From Enterprise Models to Dimensional Models: A Tool to transform your database
Data Modelling Course Project

Jonas Munch j.munch@campus.fct.unl.pt
Dusan Zeliar d.zeliar@campus.fct.unl.pt

***Abstract*— This technical report describes our work of the Data Modelling class 2017. The goal of this work is to implement a tool that supports the user to transform a relational schemed database (OLTP) into a dimensional model (OLAP). Based on an approach proposed by Moody a so called star schema will be produced out of an arbitrary database.**

The program guides the user via commandline interface through all the neccessary steps to find an agreement of which tables are selected as transaction entities, component entities and classification entities.

As soon as the selection phase is done, our software will generate a dimensional model based on the input scheme and populate it from the original data.

I. Introduction

In recent years the increasing amount of collected data led to the possibility on the one hand, but also to the requirement of big data analysis. More and more data was being produced. Especially retailers and the sector of e-commerce are producing enormous amounts of data every day and have the urge to get aggregated information out of this data. In order to cope with the growing data, extract programs were built that made the data usable for specific applications. In this manner data was pulled and copied to other places and under the control of someone else. Soon problem occured like Lack of data credibility, issues in productivity and the inability to transform data into information and a solution was needed to clear the mess. Therefore the the idea of *data warehouses* arose. A data warehouse is an analytical database that is used as the foundation of a decision support system. It is designed for large volumes of read-only data. providing intuitive access to information that will be used in making decisions. Particularly it is designed to be a "self-service" for analysts to generate information out of the data rather relying on IT specialists.

A. The approach of data warehouses

Moody *et al.* propose in [1] an approach of how a data warehouse can be structured as well as the methodology how to transform an enterprise database into the proposed structure. They mention several possible schemes, namely *Flat*, *Terraced*, *Star*, *Snowflake Scheme*, that are suitable for a data warehouse. Especially they name the *Star Scheme* as a generic model.

In contrast to conventional OLTP Enterprise models the star scheme based OLAP models have the following advantages. While OLTP models are desinged in a normalized way to get rid of redundancy, the data in dimensionals models is highly redundand and denormalized. Due to the introduced redundancy the system is way more performant because only a few join need to be made for a query. Besides that the denormalized model improves the ease-of-use, because the small number of tables can be understood easily. Using the metaphor of a dimensional cube, it is made easier to think of operations that need to be made in order to get the desired results.

B. A tool to generate dimensional models

At a point where there exists a lot of interesting data in an enterprise database a tool comes in handy that can transfer this data into a dimensional model. According to this we implemented in this project a tool which transforms an arbitrary database scheme into a Star Scheme for a data warehouse. This paper describes how we achieved that goal. In chapter II we will clarify the methodology proposed by [1]. Section III gives detailed information about how we implemented the system, which algorithms we came up with and how the tool is designed. In order to show how the tool performs there are some results and evaluation shown in section IV.

II. Theory

The method for developing dimensional models from Entity Relationship models proposed by Moody *et al.* in [1] was used by us as a basis to implement the program. The method is here only quicly outlined, for deeper information refer to the original paper. Moody's method consists basically of three steps:

a) Classify Entities: In the first step the entities of the Entity Relationship model need to be classified into one of the following categories: *transaction*, *component* or *classification*.

The transaction tables describe certain events that occur in the business (e. g., sales in a retail store) and contain measurements or quantities that my be summarized. The transaction tables are the most important in a data warehouse and will result later in so called *fact tables*.

A component entity contains further information about the facts in the transaction tables and will therefore result later in so called *dimension tables* of a star schema. The components are directly related to the transaction entities via a one-to-many relationship.

