanent
  - Integration vor dem Abfragezeitpunkt

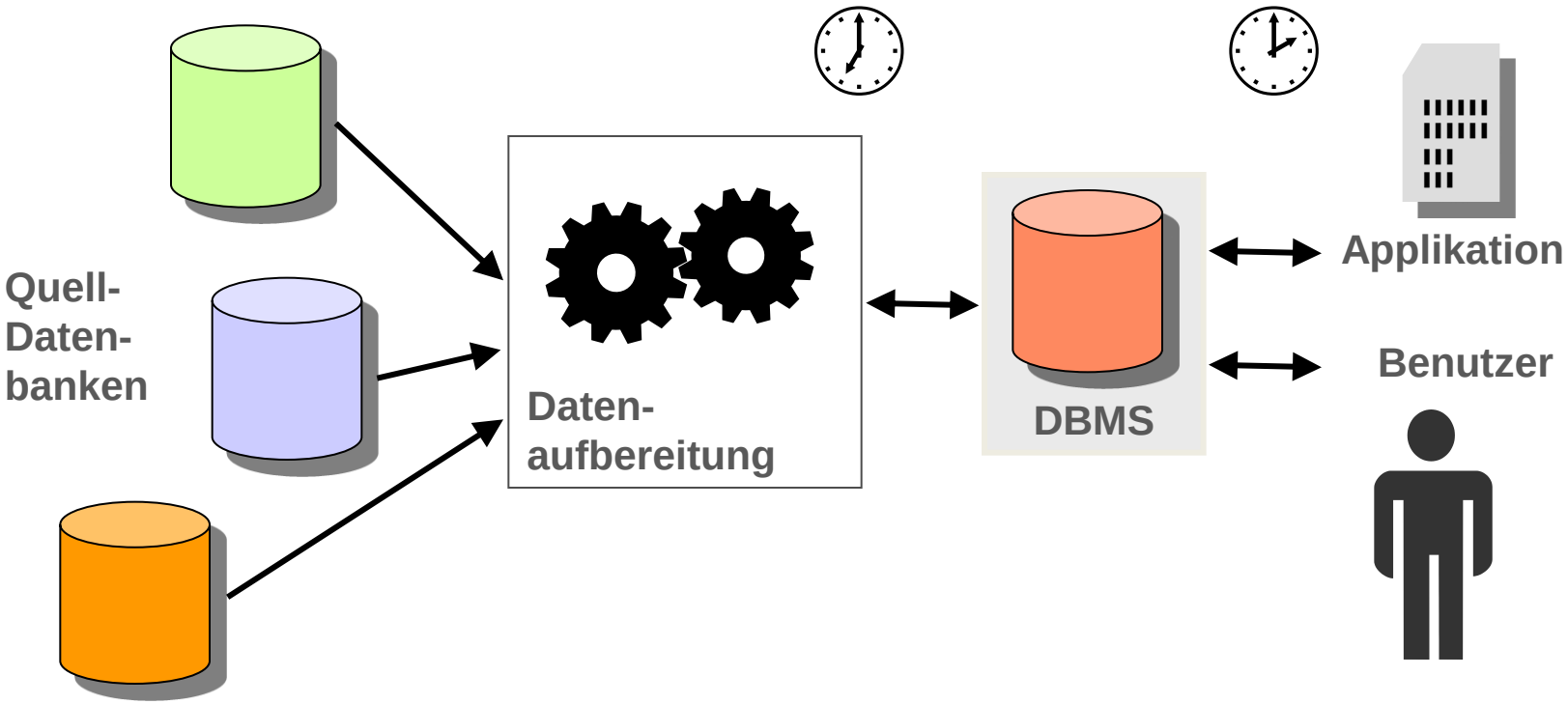
Virtuelle integrierte Datenbanksysteme

- Datenaufbereitung direkt für Anwendungen und Dialoge



Materialisierte integrierte Datenbanksysteme

- Aufbereitete Daten in einer separaten Datenbank gespeichert



Integrierte Datenbanksysteme

Kriterien für die Form der Datenintegration

- Verbindung Netzwerk
- Nutzerorientierung (Antwortzeitverhalten)
- Wartung
- Synchronisationsbedarf
- Aktualität
- Quelldatenbank(en):
  - Performance
  - operative Bedeutung
  - Sicherheitsempfindlichkeit

Data Warehouses

Data Warehouses als spezielle Form materialisierter Datenintegrationssysteme

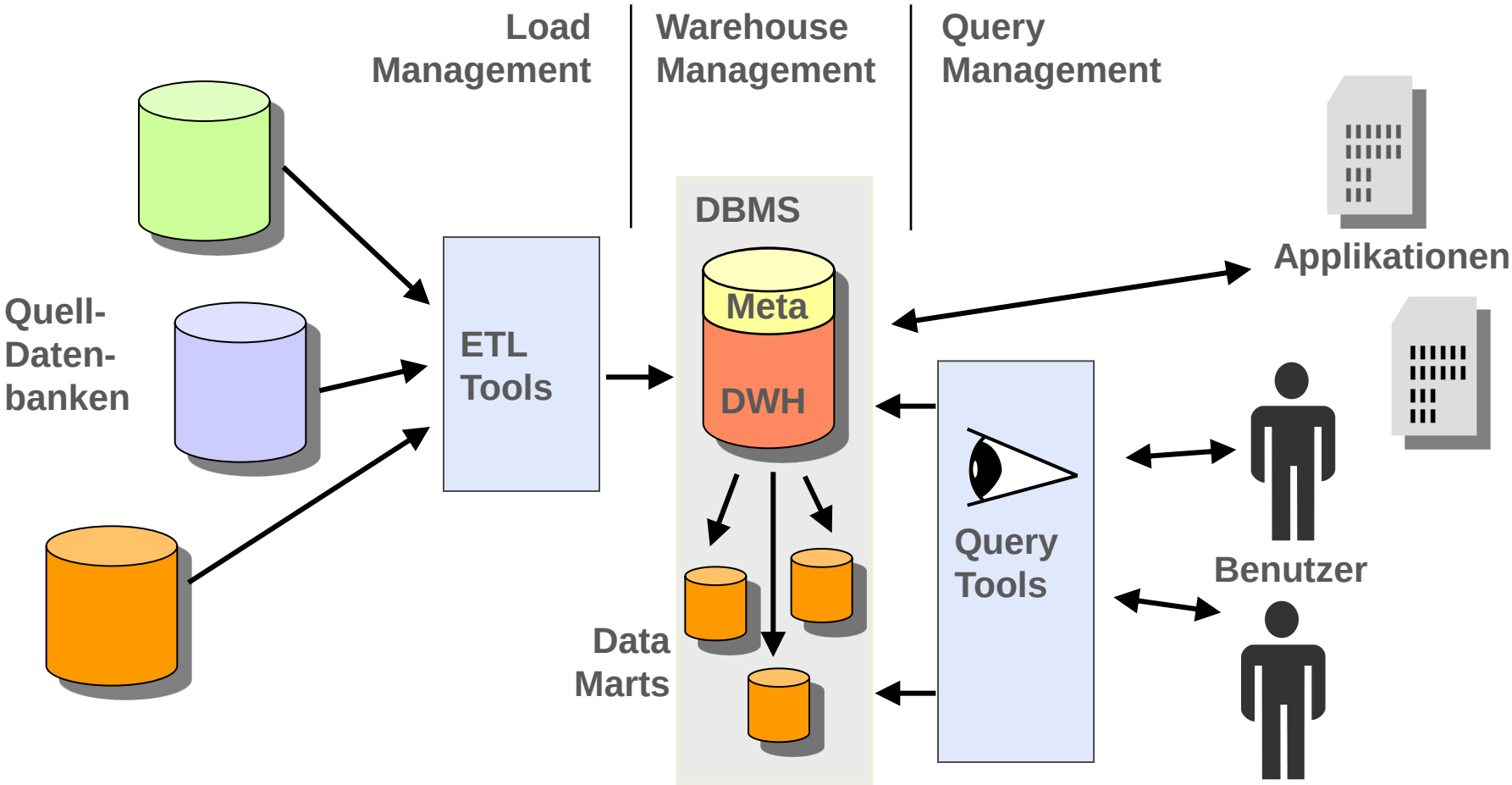
- Verwertung von im operationellen Betrieb angefallenen Daten
- Verfügbarhaltung von "historischen" Daten
- Herstellen einheitlicher Datenqualität
- Aufbau gezielt für Decision Support
  - spezifische Auswahl und Aufbereitung der Quelldaten

Data Warehouse Layout

- Sekundärdatenbank
  - aus einer oder mehreren operativen Datenbanken erzeugt
- separate "Meta"-Datenbank
  - mit allen Informationen zu Aufbau und Struktur
  - das Data Warehouse Data Dictionary
- Data Marts
  - themenspezifische Extrakte (für schnelleren Zugriff)
  - nicht-permanent (DWH ist Master)

Data Warehouse Systemlandschaft

- Spezifische Aufgabenbereiche



Data Warehouse Eigenschaften

- Sehr grosse Datenmengen
  - oft im Terabyte (1000 GB)-Bereich
  - rasche Datenzunahme
    - 50% Zuwachs im Jahr nicht aussergewöhnlich
- Endbenutzer haben nur Lesezugriff
  - Keine Kontrolle der Nebenläufigkeit (concurrency control) notwendig
- Schreiben nur beim Laden
  - durch spezielle ETL-Tools
  - in sehr grossen lang dauernden Transaktionen

ETL

Extraction, **T**ransforming and **L**oading

- **Extrahieren** von Daten aus den operationellen Systemen
- **Transformieren** in die vom Data Warehouse geforderte Form
- **Laden** ins Data Warehouse
  - Bewegen sehr grosser Datenmengen in kurzer Zeit
  - periodisch wiederholt (z.B. jede Nacht)

Extraction

- Bezug der Daten in den operationellen Systemen
  - Gefahr der Beeinträchtigung des geschäftskritischen Transaction Processing
- oft als physische Extrakte
  - aus Performanzgründen
- Keine 1:1-Kopie der operationellen Daten
  - spezifische Auswahl für späteren Analysebedarf

Transformation

= Datenkonsolidierung:

- Herstellen einheitlicher Datenqualität
  - über alle Quellsysteme
- Keine 1:1-Kopie der operationellen Daten
  - Summierung, Aggregation, Verdichtung
  - Filterung
  - Umstrukturierung
  - Umformatierung
  - Ergänzungen, Anmerkungen

  spezifisch für späteren Analysebedarf

Loading

- Einfügen ins DWH
  - auch vorformatiert im physischen Format
- Integritätsprüfung
  - vor dem Laden
  - nach dem Laden gegenüber den Daten im DWH (z.B. unique constraints)
- Aufbau, Reorganisation von Indexen
  - zeitaufwändig
  - Ändern ← → Löschen und Neuaufbau

Aktualisierung von Data Warehouses

- updates immer nur in einer Richtung
  - vom Quellsystem zum DWH
  - Datenänderungen (normaler-weise) nie im Data Warehouse
  - Im Fehlerfahl Korrektur im Quellsystem und Warten auf den nächsten Ladeprozess

- Datawarehouse ist ein autonomes System
  - Problem der Synchronisation mit den Quellsystemen
  - nie (ganz) aktuell

Ausblick

- ☑ Von Decision Support zu Data Warehouses
- ☑ OLAP – Online Analytical Processing
- ■ **Data Mining – Klassifikation**
- ☐ Data Mining – Assoziationsanalyse
- ☐ Data Mining – Clustering

Vorbereitung auf Data Mining

- Installation von WEKA
  - auf dem eigenen Rechner
  - für Übungen zuhause
  - Bedienungshinweise zusammen mit Hausaufgaben dann nächste Woche

- Data Mining Software in Java
- Kostenlos
- Download unter
  http://www.cs.waikato.ac.nz/ml/weka/
  (Versionen 3.6 oder 3.4)

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Data Mining: Klassifikation

Datenbanksysteme 2

Dr. Klaus Wolfertz

18. Mai 2010

Überblick

- ☑ Von Decision Support zu Data Warehouses
- ☑ OLAP – Online Analytical Processing
- ■ **Data Mining – Klassifikation**
- ☐ Data Mining – Assoziationsanalyse
- ☐ Data Mining – Clustering

Analyse strukturierter Daten im Hinblick auf unbekannte Muster

- unbekannte Muster
  - Strukturen erkennen, entdecken
  - das Wesentliche herausfiltern
  - Modelle aufstellen
  - Eintreffen von Ereignissen vorhersagen

die 3 wichtigsten Verfahren:

- Klassifikation (heute)
- Assoziationsanalyse (nächstes Mal)
- Clustering (übernächstes Mal)

frühes Beispiel für Data Mining

Johannes Kepler (1571 - 1630)

- Rudolphinische Tafeln (Tabulae Rudolphinae)
- Berechnung der Planetenbahnen
- Auswertungen der von Tycho Brahe gesammelten Daten

TABVLÆ
RVDOLPHINÆ,
QVIBVS ASTRONOMICÆ SCIENTIÆ, TEMPO-
rum longinquitate collapsæ RESTITVERE illio conditore,
A Phœnicelio Astronomorum
TYCHONE,
Ex Illustrique Generosâ Thraeorum in Regno Dania
familia oriundo Equite,
PRIMUM ANIMO CONCEPTA ET DESTINA-
TA ANNO CHRISTI MDLXIV: EXINDE OBSERVATIONIBUS
SIDERUM ACCURATISSIMIS, POST ANNUM PRÆCIPUE MDLXXII,
TANDEM TRADUCTA IN GERMANIAM, INQUE AULAM ET
Nomen RUDOLPHI Imp. anno MDIIC.
TABULAS IPSAS, JAM ET NUNCUPATAS, ET AFFECTAS, SED
MORTE AUTHORIS SUI ANNO MDCL. DEBITAS.
JUSSU ET STIPENDIIS FRETUS TRIUM IMPPP.
RUDOLPHI, MATTHIÆ, FERDINANDI,
IOANNES KEPLERUS,
Cum Privilegiis, Imp. & Regum Rerumq; publ. vivo TYCHONI ejusq; Hæredibus,
& speciali Imperatoris, ipsi KEPLERO concesso, ad Annos XXX.
ANNO M. D C. XXVII.

Eigenschaften des Data Mining

- Suchen nach Zusammenhängen, Mustern
  - vorher nicht bekannt, allenfalls vermutet
- Exploratives Vorgehen ("Trial and error")
  - Man weiss nicht genau, wonach man sucht
- Automatisiert
  - zumindest Unterstützung durch Software
- adressieren des Problems wachsender Datenmengen
  - Situation des "data rich but information poor"

Motivation für Data Mining

- wieder: Decision Support
  - Entscheidungen verbessern: Regelmässigkeiten sind erkannt und können berücksichtigt werden
- Grenzen konventioneller Verfahren
  - Datenmenge erfordert besondere Methoden
  - Höhere Reaktionsgeschwindigkeit gefordert
- schiere Menge erlaubt erst Entdeckung
  - in kleineren Mengen nicht sichtbar
  - zumindest weniger gut daraus erschliessbar
- Zusätzliche Wertschöpfung
  - aus - zunächst - nur für operationelle Zwecke generierten Daten

Einordnung und Abgrenzung

- Query and Reporting
  - Abfrage von Daten
    - Anzahl Verkäufe von Produkt X in Filiale Y im Monat Mai
- OLAP
  - Analyse integrierter Daten und über lange Zeiträume
    - Konzernweiter Jahresumsatz mit Kundensegment Z seit 2000
- Data Mining
  - Verstehen und Vorhersage
    - Welche Kundengruppen für Glace gibt es?
    - Woran erkenne ich sofort die Zugehörigkeit?
- Expertensysteme
  - Simulation menschlichen Schlussfolgerns
    - Wenn Aussentemperatur > 30°C und nicht Ferien dann erhöhe Bestellmenge Glace um 15%.

Agenda

Data Mining – Klassifikation

- **Einführung**
- Klassifikation mit Entscheidungsregeln
- Klassifikation mit Entscheidungsbäumen
- Validierung von Klassifizierern

Klassifikation

- was wir hundertmal am Tag machen
- etwas ein"ordnen"
  - wichtige, unwichtige, spam Mails
  - normale und gefährliche Verkehrssituationen
  - leichte, gewohnte, schwierige Aufgaben
  - langweilige und interessante Vorlesungen
  - …
- Beurteilen anhand von bestimmten Eigenschaften, Merkmalen

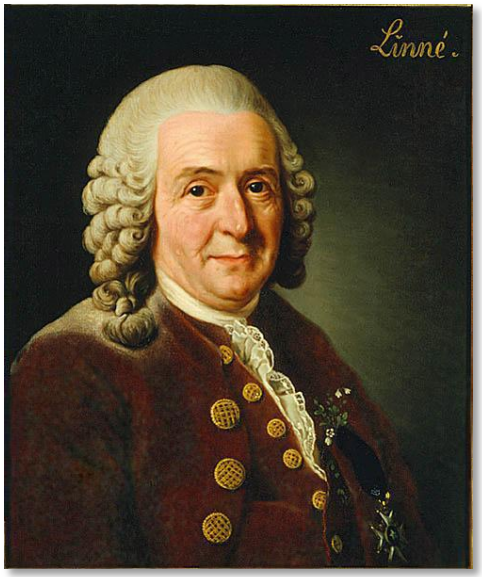
Das Wesen der Klassifikation

- untersuchen eines (bislang unbekannten) Objekts
  - anhand seiner Eigenschaften, Attribute
- *unbekannt* ist welches "Etikett" auf *genau dieses* Objekt gehört
  - die Eigenschaften des Objekts müssen bekannt sein
  - die möglichen Etiketten sind ebenfalls bekannt
- zuordnen zu einer von mehreren "Klassen"
  - einen Aufkleber aufbringen, "etikettieren"
  - "in eine Schublade stecken"

Klassifikationssysteme

- Klassifikationen sind systematische Ordnungen:
  - Bibliothek
  - Wirtschaftszweige
  - Sprachen
  - Warengruppen
  - Krankheiten
  - Tier- und Pflanzenreich
    - Domäne *Lebewesen mit Zellkern*
    - Reich *Tierreich* (Animalia)
    - Abteilung *Gewebetiere* (Eumetazoa)
    - Stamm *Gliederfüsser* (Arthropoda)
    - Klasse *Spinnentiere* (Arachnida)
    - Ordnung *Webspinnen* (Araneae)
    - Familie *Vogelspinnen* (Theraphosidae)
    - Gattung *Theraphosa*
    - Art *Theraphosa blondii*

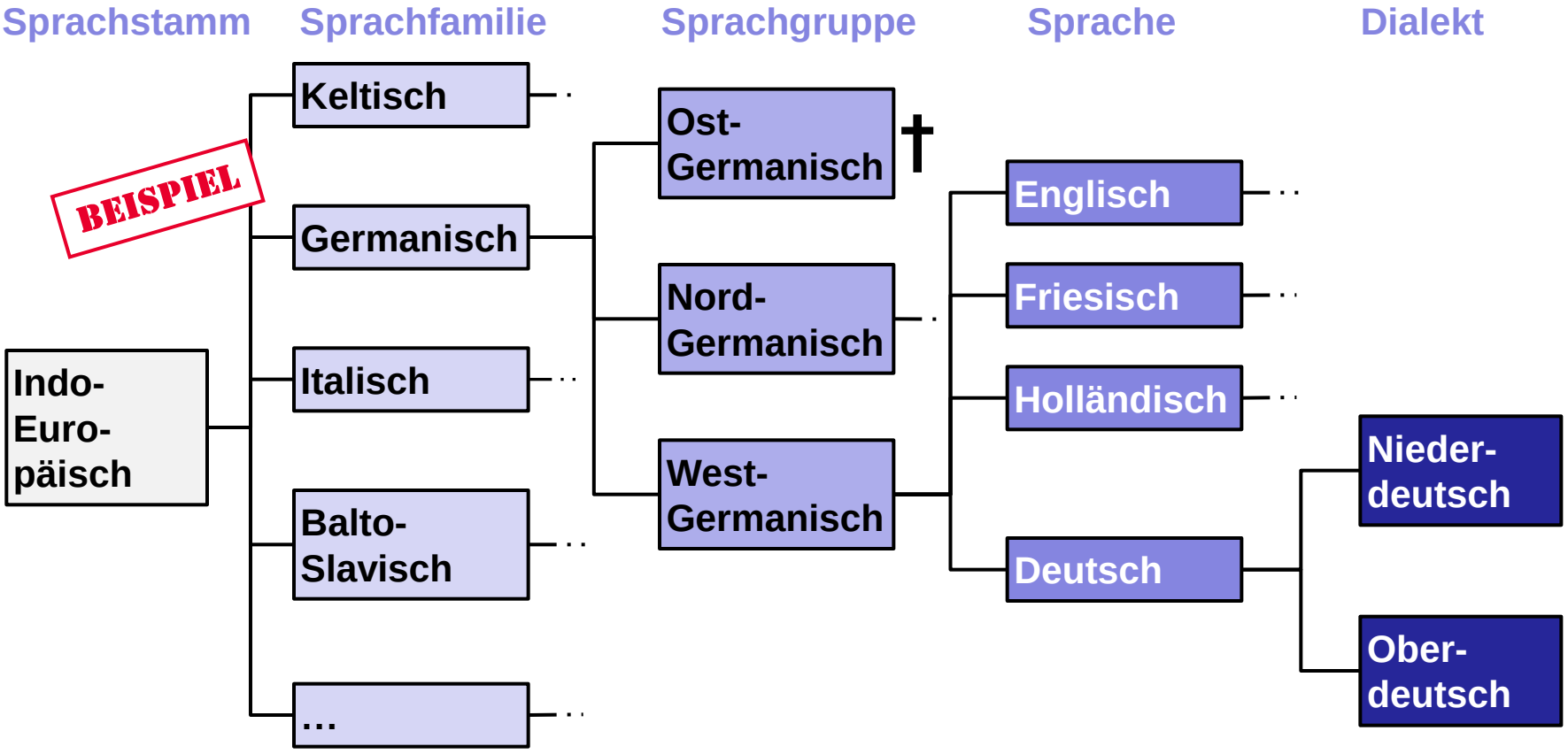
BEISPIEL



Der schwedische Wissenschaftler Carl von Linné erarbeitete im 18. Jh. die noch heute gebräuchliche Einteilung der Tiere in verschiedene Arten

baumartige Hierarchien

- Klassifikationssysteme sind Baumstrukturen
  - eindeutige Zuordnung der Objekte



Klassifikation im Data Mining

- **Vorhersage** einer bestimmten Eigenschaft, eines Attributs
  - eines Onlineshop-Besuchers: Kauft er? Nicht?
  - eines Kreditbewerbers: Wird er zurückzahlen?
  - eines Office-Druckers: Welches Bauteil wird in den nächsten 6 Monaten ausfallen?
- Der Attribut*wert* ist bislang unbekannt
- Der Werte*bereich* ist bekannt = die **Klassen**
- **Bestimmen der definitiven Klasse einer bislang unbekannten Instanz**

Begriffe und Definitionen

- Instanz (Datensatz, Tupel, Objekt, Sample)
  - einzelnes, unabhängiges Beispiel, z.B. eine Kundenbestellung
- Attribut
  - Eigenschaft von Instanzen ("Beruf", "Alter", "Ausbildung" etc.)
- Attributwert
  - bestimmte Ausprägung eines Attributs ("IT Analyst", 37, "Bachelor in Informatik")
- Klassenattribut (class label)
  - Attribut, das für unbekannte Instanzen bestimmt werden soll (z.B. "potentieller Kunde")
- Klasse
  - ein möglicher Wert des Klassenattributs (z.B. "Ja")

Agenda

Data Mining – Klassifikation

- Einführung
- **Klassifikation mit Entscheidungsregeln**
- Klassifikation mit Entscheidungsbäumen
- Validierung von Klassifizierern

Beispielhafte Problemstellung

- Irgendeine Marketingkampagne
  - Kunden werden telephonisch kontaktiert
  - ihnen wird ein neues Produkt und/oder ein Sonderangebot angepriesen

- Wird dieser Kunde kaufen? Oder nicht?
  - dies zu wissen würde viel Zeit und Geld sparen
  - gibt es eine Möglichkeit, das vorauszusagen?

