DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MODELING AND QUERYING BITEMPORAL SEMISTRUCTURED DATA WAREHOUSES

by

Gözde ASLAN

July, 2013

İZMİR

MODELING AND QUERYING BITEMPORAL SEMISTRUCTURED DATA WAREHOUSES

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Computer Engineering, Computer Engineering Program

by

Gözde ASLAN

July, 2013

İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **“MODELING AND QUERYING BITEMPORAL SEMISTRUCTURED DATA WAREHOUSES”** completed by **GÖZDE ASLAN** under supervision of **ASST. PROF. DR. CANAN EREN ATAY** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Yrd.Doç.Dr. Canan Atay

[illegible]

Supervisor

[illegible]

Y.Doç.Dr. Reyat YILMAZ

(Jury Member)

[illegible]

Yrd.Doç.Dr. Semih UTKU

(Jury Member)

[illegible]

Prof.Dr. Ayşe OKUR

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

I would like to present my sincere gratitude to my advisor Asst. Prof. Dr. Canan Eren Atay for her significant guidance, help, suggestions, patience, and support throughout all the phases of this study.

I would like to also thank to parents Ayla Aslan and Reşat Aslan and my brother Ali Uğur Aslan for their worthy patient and support.

Furthermore, I also thank to my fiancée Ahmet Burak Alp for his continuous patient and support. He listen my concerns every time and encouraged me.

Lastly, I would like to extend my sincere gratitude to all the staff and academic personnel of the Computer Engineering Department and Graduate School of Natural and Applied Sciences.

Gözde ASLAN

MODELING AND QUERYING BITEMPORAL SEMISTRUCTURED DATA WAREHOUSES

ABSTRACT

Data Warehouse is very common and achieved prominence in recent years is a system that supplies more effective and rapid reports. Warehouse data is stored in a non-transactional repository. Easy reporting and reasoning via prepared warehouse cubes is ensured. It is important to understand customer's tendencies and increase the customer satisfaction in corporations. At this point warehouse is a great solution.

Temporal approaches are very important while analyzing historical data. Knowledge of old times is required on data analysis. To gain time- based realistic reports, changed data has to be stored in a certain order. Bitemporal approach is one of the temporal data storage approaches.

Various methods may be used to store data in less space. Generating nested database tables is one of these methods. Nested tables compose a semi-structured format in database.

The aim of this study is designing warehouse cubes which stores temporal values in bitemporal form, semi-structured format. Additionally cubes are queried and results are discussed. Outcomes may be used for analysis and data strategy development of companies.

For supplying semi-structured form nested tables and nested types are used. 66 percent less disk space usage is provided. A dimensional database is designed. Queries are run in this dimensional database and results are discussed. Results are explained and supported with diagrams and charts.

Keywords: Data warehouse, bitemporal data model, semi-structured data, Oracle warehouse builder.

YARI- YAPILI, ÇİFT ZAMANLI VERİ AMBARLARININ MODELLENMESİ VE SORGULANMASI

ÖZ

Son yıllarda yaygın olarak kullanılan ve önem kazanan Veri Ambarı, daha hızlı ve efektif raporlama imkanı sağlayan bir sistemdir. Veri ambarında veriler, işlem olmayan depolama alanlarında tutulur. Veri ambarı küpleri aracılığıyla, kolay raporlama ve muhakeme sağlanır. İşletmelerde müşterinin eğilimlerini saptamak ve müşteri memnuniyetini arttırmak önemlidir. Bu noktada veri ambarı çok iyi bir çözümdür.

Veri analizi yaparken eski tarihlere ait bilgilere ihtiyaç duyulur. Zamana dayalı gerçekçi raporlar alabilmek için, değişen veriler geçerlilik sürelerine göre belli bir düzende tutulmalıdır. Çift zamanlı yaklaşım zamansal veri saklama yaklaşımlarından biridir.

Verilerin daha az yer kaplamaları için çeşitli metotlar kullanılabilir. Veritabanındaki tabloları iç içe oluşturmak bu yöntemlerden biridir.

Bu uygulamanın amacı, zamansal değerleri çift zamanlı yapıda tutan, yarı yapılı bir dataset üzerinden veri ambarı küpleri oluşturmak ve oluşan küpleri sorgulamaktır. Ayrıca sorguların sonuçları da incelenmiştir. Sonuçlar veri analizinde ve şirketlerin strateji geliştirmesinde kullanılabilir.

Yarı yapılılığı sağlamak için nested tablolar ve nested typelar kullanılmıştır. Yüzde 66 daha az disk alanı kullanımı sağlanmıştır. Boyutsal bir veritabanı modellenmiştir. Sorgular boyutsal veritabanında çalıştırılmış, sonuçlar tartışılmıştır. Sonuçlar diyagramlar ve grafikler ile açıklanmış ve desteklenmiştir.

Anahtar Sözcükler: Veri ambarı, çift zamanlı veri modeli, yarı yapılı veri, Oracle veri ambarı yaratıcısı.

CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

CHAPTER ONE
INTRODUCTION

1.1 General

Since technology became an indispensable part of people's life, communication and shopping habits revolutionized. People are online nearly 7/24. When surfing on social web sides a lot of data is left unconsciously. A lot of companies collects this data and generates strategies for offering impressive campaigns to their customers.

This big data is processed and rendered into expressive form. There are some techniques to operate this data. The effective applications are prepared by data mining or decision support system algorithms. Data warehousing is a reporting system, to provide gaining more effective reports.

Data warehouse is architecture, corporates prefer for data analysis. Detailed reports are taken from data warehouses. Problems about specific work area can be solved by warehousing reports. Data in data warehouses is stored in a special repository. There are no update, insert, delete actions in warehousing repository; it is only used for querying. Therefore querying performance is high. Warehousing reports are faster and more effective than database reports.

Time variable has major importance on analysis reports of corporate. The change or data in a specific period, in a specific year is asked. Time limitation is one of the most important components in warehousing reports. Data changes by time. Keeping old data is important for temporal deepness. Bitemporal database model is an appropriate approach in order to keep temporal depth.

Semi-structured form is provided by nested tables in databases. Storing tables as nested tables improve performance and ensure using less disk space.

1.2 Purpose

The aim of this study is to design and query a data warehouse to obtain more effective and faster reporting system with using most proper temporal data modeling approach and data storage method.

The most suitable temporal approach is discussed. Bitemporal approach is used in design of data warehouse for providing efficiency. Reports may be taken according to different time intervals that are supported by bitemporal date-time data modeling approach. However bitemporal approach causes redundant growth of data in some cases.

Semi-structured data storage method is used for avoiding unnecessary growth of data. Semi-structured structure provides wasting less disk space, using less database objects and fewer relations. Database operations are simpler in semi-structured form and it provides high performance.

Hereby the most suitable methods are elected and used while designing the database application for ensuring more effective and faster reports. To the best knowledge of the author of this thesis, this study is the first which uses bitemporal approach with semi-structured model in data warehouses for better reporting.

1.3 Organization of Thesis

This thesis consists seven chapters. The organization of chapters as follows.

In chapter 2, data warehousing systems and concepts are explained.

In chapter 3, database model explained. The reason of using bitemporal structure is explained.

In chapter 4, semi-structured data model in databases is mentioned. The reason of using semi-structured form is explained.

In chapter 5, data used in this study is described. Data is changed in order to provide semi-structured and bitemporal form. This chapter contains information about data transformation.

In chapter 6, creating dimensional objects from relational database tables is explained by created warehousing cubes and dimensions. Query and query results of dimensional objects are discussed.

Finally in chapter 7, conclusion of this thesis is reported.

CHAPTER TWO
DATA WAREHOUSING SYSTEMS

2.1 Big Data

The global size of the data is dazzling. Some authorities use ‘Big Data’ term for describing this data.

There are 6 billion mobile subscribers around the world. There are 1.1 billion Facebook users. 400 million tweets are throwing in average every day. 90 percent of the data that exists in the world developed in last two years and it is getting bigger in every second due to the mobile phones and internet. In 2016 61 percent of internet traffic will be provided from the wireless machines and the other 39 percent will be provided by cabled networks. When digital content in online world is 2.7 zeta byte now in 2012, it will reach 7.9 zeta byte in 2016.

What can the trade marks gain by using ‘Big Data’? Firstly attaining the user data, giving meaning to this data, and developing strategies make ‘Big Data’ significant. If we list benefits of this processes:

1. Transparent and useful information can be obtained. ’Big Data’ can give us key information. This information helps companies to make better decisions for the purpose of accessing to the customers.
2. By collecting and storing more digital data, companies can aggregate more correct and detailed information about their purchase and order. Companies use this data in performance increasing actions.
3. ‘Big Data’ gives opportunity of presenting more special product or services to the narrow segmentation of customer.
4. The good analyze of ‘Big Data’ renders complicated data. Mixed data become more meaningful and provide serious convenience to trade marks in having decision.
5. Companies can have an idea about the next product or service by using this data.

Big Analyze necessity has been brought by Big Data. The old analyze methods that examine age; gender and cell number not enough in today’s world. Nowadays, advanced methodologies, new analyzing methods that come from high technology and experience combinations appeared. Before analyzing the existing data, it have to be decided that for which purposes will be used the results. The special solutions can be gathered with respect to the problems.

2.2 What is Data Warehouse

Data warehousing is an activities chain that contains:

- Collecting and sorting out the data in settled or outer systems.
- Preparing data for servicing to decision support systems.
- Storing data in the best way.
- Providing access to data via end user applications
- Finding deterministic data relations.

Data warehousing is a technical, organizational and financial multi-dimensional investment. Customer relationship management (CRM), global e-commerce enterprises, supply chain management (SCM), Enterprise resource planning (ERP) systems, corporate information portals, strategy management informatics are related with Warehousing (Adamson & Venerable, 1998) . Warehousing becomes the basic integral part of the mentioned systems.

Warehousing presents attaining specially prepared data in an easy and quick way. This data is used in management reports, in a variety of queries, decision support systems, manager information systems and data mining applications.

Temporal features are very important in data warehousing. Data warehouse has abundant data collection. This data grows rapidly and has a historical depth. Warehousing queries usually look into long time periods.

Data is extracted from special repositories and used just for reporting. Because of this, reports are getting faster and useful reports can be obtained.

The aim of using data warehouses is:

- Identifying hidden purchasing disposition of customers.
- Focusing on sales analysis and trends.
- Financial analysis.
- Strategic analysis.

2.3 Warehousing Concepts

2.3.1 Metadata

Metadata is data about data. This concept appeared on librarian's area. There are a lot of books in library, namely charged amount of data exist in librarian area. The problem is indexing this data and figuring out the content data of books. Metadata, in other words data about data is appeared from this requirement. Publisher of the book, publishing time, print numbers all of them are data about data.

Database objects also have metadata. Some examples are:

- Number of user in database.
- Number of tables.
- Total records in tables.
- Total utilized disk area.
- Data type of table attributes.

2.3.2 Fact Table

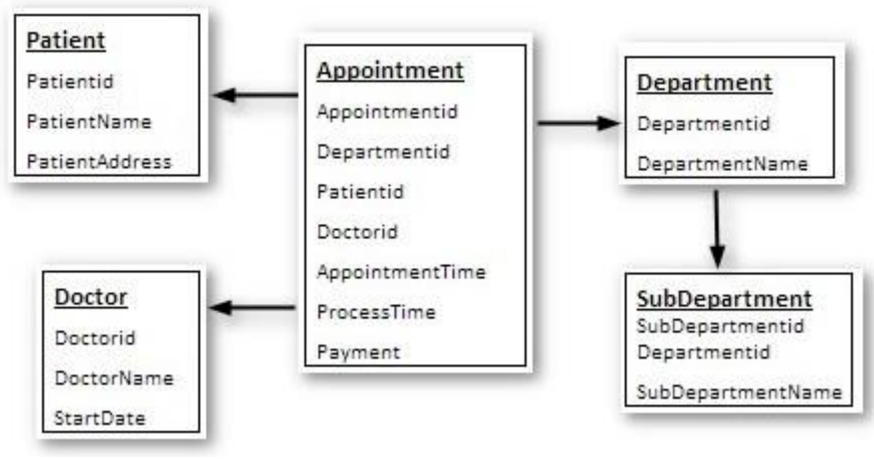


Figure 2.1 Example snowflake schema

Figure 2.1 is used to describe fact table and contains a snowflake schema. Appointment table is the fact table and the other tables connected to this fact table via foreign keys. Departmentid, Patientid, Doctorid columns in Appointment are foreign key attributes. Appointment table comprises data about appointment in hospital. Other attributes of Appointment table are fact attributes. Fact table is the center table in snowflake or star schemas.

2.3.3 Dimension

Dimension tables are the tables around the fact table. Patient, Doctor, Department are dimension tables. Dimension tables are not connected to each other. Dimensions often have one or multiple hierarchies (Inmon, 2002). Hierarchies categorize data. Dimensions have attributes about dimensional value. Multiple dimensions related to fact objects provide to solve questions. Dimensions have hierarchies for aggregating data.

Data Warehouses aims to present flexible data reports. It must support to contribute or compare data along dimensions. Bebel, Eder, Koncilia, Morzy & Wrembel (2004) aimed improving a useful architecture which combines new temporal versions, dimensions in their study.

Ram´ırez & Guerrero (2006) created a model for changing, adding, removing dimensions easily and a query language is presented to manage multidimensional schemas. When a change is made to a multidimensional schema, a new multidimensional database version having a new associated temporal pertinence is created.

2.3.4 Level

As in the Figure 2.1, Department is a dimension. Department is related to Subdepartment table. Every department can have one or more subdepartments. Subdepartment table contains detailed department information. In this example Department is a dimension, department and subdepartment are levels of this dimension.

There can be one or more attributes in a level. Attributes in levels can be categorized by their functionalities.

Level key attributes: The unique instance of the level is key attribute. This attribute is not duplicated. SubDepartmentid column in Subdepartment table is a key attribute of Subdepartment level of Department dimension.

Related attributes: A collection of optional attributes that provide additional information about the instances of the levels that are defined as level key attributes. All related attributes must be functionally determined by the level key attributes. For example, a SubDepartment level might have related SubDepartmentname attribute.

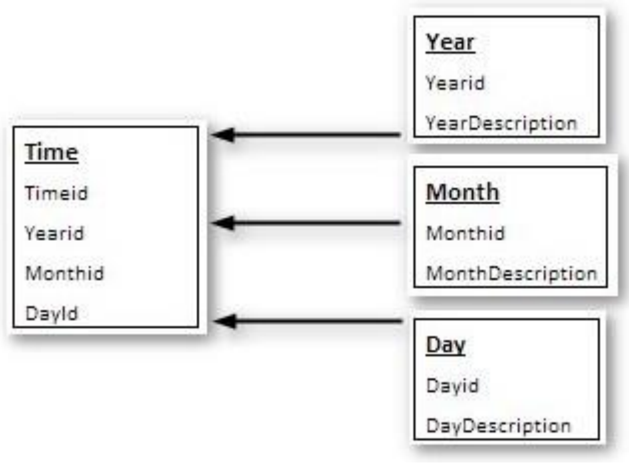


Figure 2.2 Time dimension diagram

There are four tables in Figure 2.2. This view is about time dimension and its levels.

2.3.5 Hierarchy

Hierarchies are the navigations of levels. Hierarchies provide a parent child relationship between levels and help indexing data and measures. In the Figure 2.2 time is a dimension. Year, Month, Day are levels. There is a time hierarchy. Year comprises months, month comprises days in it. Time hierarchy is shown in Figure 2.3.

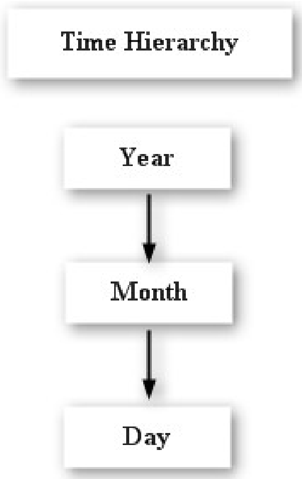


Figure 2.3 Hierarchy structure

2.3.6 Cube

Cubes are the main components of warehousing reports. All previous objects, dimensions, hierarchies, levels are created for generating a cube. Cube is a result for dimensional systems.

Cubes in data warehouses are formed by foreign key attributes which connects to dimension tables and measure attributes. Warehousing cubes not like cube shapes. They do not have to be three dimensioned. Cubes can be designed with one or more dimensions. The fact tables in relational databases are matched with cube tables in dimensional databases. Dimension tables were connected to cube table and they gave a brief of one or more table's information. The primary keys of these summary tables are connected to cube table. The dimensions that are bind to cube is connected each other via cube table. Other attributes in fact table are measure attributes.

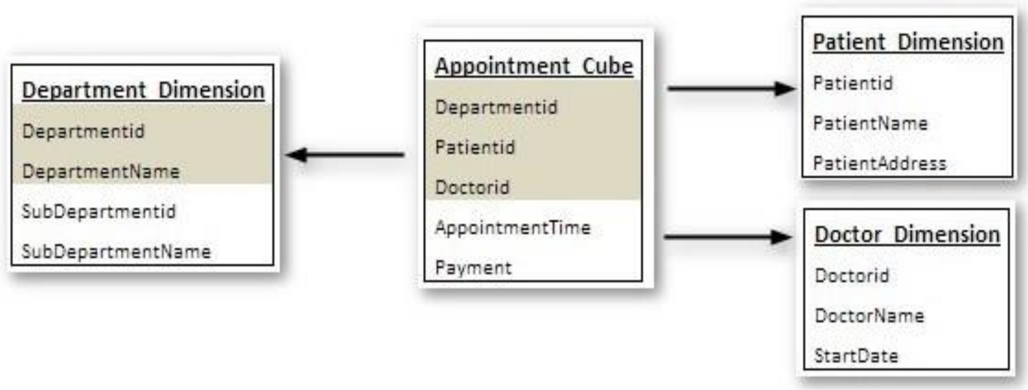


Figure 2.4 The diagram of dimensional objects

In appointment cube table, AppointmentTime and Payment are measurable attributes. Cube table's give result of some questions which have criterions about its dimensions. The results are the measures of the cube. Some questions that the cube is answered can be:

Which doctor gains the maximum payment?

How much many totally gained per department in June of 2013?

What is the earning average of departments according to months of 2013?

Which patient pays the maximum charge on May of 2013?

These questions also can be answered by relational database tables with sql queries but the answer of question comes with more simple queries and quicker with dimensional structure.

2.3.7 Measures

Measure is the measurable attributes of cube. Measure is the result attribute. In Appointment_Cube, AppointmentTime and Payment are measures.

2.4 Warehousing Structures

2.4.1 ETL

ETL (Extract, Transform and Load) layer is the most important part of the any data warehouse application. ETL gets source data from the source system and transform data into new data model and provides to see outcome in warehouse. ETL systems consumes more time than the other operations in business intelligence environment.

Extract: Extract is fetching data from the source system. There are a lot of kinds of different source system, and flat files are used in warehouse systems. Data is collected from many different source systems by helpful systems or code.

Transform: Data in warehouses is fetched from different sources. Transferred data must be converted in form of warehouse structure. Also this data must be cleared and quality of data must be increased.

Load: Load is fetching data from source system and loading it to target system.

2.4.2 OLAP

Olap is a multidimensional query based method that supports multi-dimensional data analysis. Olap (Online Analytical Processing) enables to reach, live, real and prepared data.

Olap technology provides building multi-dimensional data cubes from the data that is stored in relational databases. Users use data for answering complicated problems. Olap provides more supreme performance than the relational databases. In addition, Olap provides opportunity to find answers for complicated queries that is hard or impossible to do in relational databases. Moreover, it is possible to extract future analysis with Olap reports with a good statistical knowledge (Hurtado, Mendelzon & Höfling, 1999).

The data in olap cubes is updated and worked out again in certain hours of the day (generally at night). Totals, averages and the other operations is calculated again with this new live data. When a report is presented via Olap cubes there is no calculation when reporting. All calculated values generally stored in Olap cubes before. The only process is calling the report and showing it.

Olap Data Warehouses are generally stored in separate machines in the companies that have up to date information. Division of weight has good reflect on user. The important criterions are given below:

- Data Propriety: This data has to be designed for the company's requirements. If you want to see; how many cars will you sell next year; the data in your warehouse has to be useful and available for collecting car and sales statistics data. Unnecessary data is a burden for you.
- Data Quality: Data must be clear and in good quality. Imagine that, you will organize a special offer with respect to gender for selling your car. And you are analyzing for future prediction. Gender data is stored as F/M in your sales system while gender data is stored as Female/Male in your customer system. This data do not provide integrity. Data also becomes bad if user enters wrong data instead of F/M or Female/Male. Dirty data has to be cleared.

- Historical Depth: Data warehouse is already set up in this structure. But it is an important concept. It is useful to know this. If you want to guess next three years, you cannot do this with analyzing past one year. For instance, when you wonder the fullness rate of a dam for next year, analyzing 10 years data will give more certain results. It will be more useful if we can analyze statistics in seasonal or monthly sliced time zones.

2.4.3 Data Marts

Data Marts are subsets of data warehouses. While data warehouses provide a complete view for a business problem, data marts provide view for only a part of it.

All employees of a company do not need to analyze all data. Otherwise some users must be permitted to reach to limited area. Data mart represents subset of data warehouse about a specific subject. The information in data marts is not detailed like in data warehouses. So data marts are more understandable and routable.

There are two types of data marts; these are dependent data marts and independent data marts.

- Independent Data Marts: Data in data mart is directly fetched from the operational systems or outer resources. Independent data marts are preferred while there is an analyze necessity in separate departments or branches of a company.
- Dependent Data Marts: Data in data mart is directly fetched from the data warehouse. When there is an analyze necessity for a specific subject of company, dependent data marts are preferred.

Data marts are created, queried, replied faster than the data warehouses. Data marts save performance for analytical querying process.

2.5 Oracle Warehousing Tools

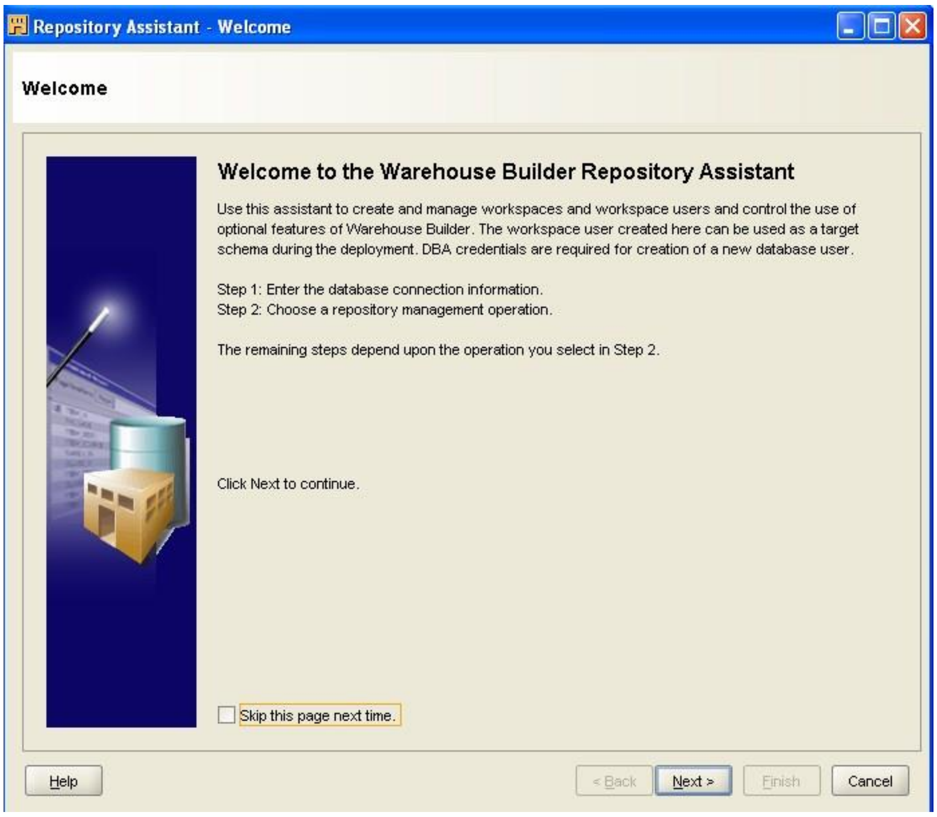
Oracle Warehouse Builder is used to design cubes. Oracle Database 11g R2 edition is free available. Oracle 11g R2 automatically comes with the warehouse builder installed. Oracle 11g R2 Warehouse Builder does not correctly work in every operating system.

2.5.1 Sql Developer

Sql Developer is free Oracle software for managing database. Developer connects to Oracle users via Sql developer. Queries can be written in this platform and developer can manage all objects of database and write queries easily on code editor.

2.5.2 Repository Browser

While generating data warehouse, data is transferred to special storage area. Dimension, cube, measures etc. all elements are created in this special field. Repository Browser is a tool for creating, browsing and managing warehouse repository. The screenshot of Repository Browser welcome page is given in figure 2.5.


Repository Assistant - Welcome

Welcome

Welcome to the Warehouse Builder Repository Assistant

Use this assistant to create and manage workspaces and workspace users and control the use of optional features of Warehouse Builder. The workspace user created here can be used as a target schema during the deployment. DBA credentials are required for creation of a new database user.

Step 1: Enter the database connection information.
Step 2: Choose a repository management operation.

The remaining steps depend upon the operation you select in Step 2.

Click Next to continue.

Skip this page next time.

Figure 2.5 Welcome page of Repository Assistant

2.5.3 Warehouse Builder Design Center

Warehouse builder itself has enough tools for designing dimensions and cubes also mapping data from real database tables to cube or dimension tables stored in warehouse builder repository.

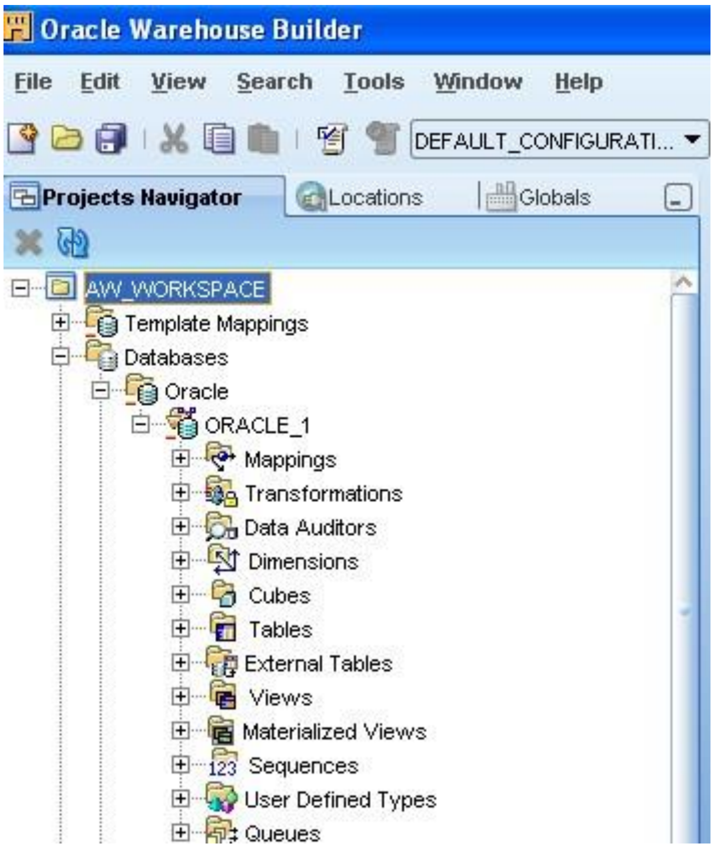


Figure 2.6 General view from Oracle Warehouse Builder. Mapping, transformations, dimensions, cubes... parts are used for creating dimensional objects.

Figure 2.6 is a general view of Oracle Warehouse builder. Database objects can be imported to warehouse builder workspace.

