Kapitel 5

Schemaabbildung und Anfragetransformation[1]

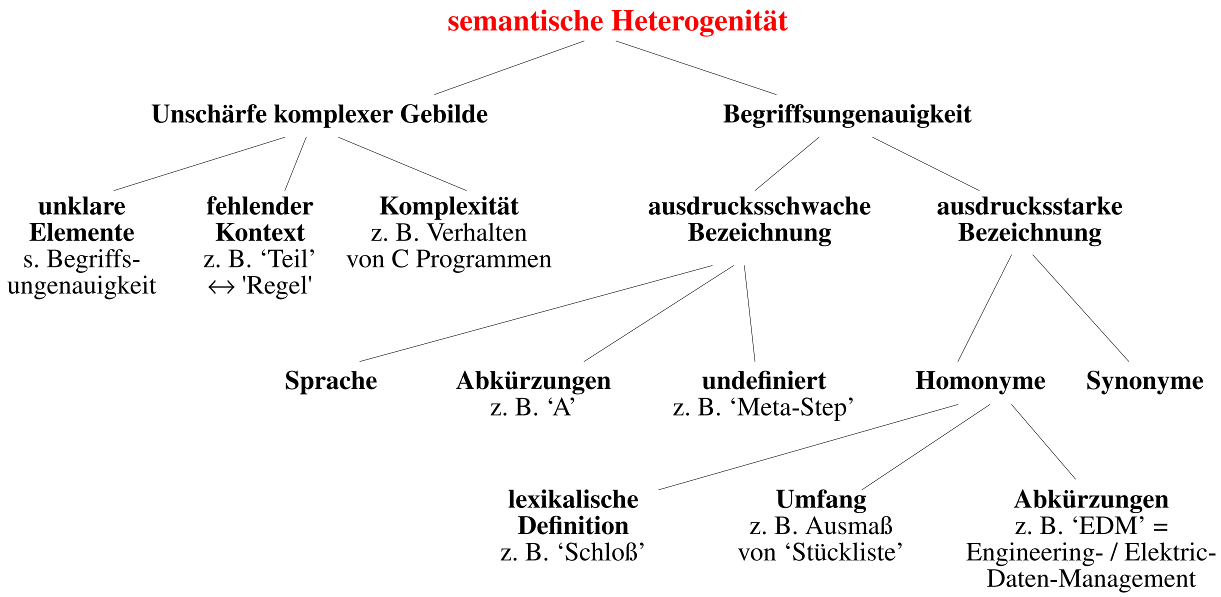
Inhalt

- ❑ Formen der Heterogenität (semantische vs strukturelle Heterogenität)
- ❑ Probleme bei der Überwindung struktureller Heterogenität
- ❑ Die Abbildungssprache BRIITY
- ❑ Überblick über das Ausführungsmodell von BRIITY
- ❑ Vergleich mit IBM-DataJoiner
- ❑ Zusammenfassung

1. Härder, T., Sauter, G., Thomas, J.: The Intrinsic Problems of Heterogeneity and an Approach to their Solution. 1999.

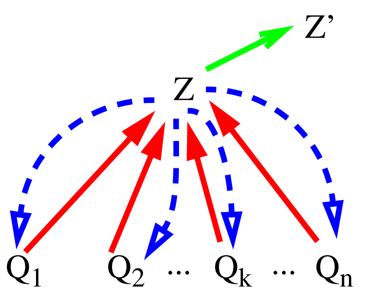
Formen der Heterogenität (1)

- ❑ Semantische Heterogenität
  - ➪ bezieht sich auf Modellierungsinhalte



Formen der Heterogenität (2)

- Strukturelle Heterogenität
  - bezieht sich auf die jeweils zur Verfügung stehenden Modellierungskonstrukte
  - Szenario

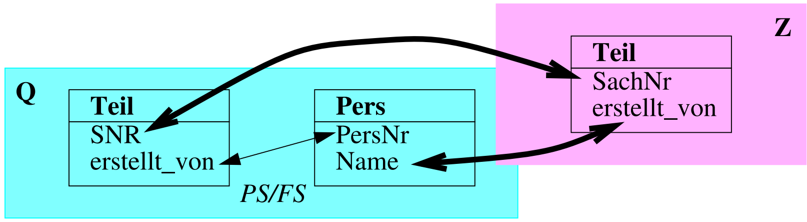


Q1, ... Qn	Quell-DBS, Legacy oder Anwendungssysteme
Z	Integrierte/heterogene Sicht (Zielschema bzw. -system)
Q1, ... Qn → Z	Schema-Integration (Datentransfer nach Anfrage)
Z → Q1, ...Qn	Propagierung von Änderungen
Z → Z'	Migration