Einbeziehen von Erfahrungswerten

Attribute mit bekannten Werten

Klassenattribut

		Last "regular" order			Campaign
No	**Gender**	**Ordering method**	**Order value > 250 CHF**	**Payment**	**Purchase**
1	m	e-mail	true	credit card	no
2	m	e-mail	true	cash	yes
3	m	web-form	true	invoice	no
4	m	web-form	true	cash	yes
5	f	e-mail	true	cash	yes
6	f	web-form	true	credit card	no
7	f	web-form	true	invoice	yes
8	m	phone	false	invoice	no
9	m	phone	false	cash	no
10	m	e-mail	false	credit card	no
11	m	web-form	false	invoice	no
12	f	phone	false	credit card	no
13	f	phone	false	invoice	yes
14	f	web-form	false	invoice	no
15	f	e-mail	true	cash	yes

- Muster?

  Verhalten von bereits angerufenen Kunden

- Klassen:

  yes (hat gekauft)

  no (hat nicht gekauft)

Simpler Ansatz: One Rule (1R) Algorithmus

- Klassenattribut identifizieren, Klassen identifizieren
- Alle anderen Attribute und deren Wertebereich feststellen
- Für jedes Attribut
  - Für jeden Attributwert
    - Für jede Klasse
      Zähle Auftreten jedes Attributwertes
    - Welche Klasse ist die häufigste?
    - Mit häufigster Klasse eine Regel aufstellen:
      "**WENN** *Attributwert* **DANN** *Klasse*"
    - Fehler dieser Regel berechnen:
      Zähle Auftreten Attributwert mit anderer Klasse
  - Gesamtfehler für dieses Attribut berechnen:
    - Summiere Fehler aller Regeln für dieses Attribut
- Wähle **Regelsatz** mit dem geringsten Gesamtfehler

Anwendung des 1R-Algorithmus

- bestimme Regel**satz** mit kleinstem Gesamtfehler

Attribut	Attribut-wert	Klassen: no	Klassen: yes	Regeln	Fehler Regel	Gesamt-fehler
Gender	m	6	2	m --> no	2	5
	f	3	4	f --> yes	3	
Ordering method	phone	3	1	phone --> no	1	5
	e-mail	2	3	e-mail --> yes	2	
	web form	4	2	web-form --> no	2	
Order value > 250	true	3	5	true --> yes	3	4
	false	6	1	false --> no	1	
Payment	credit card	4	0	credit card --> no	0	3
	invoice	4	2	invoice --> no	2	
	cash	1	4	cash --> yes	1	

Ergebnis von 1R

- Ergebnis ist eine Regel**menge**
  - **alle** Regeln für dasjenige Attribut mit dem geringsten Gesamtfehler
- Regelmenge:
  - WENN credit card DANN no (kauft nicht)
  - WENN invoice DANN no (kauft nicht)
  - WENN cash on delivery DANN yes (kauft)
- Fehler
  - 3 von 15 Instanzen der Trainingsmenge werden damit falsch klassifiziert (aber 12 korrekt!)

Anwendung des 1R-Ergebnisses

- Regelmenge kann nun verwendet werden
  - um unbekannte Instanzen zu klassifizieren
- klassifiziere No. 16, No. 17 und so weiter
  - Klasse unbekannt (noch nicht angerufen)

No	Gender	Ordering method	Order value > 250 CHF	Payment	Purchase
16	m	phone	true	cash	?
17	m	e-mail	false	credit card	?
		web form	true		?

Klassifikation als

yes (kauft)

no (kauft nicht)

- 1R liefert oft schon gute Ergebnisse
  - immer einen Versuch wert – vor dem Einsatz komplexerer Algorithmen

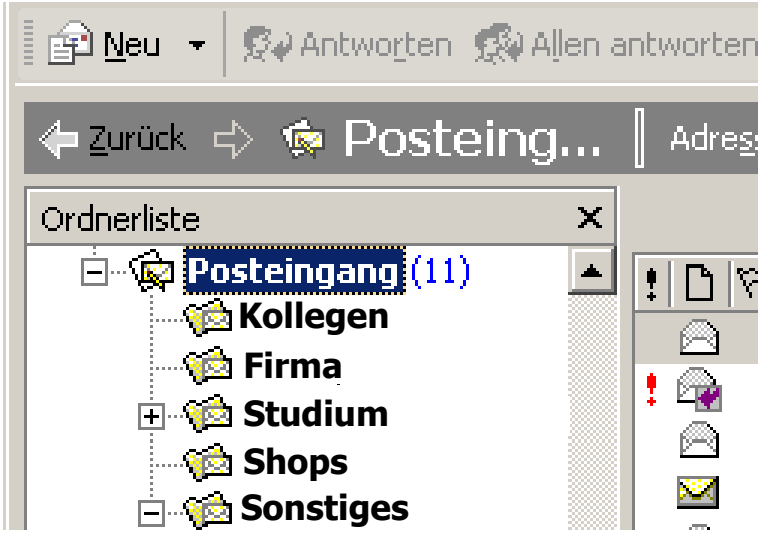
Verschiedene Datensets (Instanzenmengen)

- Trainingsmenge (training set)
  - Klasse bekannt (Vergangenheitsdaten)
  - mit diesen wird das Modell aufgestellt
- Unbekannte Instanzen (unknown instances)
  - Klasse unbekannt
  - diese sollen klassifiziert werden (d.h. auf diese wird das Modell angewandt)
- Testmenge (test set)
  - Instanzen deren Klasse ebenfalls bekannt ist
  - mit diesen wird das Modell validiert (überprüft)

Was ist ein Modell (model)?

- allgemein: Was "gelernt" werden soll
- Beispiele
  - 5 Türen, Alufelgen, < 20'000 CHF

→ Modell: Entscheiden können, welche einen zweiten Blick verdienen (und welche nicht)

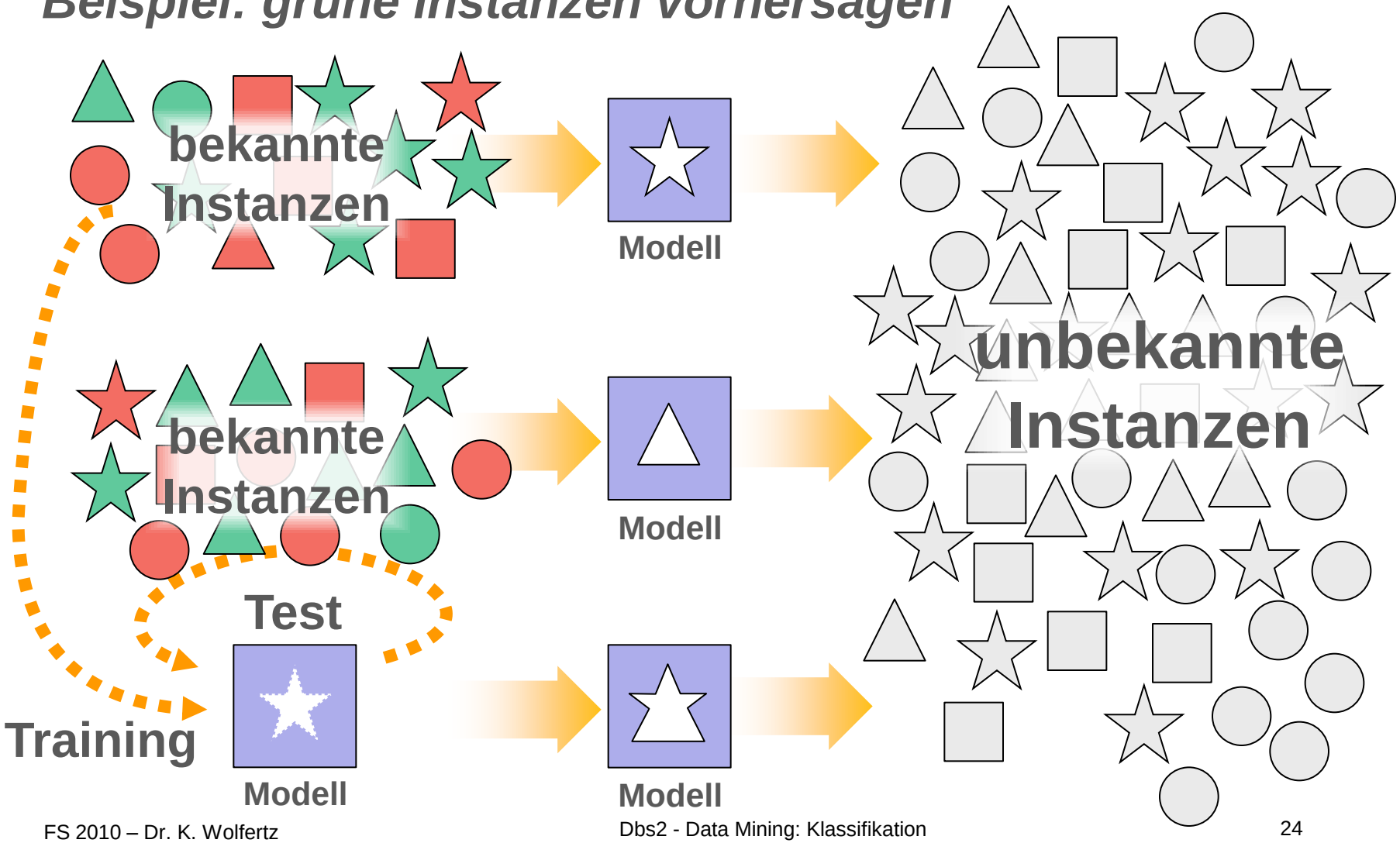


  - Sender, Subject, Text

→ Modell: entscheiden, in welchen Ordner

Aufstellen eines Modells

Beispiel: grüne Instanzen vorhersagen



Modellbildung

- Das Modell des 1R-Algorithmus:
  - die Regelmenge
- andere oder grössere Trainingsmenge:
  - es resultiert eine andere Regelmenge (ein anderes Modell)
- ausgefeiltere Algorithmen bieten eine ganze Reihe von Parametern zur Beeinflussung der Modellbildung

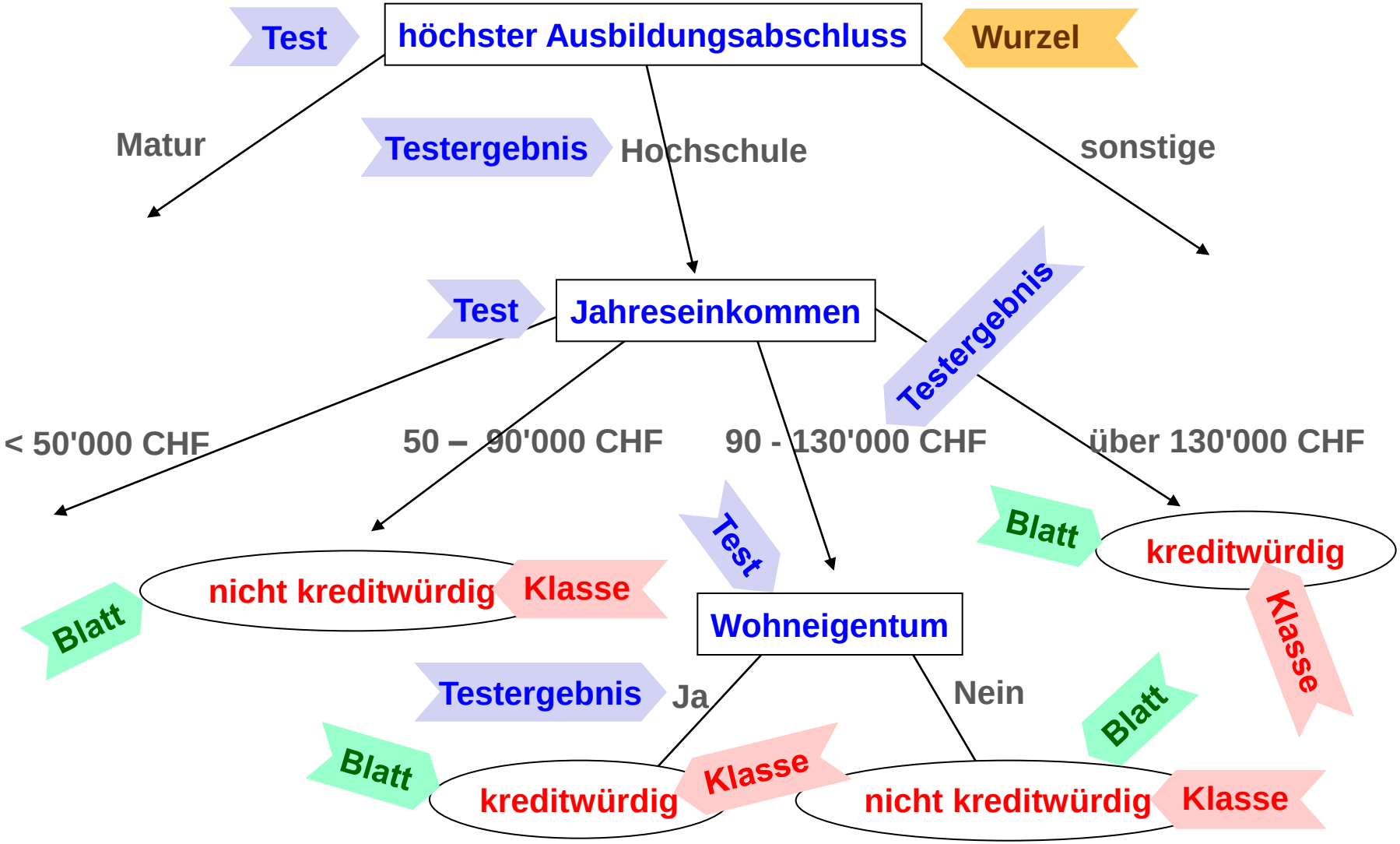
Agenda

Data Mining – Klassifikation

- Einführung
- Klassifikation mit Entscheidungsregeln
- **Klassifikation mit Entscheidungsbäumen**
- Validierung von Klassifizierern

Beispiel Entscheidungsbaum

Vorhersage der Kreditwürdigkeit eines Kunden



Was ist ein Entscheidungsbaum?

- ein Graph
  - flussdiagrammähnliche Baumstruktur
- interne Knoten
  - Test auf ein bestimmtes Attribut
- Kanten
  - Testergebnisse
- Wurzelknoten
  - Start und erster Test
- Blätter
  - Klassenzuweisungen
- das "Modell"
  - bei einer Klassifikation

Klassifikation mit Entscheidungsbäumen

- Aufteilung der Trainingsmenge nach Attributwerten
  - am Beispiel von Attribut "höchster Abschluss":
    - erzeuge 3 Teilmengen: eine mit allen "Matur", eine mit allen "Hochschule" und eine mit allen "sonstige"
- Aufteilung wiederholen
  - bis nur noch Instanzen einer einzigen Klasse in den Teilmengen übrig sind
- diese Klasse zuordnen
  - allen unbekannten Instanzen mit Attributwerten = entsprechendem Pfad von Wurzel zum Blatt

Aufbau von Entscheidungsbäumen

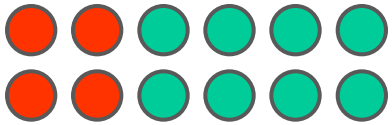
grundlegendes **Problem**: die **Attributauswahl**

- wo beginnen?
  - welches Attribut ist der Wurzelknoten?
  - welches Attribut teilt die gesamte Trainingsmenge am besten auf?
- wie fortfahren?
  - welches Attribut teilt die erhaltenen Teilmengen am besten auf?

  und so weiter

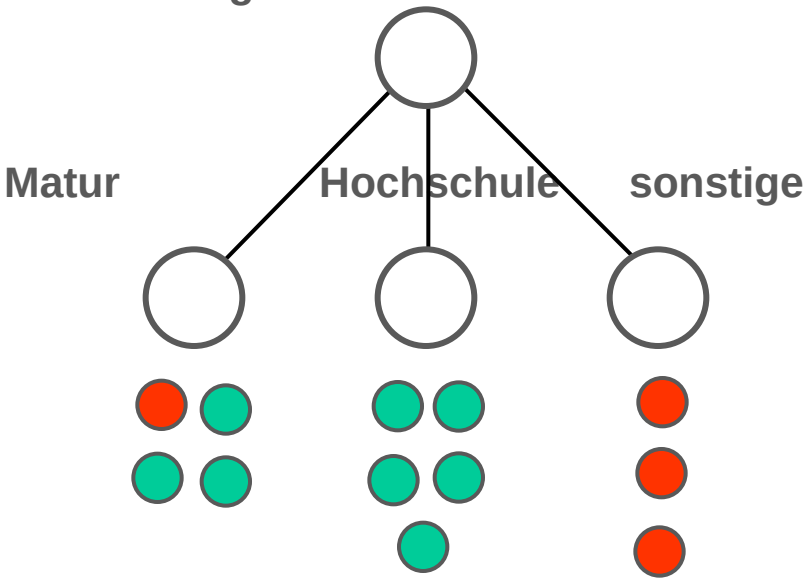
Alternativen bei der Attributauswahl

Trainingsmenge:
(12 Instanzen)

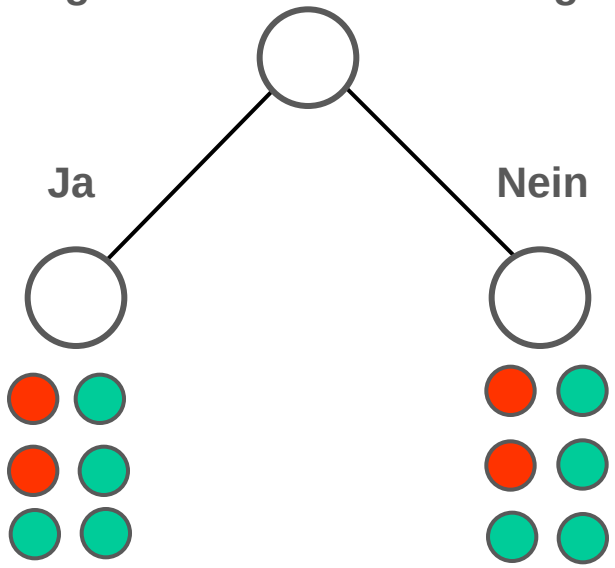


Klasse:
- nicht kreditwürdig
- kreditwürdig

Aufteilung nach Attribut "Abschluss"



