

## Advanced & Distributed Databases - Assessment Criteria and Feedback Sheet

Name: J.Peters

Mark: 70

Higher levels of achievement are described on the RHS of the grid. Each level subsumes the previous level. Each of the three criteria below contributes a third to the overall mark.

Primarily, Analysis & Design and Learning Outcome CS2				
Does not reach required threshold.	Analysis and design artefacts meet most scenario requirements, but with some deficiencies.	Described analysis and design artefact meet scenario requirements and using appropriate tools and methods.	Rationale for design decisions explained (inc. most higher-level requirements).	Comprehensive, fully justified and documented analysis and design artefacts meet all requirements in full.
Primarily, Implementation and Learning Outcome P&PS4				
Does not reach required threshold.	Implementation artefacts meet most scenario requirements, but with some deficiencies.	Documented implementation and test artefacts meet scenario requirements.	Discussion of encountered implementation issues and solutions (inc. most higher-level requirements).	Comprehensive and fully documented implementation and test artefacts meet all requirements in full.
Primarily, Evaluation Report and Learning Outcomes CS2, K&U1 and P&PS3				
Does not reach required threshold.	Evaluation report meets most requirements but with some deficiencies. Evidence that relevant background reading has been undertaken.	Richer and referenced evaluation against some appropriate criteria. Some logical recommendations developed.	Well written evaluation report against full set of criteria and full set of recommendations developed. Most higher-level requirements addressed. References clearly underpin the evaluation throughout.	Comprehensive and critical evaluation report addressing all requirements, and based on extensive research.

Notes :

1. Normally, artefacts for all of the 5 themes must be submitted and reach the required threshold.
2. Each of the activities indicates work which could contribute to higher-level achievement.

Additional Comments (particularly for borderline and exception cases)

A well written and comprehensive report.

# Evaluation Report

BSc (Hons) Software Engineering

Author: Jan David Peters

Unit Leader: Alastair Monger

*Advanced and Distributed Databases*

30<sup>th</sup> April 2010

Southampton Solent University

Faculty of Technology

## Table of Contents

Table of Contents .....	1
1. Exploiting DBMS Data Models and Server Functionality .....	3
1.1. Task 1) Development of Product and Price Information.....	3
1.2. Task 2) External Availability of Product and Price information.....	4
1.3. Evaluation of Technologies, Tools and Methods .....	8
1.3.1. Price History and Database Triggers .....	8
1.3.2. Oracle SQL/XML .....	8
1.3.3. Oracle XMLType .....	8
1.3.4. SQL Developer .....	9
1.4. References .....	9
2. Accessing and Manipulating Data in Applications .....	10
2.1. Design, implement and test a transaction processing program .....	10
2.1.1. Software Design .....	10
2.1.2. Implementation.....	12
2.1.3. Source Code .....	13
2.1.4. Test.....	19
2.2. Evaluation of Technologies, Tools and Methods .....	20
2.2.1. Java or PL/SQL .....	20
2.2.2. JDBC and Prepared Statements .....	21
2.2.3. Eclipse IDE .....	21
2.2.4. Oracle Application Express (APEX).....	21
2.3. References .....	22
3. Improving Data Access by Data Distribution and Replication .....	23
3.1. Task 2) Creation and Testing of a SQL View .....	23
3.2. Task 3) Production of a Distributed Database Design Solution .....	26
3.3. Evaluation of Technologies, Tools and Methods .....	27
3.3.1. Concept of Distributed Databases and Replication .....	27
3.3.2. Database Design and Effort.....	28
3.3.3. Applicability in Praxis .....	29
3.4. References .....	29
4. Multi-Dimensional Modelling and Analysis for Decision Support .....	30
4.1. Task 1) Implementation of Analysing Data for Decision Support .....	30
4.1.1. Preparing the Data using Microsoft Excel.....	33
4.1.2. Preparing the Data using Oracle SQL statements.....	34
4.1.3. Evaluate Data and Derive Decision Support .....	36
4.2. Task 2) Development of a Dimension Model for the HR Database .....	38
4.3. Evaluation of Technologies, Tools and Methods .....	41
4.3.1. Online Analytical Processing (OLAP) .....	41
4.3.2. OLAP with Microsoft Excel and Oracle SQL.....	41

---

4.3.3.	MySQL Workbench 5.1.....	42
4.4.	References.....	42
5.	Mining Databases for Decision Support.....	43
5.1.	Task 1) Design and Implementation of an example Market Basket Analysis .....	43
5.1.1.	Choice of Data Mining Tools and Technique .....	43
5.1.2.	Background Discussion (→ Business / Data Understanding).....	43
5.1.3.	Preparation of required Data (→ Data Preparation).....	44
5.1.4.	Performing the Analysis using WEKA Data Mining Tool (→ Modelling).....	47
5.1.5.	Significance of discovered Rules for Decision Support (→ Evaluation).....	53
5.2.	Evaluation of Technologies, Tools and Methods .....	53
5.2.1.	CRISP Methodology.....	53
5.2.2.	WEKA Data Mining Tool .....	54
5.3.	References .....	54
6.	Mondrian OLAP Tool .....	56
6.1.	Introduction to Mondrian.....	56
6.2.	Scenario .....	57
6.3.	Analysis, Design, Implementation and Test .....	59
6.3.1.	Analysis.....	59
6.3.2.	Design.....	59
6.3.3.	Implementation.....	60
6.3.4.	Test.....	63
6.4.	Evaluation of Technologies, Tools and Methods .....	66
6.5.	References .....	67

## 1. Exploiting DBMS Data Models and Server Functionality

### 1.1. Task 1) Development of Product and Price Information

The first step is to create a table for storing the data that are collected. A possible solution would be the following SQL-Statement:

```
CREATE TABLE product_pricehistory (  
    PricehistoryID number NOT NULL Primary key,  
    OccuredAt date not null,  
    UpdatedBy varchar(100) not null,  
    ProductID number not null,  
    ProductName varchar2(50) not null,  
    PriceOld number(8,2) null,  
    PriceNew number(8,2) not null  
);
```

#### Listing 1: Creating a new Database Table for Price History

Every row is identified by a unique ID which is automatically generated using a sequence:

```
CREATE SEQUENCE priceIdSeq MINVALUE 1 MAXVALUE 99999999999999999999999999999999  
START WITH 1 INCREMENT BY 1 CACHE 20;
```

#### Listing 2: Creating a new Sequence for automatic Incrementation of ID

Information about the date of change is necessary according to the requirements specified. Additional information about the Oracle-User could be useful for tracking reasons. The fields PriceOld and PriceNew record the actual values. The field PriceOld has to be "null", because if a new record is inserted, it has no old value for the price. Logging these changes is realised by creating the following trigger:

```
CREATE OR REPLACE TRIGGER auditProductPriceHistory  
AFTER UPDATE OR INSERT OR DELETE OF list_price ON product_information  
FOR EACH ROW  
BEGIN  
    IF UPDATING THEN  
        INSERT INTO product_pricehistory(PricehistoryID, OccuredAt, updatedBy,  
        ProductID, ProductName, PriceOld, PriceNew)  
        VALUES (priceIdSeq.nextval, SYSDATE, USER, :new.Product_ID,  
        :new.Product_Name, :old.list_price, :new.list_price);  
    end if;  
    IF INSERTING THEN  
        INSERT INTO product_pricehistory(PricehistoryID, OccuredAt, updatedBy,  
        ProductID, ProductName, PriceOld, PriceNew)  
        VALUES (priceIdSeq.nextval, SYSDATE, USER, :new.Product_ID,  
        :new.Product_Name, null, :new.list_price);  
    End if;  
    IF DELETING THEN  
        DELETE FROM product_pricehistory WHERE ProductID = :old.Product_Id;  
    END IF;
```

```
END;
```

**Listing 3: Creating a Trigger for to React on Updates, Inserts and Deletes**

The created trigger is covering all DML statements (Burleson Consulting 2009), while the cases “UPDATING” and “INSERTING” deal with the price data, “DELETING” a product\_information causes removing of all information about price history for the given record.

Uniqueness is provided by using the unique key constraint (Tech on the net 2009). At first, it is necessary to make sure that there are no identical product names in the table:

```
SELECT product_name, count(*) FROM product_information GROUP BY
product_name HAVING count(*) > 1;
```

**Listing 4: Check if there are any duplicate Keys**

In this case, a record had to be renamed like this:

```
UPDATE product_information SET product_name = 'Client ISO CP - S V4.0'
WHERE product_id = '2416';
```

**Listing 5: Updating a Record which has had a non-unique Key**

If the table is prepared well, it will be altered by using the following SQL-statement:

```
ALTER TABLE product_information MODIFY (PRODUCT_NAME varchar2(50 Byte) not
null) ADD CONSTRAINT product_information_uk1 UNIQUE ("PRODUCT_NAME")
ENABLE;
```

**Listing 6: Changing the Table for to not allow Duplicate Keys**

It is important to use a constraint identifier with a maximum of 30 characters and to spell the affected column name right. Inserting an invalid record tests the added functionality.

## **1.2. Task 2) External Availability of Product and Price information**

Providing the price history information can be implemented in different ways. The data is already stored in the table “product\_pricehistory”, therefore it can be exported to XML by using the following SQL-statement:

```
SELECT      ProductID, XMLElement(
            "Product",
            XMLAttributes(
                productid AS "id", productname as "Name",
                TO_CHAR(occuredat, 'DD/MM/YYYY HH:Mi:SS') AS "DateChanged"
            ),
            XMLElement("OldPrice", PriceOld),
            XMLElement("NewPrice", PriceNew)
        ) AS PriceHistoryXmlData
FROM        product_pricehistory
```

```
WHERE ProductID = 2416
ORDER BY OccuredAt DESC;
```

**Listing 7: Statement for Retrieving Product Data in an XML Structure**

PRODUCTID	PRICEHISTORYXMLDATA
2416	<Product id="2416" Name="Client ISO CP - S V4.0" DateChanged="27/10/2009 01:42:31"><OldPrice>61</OldPrice><NewPrice>71</NewPrice></Product>
2416	<Product id="2416" Name="Client ISO CP - S V4.0" DateChanged="27/10/2009 01:12:27"><OldPrice>51</OldPrice><NewPrice>61</NewPrice></Product>
2416	<Product id="2416" Name="Client ISO CP - S V4.0" DateChanged="27/10/2009 01:00:32"><OldPrice>41</OldPrice><NewPrice>51</NewPrice></Product>

**Figure 1: Evidence of the Result from Listing 7**

Oracle displays date data types in different formats, depending on server and user settings. An applicable way for a consistent date format is the usage of the function TO\_CHAR (Tech on the net 2009). The SQL-statement above produces XML for one history record only, therefore it is necessary to modify it for getting a structured XML document for a specified product. It seems to be a good idea to use the "CREATE FUNCTION" command (Oracle 2009). An example is listed here:

```
CREATE OR REPLACE
FUNCTION getProductPriceHistoryXML (productid NUMBER) return String IS
strXML string(32000);
BEGIN
  /* Get the product name for the given product-ID */
  FOR nameRow IN (SELECT product_name FROM product_information WHERE
                  product_id = getProductPriceHistoryXML.productid)
  LOOP
    strXml := '<ProductHistory ProductID="' ||
              getProductPriceHistoryXML.productid || '" ProductName="'
              || nameRow.product_name || '">';
  END LOOP;
  /* Get XML Information about Price history */
  FOR curRow IN (SELECT XMLElement(
                    "PriceChanged",
                    XMLAttributes(TO_CHAR(occuredat, 'DD/MM/YYYY HH:Mi:SS')
                    As "DateChanged"),
                    XMLElement("OldPrice", PriceOld),
                    XMLElement("NewPrice", PriceNew)
                  ) AS PriceHistoryXmlData
                FROM product_pricehistory
                WHERE ProductID = getProductPriceHistoryXML.productid
                ORDER BY OccuredAt DESC)
  LOOP
    strXml := strxml || curRow.PriceHistoryXmlData.getStringVal();
  END LOOP;
  /* Close XML-Root-Element */
  strXml := strxml || '</ProductHistory>';
```

```

RETURN strXML;
END;