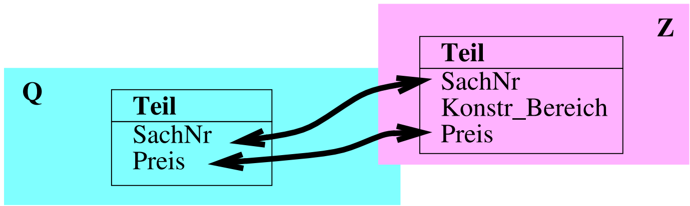
  - Überwindung der Heterogenität durch Entwicklung einer Abbildungssprache
    - Integration mehrerer Schemata, die möglicherweise in verschiedenen Datenmodellen erstellt sind, d. h., Abbildung von Daten zwischen heterogenen Schemastrukturen
    - Deskriptive Sprache, d. h. deklarative Abbildungsspezifikationen
    - Technologieunabhängigkeit der Abbildungsspezifikation
    - Unterstützung von sowohl Retrieval als auch Update (vergleiche: View-Update-Problematik)

Probleme der strukturellen Heterogenität (1)

- Ausgangsszenario
  - homogene, relationale Schemata
  - transiente Verwaltung der Zieldaten
  - unidirektionale (1:1)-Beziehung: Q → Z
- Elementare Abbildungsprobleme
  - Vertikale semantische Kongruenz
    - Projektion: Z beschreibt kleineren Ausschnitt des Anwendungsbereiches als Q
      - View-Update-Problematik
      - Beispiel



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

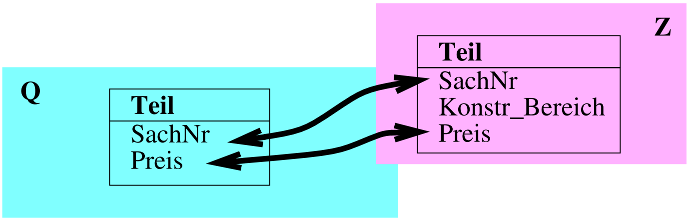


Probleme der strukturellen Heterogenität (2)

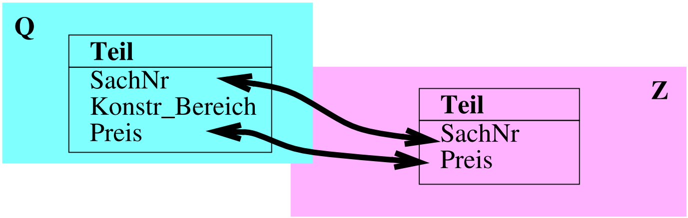
❑ Elementare Abbildungsprobleme (Forts.)

➩ 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”

Probleme der strukturellen Heterogenität (3)

❑ Elementare Abbildungsprobleme (Forts.)

➩ Datentypkorrespondenz

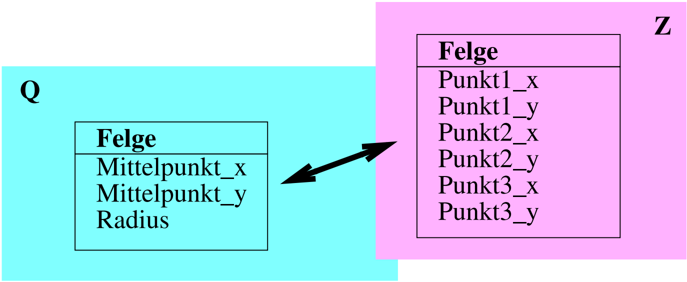
- Unterstützung unterschiedlicher Datentypen durch Q und Z
- Problem: finde Abbildung, die semantikerhaltende Rücktransformation erlaubt
  - Beispiel: REAL (Q) → INTEGER (Z) durch Rundung (genaue Rücktransformation nicht möglich)
  - generell: korrekte und vollständige Transformation von Daten eines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

Probleme der strukturellen Heterogenität (4)

❑ Elementare Abbildungsprobleme (Forts.)

➪ 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



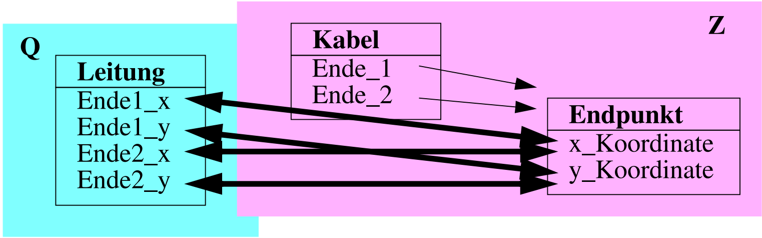
- i. a. komplexe Berechnungsfunktionen notwendig für die Umrechnung von Q 'nach' Z
- (komplexe) Berechnungsfunktionen für die Umrechnung von Z 'nach' Q ?

Probleme der strukturellen Heterogenität (5)

❑ Elementare Abbildungsprobleme (Forts.)

➪ Entitykorrespondenz

- (n:1)-Zusammenhang zwischen Entities (Objekttypen) 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



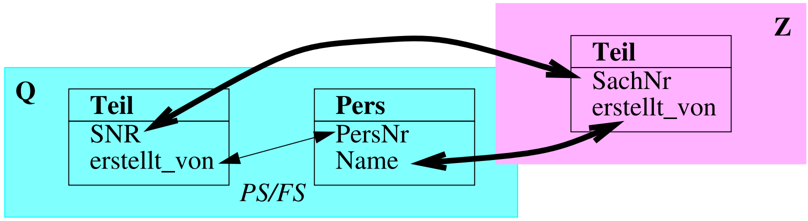
  - kontextabhängige Abbildung (Q → Z):
    Endpunkte können nur im entsprechenden 'Kabel'-Kontext (in Z) instanziert werden
  - Verdrängungsabhängigkeit (Z → Q):
    in Z könnte für ein Kabel nur ein Endpunkt instanziert sein; dann kann keine korrespondierende Leitungsinstanz in Q erzeugt werden
- (n:m)-Zusammenhang: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

Probleme der strukturellen Heterogenität (6)

❑ Elementare Abbildungsprobleme (Forts.)