Other tables that are related, directly or transitively, to the component via one-to-many relationships will be categorized as classification entities. They are adding even more information (can be more abstract or more detailed) to the components and will accordingly end up in the dimension tables.

b) Identify Hierarchies: Hierarchies in terms of [1] are chains of one-to-many relationships, all aligned in the same direction. The terms *minimal* and *maximal* are introduced in this context. Entities are called minimal if they occur at the bottom of a hierarchy chain, and are called maximal if they occur at the top. Hierarchies are important, because they can be collapsed. Collapsing is an important step in simplifying the database schema by introducing redundancy.

c) Produce Dimensional Models: In order to transform the database, Moody describes two operators. Firstly the hierarchies found in the step before get *collapsed*. Beginning at the maximal entities the data is gradually merged into the entity's child table: By resolving foreign key constraints the data is duplicated and redundancy is produced. This process can be continued until the minimal entity is reached and all the data is located in a single table. However, for generating a star schema the collapsing process should be stopped when a component entity is reached in order to construct a proper dimension.

The second operator is called *aggregation* and can be applied optionally. The goal here is to summarize the data in order to present it in a more compressed way (by doing this we will lose detail of information that cannot be reversed). To achieve aggregation, columns of a table must be selected to act as aggregation keys (by which the data will be grouped) and an aggregation function must be specified for the aggregated columns (numerical attributes like price, quantity, etc.).

III. Implementation

A. Application Design

The tool we implemented is designed as pipeline that takes an arbitrary database as input and outputs a set of SQL scripts, which can be run to create a new database having a dimensional scheme. The user is guided throughout the whole process via a command line interface, which enforces him to do the steps in the right order. As a special feature we implemented a mechanism to save the current working state in order to continue the work at a later point in time. As soon as all the specifications about the resulting scheme are made, the program will output and run the resulting SQL scripts in a way that the new database is created and the original data is populated.

a) Making suggestions: The program tries to support the user in a helpful way. After the input database scheme is read, the system will come up with some suggestions for the categorization of tables (according to II-.0.a) into *transaction*, *component* and *classification*.

Since it is a hard job to find transaction candidates from only the database schema without knowing about the business, we decided to use a different approach. It is likely that the user will choose a minimal entity as transaction because it is one of the roots of the database and therefore a very important table for the business. In order to obtain every minimal entity we find every table that is not referenced by any foreign key. As we consider the database scheme as a directed graph (tables are nodes and foreign keys are directed edges) we can speak about the transaction tables as roots of a spanning tree. The direct neighbours of the roots are suggested to be classified as components. Every other table that is part of the spanning tree, i. e., reachable by foreign keys from the root, is suggested to be a classification table.

For sure, if the user not satisfied with this suggestion he can alter the classification at the given time to fulfill his needs.

b) User interaction: The generation of a dimensional model cannot be done completely automatically. The user's needs and desires are unpredictable and the according structure of dimensional model as well. Therefore we need user interaction letting him input his needs as well as supervision and confirmation throughout the process. In order to accomplish that, we implemented a commandline interface which guides the user step by step helping him to give the commands in the right order. Every time a suggestion is made, it is presented to the user, letting him modify for his special needs and confirm when he thinks the right design is chosen. Since we aware that mistakes can happen, the program offers an undo-option at certain points in the process.

At very first, the input database scheme is parsed and displayed to the user to give a overview of what to do. Figure 3 shows a short version of this output. After the database scheme is parsed, a suggestion of transaction tables is made. See in figure 4 how the suggestion is presented to the user. The transaction entities form the core of the dimensional model because they will result into fact tables. That is why the user must select the transaction entities prior to any other step.

When the transaction entities are selected the system can proceed to suggest component and classification entities. Figure 5 shows the interface of this step. Here the user can also classify some table as *unclassified* which basically means, that the table's columns and data will not occur in the resulting dimensional scheme.

The information collected by now is sufficient to create a dimensional scheme. Moody *et al.* propose in [1] a second optional step, the *aggregation*. Hence the user is given the ability now to choose columns from every transaction table to be aggregated. Doing this he also needs to specify an aggregation function.

Are all this steps completed, the program will generate the necessary SQL scripts and run them automatically.