CHAPTER THREE
TEMPORAL APPROACHES

3.1 Modeling Time

Databases are an information world that real world's data is stored. Data may be as string, numeric or logical type and may be set to value or null. Some part of this information is composed by time values. Now divides time into two fragments, past and future. There can be other values like now, that divides time into several sections. Data in databases are important in a specific time-line. Dividing time into several logical sections and modeling time is required for reasoning important outcomes.

3.1.1 Time Point

Time point is a moment in time. When an event occurs, there is a specific realization moment of event in time plane. Status of the object is changed at t1 time point in Figure 3.1. This condition is changed again at t2 time point. Condition is valid from t2 moment to t2 moment. Time points are important in terms of determining state changes.

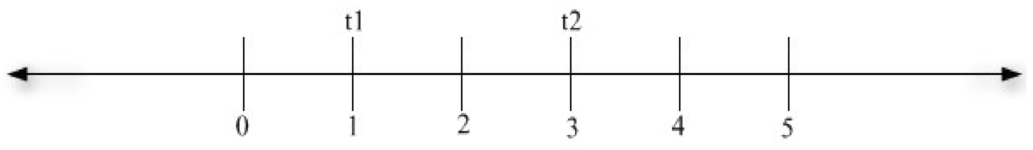


Figure 3.1 Time line

3.1.2 Time Interval

Time interval is a time period that has beginning and end. Some throughputs are always true and time independent. "Turkish Republic is founded on 1923", is an independent knowledge. Some throughputs change. Changing data has effectiveness session. This session is the span between begin and end times. Point interval is the time period between beginning and end time points. The condition in Figure 3.1 is valid in time interval t2- t1.

3.1.3 Temporal Element

A temporal element is the finite union of disjoint time intervals (Gadia, 1988). Temporal element is an element that has time points and time intervals in it. Figure 4.2 contains a temporal element. X, z, t, v are time intervals, y is a time point. These five time elements comprise a temporal element.

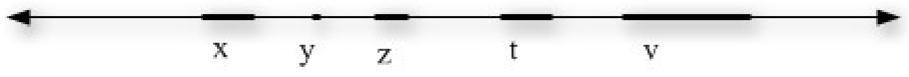


Figure 3.2 A temporal element

3.2 Representing Temporal Data

Temporal data means that the data is defined to have some time-related information associated with them.

3.2.1 User Defined Time

Temporal attributes are stored in DATE, TIME or DATE-TIME types in databases. These attributes are not rendered by DBMS and are called as user defined time. There is no difference of temporal attributes than the attributes in other types (NUMBER, VARCHAR, BYTE…) for DBMS. The meaning of temporal data is significant only for the user.

3.2.2 Valid Time

Valid time indicates the validity period of a fact according to the real world. For instance, list price of a product may be changed in time. If we examine Table 3.1; validity lower bound is pointed as VT_LB, validity upper bound is pointed as VT_UB. List price of “Mountain Bike 1” was 1191.17 between 25.09.2001 and

01.07.2002. Validity period of 1191.17 list price is the time period between two date value. After 01.07.2002 list price of “Mountain Bike 1” is set 1226.9 validity begins and does not end. It is still valid.

Table 3.1 The view of table that has valid time attributes

Product Name	**Color**	**List Price**	**VT_LB**	**VT_UB**
Mountain Bike 1	Black	1191.17	25.09.2001	01.07.2002
Mountain Bike 1	Black	1226.9	01.07.2002	now
Metal Bar 2	Yellow	120.43	18.10.2002	now
Metal Plate	Red	150	30.07.2002	now
Metal Angle	Black	35.89	18.10.2002	now
Touring Rim	Black	22.11	30.07.2002	now

3.2.3 Transaction Time

Transaction time represents the recording time of the values in the database. When a record is added, updated namely a transaction happen, happening begin and end times of this transaction is recorded in database. Transaction time lower bound is pointed as TT_LB; transaction upper bound is pointed as TT_UB in this study. Transaction time is a system-generated value.

3.3 Time-Stamping Data

A timestamp is the date or time value, connected to data value. Multiple time-related attributes may exist about a data. Time-related information is used for recording varied views of temporal truths (Jensen, Soo & Snodgrass, 1994).

3.3.1 Tuple Time Stamping

Tuple is considered to be a row in the table. Temporal attributes of a row is row’s or tuple’s time-stamps. When row is updated, a new row is added to table and temporal attributes are updated. One or more temporal attribute may exist in tuple. Data may be specified as a time-point or time-interval.

In Table 3.2 product name, color, list price, price time columns are existed. Price time attribute is a time- point for each object.

Table 3.2 Tuple is stamped with time points.

Product Name	**Color**	**List Price**	**Price Time**
Mountain Bike 1	Black	1191.17	25.09.2001
Mountain Bike 1	Black	1226.9	01.07.2002
Metal Bar 2	Yellow	120.43	18.10.2002
Metal Plate	Red	150	30.07.2002
Metal Angle	Black	35.89	18.10.2002
Touring Rim	Black	22.11	30.07.2002

In Table 3.3 price start and price end attributes exist for determining temporal validity period. Each list price value in rows has a validity start and end time. Namely each row has a valid time-interval.

Table 3.3 Tuple is stamped with validity time intervals.

Product Name	**Color**	**List Price**	**Price Start**	**Price End**
Mountain Bike 1	Black	1191.17	25.09.2001	01.07.2002
Mountain Bike 1	Black	1226.9	01.07.2002	now
Metal Bar 2	Yellow	120.43	18.10.2002	now
Metal Plate	Red	150	30.07.2002	now
Metal Angle	Black	35.89	18.10.2002	now
Touring Rim	Black	22.11	30.07.2002	now

Transaction times may be included into tuple alongside the valid time attributes. In Table 3.4 from and to date typed attributes are added for specifying transaction interval.

Table 3.4 Tuple is stamped with validity and transaction time intervals.

Product Name	**Color**	**List Price**	**Price Start**	**Price End**	**From**	**To**
Mountain Bike 1	Black	1191.17	25.09.2001	01.07.2002	26.09.2002	26.09.2002
Mountain Bike 1	Black	1226.9	01.07.2002	now	02.07.2002	02.07.2002
Metal Bar 2	Yellow	120.43	18.10.2002	now	20.10.2002	20.10.2002
Metal Plate	Red	150	30.07.2002	now	01.08.2002	01.08.2002
Metal Angle	Black	35.89	18.10.2002	now	20.10.2002	20.10.2002
Touring Rim	Black	22.11	30.07.2002	now	01.08.2002	01.08.2002

As a result tuple time stamping is adding temporal aspects to row of tables.

3.3.2 Attribute Time Stamping

Attribute is used to describe the column in database. Attributes is defined in a certain or user defined data type. In table 4.4 all temporal attributes are about the change of list price attribute. Namely all time-values in table are one attribute’s time stamps. Attribute time stamping requires nested relations. Table 3.5 has product name, color, list price and cost attributes. List price and cost attributes are nested attributes. These attributes have value and temporal data in it. Validity of list price “Mountain Bike 1” is 1191.17 between 25.09.2001 and 01.07.2002. It is 1226.9 after 01.07.2002. Cost of “Mountain Bike 1” is 605 between 25.09.2001 and 01.01.2003. It is 750 after 01.01.2003.

Table 3.5 List price and cost attributes are time stamped

Product Name	**Color**	**List Price**	**Cost**
Mountain Bike 1	Black	{<[25.09.2001,01.07.2002],1191.17>, <[01.07.2002,now],1226.9>}	{<[25.09.2001,01.01.2003],605>, <[01.01.2003,now],750>} }
Metal Bar 2	Yellow	{<[18.10.2002,now],120.43>}	{<[18.10.2002,now],100>}
Metal Plate	Red	{<[30.07.2002,now],150>}	{<[30.07.2002,now],122.11>}
Metal Angle	Black	{<[18.10.2002,now],35.89>}	{<[18.10.2002,now],22.11>}
Touring Rim	Black	{<[30.07.2002,now],22.11>}	{<[30.07.2002,now],18.11>}

3.4 Temporal Databases

Temporal databases are the databases which have temporal data modeled.

In this section some temporal database types are mentioned. These are: Snapshot databases, historical databases, transactional databases and bitemporal databases. The most suitable database type for Data Warehousing is tried to figure out.

3.4.1 Snapshot Databases

Snapshot database is a copy or image of database at that moment. Snapshots contain committed data and transactions. If there are some uncommitted transactions in database, these changes are not existed in snapshot.

The Advantages of Snapshot Databases:

1. In some cases the report of a particular time is needed, taking snapshots is an excellent feature for these conditions. Through this feature steady data is read and reported.
2. Snapshot provides keeping historical data for creating report.

3. Snapshot does not generate physical copy that is why it can be used as a replica.
4. Backing up is quicker than the database.

The Disadvantages of Snapshot Databases:

1. For the databases that require performance, snapshotting is overcharge. It copies every changing page on database to disk. This overcharges disk.
2. If main database collapses, snapshot cannot be reached.
3. Snapshot is not an effective back-up restore process.
4. Snapshot database is read only. No data changes. If it is prompted to change, snapshot is taken again.
5. Snapshot and database have to stand on the same instance.
6. It is not recommended for the databases that too much transaction.
7. Snapshot process supports only NTFS file system.

3.4.2 Historical Databases

Historical databases contain cases over the valid time line. Historical database consist historical data storage structure in it. Historical time values change according to historic knowledge. The current time of historical database is always now. It may not be set to a past time. In this regard, historical databases are parallel with snapshot databases. If an error acquired, and an update process has to be applied; the previous values are discarded and lost in historical databases.

3.4.3 Transaction Databases

Insert, update, create, select operations are transactions in databases. When a query run, the alterations are not saved in the first moment, for saving changes transactions must be committed. Or if someone is prompted to undo query, transaction may be rolled back. The revocable activities are stored in transaction databases. Rolling back the transactions in other words rolling the database to a point in the past is probable in transaction databases.

3.4.3 Bitemporal Databases

Historical databases design temporal information of objects and do not keep system modifications. Transaction databases do not form the real changes of system. If the reality is formed completely, historical and transactional databases must be combined (also valid and transaction time components must be existed). Bitemporal databases have both valid time and transaction time. In this manner, bitemporal databases demonstrate real world’s data according to real time (Koncilia, 2003).

Sample table is shown in the Table 3.6. Validity period of product's list price is the time period between VT_LB and VT_UB. If a record is valid in current time VT_UB value is set to null or now. In this structure all previous status and the current status are stored with their validity period. Beside this, all the previous and the current list price values have TT_LB and TT_UB. The transaction time for “Mountain Bike 1” list priced 1191.17 is 26.09.2002. The last transaction applied on this value is insert, when list price of “Mountain Bike 1” updated another row is added into table and VT_UB attribute of first record is updated. If so many updates acquire, table can be expanded vertically and consume too much disk space.

Table 3.6 The view of example table after updating according to bitemporal approach

Product Name	**Color**	**List Price**	**VT_LB**	**VT_UB**	**TT_LB**	**TT_UB**
Mountain Bike 1	Black	1191.17	25.09.2001	01.07.2002	26.09.2002	26.09.2002
Mountain Bike 1	Black	1226.9	01.07.2002	now	02.07.2002	02.07.2002
Metal Bar 2	Yellow	120.43	18.10.2002	now	20.10.2002	20.10.2002
Metal Plate	Red	150	30.07.2002	now	01.08.2002	01.08.2002
Metal Angle	Black	35.89	18.10.2002	now	20.10.2002	20.10.2002

CHAPTER FOUR
SEMI STRUCTURED DATA MODEL

4.1 Semi-Structured Data

A lot of techniques are improved for storing data. Information may be stored in excel tables or relational databases. However some information has hierarchy inside. Organizing and creating a suitable form according to some tags is required for storing data. The necessity of using semi-structured data is increased by the expansion of internet and application variety.

4.2 Semi-Structured Form

For avoiding overmuch expansion of table, using semi-structured table forms is beneficial. Change in sampled table is occurred in list price column. VT_LB(Valid time lower bound), VT_UB(Valid time upper bound), TT_LB(Transaction time lower bound), TT_UB(Transaction time upper bound) attributes also for storing temporal depth of list price column. In other words temporal attributes in this table is about one attribute, they are attribute's timestamps.

List price and its temporal attributes represent a structure or a new type. This type is not like regular database types e.g. number, varchar, date etc. Table can be converted to semi-structured nested table as shown in Table 4.1. List price column now is a table inside table, that has five columns Value, VT_LB, VT_UB, TT_LB, TT_UB. When update of list price occurred, new row is added to specified person's list price column's nested table. Only nested table expands on updates, changing attribute and its temporal components grow. Consequently, semi- structured form prevents unnecessary grow of data.

If Mountain Bike 1’s list price is determined as 1300.00 on 05.10.2003 and this record is committed on 06.10.2003 (Transaction time) then the change on table is shown in Table 4.2.

Table 4.1 Nested table with bitemporal attributes

Product Name	**Color**	**List Price**
Mountain Bike 1	Black	{<1191.17,[25.09.2001,01.07.2002],[01.07.2002,01.07.2002] >, <1226.9, [01.07.2002, now], [02.07.2002, 02.07.2002]>}
Metal Bar 2	Yellow	{<120.43 , [18.10.2002,now], [20.10.2002,20.10.2002]>}
Metal Plate	Red	{<150 , [30.07.2002,now], [01.08.2002,01.08.2002]>}
Metal Angle	Black	{<35.89 , [18.10.2002,now], [20.10.2002, 20.10.2002]>}
Touring Rim	Black	{<22.11 , [30.07.2002,now], [01.08.2002, 01.08.2002]>}

Table 4.2 The view of nested table after update transaction

Product Name	**Color**	**List Price**
Mountain Bike 1	Black	{<1191.17,[25.09.2001,01.07.2002],[01.07.2002,01.07.2002] >, <1226.9, [01.07.2002,now], [02.07.2002, 02.07.2002]>, <1300, [05.10.2003, now], [06.10.2003, 06.10.2003]>}
Metal Bar 2	Yellow	{<120.43 , [18.10.2002,now], [20.10.2002,20.10.2002]>}
Metal Plate	Red	{<150 , [30.07.2002,now], [01.08.2002,01.08.2002]>}
Metal Angle	Black	{<35.89 , [18.10.2002,now], [20.10.2002, 20.10.2002]>}
Touring Rim	Black	{<22.11 , [30.07.2002,now], [01.08.2002, 01.08.2002]>}

According to the data in Table 4.1, if list price value changes new situation will be as in Table 4.2. A similar study is worked out by Malinowski & Zimanyi (2006). They designed Multidimensional ER model that use Valid and transaction time together. This study tries to find the answer of, “If one row changes, how it effects to other rows and how it effects to relationships between them?” question.

The disadvantage of bitemporal approach is the risk of expanding too much vertically. With semi-structured form of table, redundant expand risk is prevented.

Combi, Oliboni & Pozzi (2009) deal with temporal semi structured data warehouses, their modeling and querying. They proposed a graph-based data model to represent semi structured temporal data warehouses and a query language to suitably retrieve the considered information. The data is stored as xml document. And also a query language is generated to manage and order data.

4.3 Nested Types

Nested types are used to define special table column types. These columns contain multiple attributes. If there is a necessity for storing more than one attributes in a column, nested types are created. An example creation script of nested type is shown in Table 4.3. TAX type has five attributes in it. TT_UB, TT_LB, VT_LB, VT_LB are DATE attributes, VALUE is NUMBER attribute.

Table 4.3 Creation script of nested type

```
CREATE TYPE TAX AS OBJECT (
      TT_UB DATE,
      TT_UB DATE,
      VT_LB DATE,
      VT_UB DATE,
      VALUE  NUMBER  );
```

4.4 Nested Tables

NESTED TABLE is an Oracle data type used to support columns containing multivalued attributes. In this case, columns can hold an entire sub-table. A sample for usage nested types when creating tables are shown in Table 4.4. TAX attribute's data type is TAX. Also nested tables must be specified as shown in bottom of statement.

Table 4.4 Creation Script of Nested table

```
CREATE TYPE Tax AS TABLE OF Tax;
CREATE TABLE PRODUCT
  (   PRODUCTID NUMBER  ,
      NAME VARCHAR2(50 BYTE)  ,
      TAX TAX,
      SIZEE VARCHAR2(5 BYTE),
      WEIGHT NUMBER,
      DAYSTOMANUFACTURE NUMBER  ,
      PRODUCTLINE CHAR(2 BYTE),
      CLASSS CHAR(2 BYTE),
      STYLEE CHAR(2 BYTE),
      PRODUCTMODELID NUMBER,
      SUBCATEGORYID NUMBER)
 NESTED TABLE TAX STORE AS TAX_TABLE2;
```

4.5 XML Databases

Xml is Extensible Markup Language and a semi-structured data storage method that created after SGML. Structured Generalized Markup Language (SGML) is a XML like technology that exists before XML. It is evolved before 1980, it become an ISO standard in 1986. HTML technology is started to develop in 1990. The improvement of XML started in 1996 and recommended by World Wide Web Consortium by 1998. The developers of XML took the advantages of SGML, combined HTML experiences.

Xml database is a software system that gives permission to store data in XML format. XML is used for organizing and modeling data with customizable tags and one of the basic storing methods of data. It entailed using special data structures and database systems on data exchange. It provides flexibility on storage.

XML databases are easy accessible, thus it is commonly used. XML databases are separated into three categories. These are Native XML (NXD), XML Enabled Database (XEDB) and Hybrid XML Database (HXD). These types are used to store different sort of data.

1. Native XML Database (NXD): NXD describes a pattern for XML documents. It stores and regulates documents with respect that pattern. XML info set, Xpath data model are some examples of NXD. XML documents are used as elementary unit of logical storage in NXD. The similar relationship is seen between relational databases and rows. NXD databases can be established on any kind of database, it does not need a special format.
2. XML Enabled Database (XEDB): XEDB is an XML mapping stage added database. XML mapping systems are added database applications. XML solutions of Sql Server, Oracle, Mysql or third party applications are in this class.
3. Hybrid XML Database (HXD): HXD can be acted as NXD or XEDB with respect to the requirements of the application.

XML databases are sometimes the best solution for storing data. In some situations XML databases predominate on relational database systems. They are often used in:

- Information services of companies.
- Membership databases.
- Product catalogs.
- Hospital Database applications.
- Business document exchange applications.

CHAPTER FIVE
BENCHMARKING AND DATA TRANSFER

5.1 Importance of Benchmarking

One of the major problems of academic studies about computer science is finding data. In terms of the application results' reliability, real data is preferred to use. Random generated imaginary data is not generally used to prove scientific realities. Importance of using real data, changes according to subject of study. In decision support systems or data mining applications real data usage is prominent. In these systems exactness of system is realized from data.

Besides this; the data which is verified and specially created for scientific or educational studies is preferable for scientific applications.

Thereupon; ready, authentic data is explored. Microsoft Adventure Works database is selected among many datasets and sample databases.

5.2 The Data Used in This Study

Adventure Works is a relational database established for Sql Server. It has five modules in it. These are Sales, Purchasing, Person, Production, Human Resources. Adventure works database firstly installed on Sql server. Production module of the database transferred to Oracle Database 11g R2.

The fact table of Production module is Product table, depicted in Figure 5.1, Figure 5.2, Figure 5.3 and Figure 5.4.

