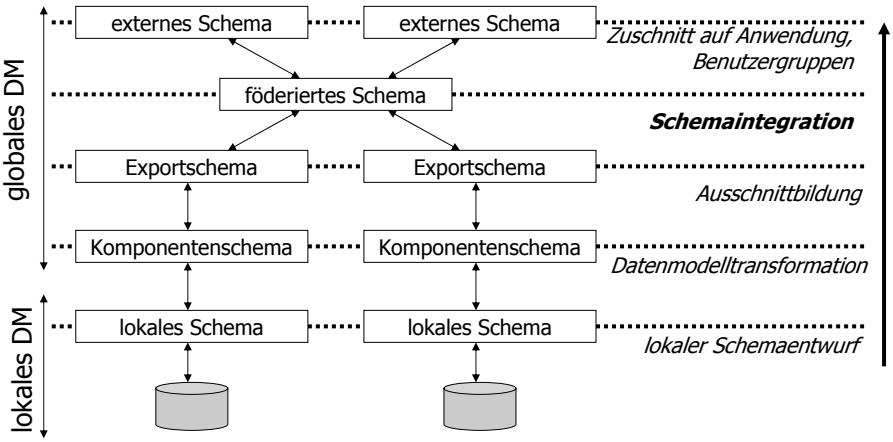
# Kapitel 9
# Schemaabbildung und -integration

## Schemaintegration

- Datenintegration als ein wesentlicher Aspekt der Entwicklung neuer, globaler Informationssysteme
  - Randbedingung: Autonomie der beteiligten lokales Systeme soll erhalten bleiben
- Verschiedene Systemarchitekturen möglich
  - Föderiertes DBS
    - logische Datenintegration
    - Föderierungsdienst integriert lokale Komponentensysteme in ein Gesamtsystem
    - Einheitlicher Zugriff auf integrierten Datenbestand
  - Data Warehouse
    - physische Datenintegration
    - Daten aus verschiedenen Quellsystemen werden in ein gemeinsames Format kopiert
    - komplexe Auswertung und Analyse auf integriertem und aufbereitetem Datenbestand
- Schemaintegration(1) ist ein zentrales Problem

(1)S. Conrad: *Schemaintegration: Integrationskonflikte, Lösungsansätze, aktuelle Herausforderungen,* in Informatik Forschung und Entwicklung 17(3), 2002.

# Schema-Referenzarchitektur






# Komponenten der Schema-Referenzarchitektur

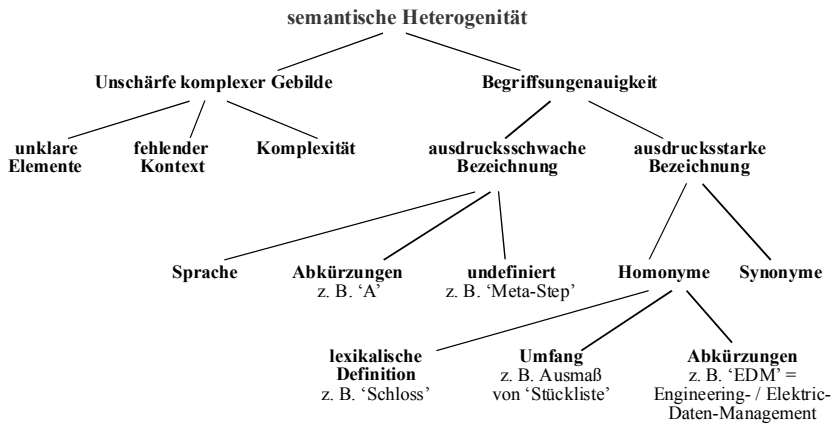
- Lokales Schema
  - entspricht dem konzeptionellen Schema der lokalen Datenbank
  - basiert auf lokalem Datenmodell
- Komponentenschema
  - Beschreibung der lokalen Datenbank in globalem Datenmodell
  - Überwindung der DM-Heterogenität
- Exportschema
  - Beschreibung des Ausschnitts der lokalen Daten, die für globale Anwendungen zur Verfügung stehen sollen
  - Festlegung in der Praxis oft lokal durch das Komponentensystem getroffen
    - "Vertauschen" von Exportschema und Komponentenschema
- Föderiertes Schema
  - beschreibt die Gesamtheit des föderierten Datenbestandes
  - wird durch Schemaintegration gebildet
- Externes Schema
  - entspricht dem klassischen ext. Schema für das föderierte System

# Integrationskonflikte

siehe:
Theo Härder, Günter Sauter, Joachim Thomas: *The Intrinsic Problems of Structural Heterogeneity and an Approach to Their Solution*. VLDB Journal 8(1): 25-43 (1999)

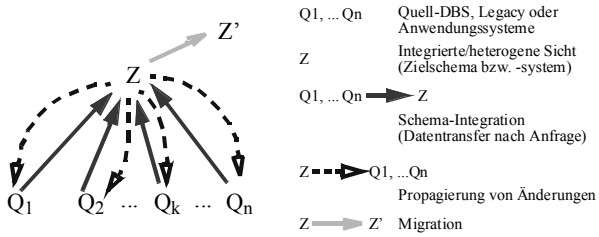
# Semantische Heterogenität

- Bezieht sich auf Modellierungsinhalte



# Strukturelle Heterogenität

- Bezieht sich auf die jeweils zur Verfügung stehenden Modellierungskonstrukte
- Szenario




- Schemata sind *kongruent*, wenn sie semantisch identische Objekttypen und Eigenschaften beinhalten
  - meist inkongruent, überlappend
  - Kongruenzeigenschaft zwischen Quell- und Zielschema (*vertikal*) oder zwischen versch. Quellschemata (*horizontal*)

