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).

As proposed in [1] as a first step the tables must be categorized in *transaction*, *component* and *classification*.

b) User interaction: SaveAndLoad

c) Generating SQL: TODO Dusan

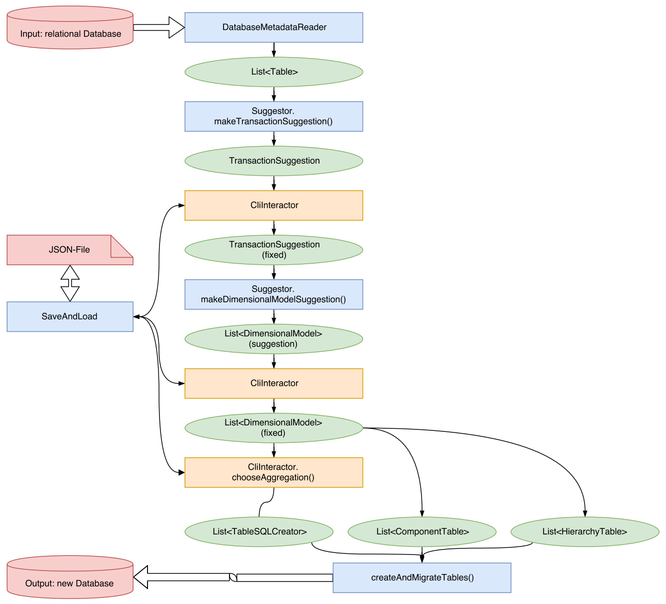
Input: relational Database
DatabaseMetadataReader
List<Table>
Suggestor. makeTransactionSuggestion()
TransactionSuggestion
CliInteractor
JSON-File
TransactionSuggestion (fixed)
SaveAndLoad
Suggestor. makeDimensionalModelSuggestion()
List<DimensionalModel> (suggestion)
CliInteractor
List<DimensionalModel> (fixed)
CliInteractor. chooseAggregation()
List<TableSQLCreator>
List<ComponentTable>
List<HierarchyTable>
Output: new Database
createAndMigrateTables()

Fig. 1: Dataflow Diagram of the Pipeline.

B. Implementation Details

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

C. Usage of the Tool

IV. Evaluation

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 schemas, each start consists of a fact table and its dimensions. The schema shares the same dimensions between fact tables but does not contain connection between fact tables. The developed application can be extended by a connection between facts table to contain full star scheme of source database.

Acknowledgment

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.

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

```
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. 2: Output of parsed metadata

```
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. 3: 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. 4: 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. 5: Commandline interface: Selection of *Aggregation functions*

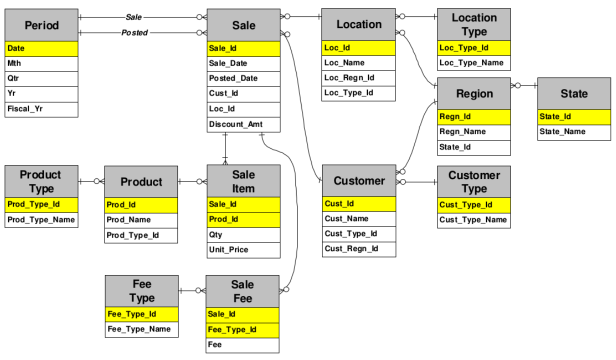

Fig. 6: Example OLTP database scheme by Moody *et al.* [1]