```

**Listing 8: Creating a Function that returns an XML Structure**

The result will be of the data type “String”, the XML-Element is converted into a string using the “getStringVal()” method (Oracle 2009). The function will be callable using the following statements:

```

SET SERVEROUTPUT ON;
DECLARE
resultValue string(32000);
BEGIN
    resultValue:=getProductPriceHistoryXML(2424);
    dbms_output.put_line(resultValue);
END;

```

**Listing 9: Example Query for the Created Function**

```

<ProductHistory ProductID="2424" ProductName="CDW 12/24"><PriceChanged
DateChanged="27/10/2009
01:42:31"><OldPrice>241</OldPrice><NewPrice>251</NewPrice></PriceChanged>
<PriceChanged DateChanged="27/10/2009
01:12:27"><OldPrice>231</OldPrice><NewPrice>241</NewPrice></PriceChanged>
<PriceChanged DateChanged="27/10/2009
01:00:32"><OldPrice>221</OldPrice><NewPrice>231</NewPrice></PriceChanged>
</ProductHistory>
PL/SQL procedure successfully completed.

```

**Listing 10: Example Result for the Query Listing 9 (copied from ISQL Plus)**

It would also be possible to use the Oracle XMLType, which enables to store and update an XML document in a table, column or row (Ritchie 2008). This XMLType data can be queried using the “Extract” and “ExtractValue” functions in order to retrieve data. A solution for the given scenario might be implemented as follows:

A table is created; the structure includes a column for the ProductID and an XMLType column for the XML document

```

CREATE TABLE priceHistoryXML (
    ProductID int PRIMARY KEY,
    PriceHistory XMLType
)

```

**Listing 11: Create a Table for Storing the Price History with XMLType**

The next step is to create a trigger that listens on the “list price” of the “product\_information” table. The trigger works similar to the one created in Task 1) but



inserts or updates the XML document stored in the XMLType column for the specified ProductID instead of adding a record to the "auditProductPriceHistory" table:

```
DECLARE
newprice CLOB := '<prices>
                <price new="14.00" old="12.00" date="12/01/2010"/>
                </prices>';
BEGIN
  INSERT INTO priceHistoryXML VALUES (2416, XMLType(newprice));
END;
```

#### Listing 12: Inserting a new Product and Price in the XML History Table

```
DECLARE
newprice CLOB := '<price new="16.00" old="14.00" date="15/01/2010"/>';
BEGIN
  UPDATE pricehistoryxml SET PriceHistory = INSERTXMLBEFORE(PriceHistory,
    '/prices/price[1]', XMLType(newprice))
  WHERE ProductID = 2416;
END;
```

#### Listing 13: Updating the XML Price History by adding a new Price

The retrieval of data for providing the price history to the customers could be performed by the following query:

```
SELECT XMLElement("Product",
  XMLAttributes( I.Product_id AS "ID",
                I.Product_name AS "Name",
                I.Product_description AS "Description"),
  H.PriceHistory
) AS PriceHistoryXmlData
FROM product_information I
LEFT OUTER JOIN pricehistoryxml H ON i.product_id = H.productid
WHERE I.product_id = 2416
```

#### Listing 14: Querying the created Table

```
PRICEHISTORYXMLDATA
<Product ID="2416" Name="Client ISO CP - S V4.0" Description="ISO Communication Package add-on
license for additional SPNIX V4.0 client."><prices> <price new="16.00" old="14.00" date="15/01/2010"/>
<price new="16.00" old="14.00" date="15/01/20 10"/> <price new="14.00" old="12.00" date="12/01/2010"/>
</prices> </Product>
```

Figure 2: The Result for the Query in Listing 14

### **1.3. Evaluation of Technologies, Tools and Methods**

#### **1.3.1. Price History and Database Triggers**

The usage of triggers is regarded as essential for the implementation of the price history because they could easily be added to the existing data model using PL/SQL. Another approach to the problem would cause a change in the data model: Instead of storing the former prices in a separate table, the structure requires a change in order to store all prices (including the current price) in an own table because it can be implemented as a 1:n relation. Alternatively, the history of prices could be stored within a data warehousing solution.

Basically, all operations which are performed by triggers could also be performed by the logic of software systems which are accessing the database. These applications could be responsible for validations and storage of related data.

#### **1.3.2. Oracle SQL/XML**

Oracle provides an XML component since the release of the Oracle 10g database (Ritchie 2008, p.349). SQL/XML contains a set of commands which can generate XML documents from relational database tables (Ritchie 2008, p.351). For experienced SQL developers, the appliance of SQL/XML is regarded as easy, also because of its in detail documentation (Oracle 2010).

SQL/XML mainly provides retrieving and converting functions such as "XMLElement" and "XMLAttributes". With SQL/XML it is possible to develop complex XML documents within an acceptable time and effort. Another advantage is the good server performance which leads to short query times.

Oracle's XML functionality is a powerful extension and regarded as a valuable for database developers as well as companies. It can save the cost because it is no longer necessary to develop XML converters programmatically. On the other hand it needs to be said that the scope of SQL/XML does not exceed the possibilities which manually programmed solutions have. There are some constraints existing that might influence the decision if to use SQL/XML or not, for example merging multiple rows into one XML document by using a single SELECT-command.

#### **1.3.3. Oracle XMLType**

The XMLType is an approach to store and retrieve XML data in tables and columns. It can be regarded as the opposite of SQL/XML because XML data can be queried in order to receive a traditional result set. Using the XMLType allows a software system to establish central data

storage for XML documents. Oracle supports querying XMLType columns with XPath (W3Schools.com 2010) and therefore uses a best practice for handling XML documents.

Beside the querying and storage of whole XML documents, there also is the possibility to manipulate parts of the XML document. This might be helpful when dealing with large documents in terms of increasing performance.

#### **1.3.4. SQL Developer**

Oracle SQL Developer is client software for Oracle database development written in Java. Besides running SQL scripts, it is possible to create PL/SQL statements, develop database functionality like triggers and generate reports. SQL Developer is a recommended tool for accessing Oracle databases. Another alternative approach for querying the database is iSQL Plus, which is a web-based application.

### **1.4. References**

BURLESON CONSULTING, 2009. *Oracle DML statements* [online]. Available: [http://www.dba-oracle.com/t\\_dml\\_statements.htm](http://www.dba-oracle.com/t_dml_statements.htm) [accessed 22 October 2009]

ORACLE, 2010. *How to use Oracle XML functions* [online]. Available: [http://www.oradev.com/xml\\_functions.jsp](http://www.oradev.com/xml_functions.jsp) [accessed 19th March 2010]

ORACLE, 2009. *CREATE FUNCTION* [online]. Available: [http://download.oracle.com/docs/cd/B19306\\_01/server.102/b14200/statements\\_5009.htm](http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/statements_5009.htm) [accessed 27 October 2009]

TECH ON THE NET, 2009. *Oracle/PLSQL: Unique Constraints* [online]. Available: <http://www.techonthenet.com/oracle/unique.php> [accessed 27 October 2009]

W3SCHOOLS.COM, 2010, *XPath Syntax* [online]. Available: [http://www.w3schools.com/XPath/xpath\\_syntax.asp](http://www.w3schools.com/XPath/xpath_syntax.asp) [accessed 17th March 2010]

## 2. Accessing and Manipulating Data in Applications

### 2.1. Design, implement and test a transaction processing program

#### 2.1.1. Software Design

Initially, a basic analysis of involved data structures and the expected result has been undertaken. Beside the class "Action" which represents the database table structure, a class for accessing the database and processing transactions is needed. The program needs no graphical user interface (GUI) because it is merely regarded as a batch procedure. However, some information about progress of transactions and function calls might be helpful for understanding the performed operations.

Beside the GUI, certain software architecture is unnecessary because of the small scope of this program; therefore a class diagram, basic sequence diagram and a program flowchart suffice for specifying the programs structure.