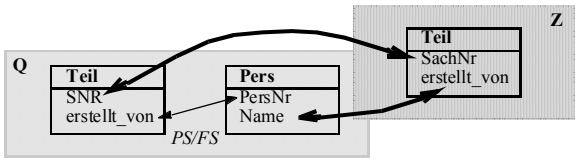


# Probleme der strukturellen Heterogenität

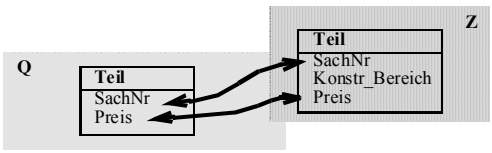
- Problembereiche
  - Elementare Abbildungsprobleme
  - Probleme durch den Einsatz objektorientierter Datenmodelle
  - Abbildungskardinalität
  - Schemakardinalität
  - Weitere Probleme
- Ausgangsszenario (Betrachtung elementarer Probleme)
  - homogene, relationale Schemata
  - transiente Verwaltung der Zieldaten
  - unidirektionale (1:1)-Beziehung: Q $\rightarrow$ Z

# Vertikale Kongruenz

- Projektion: Z beschreibt kleineren Ausschnitt des Anwendungsbereichs als Q
  - View-Update-Problematik
  - Beispiel:



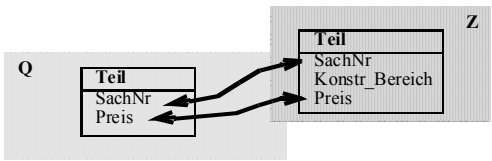

- Q beschreibt kleineren Ausschnitt als Z
  - häufig bei integrierten Sichten
  - Beispiel:





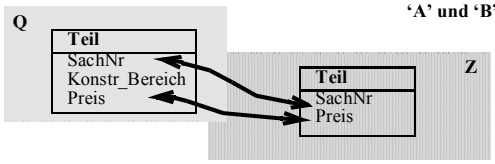

# Vertikale Datenverteilung

- Z verwaltet mehr Daten als Q
  - z. B. abgeleitete Attribute, die in Q nicht enthalten
  - Beispiel: in Q Nullwerte für Preis erlaubt, nicht jedoch in Z




- Selektion: Z erfasst weniger Daten als Q
  - über Prädikate zu erfassen
  - Beispiel

**Prädikat: "nur Konstruktionsbereiche 'A' und 'B' aus Q"**



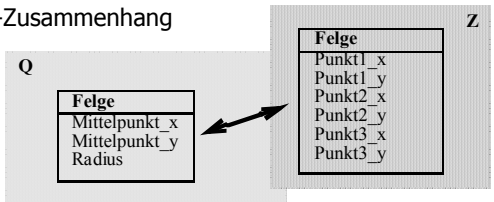
# Datentypkorrespondenz

- Unterstützung unterschiedlicher Datentypen durch Q und Z
- Problem: finde Abbildung, die semantikerhaltende Rücktransformation erlaubt
  - Beispiel: REAL (Q) $\rightarrow$ INTEGER (Z) durch Rundung
    (genaue Rücktransformation nicht möglich)
  - generell: korrekte und vollständige Transformation von Daten eines semantisch mächtigeren Modells in ein semantisch ärmeres Modell kann nicht durchgeführt werden!



# Attributkorrespondenz

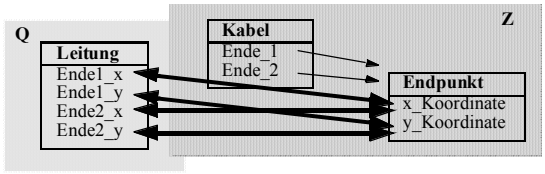
- Zusammenhang zwischen Attributen aus Q und Z
  - Verknüpfung/Aggregation 'nach' Z
  - Trennung 'nach' Q ?
  - bedingte Abbildung (sowohl bzgl. der Zuordnung der Attribute als auch bzgl. der Abbildung der Werte)
- Beispiel für (n:m)-Zusammenhang




  - i.a. komplexe Berechnungsfunktionen notwendig für die Umrechnung von Q 'nach' Z
  - (komplexe) Berechnungsfunktionen für die Umrechnung von Z 'nach' Q ?

# Entitykorrespondenz

- (n:1)-Zusammenhang zwischen Entities aus Q und Z
  - Instanzen aus Q, die einem Entity in Z entsprechen, müssen identifiziert werden können
- (1:n)-Zusammenhang zwischen Entities aus Q und Z

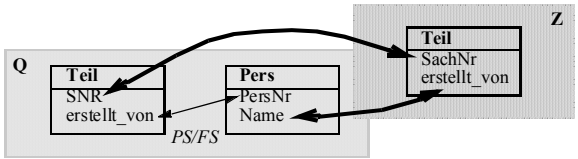



  - kontextabhängige Abbildung (Q → Z): Endpunkte können nur im entsprechenden 'Kabel'-Kontext (in Z) instanziiert werden
  - Verdrängungsabhängigkeit (Z → Q): in Z könnte für ein Kabel nur ein Endpunkt instanziiert sein; dann kann keine korrespondierende Leitungsinstanz in Q erzeugt werden
- (n:m)-Zusammenhang: alle Probleme können auftreten



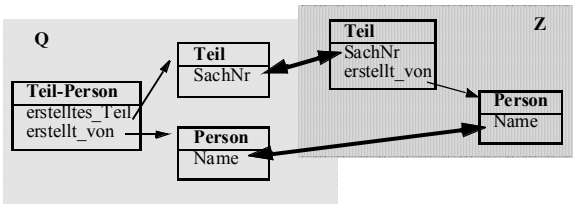
# Vernetzung von Quellkonstrukten

- abhängige Quellentities
- vgl. Entitykorrespondenz
- besondere Mechanismen der Abbildung notwendig bei (n:1)-Abbildung (Zielentity entsteht durch Join)