However, we know that using the program is hard work including many steps, so the user may want to have a coffee when the work is halfway done. Being aware of this the program comes with a handy feature to save the current working state at any time. The work done so far can be saved as a JSON-file and loaded into the program again to

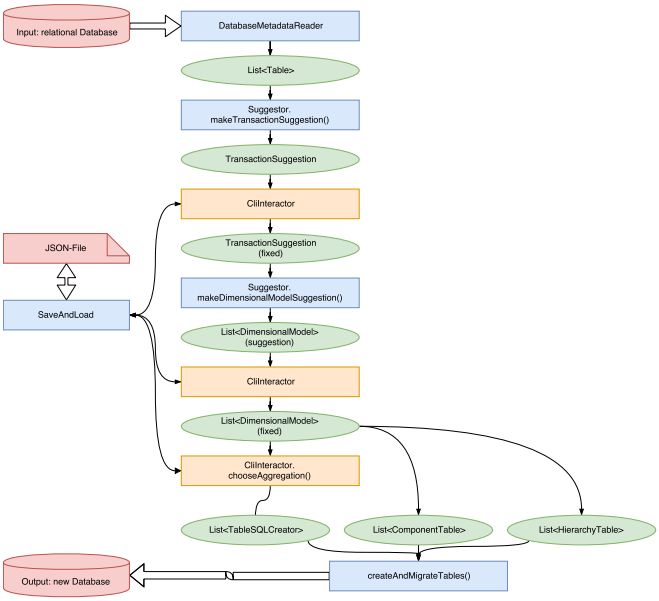

Fig. 1: Dataflow Diagram of the Pipeline.

continue working from that point.

c) Generating SQL: The generating process starts by forming a graph-like structure of entities. Independent structures are created for each of the transaction and component entities, which are set as the roots. Using depth first search approach, all of the referenced entities from root are processed. Processing of entities consists of creating parts of the final SQL query. Depending of entity type and its position in graph, different SQL query parts are created. By aggregating data from all structures, final SQL queries for creating new table and migrating data are made.

Process of aggregating columns data in transaction tables consists of defining data types of aggregated columns, setting columns names and generating migrating SQL scripts.

B. Implementation Details

As a first step the input database is parsed using the JDBC library[1].

JDBC is also used to retrieve information about foreign key constraints: The transaction table suggestions, which are based on the number of referencing tables, are made by using this information.

With list of all tables in source database, `TransactionSuggestion` creates default suggestion of classification of entities based on number of foreigh keys referencing each table. Suggestion is presented to user by `TransactionSuggestion` class which also handles following user modification of final tables classification.

The main algorithm of collapsing multiple tables to one is implemented in class `HierarchyTable` and its method `createReferencedTableList(List<Table> allTabs)`.

[1]`http://www.oracle.com/technetwork/java/javase/jdbc/index.html`

```
void createReferencedTableList(List<Table>
    allTabs){
 ExpandTransTable ht;
 this.consideredTabs=allTabs;
 for (Column col :
     this.table.getFKColumns()){
  for (Table tab: allTabs){
   if
       (tab.getName().equals(col.getFKeyTable())){

    String tableToJoin;
    if (hasMultipleKeysToTheSameTable(col)){

     tableToJoin=
         createTableToJoinStr(distrPrefix,
         col.getFKeyTable());
     ht=new ExpandTransTable(tab, prefix,
         allTabs);
    }
    else{
     tableToJoin=
         createTableToJoinStr(distrPrefix,
         col.getFKeyTable());
     ht=new ExpandTransTable(tab,
         prefixMultipleFK, allTabs);
    }

    this.addTableToSelectFromPart(col,
        tableToJoin);
    ht.createReferencedTableList(allTabs);
    this.referencedTabs.add(ht);
   }
  }
 }
 if (!this.getIsRootTable()){
  this.createChildSqlParts();
 }
 else
  this.createFinalSqlScripts();
}
```

Fig. 2: Process intity structure

Starting from root table, this function is recursive called on all of the referred tables. To every lower level of graph, a table prefix is distributed to keep track of actual table level and its path from root. In case of multiple keys referring one table, foreigh key is taken into account. After all refered tables are processed, actual table creates its own parts of SQL query. The root table then creates final sql query with `createFinalSqlScripts()`, which extracts query parts of all tables and put it into final queries. Migration is handled by `createAndMigrateTable(Credentials credentials, String select, String insert, String create)` by running create, select and in sert query creating a new record for all retrieved source data.