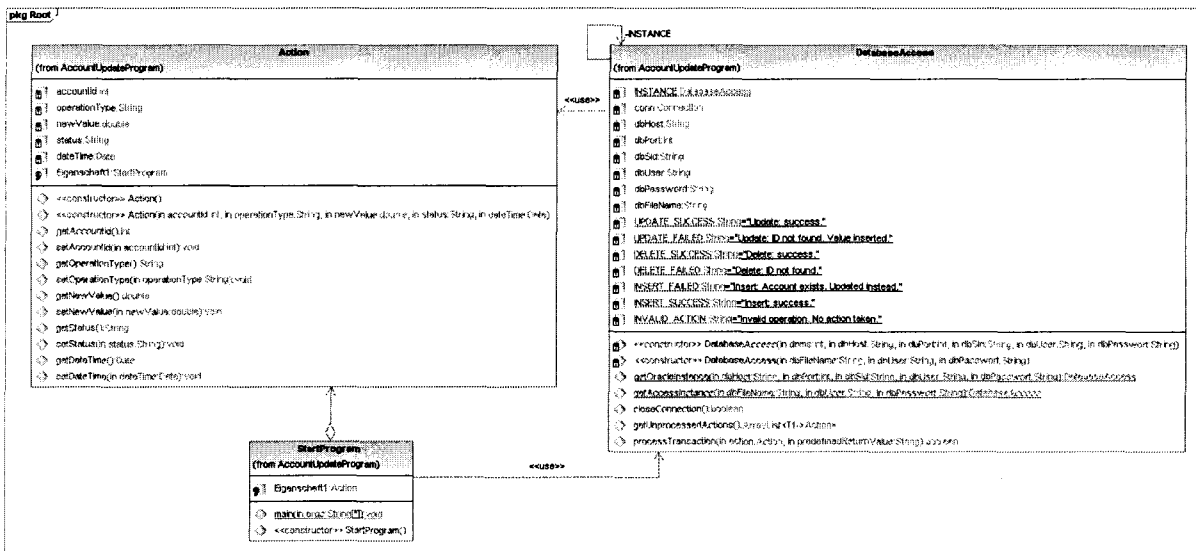
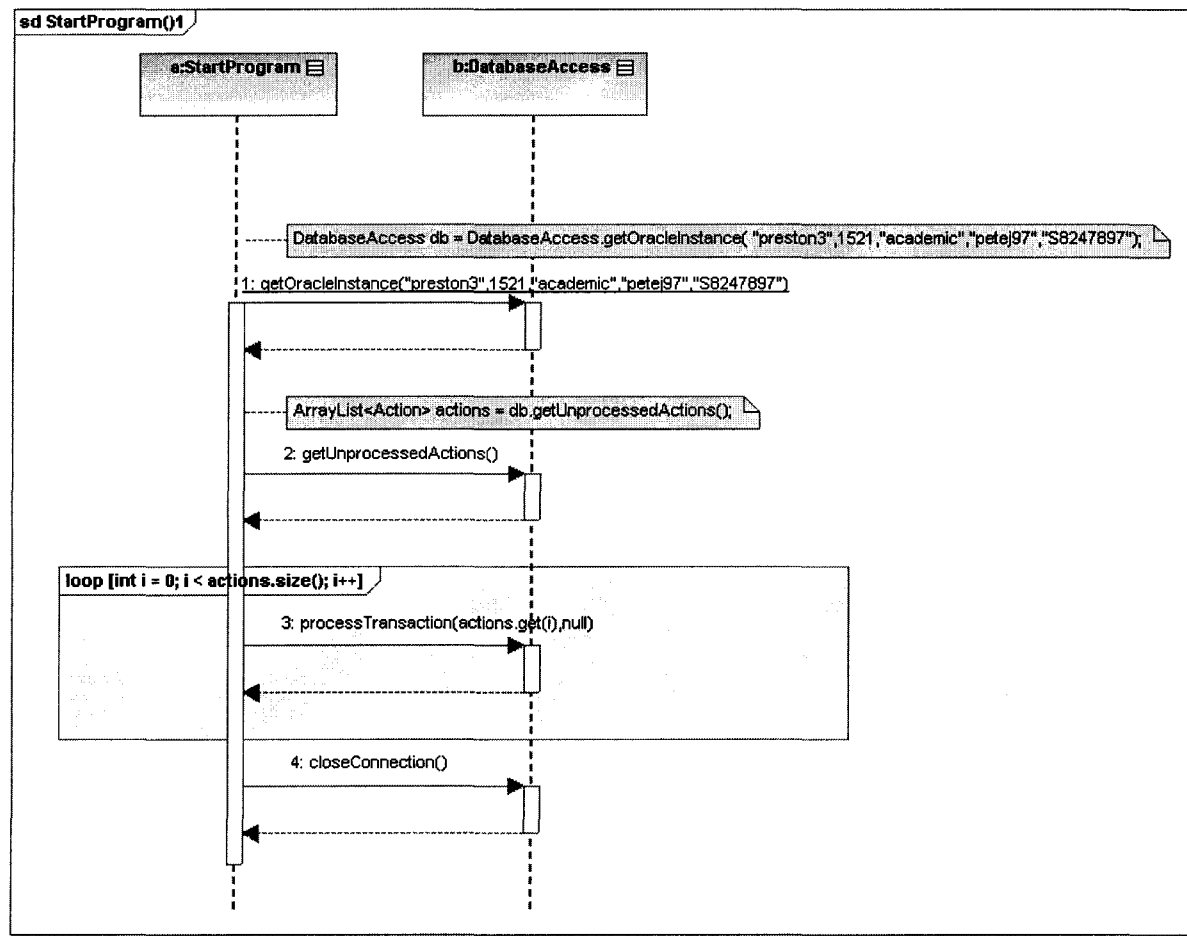


Figure 3: Class Diagram

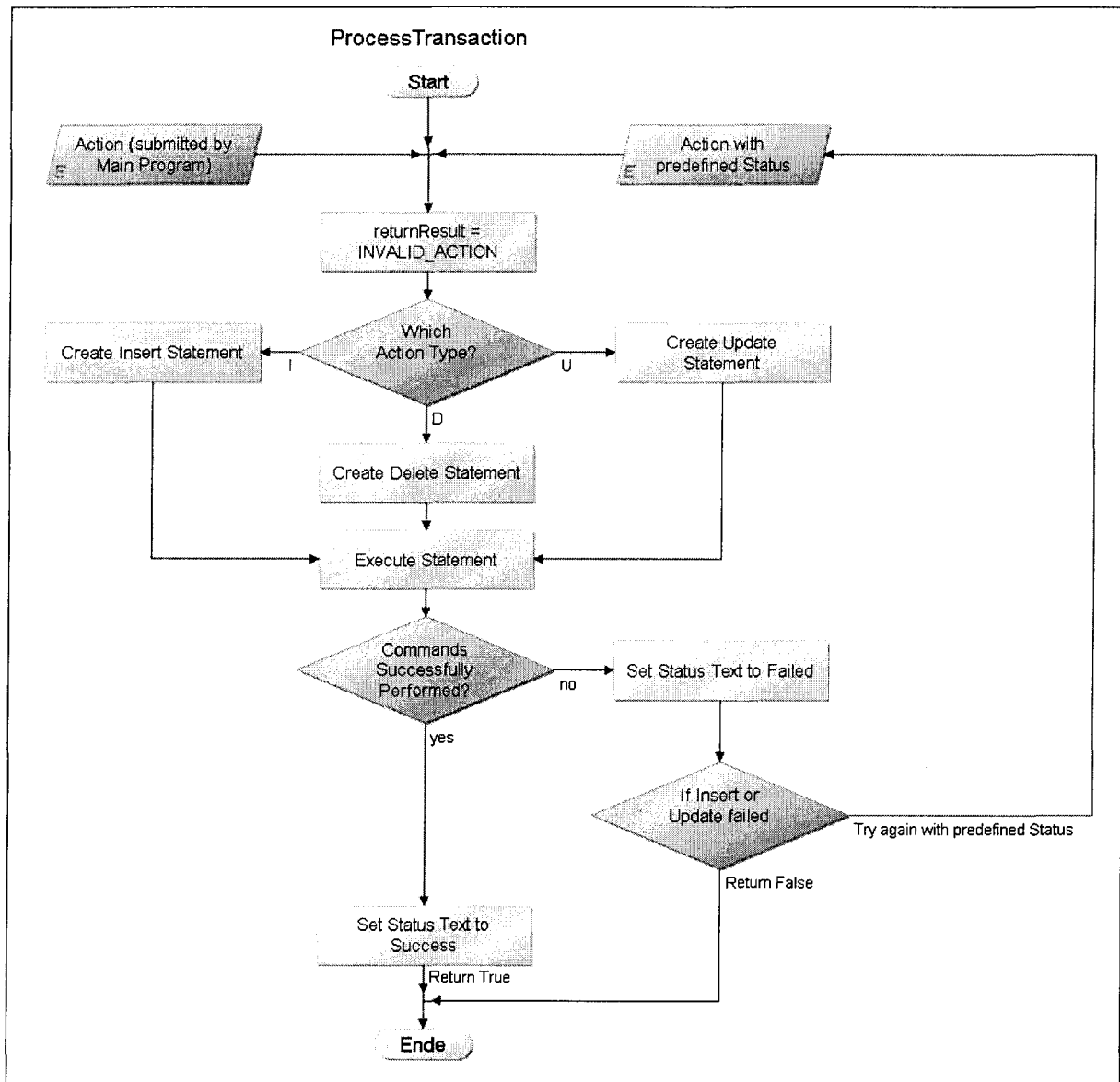
*DBMS independence?*



Generated by UModel

www.altova.com

Figure 4: Sequence Diagram "Connecting to Database and Loading unprocessed Transactions"



**Figure 5: Program Flowchart for Processing a Transaction**

### 2.1.2. Implementation

The implementation in Java is based on the derived UML diagrams and the flowchart for the main algorithm. All database operations are performed by the "Database" class which uses JDBC. This class is implemented in "Singleton pattern" to make sure that there is only one instance of this class (Larman 2005, p.443) and therefore only one corresponding connection to the database at any time. It is developed to allow easy migration to other DBMS by just adding an additional private constructor in combination with a public "getInstanceFor...()" - function. The only thing that needs to be adjusted is the connection string and its parameters to provide a more flexible solution. For example, connecting to Microsoft Access databases is already implemented and tested.

Generally, the database is queried using so called Prepared Statements which are available in the “java.sql” package. All possible errors are handled and outputted, because this solution is regarded as a prototype.

The main algorithm is contained in the “processTransaction(…)” function. It is responsible for creating the SQL statements and monitors the success. A speciality of this function is the ability for recursive calls in the case of failed database transaction. For example:

An account should be updated, but it does not exist. The error is recognised and the function is called again with the additional parameter “UPDATE\_FAILED” which tells the function to insert this new account into the database.

### 2.1.3. Source Code

```
/**
 * Action.java - the class represents a row of the action table
 *               and contains getter and setter methods
 * @author Jan Peters
 * @date 2009-12-05
 * @version 1.0
 */

import java.sql.Date;

public class Action {

    private int accountId;
    private String operationType;
    private double newValue;
    private String status;
    private Date dateTime;

    public Action() {

    }

    public Action( int accountId, String operationType,
                  double newValue, String status, Date dateTime) {
        this.accountId = accountId;
        this.operationType = operationType;
        this.newValue = newValue;
        this.status = status;
        this.dateTime = dateTime;
    }

    public int getAccountId() {
        return accountId;
    }

    public void setAccountId(int accountId) {
        this.accountId = accountId;
    }

    public String getOperationType() {
        return operationType;
    }

    public void setOperationType(String operationType) {
        this.operationType = operationType;
    }

    public double getNewValue() {
```

```

        return newValue;
    }
    public void setNewValue(double newValue) {
        this.newValue = newValue;
    }
    public String getStatus() {
        return status;
    }
    public void setStatus(String status) {
        this.status = status;
    }
    public Date getDateTime() {
        return dateTime;
    }
    public void setDateTime(Date dateTime) {
        this.dateTime = dateTime;
    }
}

```

### Listing 15: Action.java

```

/**
 * DatabaseAccess.java - the class provides database access for
 *
 *                               different DBMS. It is implemented in singleton
 *                               pattern for assuring that no concurring
 *                               connections are opened at any time.
 *
 * @author Jan Peters
 * @date 2009-12-05
 * @version 1.0
 */

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;

public class DatabaseAccess {

    private static DatabaseAccess INSTANCE;
    private Connection conn;

    // connection parameters for oracle and access
    private String dbHost;
    private int dbPort;
    private String dbSid;
    private String dbUser;
    private String dbPassword;
    private String dbFileName;

    // possible status texts
    static String UPDATE_SUCCESS = "Update: success.";
    static String UPDATE_FAILED = "Update: ID not found. Value inserted.";
    static String DELETE_SUCCESS = "Delete: success.";
    static String DELETE_FAILED = "Delete: ID not found.";
    static String INSERT_FAILED = "Account exists. Updated instead.";
    static String INSERT_SUCCESS = "Insert: success.";
    static String INVALID_ACTION = "Invalid operation. No action taken.";

    // create an instance for oracle access
    private DatabaseAccess(int dbms, String dbHost, int dbPort, String dbSid,
        String dbUser, String dbPassword)