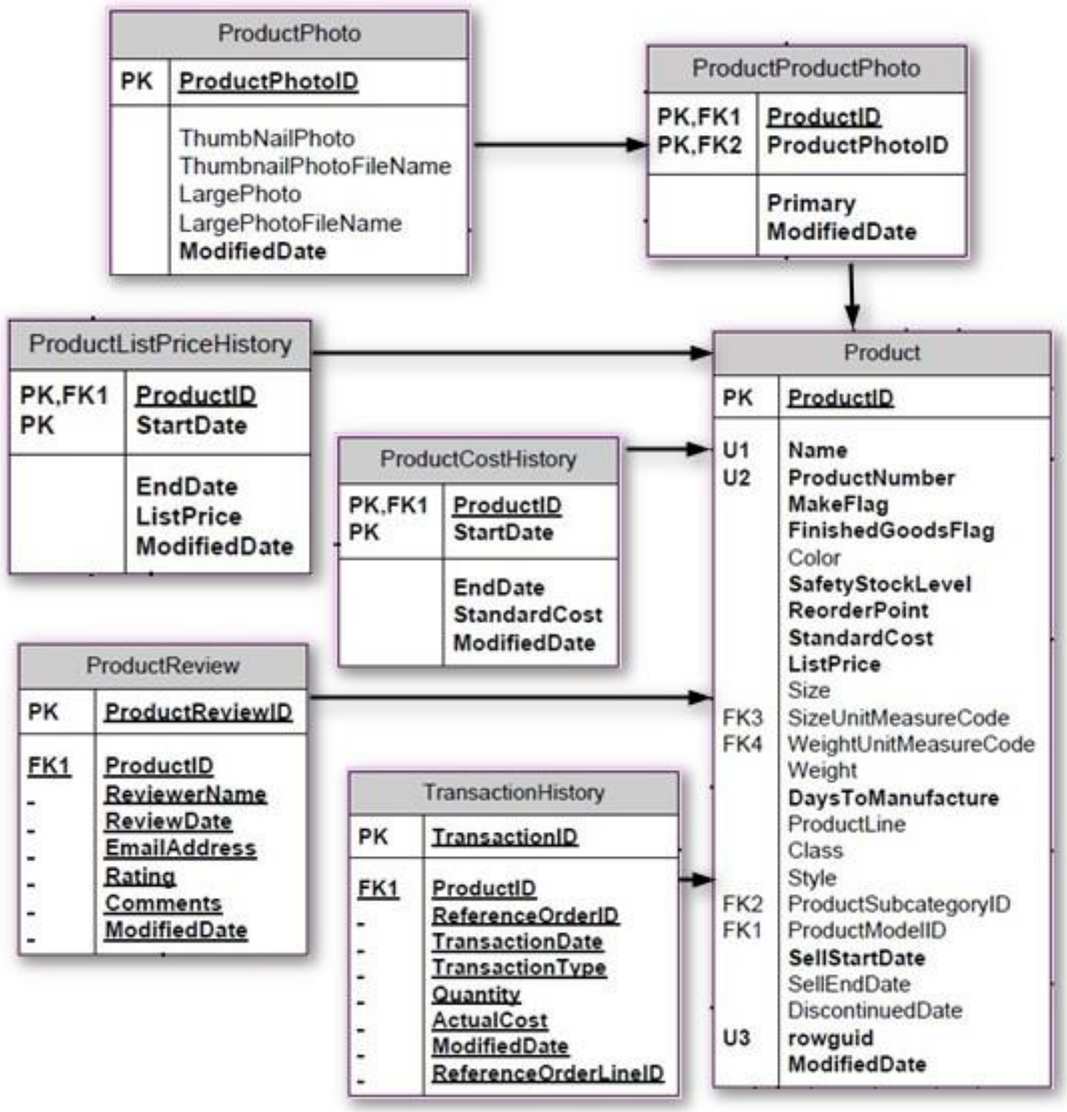


Figure 5.1 Adventure Works relational database diagram, 1-1

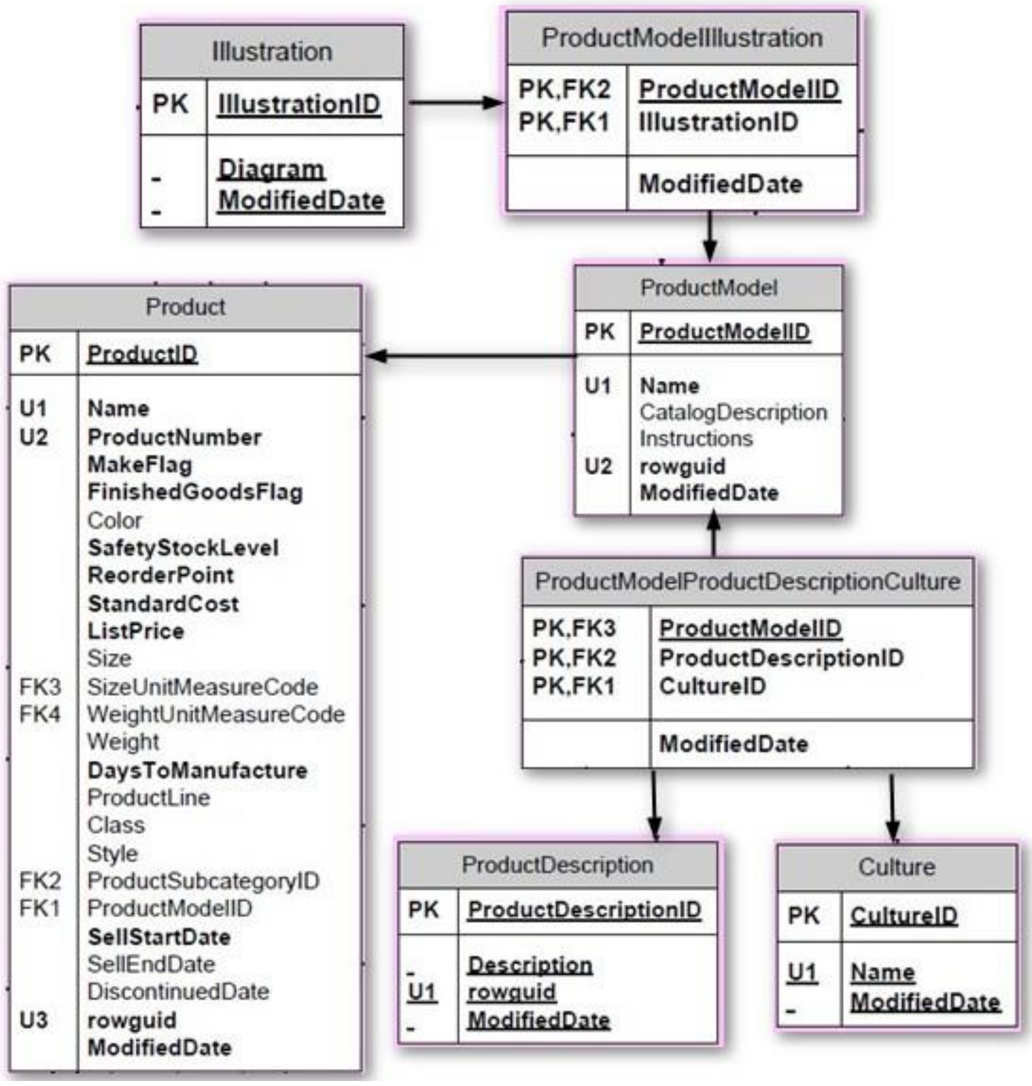


Figure 5.2 Adventure Works relational database diagram, 1-2

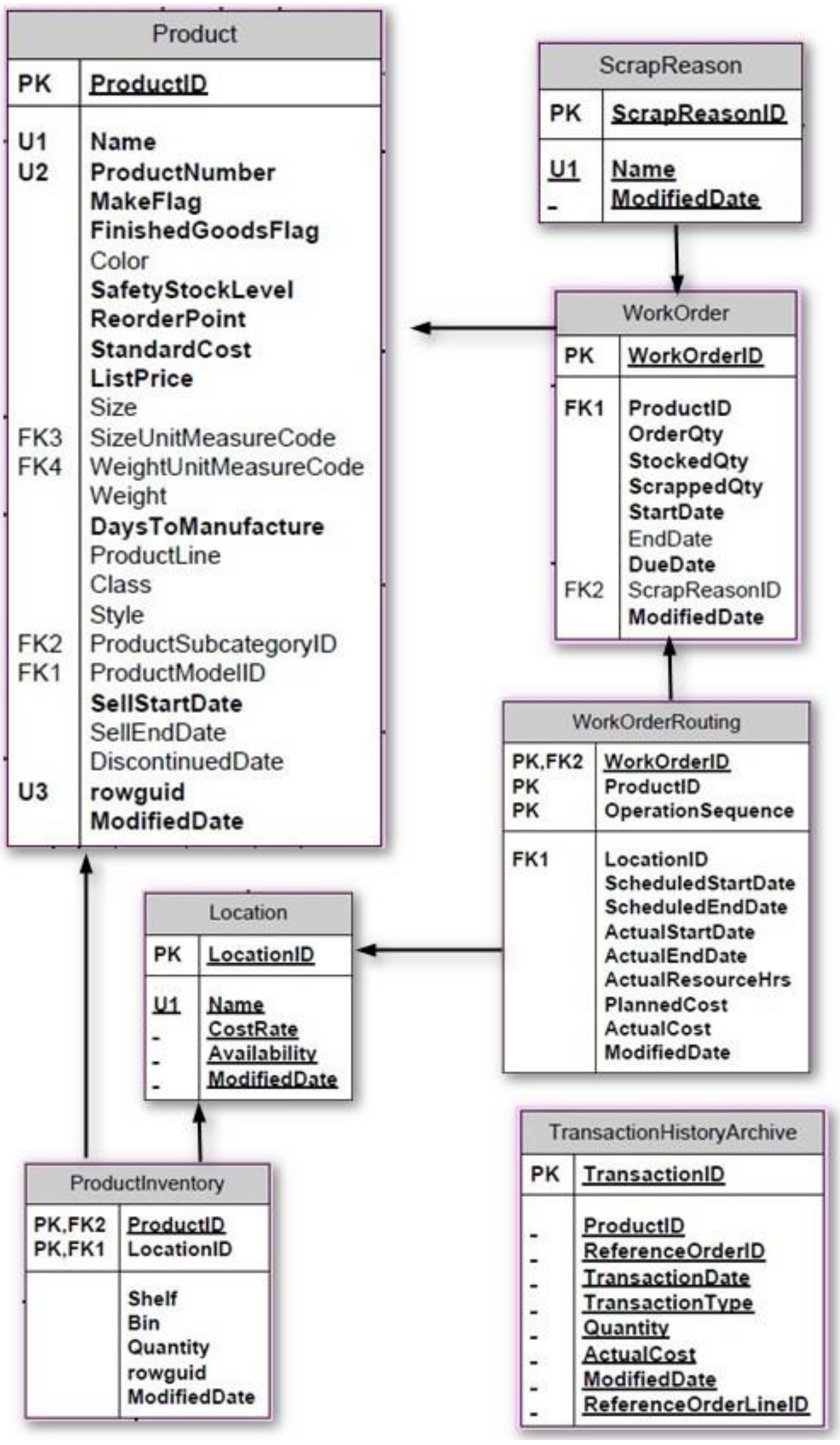


Figure 5.3 Adventure Works relational database diagram, 1-3

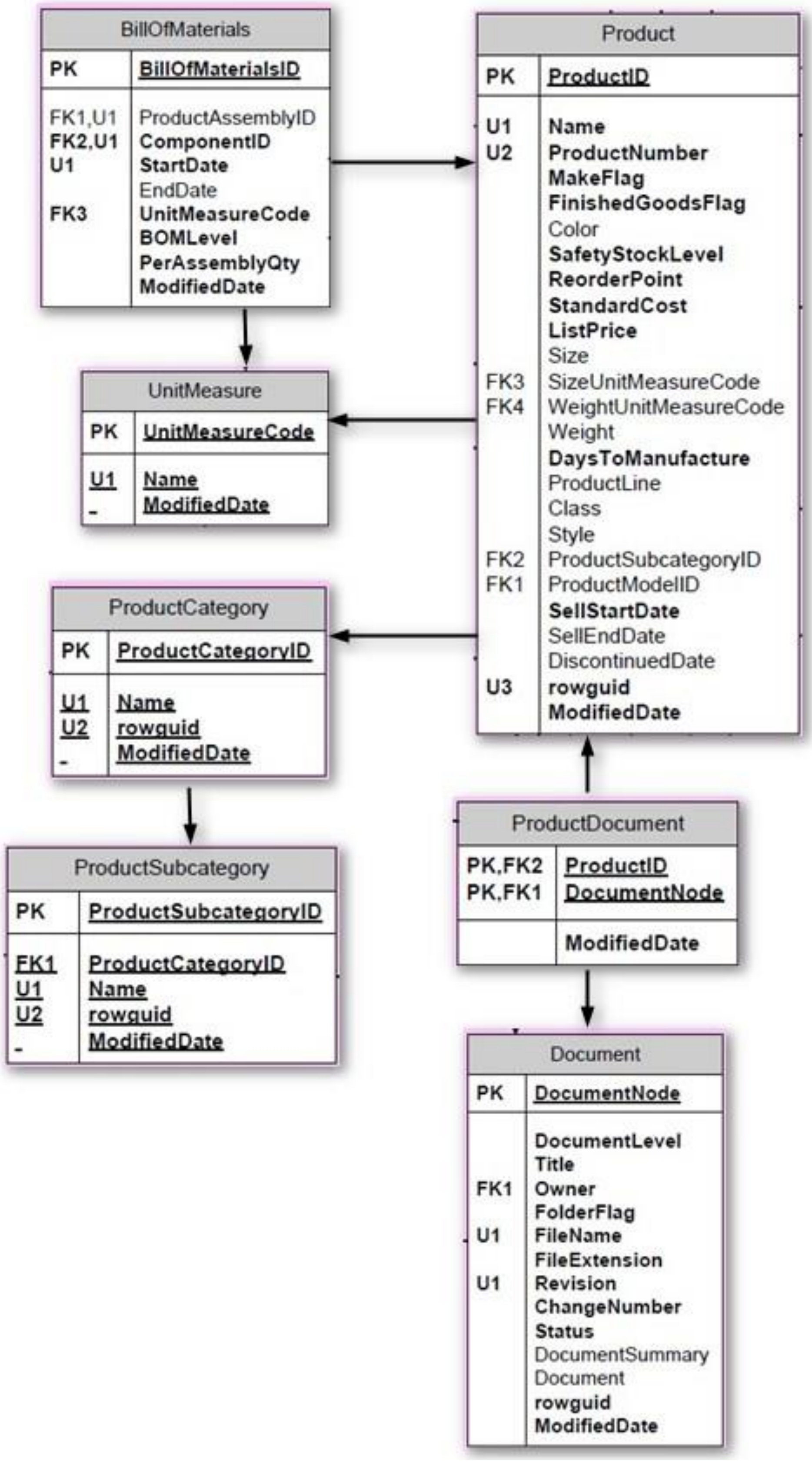


Figure 5.4 Adventure Works relational database diagram, 1-4

5.3 Data Transfer

The data and tables created in data warehouse fetched from AdventureWorks database. The tables in Sql server AdventureWorks and Oracle is not exactly same. Some tables rendered into nested structure. While there were 25 tables in Sql Server AdventureWorks' database product module, there are 15 tables in Oracle AdventureWorks user product module. As a result; necessities of joining two or more tables' data as one insert statement occurred while transferring data. Relational database design of Oracle product module is shown in the Figure 5.5 and Figure 5.6.

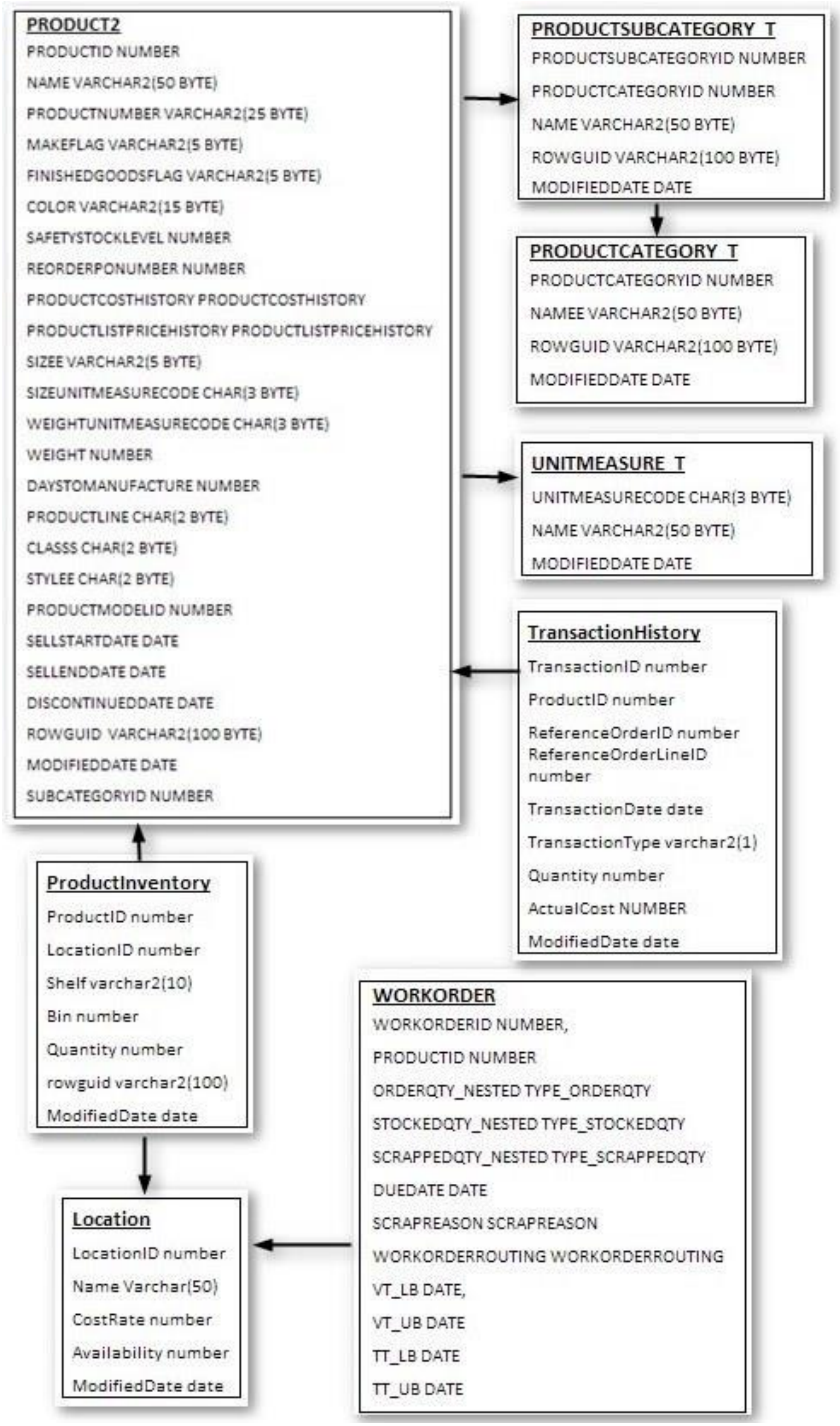


Figure 5.5 Oracle Adventure Works, relational, bitemporal and semi-structured database diagram Part 1

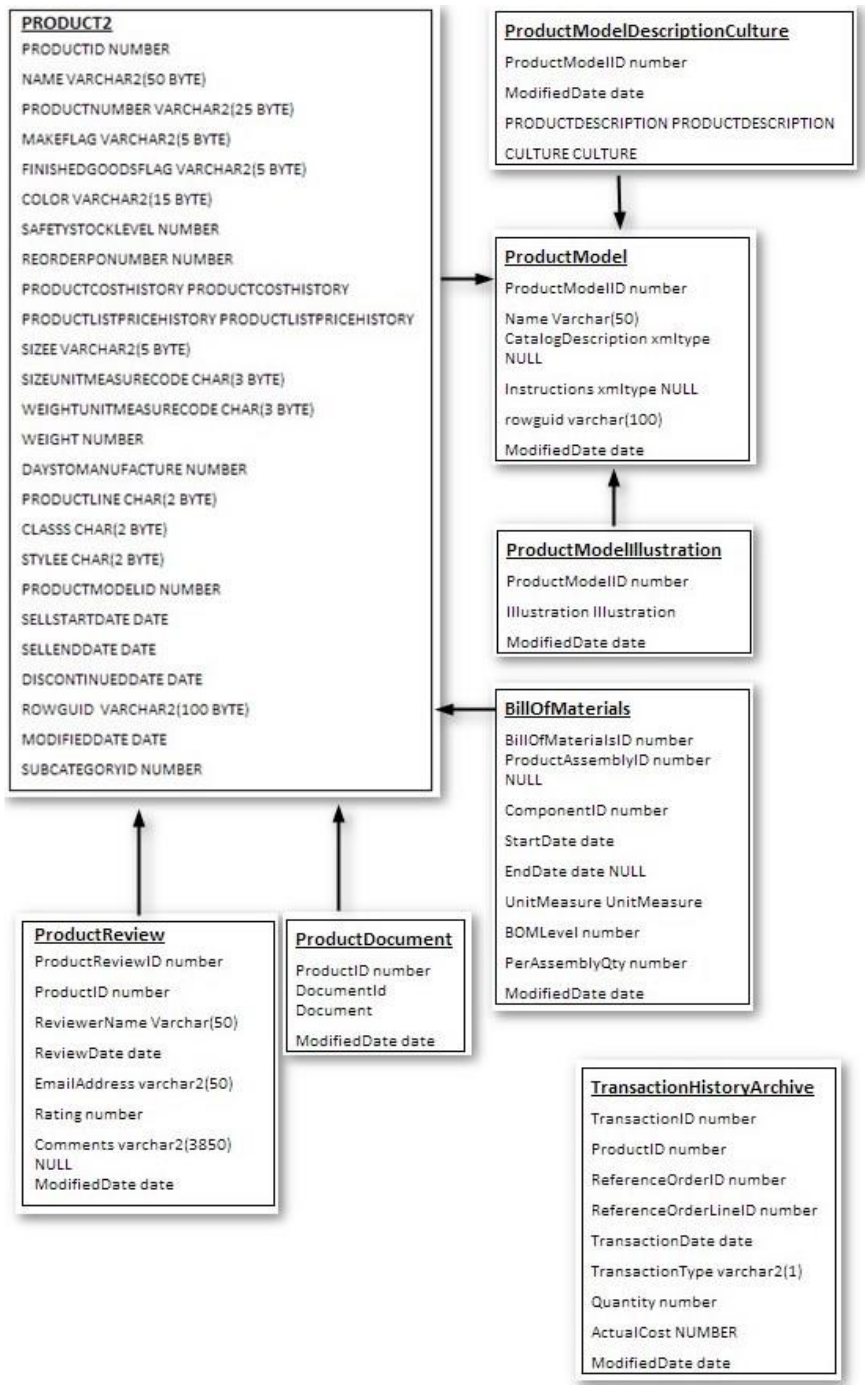


Figure 5.6 Oracle Adventure Works, relational, bitemporal and semi-structured database diagram Part 2

5.3.1 Creating Tables

While transferring tables, creation script of tables extracted from Sql Server and organized as available on Oracle database. One of these translations is shown in Table 5.1. This example does not contain nested type in it.

Table 5.1 Comparison of regular table creation scripts

Sql Server Creation Script	Oracle Creation Script
CREATE TABLE [Production].[Location]([LocationID] [smallint] IDENTITY(1,1) NOT NULL, [Name] [dbo].[Name] NOT NULL, [CostRate] [smallmoney] NOT NULL, [Availability] [decimal](8, 2) NOT NULL, [ModifiedDate] [datetime] NOT NULL);	CREATE TABLE ADVENTUREWORKS2.LOCATION (LOCATIONID NUMBER, NAME VARCHAR2(50 BYTE), COSTRATE NUMBER, AVAILABILITY NUMBER, MODIFIEDDATE DATE);

ProductCostHistory, ProductListPriceHistory, Product tables's creation scripts are given on left side of the Table 5.2 and Table 5.3. ProductCostHistory, ProductListPriceHistory tables are embedded as nested table into Product table in Oracle. Creation scripts of nested types and tables are given on the right side of the Table 5.2.

ProductCostHistory table have StartDate, EndDate, ModifiedDate as date attributes. If these date values are thought as bitemporal attributes, StartDate is VT_LB (Valid Time Lower Bound), EndDate is VT_UB (Valid Time Upper Bound). Transaction happens at a time, ModifiedDate is TT_LB(Transaction Time Lower Bound) and TT_UB(Transaction Time Upper Bound). Namely TT_LB and TT_UB supposed to be same.

Table 5.2 Comparison of nested table and regular tables creation scripts part 1

Sql Server Creation Script	Oracle Creation Script
CREATE TABLE [Production].[ProductCostHistory]([ProductID] [int] NOT NULL, [StartDate] [datetime] NOT NULL, [EndDate] [datetime] NULL, [StandardCost] [money] NOT NULL, [ModifiedDate] [datetime] NOT NULL); CREATE TABLE [Production].[ProductListPriceHistory]([ProductID] [int] NOT NULL, [StartDate] [datetime] NOT NULL, [EndDate] [datetime] NULL, [ListPrice] [money] NOT NULL, [ModifiedDate] [datetime] NOT NULL); CREATE TABLE [Production].[Product]([ProductID] [int] IDENTITY(1,1) NOT NULL, [Name] [dbo].[Name] NOT NULL, [ProductNumber] [nvarchar](25) NOT NULL, [MakeFlag] [dbo].[Flag] NOT NULL, [FinishedGoodsFlag] [dbo].[Flag] NOT NULL, [Color] [nvarchar](15) NULL, [SafetyStockLevel] [smallint] NOT NULL, [ReorderPoint] [smallint] NOT NULL,	CREATE OR REPLACE TYPE BT_NUMBER AS OBJECT (TT_LB DATE, TT_UB DATE, VT_LB DATE, VT_UB DATE, VALUE NUMBER); CREATE OR REPLACE TYPE PRODUCTCOSTHISTORY AS TABLE OF BT_NUMBER; CREATE OR REPLACE TYPE PRODUCTLISTPRICEHISTORY AS TABLE OF BT_NUMBER; CREATE TABLE ADVENTUREWORKS2.PRODUCT (PRODUCTID NUMBER NOT NULL ENABLE, NAME VARCHAR2(50 BYTE) NOT NULL ENABLE, PRODUCTNUMBER VARCHAR2(25 BYTE) NOT NULL ENABLE, MAKEFLAG VARCHAR2(5 BYTE) NOT NULL ENABLE, FINISHEDGOODSFLAG VARCHAR2(5 BYTE) NOT NULL ENABLE, COLOR VARCHAR2(15 BYTE), SAFETYSTOCKLEVEL NUMBER NOT NULL ENABLE, REORDERPONUMBER NUMBER NOT NULL ENABLE,

Table 5.3 Comparison of nested table and regular tables creation scripts part 2

Sql Server Creation Script	Oracle Creation Script
[StandardCost] [money] NOT NULL, [ListPrice] [money] NOT NULL, [Size] [nvarchar](5) NULL, [SizeUnitMeasureCode] [nchar](3) NULL, [WeightUnitMeasureCode] [nchar](3) NULL, [Weight] [decimal](8, 2) NULL, [DaysToManufacture] [int] NOT NULL, [ProductLine] [nchar](2) NULL, [Class] [nchar](2) NULL, [Style] [nchar](2) NULL, [ProductSubcategoryID] [int] NULL, [ProductModelID] [int] NULL, [SellStartDate] [datetime] NOT NULL, [SellEndDate] [datetime] NULL, [DiscontinuedDate] [datetime] NULL, [rowguid] [uniqueidentifier] ROWGUIDCOL NOT NULL, [ModifiedDate] [datetime] NOT NULL);	PRODUCTCOSTHISTORY ADVENTUREWORKS2.PRODUCTCOSTHISTORY , PRODUCTLISTPRICEHISTORY ADVENTUREWORKS2.PRODUCTLISTPRICEHISTORY , SIZEE VARCHAR2(5 BYTE), SIZEUNITMEASURECODE CHAR(3 BYTE), WEIGHTUNITMEASURECODE CHAR(3 BYTE), WEIGHT NUMBER, DAYSTOMANUFACTURE NUMBER NOT NULL ENABLE, PRODUCTLINE CHAR(2 BYTE), CLASSS CHAR(2 BYTE), STYLEE CHAR(2 BYTE), PRODUCTMODELID NUMBER, SELLSTARTDATE DATE NOT NULL ENABLE, SELLENDDATE DATE, DISCONTINUEDDATE DATE, ROWGUID_UNIQUEIDENTIFIER VARCHAR2(100 BYTE) NOT NULL ENABLE, MODIFIEDDATE DATE NOT NULL ENABLE, SUBCATEGORYID NUMBER) NESTED TABLE PRODUCTCOSTHISTORY STORE AS PRODUCTCOSTHISTORY_TABLE2, NESTED TABLE PRODUCTLISTPRICEHISTORY STORE AS PRODUCTLISTPRICEHISTORY_TABLE2;

5.3.2 Inserting Data

In order to insert data to Oracle nested Product table, combining three tables' data into one insertion script is required. There can be multiple ProductListPriceHistory or ProductCostHistory record for one product record. Insertion script is organized via Microsoft excel by taking into account any condition. The insertion script which adds record to nested Product table is given in Table 5.4.

Table 5.4 Example insert statement to nested product table

```
    Insert into PRODUCT (PRODUCTID,NAME,PRODUCTNUMBER,
    MAKEFLAG,FINISHEDGOODSFLAG,COLOR,SAFETYSTOCKLEVEL,
    REORDERPONUMBER,PRODUCTCOSTHISTORY,PRODUCTLISTPRICEHISTORY,
    SIZEE,SIZEUNITMEASURECODE,WEIGHTUNITMEASURECODE,WEIGHT,
    DAYSTOMANUFACTURE,PRODUCTLINE,CLASSS,STYLEE,PRODUCTMODELID,
    SELLSTARTDATE,SELLENDDATE,DISCONTINUEDDATE,
ROWGUID_UNIQUEIDENTIFIER, MODIFIEDDATE,SUBCATEGORYID) values (759,'Road-
650 Red, 58','BK-R50R-58','1','1','Red',100,75,
    ADVENTUREWORKS2.BT_NUMBER(ADVENTUREWORKS2.BT_NUMBER(2001-07-01
00:00:00.0,2002-06-30 00:00:00.0, 2003-10-06 00:00:00.0,2003-10-06 00:00:00.0,4131463),
    ADVENTUREWORKS2.BT_NUMBER(2002-07-01 00:00:00.0,2003-06-30
    00:00:00.0,2003-10-06 00:00:00.0,2003-10-06 00:00:00.0,4867066)),
    ADVENTUREWORKS2.BT_NUMBER(ADVENTUREWORKS2.BT_NUMBER(2001-01-07
00:00:00.0,
    2002-06-30 00:00:00.0,2002-06-30 00:00:00.0,2002-06-30 00:00:00.0,699.0982),
    ADVENTUREWORKS2.BT_NUMBER(2002-01-07 00:00:00.0,2003-06-30 00:00:00.0,2003-
06-30 00:00:00.0,2003-06-30 00:00:00.0,782.99)),
    '58','CM ','LB ',19.79,4,'R ','L ','U ',30,     to_timestamp('01/07/2001','DD/MM/YYYY'),
    to_timestamp('30/06/2003','DD/MM/YYYY'),
    null,'6711d6bc-664f-4890-9f69-
af1de321d055',to_timestamp('11/03/2004','DD/MM/YYYY'),17);
```

CHAPTER SIX
IMPLEMENTATION AND RESULTS

Data warehousing data is stored in an independent repository. Firstly a new repository is created by Oracle Repository Browser. Then AdventureWorks database objects imported into warehouse builder workspace. Cubes and dimensions are designed with respect to dimensional system requirements.

6.1 Cube1- CubeProductListPrice

6.1.1 Design

Small part of product module is shown in Figure 6.1. These tables comprise a snowflake schema. Arrows show foreign keys between tables. Product table is the fact table with ProductModel, ProductModelDescriptionCulture, Unitmeasure_t, ProductSubCategory_t, ProductCategory_t are dimension tables. Model dimension has three dimension levels ProductModel and Description and Culture, UnitMeasure dimension has one level, Subcategory dimension has two levels ProductSubCategory and ProductCategory.

The primary key in each primary dimension table (Subcategory, Model, and UnitMeasure) is joined to the corresponding foreign key in the Product fact table. For example:

- ProductSubcategory_t.Productsubcategoryid=Product.SubcategoryId
- ProductModel.ProductModelId=Product.ProductModelId
- UnıtMeasure_T.Unitmeasurecode=Product.Sizeunitmeasurecode
- UnıtMeasure_T.Unitmeasurecode= Product.Weightunitmeasurecode

The cube model based on Product snowflake schema is constituted around the Product fact object. Cubes have dimensions and measures. Measures describe how to calculate data from columns in the Product fact table. Product cube have Subcategory, Model, and UnitMeasure dimensions. Cube table fact object includes attributes that correspond to the foreign keys in the fact table that are used to join the dimensions to the facts object. The fact object has six measures: VT_LB (Valid time lower bound), VT_UB (Valid time upper bound), TT_LB (Transaction time lower bound), TT_UB (Transaction time upper bound), ListPriceValue, ProductId. Cube and has 3 attributes: ModelId, SubcategoryId, UnitMeasureId.

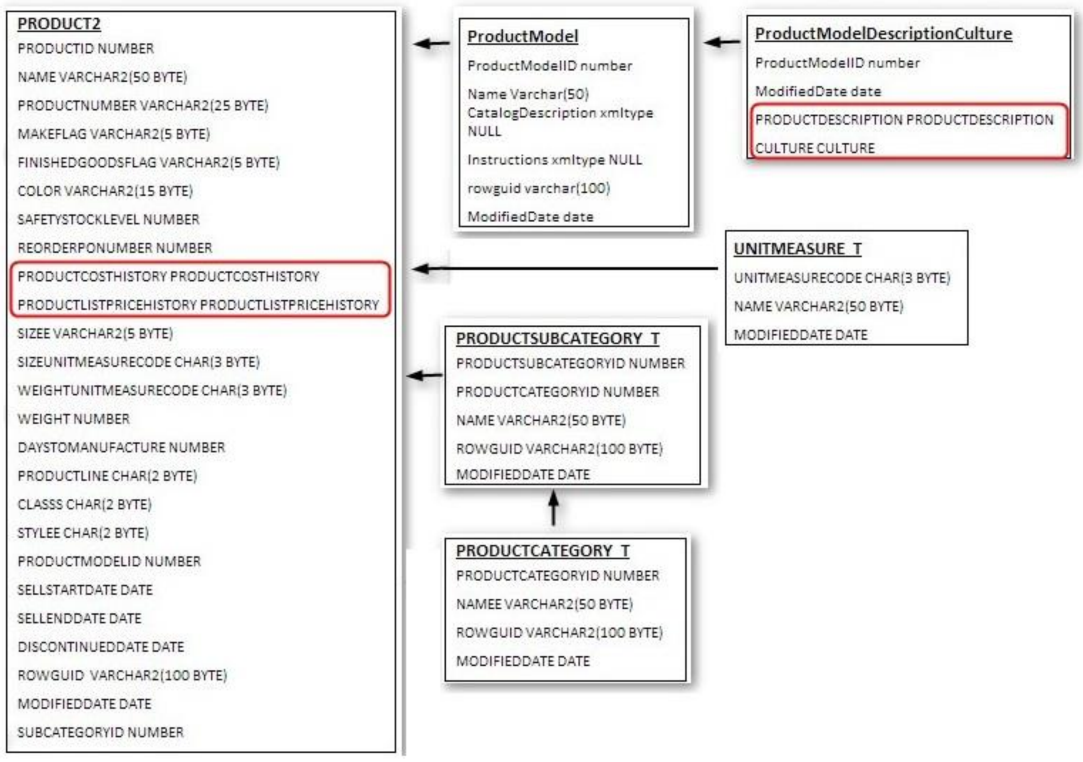


Figure 6.1 A snowflake schema based on Product fact table

Dimensions are connected to the facts object in a cube model like the dimension tables are connected to the fact table in a star schema. Columns of data from relational tables are represented by attribute objects referenced by the dimension.

Subcategory dimension references the following attributes.

- DIMENSION_KEY
- SUBCATEGORY_ID
- SUBCATEGORY_NAME
- SUBCATEGORY_MDFYDATE
- CATEGORY_ID

- CATEGORY_NAME
- CATEGORY_MDFYDATE

Model dimension references the following attributes

- DIMENSION_KEY
- MODEL_ID
- MODEL_NAME
- MODEL_MDFYDATE
- DESCRIPTION_ID
- DESCRIPTION_MDFYDATE
- LONG_DESC
- CULTURE_ID
- CULTURE_NAME
- IDVARCHAR

UnitMeasure dimension references the following attributes. Only one level is forbidden in creating dimensions, therefore two level created for unitmeasure dimension. Only one of them unitmeasure level is used.