C. Usage of the Tool

Usage consists of three steps. First step lists first suggestion of tables classification and allows to change transaction entities. Seconds step allows to filter component and clas-

```
Customer (importedKeys: 2, exportedKeys: 1)
 Cust_Id [INT] (PK)
 Cust_Name [VARCHAR(255)]
 Cust_Type_Id [INT]
     (FK->CustomerType:Cust_Type_Id)
 Cust_Regn_Id [INT] (FK->Region:Regn_Id)

CustomerType (importedKeys: 0, exportedKeys:
    1)
 Cust_Type_Id [INT] (PK)
 Cust_Type_Name [VARCHAR(255)]

FeeType (importedKeys: 0, exportedKeys: 1)
 Fee_Type_Id [INT] (PK)
 Fee_Type_Name [VARCHAR(255)]

Location (importedKeys: 2, exportedKeys: 1)
 Loc_Id [INT] (PK)
 Loc_Name [VARCHAR(255)]
 Loc_Regn_Id [INT] (FK->Region:Regn_Id)
 Loc_Type_Id [INT]
     (FK->LocationType:Loc_Type_Id)

LocationType (importedKeys: 0, exportedKeys:
    1)
 Loc_Type_Id [INT] (PK)
 Loc_Type_Name [VARCHAR(255)]

Period (importedKeys: 0, exportedKeys: 2)
 Date [INT] (PK)
 Mth [INT]
 Qtr [INT]
 Yr [INT]
 Fiscal_Yr [INT]

Sale (importedKeys: 4, exportedKeys: 2)
 Sale_Id [INT] (PK)
 Sale_Date [INT] (FK->Period:Date)
 Sale_Posted [INT] (FK->Period:Date)
 Cust_Id [INT] (FK->Customer:Cust_Id)
 Loc_Id [INT] (FK->Location:Loc_Id)
 Discount_Amt [FLOAT]

SaleFee (importedKeys: 2, exportedKeys: 0)
 Sale_Id [INT] (PK) (FK->Sale:Sale_Id)
 Fee_Type_Id [INT] (PK)
     (FK->FeeType:Fee_Type_Id)
 Fee [FLOAT]

SaleItem (importedKeys: 2, exportedKeys: 0)
 Sale_Id [INT] (PK) (FK->Sale:Sale_Id)
 Prod_Id [INT] (PK) (FK->Product:Prod_Id)
 Qty [INT]
 Unit_Price [FLOAT]

State (importedKeys: 0, exportedKeys: 1)
 State_Id [INT] (PK)
 State_Name [VARCHAR(255)]

...
```

Fig. 3: Output of parsed metadata

sification tables, which will be migrated. Finally, a form of star schema is specified, either one connected star schema, or own star for each transaction entity.

IV. Evaluation

The evaluation was perfomed by recreating example OLAP database from moody. We created copy of OLTP database and run application with the same entities classification. A new star schema database was created and the data were successfully migrated.

V. Conclusion

In this project, we implemented and tested creation of a basic dimensional model from Entity Relationship model. After studying existing methods of creating dimensional models we proposed an application. The application work flow starts with getting source database schema, suggesting a classification of entities and identifying hierarchies in model. The next step is collapsing hierarchies and aggregating data. Finally a new dimensional model database is created and the data is migrated.

The created dimensional database is in a form of a star schema. The star schema is either one, connecting all transaction entities, or multiple independent stars where each of them consists of a transaction entity and its dimensions. The schema shares the same dimensions between entities.
The developed application can be extended by a graphical interface and a more complex suggestion system.