```



```

    {
        System.out.println("DatabaseAccess()");
        this.dbHost = dbHost;
        this.dbPort = dbPort;
        this.dbSid = dbSid;
        this.dbUser = dbUser;
        this.dbPassword = dbPassword;
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            this.conn = DriverManager.getConnection("jdbc:oracle:thin:@" +
                this.dbHost + ":" + this.dbPort + ":" +
this.dbSid,
                this.dbUser, this.dbPassword);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    // create an instance for ms access
    private DatabaseAccess(String dbFileName, String dbUser, String dbPassword) {

        System.out.println("DatabaseAccess()");
        this.dbFileName = dbFileName;
        this.dbUser = dbUser;
        this.dbPassword = dbPassword;
        try {
            this.conn = DriverManager.getConnection(
                "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=" +
                this.dbFileName, this.dbUser, this.dbPassword);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    // return a connection to oracle
    // singleton pattern!
    public static DatabaseAccess getOracleInstance(String dbHost, int dbPort,

String dbSid, String dbUser,

String dbPassword)
    {
        System.out.println("getOracleInstance()");
        if(INSTANCE == null) {
            INSTANCE = new DatabaseAccess(0, dbHost, dbPort, dbSid, dbUser,
                dbPassword);
        }
        return INSTANCE;
    }

    // return a connection to ms access
    // singleton pattern!
    public static DatabaseAccess getAccessInstance(String dbFileName, String dbUser,

String dbPassword) {
        System.out.println("getAccessInstance()");
        if(INSTANCE == null) {
            INSTANCE = new DatabaseAccess(dbFileName, dbUser, dbPassword);
        }
        return INSTANCE;
    }

    // close connection (will be called after processing all data)
    public boolean closeConnection() {

```

```

        System.out.println("closeConnection()");
        try {
            if(!this.conn.isClosed()) {
                this.conn.close();
            }
            return true;
        } catch(Exception e) {
            return false;
        }
    }

    // return a list of all unprocessed actions
    // - an unprocessed action has no status text
    public ArrayList<Action> getUnprocessedActions() {
        System.out.println("getUnprocessedActions()");
        // create an empty list which can contain action objects
        ArrayList<Action> arrList = new ArrayList<Action>();

        try {
            // create variable for temporary action objects
            Action tmpAction;
            // create a statement for reading in the database
            Statement stmt =
this.conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,

            ResultSet.CONCUR_UPDATABLE);
            // fetch all unprocessed actions (unprocessed actions do not have a
            status text)
            ResultSet rs = stmt.executeQuery("SELECT account_id, oper_type,
            new_value, status,
            time_tag FROM action WHERE status IS NULL OR status = '' ORDER BY TIME_TAG");

            // for each row
            while(rs.next()) {
                // create a temporary action object
                tmpAction = new Action();
                // put in the data from the sql query
                tmpAction.setAccountId(rs.getInt("account_id"));
                tmpAction.setOperationType(rs.getString("oper_type"));
                tmpAction.setNewValue(rs.getDouble("new_value"));
                tmpAction.setStatus(rs.getString("status"));
                tmpAction.setDateTime(rs.getDate("time_tag"));
                // add the temporary action to the list
                arrList.add(tmpAction);
            }
            // close database connection
            rs.close();
        } catch(Exception e) {
            e.printStackTrace();
        }
        // return list
        return arrList;
    }

    // main function to process a certain action
    // - the parameter "predefinedReturnValue" can contain a previously
    // allocated text in case of a failed command
    public boolean processTransaction(Action action, String predefinedReturnValue) {
        System.out.println("processTransaction(" + action.getAccountId() + ")");
        // Default Result is set to Invalid Action
        String returnResult = INVALID_ACTION;
        try {
            // counts the affected records
            int updateCount;

```

```

// sql statement object for assuring high reliability
// and safeguarding against failure
PreparedStatement pstmt = null;

// if operation is an update
if(action.getOperationType().equals("u")) {
    // prepare statement for updating
    pstmt = this.conn.prepareStatement("UPDATE accounts SET BAL = ? WHERE
ACCOUNT_ID =
    ?");

    // add the new balance as first parameter
    pstmt.setDouble(1, action.getNewValue());
    // add the account id as second parameter
    pstmt.setInt(2, action.getAccountId());
    // if there is no predefined value, then the
    // default returned result will be UPDATE_SUCCESS,
    // otherwise it will be the predefined value
    if(predefinedReturnValue == null) {
        returnResult = UPDATE_SUCCESS;
    } else {
        returnResult = predefinedReturnValue;
    }
    // execute the created statement
    pstmt.executeUpdate();
    // count the number of results for success verification
    updateCount = pstmt.getUpdateCount();
    // if update failed because there exist no records,
    // an attempt for inserting data will be undertaken
    if(updateCount == 0) {
        // setup a new action object by copying the old one
        Action newAction = action;
        // change operation to insert instead of update
        newAction.setOperationType("i");
        // process a new transaction with the additional
information

        // of the previous result (UPDATE_FAILED)
        this.processTransaction(newAction, UPDATE_FAILED);
        // return false if both attempts did not succeed
        return false;
    }

// if operation will be an insert
} else if(action.getOperationType().equals("i")) {
    // prepare statement for inserting
    pstmt = this.conn.prepareStatement("INSERT INTO accounts
(account_id, bal) VALUES
    (?, ?)");

    // add the account id as first parameter

    pstmt.setInt(1, action.getAccountId());
    // add the new balance as second parameter
    pstmt.setDouble(2, action.getNewValue());
    // if there is no predefined value, then the
    // default returned result will be INSERT_SUCCESS,
    // otherwise it will be the predefined value
    if(predefinedReturnValue == null) {
        returnResult = INSERT_SUCCESS;
    } else {
        returnResult = predefinedReturnValue;
    }
}
try {
    // execute the created statement
    pstmt.executeUpdate();

```

```

        } catch(SQLException e) {
            // if execution fails and sql state is 23000 which
            // means that a row cannot be inserted because it
            // is already existing
            if(e.getSQLState() == "23000" ||
e.getMessage().equals("General error")) {
                // setup a new action object by copying the old
                one
                Action updateAction = action;
                // change operation to update instead of insert
                updateAction.setOperationType("u");
                // process a new transaction with the additional
                information
                // of the previous result (INSERT_FAILED)
                this.processTransaction(updateAction,
INSERT_FAILED);
                return false;
            } else {
                e.printStackTrace();
            }
        }
        // if operation will be a deleting of an account
    } else if(action.getOperationType().equals("d")) {
        // prepare statement for deleting
        ACCOUNT_ID = ?");
        pstmt = this.conn.prepareStatement("DELETE FROM accounts WHERE
        // add the account id as parameter
        pstmt.setInt(1, action.getAccountId());
        // define the expected result
        returnResult = DELETE_SUCCESS;
        // execute the statement
        pstmt.executeUpdate();
        // get the result
        updateCount = pstmt.getUpdateCount();
        // if no row has been deleted
        if(updateCount == 0) {
            // set status as failed, because no row was found or
            deleted
            returnResult = DELETE_FAILED;
        }
    }
    // write the status to the action row
    pstmt = this.conn.prepareStatement("UPDATE action SET status = ? WHERE
account_id =
    ?");
    // set status text as first parameter
    pstmt.setString(1,returnResult);
    // set account_id as identifier
    pstmt.setInt(2,action.getAccountId());
    try {
        // execute statement
        pstmt.executeUpdate();
    } catch(Exception e) {
        e.printStackTrace();
    }
} catch (SQLException e) {
    e.printStackTrace();
}
return true;
}
}

```

Listing 16: DatabaseAccess.java

```
/**
 * StartProgram.java - the start class for the accounts update program
 * @author Jan Peters
 * @date 2009-12-05
 * @version 1.0
 */

import java.util.ArrayList;

public class StartProgram {

    public static void main(String[] args) {
        new StartProgram();
    }

    public StartProgram() {
        System.out.println("StartProgram()");

        // for use in university (oracle)
        DatabaseAccess db = DatabaseAccess.getOracleInstance(
            "preston3",1521,"academic","petej97","S8247897");

        // for use at home (no oracle server, use of ms access)
        //DatabaseAccess db = DatabaseAccess.getAccessInstance("database.mdb","","");

        // load all unprocessed actions
        ArrayList<Action> actions = db.getUnprocessedActions();
        // for each action
        for(int i = 0; i < actions.size(); i++ ) {
            // process the given transaction
            db.processTransaction(actions.get(i), null);
        }
        // close database connection
        db.closeConnection();
    }
}
```

**Listing 17: StartProgram.java**

#### 2.1.4. Test

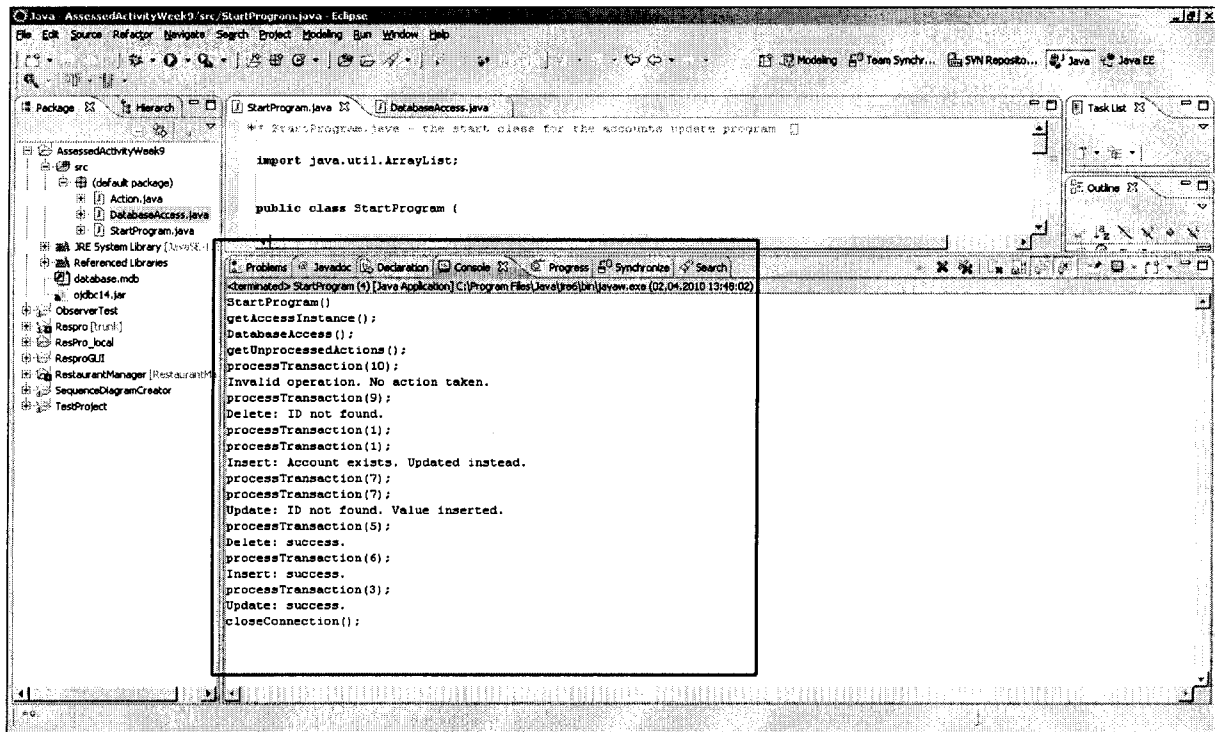
Testing has been performed by creating a set of test data which was applied to the program. The application of an “action” can cause either a change in the accounts table or a failure. For verify the appropriate running of the program, a table of expected results has been created. After running the program, the received results have been compared with the expected ones:

Action	Expected Result
Update ID 3, Set Bal. to 599	Update successful
Insert ID 6, Set Bal. to 20099	Insert successful
Delete ID 5	Delete successful

Update ID 7, Set Bal. to 1599	Update failed, Record does not exist, Record was inserted instead
Insert ID 1, Set Bal. to 399	Insertion failed, Record already exists, Record was updated
Delete ID 9	Deletion failed, Record does not exist
Invalid Command on ID 10	Invalid Command

**Figure 6: Table of Test Data and Expected Results**

The following figure shows an example output of the programming. This example is started in Eclipse IDE and connects to a Microsoft Access database:



**Figure 7: Example Output of the Accounts Update Program in Eclipse**

## 2.2. Evaluation of Technologies, Tools and Methods

### 2.2.1. Java or PL/SQL

Java has been chosen as the programming language for the accounts update program. Beside the familiarity to Java, the crucial point is that PL/SQL is a proprietary scripting language that originally was designed only for Oracle database management systems and therefore is not platform or database independent. A PL/SQL script would need to be

translated into another procedural language like Transactional SQL (T-SQL) for Microsoft SQL Server. This potentially causes an increased effort in time and therefore cost. Furthermore, a Java program runs outside the DBMS and therefore can make use of more or less unlimited processor resources which can increase performance (Lott 2007).