- UNITMEASURE_ID
- UNITMEASURE_NAME
- UNITMEASURE_DATE
- LEVEL2_ID
- LEVEL2_NAME
- LEVEL2_DATE

A join is created to connect each dimension to the facts object. The three joins in this example are Model, Unitmeasure and Subcategory.

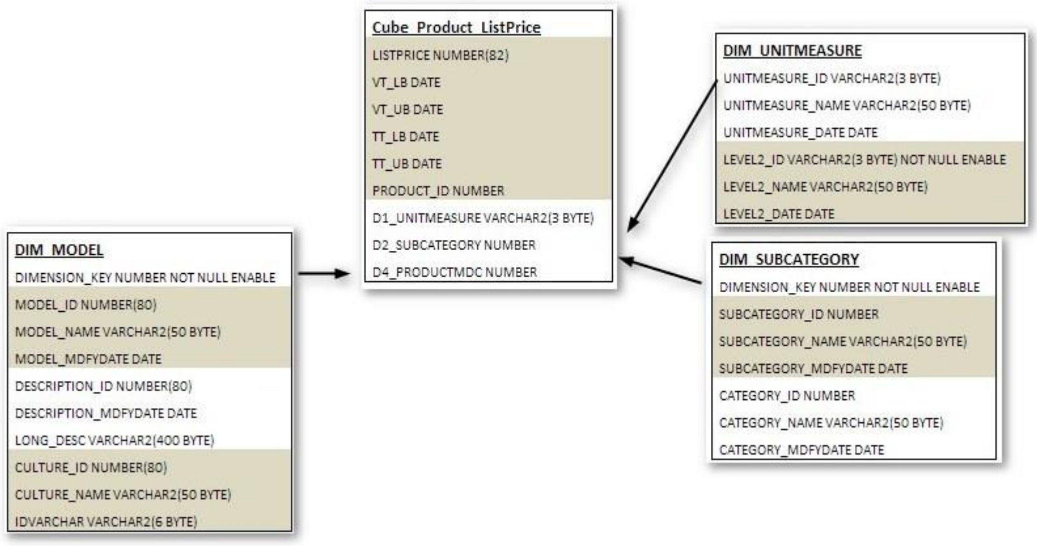

Cube_Product_ListPrice
LISTPRICE NUMBER(82)
VT_LB DATE
VT_UB DATE
TT_LB DATE
TT_UB DATE
PRODUCT_ID NUMBER
D1_UNITMEASURE VARCHAR2(3 BYTE)
D2_SUBCATEGORY NUMBER
D4_PRODUCTMDC NUMBER
DIM_UNITMEASURE
UNITMEASURE_ID VARCHAR2(3 BYTE)
UNITMEASURE_NAME VARCHAR2(50 BYTE)
UNITMEASURE_DATE DATE
LEVEL2_ID VARCHAR2(3 BYTE) NOT NULL ENABLE
LEVEL2_NAME VARCHAR2(50 BYTE)
LEVEL2_DATE DATE
DIM_MODEL
DIMENSION_KEY NUMBER NOT NULL ENABLE
MODEL_ID NUMBER(80)
MODEL_NAME VARCHAR2(50 BYTE)
MODEL_MDFYDATE DATE
DESCRIPTION_ID NUMBER(80)
DESCRIPTION_MDFYDATE DATE
LONG_DESC VARCHAR2(400 BYTE)
CULTURE_ID NUMBER(80)
CULTURE_NAME VARCHAR2(50 BYTE)
IDVARCHAR VARCHAR2(6 BYTE)
DIM_SUBCATEGORY
DIMENSION_KEY NUMBER NOT NULL ENABLE
SUBCATEGORY_ID NUMBER
SUBCATEGORY_NAME VARCHAR2(50 BYTE)
SUBCATEGORY_MDFYDATE DATE
CATEGORY_ID NUMBER
CATEGORY_NAME VARCHAR2(50 BYTE)
CATEGORY_MDFYDATE DATE

Figure 6.2 Diagram of CubeProductListPrice cube's dimensional objects

Hierarchies store information about how the attributes grouped into levels within a dimension are related to each other and structured. As a metadata object, a hierarchy provides a way to calculate and navigate across the dimension. Each dimension has a corresponding hierarchy with levels that group related attributes as in Figure 6.3. In a cube model, each dimension can have multiple hierarchies.

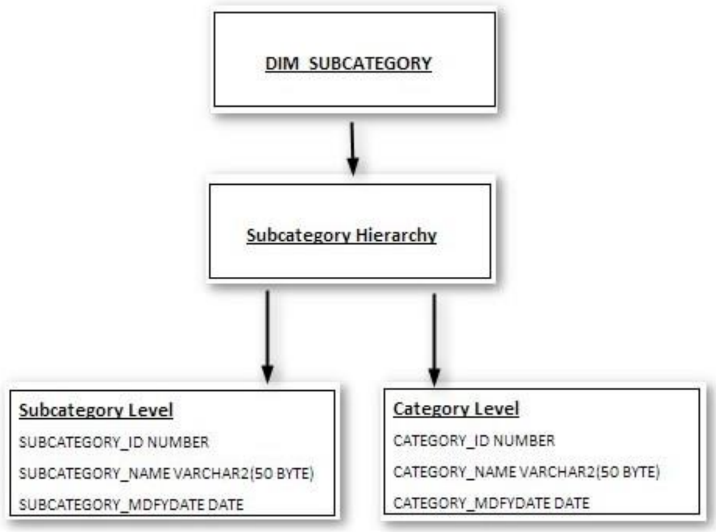
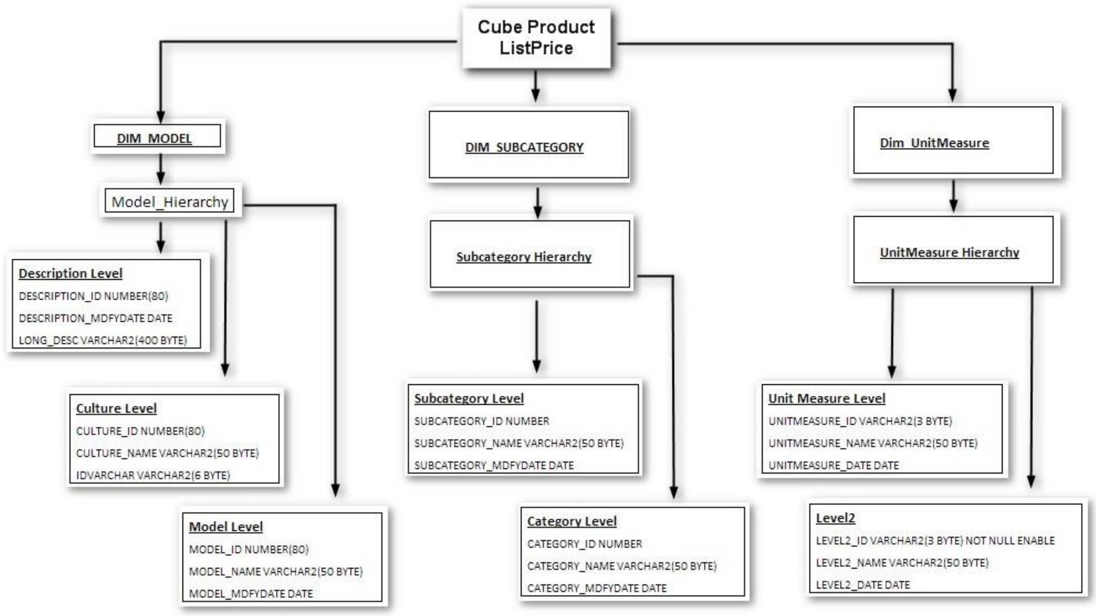


Figure 6.3 The hierarchy of DIM_SUBCATEGORY dimension

One or more cubes may be built for the cube model. The AdventureWorks user product module has five cubes, but only the CubeProductListPrice is described here. Cube Product ListPrice is shown in Figure 6.4.



Cube Product ListPrice
DIM_MODEL
Model_Hierarchy
Description Level
DESCRIPTION_ID NUMBER(80)
DESCRIPTION_MDFYDATE DATE
LONG_DESC VARCHAR2(400 BYTE)
Culture Level
CULTURE_ID NUMBER(80)
CULTURE_NAME VARCHAR2(50 BYTE)
IDVARCHAR VARCHAR2(6 BYTE)
Model Level
MODEL_ID NUMBER(80)
MODEL_NAME VARCHAR2(50 BYTE)
MODEL_MDFYDATE DATE
DIM_SUBCATEGORY
Subcategory Hierarchy
Subcategory Level
SUBCATEGORY_ID NUMBER
SUBCATEGORY_NAME VARCHAR2(50 BYTE)
SUBCATEGORY_MDFYDATE DATE
Category Level
CATEGORY_ID NUMBER
CATEGORY_NAME VARCHAR2(50 BYTE)
CATEGORY_MDFYDATE DATE
Dim_UnitMeasure
UnitMeasure Hierarchy
Unit Measure Level
UNITMEASURE_ID VARCHAR2(3 BYTE)
UNITMEASURE_NAME VARCHAR2(50 BYTE)
UNITMEASURE_DATE DATE
Level2
LEVEL2_ID VARCHAR2(3 BYTE) NOT NULL ENABLE
LEVEL2_NAME VARCHAR2(50 BYTE)
LEVEL2_DATE DATE

Figure 6.4 All dimensional objects in cube

6.1.2 Mapping

After creating necessary dimensional objects, we have loaded data into them. Warehouse builder mapping tool is used to extract data from database and load to target warehouse builder repository.

Mapping tool has a design page, dimensional objects and the database objects can be put in this designer. The component pallet in mapping tool is shown in Figure 6.5.

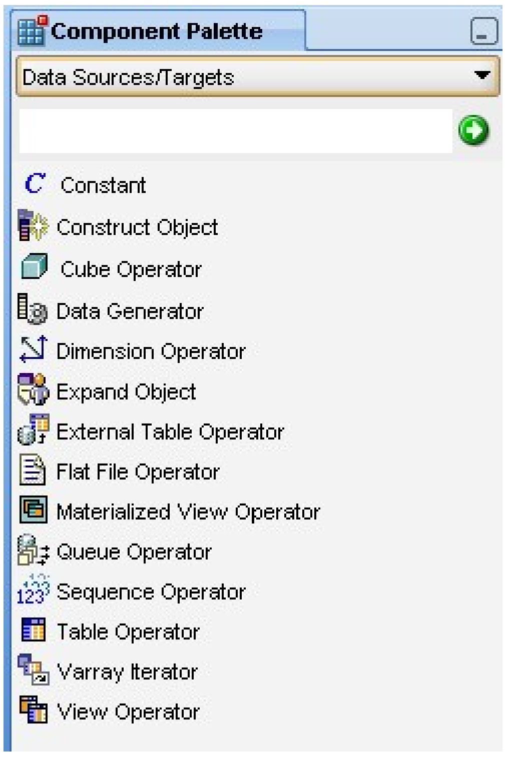


Figure 6.5 Some mapping objects in Warehouse Builder

6.1.2.1 Mapping Dim_UnitMeasure

When Dim_unitmeasure created, D1_UNITMEASURE_TAB is automatically created in the same form with dimension in warehousing repository. There are no data in this dimension table at first.

Figure 6.6 is the map for loading D1_UNITMEASURE_TAB. Unitmeasure_t is the database object. D1_UNITMEASURE_TAB dimension table is the warehouse object. As seen in Figure 6.6, related columns are connected each other via arrows. This mapping is compiled with no errors. Then for starting data transformation the red marked part is clicked. D1_UNITMEASURE_TAB table is filled by data. This is the simplest mapping example because one dimension is loaded from only one regular table.

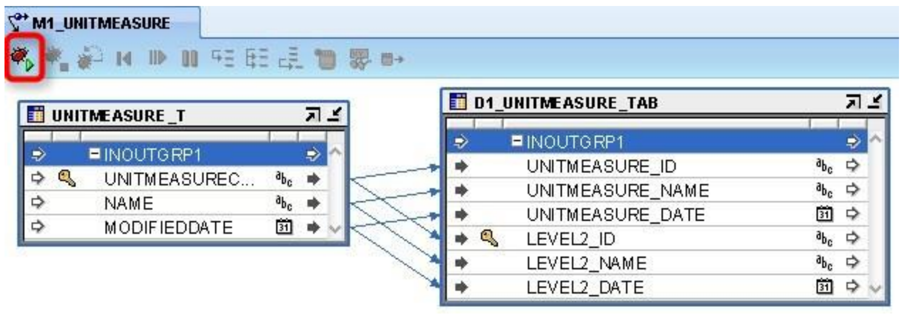


Figure 6.6 Mapping diagram of DIM_UNITMEASURE

6.1.2.2 Mapping Dim_SubCategory

When Dim_SubCategory created, D2_SUBCATEGORY_TAB is automatically created in the same form with dimension in warehousing repository. There is no data in this dimension table at first.

Dim_SubCategory gets data from two different tables. One of them is ProductSubcategory_t and the other one is ProductCategory_t. For connecting multiple tables with dimension table, joiner control must be used.

Joiner is a mapping control which is stated in component palette. This control has two input groups and an output group. Two input group is bounded via a join condition. All inputs are shown up in output group. After joining two tables, outputs are bind to D2_SUBCATEGORY_TAB. Mapping compiled with no errors and run. D2_SUBCATEGORY_TAB is filled by dimension table's data.

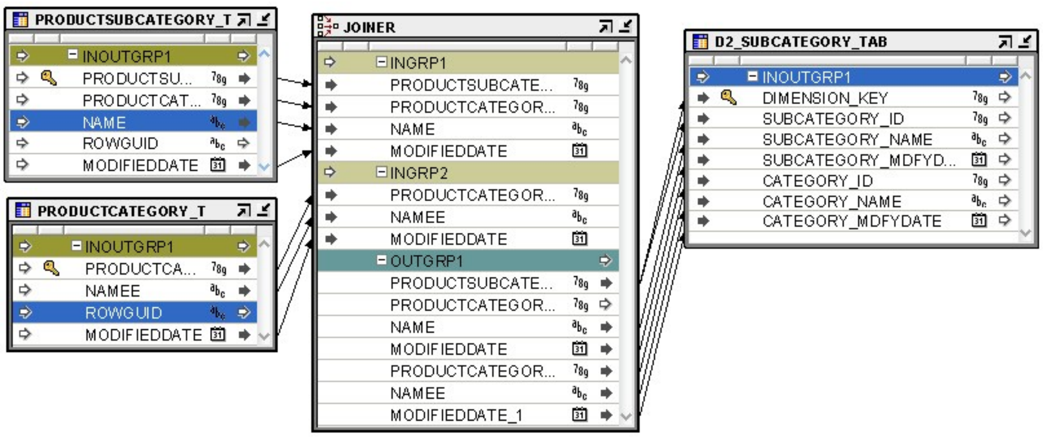


Figure 6.7 Mapping diagram of DIM_SUBCATEGORY

6.1.2.3 Mapping Dim_Model

When Dim_Model created, D4_PRODUCTMDC_TAB is automatically created in the same form with dimension in warehousing repository. There are no data in this dimension table at first.

While creating dimensions, a lot of experiments gained. Dimension table start with D4.., in here 4 means, this is the fourth dimension creation experiment of dim_model.

D4_PRODUCTMDC_TAB gets data from Productmodel and ProductModelDescriptionCulture tables. Culture and Description levels' data come from ProductModelDescriptionCulture's ProductDescription and Culture nested tables.

In the Figure 6.8 ProductModelDescriptionCulture table is seen. This table contains two nested tables. These nested tables is expanded by using Varray Iterator control and Expand Object.

Varray iterator gets nested table types and converts them to nested types. Expand objects gets nested types and returns attributes of mentioned nested type as output.

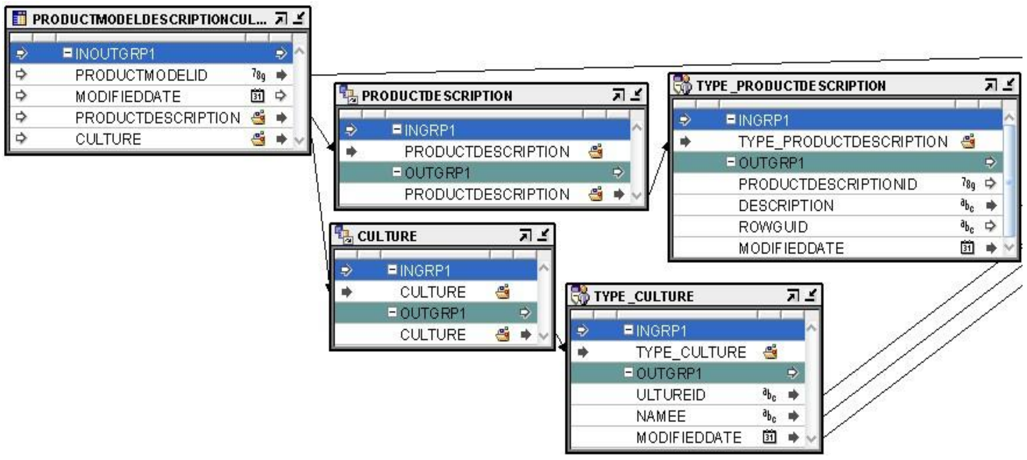


Figure 6.8 Mapping diagram of DIM_MODEL part 1

The outputs of expand objects and productmodel table is merged by joiner object. Outputs are bind to D4_PRODUCTMDC_TAB dimension table. This mapping is compiled and run successfully.

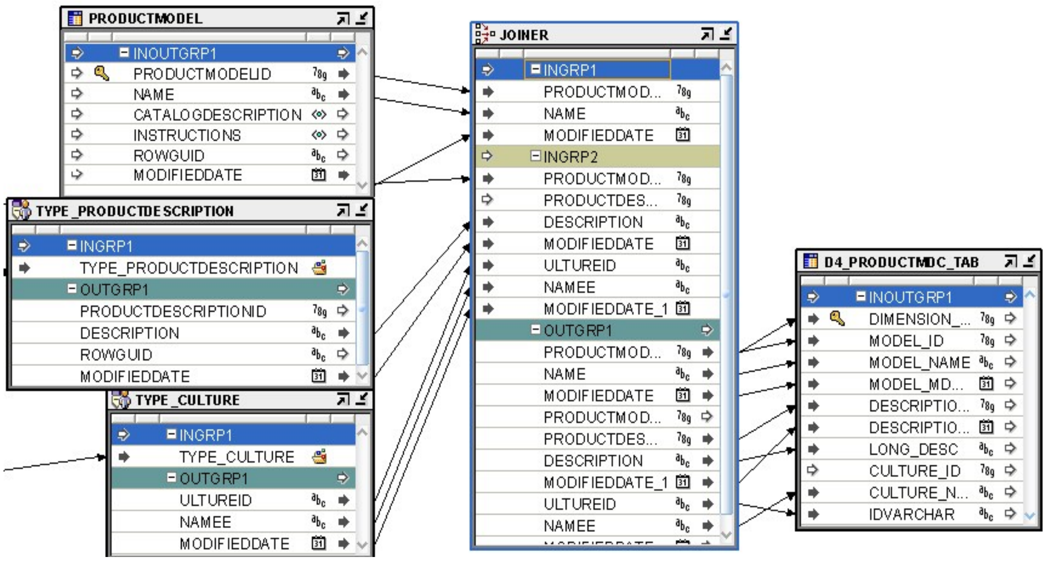


Figure 6.9 Mapping diagram of DIM_MODEL part 2

6.1.2.4 Mapping Cube_ProductListPrice

When Cube_ProductListPrice created, C_LISTPRICE_3DIM_TAB is automatically created in the same form with cube in warehousing repository. There are no data in this cube table at first. C_LISTPRICE_3DIM_TAB's dimensional attributes are bind from Product fact table by arrows. The measure of the cube is come from nested ProductListPriceHistory column. ProductListPriceHistory expanded by using Varray iterator and Expand objects components. VT_LB, VT_UB, TT_LB, TT_UB and value outputs of BT_NUMBER nested table are connected to cube table's measure attributes by arrows.

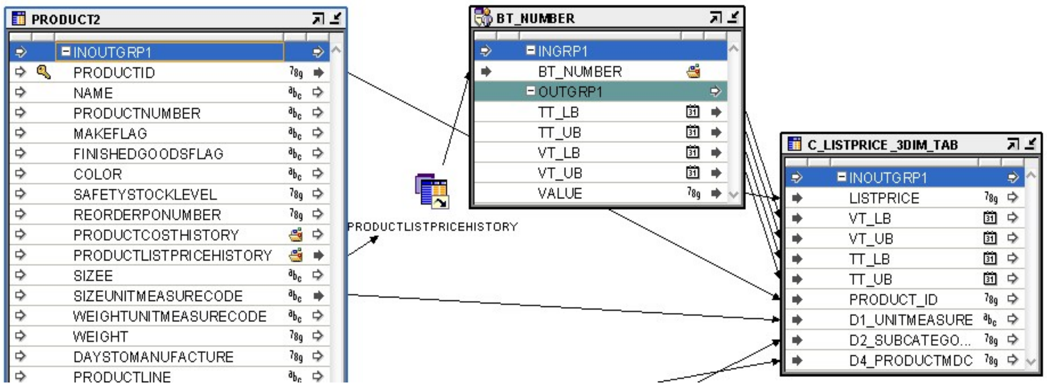


Figure 6.10 Mapping diagram of Cube_ProductListPrice

6.1.3 Querying

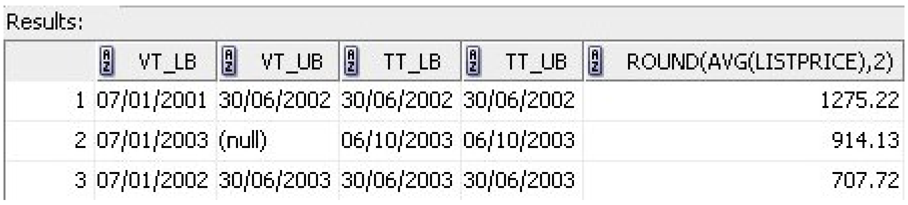
Olap queries are the queries that get result from cube or dimensions. Some olap queries are written with cubeproductlistprice and its dimensions for understanding resulting features of data warehouse.

Query 1: What is the average list price of Bikes according to years?

The query which answers these questions is written in Table 6.1. Cubeproductlistprice cube and Dim_Subcategory dimension tables are used. The result contains three rows. Each of them is the result of year (2001, 2002, 2003).

Table 6.1 Query of question

```
SELECT VT_LB,VT_UB,TT_LB,TT_UB,ROUND(AVG(LISTPRICE),2)
FROM cubeproductlistprice WHERE dim_subcategory_key
IN (SELECT DIMENSION_KEY FROM DIM_SUBCATEGORY
WHERE CATEGORY_NAME='Bikes') GROUP BY VT_LB,VT_UB,TT_LB,TT_UB;
```



```
Results:
     VT_LB      VT_UB      TT_LB      TT_UB      ROUND(AVG(LISTPRICE),2)
1  07/01/2001 30/06/2002 30/06/2002 30/06/2002                  1275.22
2  07/01/2003 (null)     06/10/2003 06/10/2003                   914.13
3  07/01/2002 30/06/2003 30/06/2003 30/06/2003                   707.72
```

Figure 6.11 Query result screenshot

The solution of this query also can be written in relational database form of AdventureWorks database. This solution query is given in Table 6.2; it is a longer statement than the statement which was constituted with warehousing objects.

Table 6.2 Query of question written with relational objects

```
select StartDate AS VT_LB, EndDate AS VT_UL, ModifiedDate AS TTLB, ModifiedDate AS
TT_UB, AVG( ListPrice) from Production.ProductListPriceHistory
where ProductID in (select ProductID from Production.Product
where ProductSubcategoryID in(select ProductSubcategoryID  from
Production.ProductSubcategory  where ProductCategoryID= (select ProductCategoryID from
Production.ProductCategory where Name='Bikes'))) GROUP BY StartDate,EndDate,ModifiedDate;
```

System performance or some other variants can effect on query performance. Both queries are run 6 times and query execution times are calculates according to milliseconds. The chart in Figure 6.12 shows execution times of queries in 6 operations.

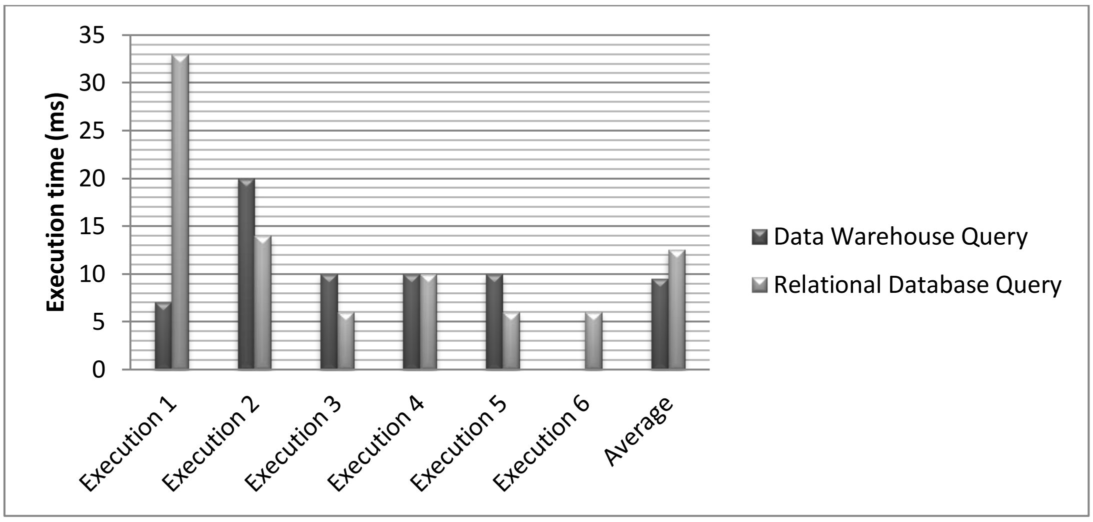


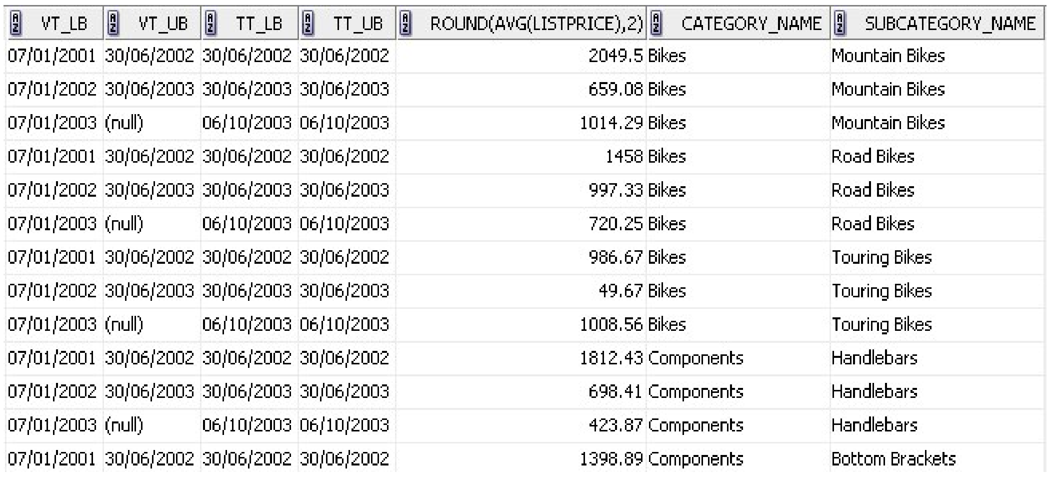
Figure 6.12 Chart shows six average execution time in each environment. The seventh measure is the average execution times.

