Kapitel 5

Schemaabbildung und Anfragetransformation[1]

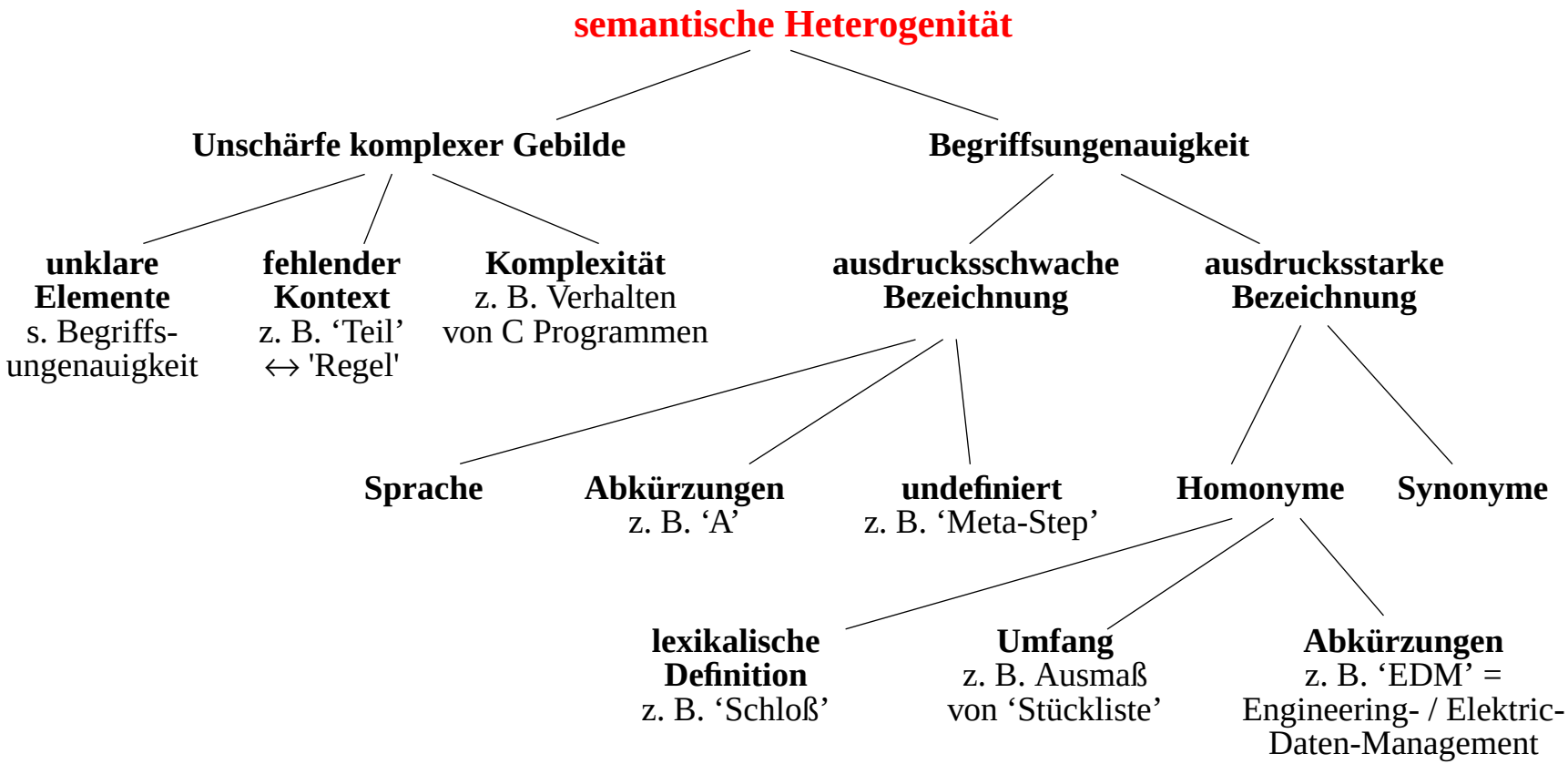
Inhalt

- 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.

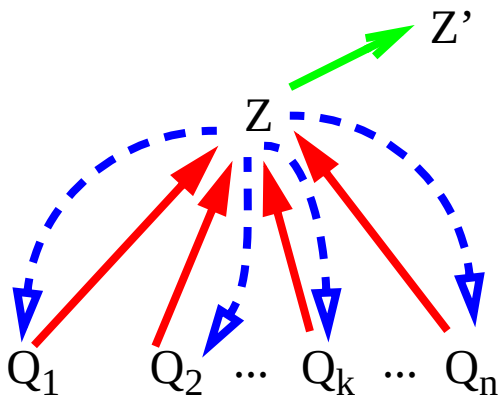
Formen der Heterogenität (1)

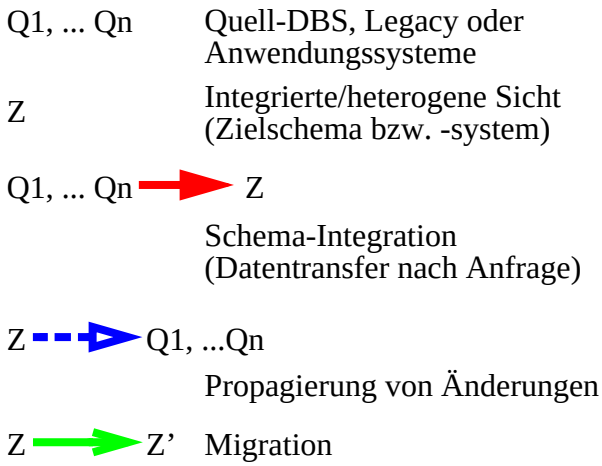
- Semantische Heterogenität
  - bezieht sich auf Modellierungsinhalte



Formen der Heterogenität (2)

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

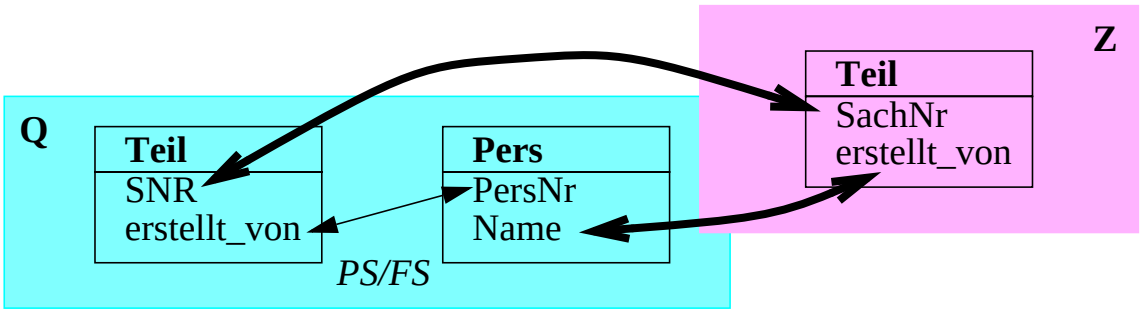




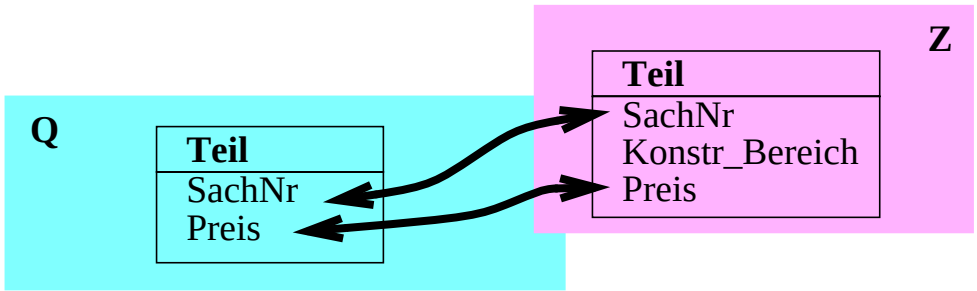
  - Überwindung der Heterogenität durch Entwicklung einer Abbildungssprache
    - Integration mehrerer Schemata, die möglicherweise in verschiedenen Datenmodellen erstellt sind, d. h., Abbildung von Daten zwischen heterogenen Schemastrukturen
    - Deskriptive Sprache, d. h. deklarative Abbildungsspezifikationen
    - 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