### **2.2.2. JDBC and Prepared Statements**

Usage of JDBC and Prepared Statements highly contributed to realise easy swapping of DBMS. Nowadays, the majority of relational database management systems bring a JDBC driver along that can easily be used in Java. Beside a specific driver, the only thing that needs to be changed is the connection string which contains information like the host address, database name, user account and password.

Prepared Statements are regarded as an improved alternative to common SQL statements especially when dealing with parameters. A Prepared Statement is precompiled when it is created and does not need further compilation when running. The advantage is that one Prepared Statement can be used with different parameters and needs to be compiled only once (Oracle Corporation 2010, "Using Prepared Statements"). Another advantage is that values are not directly written into the SQL string but step-by-step added to the Prepared Statement object by using functions that correspond to the parameters data type. This assures that only valid values are reaching the database, because the Prepared Statement takes care of escaping values (De Vries 2006, p.17). This seems to be an effective way to prevent SQL injections and therefore increases application and database security.

### **2.2.3. Eclipse IDE**

Using the Eclipse Integrated Development Environment was quite useful, primarily because of its good usability. It contains comfortable features like syntax highlighting, code completion and code refactoring. Eclipse is regarded as one of the best open source solutions for Java development and is used prestigious IT companies like SAP, Ericsson and Hewlett Packard as well as strategic developers like Intel or IBM, which create products that are based on the Eclipse project (Milinkovich 2004).

### **2.2.4. Oracle Application Express (APEX)**

Oracle tries to position APEX as an easy-to-use web application development tool. It can be useful for rapid development of common requirements like data entry masks, reports and charting (Oracle Corporation 2010, "Oracle Application Express"). These typical fields of appliance are covered by wizards, which causes a minimal amount of programming effort. APEX is available as a free add-on for "every edition of Oracle database" (Oracle Corporation

2010, "Application Express Flash Demonstration"). In converse argument that means, that an Oracle database license has to be owned before using APEX.

However, APEX has not achieved a noteworthy acceptance by software engineers. Research about APEX discovered only a handful of books and, beside the Oracle APEX web site, no frequently used exchange platforms on the web. One reason might be its limited customisation capabilities because of predefined elements. It could be difficult to implement more complex business processes or work flows with this kind of modular construction system.

### **2.3. References**

DE VRIES, S., 2006. *A Modular Approach to Data Validation* [online]. Available: <http://research.corsaire.com/whitepapers/060116-a-modular-approach-to-data-validation.pdf> [accessed 3rd April 2010]

LARMAN, C., 2005. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. 3<sup>rd</sup> ed. Upper Saddle River: Prentice Hall

LOTT, S., 2007. *PL/SQL vs. Java – Which is really faster?* [online]. Available: [http://homepage.mac.com/s\\_lott/iblog/architecture/C465799452/E20070322201220/index.html](http://homepage.mac.com/s_lott/iblog/architecture/C465799452/E20070322201220/index.html) [accessed 2nd April 2010]

MILINKOVICH, M., 2004. *Eclipse bietet Potential für Softwareunternehmen und IT-Abteilungen* [online]. Available: <http://de.sap.info/%E2%80%9Eclipse-bietet-potenzial-fur-softwareunternehmen-und-it-abteilungen%E2%80%9C/1997> [accessed 3rd April 2010]

ORACLE CORPORATION, 2010. *Using Prepared Statements* [online]. Available: <http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html> [accessed 2nd April 2010]

ORACLE CORPORATION, 2010. *What is APEX?* [online]. Available: [http://www.oracle.com/technology/products/database/application\\_express/html/what\\_is\\_apex.html](http://www.oracle.com/technology/products/database/application_express/html/what_is_apex.html) [accessed 2nd April 2010]

ORACLE CORPORATION, 2010. *Application Express Flash Demonstration* [online]. Available: [http://www.oracle.com/pls/ebn/swf\\_viewer.load?p\\_shows\\_id=6392594](http://www.oracle.com/pls/ebn/swf_viewer.load?p_shows_id=6392594) [accessed 2<sup>nd</sup> April 2010]



### 3. Improving Data Access by Data Distribution and Replication

#### 3.1. Task 2) Creation and Testing of a SQL View

Designing a view which includes tables of different databases requires careful thoughts about the required fields and the construction of the SQL statement. Basically, there are three steps necessary in order to create the basic view:

At first, a SQL statement needs to be created for to retrieve data from one database. The required tables and fields need to be determined and the tables need to be joined, e.g. like this:

```
SELECT EM.employee_id, EM.first_name, EM.last_name, lo.city, JO2.job_title, jo.job_title AS
former_job_title, jh.start_date, jh.end_date
FROM hrd.employees@student3 EM
LEFT OUTER JOIN hrd.departments@student3 DE ON EM.department_id = DE.department_id
LEFT OUTER JOIN hrd.locations@student4 LO ON DE.location_id = LO.location_id
LEFT OUTER JOIN hrd.job_history@student4 JH ON EM.employee_id = jh.employee_id
LEFT OUTER JOIN hrd.jobs@student4 JO ON JO.job_id = JH.job_id
LEFT OUTER JOIN hrd.jobs@student4 JO2 ON JO2.job_id = EM.job_id
```

#### Listing 18: Query for to retrieve the List of all German Employees

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	CITY	JOB_TITLE	FORMER_JOB_TITLE	START_DATE	END_DATE
204	Hermann	Baer	Munich	Public Relations Representative			

Figure 8: Result for Listing 18

Beside basic employee data which is on the corresponding database server, the information about locations and former jobs is available on the “student4” server. This query contains the table “departments” as an intermediate table, because it contains the reference to the actual location of the employee which is included in the “locations” table. The table “jobs” is joined in two ways. On the one hand, it is linked to the current job of the employee and on the other hand it references to all former jobs that are contained in the “job\_history” table.

Although, this query obviously contains all required fields from the given tables, an important column is missing: an identifier for the database that holds the queried records. Therefore, an additional field that contains the “database country” needs to be added manually. This can be performed as follows:

```
SELECT EM.employee_id, EM.first_name, EM.last_name, lo.city, JO2.job_title, jo.job_title AS
former_job_title, jh.start_date, jh.end_date, 'Germany' AS DbCountry
...
```

#### Listing 19: Extended Query which includes a column for the corresponding Country

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	CITY	JOB_TITLE	FORMER_JOB_TITLE	START_DATE	END_DATE	DBCOUNTRY
204	Hermann	Baer	Munich	Public Relations Representative				Germany

**Figure 9: Result of Listing 18 with the Changes made in Listing 19**

This query has to be adjusted for the “student4” and “student5” database as well, but with different presets in the “DbCountry” column (values for “USA” and “UK”).

The next step is to combine all three queries in order to retrieve one set of records. Oracle uses the UNION ALL command for this operation. Finally, the constructed SQL statement needs to be stored as a view using the CREATE OR REPLACE VIEW command:

```
CREATE OR REPLACE VIEW employee_list AS (
  SELECT EM.employee_id, EM.first_name, EM.last_name, lo.city, JO2.job_title, jo.job_title AS
    former_job_title, jh.start_date, jh.end_date, 'Germany' AS DbCountry
  FROM hrd.employees@student3 EM
  LEFT OUTER JOIN hrd.departments@student3 DE ON EM.department_id = DE.department_id
  LEFT OUTER JOIN hrd.locations@student4 LO ON DE.location_id = LO.location_id
  LEFT OUTER JOIN hrd.job_history@student4 JH ON EM.employee_id = jh.employee_id
  LEFT OUTER JOIN hrd.jobs@student4 JO ON JO.job_id = JH.job_id
  LEFT OUTER JOIN hrd.jobs@student4 JO2 ON JO2.job_id = EM.job_id
UNION ALL
  SELECT EM.employee_id, EM.first_name, EM.last_name, lo.city, JO2.job_title, jo.job_title AS
    former_job_title, jh.start_date, jh.end_date, 'USA' AS DbCountry
  FROM hrd.employees@student4 EM
  LEFT OUTER JOIN hrd.departments@student4 DE ON EM.department_id = DE.department_id
  LEFT OUTER JOIN hrd.locations@student4 LO ON DE.location_id = LO.location_id
  LEFT OUTER JOIN hrd.job_history@student4 JH ON EM.employee_id = jh.employee_id
  LEFT OUTER JOIN hrd.jobs@student4 JO ON JO.job_id = JH.job_id
  LEFT OUTER JOIN hrd.jobs@student4 JO2 ON JO2.job_id = EM.job_id
UNION ALL
  SELECT EM.employee_id, EM.first_name, EM.last_name, lo.city, JO2.job_title, jo.job_title AS
    former_job_title, jh.start_date, jh.end_date, 'UK' AS DbCountry
  FROM hrd.employees@student5 EM
  LEFT OUTER JOIN hrd.departments@student5 DE ON EM.department_id = DE.department_id
  LEFT OUTER JOIN hrd.locations@student4 LO ON DE.location_id = LO.location_id
  LEFT OUTER JOIN hrd.job_history@student4 JH ON EM.employee_id = jh.employee_id
  LEFT OUTER JOIN hrd.jobs@student4 JO ON JO.job_id = JH.job_id
  LEFT OUTER JOIN hrd.jobs@student4 JO2 ON JO2.job_id = EM.job_id
)
```