Query 2: What is the average list price of all products according to years, categories and subcategories?

The key query for this question is given in Table 6.3. Category and Subcategory were in separate tables before. After designing dimensional model, Dim_SubCategory is created automatically. Only Dim_SubCategory table is enough to reach Subcategory and Category tables' information.

Table 6.3 Query of question with dimensional database objects

```
SELECT VT_LB,VT_UB,TT_LB,TT_UB,ROUND(AVG(LISTPRICE),2),
(SELECT category_name FROM DIM_SUBCATEGORY
WHERE dimension_key= dim_subcategory_key) CATEGORY_NAME,
(SELECT subcategory_name FROM DIM_SUBCATEGORY
WHERE dimension_key= dim_subcategory_key) SUBCATEGORY_NAME
FROM cubeproductlistprice
GROUP BY VT_LB,VT_UB,TT_LB,TT_UB,dim_subcategory_key
ORDER BY dim_subcategory_key;
```



VT_LB	VT_UB	TT_LB	TT_UB	ROUND(AVG(LISTPRICE),2)	CATEGORY_NAME	SUBCATEGORY_NAME
07/01/2001	30/06/2002	30/06/2002	30/06/2002	2049.5	Bikes	Mountain Bikes
07/01/2002	30/06/2003	30/06/2003	30/06/2003	659.08	Bikes	Mountain Bikes
07/01/2003	(null)	06/10/2003	06/10/2003	1014.29	Bikes	Mountain Bikes
07/01/2001	30/06/2002	30/06/2002	30/06/2002	1458	Bikes	Road Bikes
07/01/2002	30/06/2003	30/06/2003	30/06/2003	997.33	Bikes	Road Bikes
07/01/2003	(null)	06/10/2003	06/10/2003	720.25	Bikes	Road Bikes
07/01/2001	30/06/2002	30/06/2002	30/06/2002	986.67	Bikes	Touring Bikes
07/01/2002	30/06/2003	30/06/2003	30/06/2003	49.67	Bikes	Touring Bikes
07/01/2003	(null)	06/10/2003	06/10/2003	1008.56	Bikes	Touring Bikes
07/01/2001	30/06/2002	30/06/2002	30/06/2002	1812.43	Components	Handlebars
07/01/2002	30/06/2003	30/06/2003	30/06/2003	698.41	Components	Handlebars
07/01/2003	(null)	06/10/2003	06/10/2003	423.87	Components	Handlebars
07/01/2001	30/06/2002	30/06/2002	30/06/2002	1398.89	Components	Bottom Brackets

Figure 6.13 Query result screenshot

The answer query of this question is also written with relational database objects. This statement is given in Table 6.4. As it appears, statement has more join conditions than the one written with dimensional objects.

Table 6.4 Query of question written with relational objects

```
SELECT StartDate AS VT_LB, EndDate AS VT_UL,
PLP.ModifiedDate AS TTLB,PLP.ModifiedDate AS TT_UB,
PC.Name AS CATEGORY_NAME,
PSC.Name AS SUBCATEGORY_NAME, AVG(PLP.ListPrice)
FROM Production.ProductListPriceHistory PLP,
Production.ProductSubcategory PSC, Production.Product P,
Production.ProductCategory PC WHERE PLP.ProductID=P.ProductID
AND PSC.ProductSubcategoryID=P.ProductSubcategoryID
AND PC.ProductCategoryID=PSC.ProductCategoryID
GROUP BY StartDate,EndDate,PLP.ModifiedDate,PC.Name,PSC.Name;
```

Figure 6.14 shows query execution times according to data warehouse query and relational database query as milliseconds for six times. Seventh measurement is average performance.

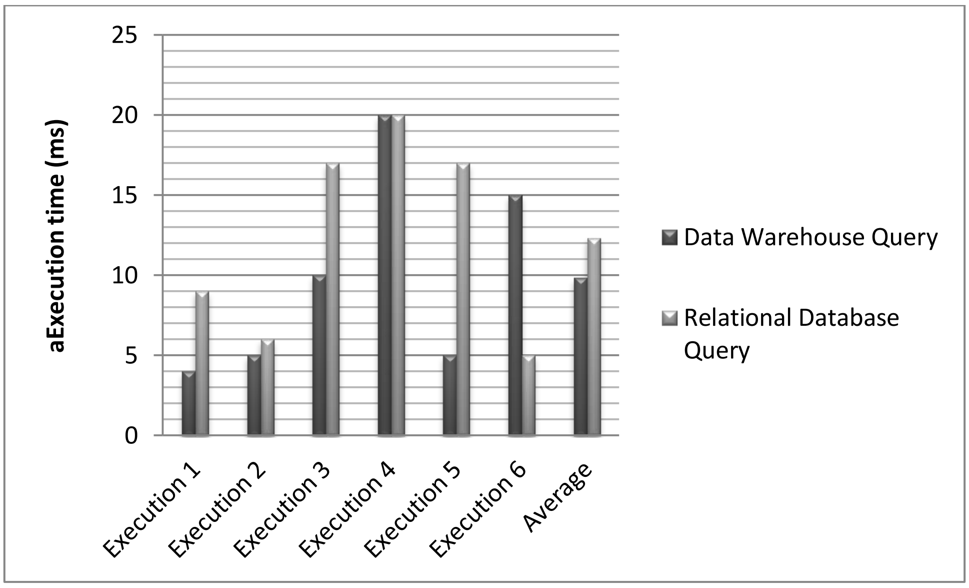
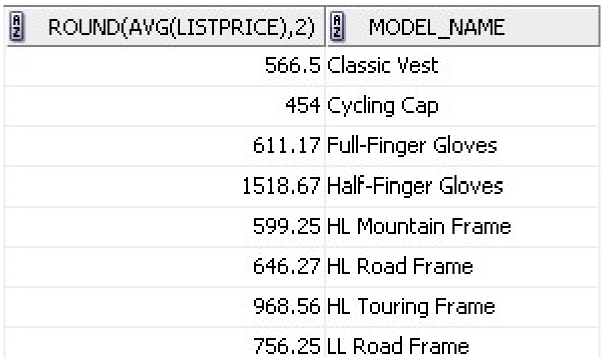


Figure 6.14 Chart shows six average execution time in each environment. The seventh measure is the average execution times.

Query 3: What is the average list price according to models?

Table 6.5 Query of question

```
SELECT ROUND(AVG(LISTPRICE),2),
(SELECT MODEL_NAME FROM DIM_MODEL
WHERE dimension_key=DIM_PRODUCTMDC_KEY) MODEL_NAME
FROM cubeproductlistprice GROUP BY DIM_PRODUCTMDC_KEY
ORDER BY DIM_PRODUCTMDC_KEY;
```



ROUND(AVG(LISTPRICE),2)	MODEL_NAME
566.5	Classic Vest
454	Cycling Cap
611.17	Full-Finger Gloves
1518.67	Half-Finger Gloves
599.25	HL Mountain Frame
646.27	HL Road Frame
968.56	HL Touring Frame
756.25	LL Road Frame

Figure 6.15 Query result screenshot

Table 6.6 Query of question written with relational objects

```
SELECT StartDate AS VT_LB, EndDate AS VT_UL, PLP.ModifiedDate AS
TTLB,PLP.ModifiedDate AS TT_UB, PM.Name AS MODEL_NAME,AVG(PLP.ListPrice)
FROM Production.ProductListPriceHistory PLP, Production.ProductModel PM,
Production.Product P WHERE PLP.ProductID=P.ProductID AND
PM.ProductModelID=P.ProductModelID
GROUP BY StartDate,EndDate,PLP.ModifiedDate,PM.Name;
```

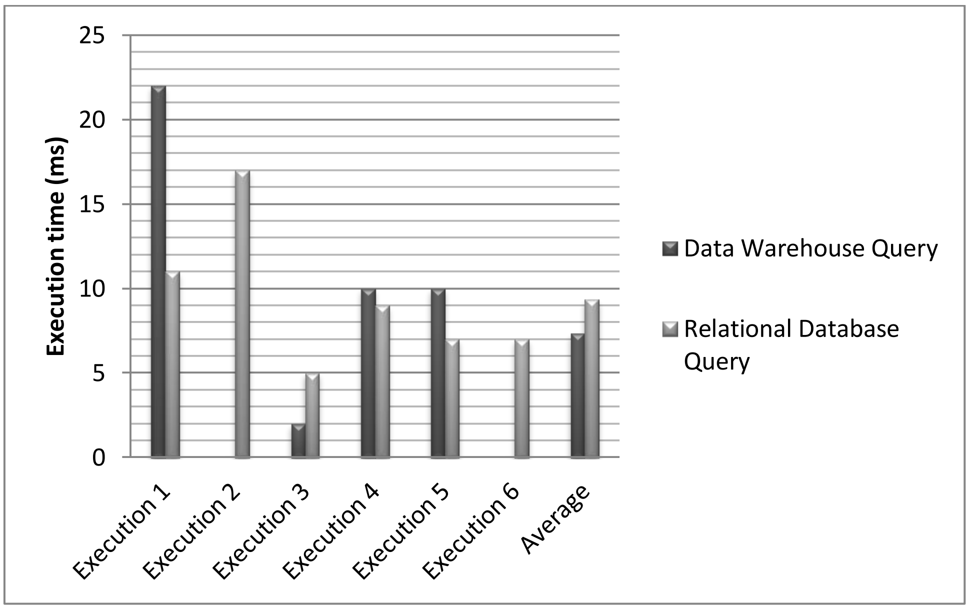


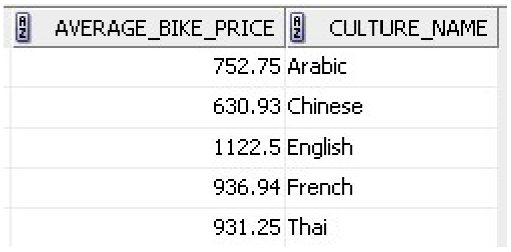
Figure 6.16 Query comparison chart

Query 4: Query 4: What is the average list price of Bikes According to Cultures between 01.01.2000 and 01.01.2004?

Query result shows that Chinese originate bikes are the cheapest ones. English originate bikes are the most expensive ones.

Table 6.7 Query of question

```
    SELECT ROUND(AVG(LISTPRICE),2) Average_Bike_Price,
    DIM_MODEL.CULTURE_NAME FROM cubeproductlistprice,
    DIM_MODEL WHERE DIM_MODEL.DIMENSION_KEY=
    cubeproductlistprice.dim_productmdc_key
    and cubeproductlistprice.PRODUCT_ID IN (SELECT PRODUCTID
    FROM PRODUCT2 WHERE SUBCATEGORYID IN(SELECT SUBCATEGORY_ID
    FROM DIM_SUBCATEGORY WHERE CATEGORY_NAME='Bikes' )) and
vt_lb>to_date('01.01.2000','DD/MM/YYYY') and vt_ub<to_date('01.01.2004','DD/MM/YYYY')
    GROUP BY DIM_MODEL.CULTURE_NAME
    ORDER BY DIM_MODEL.CULTURE_NAME;
```



AVERAGE_BIKE_PRICE	CULTURE_NAME
752.75	Arabic
630.93	Chinese
1122.5	English
936.94	French
931.25	Thai

Figure 6.17 Query result screenshot

Table 6.8 Query of question written with relational objects

```
select AVG( PL.ListPrice), C.Name from Production.ProductListPriceHistory PL,
Production.ProductModel PM, Production.Product P,
Production.ProductModelProductDescriptionCulture PMDC, Production.Culture C
where P.ProductModelID=PM.ProductModelID AND PL.ProductID=P.ProductID
AND PM.ProductModelID=PMDC.ProductModelID AND PMDC.CultureID=C.CultureID
AND  PL.StartDate>convert(DATE, '01.01.2000', 103) and PL.EndDate<convert(DATE,
'01.01.2004', 103) AND PL.ProductID in (select ProductID from Production.Product
where ProductSubcategoryID in(select ProductSubcategoryID
from Production.ProductSubcategory  where ProductCategoryID=
(select ProductCategoryID from Production.ProductCategory where Name='Bikes'))) GROUP BY
C.Name ;
```

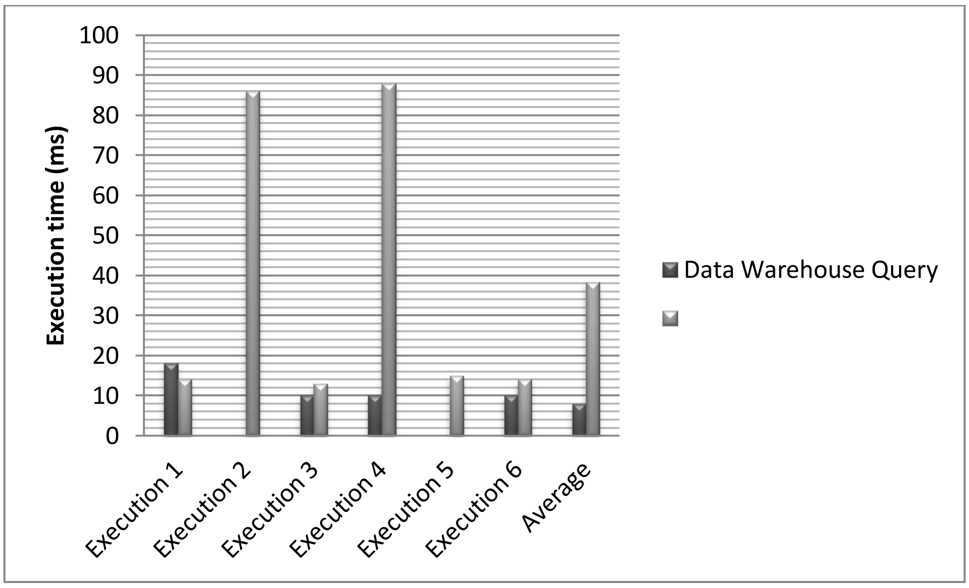


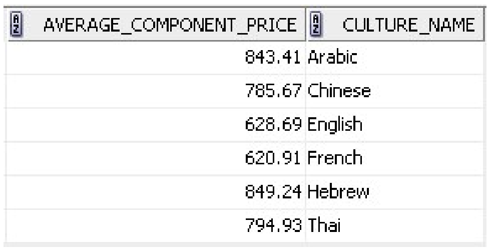
Figure 6.18 Query comparison chart

Query 5: What is the average list price of Components According to Cultures?

Query result in Table 6.19 shows that French originate components are the cheapest ones in average. Hebrew originate components are the most expensive ones.

Table 6.9 Query of question

```
SELECT ROUND(AVG(LISTPRICE),2) Average_Component_Price,
DIM_MODEL.CULTURE_NAME
FROM cubeproductlistprice,DIM_MODEL WHERE DIM_MODEL.DIMENSION_KEY=
cubeproductlistprice.dim_productmdc_key
and cubeproductlistprice.PRODUCT_ID IN (SELECT PRODUCTID
FROM PRODUCT2 WHERE SUBCATEGORYID IN(SELECT SUBCATEGORY_ID
FROM DIM_SUBCATEGORY WHERE CATEGORY_NAME='Components' ))
GROUP BY DIM_MODEL.CULTURE_NAME;
ORDER BY DIM_MODEL.CULTURE_NAME;
```



AVERAGE_COMPONENT_PRICE	CULTURE_NAME
843.41	Arabic
785.67	Chinese
628.69	English
620.91	French
849.24	Hebrew
794.93	Thai

Figure 6.19 Query result screenshot

Table 6.10 Query of question written with relational objects

```
select AVG( PL.ListPrice), C.Name from Production.ProductListPriceHistory PL,
Production.ProductModel PM, Production.Product P,
Production.ProductModelProductDescriptionCulture PMDC, Production.Culture
where P.ProductModelID=PM.ProductModelID AND PL.ProductID=P.ProductID
AND PM.ProductModelID=PMDC.ProductModelID AND PMDC.CultureID=C.CultureID AND
PL.ProductID in (select ProductID from Production.Product
where ProductSubcategoryID in(select ProductSubcategoryID
from Production.ProductSubcategory  where ProductCategoryID=
(select ProductCategoryID from Production.ProductCategory where Name='Components')))
GROUP BY  C.Name ;
```

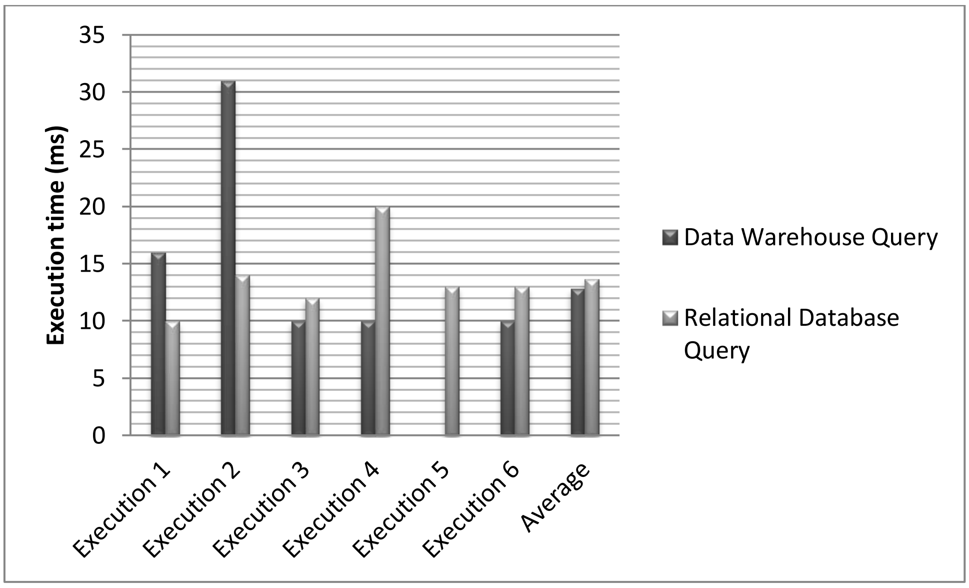


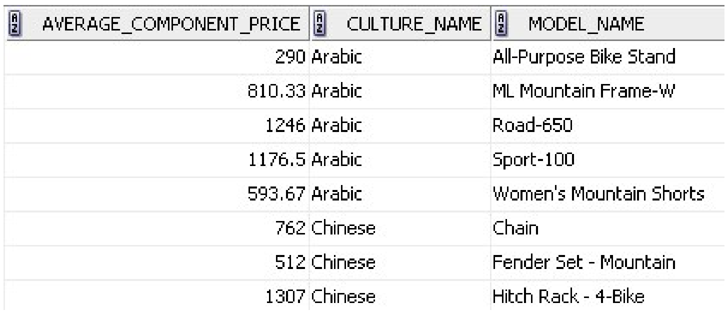
Figure 6.20 Query comparison chart

Query 6: What is the average list price of Components According to Cultures AND Models?

The query of this question is given in Table 6.11. CubeProductListPrice and Dim_model is joined. In oracle relational database design there were two tables about model dimension. One of them was ProductModelDescriptionCulture which contains PRODUCTDESCRIPTION and CULTURE nested types in it. This three different table and nested tables are stored in one dimension table.

Table 6.11 Query of question

```
SELECT ROUND(AVG(LISTPRICE),2) Average_Component_Price,
DIM_MODEL.CULTURE_NAME, DIM_MODEL.MODEL_NAME
FROM cubeproductlistprice,DIM_MODEL WHERE DIM_MODEL.DIMENSION_KEY=
cubeproductlistprice.dim_productmdc_key
and cubeproductlistprice.PRODUCT_ID IN (SELECT PRODUCTID
FROM PRODUCT2 WHERE SUBCATEGORYID IN(SELECT SUBCATEGORY_ID
FROM DIM_SUBCATEGORY WHERE CATEGORY_NAME='Components' ))
GROUP BY DIM_MODEL.MODEL_NAME,DIM_MODEL.CULTURE_NAME
ORDER BY DIM_MODEL.CULTURE_NAME, DIM_MODEL.MODEL_NAME;
```



AVERAGE_COMPONENT_PRICE	CULTURE_NAME	MODEL_NAME
290	Arabic	All-Purpose Bike Stand
810.33	Arabic	ML Mountain Frame-W
1246	Arabic	Road-650
1176.5	Arabic	Sport-100
593.67	Arabic	Women's Mountain Shorts
762	Chinese	Chain
512	Chinese	Fender Set - Mountain
1307	Chinese	Hitch Rack - 4-Bike

Figure 6.21 Query result screenshot

Table 6.12 Query of question written with relational objects

```
select AVG( PL.ListPrice), C.Name , PM.Name
from Production.ProductListPriceHistory PL, Production.ProductModel PM,
Production.Product P,
Production.ProductModelProductDescriptionCulture PMDC, Production.Culture C where
P.ProductModelID=PM.ProductModelID AND PL.ProductID=P.ProductID
AND PM.ProductModelID=PMDC.ProductModelID AND PMDC.CultureID=C.CultureID
AND PL.ProductID=P.ProductID  AND PL.ProductID in (select ProductID from
Production.Product where ProductSubcategoryID in(select ProductSubcategoryID from
Production.ProductSubcategory  where ProductCategoryID= (select ProductCategoryID from
Production.ProductCategory where Name='Components'))) GROUP BY  C.Name,PM.Name ;
```

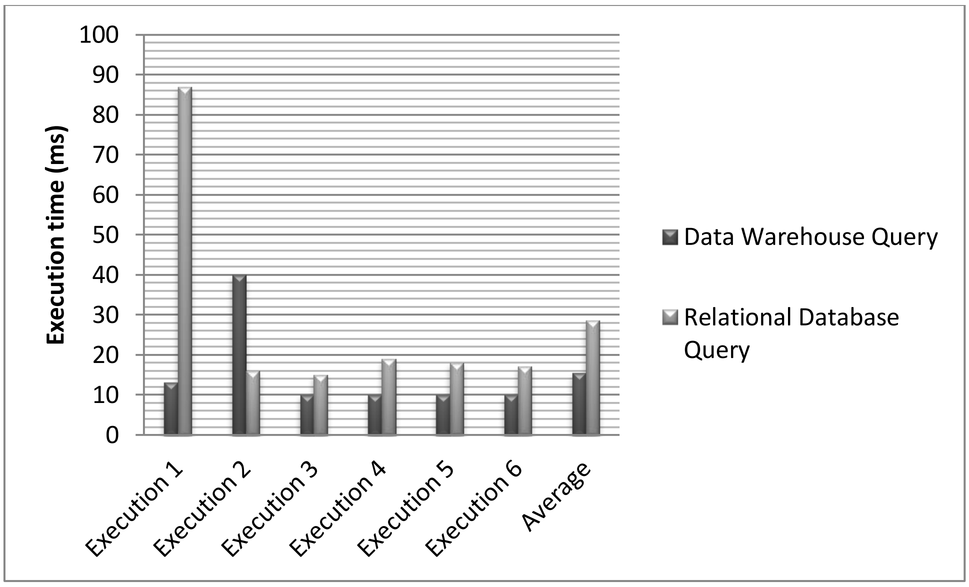
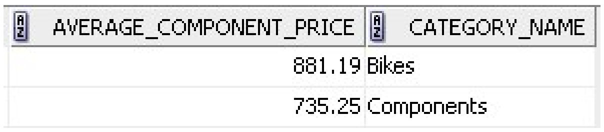


Figure 6.22 Query comparison chart

Query 7: What is the average list price of Components According to Categories?

Table 6.13 Query of question

```
    SELECT ROUND(AVG(LISTPRICE),2) Average_Component_Price,
DIM_SUBCATEGORY.CATEGORY_NAME FROM cubeproductlistprice,DIM_SUBCATEGORY
WHERE DIM_SUBCATEGORY.DIMENSION_KEY=cubeproductlistprice.DIM_SUBCATEGORY_KEY
GROUP BY DIM_SUBCATEGORY.CATEGORY_NAME ORDER BY
DIM_SUBCATEGORY.CATEGORY_NAME;
```



AVERAGE_COMPONENT_PRICE	CATEGORY_NAME
881.19	Bikes
735.25	Components

Figure 6.23 Query result screenshot

Table 6.14 Query of question written with relational objects

```
select AVG( PL.ListPrice), PC.Name from Production.ProductListPriceHistory PL, Production.Product P,
Production.ProductSubcategory PS,
Production.ProductCategory PC where P.ProductSubcategoryID= PS.ProductSubcategoryID AND
PS.ProductCategoryID=PC.ProductCategoryID
 AND PL.ProductID=P.ProductID AND PL.ProductID in (select ProductID from Production.Product where
ProductSubcategoryID in(select ProductSubcategoryID  from Production.ProductSubcategory  where
ProductCategoryID= (select ProductCategoryID from Production.ProductCategory where
Name='Components'))) GROUP BY PC.Name;
```

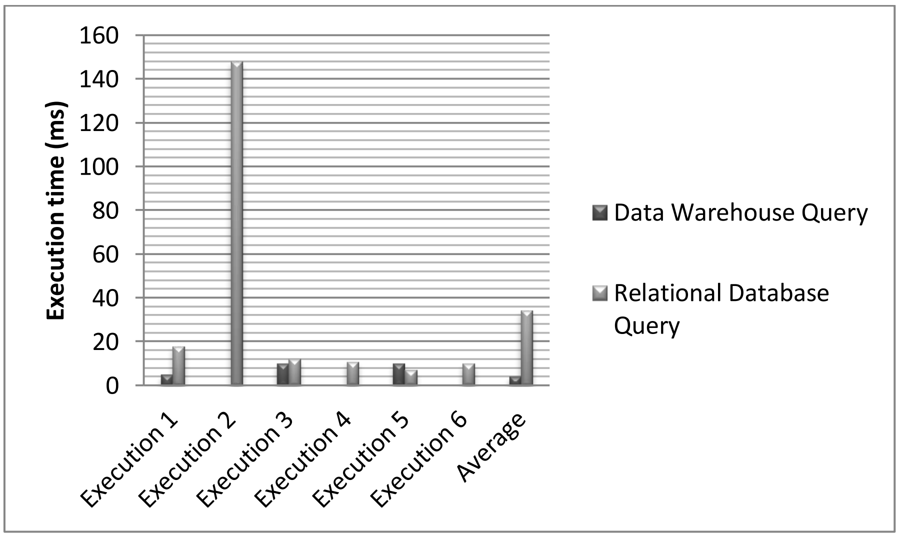


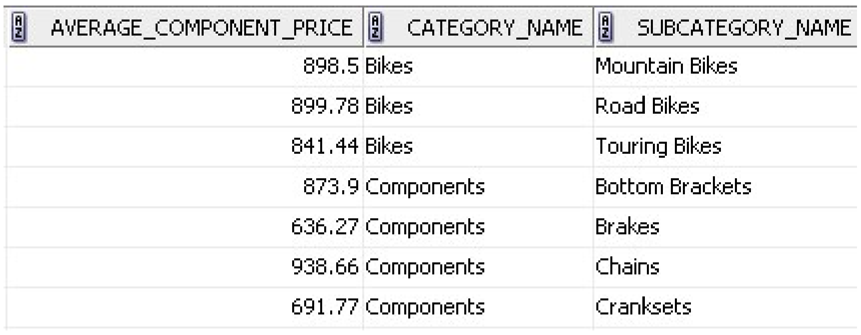
Figure 6.24 Query comparison Chart

Query 8: What is the average list price of Components According to Categories and SubCategories?

The result query of this question is given in Table 6.15. It provides acces to two levels of Dim_Subcategory dimension.