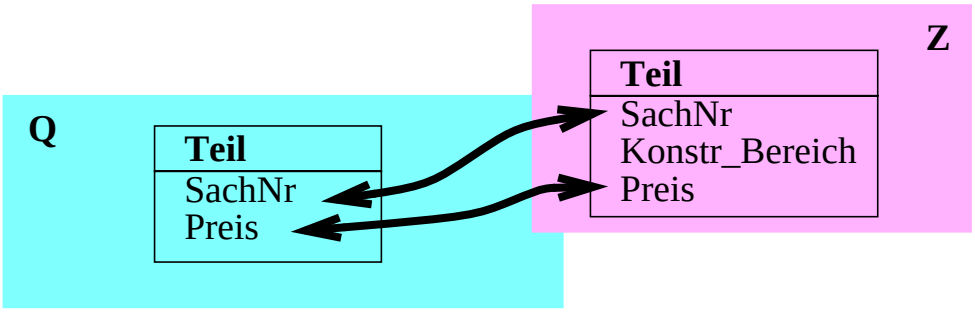


Probleme der strukturellen Heterogenität (2)

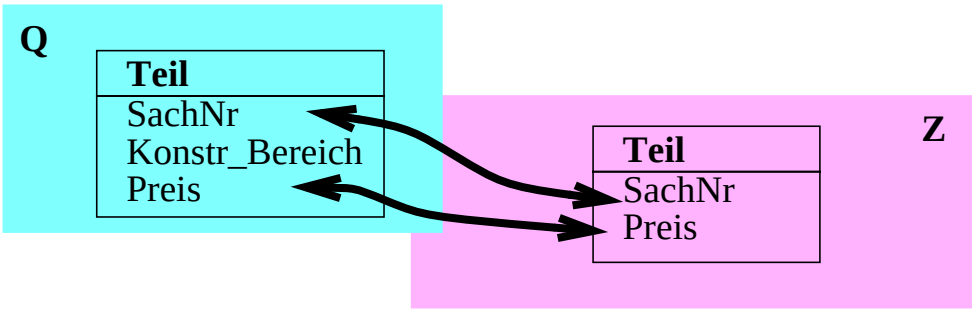
❑ Elementare Abbildungsprobleme (Forts.)

➪ 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



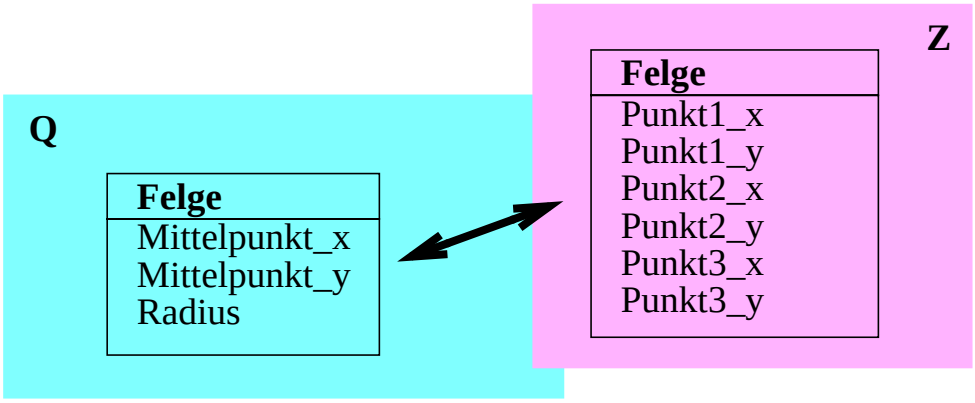
Prädikat: “nur Konstruktionsbereiche ‘A’ und ‘B’ aus Q”

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



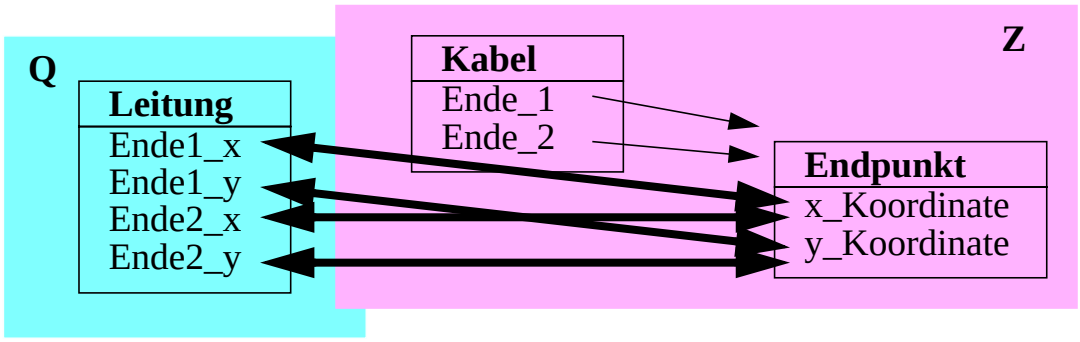
- i. a. komplexe Berechnungsfunktionen notwendig für die Umrechnung von Q ‘nach’ Z
- (komplexe) Berechnungsfunktionen für die Umrechnung von Z ‘nach’ Q ?

Probleme der strukturellen Heterogenität (5)

❑ Elementare Abbildungsprobleme (Forts.)

➪ Entitykorrespondenz

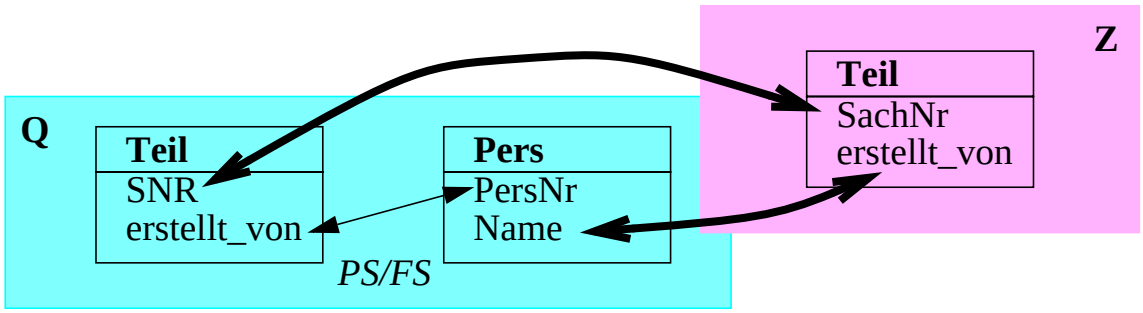
- (n:1)-Zusammenhang zwischen Entities (Objekttypen) aus Q und Z
  - Instanzen aus Q, die einem Entity in Z entsprechen, müssen identifiziert werden können
- (1:n)-Zusammenhang zwischen Entities aus Q und Z



  - kontextabhängige Abbildung (Q → Z):
    Endpunkte können nur im entsprechenden ‘Kabel’-Kontext (in Z) 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 genannten Probleme können auftreten