**Listing 20: SQL Statement for to Create a View over Different Databases**

A simple query for retrieving the first few rows is shown below:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	CITY	JOB_TITLE	FORMER_JOB_TITLE	START_DATE	END_DATE	DBCOUNTRY
204	Hermann	Baer	Munich	Public Relations Representative				Germany
205	Shelley	Higgins	Seattle	Accounting Manager				USA
206	William	Gietz	Seattle	Public Accountant				USA
108	Nancy	Greenberg	Seattle	Finance Manager				USA
111	Ismael	Sciarra	Seattle	Accountant				USA
112	Jose Manuel	Urman	Seattle	Accountant				USA
110	John	Chen	Seattle	Accountant				USA
113	Luis	Popp	Seattle	Accountant				USA
109	Daniel	Faviet	Seattle	Accountant				USA
100	Steven	King	Seattle	President				USA
102	Lex	De Haan	Seattle	Administration Vice President	Programmer	13-JAN-93	24-JUL-98	USA
101	Neena	Kochhar	Seattle	Administration Vice President	Public Accountant	21-SEP-89	27-OCT-93	USA
101	Neena	Kochhar	Seattle	Administration Vice President	Accounting Manager	28-OCT-93	15-MAR-97	USA
114	Den	Raphaely	Seattle	Purchasing Manager	Stock Clerk	24-MAR-98	31-DEC-99	USA
118	Guy	Himuro	Seattle	Purchasing Clerk				USA
119	Karen	Colmenares	Seattle	Purchasing Clerk				USA
117	Sigal	Tobias	Seattle	Purchasing Clerk				USA
116	Shelli	Baida	Seattle	Purchasing Clerk				USA
115	Alexander	Khoo	Seattle	Purchasing Clerk				USA
200	Jennifer	Whalen	Seattle	Administration Assistant	Public Accountant	01-JUL-94	31-DEC-98	USA
200	Jennifer	Whalen	Seattle	Administration Assistant	Administration Assistant	17-SEP-87	17-JUN-93	USA
146	Karen	Partners	Oxford	Sales Manager				UK
149	Eleni	Zlotkey	Oxford	Sales Manager				UK
145	John	Russell	Oxford	Sales Manager				UK

Figure 10: Excerpt of the Full Result for the Query in Listing 20

### 3.2. Task 3) Production of a Distributed Database Design Solution

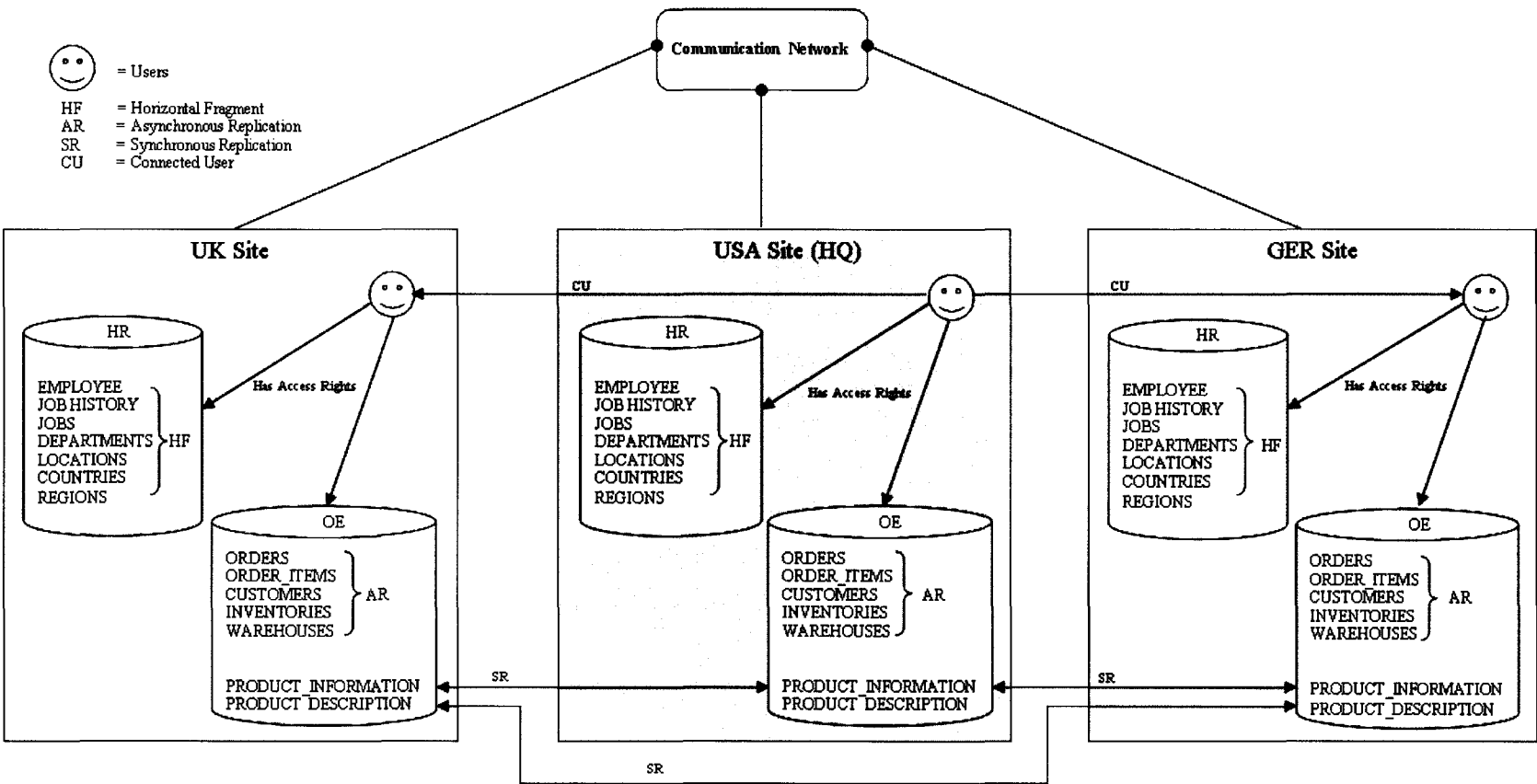


Figure 11: Distributed Database Model

This diagram describes the basic architecture of a solution for to distribute the company's database. Basically, the scenario requires a "truly distributed database architecture" as it is shown by Elmasri (2000, p.768). The chosen approach combines Elmasri's model, which contains information about the different transparencies (2000, p.769), with symbols and presents detailed descriptions of the applied replications, fragmentations and accessibilities.

On the top of the model, there is the communications network which stands for the basic connectivity between the databases in order to enable distributed query processing (Elmasri 2000, p.770). The three rectangles do represent the different sites with their database servers which are connected to the communication network.

Each database server contains two databases with identical data structures. Location specific data (especially the HR database) are stored horizontal fragmented because these tables all have the same structure but different contents. In other words, the rows of the table are distributed to the corresponding sites (Monger 2009, p.6).

While the distribution of the HR database is regarded as simple, the requirements for the OE database need to be carefully analysed. Updates of orders do not need to be available on other sites instantaneously; therefore this partial replication can be asynchronous, e.g. once a day. According to the requirements, the product and price information needs to be as consistent and up-to-date as possible. This information is stored in the tables "product\_information" and "product\_description" which are synchronously replicated to the other databases. In this case, synchronously means an immediate replication. It has to be said that this replication technique requires a concurrency control for "multiple copies" in order to maintain a consistent database (Elmasri 2000, p.786).

Finally, the model contains connected users in order to allow one site to look up data of another location. These users have access rights to all tables of the corresponding database.

### ***3.3. Evaluation of Technologies, Tools and Methods***

#### **3.3.1. Concept of Distributed Databases and Replication**

Basically, the idea of distributed databases is regarded as a great extension for "normal" database management systems. Bigger companies can profit of these concepts, especially when using a distributed computing system for different locations instead of stand-alone applications (Elmasri 2000, p.766). On the one hand, it enables the different sites to store their data autonomic and therefore increase sovereignty of their data pool by using a certain limited transparency level. On the other hand it allows easy exchange of productive data with other sites and can enhance communication and transparency with all its advantages and synergetic effects. Elmasri mentions some of the most important pros (2000, p.770):

- Improved performance by spreading the CPU workload to different systems
- Better availability, because if one database should fail, the other sites could continue work or another database might be accessed
- Easier expansion can be performed easier, because the architecture of the database management system is designed for increasing the database size or adding more processors

Management can highly profit from immediate reports. Processing distributed queries with various databases can for example show the differences between certain sites and serve as a basis for decision support.

### 3.3.2. Database Design and Effort

All of these advantages come along with an enormous effort and serious risks. Good planning of a distributed system is essential for a successful system and is seen as the major risk. Especially the requirements elicitation should be detailed, well-designed and validated multiple times. Later changes in basic needs could easily cause a complete re-design of the architecture and possibly have a bigger impact than common requirement changes in software systems development.

The effort can be divided into two parts:

1. The effort for to design and implement a distributed database. It takes more time to establish a distributed database than a normal one, because of the complexity and constraints that are derived from the requirements. Furthermore, an extensive testing must include availability and reliability tests as well as auditing concurrency control.
2. The effort for to run a distributed database. Beside the financial aspects of owning and maintaining multiple servers with numerous licenses, queries may cause huge efforts in system performance. Elmasri provides basic calculation for different strategies in order to determine the effort of distributed query processing (2000, pp.781-782)

The effort and costs for distributed database systems depends on the chosen approach:

- Centralised databases are the cheapest concept, but they go along with high communication effort and provide low reliability. Furthermore, they are not seen as a distributed database.
- Fragmentation of a database distributes the responsibility for the content to different servers. The storage cost is quite low and the system itself might be highly available. Nevertheless, retrieving single records might fail because they are only stored at one place.

- Replication provides the best quality in terms of communication, reliability and availability. However, it distributes the data to all involved servers and therefore causes a continuous huge effort.

### 3.3.3. Applicability in Praxis

Implementation of a distributed database design for Oracle database servers requires thoughts about the different techniques. Databases and users need to be linked and replication models need to be determined. The replication itself requires careful thoughts about concurrency. Furthermore, the necessities for defining so called snapshots have to be evaluated and other techniques for increasing the speed of queries should be applied.

### 3.4. References

ELMASRI, R., 2000. *Fundamentals of Database Systems*. 3<sup>rd</sup> ed. Harlow: Addison-Wesley

MONGER, A., 2009. *Distributed Databases* [online]. Available:

<http://mycourse.solent.ac.uk/file.php/1795/DatabaseSystems/Topics/Distributed%20Databases/Distributed%20Databases%20Test.ppt#1> [accessed 21<sup>st</sup> April 2010]



## 4. Multi-Dimensional Modelling and Analysis for Decision Support

### 4.1. Task 1) Implementation of Analysing Data for Decision Support

This analysis deals with providing a presentation for the management. This report contains sales information generalised by product categories and split into customer's income level and the year. On the one hand, it can be used for to identify uneconomical product categories that might be rejected. On the other hand it can detect lucrative product categories that could be expanded in a better way. The information about the customer's income level, and therefore their credit rating, could lead to decisions if to put the business's focus on richer customer groups.

