Information Integration – Exercise 5 – Gabriel Glaser

Task 1: Schema Mapping

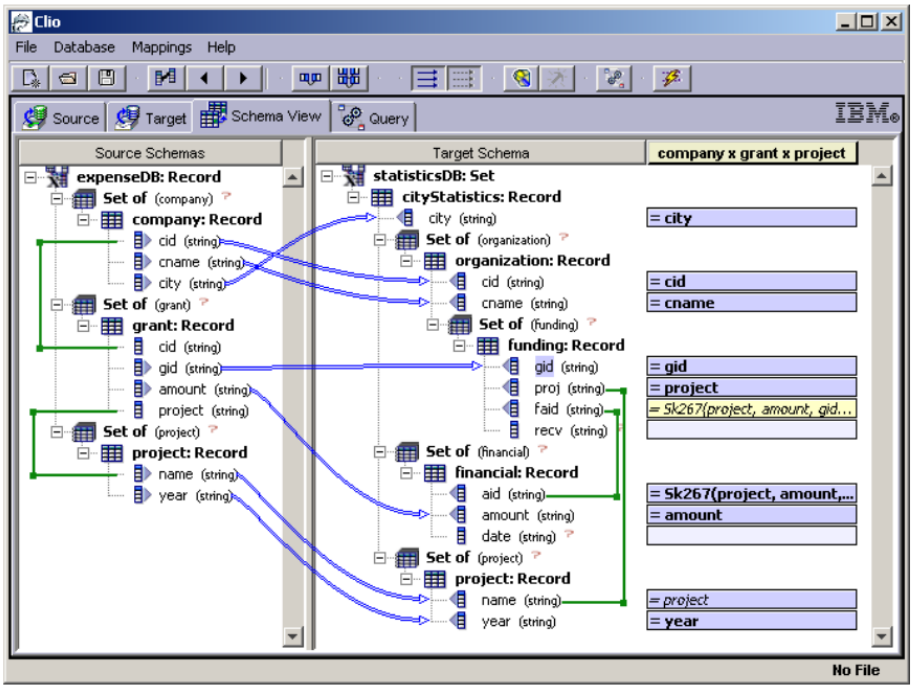

Figure 1: High-level mapping between source and target schema.

a) Provide a list of the **primary paths** of your source and target schema.

b) Provide pseudo-code that, given a schema S and integrity constraints C, enumerates all **intra-schema associations** after considering **integrity constraints**.

c) Provide the **output of your algorithm** when applied to the given source and target schema, respectively.

d) How many association combinations need to be tested to identify inter-schema associations?

e) Which of the association combinations are valid?

f) Identify which interpretation is depicted in the above screenshot. For this interpretation, write SQL queries (one or more) that fetch the data necessary for the target representation.

a) *Primary paths of source schema*:

- company

- grant
- project

Primary paths of target schema:

- cityStatistics
- cityStatistics, organization
- cityStatistics, organization, funding
- cityStatistics, financial
- cityStatistics, project

b)

```
function enumerateIntraSchemaAssociationsWithIC(S, C):
    List<List<Attribute>> primaryPaths = getPrimaryPaths(S);
    List<List<Attribute>> primaryPathsWithIC = [ ];
    for (List<Attribute> primaryPath in primaryPath):
        List<Attribute> primaryPathWithIC = [ ];
        for (0 ≤ i < length(primaryPath)):
            Attribute attribute = primaryPath[i];
            primaryPathWithIC.append(attribute);
            Set<Attribute> foreignKeyAttributes = getForeignKeys(attribute, C);
            primaryPathWithIC.appendAll(foreignKeyAttributes);
        primaryPathsWithIC.append(primaryPathWithIC);
    return primaryPathsWithIC;
```

```
function getPrimaryPaths(S):
    if isRelational(S):
        return getAllTables(S);
    else:
        Tree<Attribute> tree = recursivalyDescend(S);
        return getAllSubPathsToLeafs(tree);
```

c) *Output of algorithm applied on source schema*:

- company
- grant, **company**, **project**
- project

Output of algorithm applied on target schema:

- cityStatistics
- cityStatistics, organization

- cityStatistics, organization, funding, **financial**, **project**
- cityStatistics, financial
- cityStatistics, project

d) Three associations combinations for source schema, Five associations for target schema, i.e., $3 \cdot 5 = 15$ tests.

e) Valid association combinations:

- company → cityStatistics, organization
- grant, company, project → cityStatistics, organization, funding, financial, project
- project → cityStatistics, project

f) Interpretation depicted in screenshot:

- For each **company** element, create a **cityStatistics.organization** element with respective **cid** and **cname**. Group the created elements under **company.city** as **cityStatistics.city**.
- For each **grant** element, add a **funding** element to the set of the **cityStatistics.organization** with same **cid**. The **gid** is given, but **faid** and **recv** have to be invented (e.g., with skolem function), because there are no key-constraints or high-level mappings.

  To finish the funding element, use the **project.name** value for **funding.proj**, because **project** of source and **proj** of target are involved in a key constraint of value which are connected with a high-level mapping.

  Finally, create a **financial** entry (for the **cityStatistics** in which the a **cityStatistics.organization** with respective **cid** occurs) with the given **amount** and invented **date** and previously invented **faid** as **aid**.
- For each source **project** element (with **name** and **year**) create **cityStatistics.project** element for each **cityStatistics** for which there exists a **funding** with **project.name** = **cityStatistics.organization.funding.proj**.

SQL-Queries:

```
CREATE VIEW GrantData AS
  SELECT * FROM grant
    INNER JOIN company ON grant.cid=company.cid
    INNER JOIN project ON grant.project=project.name;
```

Get data of remaining companies which don't have a grant:

```
CREATE VIEW CompanyData AS
  SELECT * FROM company
    WHERE NOT EXISTS (SELECT grant.cid FROM grant
      WHERE company.cid=grant.cid);
```

Get data of projects and join with company such that the projects can be listed under the right **cityStatistics.city**. (A project can be listed under multiple)

```
CREATE VIEW ProjectData AS
  SELECT * FROM project
  INNER JOIN company WHERE EXISTS
    (SELECT * FROM grant WHERE grant.cid = company.cid
    AND grant.project=project.name);
```