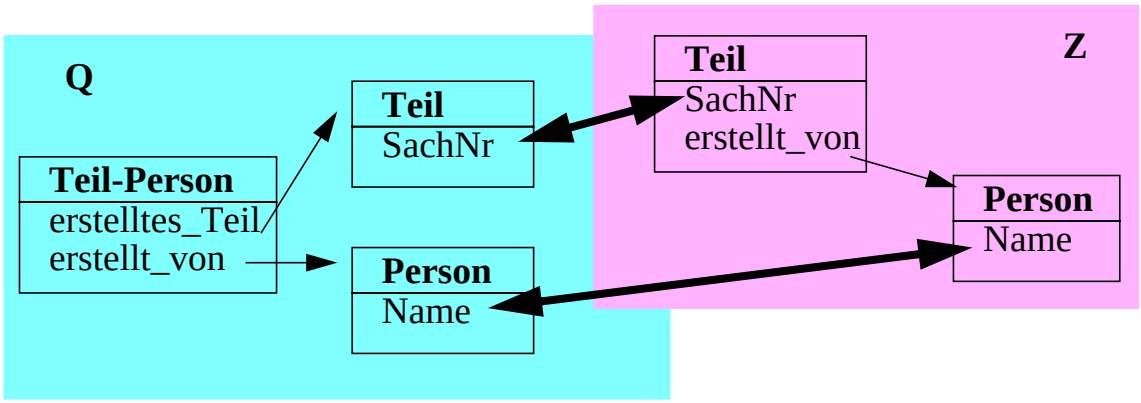
Probleme der strukturellen Heterogenität (6)

- 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)

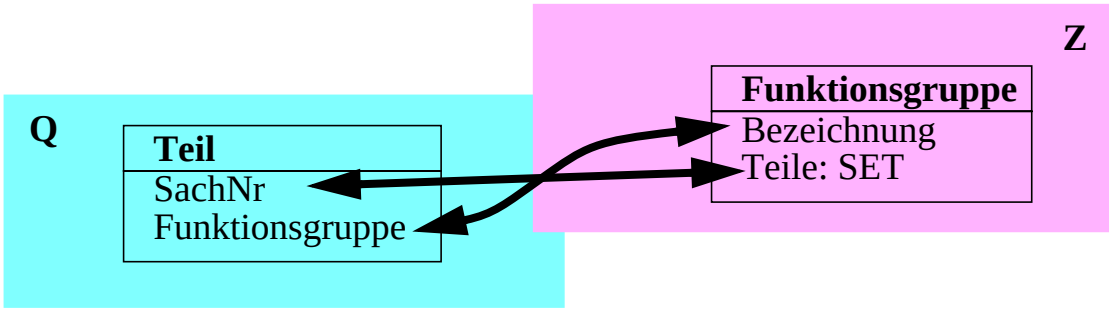
- Elementare Abbildungsprobleme (Forts.)
  - Vernetzung von Zielentities
    - abhängige Zielentities
    - bei Abbildung vernetzter Quellstrukturen in vernetzte Zielstrukturen können die Referenzen voneinander abweichen
      - z. B. können (gerichtete) Beziehungen invertiert sein
      - Beziehungstypen 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

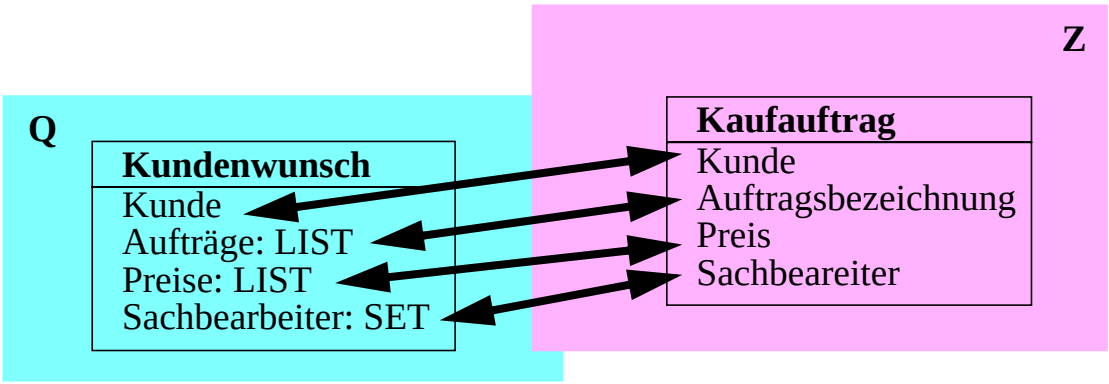


Probleme der strukturellen Heterogenität (9)

❑ Konflikte durch Einsatz objektorientierter Datenmodelle (Forts.)

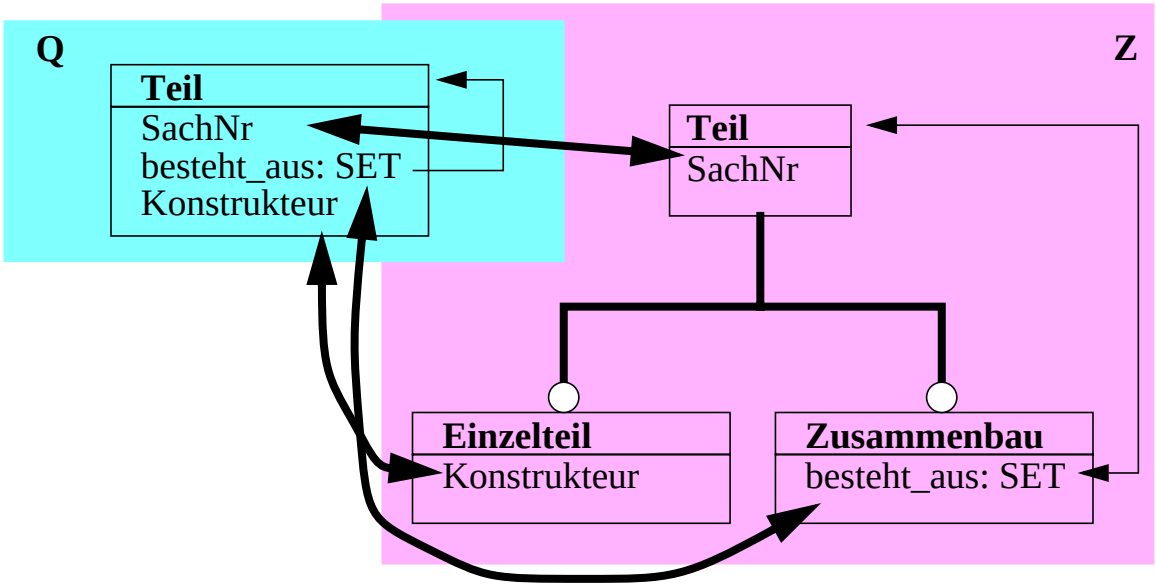
➪ Abbildung mengenwertiger Attribute

- 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

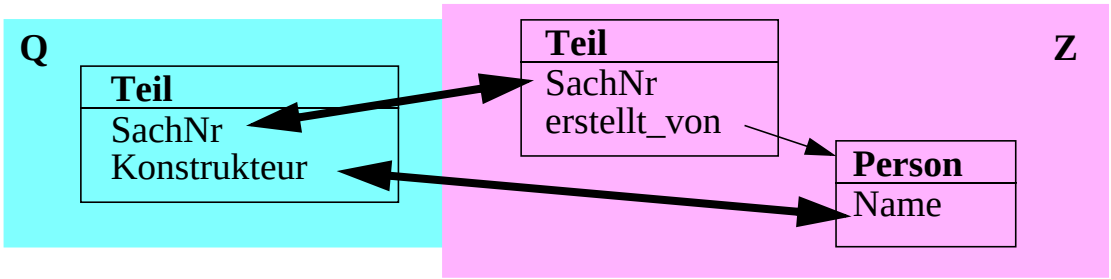


Probleme der strukturellen Heterogenität (10)

- Konflikte durch Einsatz objektorientierter Datenmodelle (Forts.)
  - Abbildung von Abstraktionskonzepten
    - Darstellung gleicher Sachverhalte auf unterschiedlichen Abstraktionsebenen
    - Beispiel: Datenebene vs. Typebene