# Vernetzung von Zielentities

- abhängige Zielentities
- bei Abbildung vernetzter Quellstrukturen in vernetzte Zielstrukturen können die Referenzen voneinander abweichen
  - z. B. können (gerichtete) Beziehungen invertiert sein
  - Beziehungstypen zwischen Entities in Q können auf Beziehungstypen zwischen 'anderen' Entities in Z abgebildet werden

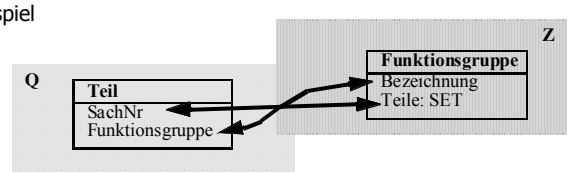



  - Anmerkung: dieses Problem tritt häufig bei Abbildungen zwischen normalisierten relationalen und objektorientierten Schemata auf



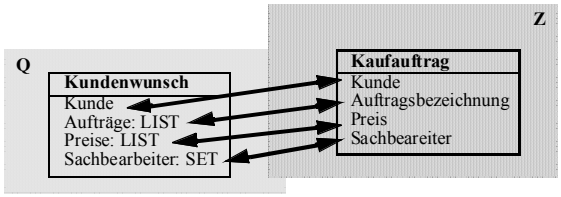
# Identitätskonflikt

- Durch Einsatz objektorientierter Datenmodelle
- ID-erhaltende Abbildung
  - nicht möglich, falls (n:m)-Verhältnis zwischen Instanzen aus Q und Z mit n < m
- ID-erzeugende Abbildung
  - Erzeugung der ID bei Erzeugung von Instanz in Z
  - Zuordnung von neuen IDs zu Instanzen in Z muss jedoch dokumentiert werden, um Propagierungen zu ermöglichen
- hybride Abbildung
  - sowohl ID-erhaltende als auch ID-erzeugende Abbildung
  - Beispiel



# Abbildung mengenwertiger Attribute

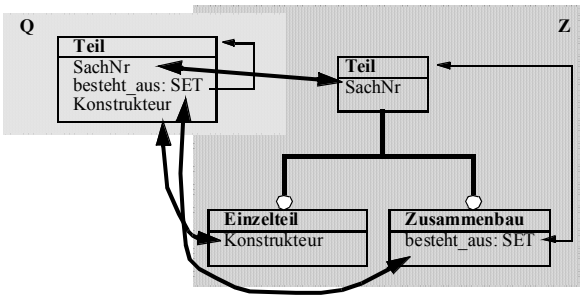
- Durch Einsatz objektorientierter Datenmodelle
- unterschiedliche Sortierung in Kollektionen erfordert die Beachtung der entsprechenden Sortierprädikate bei der Transformation
- falls geordnete Kollektion aus Q in ungeordnete Kollektion in Z transformiert wird, kann Propagierung unmöglich gemacht werden
- Abbildung einer Instanz mit einem mengenwertigen Attribut in mehrere Instanzen mit jeweils einwertigem (korrespondierendem) Attribut
- Nest-/Unnest-Operationen notwendig
- Beispiel:





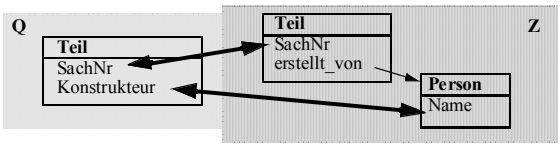

# Abbildung von Abstraktionskonzepten

- Durch Einsatz objektorientierter Datenmodelle
- Darstellung gleicher Sachverhalte auf unterschiedlichen Abstraktionsebenen
- Beispiel: Datenebene vs. Typebene



# Abbildung von Abstraktionskonzepten (2)

- Beispiel: Datentypebene vs. Objektebene






# Dynamische Aspekte und Integritätsbedingungen

- zu berücksichtigen: Seiteneffekte von Funktionen, Rückgabewerte von Methoden, Repräsentation der Zeit, unterschiedliche Programmiersprachen, ...
- Abbildung nicht automatisierbar !

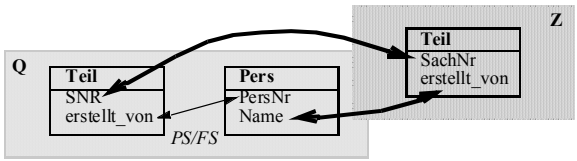
# Abbildungskardinalität

- in bisheriger Diskussion Annahme der unidirektionalen Abbildung
  - bzw. impliziter Definition der Rücktransformation
  - hier jedoch Einschränkungen wie bei View-Update
  - evtl. mit speziellen Vorkehrungen für Rücktransformation
- bidirektionale Abbildung
  - Q → Z und Z → Q
  - explizite Definition der Rücktransformation

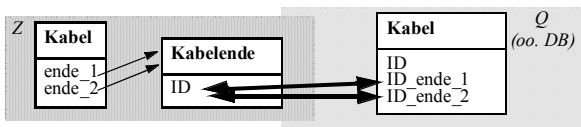


# Abbildungskardinalität (Forts.)

- bidirektionale Abbildung
  - Beispiel: mögliche Semantik bei Änderung des Attributs erstellt_von in Z