➪ Vernetzung von Quellkonstrukten

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

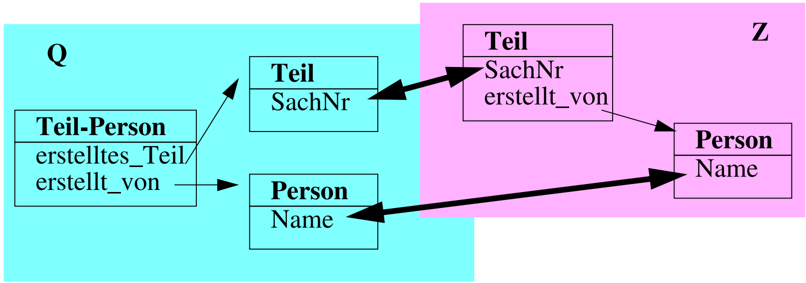


Probleme der strukturellen Heterogenität (7)

❑ Elementare Abbildungsprobleme (Forts.)

➪ Vernetzung von Zielentities

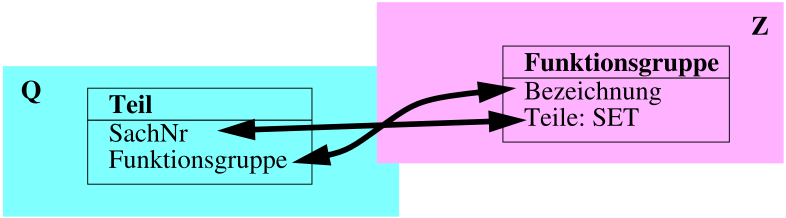
- abhängige Zielentities
- bei Abbildung vernetzter Quellstrukturen in vernetzte Zielstrukturen können die Referenzen voneinander abweichen
  - z. B. können (gerichtete) Beziehungen invertiert sein
  - Beziehungstypen zwischen Entities in Q können auf Beziehungstypen zwischen ‘anderen’ Entities in Z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



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

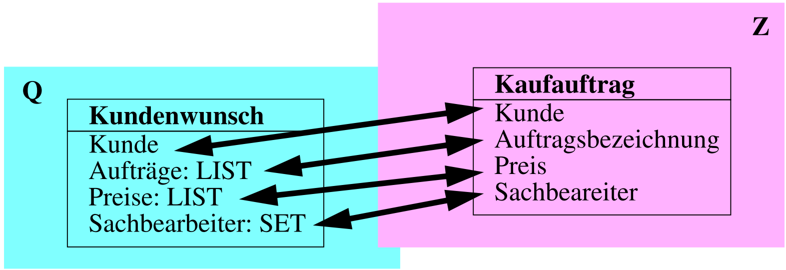
Probleme der strukturellen Heterogenität (8)

- Konflikte durch Einsatz objektorientierter Datenmodelle
  - Identität
    - 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



Probleme der strukturellen Heterogenität (9)

- Konflikte durch Einsatz objektorientierter Datenmodelle (Forts.)
  - Abbildung mengenwertiger Attribute
    - 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

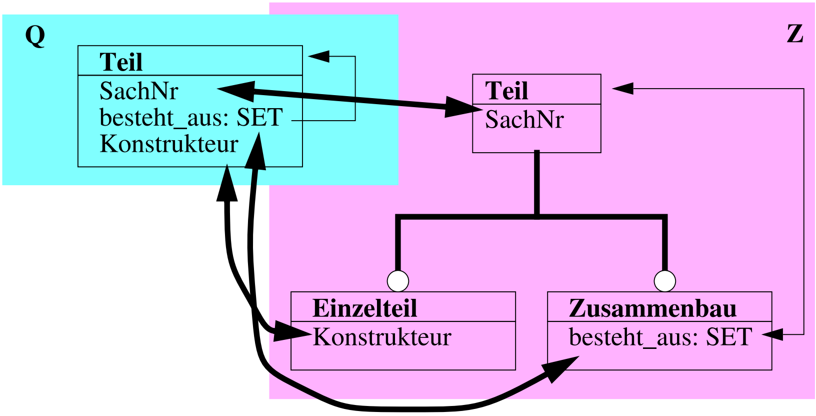


Probleme der strukturellen Heterogenität (10)

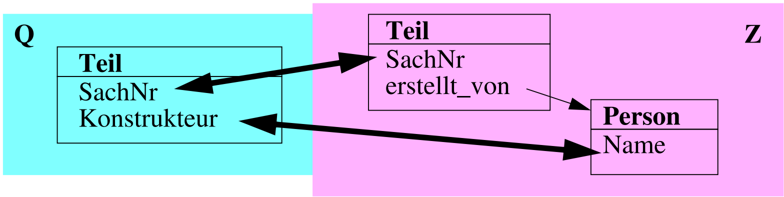
❑ Konflikte durch Einsatz
objektorientierter Datenmodelle (Forts.)