Basically, the given model contains all relevant dimensions for this reporting. According to Smith (2004, pp.384-385) and Mallach (2000, p.496), this model can be characterised as a star scheme that is commonly used in data warehouses. The first step is to analyse the dimensions and measures (or fact data) in order to determine the so called "cube". A cube typically consists of three dimensions and the measure. Larger cubes with four or more dimensions are called "hypercubes" (Mallach 2000, p.499). In this example the basic dimensions are given, but the values need to be determined:

	<b>Dimension</b>	<b>Values</b>
<b>1<sup>st</sup> Dimension</b>	Time	Year
<b>2<sup>nd</sup> Dimension</b>	Product	Category
<b>3<sup>rd</sup> Dimension</b>	Customer	Income Level

**Table 1: The Chosen Dimensions**

The fact data is defined as well; it is either the number of sold products or the summarised amount that has been earned by selling the products. After changing the dimensions, the modified cube looks like this:



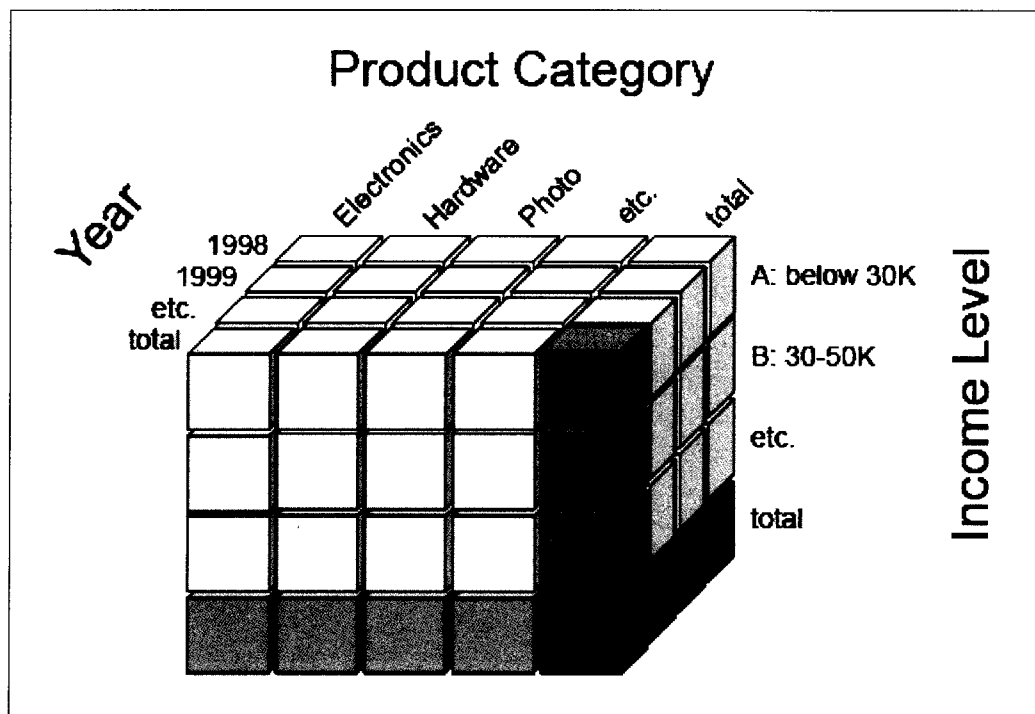


Figure 12: Multi-Dimensional Model (Cube) for the example Analysis

After modelling the cube it needs to be implemented as a SQL statement. The following SQL statement contains the basic query that returns all combinations of records:

```
SELECT      T.calendar_year, P.prod_category, C.cust_income_level,
            sum(S.amount_sold) AS FactData
FROM        sales_transactions_ext S, products P, times T, customers C
WHERE       S.prod_id = P.prod_id
AND         S.time_id = T.time_id
AND         S.cust_id = C.cust_id
GROUP BY   T.calendar_year, P.prod_category, C.cust_income_level
ORDER BY   T.calendar_year, P.prod_category, C.cust_income_level
```

Listing 21: Querying the cube for to retrieve all records (combined with all dimensions)

Retrieving data from this cube should be done by assuming values for the dimensions in order to provide “a more useable, generic solution” (Monger 2010, p.5). The whole set of records might be very large and could therefore demand a lot of unnecessary processing power. A short example statement retrieves the 1998’s summarised amount for the category “Electronics” listed by the different income levels:

```
SELECT      T.calendar_year, P.prod_category, C.cust_income_level,
            sum(S.amount_sold) AS FactData
FROM        sales_transactions_ext S, products P, times T, customers C
WHERE       S.prod_id = P.prod_id
AND         S.time_id = T.time_id
AND         S.cust_id = C.cust_id
```

```

AND          T.calendar_year = '1998'
AND          P.prod_category = 'Electronics'
GROUP BY    T.calendar_year, P.prod_category, C.cust_income_level
ORDER BY    T.calendar_year, P.prod_category, C.cust_income_level

```

**Listing 22: Querying the cube for a certain year and product category**

The database result for this query looks like this:

CALENDAR_YEAR	PROD_CATEGORY	CUST_INCOME_LEVEL	FACTDATA
1998	Electronics	A: Below 30,000	80432.73
1998	Electronics	B: 30,000 - 49,999	98595.35
1998	Electronics	C: 50,000 - 69,999	171791.26
1998	Electronics	D: 70,000 - 89,999	173268.92
1998	Electronics	E: 90,000 - 109,999	269882.92
1998	Electronics	F: 110,000 - 129,999	308938.59
1998	Electronics	G: 130,000 - 149,999	171013.77
1998	Electronics	H: 150,000 - 169,999	170221.12
1998	Electronics	I: 170,000 - 189,999	130022.95
1998	Electronics	J: 190,000 - 249,999	86140.1
1998	Electronics	K: 250,000 - 299,999	55149.8
1998	Electronics	L: 300,000 and above	57665.01
1998	Electronics		868.47

**Table 2: Example result for query in Listing 22**

The last record contains an empty column for income level. It means that there are customer records in the database that contain no information about their income level. The circumstance of missing or empty data is common for data warehouses and the reason for it is often poor information quality in data warehouses database (Mallach 2000, p.507). In order to achieve a meaningful and valid result this exceptional case should be handled either by excluding the data or better naming it as “unknown” for example. According to Mallach, this process is called cleansing and aims to reduce missing, incorrect, inconsistent and conflicting data (2000, p.508).

The next step is to prepare the data for evaluation and presentation. This can be done using different approaches.

### 4.1.1. Preparing the Data using Microsoft Excel

Copying the raw data to Microsoft Excel and create a pivot table that is able to display multi-dimensional data on a spreadsheet with summarised columns and rows:

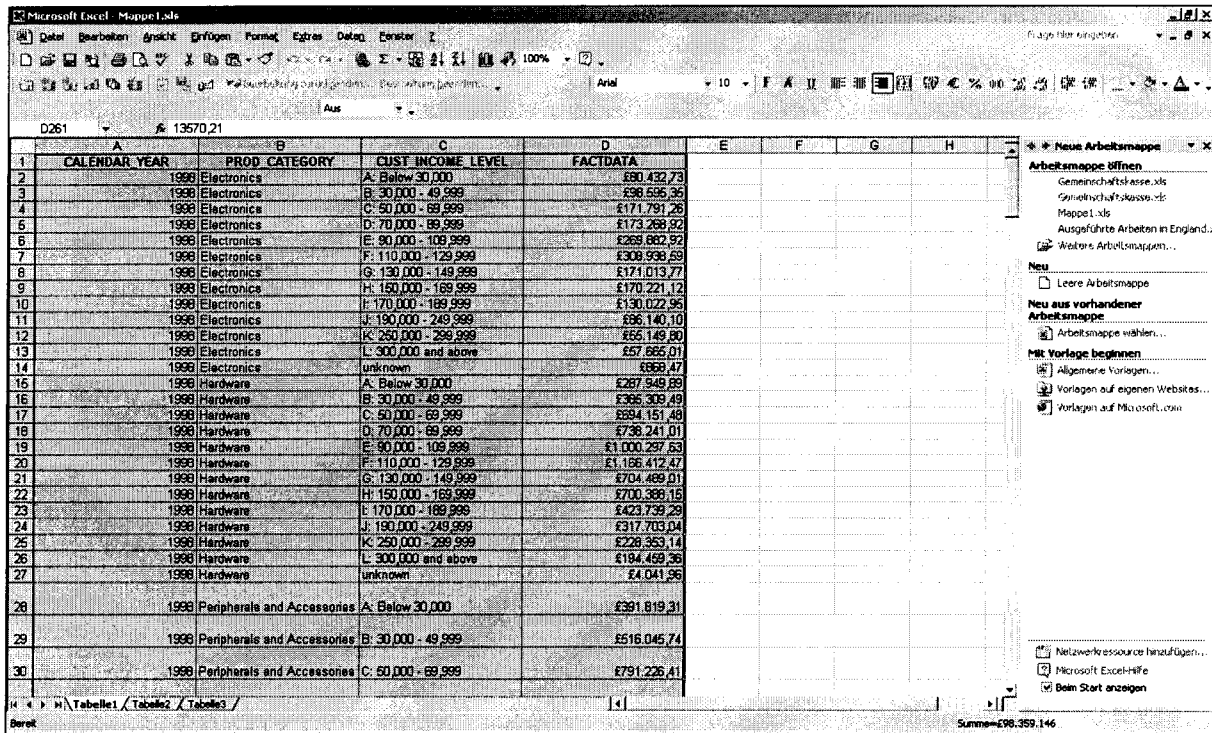


Figure 13: Raw Database records copied into an Excel document (with cleansing)