  - Beispiel: Checkin-Abhängigkeit



# Schemakardinalität

- In der bisherigen Diskussion Annahme der Zuordnung genau eines Quellschemas zu genau einem Zielschema (Schemaabbildung)
- Nun: mehrere Quellschemata zu einem Zielschema (Schemaintegration)
- Horizontale Kongruenz
  - **identische** Anwendungsbereiche
    - semantisch gleiche Information, aber unterschiedliche Darstellungsformen und unterschiedliche ID (Entity-Identifikationsproblem)
    - Problem der DB-übergreifenden Konsistenz
  - **teilweise überlappende** Anwendungsbereiche
    - nicht alle Informationen können in der integrierten Sicht dargestellt werden
    - impliziert Projektion von Elementen der Quellschemata
    - höhere Wahrscheinlichkeit der Verwendung unterschiedlicher Darstellungsformen und ID in den Qi
  - **disjunkte** Anwendungsbereiche
    - Integration wenig sinnvoll



# Horizontale Datenverteilung

- disjunkte Verteilung der Daten auf die Qi
  - es sind Prädikate zu spezifizieren, anhand derer abgeleitet werden kann, auf welche Qi neu erzeugte Instanzen aus Z zu propagieren sind
- replizierte Verteilung der Daten auf die Qi
  - es ist zu beachten, dass Daten unterschiedlich strukturiert und identifiziert werden können
- teilweise überlappende Verteilung der Daten auf die Qi
  - Vorkehrungen der beiden vorgenannten Punkte sind zu beachten
  - beim Integrationsprozess müssen Daten zusammengefügt werden

# Weitere Probleme

- Dauerhaftigkeit von Zieldaten
  - bisher Annahme der transienten Verwaltung der Zieldaten
  - persistente Datenhaltung in Z in der Regel nur zur Datenmigration
  - dabei in der Regel nur unidirektionale Abbildung und keine Propagierung
- Heterogenität von Datenmodellen
  - unterschiedliche Mächtigkeit kann zum Verlust von Information führen
- Datenstrukturierungsgrad
  - bisher Annahme der Nutzung von 'strukturierten' Quellsystemen mit 'generischen' Schnittstellen ('strukturiert' heißt: es gibt ein Schema; 'generisch' heißt: Datenzugriffsschnittstelle ist unabhängig von der Semantik der Daten)
  - bei semi-strukturierten Daten (z. B. ADTs, HTML) oder unstrukturierten Daten trifft dies nicht zu
    - hier ist keine Schemaintegration möglich
    - lediglich operationale, wrapper-basierte Abbildung



# Schemaintegrationstechniken

# Schemaintegrationstechniken

- Zusicherungsbasierte Schemaintegration
  - Zusicherungen: Inter-Schema-Korrespondenzen
    - beschreiben welche Schemabestandteile in Beziehung zueinander stehen
  - Integrationsregeln
    - bestimmen wie die korrespondierenden Bestandteile in das integrierte Schema eingebracht werden
- Integration von Klassenhierarchien mit Upward Inheritance
  - Erweiterung der zusicherungsbasierten Schemaintegration
- Generic Integration Model – GIM
  - Analyse extensionaler Beziehungen zwischen Objekttypen/-klassen der Schemata
  - Zerlegung der Klassen in sog. Basisextensionen
- Multidatenbanksprachen
  - Sichtenbildung als Integrationsmittel
- Abbildungssprachen
  - Beschreibung der Abbildung zwischen Schemata und Konfliktlösungschritte in deklarativer oder prozeduraler Form



# Zusicherungsbasierte Schemaintegration

- Grundidee
  - Beschreibung von paarweisen Beziehungen zwischen Elementen der Schemata als Inter-Schema-Korrespondenzen
  - Integrationsregeln für die zu Verfügung stehenden Korrespondenzformen bestimmen das Auftreten im integrierten Schema
  - Korrepondenzen entsprechen datenquellenübergreifende Integritätsbedingungen
    - Garantie durch das System?
- Generisches Datenmodell als Grundlage
  - Bestandteile
    - Objekttypen (und Extensionen)
    - wertbasierte Attribute
    - Links (Referenzattribute)
  - abstrahiert von spezifischen Datenmodellen
    - Besonderheiten der spezifischen Datenmodelle werden jedoch nicht vollständig abgedeckt

# Inter-Schema-Korrespondenzen

- Semantische Korrespondenz von Schemabestandteilen, weiter beschrieben durch Beziehung auf extensionaler Ebene
  - Äquivalenz, Teilmenge, Überlappung, Disjunktheit
- Korrespondenzformen
  - Element-Korrespondenz
    - Beispiel: S1.Personal $\supseteq$ S2.Projektmitarbeiter
  - Element-und-Attribut-Korrespondenz
    - Beispiel: S1.Personal.PNr = S2.Projektmitarbeiter.PersNr
  - Pfad-Korrespondenz
    - Korrespondenz von Beziehungen
    - Beipiel: S1.Personal-Abteilung $\subseteq$ S2.Projektmitarbeiter-Projekt-Abteilung

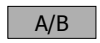


# Integrationsregeln

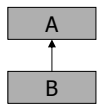
- Jeweils für Form der Korrepondenz
  - Beipiel: S1.X1 $\equiv$ S2.X2
    - S.X im integrierten Schema, Bezeichner X kann mit X1 oder X2 übereinstimmen
    - Für korrespondierende Attributpaare wird jeweils ein Attribut übernommen, Wert repräsentiert immer die Vereinigung der einzelnen Werte
    - Zusätzlich werden alle Attribute ohne Korrepondenz aus den beteiligten Schemata übernommen.
- Zusätzliche Regeln für Übernahme von Schemabestandteilen ohne Korrespondenz

# Upward Inheritance

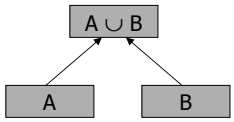
- Ausgangspunkt: Schemazusicherungen mit extensionalen Korrespondenzen
  - betrachtet ausschliesslich objektorientierte Klassenhierarchien
- Integrationsregeln
  - A = B