    - Beispiel: Datentypebene vs. Objekttypebene

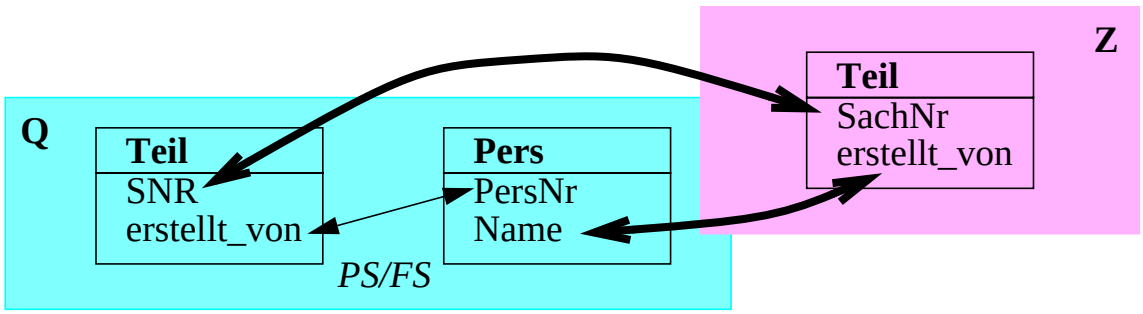


Probleme der strukturellen Heterogenität (11)

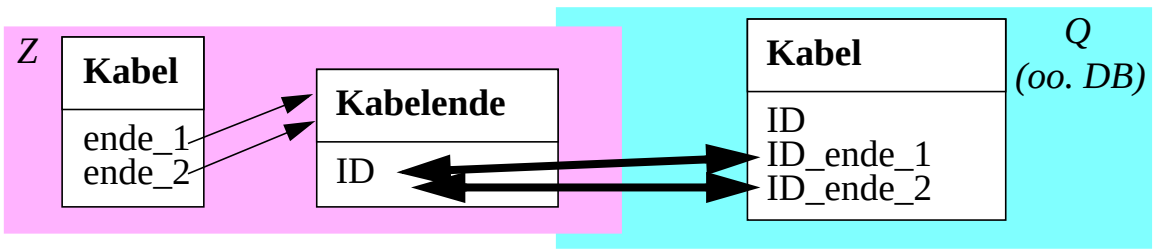
- Konflikte durch Einsatz objektorientierter Datenmodelle (Forts.)
  - Dynamische Aspekte und Integritätsbedingungen
    - zu berücksichtigen
      - Seiteneffekte von Funktionen
      - Rückgabewerte von Methoden
      - Repräsentation der Zeit
      - unterschiedliche Programmiersprachen
      - ...
    - 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)

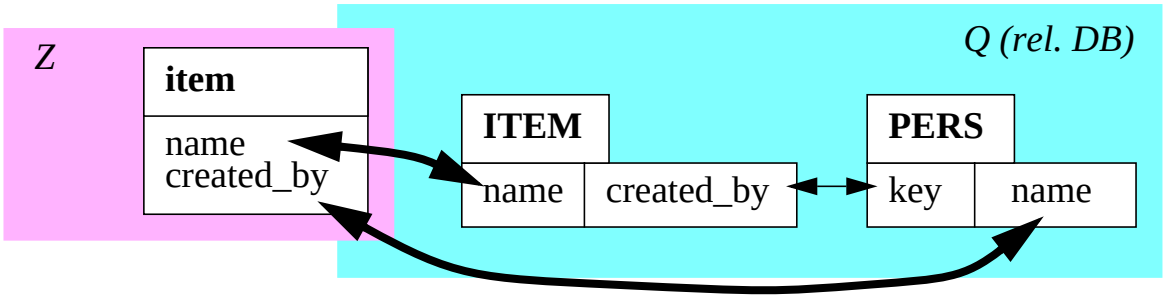
- Weitere Probleme
  - Dauerhaftigkeit von Zieldaten
    - bisher Annahme der transienten Verwaltung der Zieldaten
    - persistente Datenhaltung in Z in der Regel nur zur Datenmigration
      - dabei in der Regel nur unidirektionale Abbildung und keine Propagierung
  - Heterogenität von Datenmodellen
    - 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)

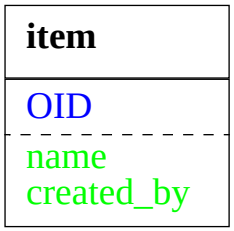
- Allgemeine Struktur einer Abbildungsspezifikation




```
 1: BEGIN
 2:   MAPPED_SCHEMAS
 3:       ts := target_schema <- rel_db:= rel_db@rel_dbs@localhost;
 4:   END_MAPPED_SCHEMAS;
 5:   INCLUDE
 6:       LIB /usr/users/sauter/libstring.a;
 7:       INC string.h;
 8:   END_INCLUDE;
 9:   TYPE_MAPPING
10:       MAP  ts.DM <- rel_db.US$;
11:                ts.DM <- 0.67 * rel_db.US$;
12:                rel_db.US$ <- 1.5 * ts.DM;
13:       END_MAP;
14:   END_TYPE_MAPPING;
15:   ENTITY_MAPPING
16:     MAP item <-_item := rel_db.ITEM, _pers:= rel_db.PERS;
17:       ON_RETRIEVE
18:           name <- _item.name;
19:           created_by <- _pers.name;
20:           IDENTIFIED_BY(_item.name, _pers.key);
21:           WHERE (_item.created_by = _pers.key);
22:       ON_UPDATE ...
23:       ON_INSERT ...
24:       ON_DELETE ...
25:     END_MAP;
26:   END_ENTITY_MAPPING;
27: END.
```

BRIITY (3)

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

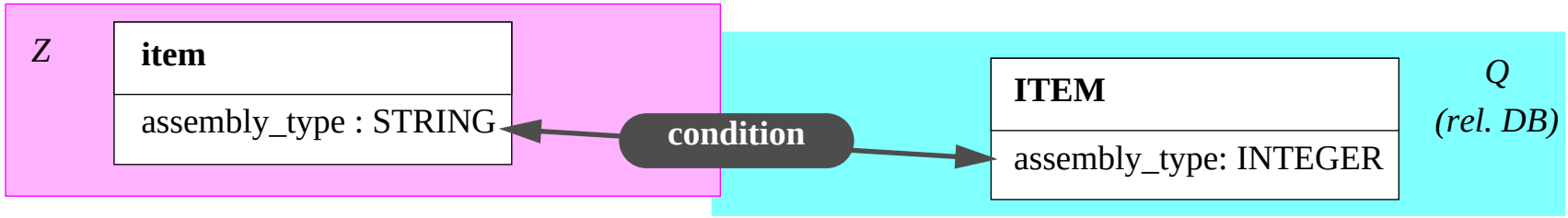



```
MAP item <- _item := rel_db.ITEM,
            _pers:= rel_db.PERS;
  ON_RETRIEVE
     IDENTIFIED_BY (_item.name, _pers.key);
     name <- _item.name;
     created_by <- _pers.name;
     WHERE (_item.created_by = _pers.key);
```

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

BRIITY (4)