Table 6.15 Query of question

```
    SELECT ROUND(AVG(LISTPRICE),2) Average_Component_Price,
    DIM_SUBCATEGORY.CATEGORY_NAME,DIM_SUBCATEGORY.SUBCATEGORY_NA
ME FROM cubeproductlistprice,DIM_SUBCATEGORY
    WHERE DIM_SUBCATEGORY.DIMENSION_KEY=
    cubeproductlistprice.DIM_SUBCATEGORY_KEY GROUP BY
DIM_SUBCATEGORY.CATEGORY_NAME,
    DIM_SUBCATEGORY.SUBCATEGORY_NAME
    ORDER BY DIM_SUBCATEGORY.CATEGORY_NAME,
    DIM_SUBCATEGORY.SUBCATEGORY_NAME;
```



AVERAGE_COMPONENT_PRICE	CATEGORY_NAME	SUBCATEGORY_NAME
898.5	Bikes	Mountain Bikes
899.78	Bikes	Road Bikes
841.44	Bikes	Touring Bikes
873.9	Components	Bottom Brackets
636.27	Components	Brakes
938.66	Components	Chains
691.77	Components	Cranksets

Figure 6.25 Query result screenshot

Table 6.16 Query of question written with relational objects

```
select AVG( PL.ListPrice), pc.Name category, ps.Name subcategory
from Production.ProductListPriceHistory PL, Production.Product P,
Production.ProductSubcategory pc, Production.ProductSubcategory PS
where PS.ProductCategoryID= PL.ProductID and pc.ProductCategoryID= ps.ProductCategoryID
and pl.ProductID in (select ProductID from Production.Product where ProductSubcategoryID
in(select ProductSubcategoryID from Production.ProductSubcategory  where
ProductCategoryID= (select ProductCategoryID from Production.ProductCategory where
Name='Components'))) GROUP BY pc.Name , ps.Name ;
```

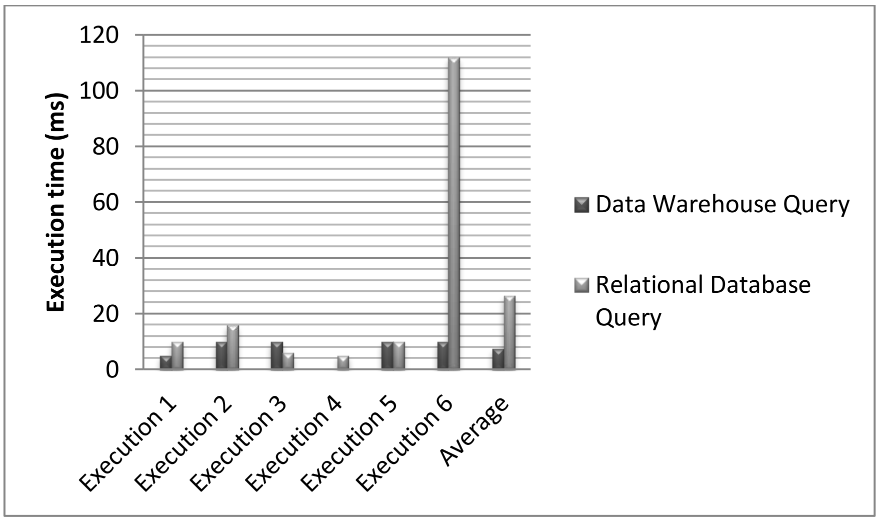


Figure 6.26 Query comparison Chart

6.2 Cube2- CubeProductCostHistory

6.2.1 Design

CubeProductCostHistory is nearly same with CubeProductListprice. CubeProductCostHistory cube also has Dim_Model, Dim_Unitmeasure, Dim_Subcategory dimensions. CubeProductCostHistory has Cost, VT_LB, VT_UB, TT_LB, TT_UB, Product_Id as measures and D1_Measure, D2_subcategory, D4_ProductMDC attributes for binding them to dimensions.

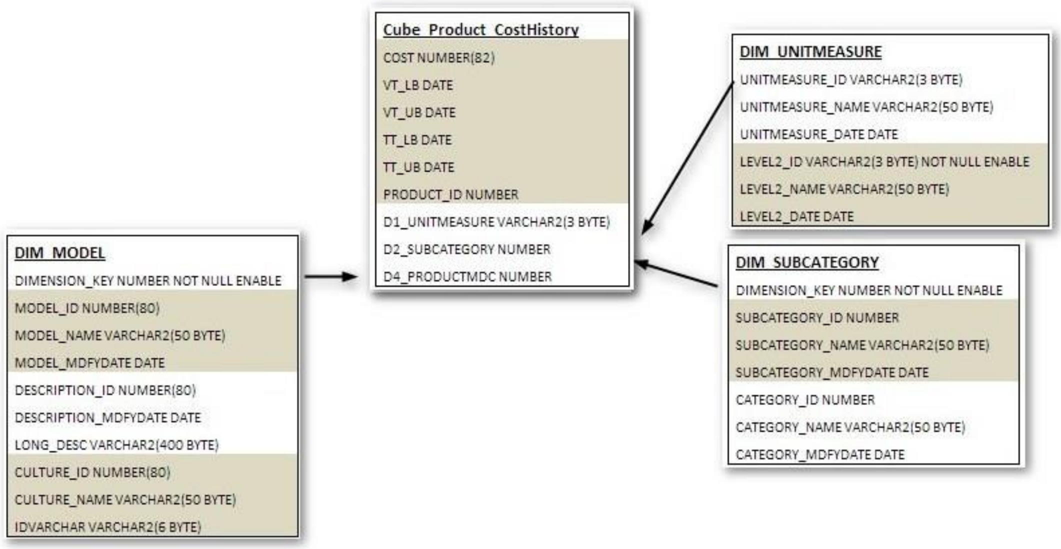

Cube_Product_CostHistory
COST NUMBER(82)
VT_LB DATE
VT_UB DATE
TT_LB DATE
TT_UB DATE
PRODUCT_ID NUMBER
D1_UNITMEASURE VARCHAR2(3 BYTE)
D2_SUBCATEGORY NUMBER
D4_PRODUCTMDC NUMBER
DIM_UNITMEASURE
UNITMEASURE_ID VARCHAR2(3 BYTE)
UNITMEASURE_NAME VARCHAR2(50 BYTE)
UNITMEASURE_DATE DATE
LEVEL2_ID VARCHAR2(3 BYTE) NOT NULL ENABLE
LEVEL2_NAME VARCHAR2(50 BYTE)
LEVEL2_DATE DATE
DIM_MODEL
DIMENSION_KEY NUMBER NOT NULL ENABLE
MODEL_ID NUMBER(80)
MODEL_NAME VARCHAR2(50 BYTE)
MODEL_MDFYDATE DATE
DESCRIPTION_ID NUMBER(80)
DESCRIPTION_MDFYDATE DATE
LONG_DESC VARCHAR2(400 BYTE)
CULTURE_ID NUMBER(80)
CULTURE_NAME VARCHAR2(50 BYTE)
IDVARCHAR VARCHAR2(6 BYTE)
DIM_SUBCATEGORY
DIMENSION_KEY NUMBER NOT NULL ENABLE
SUBCATEGORY_ID NUMBER
SUBCATEGORY_NAME VARCHAR2(50 BYTE)
SUBCATEGORY_MDFYDATE DATE
CATEGORY_ID NUMBER
CATEGORY_NAME VARCHAR2(50 BYTE)
CATEGORY_MDFYDATE DATE

Figure 6.27 Diagram of dimensional snowflake schema based on Cube_Product_CostHistory cube

CubeProductCostHistory cube as depicted in Figure 6.27 has snowflake structure and the same dimensions with CubeProductListprice. Same dimensions are used to create diffirent cubes.

6.2.2 Mapping

6.2.2.1 Mappinng CubeProductCostHistory

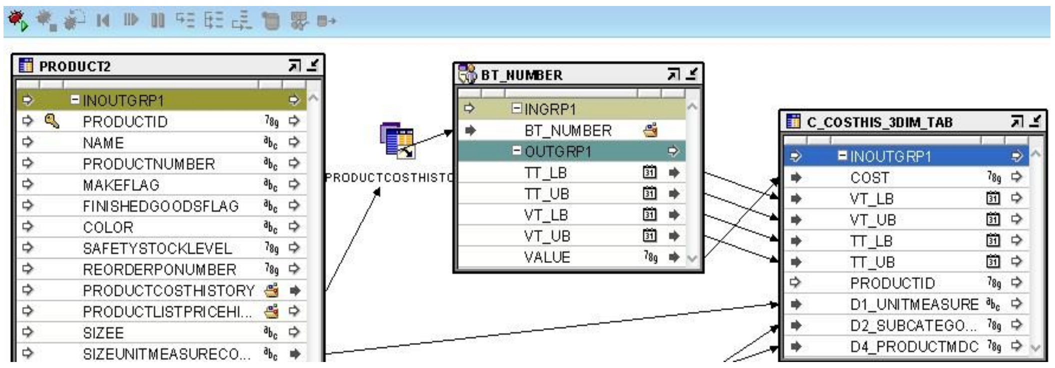


Figure 6.28 Mapping diagram of CubeProductCostHistory

6.2.3 Querying

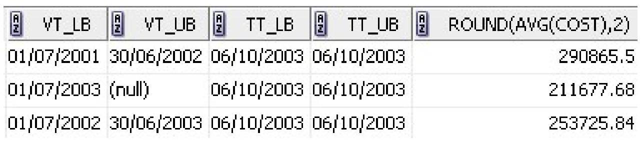
CubeProductCostHistory and its dimensions are used to solve the questions in this part.

Query 1: What is the average cost of Components according to years?

CubeProductCostHistory cube and Dim_Subcategory dimension is used to write solution query. The result according to years (2001, 2002, 2003) are also shown in Table 6.17.

Table 6.17 Query of question

```
SELECT VT_LB,VT_UB,TT_LB,TT_UB,ROUND(AVG(cost),2)
FROM cubeproductcosthistory WHERE dim_subcategory_key IN
(SELECT DIMENSION_KEY FROM DIM_SUBCATEGORY
WHERE category_id=(select productcategoryid
from productcategory_t WHERE NAMEE='Components'))
GROUP BY VT_LB,VT_UB,TT_LB,TT_UB;
```



VT_LB	VT_UB	TT_LB	TT_UB	ROUND(AVG(COST),2)
01/07/2001	30/06/2002	06/10/2003	06/10/2003	290865.5
01/07/2003	(null)	06/10/2003	06/10/2003	211677.68
01/07/2002	30/06/2003	06/10/2003	06/10/2003	253725.84

Figure 6.29 Query result screenshot

Table 6.18 Query of question written with relational objects

```
select StartDate AS VT_LB, EndDate AS VT_UL, ModifiedDate AS TTLB, ModifiedDate AS
TT_UB, AVG( PC.StandardCost) from Production.ProductCostHistory PC where ProductID in
(select ProductID from Production.Product where ProductSubcategoryID in(select
ProductSubcategoryID  from Production.ProductSubcategory  where ProductCategoryID= (select
ProductCategoryID from Production.ProductCategory where Name='Components'))) GROUP BY
StartDate,EndDate,ModifiedDate;
```

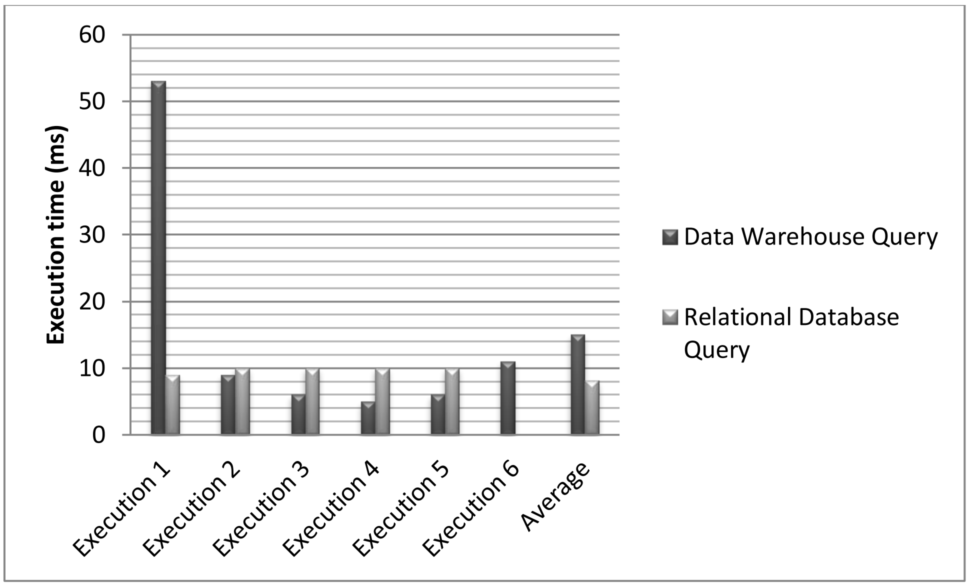
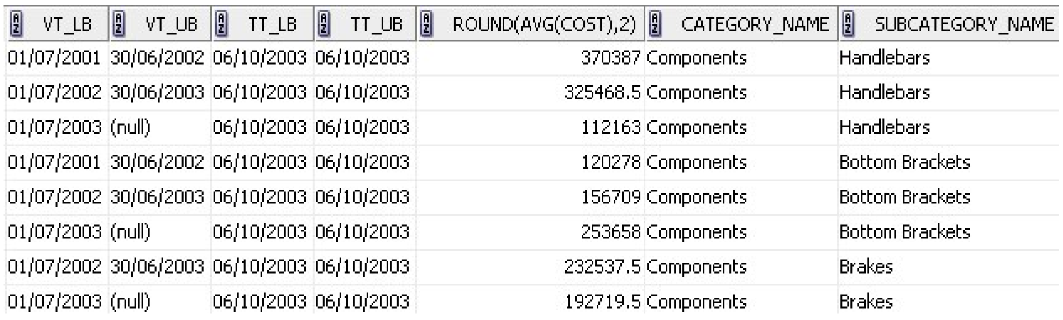


Figure 6.30 Query comparison chart

Query 2: What is the average list cost of all product according to years, categories and subcategories?

Table 6.19 Query of question

```
    SELECT VT_LB,VT_UB,TT_LB,TT_UB,ROUND(AVG(cost),2), (SELECT category_name FROM
DIM_SUBCATEGORY  WHERE dimension_key= dim_subcategory_key) CATEGORY_NAME,
    (SELECT subcategory_name FROM DIM_SUBCATEGORY  WHERE dimension_key=
dim_subcategory_key) SUBCATEGORY_NAME FROM cubeproductcosthistory GROUP BY
VT_LB,VT_UB,TT_LB,TT_UB,dim_subcategory_key ORDER BY dim_subcategory_key;
```



VT_LB	VT_UB	TT_LB	TT_UB	ROUND(AVG(COST),2)	CATEGORY_NAME	SUBCATEGORY_NAME
01/07/2001	30/06/2002	06/10/2003	06/10/2003	370387	Components	Handlebars
01/07/2002	30/06/2003	06/10/2003	06/10/2003	325468.5	Components	Handlebars
01/07/2003	(null)	06/10/2003	06/10/2003	112163	Components	Handlebars
01/07/2001	30/06/2002	06/10/2003	06/10/2003	120278	Components	Bottom Brackets
01/07/2002	30/06/2003	06/10/2003	06/10/2003	156709	Components	Bottom Brackets
01/07/2003	(null)	06/10/2003	06/10/2003	253658	Components	Bottom Brackets
01/07/2002	30/06/2003	06/10/2003	06/10/2003	232537.5	Components	Brakes
01/07/2003	(null)	06/10/2003	06/10/2003	192719.5	Components	Brakes

Figure 6.31 Query result screenshot

Table 6.20 Query of question written with relational objects

```
SELECT StartDate AS VT_LB, EndDate AS VT_UL,
PCP.ModifiedDate AS TTLB,PCP.ModifiedDate AS TT_UB,
PC.Name AS CATEGORY_NAME,
PSC.Name AS SUBCATEGORY_NAME, AVG(PCP.StandardCost)
FROM Production.ProductCostHistory PCP,
Production.ProductSubcategory PSC, Production.Product P,
Production.ProductCategory PC WHERE PCP.ProductID=P.ProductID
AND PSC.ProductSubcategoryID=P.ProductSubcategoryID
AND PC.ProductCategoryID=PSC.ProductCategoryID
GROUP BY StartDate,EndDate,PCP.ModifiedDate,PC.Name,PSC.Name;
```

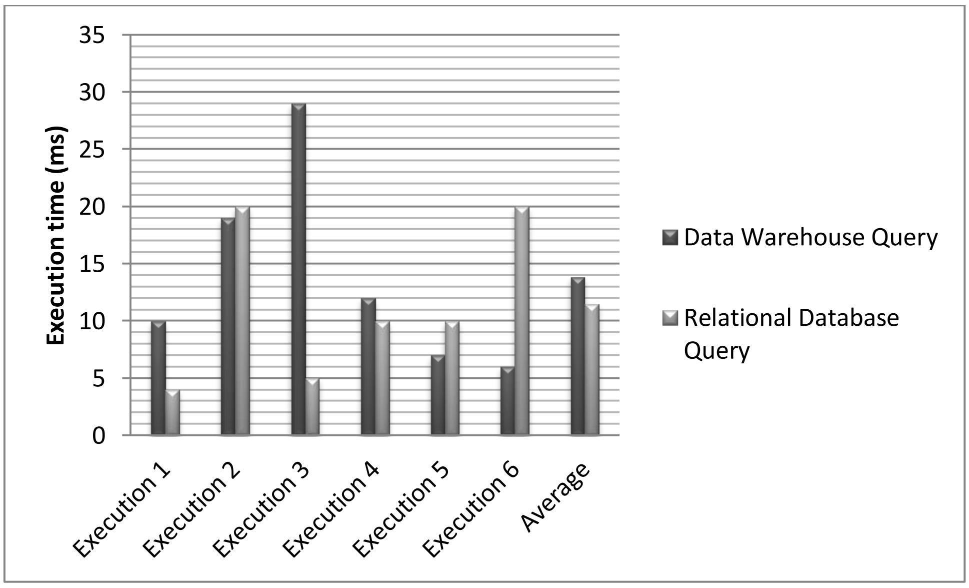
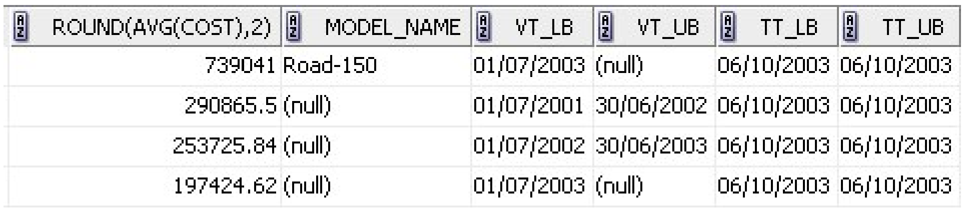


Figure 6.32 Query comparison chart

Query 3: What is the average cost for all years according to models?

Table 6.21 Query of question

```
SELECT ROUND(AVG(cost),2), (SELECT MODEL_NAME FROM DIM_MODEL
WHERE dimension_key=DIM_PRODUCTMDC_KEY) MODEL_NAME,VT_LB,VT_UB,
TT_LB,TT_UB FROM cubeproductcosthistory GROUP BY DIM_PRODUCTMDC_KEY,
VT_LB, VT_UB,TT_LB,TT_UB ORDER BY DIM_PRODUCTMDC_KEY,VT_LB,VT_UB;
```



ROUND(AVG(COST),2)	MODEL_NAME	VT_LB	VT_UB	TT_LB	TT_UB
739041	Road-150	01/07/2003	(null)	06/10/2003	06/10/2003
290865.5	(null)	01/07/2001	30/06/2002	06/10/2003	06/10/2003
253725.84	(null)	01/07/2002	30/06/2003	06/10/2003	06/10/2003
197424.62	(null)	01/07/2003	(null)	06/10/2003	06/10/2003

Figure 6.33 Query result screenshot

Table 6.22 Query of question written with relational objects

```
SELECT StartDate AS VT_LB, EndDate AS VT_UL, PCP.ModifiedDate AS
TTLB,PCP.ModifiedDate AS TT_UB,  PM.Name AS  MODEL_NAME,
AVG(PCP.StandardCost) FROM Production.ProductCostHistory PCP,
Production.ProductModel PM, Production.Product P WHERE PCP.ProductID=P.ProductID
AND PM.ProductModelID=P.ProductModelID GROUP BY
StartDate,EndDate,PCP.ModifiedDate,PM.Name;
```

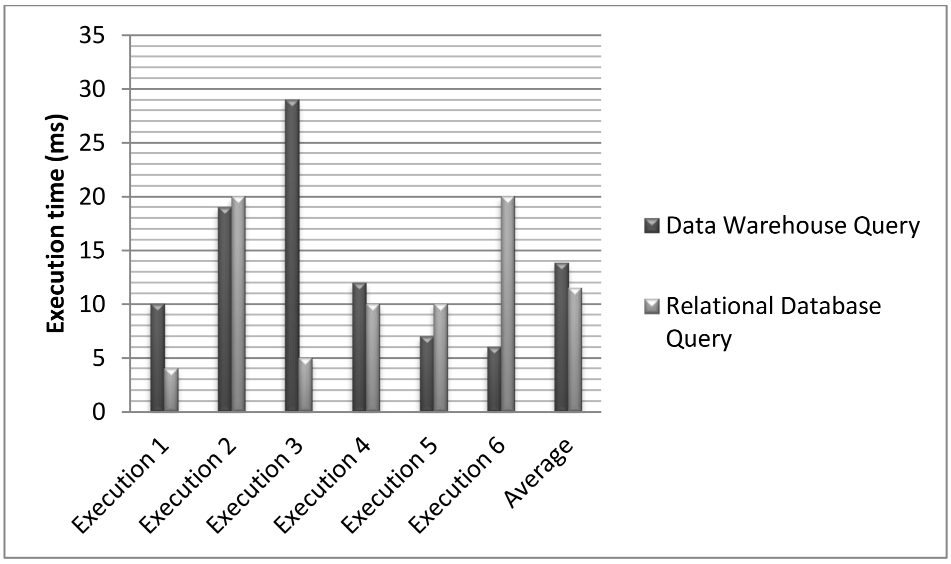
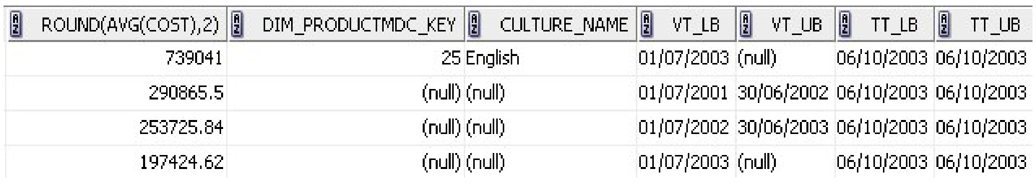


Figure 6.34 Query comparison chart

Query 4: What is the average cost for all years according to cultures?

Table 6.23 Query of question

```
SELECT ROUND(AVG(cost),2), dim_productmdc_key,
(SELECT CULTURE_NAME FROM DIM_MODEL
WHERE dimension_key=DIM_PRODUCTMDC_KEY) CULTURE_NAME,
VT_LB,VT_UB,TT_LB,TT_UB
FROM cubeproductcosthistory
GROUP BY DIM_PRODUCTMDC_KEY,VT_LB,VT_UB,TT_LB,TT_UB
ORDER BY DIM_PRODUCTMDC_KEY,VT_LB,VT_UB;
```



ROUND(AVG(COST),2)	DIM_PRODUCTMDC_KEY	CULTURE_NAME	VT_LB	VT_UB	TT_LB	TT_UB
739041	25	English	01/07/2003	(null)	06/10/2003	06/10/2003
290865.5	(null)	(null)	01/07/2001	30/06/2002	06/10/2003	06/10/2003
253725.84	(null)	(null)	01/07/2002	30/06/2003	06/10/2003	06/10/2003
197424.62	(null)	(null)	01/07/2003	(null)	06/10/2003	06/10/2003

Figure 6.35 Query result screenshot

Table 6.24 Query of question written with relational objects

```
select AVG( PC.StandardCost), C.Name
from Production.ProductCostHistory PC, Production.ProductModel PM, Production.Product P,
Production.ProductModelProductDescriptionCulture PMDC, Production.Culture C
where P.ProductModelID=PM.ProductModelID AND PC.ProductID=P.ProductID
AND PM.ProductModelID=PMDC.ProductModelID AND PMDC.CultureID=C.CultureID
AND  PC.StartDate>convert(DATE, '01.01.2000', 103) and PC.EndDate<convert(DATE,
'01.01.2004', 103)
AND PC.ProductID in (select ProductID from Production.Product
where ProductSubcategoryID in(select ProductSubcategoryID
from Production.ProductSubcategory  where ProductCategoryID=
(select ProductCategoryID from Production.ProductCategory where Name='Bikes'))) GROUP
BY  C.Name ;
```

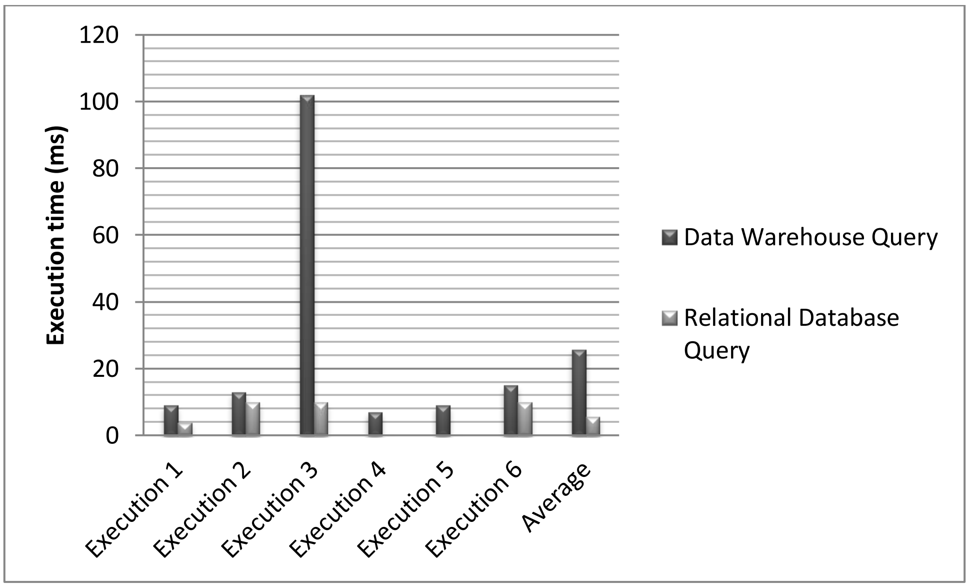
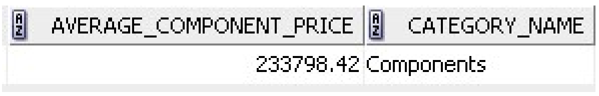


Figure 6.36 Query comparison chart

Query 5: What is the average list price of products According to Categories?