  - $A \supseteq B$




  - $A \cap B$ oder $A \neq B$






# Generic Integration Model – GIM

- Analyse der extensionalen Beziehungen für beteiligte Schemata führen zu einer Zerlegung von Klassen in sog. Basisextensionen
  - Mengen von Objekten, die gleichzeitig Mitglieder derselben Klassen sind
- GIM Matrix
  - Betrachtet werden nun die den Klassen zugeordneten Attribute
  - Menge von Extensionen mit einer gemeinsamen Attributmenge
    -> Klasse K
  - K1 besitzt Teilmenge der Extensionen von K2, K1 besitzt Obermenge der Attribute von K2
    -> **K1** ist Subklasse von **K2**
- Ableitung eines integrierten Schemas
  - Permutation von Zeilen und Spalten zur Bestimmung von formalen Konzepten
    - "maximale mit Kreuzen ausgefüllte Rechtecke"
  - Ableitung einer Klassenhierarchie

| | E1 | E2 | E3 | ... | En |
|---|---|---|---|---|---|
| A1 | X | X | | ... | X |
| A2 | X | X | X | ... | |
| A3 | X | | X | ... | |
| ... | | | | | |
| Am | | X | X | ... | |

# Abbildungssprache

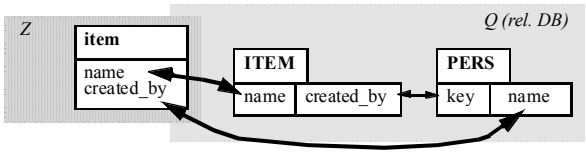
- Nachteile herkömmlicher Verfahren
  - Betrachten vor allem extensionale Konflikte, in homogenem Datenmodell
  - Unterstützen keine zufriedenstellende Auflösung struktureller Konflikte
  - Updateproblematik
- Überwindung der Heterogenität durch Entwicklung einer Abbildungssprache
  - Integration mehrerer Schemata, die möglicherweise in verschiedenen Datenmodellen erstellt sind, d. h., Abbildung von Daten zwischen heterogenen Schemastrukturen
  - Deskriptive Sprache, d. h. deklarative Abbildungsspezifikationen
  - Technologieunabhängigkeit der Abbildungsspezifikation
  - Unterstützung von sowohl Retrieval als auch Update (vergleiche: View-Update-Problematik)



# BRIITY

- Forschungsansatz einer Abbildungssprache
  - AG DBIS, FB INF, UNI KL
  - wurde entworfen, um vorgenannte Probleme zu überwinden
  - unterstützt bi-direktionale Abbildungen
  - deskriptiv
  - technologieunabhängig
  - unterstützt Benutzer-spezifizierte Update-Anweisungen
  - besonderes Anliegen: Unterstützung von objektorientierten Zielschemata, wobei Abbildung mengenorientiert und deskriptiv (d. h. wie in relationalen Systemen) beschrieben kann
  - Unterstützung von EXPRESS als Ziel-Datenmodell

# Struktur einer Abbildungsspezifikation




```
BEGIN
MAPPED_SCHEMAS
        ts := target_schema <- rel_db:= rel_db@rel_dbs@localhost;
END_MAPPED_SCHEMAS;
INCLUDE
        LIB /usr/users/sauter/libstring.a;
        INC string.h;
END_INCLUDE;
TYPE_MAPPING
        MAP ts.DM <- rel_db.US$;
                ts.DM <- 0.67 * rel_db.US$;
                rel_db.US$ <- 1.5 * ts.DM;
        END_MAP;
END_TYPE_MAPPING;
...
```



# Struktur einer Abbildungsspezifikation (Forts.)

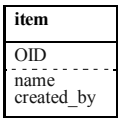
- Allgemeine Struktur einer Abbildungsspezifikation (Forts.)

```
...
ENTITY_MAPPING
    MAP item <- _item := rel_db.ITEM, _pers:= rel_db.PERS;
      ON_RETRIEVE
          name <- _item.name;
          created_by <- _pers.name;
          IDENTIFIED_BY(_item.name, _pers.key);
          WHERE (_item.created_by = _pers.key);
      ON_UPDATE ...
      ON_INSERT ...
      ON_DELETE ...
    END_MAP;
END_ENTITY_MAPPING;
END.
```

# Elementare mengenorientierte Abbildungsregeln

- Grundlegender Aufbau einer MAP-Anweisung

Ziel




Abbildung

```
MAP item <- _item := rel_db.ITEM,
            _pers:= rel_db.PERS;
  ON_RETRIEVE
     IDENTIFIED_BY (_item.name, _pers.key);
     name <- _item.name;
     created_by <- _pers.name;
     WHERE (_item.created_by = _pers.key);
```

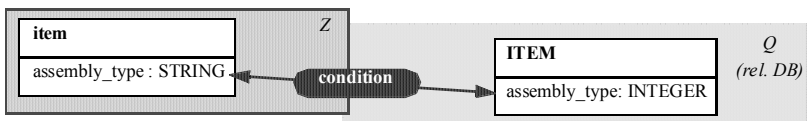
Quelle

```
SELECT _item.name, _pers.name
FROM ITEM _item, PERS _pers
WHERE _item.created_by =
      _pers.key
```



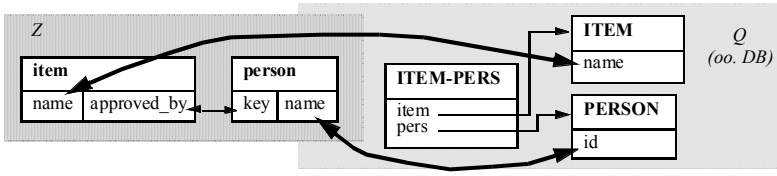
# Bedingte Abbildung

- Abbildung mehrerer Quellinstanzen auf eine Zielinstanz oder
- Konvertierung von Datentypen




```
MAP item <- _item:=rel_db.ITEM;
    ON_RETRIEVE
       assembly_type <-  IF (_item.assembly_type = 512) THEN "manufacturing"
                         ELSE IF (_item.assembly_type = 918) THEN "configurable"
                         ELSE ...;
```