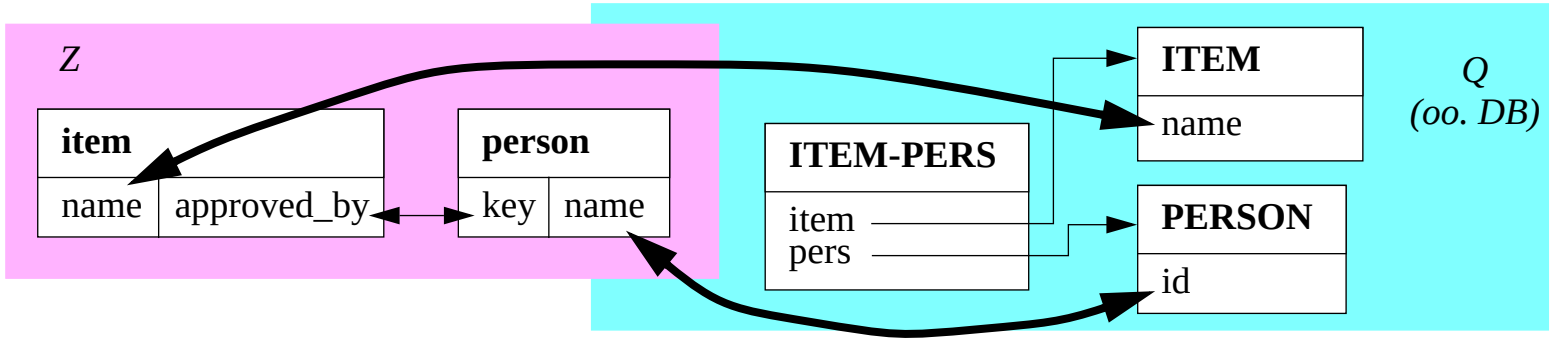
- Elementare, mengenorientierte Abbildungsregeln (Forts.)
  - Bedingte Abbildung
    - Abbildung mehrerer Quellinstanzen auf eine Zielinstanz oder
    - Konvertierung von Datentypen




```
1:  MAP item <- _item:=rel_db.ITEM;
2:      ON_RETRIEVE
3:          assembly_type <-  IF (_item.assembly_type = 512) THEN “manufacturing”
4:                            ELSE IF (_item.assembly_type = 918) THEN “configurable”
5:                            ELSE ...;
```

BRIITY (5)

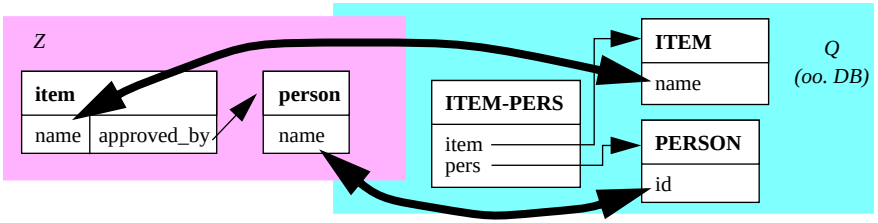
- Komplexe Abbildungsregeln
  - Referentielle Integrität in relationalen Zielschemata




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

BRIITY (6)

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

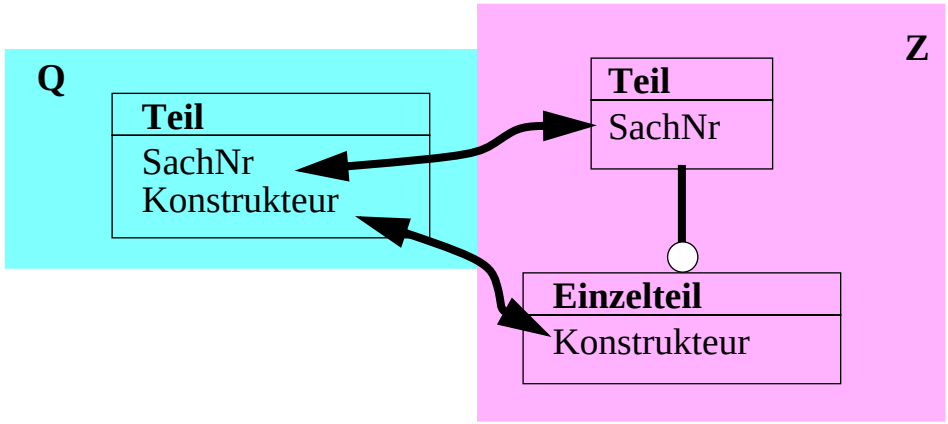



```
 1: MAP person <- oo_db.PERSON;
 2:     ON_RETRIEVE
 3:         name <- oo_db.PERSON.id;
 4:         IDENTIFIED_BY (oo_db.PERSON.id);
 5: END_MAP;
 6: MAP item <- _item:=oo_db.ITEM, _ip:=oo_db.ITEM-PERS, _pers:=oo_db.PERSON;
 7:     ON_RETRIEVE
 8:         name <- _item.name;
 9:         approved_by <-  CASCADED_MAP person
                            WITH_ID _item.INV(_ip:item).pers.id
10:         IDENTIFIED_BY   (_item.name);
11: 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>')'

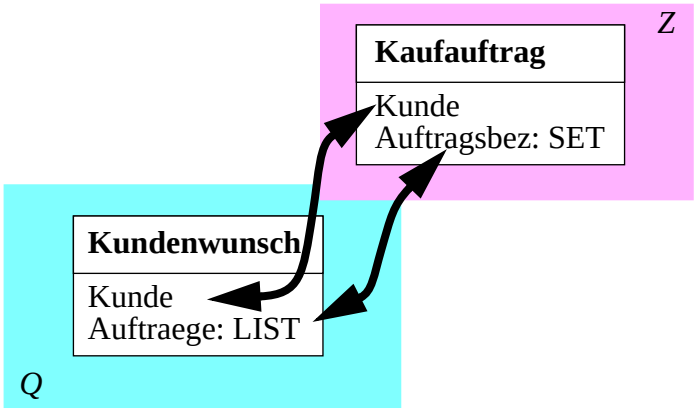
      [WHERE <DNF expression>]

      [ORDER_BY <sort expression>]

      [GROUPED_BY <source attr identification>].

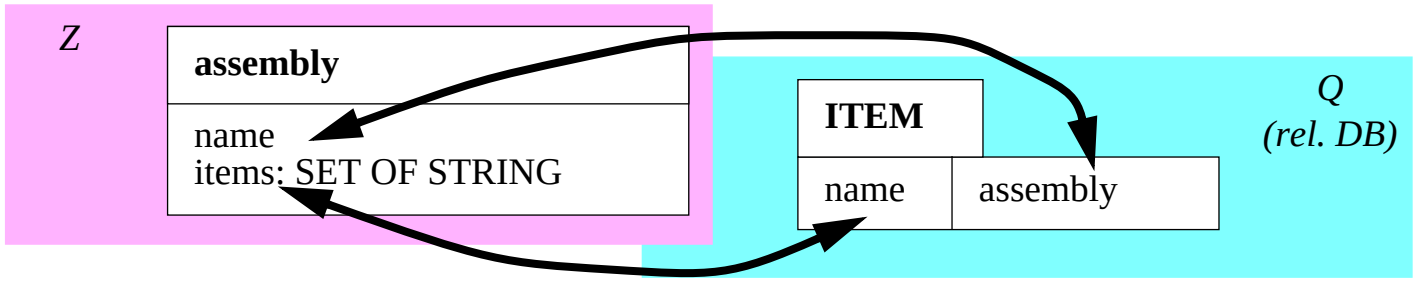
    - Beispiel

```
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)

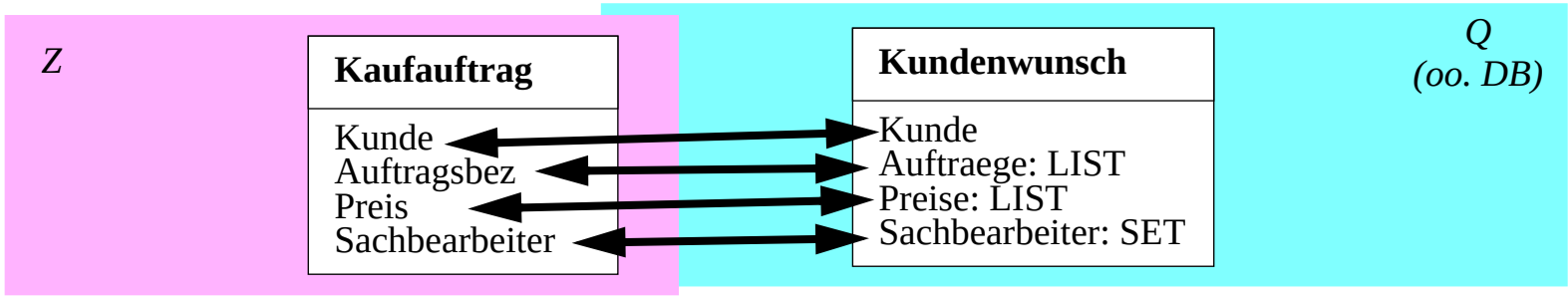
- Komplexe Abbildungsregeln (Forts.)
  - NEST-Operation




```
1:  MAP assembly <- _item:= rel_db.ITEM;
2:      ON_RETRIEVE
3:          name <- _item.assembly;
4:          items <-  NEST (_item.name)
5:                        GROUPED_BY (_item.assembly);
```