Bibliography

[1] Daniel L Moody and Mark AR Kortink. From enterprise models to dimensional models: a methodology for data warehouse and data mart design. In *DMDW*, page 5, 2000.

```
The transaction tables must be selected!

TRANSACTION TABLES:
1. SaleFee
2. SaleItem
UNCLASSIFIED TABLES:
1. Sale
2. Customer
3. Location
4. Product
5. CustomerType
6. FeeType
7. LocationType
8. ProductType
9. Region
10. State
11. Period
Possible actions: set, save, fix, undo, help
```

Fig. 4: Commandline interface: Suggestion and Selection of *Transaction entities*

```
For every selected transaction table the
    components and classifications must be
    specified!
 * SaleFee
   SaleItem

TRANSACTION TABLES:
1. SaleFee
COMPONENT TABLES:
1. Sale
2. FeeType
CLASSIFICATION TABLES:
1. CustomerType
2. Region
3. Customer
4. Location
5. Period
6. LocationType
7. State
UNCLASSIFIED TABLES:
Possible actions: set, save, fix, undo, help
```

Fig. 5: Commandline interface: Suggestion and Selection of *Component and Classification entities*

```
For every selected transaction table an
    aggregation can be specified!
 * SaleFee
   SaleItem
Do you want to aggregate this table
    'SaleFee'? Possible actions: yes, no

> yes

AGGREGATION KEYS:
1. Fee_Type_Id : SUM(Fee)
UNCLASSIFIED COLUMNS:
1. Sale_Id
2. Fee
Possible actions: set, save, fix, undo, help
```

Fig. 6: Commandline interface: Selection of *Aggregation functions*

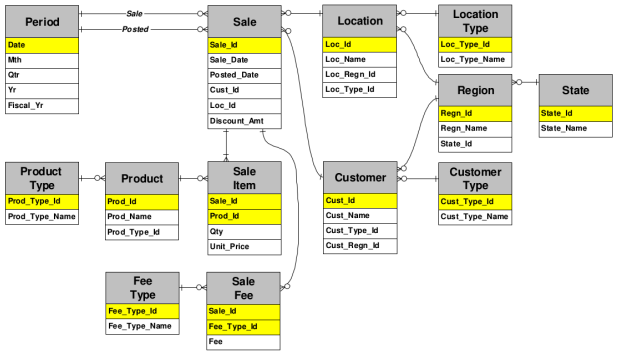

Fig. 7: Example OLTP database scheme by Moody *et al.* [1]

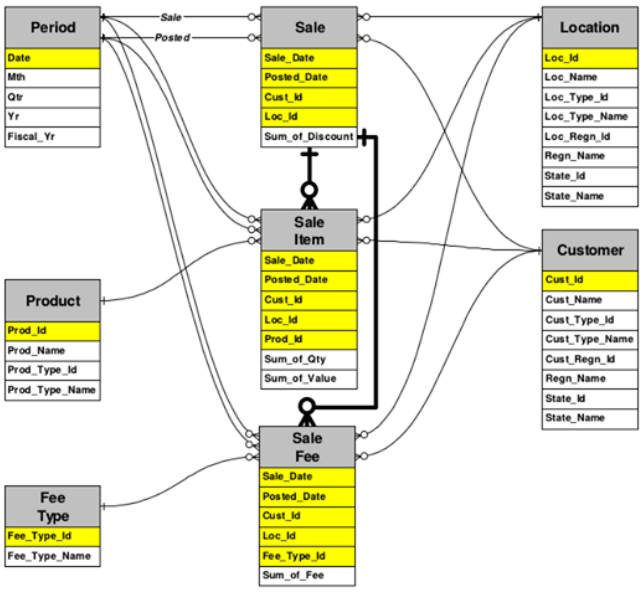

Fig. 8: Created example OLAP database scheme by Moody *et al.* [1]