# Referentielle Integrität in relationalen Zielschemata

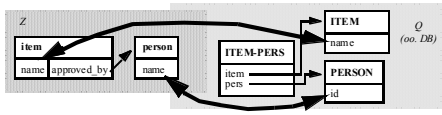



```
MAP item <- _item:=oo_db.ITEM, _ip:=oo_db.ITEM-PERS, _pers:=oo_db.PERSON;
    ON_RETRIEVE
        name <- _item.name;
        approved_by <-  CASCADED_MAP person.key
                        WITH_ID _item.INV(_ip:item).pers.id
        IDENTIFIED_BY (_item.name);
END_MAP;
```



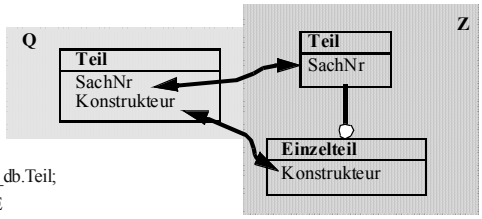
# Abbildung netzartiger Strukturen

- betrachte Beispiel auf vorangegangener Folie, wobei jedoch nun auch Z objektorientiert sein soll




```
MAP person <- oo_db.PERSON;
    ON_RETRIEVE
        name <- oo_db.PERSON.id;
        IDENTIFIED_BY (oo_db.PERSON.id);
END_MAP;
MAP item <- _item:=oo_db.ITEM, _ip:=oo_db.ITEM-PERS, _pers:=oo_db.PERSON;
    ON_RETRIEVE
        name <- _item.name;
        approved_by <-  CASCADED_MAP person
                        WITH_ID _item.INV(_ip:item).pers.id
        IDENTIFIED_BY   (_item.name);
END_MAP;
```

# Generalisierung




```
...
MAP Teil <- _t := oo_db.Teil;
    ON_RETRIEVE
        SachNr <- _t.SachNr;
        IDENTIFIED_BY (OID(_t));
END_MAP;
MAP Einzelteil SUBTYPE_OF(Teil) <- _t := oo_db.Teil;
    ON_RETRIEVE
        Konstrukteur <- _t.Konstrukteur;
        IDENTIFIED_BY (OID(_t));
END_MAP;
...
```



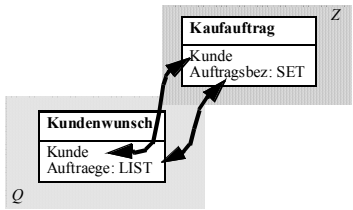
# Abbildung von Aggregaten

- dienen der Abbildung mengenwertiger Attribute sowie für NEST/UNNEST

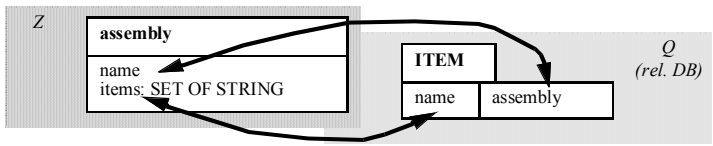
```
<set valued attribute mapping> ::=
  [NEST] (LIST|SET|ARRAY) '('<right hand side of attr mapping or nested set valued mapping>')'
  [WHERE <DNF expression>]
  [ORDER_BY <sort expression>]
  [GROUPED_BY <source attr identification>].
```

- Beispiel:

```
MAP Kaufauftrag <- _k:= oo_db.Kundenwunsch;
    ON_RETRIEVE
        Kunde <- _k.Kunde;
        Auftragsbez <- SET (_k.Auftraege * 98,5%)
                       WHERE (_k.Auftraege > 20);
        IDENTIFIED_BY (OID(_k));
END MAP
```



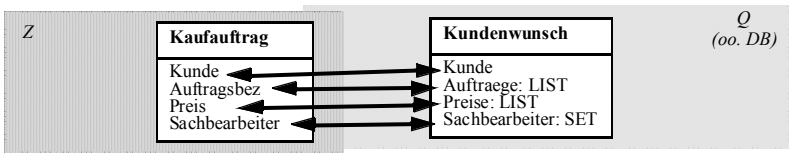
# NEST-Operation




```
MAP assembly <- _item:= rel_db.ITEM;
    ON_RETRIEVE
        name <- _item.assembly;
        items <-  NEST (_item.name)
                GROUPED_BY (_item.assembly);
```



# UNNEST-Operation




```
MAP Kaufauftrag <- _k:=oo_db.Kundenwunsch;
ON_RETRIEVE
      FOR_ALL (_unnest_auftrag := UNNEST (_k.Auftraege),
                  _unnest_preis : = UNNEST (_k.Preise))
            FOR_ALL (unnest_sachbearbeiter := UNNEST (_k.Sachbearbeiter))
                  Kunde <- _k.Kunde;
                  Auftragsbez <- _unnest_auftrag;
                  Preis <- _unnest_preis;
                  Sachbearbeiter <- _unnest_sachbearbeiter;
      IDENTIFIED_BY (OID(_k);
END_MAP;
```

# Regeln zur Propagierung von Updates

<update_clause> ::=
'ON_UPDATE' (<update_statement> {',' <update_statement>}
| 'INVERSE_TO_RETRIEVE' ';').

<update_statement> ::=
<target_attribute_id> 'OF' <update_statement_body_list>
| <update_statement_for_set_valued_target_attributes>.