BRIITY (10)

- Komplexe Abbildungsregeln (Forts.)
  - UNNEST-Operation




```
1:  MAP Kaufauftrag <- _k:=oo_db.Kundenwunsch;
2:    ON_RETRIEVE
3:        FOR_ALL (_unnest_auftrag := UNNEST (_k.Auftraege),
4:                        _unnest_preis : = UNNEST (_k.Preise))
5:              FOR_ALL (unnest_sachbearbeiter := UNNEST (_k.Sachbearbeiter))
6:                        Kunde <- _k.Kunde;
7:                        Auftragsbez <- _unnest_auftrag;
8:                        Preis <- _unnest_preis;
9:                        Sachbearbeiter <- _unnest_sachbearbeiter;
10:       IDENTIFIED_BY (OID(_k);
11: 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>.

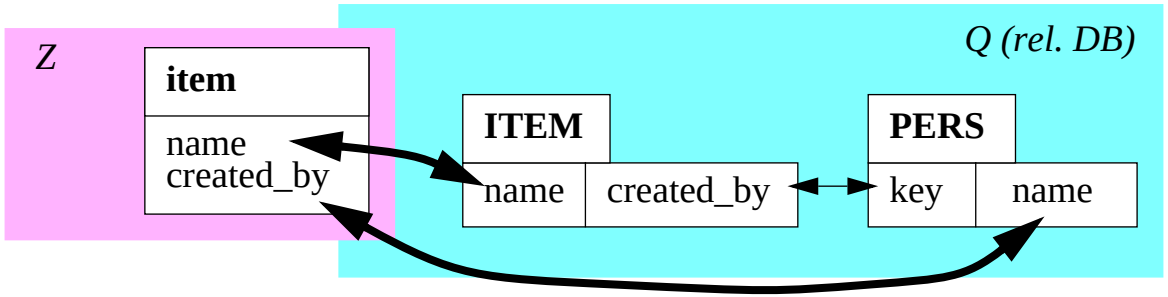
<update_statement_body> ::=
('NEW'|'MODIFIED'|'DELETED') ':' ('RESTRICTED'|'INVERSE_TO_RETRIEVE'|<assign_stmnt_list>)';'.

<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

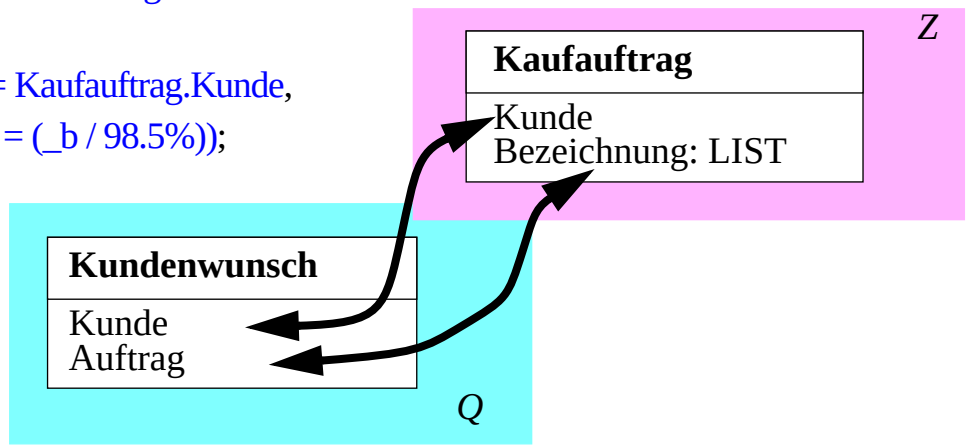



```
1:  ON_UPDATE created_by OF
2:    NEW:          ASSIGN ( IS_INSTANCE(_item: name = ts.item.name, created_by = ?1) AND
3:                           IS_INSTANCE(_pers: key = ?1, name = ts.item.created_by));
4:    MODIFIED:     ASSIGN ( IS_INSTANCE(_pers: name = ts.item.created_by));
5:                  ASSIGN ( IS_INSTANCE(_item: name = ts.item.name, created_by = ?1))
6:                  WHERE ( IS_INSTANCE(_pers: key = ?1, name = ts.item.created_by));
7:    DELETED:      ASSIGN ( NOT_IS_INSTANCE(_pers: key = ?1))
8:                  WHERE ( HAS_VALUE(?1 = ASSIGNED_ID_VALUE(2)));
```

BRIITY (13)

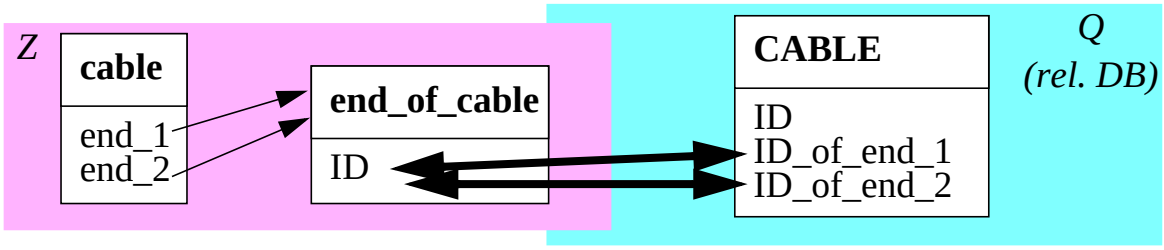
- Regeln zur Propagierung von Updates (Forts.)
  - Update-Regeln für mengenorientierte Ziel-Attribute

```
 1: MAP Kaufauftrag <- _k:=oo_db.Kundenwunsch;
 2:   ON_RETRIEVE ...
 3:   ON_UPDATE ...
 4:   ON_INSERT
 5:        FOR EACH_ELEMENT_VALUE
 6:                    _b OF Kaufauftrag.Bezeichnung DO
 7:            ASSIGN
 8:                IS_INSTANCE (_k: Kunde = Kaufauftrag.Kunde,
 9:                                Auftrag = (_b / 98.5%));
10:        END_FOR;
11: END_MAP;
```



BRIITY (14)

- Multiple Instanziierung
  - Partitionen

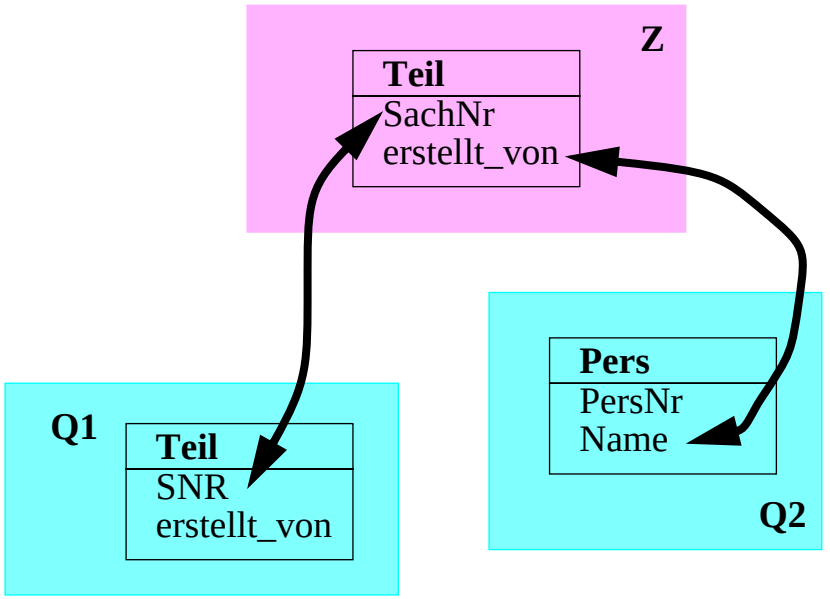