Aufteilung nach Attribut "Wohneigentum"



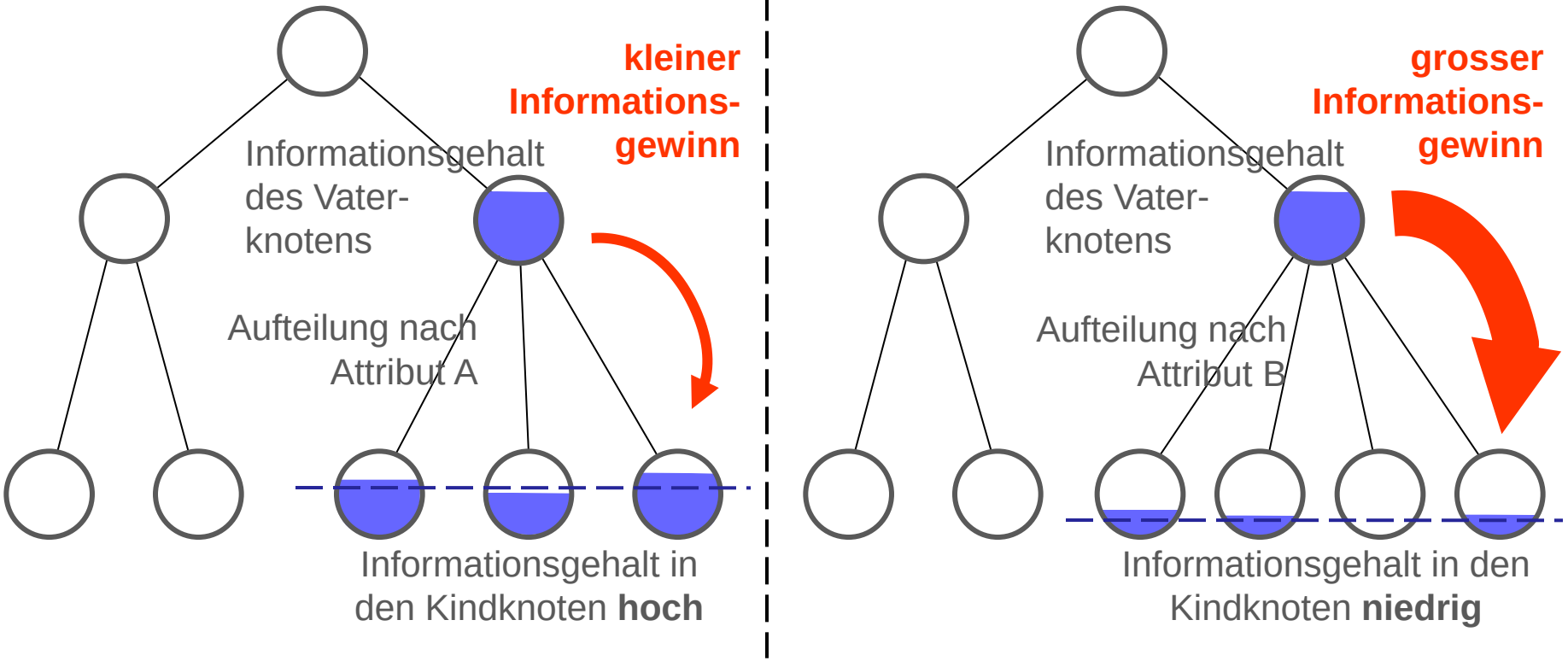
- Aufteilungen besser oder schlechter **sortiert**
- Wie "bessere Sortierung" erkennen?

Informationstheorie

- aus der Informationstheorie:
  - eine "gemischte" Datenmenge zu codieren benötigt mehr Information als eine "homogene"
- Messen des "Informationsgehalts"
  - zum Codieren der einzelnen Aufteilungen
  - identifizieren von "sortenreinen" Teilmengen (nur noch 1 einzige Klasse enthaltend)
- unterschiedliches "Niveau" der Information
  - Aufteilungen mit geringem Informationsgehalt: dort wo grosser Informationsunterschied zwischen Vater- und Kindknoten im Baum

Information Gain

- Bestimmen der grössten Unterschiede zwischen Vater- und Kindknoten
  - dort ist der Informations"gewinn" am höchsten



Berechnung des Information Gain

- Bestimme den Informationsgehalt eines Knotens
- Bestimme die "Entropie" in seinen Kindknoten
  - Entropie = *durchschnittlicher* Informationsgehalt
- Berechne den Unterschied im Informationsgehalt:

Gain = info[Vaterknoten] – Entropie[Kindknoten]

Berechnung des Informationsgehalts

- Informationsgehalt eines Knotens:

$$info[s_1 \ldots s_n] = \frac{-\sum s_i \log_2 s_i + \sum s_i \log_2 \sum s_i}{\sum s_i}$$

mit s_i = Anzahl Instanzen der Klasse i im Knoten
n = Anzahl der vorkommenden Klassen

- oder:

$$info[a, b, \ldots n]$$
$$= [- a \log_2 a \; - b \log_2 b \; \ldots \; - n \log_2 n + (a+b+\ldots+n) \log_2 (a+b+\ldots+n)] \; / \; (a+b+\ldots+n)$$

Berechnung der Entropie

- Durchschnitt des Informationsgehalts in allen Kindknoten des untersuchten Vaterknotens:
  - summiere Informationsgehalte, gewichtet mit dem Anteil der betroffenen Instanzen

$$E = \frac{(a+b+...+n)}{S} \text{info}[a,b,...n] + \ ... \ + \frac{(u+v+...+z)}{S} \text{info}[u,v,...z]$$

mit S = Anzahl aller Instanzen in den Kindknoten des untersuchten Vaterknotens

Entscheidungsbauminduktion mit Information Gain

- Wähle immer "sortenreinste" (in Bezug auf Klassen) Attributaufteilung
  - d.h. die Kindknoten sind so homogen wie möglich = geringe "Entropie"

  = Aufteilung mit dem höchsten Information Gain
- minimiert Anzahl notwendiger Tests, um eine Instanz zu klassifizieren

  = kurzer Weg durch den Baum bis Blatt erreicht
- Erzeugt vergleichsweise einfache Bäume
- aber nicht unbedingt die *einfachsten*

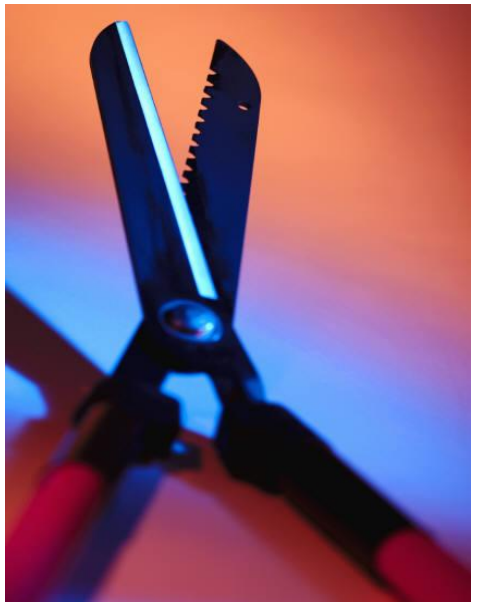
Der ID3 Algorithmus

beispielhafter Algorithmus zur Entscheidungs-bauminduktion (decision tree induction)

- "**I**terative **D**ichotomizer **3**"
- von Ross Quinlan, 1986
- verwendet Information Gain
- ein so genannter "**greedy** algorithm"
  - geht immer in Richtung des höchsten Informationsgewinns vor
  - Ergebnis nicht unbedingt optimal

Pruning

- "Beschneiden" eines Entscheidungsbaums
- Entfernen von Verzweigungen und von Teilbäumen
  - die evtl. Fehler repräsentieren: "verrauschte" Daten ("noise")
  - die Spezifika nur der Testmenge wiedergeben
    → "Überangepasstheit" (overfitting)



- Genauigkeit und Qualität der Klassifikation kann oft erhöht werden

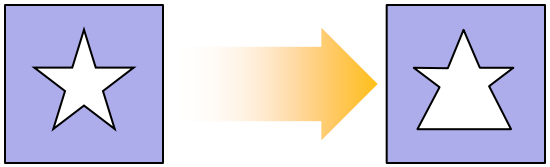
Overfitting

häufiges Problem im Data Mining

- Modell ist hervorragend geeignet
  - für die Vorhersage genau der Trainingsmenge
- Modell ist weniger gut geeignet
  - für die Vorhersage weiterer Instanzen
  - Vorhersageleistung fällt rapide ab
- Schlechtere Passung auf Trainingsmenge
  - im Durchschnitt besser für beliebige Instanzen
- auf Entscheidungsbäume angewandt
  - heisst das "Pruning"

Vorteile von Pruning

- Entfernen insignifikanter Verzweigungen
  - am wenigsten verlässlich, typisch
- kleinere, weniger komplexe Bäume
  - kürzerer Weg durch den Baum
  - schnellere Klassifikation
  - Modell wird verständlicher



- im Durchschnitt bessere Klassifikation unbekannter Instanzen
- zwei Ansätze: Pre- und Postpruning

Prepruning

- Aufteilung gar nicht erst ausführen
  - Nicht-Entwickeln bestimmter Äste
  - Knoten wird zum "Blatt" erklärt, obwohl noch nicht "sortenrein"
- Pruning noch während der Modellbildung
  - z.B.: der durch Aufspaltung erzielte Informationsgewinn erreicht eine gesetzte Mindestschwelle nicht, Aufspaltung unterdrückt, Algorithmus stoppt für diesen Teilbaum

Postpruning

- Rückgängigmachen von Aufteilungen
- Pruning erst nach vollständig abgeschlossener Modellbildung
- Verschmelzen von Kindknoten
  - z.B.: berechnen der Fehlerrate
    (falsch klassifizierte Instanzen / alle Instanzen)
    falls Kindknoten zusammengefasst würden.
    Falls unter gesetzter Maximalschwelle, dann zusammenfassen (d.h. Unterbaum wird eliminiert).

Der C4.5 Algorithmus

- C4.5
- ebenfalls von Ross Quinlan
- Erweiterung des ID3 Algorithmus
  - wendet genau so den Information Gain an
- neben anderen Verbesserungen:
  - Postpruning des Entscheidungsbaumes
- WEKA-Tool implementiert C4.5 als "J48"
  - J48 bietet zusätzlich das so genannte "reduced error pruning" an

Agenda

Data Mining – Klassifikation

- Einführung
- Klassifikation mit Entscheidungsregeln
- Klassifikation mit Entscheidungsbäumen
- **Validierung von Klassifizierern**

Validierung von Klassifikationsmodellen

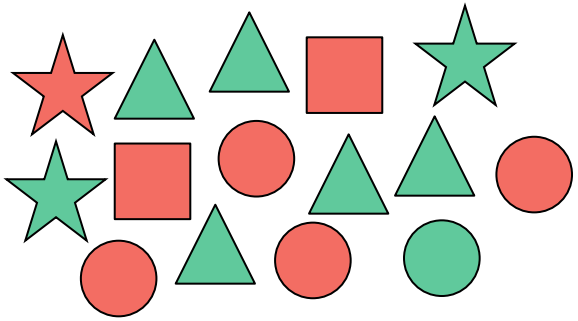
- Was "leistet" ein bestimmtes Modell?
  - wie "gut" ist das Modell?

  gebräuchlichste Massstäbe:
- Accuracy (Genauigkeit)
  - Prozentsatz an korrekt klassifizierten Instanzen
  - auch: "recognition rate"
    - wie gut werden Instanzen verschiedener Klassen identifiziert?
- Fehlerrate (error rate, misclassification rate)

  = 1 – Accuracy

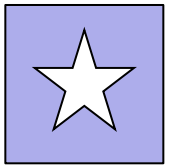
Messen der Modellgenauigkeit

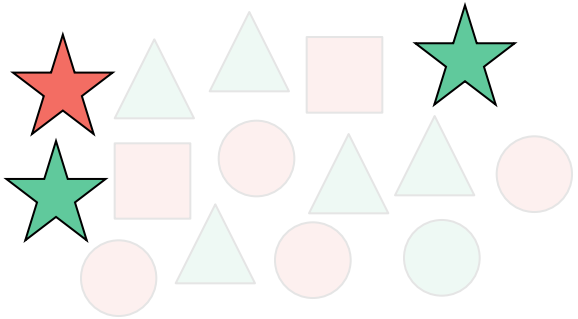
Beispiel: *grüne Instanzen vorhersagen*



- Nach Anwendung des Modells auf eine Menge von Instanzen stellen sich bezüglich des Ergebnisses folgende Fragen
- Wie viele *als grün klassifizierte* Instanzen sind tatsächlich grün?

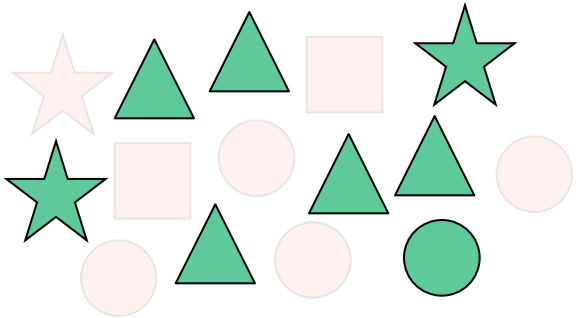
  (2 von 3)

das Modell 



- Wie viele der *tatsächlich grünen* Instanzen wurden als grün klassifiziert?

  (2 von 8)



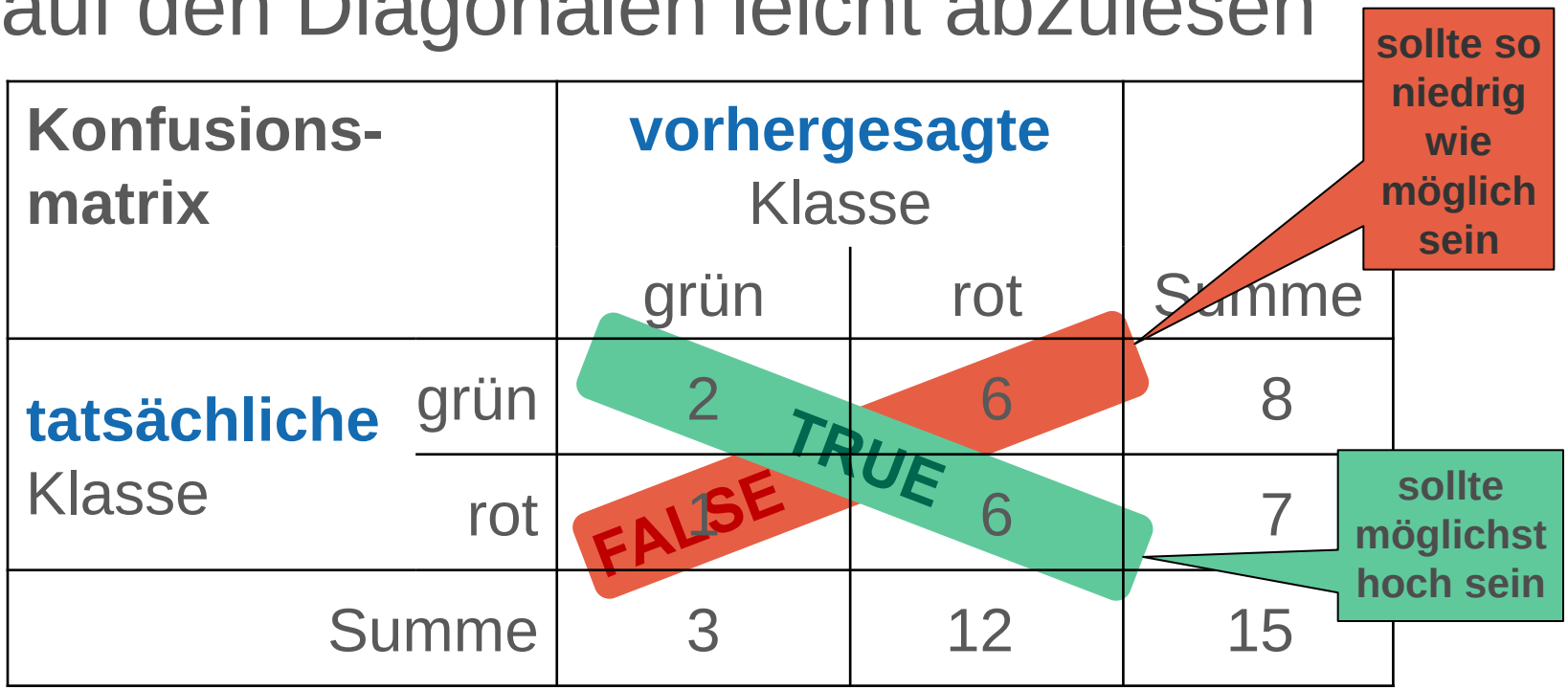
Konfusionsmatrix

- Messgrössen zur Modellgenauigkeit
  - Bestimmung einfach, Auszählen Testresultate
- Eintragen in eine so genannte "Konfusionsmatrix" (confusion matrix)

Konfusions-matrix		**vorhergesagte** Klasse		
		grün	rot	Summe
tatsächliche Klasse	grün	2	6	8
	rot	1	6	7
	Summe	3	12	15

Modellgenauigkeit auf einen Blick

- Accuracy ist hoch, wenn
  - Zahl korrekter (TRUE) Vorhersagen hoch ist
  - und Zahl falscher (FALSE) Vorhersagen niedrig
- auf den Diagonalen leicht abzulesen



Positives and Negatives

Gegeben zwei Klassen sprechen wir von

- **Positives** (Instanzen der Zielklasse)
  - Klasse, der das primäre Interesse gilt (z.B. "wird kaufen")
- **Negatives** (Instanzen der Kontrastklasse)
  - z.B. "kauft nicht")

Bei mehr als zwei Klassen

- Klasse mit Primärinteresse = positive
- alle anderen Klassen = negative

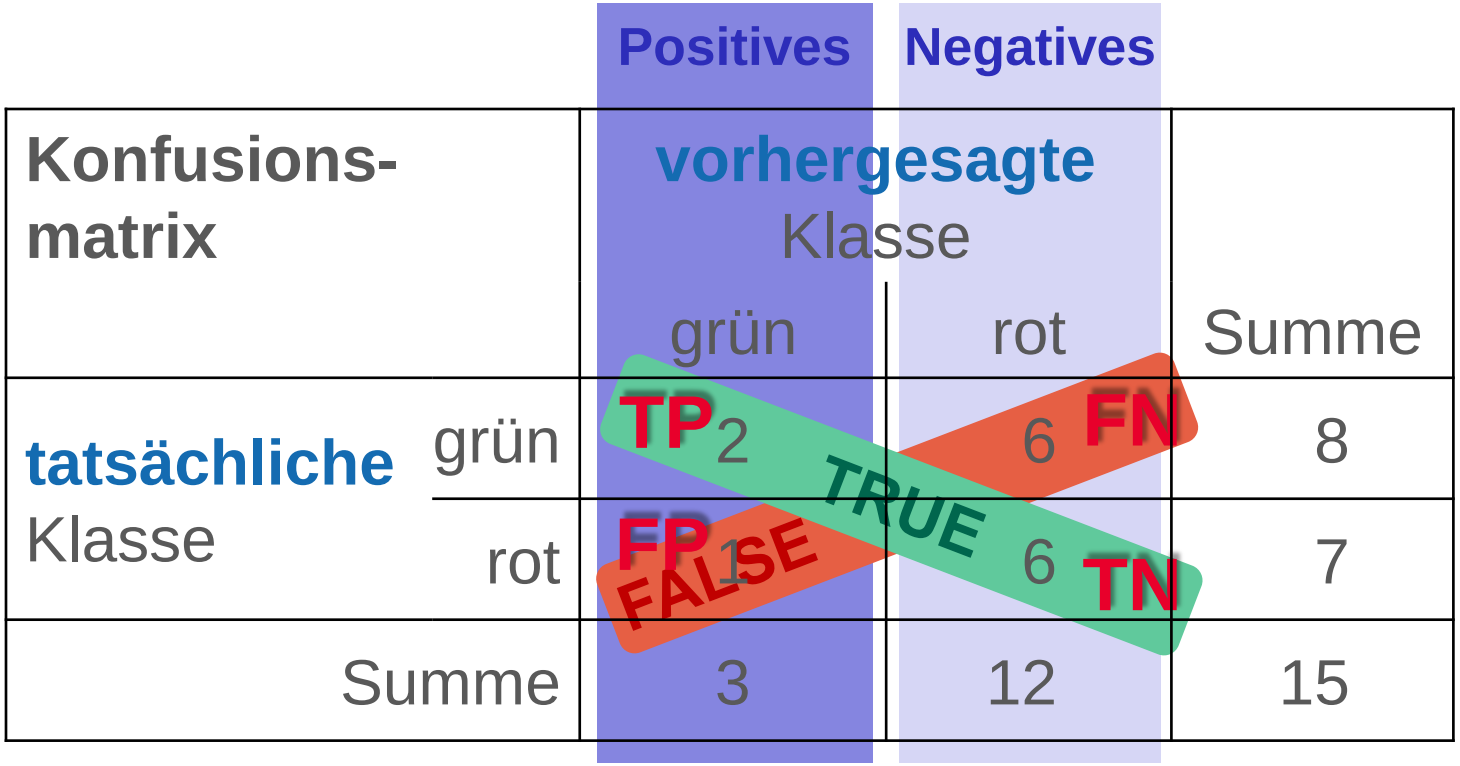
Güteindikatoren für die Testergebnisse

Wir unterscheiden 4 Arten von Instanzen

- True Positives (**TP**)
  - Zielklasse, korrekt vorausgesagt
- True Negatives (**TN**)
  - Kontrastklasse, korrekt vorausgesagt
- False Positives (**FP**)
  - vorhergesagt als Zielklasse, aber tatsächliche Klasse ist eine andere
- False Negatives (**FN**)
  - vorhergesagt als Kontrastklasse, aber tatsächliche Klasse ist eine andere