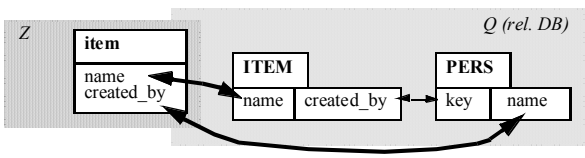
<update_statement_body> ::=
('NEW'|'MODIFIED'|'DELETED') ':' ('RESTRICTED'|'INVERSE_TO_RETRIEVE'|<assign_stmnt_list>)';'.

<assign_statement> ::=
'ASSIGN' <conjunction_of_assgn_statement_expr>
['WHERE' <where_clause_containing_bool_expr_of_assgn_statement>].

<assign_statement_expr> ::=
['NOT_']'IS_INSTANCE''('<source_entity_id>[<with_instance_id>]['':'<attr_value_expr_list>]')'
| <is_element_expr>
| <has_value_expr>.

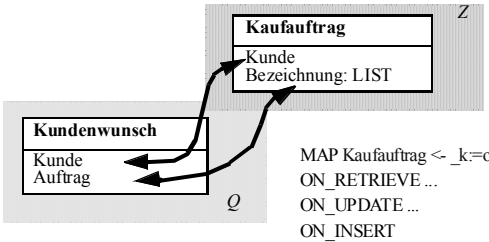


# Update-Operation




ON_UPDATE created_by OF
NEW: ASSIGN ( IS_INSTANCE(_item: name = ts.item.name, created_by = ?1) AND
IS_INSTANCE(_pers: key = ?1, name = ts.item.created_by));
MODIFIED: ASSIGN ( IS_INSTANCE(_pers: name = ts.item.created_by));
ASSIGN ( IS_INSTANCE(_item: name = ts.item.name, created_by = ?1))
WHERE ( IS_INSTANCE(_pers: key = ?1, name = ts.item.created_by));
DELETED: ASSIGN ( NOT_IS_INSTANCE(_pers: key = ?1))
WHERE ...;

# Update-Regeln für mengenorientierte Ziel-Attribute

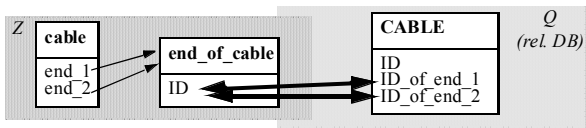



```
MAP Kaufauftrag <- _k:=oo_db.Kundenwunsch;
ON_RETRIEVE ...
ON_UPDATE ...
ON_INSERT
        FOR EACH_ELEMENT_VALUE
                        _b OF Kaufauftrag.Bezeichnung DO
            ASSIGN
                IS_INSTANCE (_k: Kunde = Kaufauftrag.Kunde,
                                Auftrag = (_b / 98.5%));
        END_FOR;
END_MAP;
```



# Multiple Instanzierung

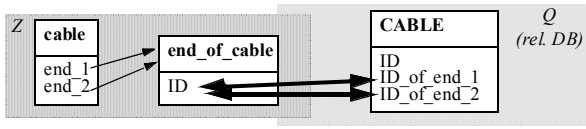
- Partitionen




```
MAP end_of_cable <-     PARTITION _part_end1: _c:=rel_db.CABLE,
                        PARTITION _part_end2: _c:=rel_db.CABLE;
PARTITION _part_end1:
        ON_RETRIEVE
            ID <- _c.ID_of_end_1;
            IDENTIFIED_BY (_c.ID);
PARTITION _part_end2:
        ON_RETRIEVE
            ID <- _c.ID_of_end_2;
            IDENTIFIED_BY (_c.ID);
END_MAP;
...
```

# Multiple Instanzierung (Forts.)

- Partitionen (Forts.)



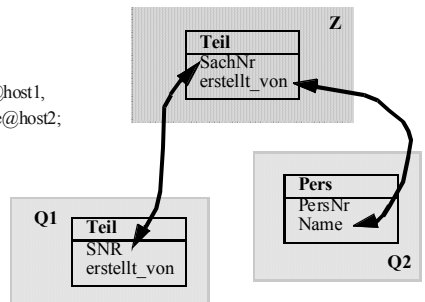

```
...
MAP cable <- _c:=rel_db.CABLE;
     ON_RETRIEVE
        end_1 <-  CASCADED_MAP end_of_cable
                  PARTITION _part_end1
                  WITH_ID (_c.ID, _c.ID_of_end_1);
        end_2 <-  CASCADED_MAP end_of_cable
                  PARTITION _part_end2
                  WITH_ID (_c.ID, _c.ID_of_end_2);
        IDENTIFIED_BY (_c.ID);
END_MAP;
```



# Integration mehrerer Quell-Schemata

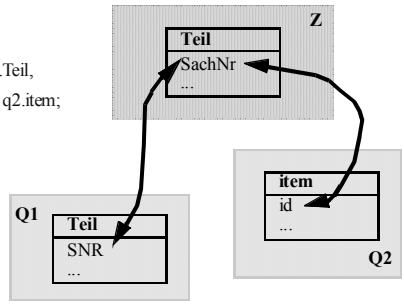
- Überlappende Anwendungsbereiche → DB-übergreifende Verbundoperationen

```
BEGIN
MAPPED_SCHEMAS
     z := integriert <-  q1 = teile_db@db2@host1,
                         q2 = pers_db@oracle@host2;
END_MAPPED_SCHEMAS;
ENTITY_MAPPING
   MAP Teil <- _t := q1.Teil, _p:= q2.Pers;
     ON_RETRIEVE
        SachNr <- _t.SNR;
        erstellt_von <- _p.Name;
        ...
        WHERE (_t.erstellt_von = _p.PersNr);
     ...
END_ENTITY_MAPPING;
END.
```