➪ Abbildung von Abstraktionskonzepten

- Darstellung gleicher Sachverhalte auf unterschiedlichen Abstraktionsebenen
- Beispiel: Datenebene vs. Typebene



- Beispiel: Datentypebene vs. Objekttypebene



Probleme der strukturellen Heterogenität (11)

❑ Konflikte durch Einsatz
objektorientierter Datenmodelle (Forts.)

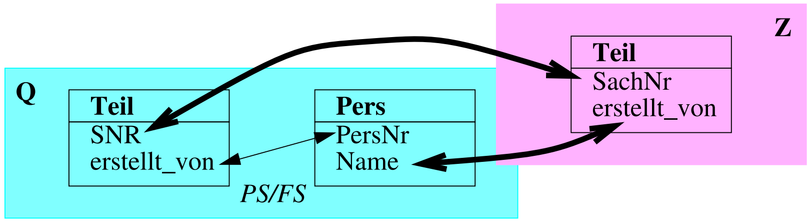
➪ Dynamische Aspekte und Integritätsbedingungen

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

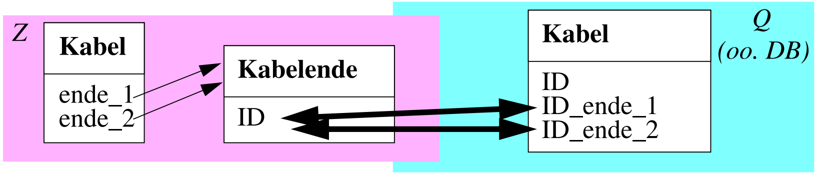
Probleme der strukturellen Heterogenität (12)

❑ 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
  - Beispiel: Mögliche Semantik bei Änderung des Attributs *erstellt_von* in Z



  - Beispiel: Checkin-Abhängigkeit



Probleme der strukturellen Heterogenität (13)

❑ Schemakardinalität

- in der bisherigen Diskussion Annahme der Zuordnung genau eines Quellschemas zu genau einem Zielschema (Schemaabbildung);
  nun: mehrere Quellschemata zu einem Zielschema (Schemaintegration)
- Horizontale semantische 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 Q_i
  - *disjunkte* Anwendungsbereiche
    - Integration wenig sinnvoll

Probleme der strukturellen Heterogenität (14)

❑ Schemakardinalität (Forts)

➭ Horizontale Datenverteilung

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

Probleme der strukturellen Heterogenität (15)

❑ 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

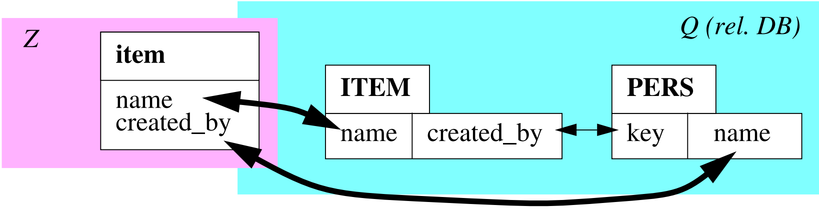
- bisher Annahme der Nutzung von ‘strukturierten’ Quellsystemen mit ‘generischen’ Schnittstellen
  (‘strukturiert’ heißt: es gibt ein Schema, ‘generisch’ heißt: Datenzugriffsschnittstelle ist unabhängig von der Semantik der Daten)
- bei semi-strukturierten Daten (z. B. ADTs, HTML) oder unstrukturierten Daten trifft dies nicht zu
  - hier ist keine Schemaintegration möglich
  - lediglich operationale, wrapper-basierte Abbildung

BRIITY (1)

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

BRIITY (2)

- ❏ Allgemeine 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;
  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.
```

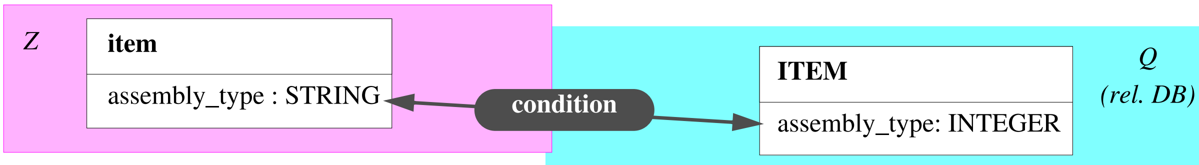
BRIITY (3)

- Elementare, mengenorientierte Abbildungsregeln
  - Grundlegender Aufbau einer MAP-Anweisung

item OID name created_by	**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);	**SELECT _item.name, _pers.name** **FROM ITEM _item, PERS _pers** **WHERE _item.created_by = _pers.key**

BRIITY (4)

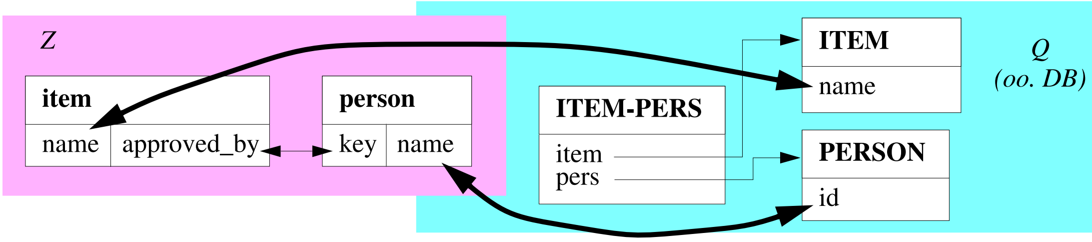
- Elementare, mengenorientierte Abbildungsregeln (Forts.)
  - Bedingte Abbildung
    - Abbildung mehrerer Quellinstanzen auf eine Zielinstanz oder
    - Konvertierung von Datentypen