Güteindikatoren in der Konfusionsmatrix

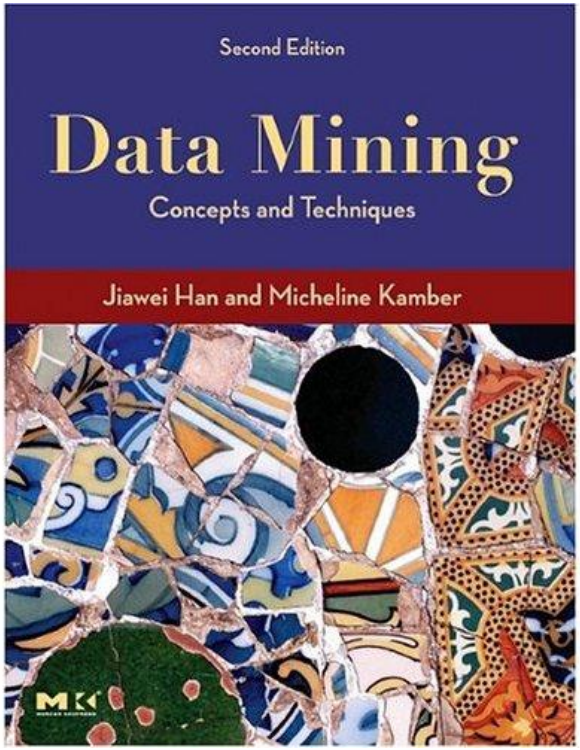
- Alle Zahlen können in der Konfusionsmatrix wiedergefunden werden



Hausaufgaben

- Lösen von Klassifikationsaufgaben mit Data Mining Software (WEKA)
- Sie finden die Aufgaben auf Wiki
- Die Aufgaben bieten Instruktionen zur Bedienung von WEKA

Weiterführende Literatur



für Interessierte:

HAN, Jiawei / KAMBER, Micheline
Data Mining
Concepts and Techniques
2nd Edition - Morgan Kaufmann, San Francisco: 2006

Ausblick

- ☑ Von Decision Support zu Data Warehouses
- ☑ OLAP – Online Analytical Processing
- ☑ Data Mining – Klassifikation
- ■ **Data Mining – Assoziationsanalyse**
- ☐ Data Mining – Clustering

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Data Mining: Assoziationsanalyse

Datenbanksysteme 2

Dr. Klaus Wolfertz

25. Mai 2010

Überblick

- ☑ Von Decision Support zu Data Warehouses
- ☑ OLAP – Online Analytical Processing
- ☑ Data Mining – Klassifikation
- ■ **Data Mining – Assoziationsanalyse**
- ☐ Data Mining – Clustering

Agenda

Data Mining – Assoziationsanalyse

- **Grundlagen der Assoziationsanalyse**
- Entdecken von Assoziationen
- systematische Ableitung von Assoziationsregeln: A priori-Algorithmus
- Korrelationsanalyse
- Mehrebenen-Assoziationsanalyse

Attributvorhersage

- nicht die Klasse, sondern andere Attribute werden vorhergesagt
- Assoziation von Attributen und Attributwerten
- assoziativ = verbindend, vereinigend
- Aufstellen von assoziativen Regeln
  - wenn A, dann B
  - versehen mit einem "Verlässlichkeitsmass"

Häufigste Praxisanwendung

Warenkorbanalyse

- A: Wie viel Prozent der Einkäufe umfassten Bier? Wie viel Windeln?
- B: Wie viel Prozent der Einkäufe umfassten Bier und Windeln gleichzeitig?
- Wie ist das Verhältnis von A und B?
- Kann abgeleitet werden, dass immer dann, wenn Bier gekauft wird, mit hoher Wahrscheinlichkeit auch Windeln gekauft werden?

Interpretation der Ergebnisse

- Mögliche Schlussfolgerungen
  - Bier und Windeln zusammen platzieren (convenience), oder
  - Bier und Windeln weit auseinander platzieren (damit man an möglichst vielen anderen Angeboten vorbei muss)
  - Sonderangebot für Bier, um Kunden anzulocken, die dann auch Windeln kaufen
  - Windeln mit hoher Gewinnmarge neben dem Bier platzieren
  - ...

Begriffe

- Item
  - eine bestimmte untersuchte Einheit
    (= ein Attribut)
- Regel
  - Herstellen eines Zusammenhangs zwischen Items (zwischen Attributen)
- Transaktion
  - zusammengehöriger Vorgang, Items betreffend
    (= ein Beispiel, eine Instanz)

Transaktionen

- z.B. 1 Einkauf, 1 Bestellung, bestimmter Warenkorb 1es Kunden pro Kauf
- dargestellt als Vektor Boole'scher Werte
  - Item ist im Warenkorb oder nicht

    $(Item_1, Item_2, Item_3, \ldots, Item_n)$

    (True, False, True, ... , False)

1 Transaktion:

Ackerbräu Harass	Ruchbrot, Laib	Shmuggies Family Pack	...	...	Edelsalami 250g
X	-	X	...	...	-

Gegenstandsmengen

- Gegenstandsmenge (= itemset)
  - Ein bestimmter Vektor von Items
- Abhängig von der Anzahl Items im Vektor spricht man von
  - Einermengen
  - Zweiermengen
  - Dreiermengen usw. …
    - (auch: "Ein-Gegenstand-Menge" etc. / k-itemset für k ϵ 1, ..., n)
- Beispiel: (Bier, Windeln) = Zweiermenge

Gesucht: "frequent patterns"

	Items			
Tr. Nr.	Tomaten	Kopfsalat	Bier	Emmentaler
1		✓	✓	
2		✓	✓	
3				✓
4	✓			✓
5		✓		✓
6	✓	✓	✓	

- Anwesenheit eines Teils der Transaktion impliziert (häufig) einen anderen Teil
- z.B. oben: "Kopfsalat" impliziert häufig "Bier"

Agenda

Data Mining – Assoziationsanalyse

- Grundlagen der Assoziationsanalyse
- **Entdecken von Assoziationen**
- systematische Ableitung von Assoziationsregeln: A priori-Algorithmus
- Korrelationsanalyse
- Mehrebenen-Assoziationsanalyse

Vorgehen

- Identifikation von Gegenstandsmengen, die häufig vorkommen (frequent itemsets)
- Aufstellen von Regeln, die Items dieser häufigen Gegenstandsmengen miteinander verknüpfen
- Bestimmen von Verlässlichkeitsmassen für diese Regeln

Regeln

- Muster (pattern) formuliert als Regel
  - Regel = Aussage, Vermutung
- Verknüpfung von zwei oder mehr Items in einer "Wenn ... dann ..."-Aussage
- allgemein:

  I = Menge aller Items mit $I = \{i_1, i_2, \ldots, i_n\}$

  Regel R: $\{i_j \ldots i_k\} \Rightarrow \{i_l \ldots i_m\}$

  oder **R: LS** (linke Seite) $\Rightarrow$ **RS** (rechte Seite)

  mit LS, RS $\subseteq I$ und LS $\cap$ RS = $\emptyset$
- konkret: "Wenn Windeln dann Bier"

  "Wenn Eier und Bier dann Tomaten"

Beispielrechnung

- Bestimmen Sie alle möglichen Regeln

Tr. Nr.	Tomaten	Kopfsalat	Bier	Emmentaler
1		✓	✓	
2		✓	✓	
3				✓
4	✓			✓
5		✓		✓
6	✓	✓	✓	

- Wie viele erhalten Sie? **16**

Tomaten ⇒ Kopfsalat
Tomaten ⇒ Bier
Tomaten ⇒ Emmentaler
Kopfsalat ⇒ Bier
Kopfsalat ⇒ Emmentaler

Kopfsalat ⇒ Tomaten
Bier ⇒ Tomaten
Emmentaler ⇒ Tomaten
Bier ⇒ Kopfsalat
Emmentaler ⇒ Kopfsalat

Tomaten, Kopfsalat ⇒ Bier
Tomaten ⇒ Kopfsalat, Bier
Tomaten, Bier ⇒ Kopfsalat
Kopfsalat ⇒ Tomaten, Bier
Kopfsalat, Bier ⇒ Tomaten
Bier ⇒ Kopfsalat, Tomaten

kombinatorische Explosion

- Wenn alle Kombinationen vorkämen
  - bei ein paar mehr Instanzen so gut wie sicher

Tomaten ⇒ Kopfsalat
Tomaten ⇒ Bier
Tomaten ⇒ Emmentaler
Kopfsalat ⇒ Bier
Kopfsalat ⇒ Emmentaler
Bier ⇒ Emmentaler
Kopfsalat ⇒ Tomaten
Bier ⇒ Tomaten
Emmentaler ⇒ Tomaten
Bier ⇒ Kopfsalat
Emmentaler ⇒ Kopfsalat
Emmentaler ⇒ Bier

Tomaten, Kopfsalat ⇒ Bier
Tomaten ⇒ Kopfsalat, Bier
Tomaten, Bier ⇒ Kopfsalat
Kopfsalat ⇒ Tomaten, Bier
Kopfsalat, Bier ⇒ Tomaten
Bier ⇒ Kopfsalat, Tomaten

Tomaten, Kopfsalat ⇒ Emmentaler
Tomaten ⇒ Kopfsalat, Emmentaler
Tomaten, Emmentaler ⇒ Kopfsalat
Kopfsalat ⇒ Tomaten, Emmentaler
Kopfsalat, Emmentaler ⇒ Tomaten
Emmentaler ⇒ Kopfsalat, Tomaten
Tomaten, Bier ⇒ Emmentaler
Tomaten ⇒ Bier , Emmentaler
Tomaten, Emmentaler ⇒ Bier
Bier ⇒ Tomaten, Emmentaler
Bier, Emmentaler ⇒ Tomaten
Emmentaler ⇒ Bier , Tomaten
Kopfsalat, Bier ⇒ Emmentaler
Kopfsalat ⇒ Bier , Emmentaler
Kopfsalat, Emmentaler ⇒ Bier
Bier ⇒ Kopfsalat , Emmentaler
Bier, Emmentaler ⇒ Kopfsalat
Emmentaler ⇒ Bier , Kopfsalat

Tomaten ⇒ Kopfsalat, Bier, Emmentaler
Kopfsalat ⇒ Tomaten, Bier, Emmentaler
Bier ⇒ Tomaten, Kopfsalat, Emmentaler
Emmentaler ⇒ Tomaten, Kopfsalat, Bier
Kopfsalat, Bier, Emmentaler ⇒ Tomaten
Tomaten, Bier, Emmentaler ⇒ Kopfsalat
Tomaten, Kopfsalat, Emmentaler ⇒ Bier
Tomaten, Kopfsalat, Bier ⇒ Emmentaler
Tomaten, Kopfsalat ⇒ Bier, Emmentaler
Tomaten, Bier ⇒ Kopfsalat, Emmentaler
Tomaten, Emmentaler ⇒ Kopfsalat, Bier
Kopfsalat, Bier ⇒ Tomaten, Emmentaler
Kopfsalat, Emmentaler ⇒ Tomaten, Bier
Bier, Emmentaler ⇒ Tomaten, Kopfsalat

50

Ziel

Wie finden wir einige wenige, aber "gute", "aussagekräftige" Regeln?

- Aufstellen von Regeln nur für **häufige** Gegenstandsmengen:
  - kauft(Kopfsalat) $\Rightarrow$ kauft(Bier)
- Auswählen nur solcher Regeln, die häufig **zutreffen**
  - also Untersuchung, wie "verlässlich" diese Regeln sind

Häufigkeit

- Zählen des Auftretens
  - eines items, eines itemsets
- relative Häufigkeit
  - (occurence) frequency
  - (support) count

  = Anteil an der Gesamtmenge (an Instanzen)
- **Support**
  - eines items: prozentuale Häufigkeit
  - einer Regel: prozentuale Häufigkeit aller items miteinander (alle in der Regel genannten items)

Beispielrechnung

- Bestimmen der Häufigkeiten

Tr. Nr.	Tomaten	Kopfsalat	Bier	Emmentaler
1		✓	✓	
2		✓	✓	
3				✓
4	✓			✓
5		✓		✓
6	✓	✓	✓	

Häufigkeit 1er-Mengen: **2** **4** **3** **3**

relative Häufigkeit 1er-Mengen: **2/6** **4/6** **3/6** **3/6**

relative Häufigkeit Kopfsalat ∪ Bier: **3/6**

relative Häufigkeit Tomaten ∪ Kopfsalat ∪ Bier: **1/6**

Verlässlichkeitsmass

- Regel

  Wenn Item1, Item2 ... **dann** ... Item n-1, Item n
  linke Seite *rechte Seite*

- **Confidence**
  - Support einer Regel im Verhältnis zum Support des "linken" Itemsets
  - Wie häufig sind alle in der Regel vorkommenden Items im Vergleich zu den "linken" Items?

$$\text{confidence} = \frac{\text{Häufigkeit (}\textit{linke Seite}\ \textbf{und}\ \textit{rechte Seite}\text{)}}{\text{Häufigkeit (}\textit{linke Seite}\text{)}}$$

Berechnung der Verlässlichkeit

- Regel

$$R: A \Rightarrow B$$

- Support

$$supp(A) = \frac{\text{Anzahl Transaktionen mit A}}{\text{Anzahl aller Transaktionen}}$$

$$supp(R) = \frac{\text{Anzahl Transaktionen mit } (A \cup B)}{\text{Anzahl aller Transaktionen}}$$

- Confidence

$$conf(R) = \frac{supp\ (R)}{supp\ (A)}$$

$$\text{oder: } conf(R) = \frac{\text{Anzahl Transaktionen mit } (A \cup B)}{\text{Anzahl aller Transaktionen mit A}}$$

Verwendung

- Support
  - Signifikanz, Wichtigkeit der itemsets
  - supp (R: A $\Rightarrow$ B) = supp (R': B $\Rightarrow$ A)
- Confidence
  - "Vertrauensmass", wie "sicher" kann gesagt werden, dass B vorkommt, wenn A vorkommt.
  - conf (R: A $\Rightarrow$ B) $\neq$ conf (R': B $\Rightarrow$ A) **!!!**

'Starke' Regeln

- Wie kommen wir zu "guten", "aussagekräftigen" Regeln?
- Setzen bestimmter Anforderungen:
  - Mindestsupport
  - Mindestconfidence
- "Starke" Regeln (strong rules)
  - erfüllen gleichzeitig Mindestsupport **und** Mindestconfidence

Agenda

Data Mining – Assoziationsanalyse

- Grundlagen der Assoziationsanalyse
- Entdecken von Assoziationen
- **systematische Ableitung von Assoziationsregeln: A priori-Algorithmus**
- Korrelationsanalyse
- Mehrebenen-Assoziationsanalyse

Automatisierung

- Anzahl möglicher Regeln wird sehr schnell sehr gross
- Auswahl durch "rhythmisches Hinsehen" unmöglich
- maschinelle Methoden zur "Sichtung" zwingend
- Bekanntester Algorithmus: A-Priori

A-priori - Bestimmen aller häufigen Mengen

- Festlegen Mindestsupport σ
- Ermitteln der häufigen Ein-Gegenstand-Mengen
  - alle mit Support $\geq \sigma$
- Kombination zu häufigen Mehr-Gegenstands-Mengen
  - Hinzunahme anderer häufiger Ein-Gegenstand-Mengen
  - Support $\geq \sigma$?
  - für alle möglichen Kombinationen

A-priori - Candidate Generation

- Bilden möglicher Zweier-Mengen
  - nur aus häufigen Einer-Mengen:
    kombinierte Menge kann nur häufig sein, falls Teilmengen auch häufig (**a-priori**-Eigenschaft)
  - "Kandidaten" für nächste Runde (Iteration)
- Häufigkeit der Zweier-Mengen bestimmen
  - Abzählen des Vorkommens der "Kandidaten"
- nicht-häufige Itemsets scheiden aus
  - "Kandidaten" anhand Mindestsupport geprüft
  - nur solche mit ausreichend Support kommen eine Runde weiter

Die A-Priori-Eigenschaft

- Bilden möglicher Dreier-Mengen
  - aus allen Kombinationen der häufigen Zweier-Mengen untereinander, wobei sich diese Zweier-Mengen "überlappen" (sog. natural join)
    Beispiel: Kopfsalat ∪ Bier und Bier ∪ Emmentaler
    kombiniert zu Kopfsalat ∪ Bier ∪ Emmentaler
- Kandidaten ausscheiden
  - anhand der **a-priori**-Eigenschaft: nicht häufig, falls irgendeine Teilmenge der neuen Dreier-Menge nicht häufig sein sollte (etwa Kopfsalat ∪ Emmentaler)
- Häufigkeit verbliebener Kandidaten bestimmen
- Bilden möglicher Vierer-Mengen usw. …

A-priori - Ende der Suche

- Suche nach itemsets endet, wenn
  - alle möglichen Kombinationen untersucht oder
  - Mindestsupport bei allen Kandidaten unterschritten
- Sobald alle häufigen Mehr-Gegenstand-Mengen feststehen, können Regeln aufgestellt werden

A-priori - Regeln aufstellen

- Festlegen Mindestconfidence γ
- Für jede häufige Menge **alle möglichen** Kombinationen als Regeln formulieren
  - Beispiel: häufige 3er-Menge {A, B, C} ergibt die Regeln

A → B,C	A,B → C
B → A,C	B,C → A
C → A,B	A,C → B

- Bestimmen Confidence für jede Regel
- Regeln mit Mindestconfidence auswählen

Veranschaulichung

sei Mindestsupport $\sigma = 0.5$

Tr. Nr.	Tomaten	Kopfsalat	Bier	Emmentaler
1		✓	✓	
2		✓	✓	
3				✓
4	✓			✓
5		✓		✓
6	✓	✓	✓	
Häufigkeit 1er-Mengen:	2/6	4/6	3/6	3/6

Einermengen mit Support ≥ 0.5
mögliche Zweier-Menge
Zweier-Menge mit Support ≥ 0.5

Regeln: Kopfsalat $\Rightarrow$ Bier, conf = 0.75; Bier $\Rightarrow$ Kopfsalat, conf = 1.0

Verschiedene Arten der Assoziation

- Boole'sche Assoziation
  - Vorhanden-/Nichtvorhandensein von Attributwerten
- Quantitative Assoziation
  - Alter(30 - 39) $\wedge$ Einkommen(50' – 80') $\Rightarrow$ Kreditrahmen(60' – 120')
- Ein- und mehrdimensionale Assoziation
  - kauft(Bier) $\Rightarrow$ kauft(Windeln) *eindimensional*
  - hat(Cumulus) $\wedge$ wohnt(Aargau) $\Rightarrow$ kauft(Aktionen) *mehrdimensional*
- Assoziation auf unterschiedlichen Abstraktionsebenen
  - kauft(Bier) $\Rightarrow$ kauft (Windeln)
  - kauft(Ackerbräu 6x0.33l) $\Rightarrow$ kauft (Hygieneartikel)

Anwendung

- Eine Befragung von insgesamt 200 Studenten ergab:
  - 150 besuchen regelmässig die Mensa
  - 120 studieren Informatik
  - 80 gehen regelmässig in die Mensa und studieren Informatik
- Bestimmen Sie die Confidence der Regel

  studiert Informatik $\Rightarrow$ geht in Mensa
- Schlussfolgerung für Mensabetreiber? Sollte er seine Öffnungszeiten nach den Informatikkursen richten?

Agenda

Data Mining – Assoziationsanalyse

- Grundlagen der Assoziationsanalyse
- Entdecken von Assoziationen
- systematische Ableitung von Assoziationsregeln: A priori-Algorithmus
- **Korrelationsanalyse**
- Mehrebenen-Assoziationsanalyse

Informatiker in die Mensa?