```
 1: MAP end_of_cable <-  PARTITION _part_end1: _c:=rel_db.CABLE,
                         PARTITION _part_end2: _c:=rel_db.CABLE;
 2:   PARTITION _part_end1:
 3:       ON_RETRIEVE
 4:           ID <- _c.ID_of_end_1;
 5:           IDENTIFIED_BY (_c.ID, _c.ID_of_end_1);
 6:   PARTITION _part_end2:
 7:       ON_RETRIEVE
 8:           ID <- _c.ID_of_end_2;
 9:           IDENTIFIED_BY (_c.ID, _c.ID_of_end_2);
10: END_MAP;
11: MAP cable <- _c:=rel_db.CABLE;
12:     ON_RETRIEVE
13:         end_1 <-  CASCADED_MAP end_of_cable
14:                   PARTITION _part_end1
15:                   WITH_ID (_c.ID, _c.ID_of_end_1);
16:         end_2 <-  CASCADED_MAP end_of_cable
17:                   PARTITION _part_end2
18:                   WITH_ID (_c.ID, _c.ID_of_end_2);
19:         IDENTIFIED_BY (_c.ID);
20: END_MAP;
```

BRIITY (15)

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

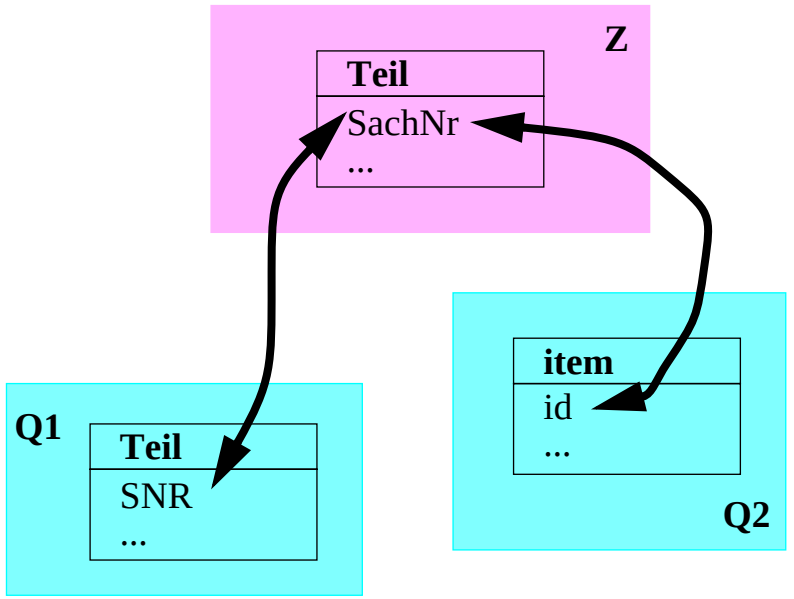
```
 1: BEGIN
 2:   MAPPED_SCHEMAS
 3:       z := integriert <-  q1 = teile_db@db2@host1,
 4:                           q2 = pers_db@oracle@host2;
 5:   END_MAPPED_SCHEMAS;
 6:   ENTITY_MAPPING
 7:     MAP Teil <- _t := q1.Teil, _p:= q2.Pers;
 8:       ON_RETRIEVE
 9:           SachNr <- _t.SNR;
10:           erstellt_von <- _p.Name;
11:           ...
12:           WHERE (_t.erstellt_von = _p.PersNr);
13:       ...
14:   END_ENTITY_MAPPING;
15: END.
```



BRIITY (16)

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

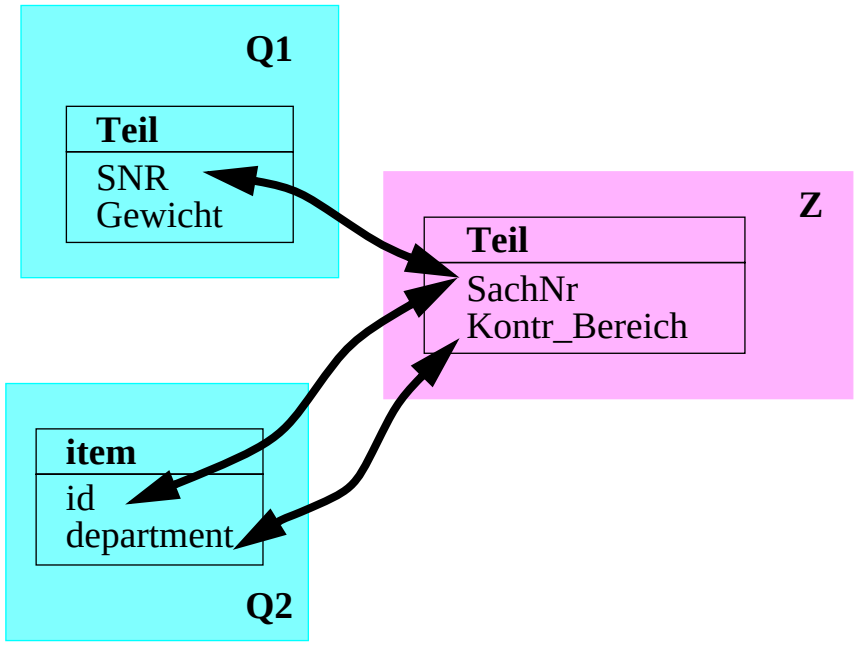
```
 1: MAP Teil <-  PARTITION  db_q1: _t := q1.Teil,
                 PARTITION  db_q2: _i := q2.item;
 2:    PARTITION db_q1
 3:      ON_RETRIEVE
 4:          SachNr <- _t.SNR;
 5:          IDENTIFIED_BY(_t.SNR);
 6:    ...
 7:    PARTITION db_q2
 8:      ON_RETRIEVE
 9:          SachNr <- _i.id;
10:          IDENTIFIED_BY(_i.id);
11:  END_ENTITY_MAPPING;
12:  ...
13: END.
```



BRIITY (17)

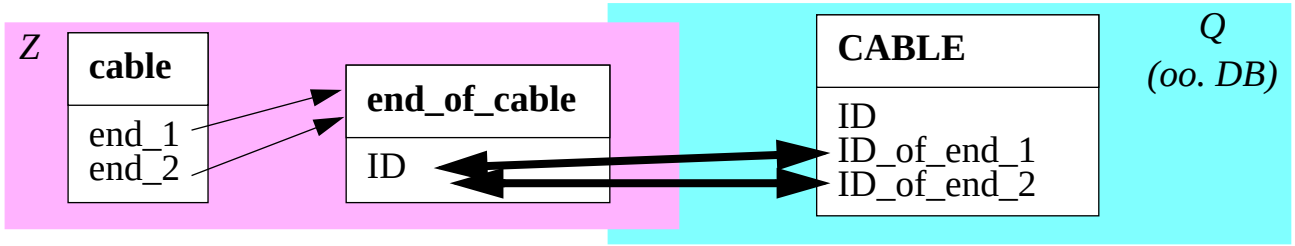
- Integration mehrerer Quellschemata (Forts.)
  - Globale Identität