# Identische Anwendungsbereiche → kontextabhängige Abbildung

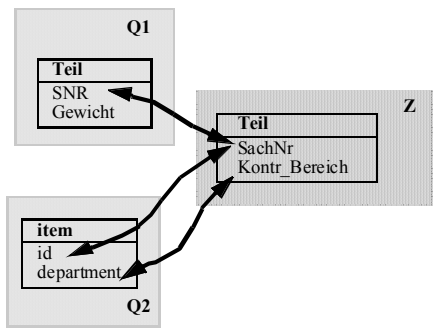
```
MAP Teil <-    PARTITION  db_q1: _t := q1.Teil,
                          PARTITION  db_q2: _i := q2.item;
    PARTITION db_q1
      ON_RETRIEVE
         SachNr <- _t.SNR;
         IDENTIFIED_BY(_t.SNR);
    ...
    PARTITION db_q2
      ON_RETRIEVE
         SachNr <- _i.id;
         IDENTIFIED_BY(_i.id);
END_ENTITY_MAPPING;
  ...
END.
```





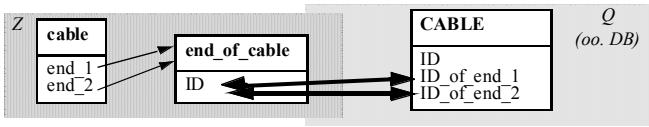

# Globale Identität

```
MAP Teil <-    PARTITION   db_q1: _t := q1.Teil,
                           PARTITION   db_q2: _i := q2.item;
    PARTITION db_q1
      ON_RETRIEVE
         SachNr <- _t.SNR;
         Konstr_Bereich <- "EntwicklungPkw"
         IDENTIFIED_BY( _t.SNR);
         GLOBAL_IDENTITY (_t.SNR,
               "EntwicklungPkw");
    ...
    PARTITION db_q2
      ON_RETRIEVE
         SachNr <- _i.id;
         Konstr_Bereich <- _i.department
         IDENTIFIED_BY(_i.id, _i.department);
END_ENTITY_MAPPING;
  ...
END.
```



# Zusätzliche Integritätsbedingungen

- IBs für Konflikte auf Instanzenebene: ECA-Regeln
- Checkin-Abhängigkeiten

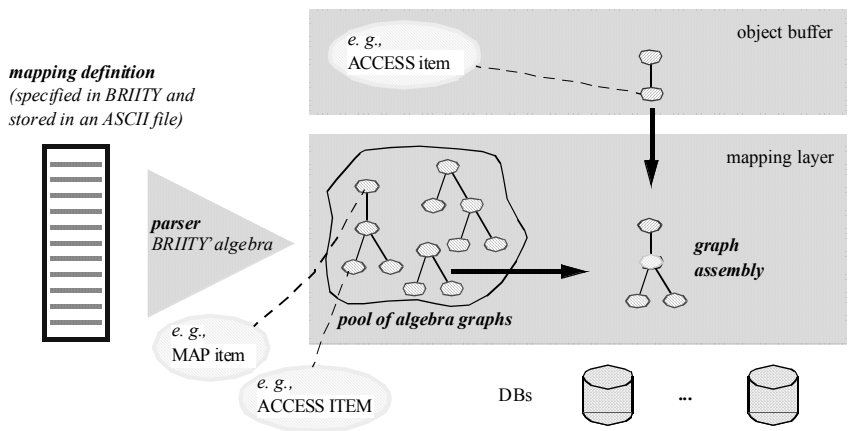



```
INTEGRITY_CONSTRAINTS
    DEPENDENCIES
        GROUP    cable, end_of_cable
        WHERE    (cable.end_1 = OID(end_of_cable)) AND (cable.end_2 = OID(end_of_cable))
    END_DEPENDENCIES;
END_INTEGRITY_CONSTRAINTS;
```

- Initialisierung von Primärschlüsselattributen in Quellschemata



# Ausführungsmodell



# Ausführungsmodell (Forts.)

- Parsen der Abbildungsspezifikation
- Anfragetransformation
  - Transformation in Abbildungsgraph (Algebra-Graph)
  - Blattknoten: Operationen zum Zugriff auf die DBSs
  - innere Knoten: Algebraoperatoren
  - Wurzelknoten: Operator zum Erzeugen von Zielinstanzen
- Auswahl der relevanten Abbildungsgraphen
  - für die Bearbeitung einer Query sind die Abbildungsgraphen relevant, die für die Erzeugung der in der Query angesprochenen Objekttypen des Zielschemas verantwortlich sind
  - Abbildungsgraph wird anhand des Wurzelknotens selektiert
- Assemblierung
  - Query-Graph und Abbildungsgraph werden zusammengefügt
  - Wurzel-Operator des Abbildungsgraphen und Zugriffsoperator des Query-Graphen werden verschmolzen
- Optimierung und Ausführung des assemblierten Graphen durch Mapping-Layer



# Zusammenfassung

- Schemaabbildung und -integration
  - Zielvorstellung
    - Bereitstellung eines integrierten Schemas
    - Bearbeitung der Zieldaten mit generischen Operatoren
    - Unterstützung eines breiten Spektrums von Quellsystemen und Auswahlmöglichkeit hinsichtlich der Nutzung eines Zielsystems
  - Schemaintegrationstechniken
  - Abbildungssprache BRIITY
    - löst die wesentlichen Probleme struktureller Heterogenität
    - unterstützt sowohl relationale als auch objektorientierte Modelle auf beiden Seiten (Quellsysteme, Zielsystem)
    - erlaubt bi-direktionale Abbildung und unterstützt so beliebige Update-Operationen auf den Zieldaten
    - hoher Spezifikationsaufwand