- **Supp** Informatik ⇒ Mensa = 80/200 = **0.4**
- **Conf** Informatik ⇒ Mensa = 80/120 = **0.67**
- relativ hohe Werte

aber...

nur 67% der Informatiker essen in der Mensa, während von allen Studenten bereits 75 % dies tun, von den Nichtinformatikern sogar 87.5 %
(das sind 70 der insgesamt 80 Nichtinformatiker)

- Informatikstudium und Mensabesuch sind negativ korreliert

Korrelation

- Auftreten von A ist unabhängig vom Auftreten von B, wenn gilt:

$$P(A \cup B) = P(A) \cdot P(B)$$

  - Beispiel: Werfen zweier Münzen
  - Wenn die Wahrscheinlichkeit, mit beiden Münzen 'Kopf' zu werfen (¼) gleich ist der Wahrscheinlichkeit 'Kopf' bei der einen (½) mal 'Kopf' bei der anderen (½)
- Ansonsten sind A und B **korreliert**
  - korrelativ = sich gegenseitig bedingend

Berechnung der Korrelation

- Die Korrelation von A und B berechnet sich nach:

$$corr_{A,B} = \frac{P(A \cup B)}{P(A) \cdot P(B)}$$

- Korrelation > 1: positiv korreliert, A impliziert B
- Korrelation < 1: negativ korreliert, A impliziert, dass B nicht auftritt

Kontingenztabelle

- Herausfiltern von irreführenden Regeln

Beispiel:

$$\frac{P(\text{Inf.} \cup \text{Mensa})}{P(\text{Inf.}) \cdot P(\text{Mensa})}$$

$$= \frac{0.4}{0.6 \cdot 0.75} = 0.89$$

	Informatik	nicht Informatik	$\sum$
Mensa	80 p=0.4 **corr = 0.89**	70 p=0.35 **corr = 1.17**	150 p=0.75
nicht Mensa	40 p=0.2 **corr = 1.33**	10 p=0.05 **corr = 0.5**	50 p=0.25
$\sum$	120 p=0.6	80 p=0.4	200 p=1.0

Negativ korreliert

Korrelationsanalyse

Aufstellen von Korrelationsregeln

- für jedes gebildete Itemset
  - Ablesen des Korrelationskoeffizienten für die beteiligten Items direkt aus der Kontingenztabelle
- Kennzeichnen gefundener Regeln
  - zusätzlich mit dem Korrelationskoeffizienten

  Tomaten $\Rightarrow$ Bier (supp 0.4, conf 0.73, corr 1.4)

Lift

Korrelationskoeffizient in der Warenkorb-analyse auch als '**Lift**' bezeichnet

- 'Lift' einer Regel wird dabei berechnet aus

$$\text{Lift}(A \rightarrow B) = \frac{\text{conf}(A \rightarrow B)}{\text{supp}(B)}$$

da $\text{conf}(A \rightarrow B) = \frac{\text{supp}(A \cup B)}{\text{supp}(A)}$ gilt:

$$\text{Lift}(A \rightarrow B) = \frac{\text{supp}(A \cup B)}{\text{supp}(A) \cdot \text{supp}(B)} = \frac{P(A \cup B)}{P(A) \cdot P(B)} = \text{corr}_{A,B}$$

zusätzliches Mass für die Aussagekraft

- Der Lift gibt an, um wie viel die Confidence der Regel die pure Wahrscheinlichkeit übertrifft

  wenn A dann B: B hat eine Eintreffenswahrscheinlichkeit, steigt diese, wenn A auch vorkommt?

- Lift zeigt die generelle Bedeutung einer Regel
  - Anders ausgedrückt: Wie viel besser sagt die Regel die 'rechte Seite' (RS) voraus, als ohne Regel gewiss wäre?
- Beispiel:
  - Regel: studiert Informatik → verwendet linux
  - Angenommen
    - 10 % aller Studenten nutzen linux ($supp_{linux}$=0.1), aber
    - 50% aller Informatikstudenten haben linux ($conf_{Inf \rightarrow linux}$=0.5)

  Dann hat die Regel einen 5-fachen Lift.

Agenda

Data Mining – Assoziationsanalyse

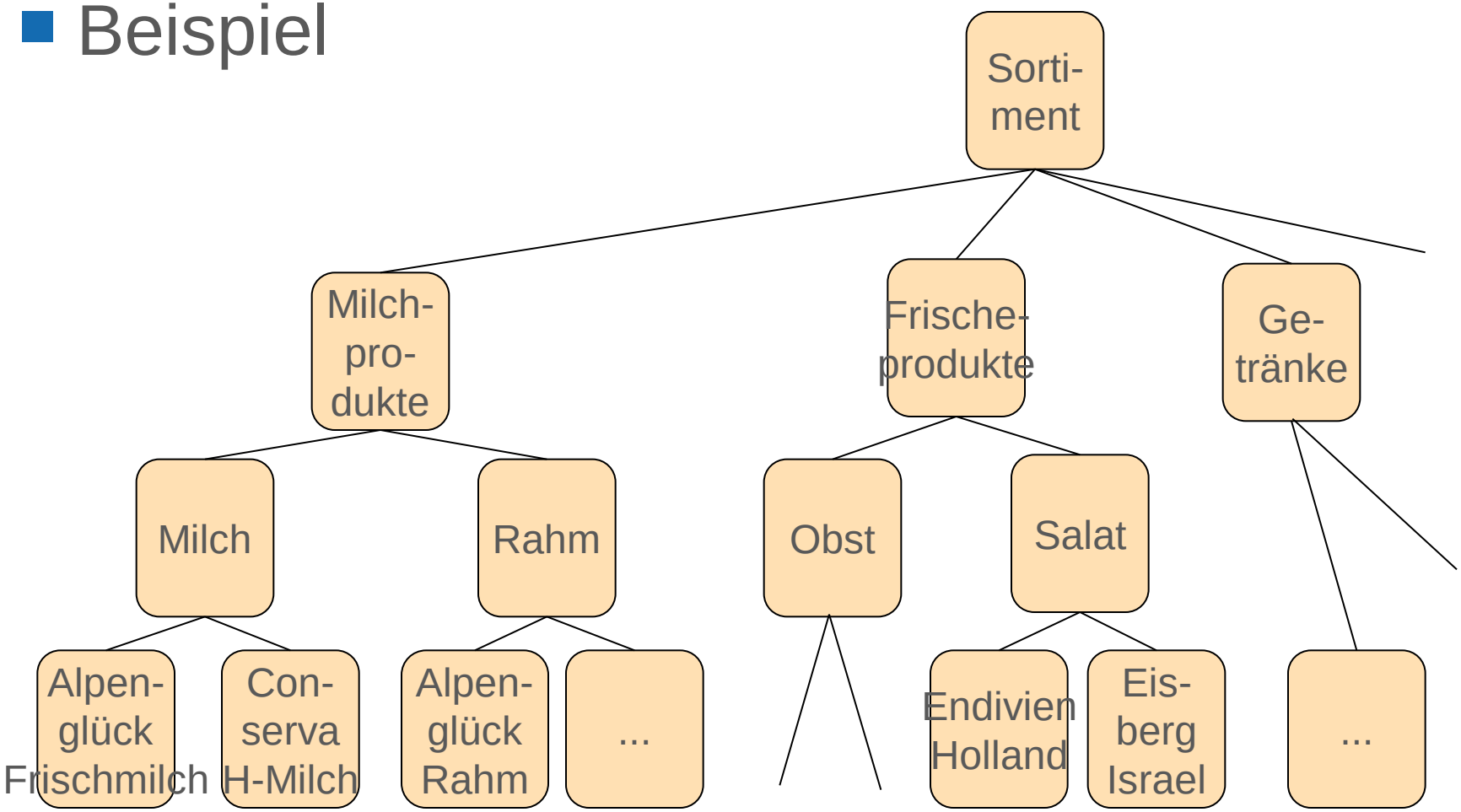
- Grundlagen der Assoziationsanalyse
- Entdecken von Assoziationen
- systematische Ableitung von Assoziationsregeln: A priori-Algorithmus
- Korrelationsanalyse
- **Mehrebenen-Assoziationsanalyse**

Finden von Assoziationen

- Häufige Schwierigkeit
  - verstreut über Dimensionen verteilte und granular vorliegende Daten
  - kaum aussagekräftige, relevante Assoziationen zwischen einzelnen Datenelementen festzustellen
- Ausweg: Über Konzepthierarchie
  - Möglichkeit, auf höheren Hierarchiestufen, wo die Daten konzentrierter vorliegen, nach Assoziationen zu suchen

Konzepthierarchie

- Beispiel



Mehrebenen-Assoziationsanalyse

- Jede Hierarchiestufe nach Assoziationen durchsuchen
- top-down
- z.B. mit dem A-priori-Algorithmus
- Mehrere Varianten möglich
  - Genereller Mindestsupport
  - Differenzierter Mindestsupport

Genereller Mindestsupport

- Mindest-Support ist auf allen Hierarchiestufen gleich
- Vorteile
  - Nur ein Wert einzustellen
  - Alle Subtrees unterhalb eines Vaterknotens mit ungenügendem Support müssen nicht mehr berücksichtigt werden
- Gefahren
  - Mindestsupport zu hoch gewählt:
    - Interessante Muster auf tieferen Ebenen unentdeckt
  - Mindestsupport zu niedrig gewählt:
    - viele uninteressante Assoziationen (obere Ebenen)
    - Performance-Probleme

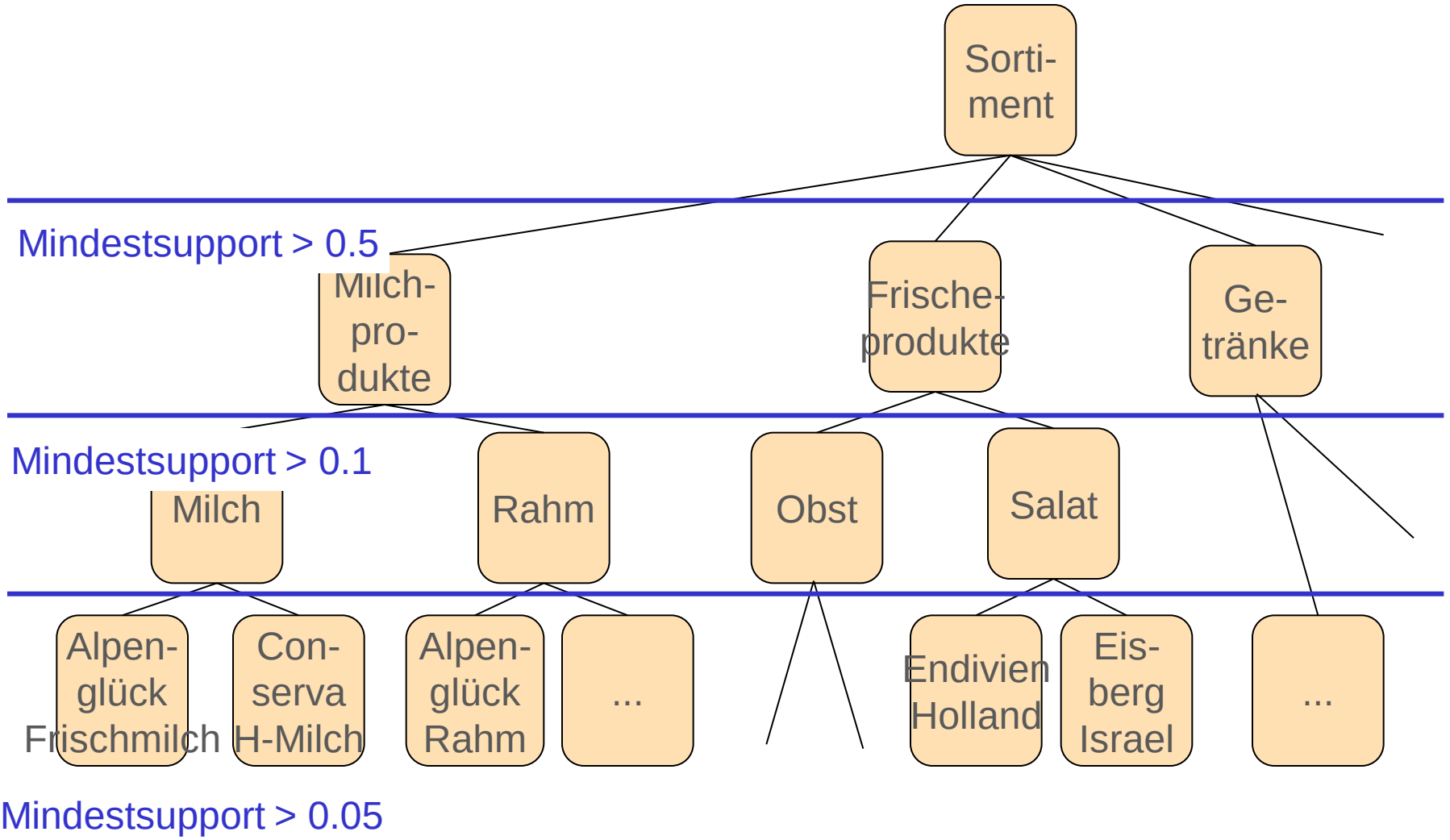
Differenzierter Mindestsupport

- Reduzierter Support auf tieferen Stufen
- Je tiefer die Abstraktionsebene, umso niedriger der geforderte Mindestsupport
- Wiederum mehrere Varianten möglich
  - Mindestsupport ist unabhängig von Häufigkeit bestimmter Items auf höheren Konzepthierarchiestufen
  - Filterung aufgrund von Information aus einer höheren Ebene
    - filtering by single item
    - filtering by k-itemset
    - filtering by level passage threshold

Support unabhängig von anderen Stufen

- Stufenbezogene Unabhängigkeit
  - "level independent reduced support"
- Die Häufigkeit von Items auf höheren Konzepthierarchiestufen beeinflusst die Untersuchung nicht
- Jeder Knoten muss geprüft (Auftreten gezählt) werden, da er häufig sein könnte
- Somit werden eben gerade viele nichthäufige Items durchsucht
- Performancehungrig

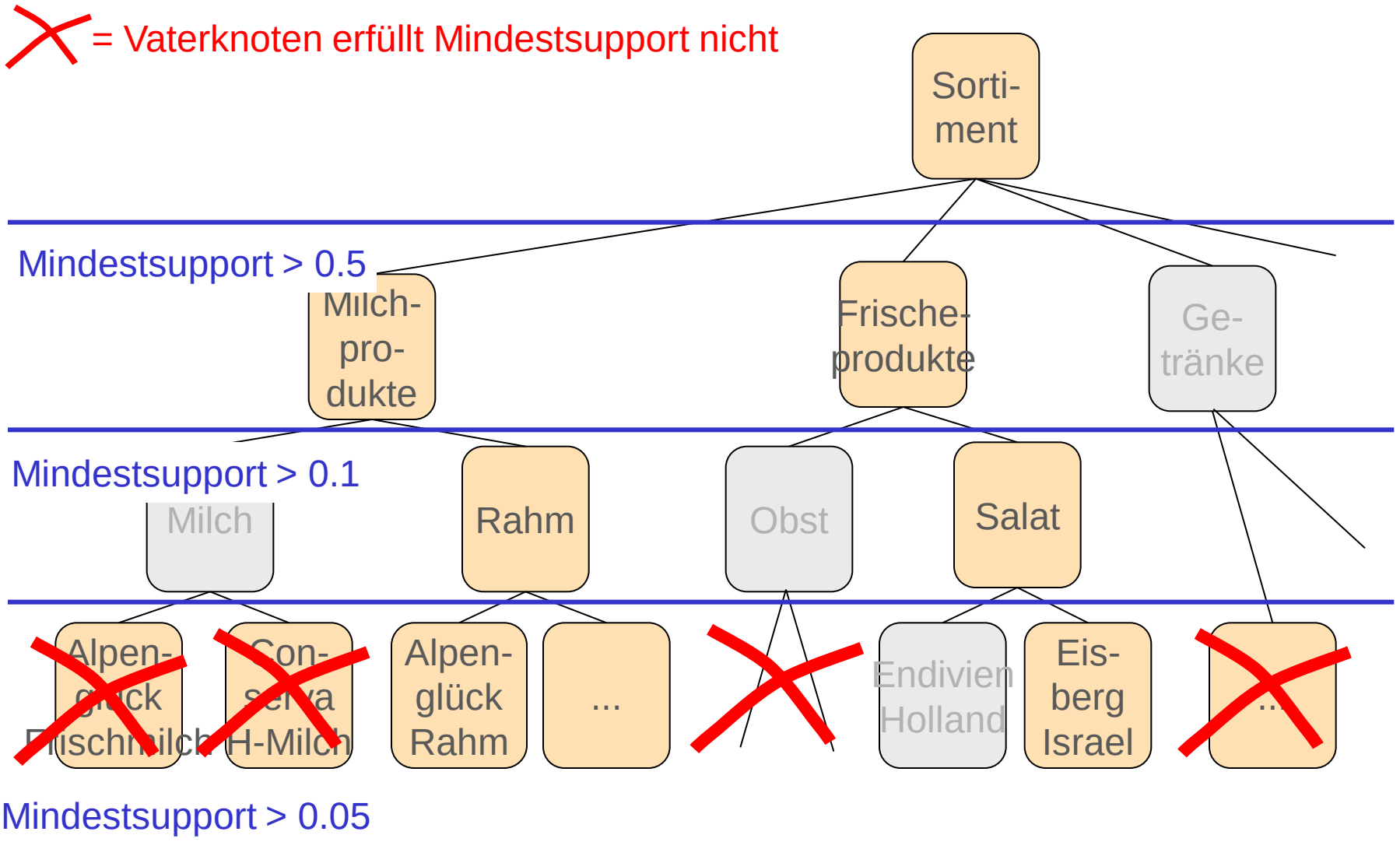
Beispiel "level independent reduced support"



Filtern nach Vorgängerknoten

- Stufenübergreifendes Filtern **pro Item**
- Items einer Hierarchiestufe nur dann durchsucht, wenn ihr Vaterknoten bereits häufig war
  - auf seiner Hierarchiestufe den dort vorgeschriebenen Mindestsupport erfüllt hat
- bestimmte Verzweigungen der Konzepthierarchie werden so ausgeklammert
- Gefahr:
  - Items, die auf ihrer Hierarchiestufe über dem dort geltenden Mindestsupport liegen, also interessante Muster bilden, werden nicht erkannt, wenn ihre Vaterknoten den für sie geltenden Support nicht erfüllten.

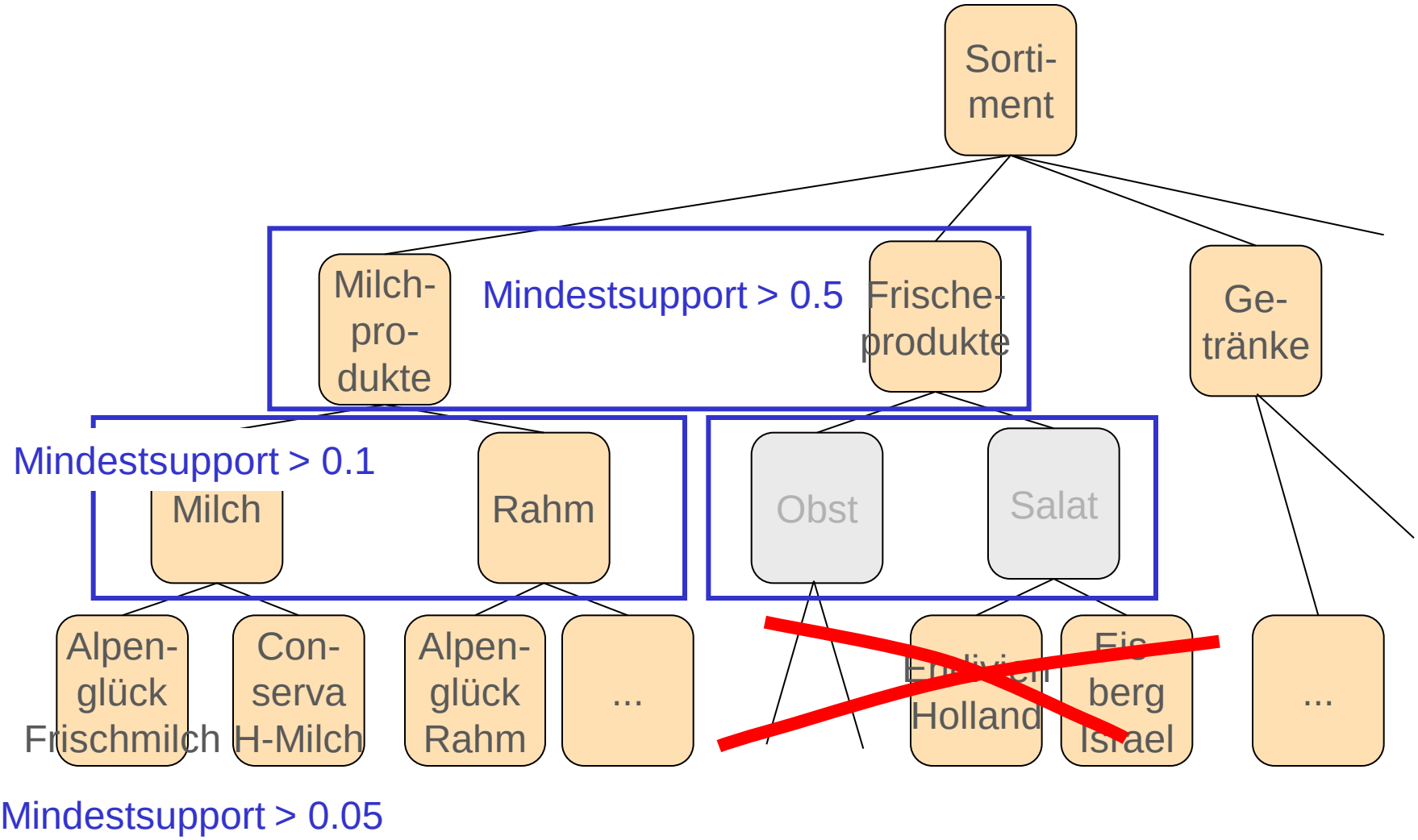
Beispiel "filtering by single item"