```
 1: MAP Teil <-  PARTITION  db_q1: _t := q1.Teil,
                 PARTITION  db_q2: _i := q2.item;
 2:    PARTITION db_q1
 3:      ON_RETRIEVE
 4:          SachNr <- _t.SNR;
 5:          Konstr_Bereich <- "EntwicklungPkw"
 6:          IDENTIFIED_BY(_t.SNR);
 7:          GLOBAL_IDENTITY (_t.SNR,
 8:                "EntwicklungPkw");
 9:    ...
10:    PARTITION db_q2
11:      ON_RETRIEVE
12:          SachNr <- _i.id;
13:          Konstr_Bereich <- _i.department
14:          IDENTIFIED_BY(_i.id, _i.department);
15:  END_ENTITY_MAPPING;
16:  ...
17: END.
```



BRIITY (18)

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

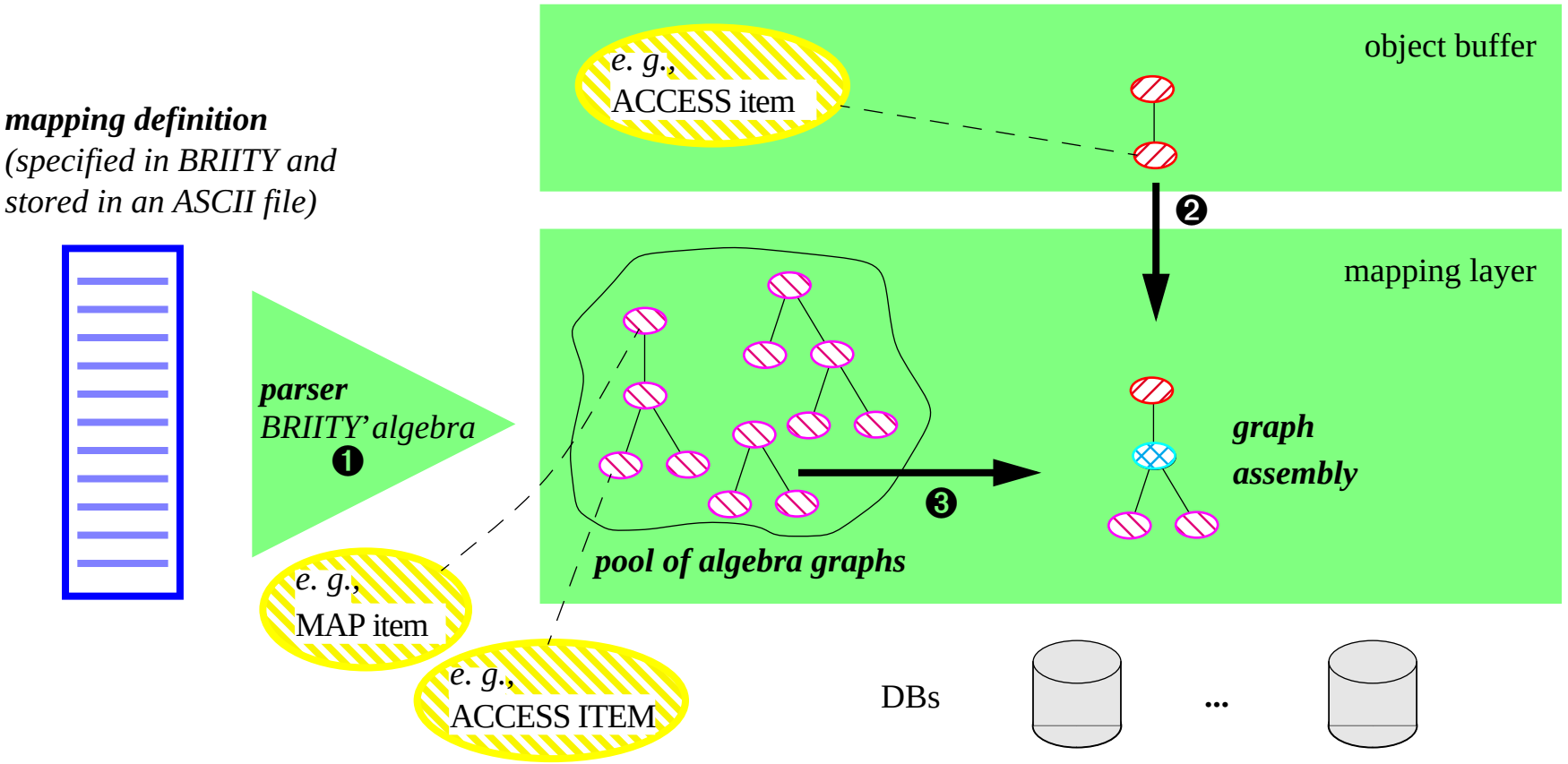



```
18:  INTEGRITY_CONSTRAINTS
19:      DEPENDENCIES
20:          GROUP     cable, end_of_cable
21:          WHERE     (cable.end_1 = OID(end_of_cable)) AND (cable.end_2 = OID(end_of_cable))
22:      END_DEPENDENCIES;
23:  END_INTEGRITY_CONSTRAINTS;
```

  - 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

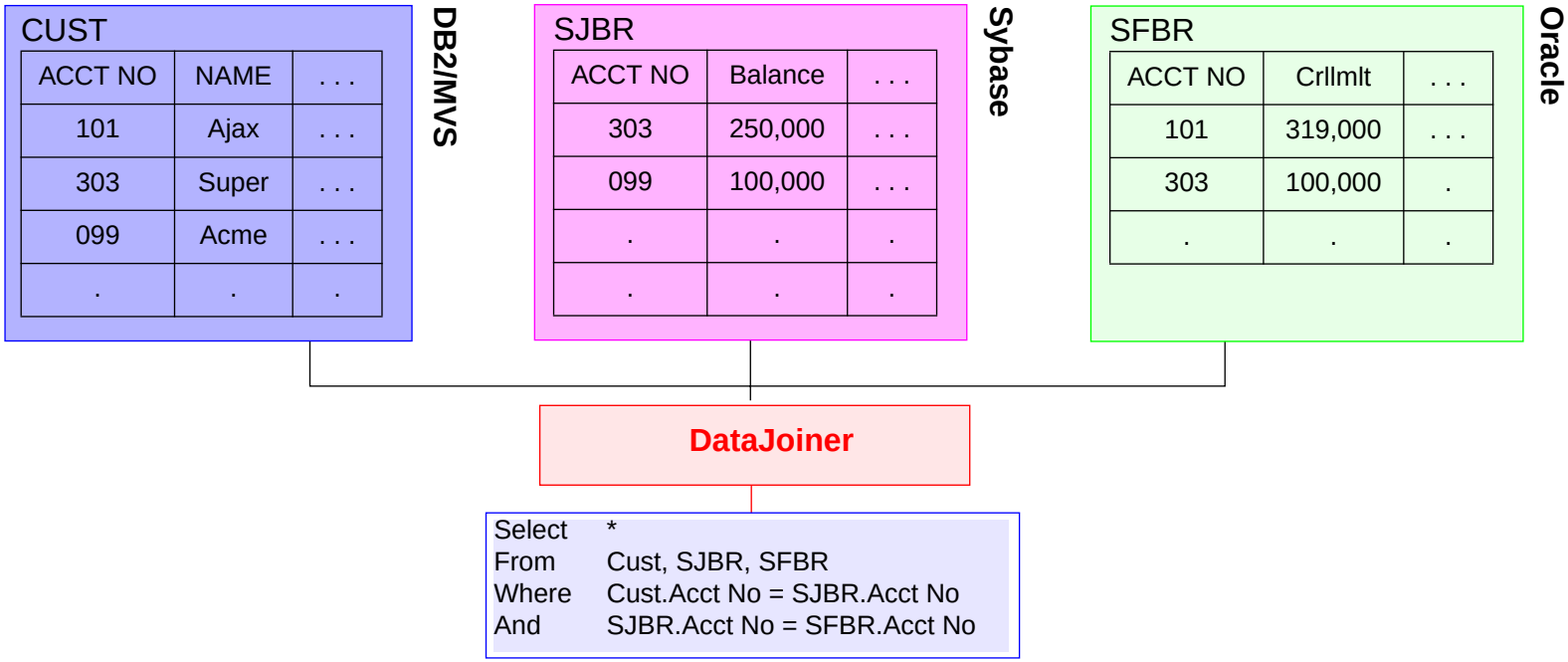
❹ Assemblierung

- 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