```
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 ...;
```

BRIITY (5)

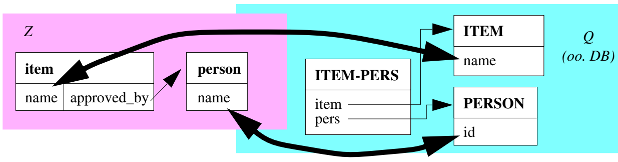
- ❑ Komplexe Abbildungsregeln
  - ➯ Referentielle Integrität in relationalen Zielschemata




```
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;
```


BRIITY (6)

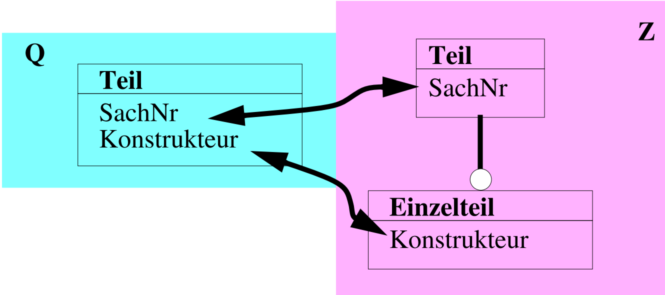
- ❑ Komplexe Abbildungsregeln (Forts.)
  - ➯ Abbildung netzartiger Strukturen
    - betrachte Beispiel auf Folie 5-23, wobei jedoch nun auch Z objektorientiert sein soll




```
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;
```

BRIITY (7)

- Komplexe Abbildungsregeln (Forts.)
  - Generalisierung




```
1:  ...
1:  MAP Teil <- _t := oo_db.Teil;
2:      ON_RETRIEVE
3:          SachNr <- _t.SachNr;
4:          IDENTIFIED_BY (OID(_t));
5:  END_MAP;
1:  MAP Einzelteil SUBTYPE_OF(Teil) <- _t := oo_db.Teil;
2:      ON_RETRIEVE
3:          Konstrukteur <- _t.Konstrukteur;
4:          IDENTIFIED_BY (OID(_t));
5:  END_MAP;
6:  ...
```

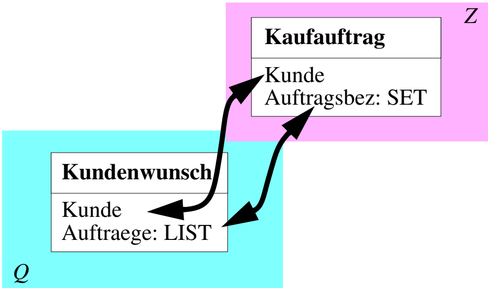

BRIITY (8)

- Komplexe Abbildungsregeln (Forts.)
  - 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>].
```

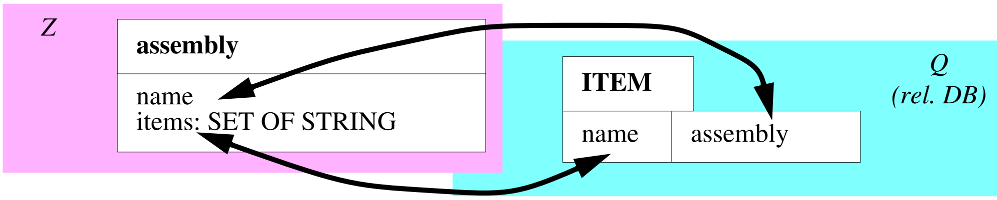
- Beispiel

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



BRIITY (9)

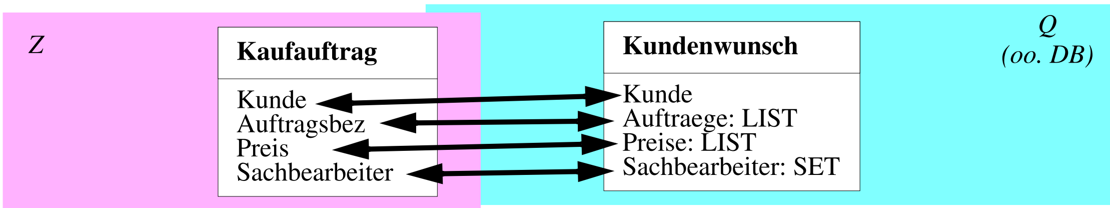
- Komplexe Abbildungsregeln (Forts.)
  - NEST-Operation