Filtern nach Gruppen von Vorgängerknoten

- Stufenübergreifendes Filtern **mit itemsets**
- Ein k-itemset wird nur dann durchsucht, wenn das entprechende k-itemset auf der nächsthöheren Stufe bereits häufig ist
- Sehr strenge Einschränkung
- Gefahr:
  - viele interessante Assoziationen werden ausgesiebt

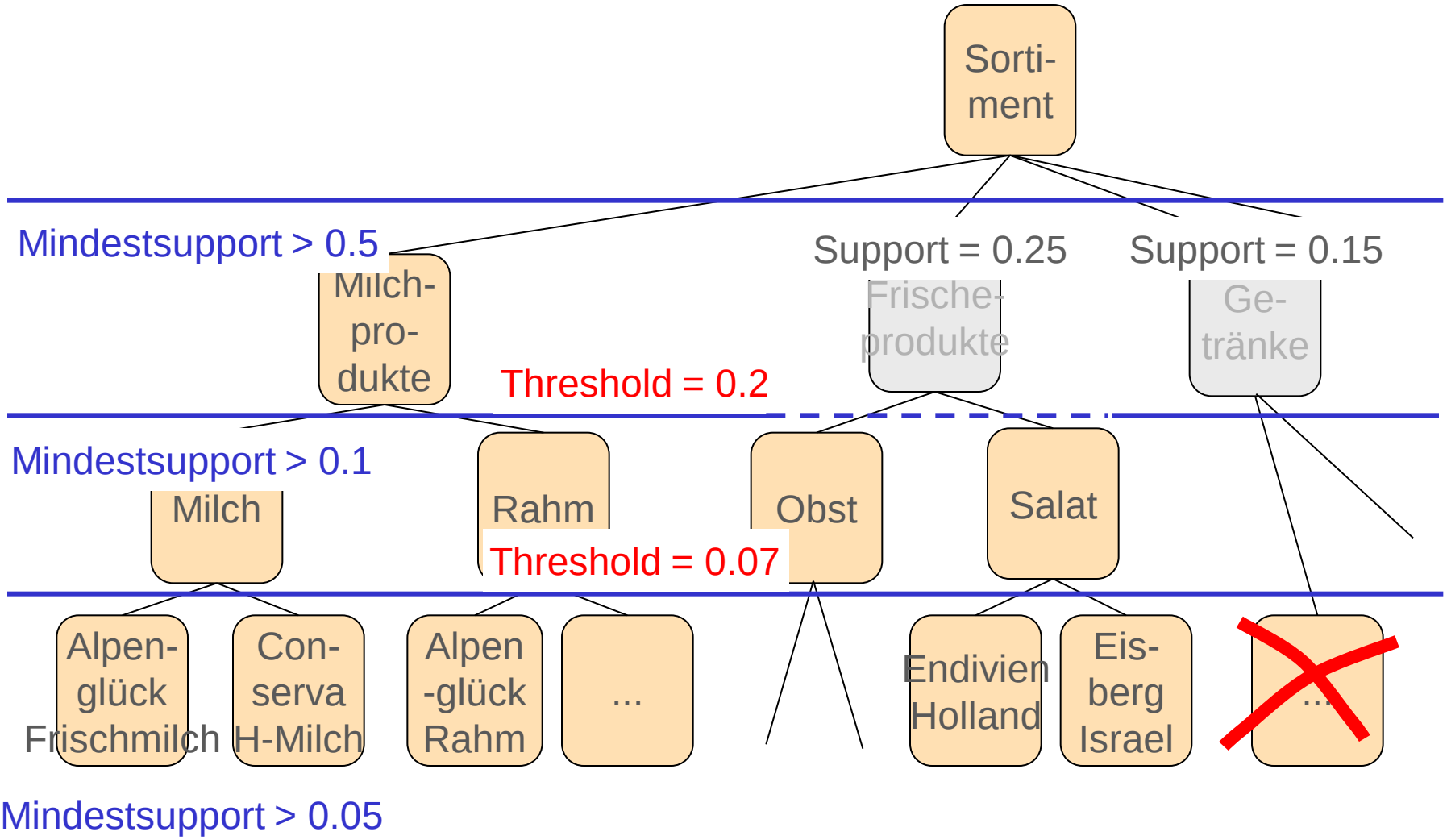
Beispiel "filtering by k-itemset"



Filtern mit "Durchlass"

- **Geregeltes** Stufenübergreifendes Filtern **pro Item**
- Einbau einer zusätzlichen Schwelle:
  - Stufendurchlass (level passage threshold)
  - Der Stufendurchlass liegt zwischen dem Mindestsupport der aktuellen Konzepthierarchieebene und dem der nächsttieferen Ebene
- Identifikation 'relativ häufiger' (*subfrequent*) Items
- Nachfolgeknoten von nichthäufigen items werden trotzdem durchsucht, falls ihr Vaterknoten wenigstens 'relativ häufig' ist.
- Senken des Stufendurchlasses führt zur Betrachtung von mehr Nachkommen nichthäufiger Items.

Beispiel "level passage threshold"



Regeln über Hierarchieebenen hinweg

- Stufenübergreifende Assoziationsregeln
  (cross level association rules)
- Beispiel:
  – Endivien Holland → Milchprodukte
- Mindestsupport der konkreteren Stufe muss auf alle betrachteten Stufen angewendet werden.

Grenzen Assoziationsanalyse

- Regeln, die sich aus der Natur der Daten ergeben, sind nicht interessant
  - besucht(Data Mining) $\Rightarrow$ studiert(Informatik)
- Nicht alle Regeln mit hoher Confidence sind interessant
  - Dass Farbe und Pinsel zusammen gekauft werden, ist nicht überraschend
- Regeln können Wirklichkeit falsch abbilden, zumindest Zusammenhänge falsch suggerieren
- Regeln können auch sinnlos sein
  - Regeln mit hoher Confidence, die zufällig zustande kommen, für die aber keine Erklärung existiert und ein tieferer Zusammenhang nicht erkennbar ist
  - Solche Regeln haben meist geringen Support

Hausaufgaben

- Lösen von Aufgaben zur Assoziationsanalyse mit WEKA
- Sie finden die Aufgaben auf Wiki

Ausblick

- ☑ Von Decision Support zu Data Warehouses
- ☑ OLAP – Online Analytical Processing
- ☑ Data Mining – Klassifikation
- ☑ Data Mining – Assoziationsanalyse
- **Data Mining – Clustering**

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Data Mining: Clustering

Datenbanksysteme 2

Dr. Klaus Wolfertz

01. Juni 2010

Überblick

- ☑ Von Decision Support zu Data Warehouses
- ☑ OLAP – Online Analytical Processing
- ☑ Data Mining – Klassifikation
- ☑ Data Mining – Assoziationsanalyse
- ■ **Data Mining – Clustering**

Agenda

Data Mining – Clustering

- **Grundlagen**
- Segmentierungsverfahren
- Abstandsmessung
- Ausreisser-Analyse

- Data Mining Wrap Up

Clustering

= Segmentierung

- Einteilen von physischen oder abstrakten Objekten in Klassen gleicher oder ähnlicher Objekte
- *Was wir alle seit der Kindheit machen*

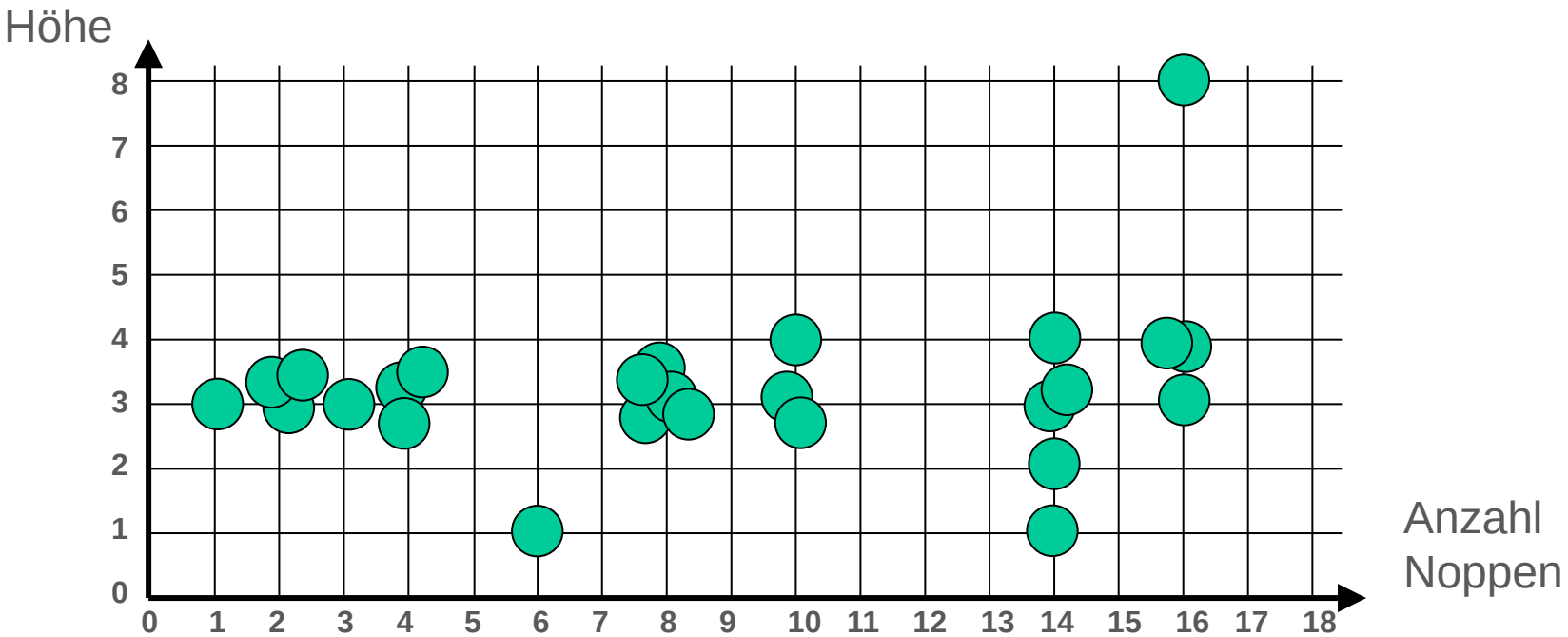
Vorgehen

- Trainingsmenge in Teilmengen zerlegen (**segmentieren**)
- Prüfen, ob
  - die Instanzen in einer Gruppe sich so **ähnlich** wie möglich sind,
  - während sie möglichst **unähnlich** zu Instanzen in anderen Gruppen sind.
- Einteilung in eine überschaubare Anzahl von Gruppen

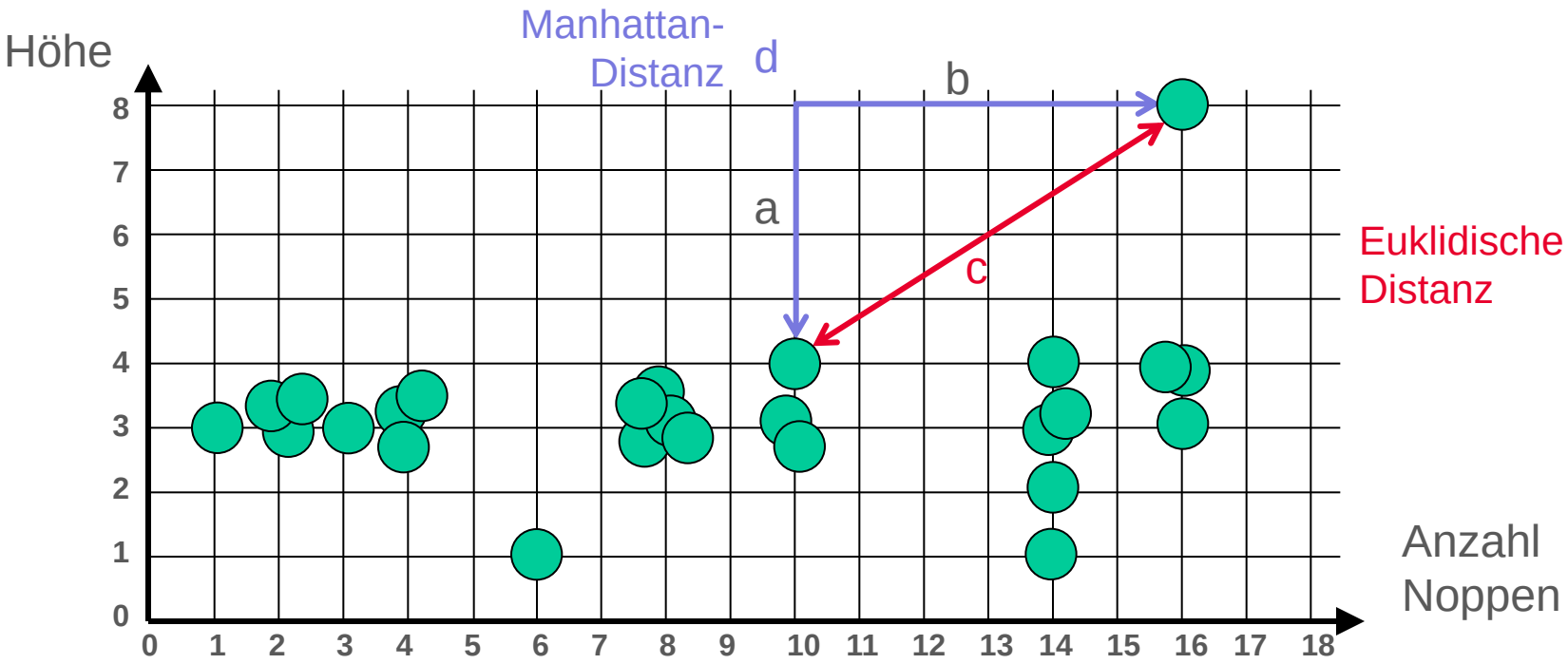
Ähnlichkeit

- Wie misst man "Ähnlichkeit"?
- Ähnlichkeit von 2 Objekten typischerweise bestimmt anhand der Entfernung voneinander
  - je kleiner die Entfernung, um so ähnlicher
  - je grösser die Entfernung, um so unähnlicher
- Wie misst man Entfernungen?

Verortung in einem Koordinatensystem

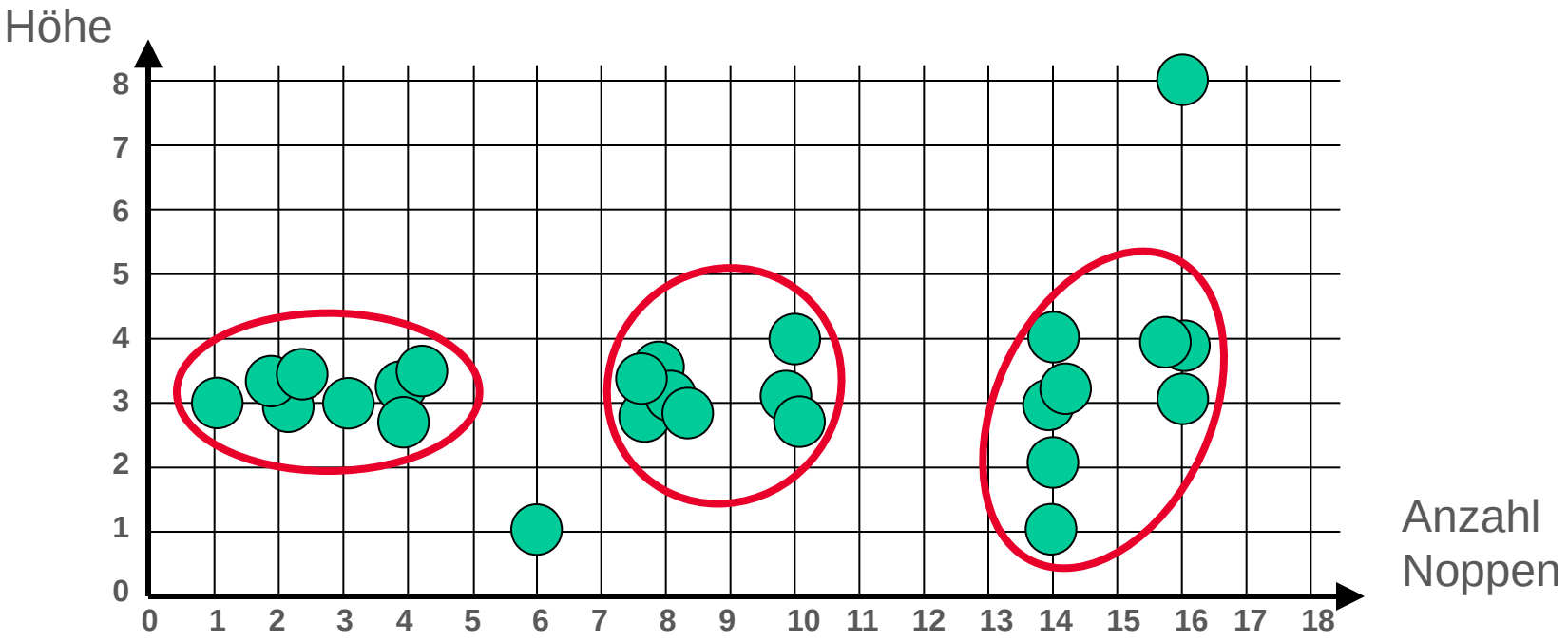


Gebräuchliche Distanz-Metriken



- Euklidische Distanz $c = \sqrt{a^2 + b^2}$
- Manhattan-Distanz $d = a + b$
(city block distance)

Erkennen von Häufungen



- Entdecken von Gruppen ähnlicher Objekte
  - Konzentrationen im Koordinatensystem
- Breite Palette an Algorithmen hierfür

Ziele der Segmentierung

- Einblick in die Verteilung der Daten
  - z.B. welche Kundensegmente gibt es?
- Charakteristik einzelner Cluster erkennen
  - z.B. Besonderheiten von Kundensegmenten
- Bestimmte Cluster identifizieren
  - z.B. das Segment mit der höchsten Profitabilität für tiefergehende Analyse
- Vorbereitung für andere Mining-Verfahren
  - Klassen definieren für anschliessende Bestimmung eines Klassifikationsmodells

unsupervised learning

- Segmentierung = unüberwachtes Lernen
  - unsupervised learning
  - im Gegensatz zum überwachten Lernen, z.B. der Klassifikation, wo die Klassen bekannt sind
- Klassen bei der Segmentierung nicht bekannt
  - müssen erst herausgefunden werden
  - man spricht auch von "Klassenbildung"

Anwendungsbeispiele

- Pflanzen- und Tiertaxonomien
- Gene mit ähnlicher Funktion
- Kundengruppen
  - Neu-, Wiederhol-, Stammkunden
  - geographische Verteilung
  - Verhalten auf der Website
- Versicherungsprämienklassen
  - Einteilung von Versicherungsnehmern in Gruppen mit ähnlichem Risiko

Agenda

Data Mining – Clustering

- Grundlagen
- **Segmentierungsverfahren**
- Abstandsmessung
- Ausreisser-Analyse