Table 6.25 Query of question

```
SELECT ROUND(AVG(COST),2) Average_Component_Price,
DIM_SUBCATEGORY.CATEGORY_NAME FROM cubeproductcosthistory,
DIM_SUBCATEGORY WHERE DIM_SUBCATEGORY.DIMENSION_KEY=
cubeproductcosthistory.DIM_SUBCATEGORY_KEY
GROUP BY DIM_SUBCATEGORY.CATEGORY_NAME
ORDER BY DIM_SUBCATEGORY.CATEGORY_NAME;
```



AVERAGE_COMPONENT_PRICE	CATEGORY_NAME
233798.42	Components

Figure 6.37 Query result screenshot

Table 6.26 Query of question written with relational objects

```
select AVG( PC.StandardCost), C.Name
from Production.ProductCostHistory PC, Production.ProductModel PM, Production.Product P,
Production.ProductModelProductDescriptionCulture PMDC, Production.Culture C
where P.ProductModelID=PM.ProductModelID AND PC.ProductID=P.ProductID
AND PM.ProductModelID=PMDC.ProductModelID AND PMDC.CultureID=C.CultureID
GROUP BY  C.Name ;
```

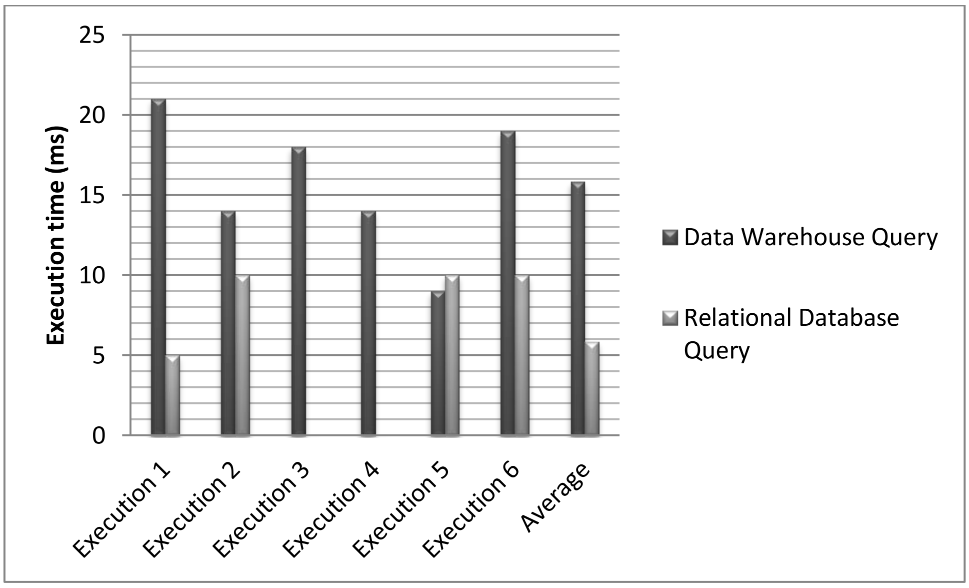
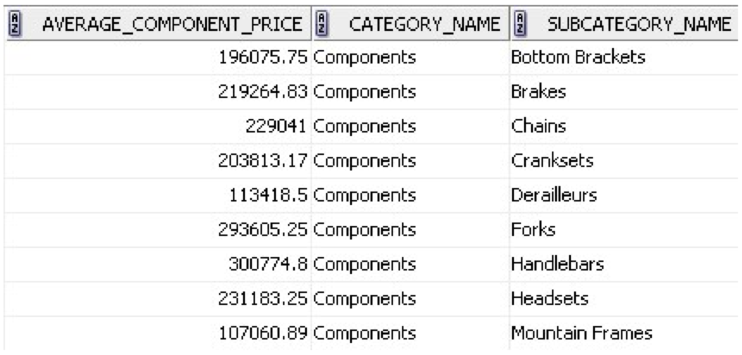


Figure 6.38 Query comparison chart

Query 6: What is the average list price of products According to Categories and SubCategories?

Table 6.27 Query of question

```
    SELECT ROUND(AVG(COST),2) Average_Component_Price,
    DIM_SUBCATEGORY.CATEGORY_NAME,DIM_SUBCATEGORY.SUBCATEGORY_NAME
FROM cubeproductcosthistory,DIM_SUBCATEGORY
    WHERE DIM_SUBCATEGORY.DIMENSION_KEY=
    cubeproductcosthistory.DIM_SUBCATEGORY_KEY
    GROUP BY DIM_SUBCATEGORY.CATEGORY_NAME,
DIM_SUBCATEGORY.SUBCATEGORY_NAME
    ORDER BY DIM_SUBCATEGORY.CATEGORY_NAME,
DIM_SUBCATEGORY.SUBCATEGORY_NAME;
```



AVERAGE_COMPONENT_PRICE	CATEGORY_NAME	SUBCATEGORY_NAME
196075.75	Components	Bottom Brackets
219264.83	Components	Brakes
229041	Components	Chains
203813.17	Components	Cranksets
113418.5	Components	Derailleurs
293605.25	Components	Forks
300774.8	Components	Handlebars
231183.25	Components	Headsets
107060.89	Components	Mountain Frames

Figure 6.39 Query result screenshot

Table 6.28 Query of question written with relational objects

```
select AVG( PC.StandardCost), C.Name , PM.Name
from Production.ProductCostHistory PC, Production.ProductModel PM, Production.Product P,
Production.ProductModelProductDescriptionCulture PMDC, Production.Culture C
where P.ProductModelID=PM.ProductModelID AND PC.ProductID=P.ProductID
AND PM.ProductModelID=PMDC.ProductModelID AND PMDC.CultureID=C.CultureID AND
PC.ProductID=P.ProductID
AND PC.ProductID in (select ProductID from Production.Product
where ProductSubcategoryID in(select ProductSubcategoryID
from Production.ProductSubcategory  where ProductCategoryID=
(select ProductCategoryID from Production.ProductCategory where Name='Components')))
GROUP BY  C.Name,PM.Name ;
```

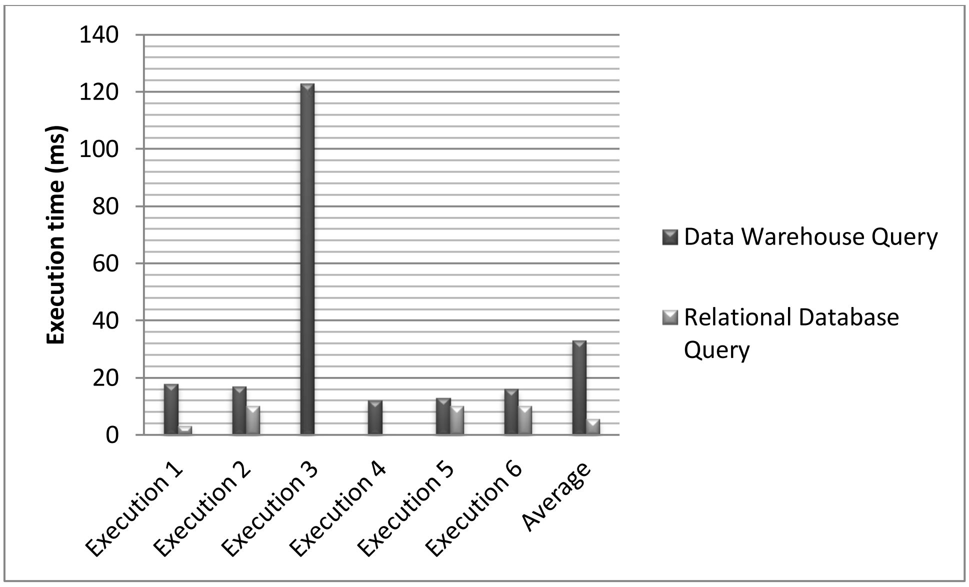


Figure 6.40 Query comparison chart

6.3 Cube3- CubeProductCostHistory2Dim

6.3.1 Design

While mathematical cubes have three dimensions, warehousing cubes can be designed by two, one, three, four or more dimensions. This CubeProductCostHistory2Dim cube is created by two dimensions. These dimensions are Dim_Model, Dim_Subcategory dimensions and Cost, VT_LB, VT_UB, TT_LB, TT_UB, ProdId measures.

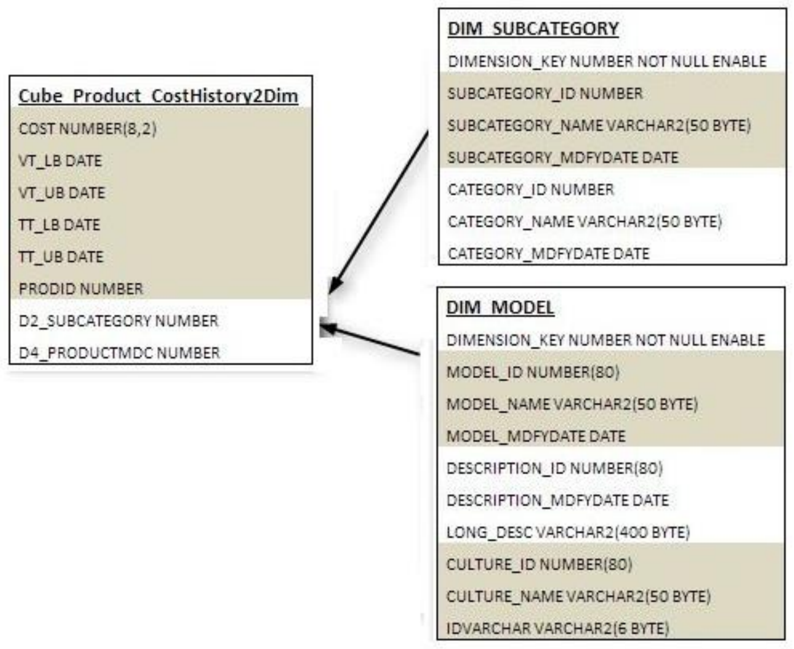

Cube_Product_CostHistory2Dim
COST NUMBER(8,2)
VT_LB DATE
VT_UB DATE
TT_LB DATE
TT_UB DATE
PRODID NUMBER
D2_SUBCATEGORY NUMBER
D4_PRODUCTMDC NUMBER
DIM_SUBCATEGORY
DIMENSION_KEY NUMBER NOT NULL ENABLE
SUBCATEGORY_ID NUMBER
SUBCATEGORY_NAME VARCHAR2(50 BYTE)
SUBCATEGORY_MDFYDATE DATE
CATEGORY_ID NUMBER
CATEGORY_NAME VARCHAR2(50 BYTE)
CATEGORY_MDFYDATE DATE
DIM_MODEL
DIMENSION_KEY NUMBER NOT NULL ENABLE
MODEL_ID NUMBER(80)
MODEL_NAME VARCHAR2(50 BYTE)
MODEL_MDFYDATE DATE
DESCRIPTION_ID NUMBER(80)
DESCRIPTION_MDFYDATE DATE
LONG_DESC VARCHAR2(400 BYTE)
CULTURE_ID NUMBER(80)
CULTURE_NAME VARCHAR2(50 BYTE)
IDVARCHAR VARCHAR2(6 BYTE)

Figure 6.41 Diagram of dimensional star schema based on Cube_Product_CostHistory2Dim cube

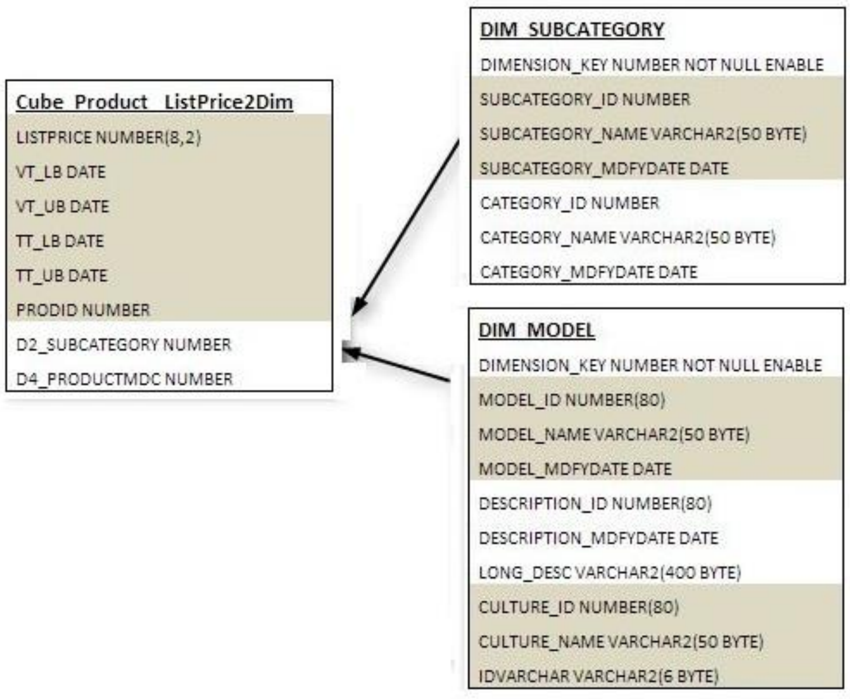
6.3.2 Mapping

CubeProductCostHistory2Dim is very similar with mapping of CubeProductCostHistory. The only difference is connecting two dimension attributes from product fact table.

6.4 Cube4- CubeProductListPrice2Dim

6.4.1 Design

This CubeProductListPrice2Dim cube is also created by two dimensions. These dimensions are Dim_Model, Dim_Subcategory dimensions and ListPrice, VT_LB, VT_UB, TT_LB, TT_UB, ProdId measures.



Cube_Product_ListPrice2Dim
LISTPRICE NUMBER(8,2)
VT_LB DATE
VT_UB DATE
TT_LB DATE
TT_UB DATE
PRODID NUMBER
D2_SUBCATEGORY NUMBER
D4_PRODUCTMDC NUMBER
DIM_SUBCATEGORY
DIMENSION_KEY NUMBER NOT NULL ENABLE
SUBCATEGORY_ID NUMBER
SUBCATEGORY_NAME VARCHAR2(50 BYTE)
SUBCATEGORY_MDFYDATE DATE
CATEGORY_ID NUMBER
CATEGORY_NAME VARCHAR2(50 BYTE)
CATEGORY_MDFYDATE DATE
DIM_MODEL
DIMENSION_KEY NUMBER NOT NULL ENABLE
MODEL_ID NUMBER(80)
MODEL_NAME VARCHAR2(50 BYTE)
MODEL_MDFYDATE DATE
DESCRIPTION_ID NUMBER(80)
DESCRIPTION_MDFYDATE DATE
LONG_DESC VARCHAR2(400 BYTE)
CULTURE_ID NUMBER(80)
CULTURE_NAME VARCHAR2(50 BYTE)
IDVARCHAR VARCHAR2(6 BYTE)

Figure 6.42 Diagram of dimensional star schema based on Cube_Product_ListPrice2Dim cube

6.4.2 Mapping

CubeProductListPrice2Dim is very similar with mapping of CubeProductListPrice. The only difference is connecting two dimension attributes from product fact table.

6.5 Cube5- Cube_Workorder_OrderQty

6.5.1 Design

A snowflake schema is shown in Figure 6.43. Arrows show foreign keys between tables. WorkOrder table is the fact table with Product, ProductSubCategory_t, ProductCategory_t dimension tables. Product dimension has three dimension levels ProductSubCategory and ProductCategory.

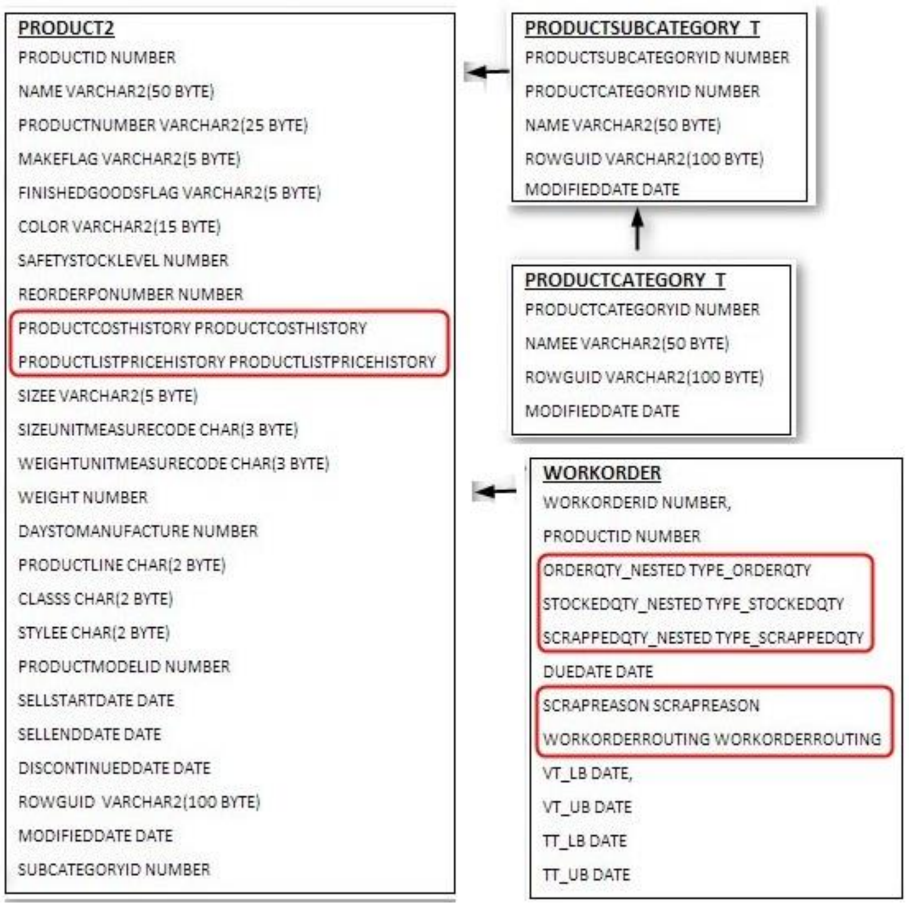


Figure 6.43 Snowflake schema based on Workorder fact table

The primary key in dimension table (Subcategory, Model, and UnitMeasure) is joined to the corresponding foreign key in the Workorder fact table. For example, Product.ProductId=Workorder. ProductId .

The cube model based on Workorder snowflake schema is constituted arount the Workorder fact object. Workorder cube have Product dimension. Cube table fact object includes attributes that correspond to the foreign keys in the fact table that are used to join the dimensions to the facts object. The fact object has five measures: VT_LB, VT_UB, TT_LB, TT_UB and Value. And has 1 attributes: D5_PRODUCT.

Product dimension references the following attributes:

- DIMENSION_KEY
- PRODUCT_ID
- PRODUCT_NAME

- PRODUCT_MDFDATE
- SUBCATEGORY_ID
- SUBCATEGORY_NAME
- SUBCATEGORY_MDFDATE
- CATEGORY_ID
- CATEGORY_NAME
- CATEGORY_MDFDATE

A join is created to connect dimension to the fact object.

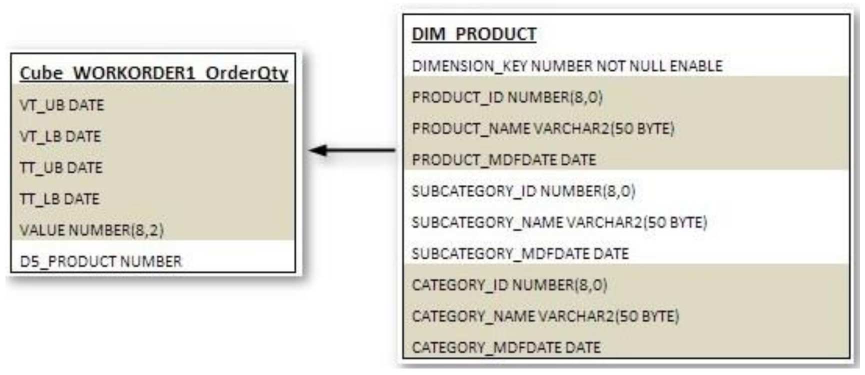


Figure 6.44 Dimensional relations diagram of objects

Product dimension has one hierarchy and three levels in it as shown in Figure 6.45.

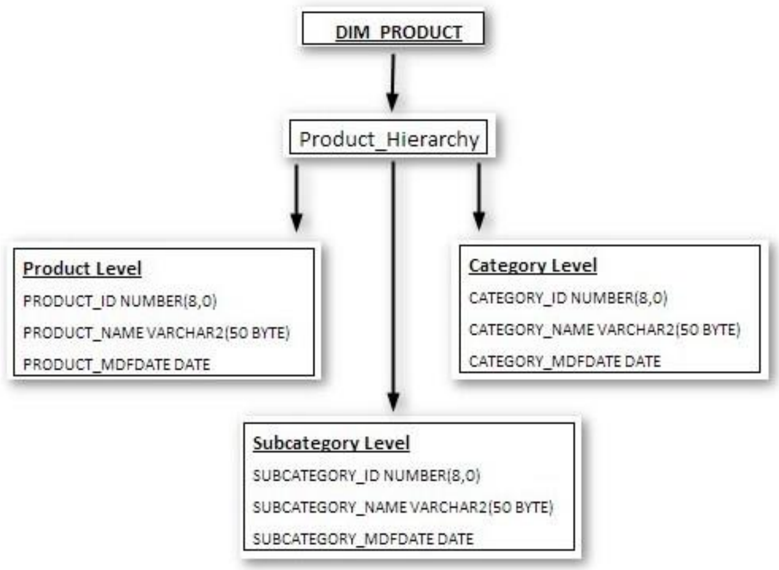


Figure 6.45 The hierarchy of DIM_PRODUCT

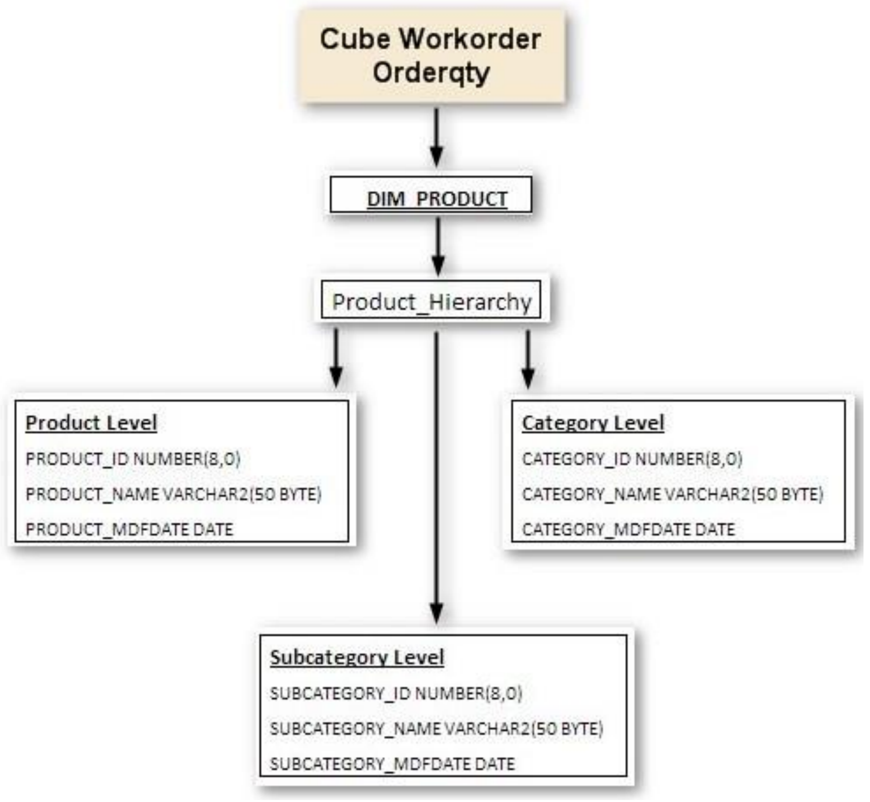


Figure 6.46 All objects in Cube_Workorder_OrderQty

6.5.2 Mapping

6.5.2.1 Mapping Dim_Product

When Dim_Product created, D5_PRODUCT_TAB is automatically created in the same form with dimension in warehousing repository. There are no data in this dimension table at first.

A mapping is designed to fill dimension table with data. Dim_Product's three levels come from three different tables. These are ProductSubCategory_T, ProductCategory_t and Product table. For loading three tables data into dimension table, using two joiner operators required.

Map is compiled and run. Dim_product is filled by data.

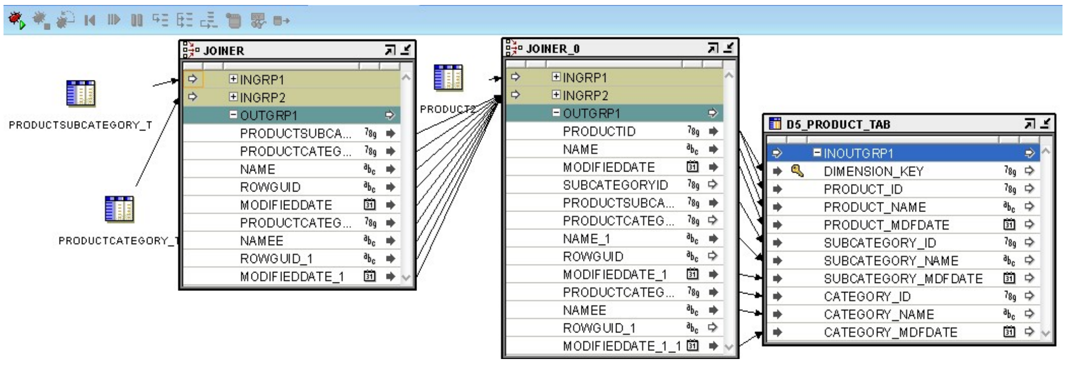


Figure 6.47 Mapping diagram of DIM_PRODUCT dimension

6.5.2.2 Mapping Cube_Workorder_OrderQty

When Cube_Workorder_OrderQty created, C_WORKORDER1_TAB is automatically created in the same form with cube in warehousing repository. There are no data in this dimension table at first.

There are four objects in Figure 6.48. The one on the left side is Workorder database table. The second one on left side is Varray Iterator. It converts OrderQty_Nested nested table to nested BT_number type. The third one is the Expand Object this object. This object expands all attributes in BT_number nested type. BT_number expand objects's outputs are connected to C_workorder1_tab's measure attributes. And ProductId column of Workorder table is connected to cube table's dimensional attribute.

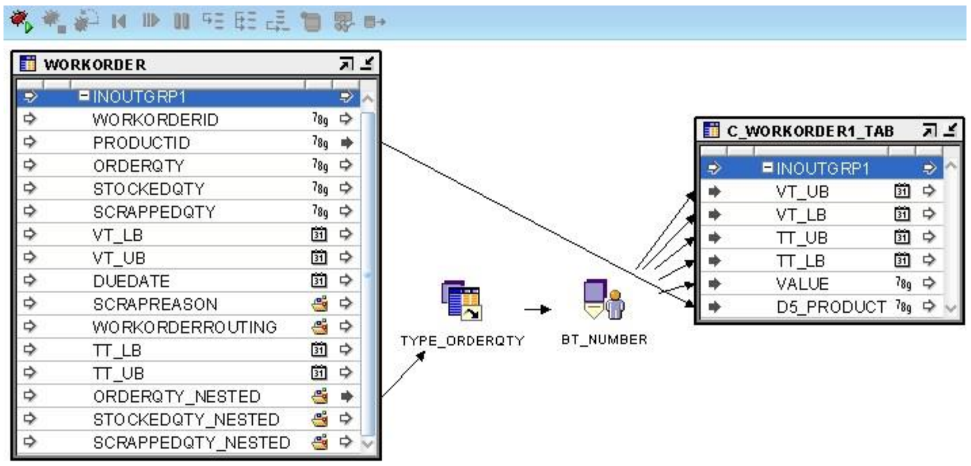


Figure 6.48 Mapping diagram of Cube_Workorder_OrderQty cube

6.5.3 Querying

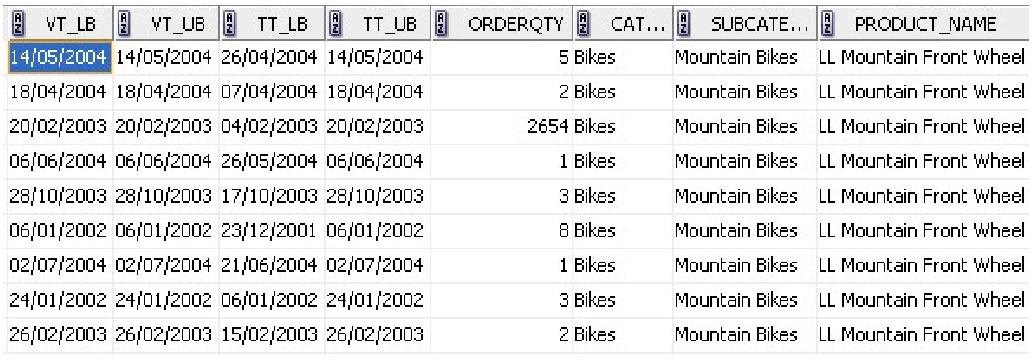
The queries about CUBE_WORKORDER_ORDERQTY and its dimension are given in this part. CUBE_WORKORDER_ORDERQTY has 24031 records.

Query 1: List of all work orders in ordered by according to category, subcategory and product_name.

CUBE_WORKORDER_ORDERQTY contains only one dimension, Dim_Product. Dim_Product has three levels. The question above requires accessing three levels of DimProduct.