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


BRIITY (10)

- Komplexe Abbildungsregeln (Forts.)
  - 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;
```

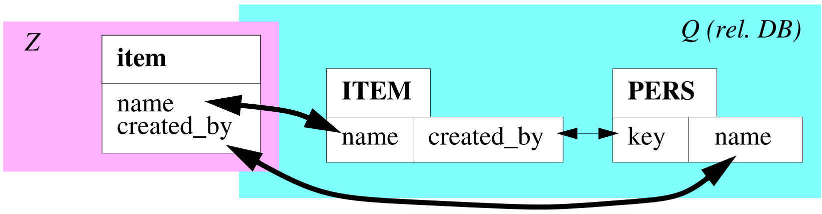
BRIITY (11)

- 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>.
```


BRIITY (12)

- Regeln zur Propagierung von Updates (Forts.)
  - Update-Operationen

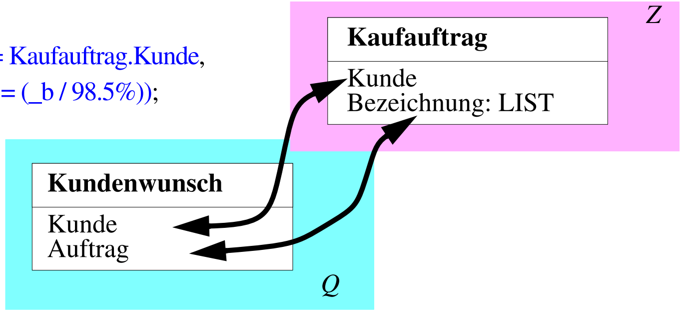



```
ON_UPDATE created_by OF
  NEW:       ASSIGN ( IS_INSTANCE(_item: name = ts.item.name, created_by = ?1) AND
                      IS_INSTANCE(_pers: key = ?1, name = ts.item.created_by));
  MODIFIED:  ASSIGN ( IS_INSTANCE(_pers: name = ts.item.created_by));
             ASSIGN ( IS_INSTANCE(_item: 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 ( HAS_VALUE(?1 = ASSIGNED_ID_VALUE(2)));
```

BRIITY (13)

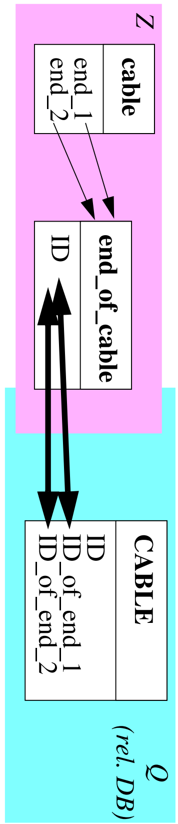
- Regeln zur Propagierung von Updates (Forts.)
  - Update-Regeln für mengenorientierte Ziel-Attribute

```
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;
```



BRIITY (14)

- Multiple Instanziierung
  - Partitionen

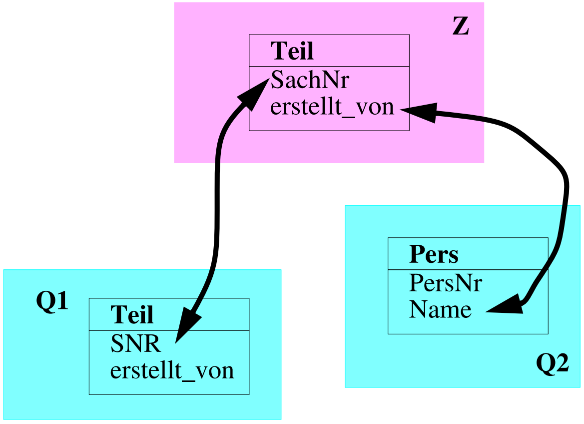



```
MAP end_of_cable <-  PARTITION _part_end1: _c:=rel_db.CABLE,
                     PARTITION _part_end2: _c:=rel_db.CABLE;
  PARTITION _part_end1:
     ON_RETRIEVE
        ID <- _c.ID_of_end_1;
        IDENTIFIED_BY (_c.ID, _c.ID_of_end_1);
  PARTITION _part_end2:
     ON_RETRIEVE
        ID <- _c.ID_of_end_2;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

BRIITY (15)

- Integration mehrerer Quellschemata
  - Überlappende Anwendungsbereiche → DB-übergreifende Verbundoperationen

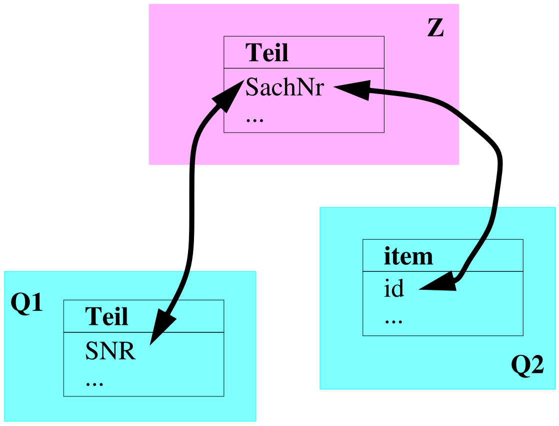
```
BEGIN
  MAPPED_SCHEMAS
      z := integriert <-  q1 = teile_db@db2@host1,
                          q2 = pers_db@oracle@host2;
  END_MAPPED_SCHEMAS;
  ENTITY_MAPPING
    MAP Teil <-_t := q1.Teil, _p:= q2.Pers;
      ON_RETRIEVE
          SachNr <- _t.SNR;
          erstellt_von <- _p.Name;
     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