- Data Mining Wrap Up

Segmentierungsverfahren

hauptsächliche Ansätze

- Partitionierung
- Hierarchische Methoden
- Dichtebasiert
- Modellbasiert

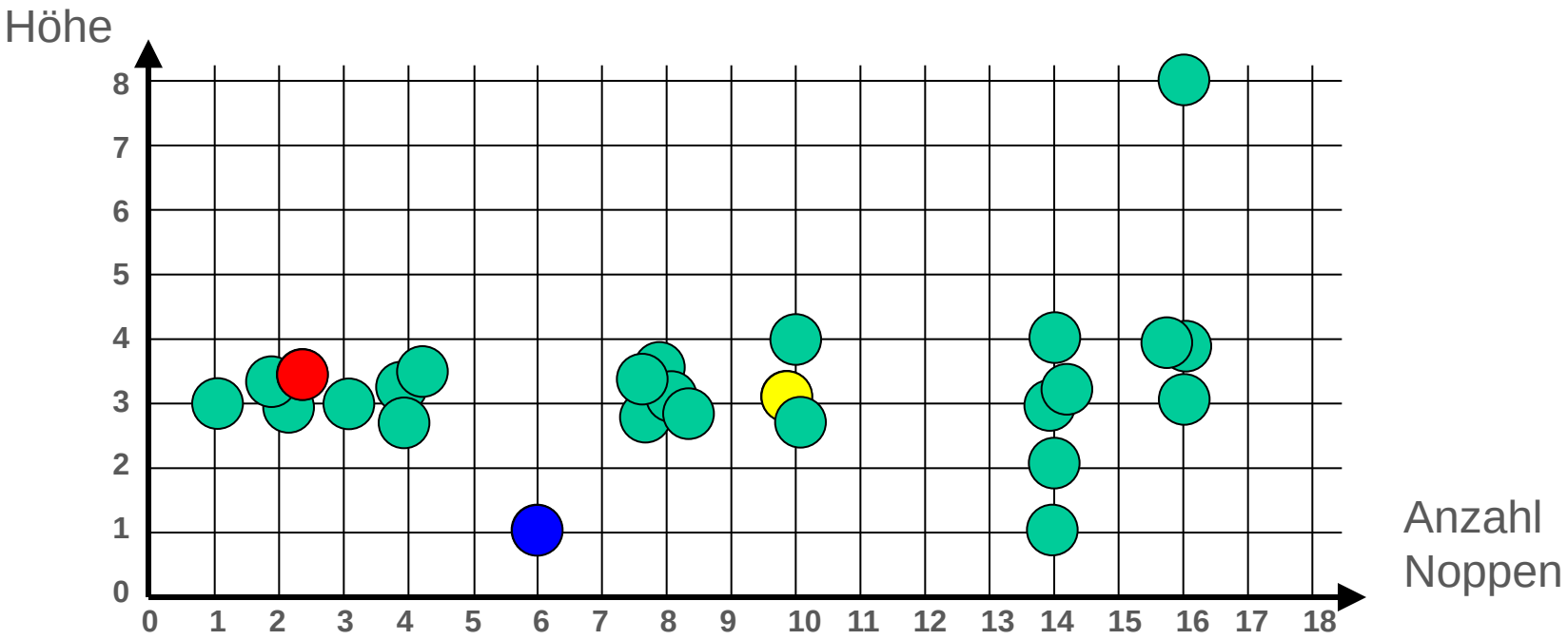
Partitionierung

- Aufteilen in **feste Anzahl** Gruppen
  - jede Gruppe enthält mindestens 1 Instanz
  - jede Instanz gehört zu genau 1 Gruppe
- iteratives Reallozieren der Instanzen
  - bis bestimmtes Gütekriterium erreicht
- Bekanntester Algorithmus: *k-means*

k-Mittelwerte-Algorithmus (k-means)

- wähle k
  - k ϵ ℕ und k < Anzahl Instanzen → = Anzahl Cluster
- wähle k Instanzen zufällig aus
  - dies sind die anfänglichen Clustermittelpunkte
- repeat
  - für jede Instanz
    - berechne Distanz zu allen Clustermittelpunkten
    - ordne Instanz dem nächstgelegenen Cluster zu
  - falls sich keine Änderungen in den Clusterzuordnungen mehr ergeben: **Ende**
  - Bestimme die geometrische Mitte für jeden Cluster und lege sie als neuen Clustermittelpunkt fest

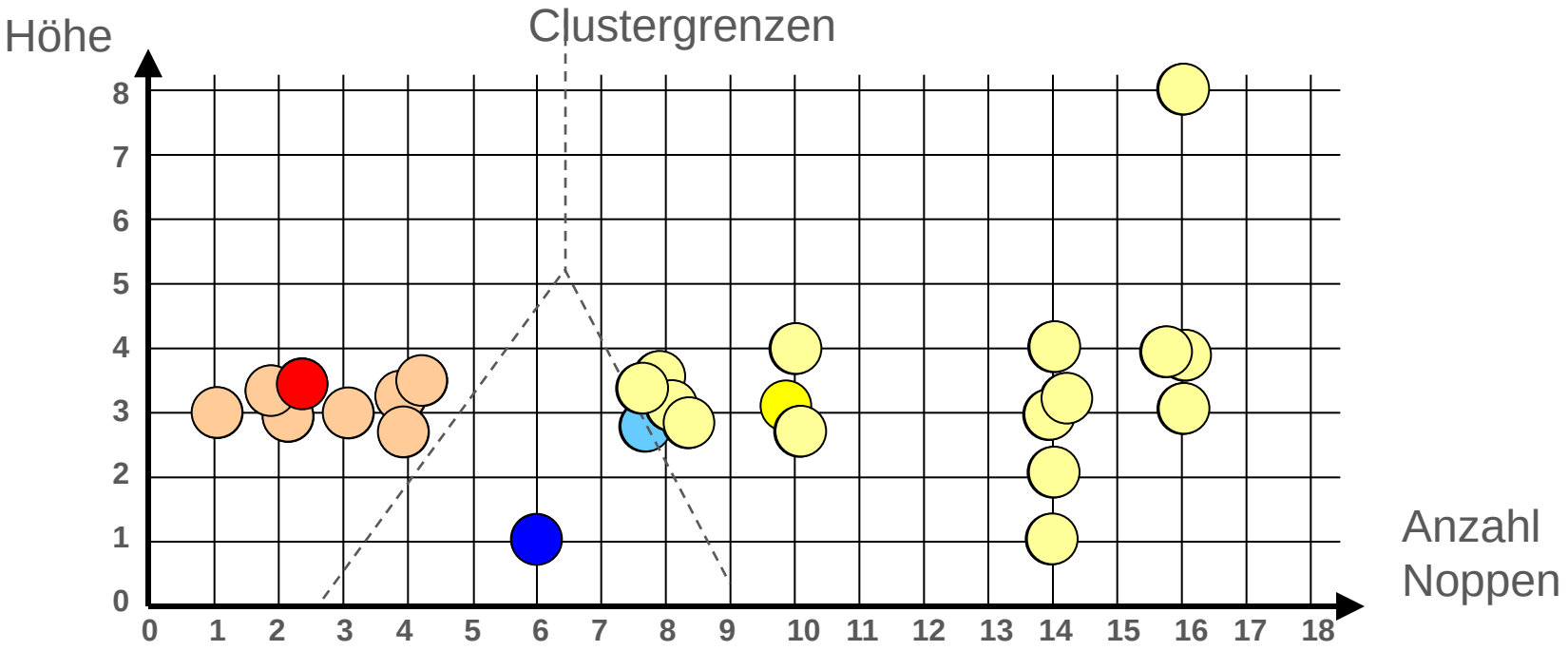
k-means - Initialisierung



Beispiel:

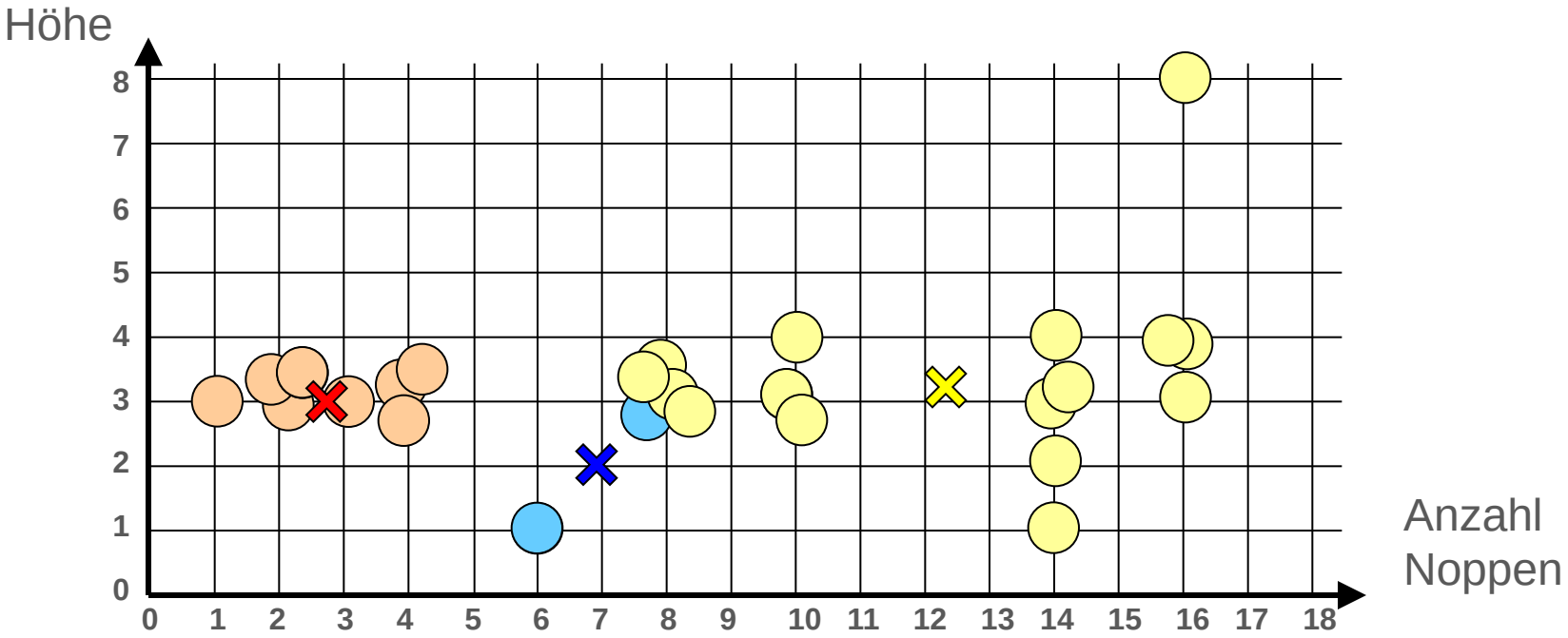
- k wird – versuchsweise– auf 3 gesetzt
- 3 zufällige Clustermittelpunkte gewählt

k-means - Clusterzugehörigkeit bestimmen



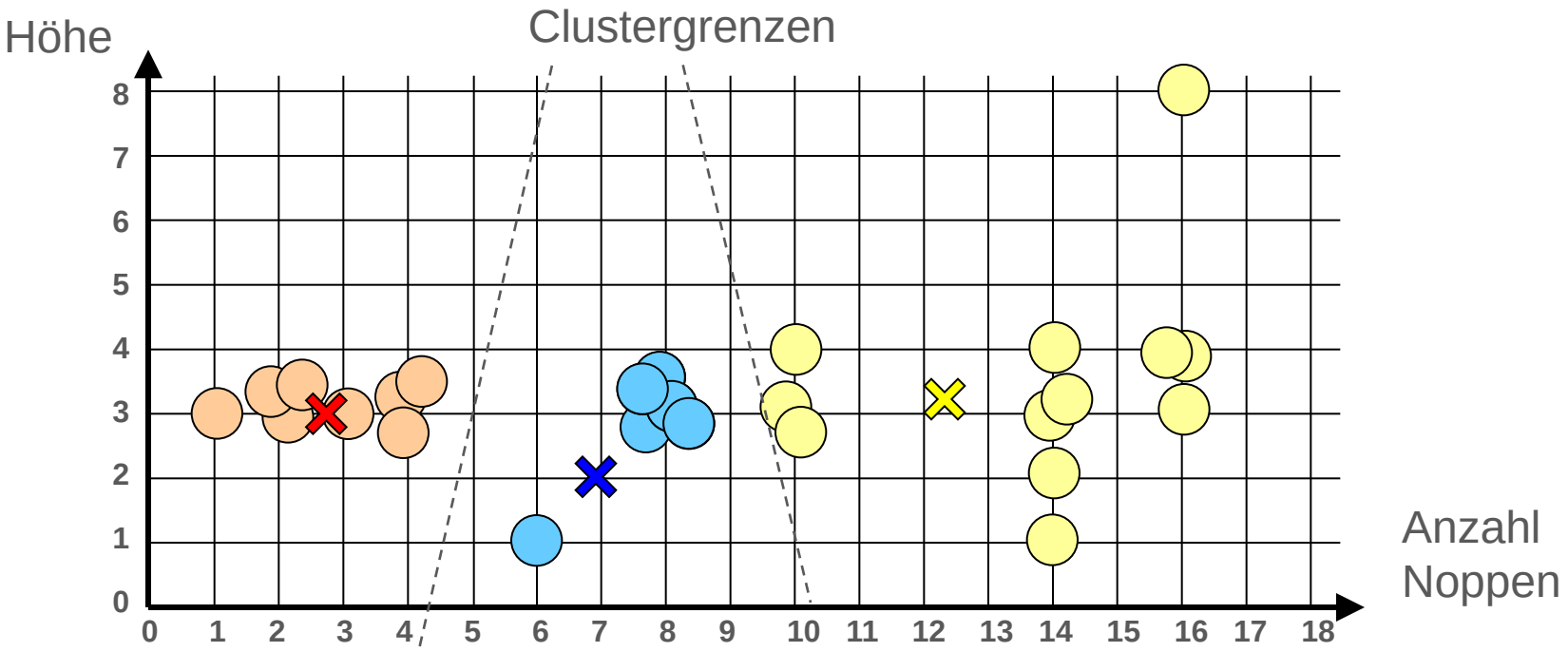
- jede Instanz wird dem Cluster des nächst-liegenden Mittelpunkts zugeschlagen

k-means - neue Mittelpunkte bestimmen



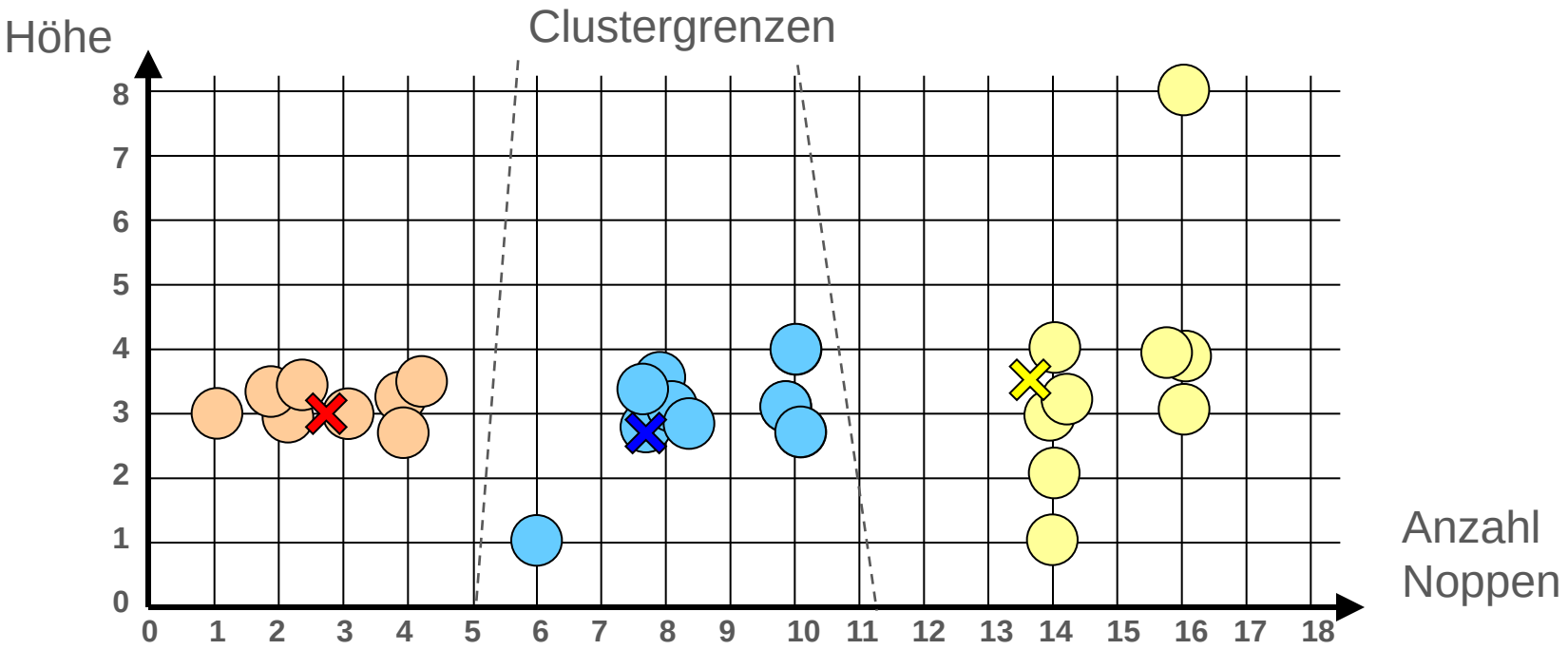
- aus dem geometrischen Mittel jedes Clusters wird dessen neuer Mittelpunkt bestimmt

k-means - Clusterzugehörigkeit überprüfen



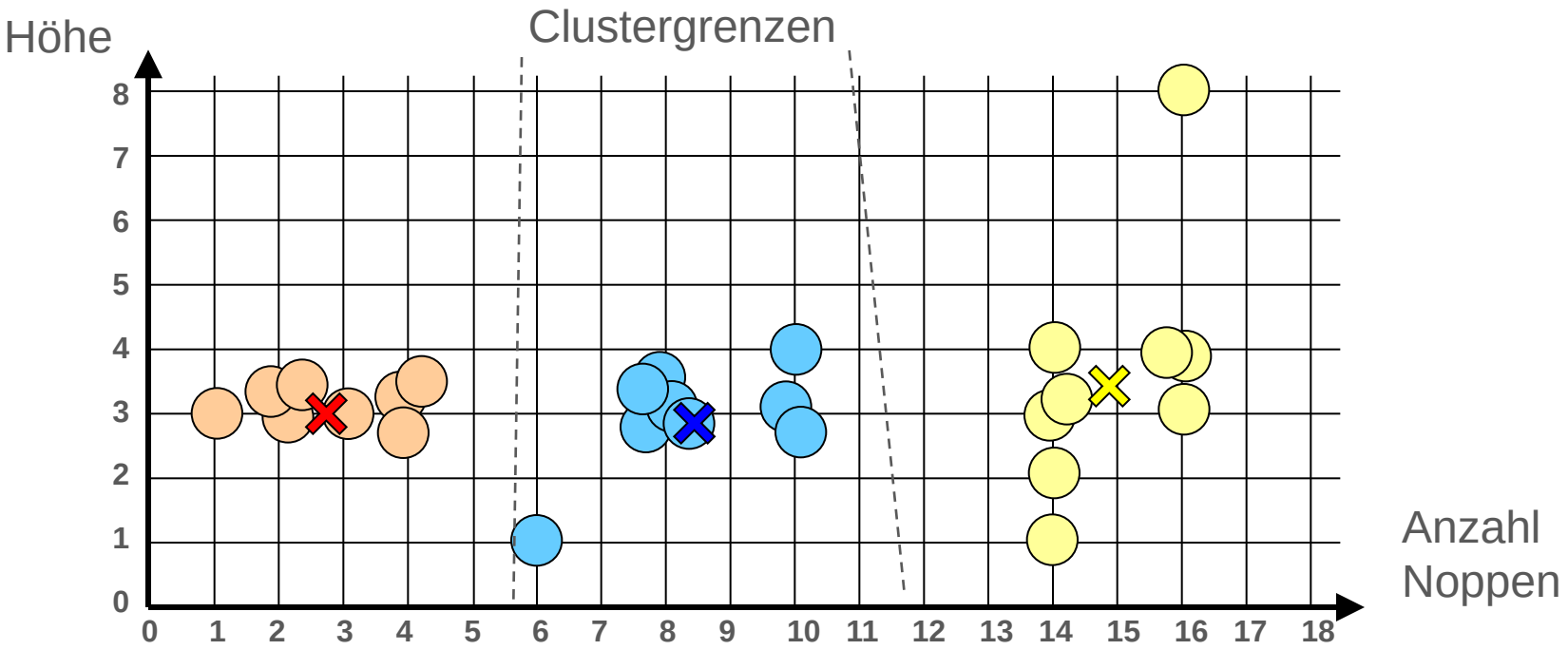
- wieder jede Instanz dem nächstliegenden Clustermittelpunkt zuordnen

k-means - Clusterzugehörigkeit überprüfen



- neue Clustermittelpunkte bestimmen
- Instanzen gegebenenfalls neu zuordnen

k-means - Endzustand



- neue Clustermittelpunkte bestimmen
- keine Veränderung in der Zuordnung mehr
- **Ende** des Algorithmus

Beurteilung k-means-Algorithmus

- schnell, auch bei grossen Datenmengen
- konvergiert rasch, hin zu lokalen Optima
- am besten geeignet für kompakte, klar getrennte kreis- oder kugelförmige Cluster
- Ergebnis stark abhängig von Initialisierung:
  - Anzahl der Cluster
  - anfänglich gewählte Instanzen (Mittelpunkte)
- hochempfindlich gegenüber Ausreissern
  - Clustermittelpunkte verzerrt in Richtung Ausreisserkoordinaten

Hierarchische Methoden

- Hierarchische Zerlegung der Instanzenmenge

in 2 Richtungen möglich:

- bottom-up (zusammenfassend)
  - jede Instanz = 1 Gruppe, sukzessives Verschmelzen, bis Gütekriterium erreicht
- top-down (aufteilend)
  - Alle Instanzen in 1 Gruppe, sukzessives Aufsplitten, bis Gütekriterium erreicht

Dichtebasierte Methoden

- Platzieren der Instanzen in einem Koordinatensystem
- Mass definieren, wie dicht Instanzen beieinander liegen
  - z.B. Abstand zum nächsten Nachbarn
- Festlegen der Clustergrenzen durch Schwellwert des Dichtemasses
  - Dichtemass über Schwellwert: im Cluster
  - Dichtemass unter Schwellwert: nicht im Cluster

Modellbasierte Methoden

wahrscheinlichkeitsbasiert

- Berechnen eines Qualitätsmasses
  - Wahrscheinlichkeit, dass Instanz im Cluster X einen bestimmten Attributwert hat
  - Wahrscheinlichkeit, dass Instanz mit bestimmtem Attributwert sich im Cluster X befindet
- Entscheidung, ob neue Instanz in ein existierendes Cluster kommt oder Cluster umgebildet werden
  - anhand des Qualitätsmasses

Modellbasierte Methoden

mit neuronalen Netzen

- **S**elf-**O**rganizing feature **M**aps – **SOM**
  - auch genannt: Kohonen-Netze
- "Landkarte" (Map)
  - Anfangsanordnung an "Kartenneuronen"
  - iterativ werden die Kartenneuronen so angeordnet, dass sie immer besser der Verteilung der Instanzen entsprechen
- Competitive learning
  - Beste "Landkarte" bleibt erhalten

Probleme des Clustering

- Skalierbarkeit und Performanz der Algorithmen
- unterschiedliche Attributtypen
- Abhängigkeit von der Initialisierung und Verarbeitungsreihenfolge der Instanzen
- Erkennen unregelmässiger Clusterformen
- Empfindlichkeit gegenüber verrauschten Daten
- Erkennen mehrdimensionaler Cluster

Agenda

Data Mining – Clustering

- Grundlagen
- Segmentierungsverfahren
- **Abstandsmessung**
- Ausreisser-Analyse

- Data Mining Wrap Up

Probleme der Abstandsmessung

- Eintrag in Koordinatensystem - und damit Abstandsmessung - einfach bei kardinalen Skalen:
  - Preis, Alter, Anzahl gekaufter Artikel etc.
- Was tun bei ordinalen oder gar nominalen Skalen?
  Also wie misst man den Abstand
  - zwischen weiss und rot?
  - zwischen Bier und Windeln?
  - zwischen Krankheiten?