Table 6.29 Query of question

```
    SELECT CUBE_WORKORDER_ORDERQTY.VT_LB, CUBE_WORKORDER_ORDERQTY.VT_UB,
CUBE_WORKORDER_ORDERQTY.TT_LB, CUBE_WORKORDER_ORDERQTY.TT_UB,
CUBE_WORKORDER_ORDERQTY.VALUE OrderQty, DIM_PRODUCT.CATEGORY_NAME,
DIM_PRODUCT.SUBCATEGORY_NAME, DIM_PRODUCT.PRODUCT_NAME FROM
CUBE_WORKORDER_ORDERQTY, DIM_PRODUCT WHERE DIM_PRODUCT.DIMENSION_KEY=
CUBE_WORKORDER_ORDERQTY.DIM_PRODUCT_KEY ORDER BY
DIM_PRODUCT.CATEGORY_NAME,DIM_PRODUCT.SUBCATEGORY_NAME,
DIM_PRODUCT.PRODUCT_NAME;
```



VT_LB	VT_UB	TT_LB	TT_UB	ORDERQTY	CAT...	SUBCATE...	PRODUCT_NAME
14/05/2004	14/05/2004	26/04/2004	14/05/2004	5	Bikes	Mountain Bikes	LL Mountain Front Wheel
18/04/2004	18/04/2004	07/04/2004	18/04/2004	2	Bikes	Mountain Bikes	LL Mountain Front Wheel
20/02/2003	20/02/2003	04/02/2003	20/02/2003	2654	Bikes	Mountain Bikes	LL Mountain Front Wheel
06/06/2004	06/06/2004	26/05/2004	06/06/2004	1	Bikes	Mountain Bikes	LL Mountain Front Wheel
28/10/2003	28/10/2003	17/10/2003	28/10/2003	3	Bikes	Mountain Bikes	LL Mountain Front Wheel
06/01/2002	06/01/2002	23/12/2001	06/01/2002	8	Bikes	Mountain Bikes	LL Mountain Front Wheel
02/07/2004	02/07/2004	21/06/2004	02/07/2004	1	Bikes	Mountain Bikes	LL Mountain Front Wheel
24/01/2002	24/01/2002	06/01/2002	24/01/2002	3	Bikes	Mountain Bikes	LL Mountain Front Wheel
26/02/2003	26/02/2003	15/02/2003	26/02/2003	2	Bikes	Mountain Bikes	LL Mountain Front Wheel

Figure 6.49 Query result screenshot

Table 6.30 Query of question written with relational objects

```
SELECT WO.StartDate VT_LB, WO.EndDate VT_UB, WO.ModifiedDate, WO.ModifiedDate
TT_UB, WO.OrderQty, PS.Name, PC.Name, P.Name FROM  Production.WorkOrder WO,
Production.Product P, Production.ProductSubcategory PS, Production.ProductCategory PC
WHERE P.ProductID=WO.ProductID AND P.ProductSubcategoryID=PS.ProductSubcategoryID
AND PS.ProductCategoryID=PC.ProductCategoryID ORDER BY PS.Name, PC.Name,P.Name
```

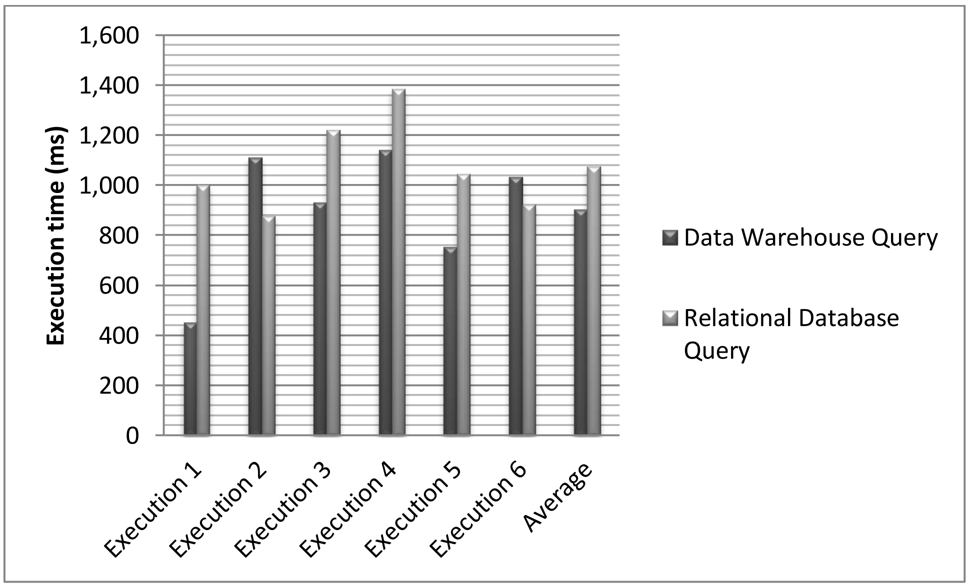


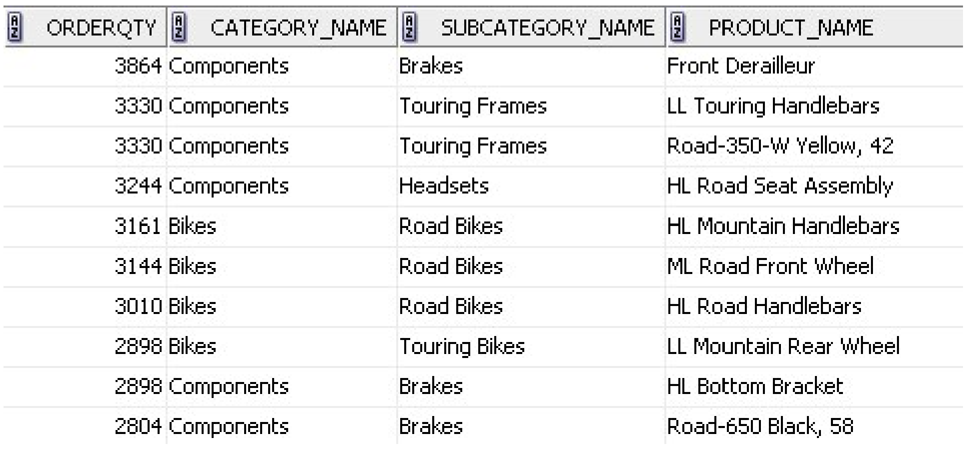
Figure 6.50 Query comparison chart

Query 2: List work orders in decreasing order according to category and subcategory.

The result of this query shows the most popular, highly preferred components and bikes together in decreasing order.

Table 6.31 Query of question

```
    SELECT * FROM(
    SELECT MAX(CUBE_WORKORDER_ORDERQTY.VALUE) OrderQty,
    DIM_PRODUCT.CATEGORY_NAME,
    DIM_PRODUCT.SUBCATEGORY_NAME, DIM_PRODUCT.PRODUCT_NAME
    FROM CUBE_WORKORDER_ORDERQTY, DIM_PRODUCT WHERE
DIM_PRODUCT.DIMENSION_KEY=
    CUBE_WORKORDER_ORDERQTY.DIM_PRODUCT_KEY
    GROUP BY DIM_PRODUCT.CATEGORY_NAME,
DIM_PRODUCT.SUBCATEGORY_NAME,
    DIM_PRODUCT.PRODUCT_NAME ORDER BY DIM_PRODUCT.CATEGORY_NAME ,
    DIM_PRODUCT.SUBCATEGORY_NAME,DIM_PRODUCT.PRODUCT_NAME)
    ORDER BY OrderQty DESC;
```



ORDERQTY	CATEGORY_NAME	SUBCATEGORY_NAME	PRODUCT_NAME
3864	Components	Brakes	Front Derailleur
3330	Components	Touring Frames	LL Touring Handlebars
3330	Components	Touring Frames	Road-350-W Yellow, 42
3244	Components	Headsets	HL Road Seat Assembly
3161	Bikes	Road Bikes	HL Mountain Handlebars
3144	Bikes	Road Bikes	ML Road Front Wheel
3010	Bikes	Road Bikes	HL Road Handlebars
2898	Bikes	Touring Bikes	LL Mountain Rear Wheel
2898	Components	Brakes	HL Bottom Bracket
2804	Components	Brakes	Road-650 Black, 58

Figure 6.51 Query result screenshot

The answer query of question can be written with relational database objects as statement in Table 6.32. As it appears, there are more join conditions than the one written with dimensional objects.

Table 6.32 Query of question written with relational objects

```
    SELECT X.ORDER_QTY, X.CATEGORY_NAME, X.PRODUCT_NAME,
X.SUBCATEGORY_NAME FROM (SELECT MAX(WO.OrderQty) AS ORDER_QTY,PC.Name AS
CATEGORY_NAME, PSC.Name AS SUBCATEGORY_NAME, P.Name AS PRODUCT_NAME FROM
Production.ProductSubcategory PSC, Production.Product P,  Production.ProductCategory PC,
Production.WorkOrder WO WHERE WO.ProductID=P.ProductID  AND PSC.ProductSubcategoryID=
P.ProductSubcategoryID AND PC.ProductCategoryID=PSC.ProductCategoryID GROUP BY PC.Name,
PSC.Name, P.Name ) X ORDER BY X.ORDER_QTY DESC
```

Both queries operate on same quantity of data. Performance changes with respect to system characteristics and querying methods. In our sample relational query is run on more strong system. Even so the data warehouse fetched results quicker. Figure 6.52 contains 6 measurements of these queries. The average of 6 measurements is shown as result 7. Data warehouse query's feedback is faster in average.

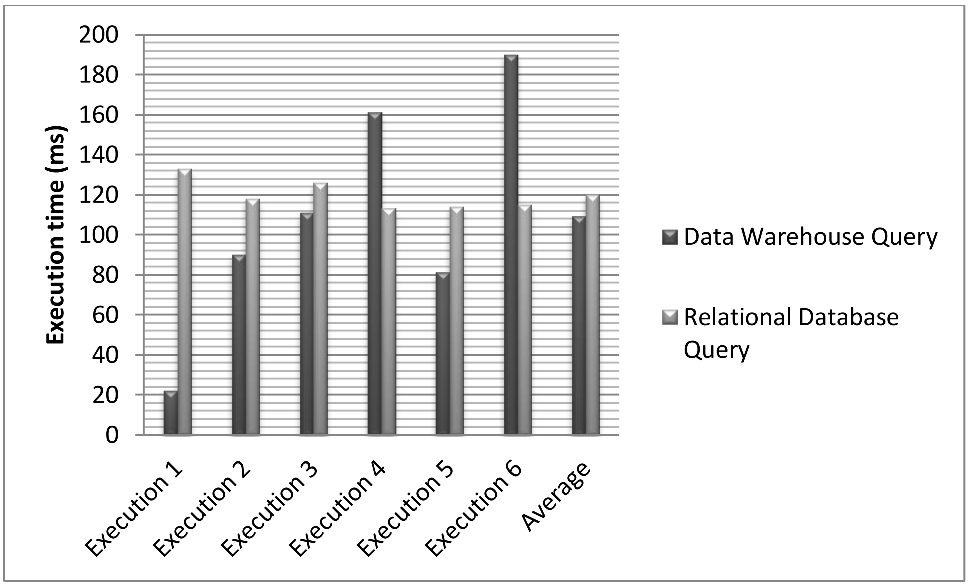


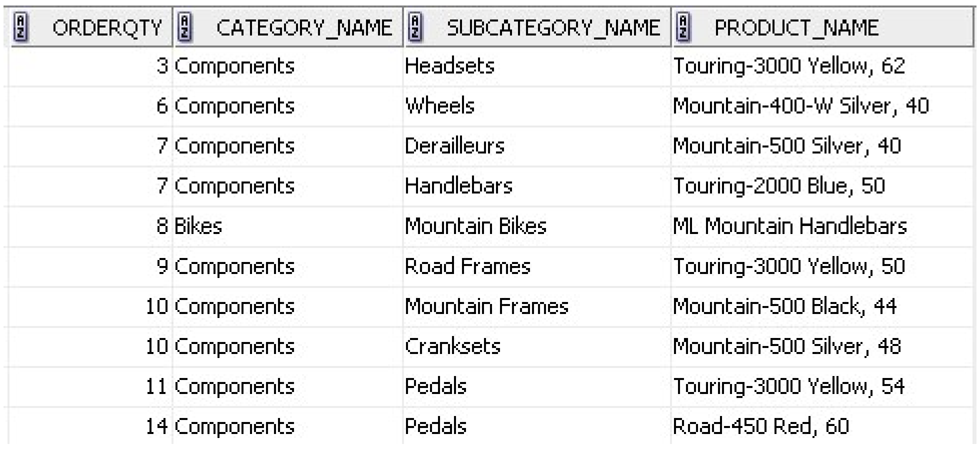
Figure 6.52 The chart contains six execution time in each system and an average measurement as seventh edge

Query 3: List work orders in increasing order according to category and subcategory.

The result of this query shows the least popular, least preferred components and bikes together in increasing order.

Table 6.33 Query of question

```
    SELECT * FROM(
    SELECT MAX(CUBE_WORKORDER_ORDERQTY.VALUE) OrderQty,
    DIM_PRODUCT.CATEGORY_NAME, DIM_PRODUCT.SUBCATEGORY_NAME,
    DIM_PRODUCT.PRODUCT_NAME FROM CUBE_WORKORDER_ORDERQTY,
DIM_PRODUCT
    WHERE DIM_PRODUCT.DIMENSION_KEY=
CUBE_WORKORDER_ORDERQTY.DIM_PRODUCT_KEY
    GROUP BY DIM_PRODUCT.CATEGORY_NAME,
DIM_PRODUCT.SUBCATEGORY_NAME,
    DIM_PRODUCT.PRODUCT_NAME ORDER BY DIM_PRODUCT.CATEGORY_NAME ,
    DIM_PRODUCT.SUBCATEGORY_NAME,DIM_PRODUCT.PRODUCT_NAME)
    ORDER BY OrderQty ASC;
```



ORDERQTY	CATEGORY_NAME	SUBCATEGORY_NAME	PRODUCT_NAME
3	Components	Headsets	Touring-3000 Yellow, 62
6	Components	Wheels	Mountain-400-W Silver, 40
7	Components	Derailleurs	Mountain-500 Silver, 40
7	Components	Handlebars	Touring-2000 Blue, 50
8	Bikes	Mountain Bikes	ML Mountain Handlebars
9	Components	Road Frames	Touring-3000 Yellow, 50
10	Components	Mountain Frames	Mountain-500 Black, 44
10	Components	Cranksets	Mountain-500 Silver, 48
11	Components	Pedals	Touring-3000 Yellow, 54
14	Components	Pedals	Road-450 Red, 60

Figure 6.53 Query result screenshot

Table 6.34 Query of question written with relational objects

```
SELECT MAX(WO.OrderQty) ORDER_QTY,
PS.Name SUBCATEGORY_NAME, PC.Name CATEGORY_NAME, P.Name
PRODUCT_NAME  FROM  Production.WorkOrder WO,
Production.Product P, Production.ProductSubcategory PS, Production.ProductCategory PC
WHERE P.ProductID=WO.ProductID AND P.ProductSubcategoryID=PS.ProductSubcategoryID
AND PS.ProductCategoryID=PC.ProductCategoryID GROUP BY PS.Name, PC.Name, P.Name
ORDER BY ORDER_QTY DESC
```

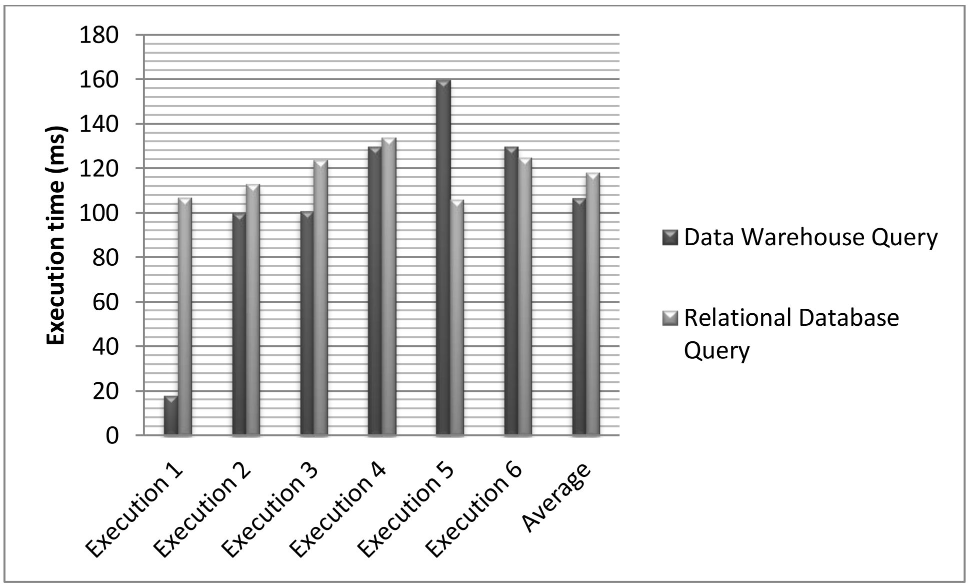


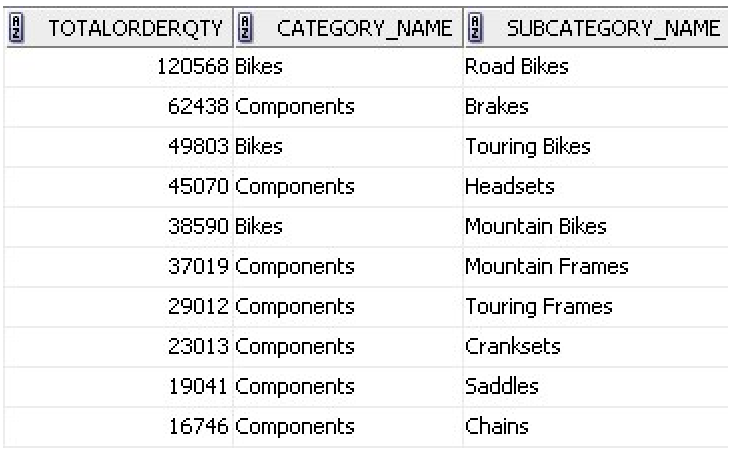
Figure 6.54 Query comparison chart

Query 4: List the total product workorder according to category and subcategory between 01/01/2002 and 31/12/2002 in decreasing order .

The result of query in Table 6.35 shows the most sold category and subcategory of products in decreasing order.

Table 6.35 Query of question

```
    SELECT * FROM
    (SELECT SUM(CUBE_WORKORDER_ORDERQTY.VALUE) TotalOrderQty,
    DIM_PRODUCT.CATEGORY_NAME, DIM_PRODUCT.SUBCATEGORY_NAME
    FROM CUBE_WORKORDER_ORDERQTY, DIM_PRODUCT
    WHERE DIM_PRODUCT.DIMENSION_KEY=
CUBE_WORKORDER_ORDERQTY.DIM_PRODUCT_KEY
    GROUP BY DIM_PRODUCT.CATEGORY_NAME,
DIM_PRODUCT.SUBCATEGORY_NAME
    ORDER BY DIM_PRODUCT.CATEGORY_NAME
,DIM_PRODUCT.SUBCATEGORY_NAME)
    ORDER BY TotalOrderQty DESC;
```



TOTALORDERQTY	CATEGORY_NAME	SUBCATEGORY_NAME
120568	Bikes	Road Bikes
62438	Components	Brakes
49803	Bikes	Touring Bikes
45070	Components	Headsets
38590	Bikes	Mountain Bikes
37019	Components	Mountain Frames
29012	Components	Touring Frames
23013	Components	Cranksets
19041	Components	Saddles
16746	Components	Chains

Figure 6.55 Query result screenshot

Table 6.36 Query of question written with relational objects

```
SELECT SUM(WO.OrderQty) ORDER_QTY,
PS.Name SUBCATEGORY_NAME, PC.Name CATEGORY_NAME, P.Name
PRODUCT_NAME  FROM  Production.WorkOrder WO,
Production.Product P, Production.ProductSubcategory PS, Production.ProductCategory PC
WHERE P.ProductID=WO.ProductID AND P.ProductSubcategoryID=PS.ProductSubcategoryID
AND PS.ProductCategoryID=PC.ProductCategoryID AND
WO.StartDate>convert(DATE, '01/01/2002', 103) and WO.EndDate<convert(DATE,
'31/12/2002', 103)GROUP BY PS.Name, PC.Name, P.Name
ORDER BY ORDER_QTY DESC
```

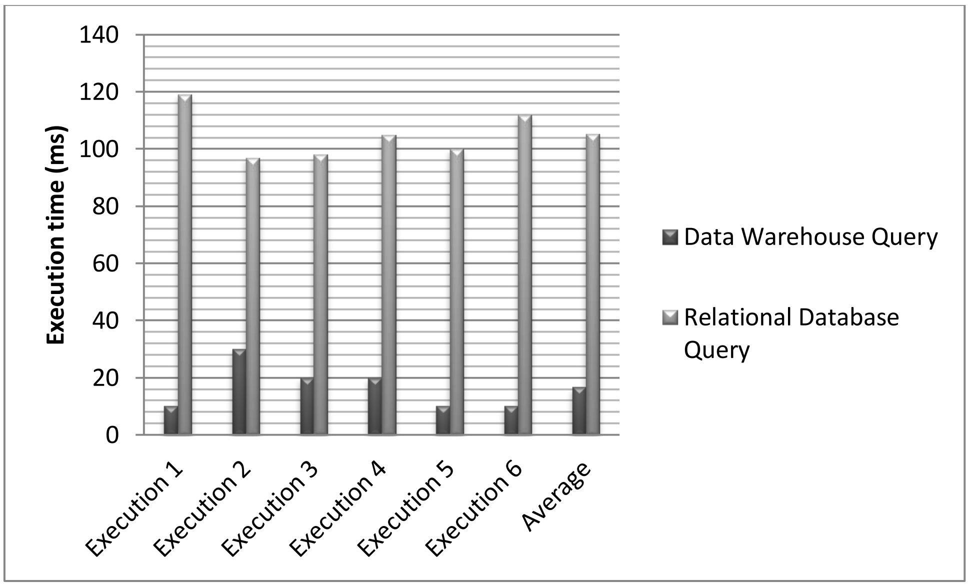


Figure 6.56 Query comparison chart

CHAPTER SEVEN
CONCLUSION AND DISCUSSION

Excessive amount of data exist in web and database systems. People share their personal information on web pages. It is very important to process data and get a grip on data. Results provide offering personal advertising and services. Some companies generate their strategies according to results of the processed data. Data warehousing is a favorite reporting system on reporting field. Today many enterprises use data warehousing technologies. Getting more effective and more expeditious results on reports are reasons of using warehousing systems. In data warehousing, reports are fetched from dimensional databases. Multiple tables are associated as levels. Less join statements are required and less background operations occur. Hence warehouse reports achieve high performance.

Temporal depth is very important on data warehousing systems. When a value or cell is updated, old data must be stored. Reporting can be required according to previous time periods. Bitemporal structure is determined to use for storing temporal data. Bitemporal structure stores validity period and transaction period of data. When a cell is updated, old data's valid time upper bound component is filled by today’s date, validity of old data ends. At the same time valid time lower bound component of new data is filled by today’s date, validity of new data begins. A new row is added to table when a column updates. If so many updates happen, tables may expand unrestrainedly. In order to avoid uncontrolled expansion, tables must be created in semi-structured form. Updated column and its bitemporal attributes are stored as nested table. Bitemporal columns are; validity lower and upper bounds: VT_LB (Valid time lower bound), VT_UB (Valid time upper bound). Transaction lower and upper bounds: TT_LB (Transaction time lower bound), TT_UB (Transaction time upper bound). Updated column, VT_LB, VT_UB, TT_LB, TT_UB compose a nested type. Temporal nested column is added to table as nested table. When an update occurs in temporal column, only a row is added to nested table. Semi-structured form prevents repeating unchanged columns.

In this study, a database module which is specially prepared in SQL server is transferred to Oracle database and semi-structured, bitemporal data warehouse is designed. For supplying semi-structured form nested tables and nested types are used. While tables in relational database use 34688 KB disk space, dimensional tables use only 11392 KB. A dimensional database is designed. Queries are run in this dimensional database and results are discussed. Results are explained and supported with diagrams and charts. Results and reports can be used for specifying the strategies of Corporates. Also the results are used to analyze conditions. In this application, a bicycle corporate data is used. Products can be reported with respect to their categories, models etc.

Data in data warehouse may be analyzed according to different time intervals with the flexibility of bitemporal structure. Purchasing habits of customers may be analyzed in time line. The efficient periods of year can be figured out. These and similar beneficial conditions may be analyzed faster than the other systems, together with the benefits of data warehouse and semi-structured data storage. Outcomes help corporates for determining their strategy.

REFERENCES

Adamson, C., & Venerable, M. (1998). *Data warehousing design solutions* (286-425). New York: Wiley Press.

AdventureWorks database summary (n.d). Retrieved March 1, 2013, from http://www.dbdesc.com/output_samples/htmlbrowse_AdventureWorks.html

Atay, C. E., & Tansel, A. U. (2009). *Bitemporal databases: Modeling and implementation*. Germany: VDM Publishing.

Atay, C.E. (2008). *Nested bitemporal relational data model*. Retrieved January, 25, 2013 from http://www.google.com.tr/books?hl=en&lr=&id=zO2-CpgVt-MC&oi=fnd&pg=PR4&dq=nested+bitemporal+relational+data+model&ots=Q-ILAOcEVB&sig=EQ31wUmDWoEj_Dxt4r7q9Je72h4&redir_esc=y

Bebel, B., Eder, J., Koncilia, C., Morzy, T., & Wrembel, R. (2004). Creation and management of versions in multiversion data warehouse. *Proceedings of the* The *Association for Computing Machinery Symposium on Applied Computing*, 717–723.

Ben-Zvi, J. (1982). *The time relational model*. Los Angeles: University Microfilms.

Benitez, E., Guerrero, C., & Adiba, M. (2003). The WHES approach to data warehouse evolution. *Digital journal e-Gnosis*. Retrieved February 14, 2013, from http://www.e-gnosis.udg.mx.

Bhargava, G., & Gadia, S. (1993). Relational database systems with zero information loss. *Institute of Electrical and Electronics Engineers Transactions on Knowledge and Data Engineering*. 5(1).

Blaschka, M., Sapia, C., & Höfling, G. (1999). On schema evolution in multidimensional databases. *Data Warehousing and Knowledge Discovery Conference*, 153–164, Florence: Springer.

Combi, C., Oliboni, B. , & and Pozzi, G. (2009). Modeling and querying temporal semistructured data warehouses. *New Trends in Data Warehousing and Data Analysis Annals of Information Systems*, 1-25, New York: Springer.

Gadia, S. K. (1988). A homogeneous relational model and query languages for temporal databases. *Association for Computing Machinery Transactions on Database Systems*, 418-448.

Hurtado, C. A., Mendelzon, A. O., & Vaisman, A. A. (1999). Updating OLAP dimensions. *2nd Institute of Electrical and Electronics Engineers – Data Warehousing and Olap Workshop*, 60–66.

Inmon, W. (2002). *Building the data warehouse*, United States: Wiley.

Janet, E., Ramirez , R., & Guerrero, E. B. (2006). A model and language for bitemporal schema versioning in data warehouses. *Proceedings of the 15th International Conference on Computing*, 309-314.

Jensen, C. S., Soo, M. D., & Snodgrass, R. T. (1994). Unifying temporal data models via a conceptual model. *Information Systems* (513-547). USA: Elsevier Press.

Koncilia, C. (2003). A bitemporal data warehouse model. *CAiSE Short Paper Proceedings, Central Europe Workshop Proceedings*, Retrieved December, 28, 2012 from http://www.CEUR-WS.org.

Malinowski, E., & Zimanyi, E. (2006). A conceptual solution for representing time in data warehouse dimensions. *3rd Asia-Pacific Conference on Conceptual Modeling*, 45–54.

Martin, C., & Abello, A. (2002). The data warehouse: A temporal database. *Jornadas de Ingeniería del Software y Bases de Datos*, 675-684.