BRIITY (16)

- Integration mehrerer Quellschemata (Forts.)
  - Identische Anwendungsbereiche → kontextabhängige Abbildung

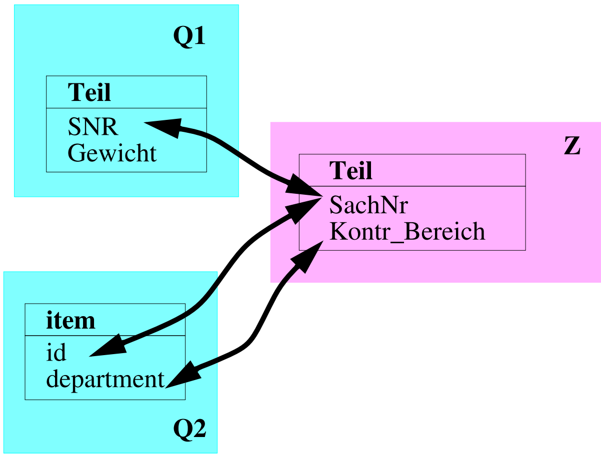
```
MAP Teil <-  PARTITION  db_q1: _t := q1.Teil,
             PARTITION  db_q2: _i := q2.item;
    PARTITION db_q1
      ON_RETRIEVE
          SachNr <- _t.SNR;
          IDENTIFIED_BY(_t.SNR);
    ...
    PARTITION db_q2
      ON_RETRIEVE
          SachNr <- _i.id;
          IDENTIFIED_BY(_i.id);
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



BRIITY (17)

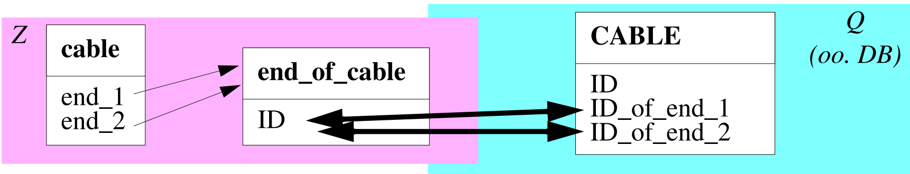
- Integration mehrerer Quellschemata (Forts.)
  - Globale Identität

```
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.
```



BRIITY (18)

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

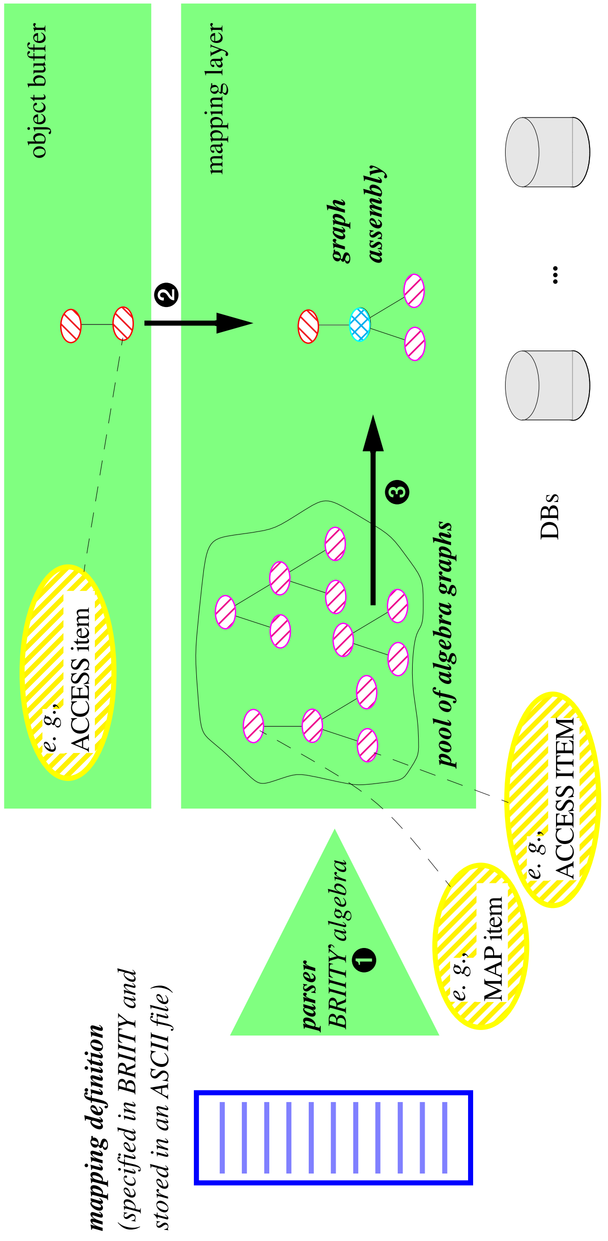