Entfernung bei binären Daten (1)

Wertebereich binär

- Beispiel: 2 Patienten mit Kinderkrankheiten
  - Instanz i:
    - Röteln nein
    - Scharlach nein
    - Mumps ja
    - Windpocken ja
  - Instanz j:
    - Röteln nein
    - Scharlach nein
    - Mumps nein
    - Windpocken ja

Kontingenztabelle				
		Instanz i		
		0 (nein)	1 (ja)	Summe
Instanz j	0 (nein)	**2**	**1**	**3**
	1 (ja)	**0**	**1**	**1**
	Summe	**2**	**2**	**4**

Entfernung bei binären Daten (2)

- Binäre Skala
  - Unähnlichkeits-berechnung:
    d = (0+1)/4=1/4

Kontingenztabelle				
beim Jaccard-Koeffizienten unterdrückt		Instanz i		
		0	1	Summe
Instanz j	0	**2**	**1**	**3**
	1	**0**	**1**	**1**
	Summe	**2**	**2**	**4**

  - gilt für symmetrische Daten (männlich/weiblich)
  - für asymmetrische (Grippevirus positiv, negativ) wird die Anzahl der negativen Übereinstimmungen unterdrückt
  - Jaccard-Abstand: d = (0+1)/(4-2) = 1/2

Entfernung bei nominalen Daten

- Nominalskala
  - 1. Attr.: rot, gelb, blau, grün, weiss
  - 2. Attr.: Kaffee, Tee, Schokolade
  - 3. Attr.:....

$$d = \frac{\text{Anzahl aller Attribute} - \text{Anzahl Entsprechungen}}{\text{Anzahl aller Attribute}}$$

- Ordinalskala
  - Bewertung der nominalen Daten mit einem Rang (z.B.: Gold, Silber, Bronze)
  - dann normale Berechnung der euklidischen oder Manhattan Distanz

Normalisierung

- Abstandsmessung wird beeinflusst durch die Wahl der Masseinheit
- daher: Umrechnung der Daten, damit sie in einen bestimmten Bereich fallen,
  z.B. 0 ... 1
- Division durch Maximalwert

  oder
- Subtraktion Minimalwert und Division durch Bereich zwischen Minimal- und Maximalwert
  - Beispiel: 11 wird zu (11-8)/(20-8) = 3/12 = 0.25

Eintrag Entfernungen in Unähnlichkeitsmatrix

- Datenmatrix
  - Aufstellung für alle Instanzen

Stein Nr.	Noppen	Höhe	Farbe
1	8	3	blau
2	2	1	weiss
3	4	3	rot

- Unähnlichkeitsmatrix
  - 0 = sehr ähnlich
  - 1 = vollkommen unähnlich

Stein Nr.	1	2	3
1	0		
2	0.65	0	
3	0.3	0.5	0

Agenda

Data Mining – Clustering

- Grundlagen
- Segmentierungsverfahren
- Abstandsmessung
- **Ausreisser-Analyse**

- Data Mining Wrap Up

Was ist ein Ausreisser (engl. Outlier)?

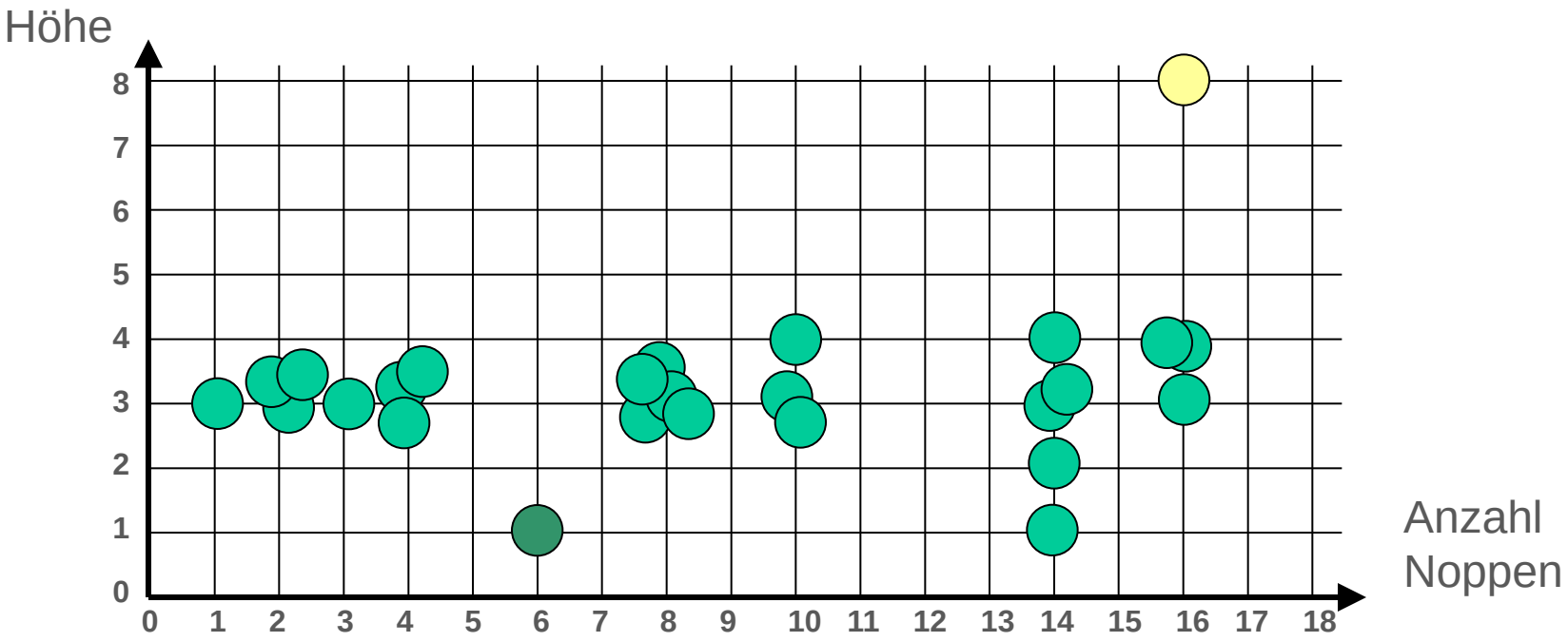
- Instanz unähnlich zu allen anderen Instanzen
  - Gehalt des CEO zum Gehalt aller anderen Mitarbeiter
  - Alter von - 9999 (nicht erfasst oder nicht angegeben)
- Gründe für Ausreisser
  - extremer aber korrekter Wert für eine bestimmte Instanz

  - Messfehler
- Definition eines Ausreissers ist relativ
  - muss letztendlich von Menschen beantwortet werden

Outlier Mining

- Rauschen oder Signal?
  - Ausreisser: stören "normale" Clusteranalyse ("noise")
  - in anderen Fällen gerade von Interesse
- **Outlier Mining** sucht gezielt nach Anomalien:
  - Betrugsentdeckung
  - Geldwäsche
  - spezielle Kundensegmente
  - Nebenwirkungen eines Medikaments

Ausreisser im Koordinatensystem



- Wo ist der Ausreisser?
- Was ist mit der dunkelgrünen Instanz?

Ansätze zur Entdeckung von Ausreissern

- basierend auf statistischen Verteilungen
  - in der Annahme, dass Attributwerte einer Verteilung genügen, z.B. Überlagerung von Normalverteilungen
  - falls Wahrscheinlichkeit, dass eine Instanz zur angenommenen Verteilung gehört, gering → Ausreisser
- dichtebasiert
  - Attributwerte der anderer Instanzen liegen jenseits eines definierten Maximalabstands → Ausreisser
- anhand von Abweichungen
  - Berechnung eines "Unähnlichkeitsmasses" für alle Instanzen insgesamt
  - falls der Ausschluss einer Instanz die Gesamtunähnlichkeit deutlich reduziert → Ausreisser

Clustering mit "Repräsentanten"

- k-means empfindlich gegenüber Ausreissern
- Robuster: **k-medoid**-Algorithmus
  - Cluster stets durch eine Instanz "**repräsentiert**"
- Clustermittelpunkt ist nicht mehr die geometrische Mitte
  - sondern die "zentralste" Instanz (= Medoid)
- Iterativer Austausch des Medoids durch einen Non-Medoid
  - solange die Clusterqualität zunimmt

Messen der Clusterqualität

- Aufstellen einer Kostenfunktion
  - E = Gesamtfehler der Clusterzuordnung (Error) mit C_j als Cluster j mit dem Clustermittelpunkt c_j und p für eine beliebige Nichtmittelpunktsinstanz

$$E = \sum_{j=1}^{k} \sum_{p \in C_j} |p - c_j|$$

- Kostenfunktion
  - misst durchschnittliche Unähnlichkeit zwischen einer Instanz und dem Medoid des Clusters
- Austausch, falls dies den Error verringert

k-medoid Algorithmus

- wähle k
- wähle zufällig k Clustermittelpunkte
- repeat
  - für jede Instanz
    - berechne Distanz zu allen Clustermittelpunkten
    - ordne Instanz dem nächstgelegenen Cluster zu
  - falls sich keine Änderungen in den Clusterzuordnungen mehr ergeben: **Ende**
  - wähle zufällig eine Nichtmittelpunktsinstanz aus
  - berechne die Gesamtkosten dafür, diese Instanz als Clustermittelpunkt zu nehmen: $E_{\text{after swap}} - E_{\text{before swap}}$
  - Austausch falls Gesamtkosten < 0

Hausaufgaben

- Lösen von Aufgaben zum Clustering mit WEKA
- Sie finden die Aufgaben auf Wiki

Agenda

Data Mining – Clustering

- Grundlagen
- Segmentierungsverfahren
- Abstandsmessung
- Ausreisser-Analyse

- **Data Mining Wrap Up**

Was wir gesehen haben

- Clustering
  - Klassen von Instanzen in Daten identifizieren
  - Entdecken von Ausreissern
- Klassifikation
  - Vorhersage der Klasse für unbekannte Instanzen
- Assoziationsanalyse
  - Vorhersage von Mustern von Attributwerten

Was wir nicht gesehen haben

- Regression
  - Vorhersage von kontinuierlichen Werten
- Attributrelevanz-Analyse
  - welches sind die bestimmenden Attribute?
- Konzeptbeschreibung
  - Herausarbeiten und Beschreiben der wesentlichen Eigenschaften eines Datenbestands
- Minimal Description Length
  - Beschränken der Komplexität eines Modells

Was wir nicht gesehen haben

- Zeitreihen- und Sequenzanalyse
  - Entdecken von Trends und Abfolgemustern
- Survival Analysis
  - Vorhersage der Kundenbindung
- Graph Mining und Link Analysis
  - Musterentdeckung in Netzwerken
- Text und Web Mining
  - Analyse von semi- und unstrukturierten Daten (Information Retrieval)

Was wir nicht gesehen haben

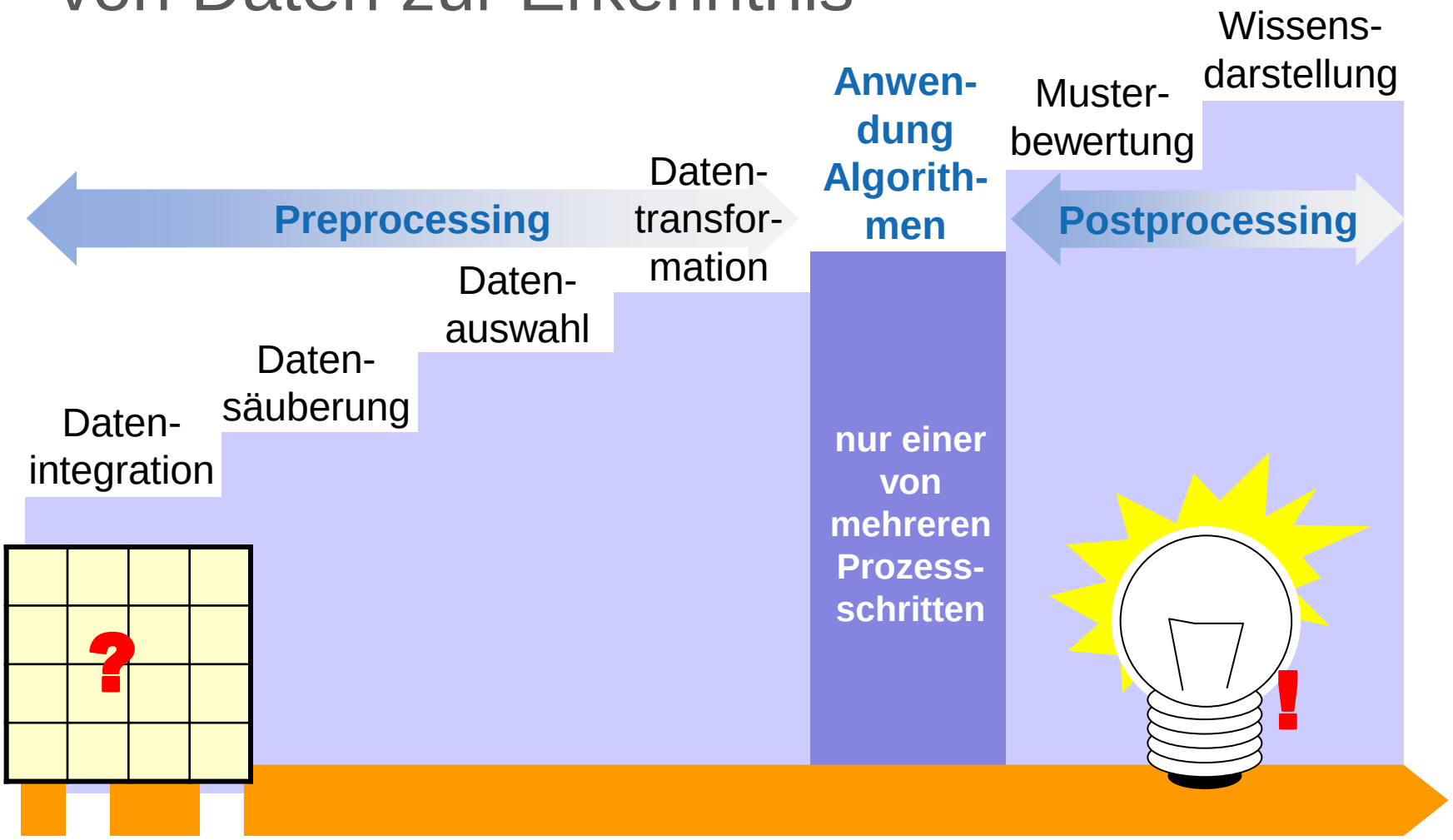
- Vorverarbeitung
  - Datenaufbereitung speziell für Algorithmen
- Validierung
  - Abschätzen der Qualität und Verlässlichkeit von Data Mining-Ergebnissen
- Leistungsvorhersage
  - Vorhersage der Leistungsfähigkeit von Klassifikationsmodellen
- Accuracy Increasing
  - Leistungssteigerung von Klassifizierern

Was wir nicht gesehen haben

- Spatial Data Mining
  - Mustererkennung in Geo-Daten
- Genetische Algorithmen
  - Auswahl und Verbesserung von Modellen über viele "Generationen" hinweg
- Weiterführende Algorithmen zu
  - Klassifikation
  - Assoziationsanalyse
  - Clustering

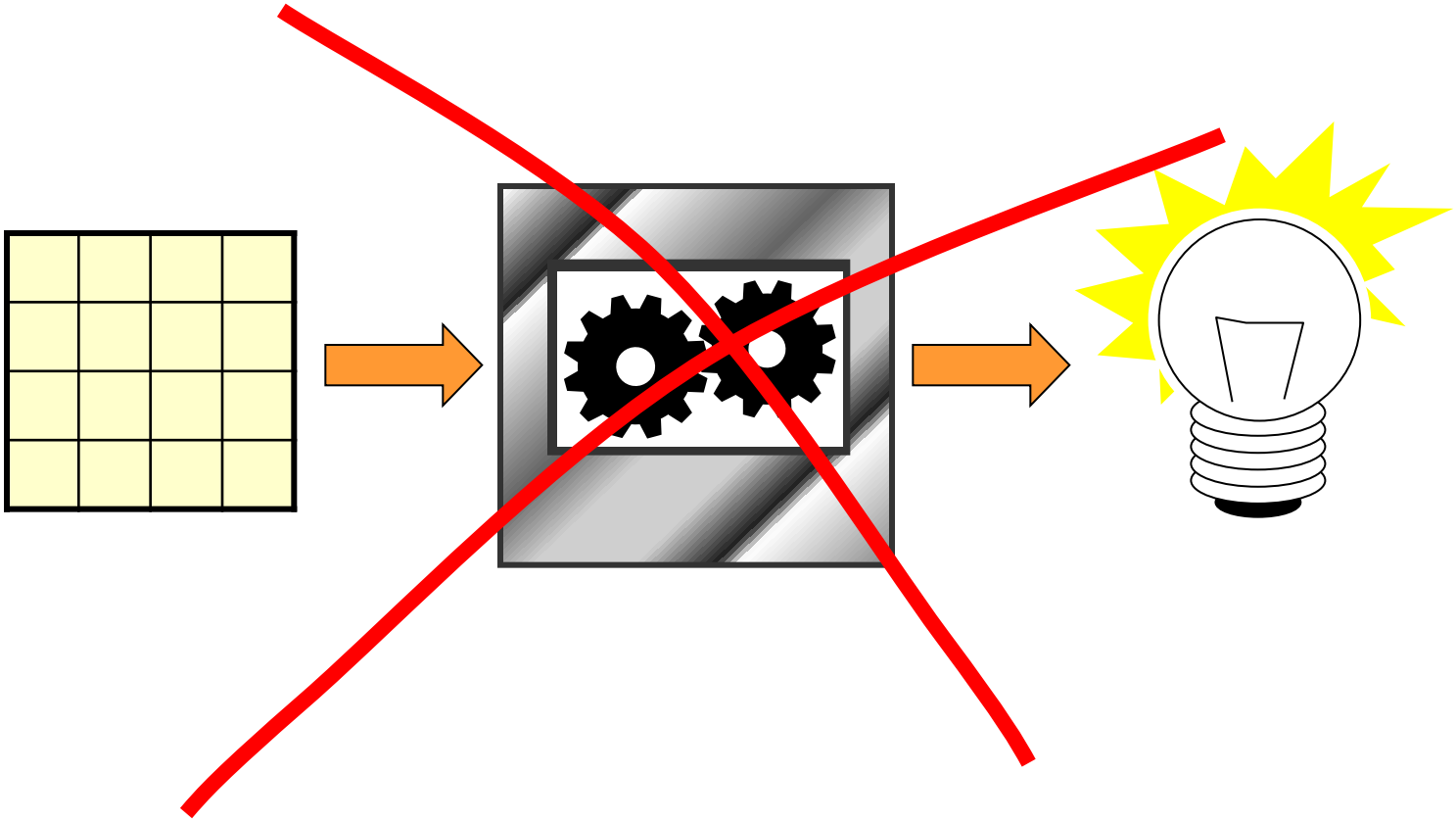
der vollständige Data Mining-Prozess

- von Daten zur Erkenntnis



"automatisches" Data Mining?

- Die "Data Mining Maschine" existiert nicht



Rückblick

- ☑ Von Decision Support zu Data Warehouses
- ☑ OLAP – Online Analytical Processing
- ☑ Data Mining – Klassifikation
- ☑ Data Mining – Assoziationsanalyse
- ☑ Data Mining – Clustering