Chapter 3
Deductive Data Warehouses: Analyzing Data Warehouses With Datalog (By Example)

Kornelije Rabuzin
University of Zagreb, Croatia

ABSTRACT
This chapter presents the concept of “deductive data warehouses.” Deductive data warehouses rely on deductive databases but use a data warehouse in the background instead of a database. The authors show how Datalog, as a logic programming language, can be used to perform on-line analytical processing (OLAP) analysis on data. For that purpose, a small data warehouse has been implemented. Furthermore, they propose and briefly discuss “Datalog by example” as a visual front-end tool for posing Datalog queries to deductive data warehouses.

INTRODUCTION
Databases (DB) have been with us for a long time: relational databases for more than four decades, and hierarchical and network databases for even longer. Codd’s papers of the 1960s and 1970s on the relational data model have greatly influenced research in past decades. The ability to store and efficiently manage large amounts of data is very important in the modern era, making relational databases quite popular. Some good database reference works include (Date, 2004; Garcia-Molina, Ullman & Widom, 2009; Paton, 1998; Date, 2012; Hernandez, 2013; Coronel & Morris, 2016; Hoffer, Ramesh, & Topi, 2015; Silberschatz, Korth, & Sudarshan, 2011; Elmasri & Navathe, 2003).

In order to build a database that end users (usually) perceive as a set of tables, various techniques may be used to build a data model for an application domain. One of the most famous such techniques is the ERA (Entity – Relationship – Attribute) diagram. This technique has been in use for almost four decades, making it one of the few ideas in the IT field that could perhaps be described as “old.” The reason for its popularity is that it is simple to understand, even for non-technical users. When building a model, the main goal is to define entities (i.e., types of entities), relationships among them (i.e., types of relation-
Deductive Data Warehouses

ships) as well as attributes (i.e., those that belong to types of entities). Once an ERA diagram is drawn, it can easily be transformed into a set of tables. With a little experience, we can avoid some common pitfalls and end up with good database design. A good book on database design is (Hernandez, 2013).

Normalization is also relevant for database design. During normalization, the set of normal forms (1NF, 2NF, 3NF, BCNF, 4NF, 5NF, etc.) is used to reduce anomalies and to avoid potential inconsistencies, which also advances good database design. Although people that work with databases are not usually aware of normalization theory, logical database design is a very important component of good database design. Since a more detailed discussion of normalization is beyond the scope of this paper, please consult (Date, 2012) for more information on database design.

In order to work with relational databases, a database language must be used. The most popular of these languages is SQL (Structured Query Language), which is both standardized and supported by many relational database management system vendors. SQL consists of a set of statements that can be divided into several categories:

- **DDL** stands for Data Definition Language; DDL is used to define (create) objects in the database (i.e., tables, functions, sequences, etc.),
- **DML** stands for Data Manipulation Language; DML is used to manipulate data, i.e., to insert, update or delete data entries, and
- **QL** stands for Query Language with a SELECT statement used to query the data from the database.

In order to use certain statements, one has to be familiar with statement syntax. The most popular way to represent (define) the syntax of an SQL statement is to use BNF (Backus-Naur Form) notation. This notation uses several symbols (like [], {}, …) whose meanings allow one to construct a valid SQL statement. For example, this is the syntax of an SELECT statement in the PostgreSQL 10 database management system (https://www.postgresql.org/docs/10/static/sql-select.html):

```
[ WITH [ RECURSIVE ] with_query [ , ... ] ]
SELECT [ ALL | DISTINCT [ ON (expression [ , ... ]) ] ]
  [ * | expression [ [ AS ] output_name ] [ , ... ] ]
  [ FROM from_item [ , ... ] ]
  [ WHERE condition ]
  [ GROUP BY grouping_element [ , ... ] ]
  [ HAVING condition [ , ... ] ]
  [ WINDOW window_name AS (window_definition) [ , ... ] ]
  [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
  [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [ , ... ] ]
  [ LIMIT { count | ALL } ]
  [ OFFSET start [ ROW | ROWS ] ]
  [ FETCH { FIRST | NEXT } [ count ] [ ROW | ROWS ] ONLY ]
  [ FOR ( UPDATE | NO KEY UPDATE | SHARE | KEY SHARE ) [ OF table_name [ , ... ] ] [ NOWAIT | SKIP LOCKED ] [ ... ] ]
```