```
INTEGRITY_CONSTRAINTS
    DEPENDENCIES
        GROUP    cable, end_of_cable
        WHERE    (cable.end_1 = OID(end_of_cable)) AND (cable.end_2 = OID(end_of_cable))
    END_DEPENDENCIES;
END_INTEGRITY_CONSTRAINTS;
```

  - Initialisierung von Primärschlüsselattributen in Quellschemata

BRIITY (19)

❑ Überblick über das Ausführungsmodell



BRIITY (20)

❑ Überblick über das Ausführungsmodell (Forts.)

❶ Parsen der Abbildungsspezifikation

- Transformation in Abbildungsgraph (Algebra-Graph)
- Blattknoten: Operationen zum Zugriff auf die DBSs
- innere Knorten: Algebraoperatoren
- Wurzelknoten: Operator zum Erzeugen von Zielinstanzen

❷ Anfragetransformation

- Transformation der Anfrage in Query-Graph (Algebra-Graph)

❸ 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

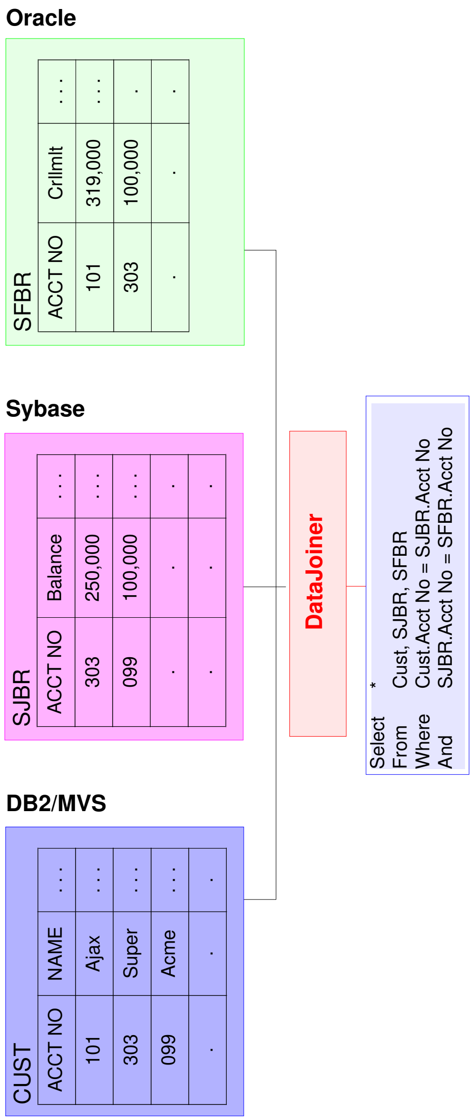
❹ Assemblierung

- 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

Vergleich mit IBM-DataJoiner (1)

- DataJoiner: ebenfalls Ansatz der Schemaabbildung und Anfragetransformation
- Beispiel einer Sichtenbildung in DataJoiner:



Vergleich mit IBM-DataJoiner (2)

- DataJoiner: Eigenschaften
  - stellt Funktionalität von IBM/DB2 in der Middleware zur Verfügung
  - unterstützt integrierten Zugriff auf
    - DB2-Datenbanken auf unterschiedlichen Plattformen
    - ORACLE- SYBASE-, IMS-Datenbanken
    - VSAM-Dateien
  - keine persistente Speicherung in DataJoiner
  - 1:1-Beziehung zwischen DB-Relationen und virtuellen Relationen in DataJoiner
    → alle SQL-Operationen sind auf den virtuellen Relationen möglich (keine bei Sichten üblichen Einschränkungen)
    → auf den virtuellen Relationen können wiederum Sichten definiert werden
- DataJoiner im Vergleich mit BRIITY
  - unterstützt nur relationale Zielschemata
  - sehr begrenztes Spektrum an möglichen Quellsystemen
  - keine tatsächliche Integration aufgrund der angesprochenen 1:1-Beziehung zwischen DB-Relationen und virtuellen Relationen in DataJoiner
  - Updates können nur unterstützt werden, wenn sie sich auf eine Datenquelle beziehen

Zusammenfassung

- Schemaabbildung und Anfragetransformation
  - Zielvorstellung
    - Bereitstellung eines integrierten Schemas
    - Bearbeitung der Zieldaten mit generischen Operatoren
    - Unterstützung eines breiten Spektrums von Quellsystemen und Auswahlmöglichkeit hinsichtlich der Nutzung eines Zielsystems
  - BRIITY
    - löst die wesentlichen Probleme der Überwindung 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
  - DataJoiner
    - Zielfunktionalität entspricht der von IBM/DB2
    - geringerer Integrationsgrad
    - geringere Flexibilität