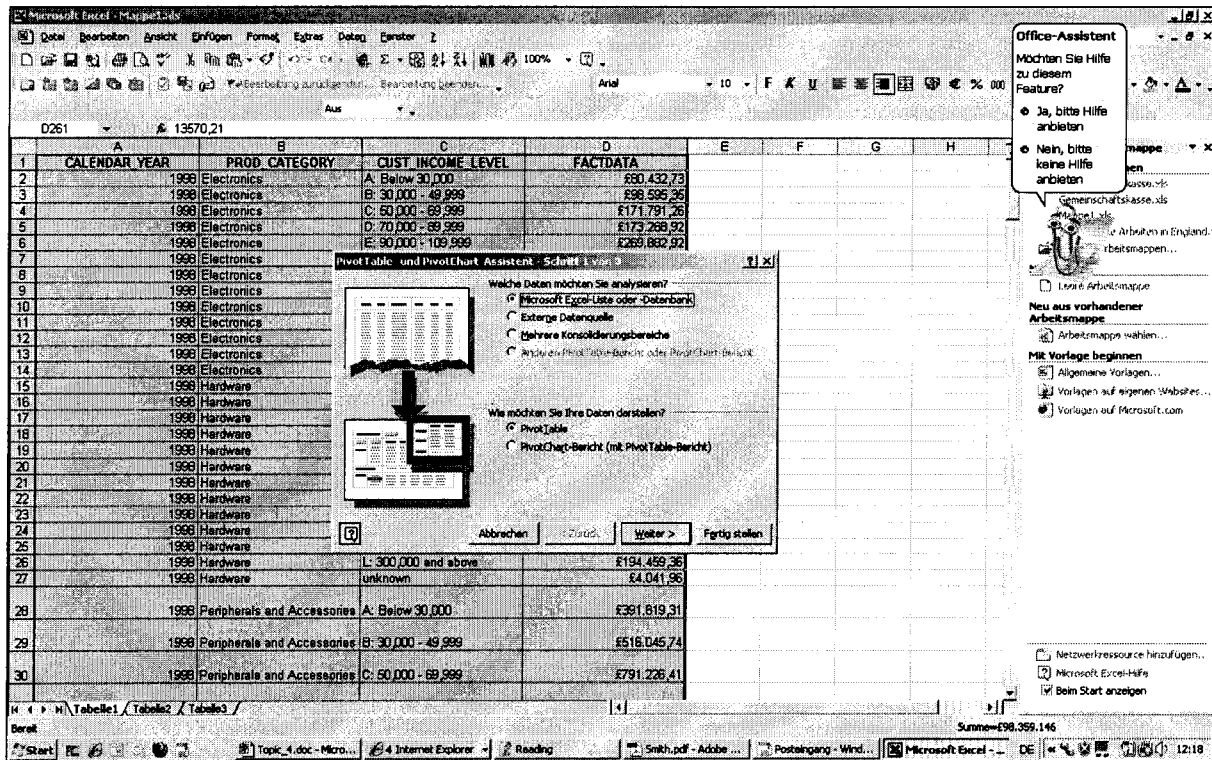


Figure 14: Creation of Pivot Table uses the Excel Wizard for Pivot Tables

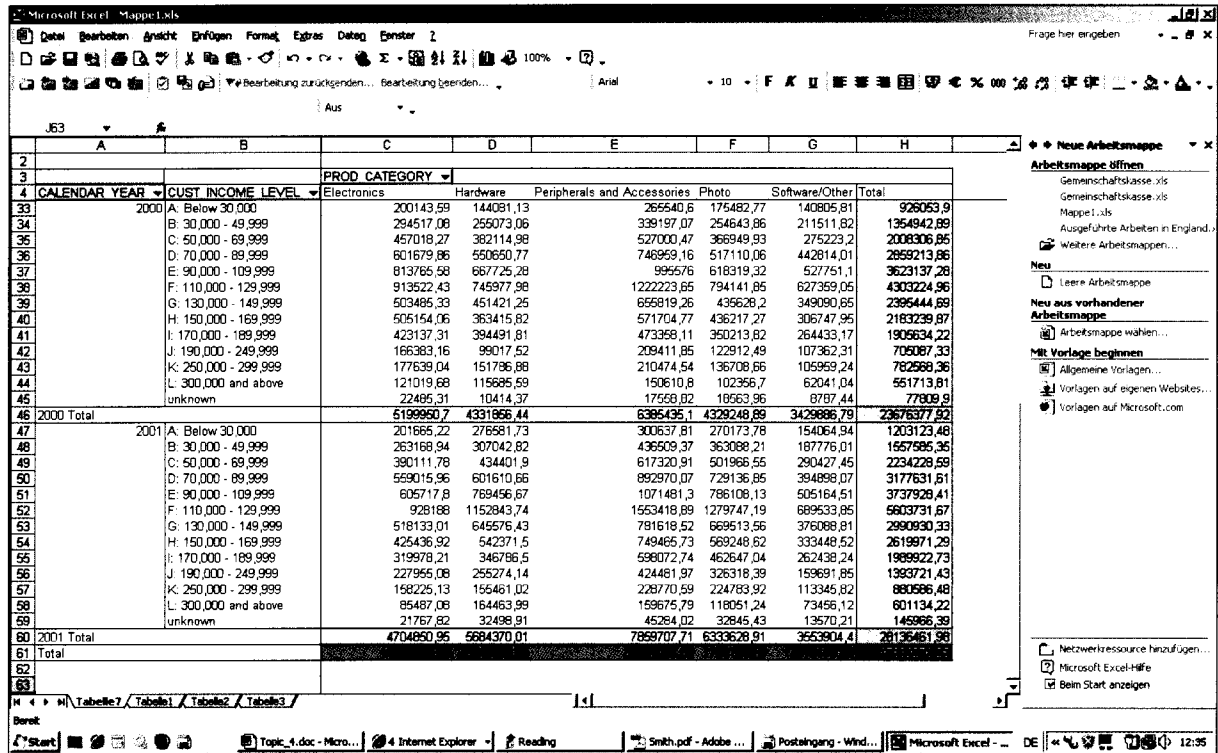


Figure 15: Final Pivot Table with Indication of Dimensions and Aggregated Measures according to Figure 12

### 4.1.2. Preparing the Data using Oracle SQL statements

Another faster approach is to use the built-in SQL command ROLLUP( ) for to rollup dimensions. The first step might be to rollup only one dimension. This can be expressed like this:

```

SELECT      T.calendar_year, C.cust_income_level, P.prod_category,
            sum(S.amount_sold) AS FactData
FROM        sales_transactions_ext S, products P, times T, customers C
WHERE       S.prod_id = P.prod_id
AND         S.time_id = T.time_id
AND         S.cust_id = C.cust_id
GROUP BY   T.calendar_year, C.cust_income_level, ROLLUP(P.prod_category)
ORDER BY   T.calendar_year, C.cust_income_level, P.prod_category
    
```

Listing 23: Using the ROLLUP Command in order to add Aggregations to the Result

An excerpt of the result looks like this:

CALENDAR_YEAR	CUST_INCOME_LEVEL	PROD_CATEGORY	FACTDATA
1998	A: Below 30,000	Electronics	80432.73
1998	A: Below 30,000	Hardware	287949.89
1998	A: Below 30,000	Peripherals and Accessories	391819.31
1998	A: Below 30,000	Photo	153120.93
1998	A: Below 30,000	Software/Other	110691.19
1998	A: Below 30,000	(All Categories)	1024014.05
1998	B: 30,000 - 49,999	Electronics	98595.35
1998	B: 30,000 - 49,999	Hardware	385309.49
1998	B: 30,000 - 49,999	Peripherals and Accessories	516045.74
1998	B: 30,000 - 49,999	Photo	216223.47
1998	B: 30,000 - 49,999	Software/Other	183310.74
1998	B: 30,000 - 49,999		1399484.79
1998	C: 50,000 - 69,999	Electronics	171791.26
1998	C: 50,000 - 69,999	Hardware	694151.48
CALENDAR_YEAR	CUST_INCOME_LEVEL	PROD_CATEGORY	FACTDATA
...	...	...	...
CALENDAR_YEAR	CUST_INCOME_LEVEL	PROD_CATEGORY	FACTDATA
1998	L: 300,000 and above	Software/Other	66083.67
1998	L: 300,000 and above		682793.56
1998	* unknown	Electronics	868.47
1998	* unknown	Hardware	4041.96
1998	* unknown	Peripherals and Accessories	42490.24
1998	* unknown	Photo	9600.99
1998	* unknown	Software/Other	10840.53
1998	* unknown		67842.19

Table 3: Raw Result from Oracle with rolled up Categories

These rollups can be expanded in any directions by adding more dimensions. Rolling up all dimensions is possible as well and can be performed by using the CUBE( ) command:

```
SELECT      T.calendar_year, C.cust_income_level, P.prod_category,
            sum(S.amount_sold) AS FactData
FROM        sales_transactions_ext S, products P, times T, customers C
```

```

WHERE      S.prod_id = P.prod_id
AND        S.time_id = T.time_id
AND        S.cust_id = C.cust_id
GROUP BY   CUBE(T.calendar_year, C.cust_income_level, P.prod_category)
ORDER BY   T.calendar_year, C.cust_income_level, P.prod_category

```

**Listing 24: SQL Statement that includes the CUBE( ) Command and retrieves all relevant Dimensions with aggregated Values**

The result for this query contains the same data like the Microsoft Excel pivot table but it displays them in slightly different way. Total number of records for this query is 420 rows; therefore a screenshot of last rows with aggregated values should be sufficient:

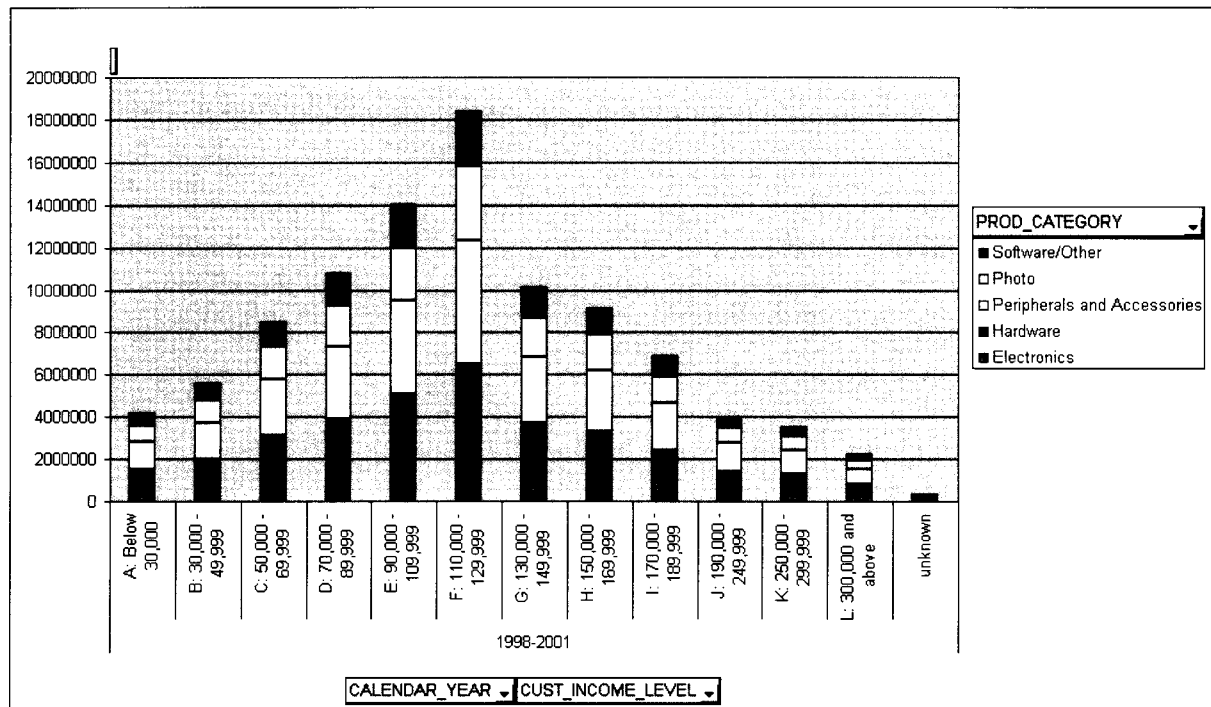
CALENDAR_YEAR	CUST_INCOME_LEVEL	PROD_CATEGORY	FACTDATA
L: 300,000 and above		Software/Other	281231.8
L: 300,000 and above			2212796.97
		Electronics	51699.42
		Electronics	14595162.6
		Hardware	72375.24
		Hardware	20643005.6
		Peripherals and Accessories	131992.22
		Peripherals and Accessories	30821066.1
		Photo	70346.04
		Photo	17961865.9
		Software/Other	42966.79
		Software/Other	13818176.1
			369369.71
			97839276.3

**Figure 16: Aggregated Result that includes all Dimensions**

Obviously, this approach requires additional manual work by adding the labels for the aggregated values.

#### 4.1.3. Evaluate Data and Derive Decision Support

The goal of this analysis is to support the decision which product category might be rejected and on which customer group should the company aim. The data of the Excel document can be put into diagrams to visualise the statistics.



**Figure 17: Chart helps to identify the most common Income Levels and the Proportion of the different Product Categories in Relationship to each other**

The outcome of this representative chart is as follows:

- Most turnover is generated by customers with “income level F”;
- Basically, customers with an income of less than 30.000 or more than 190.000 are outnumbered
- Best products, especially in the main customer group, are “Peripherals and Accessories” products
- “Software/Other” and “Electronics” are accounting only for a small part of total turnover

Therefore, some decisions could be derived by business analysts, for example according to processes of decision making (Mallach 2000, pp.38-39). Some simple but significant decisions might be:

- Focus on average income level customers with levels “C” to “I”. This also provides some security when dealing with debits or credit card payments because of increased credit rating!
- Try to get rid of either the categories “Software/Other” or “Electronics” and concentrate business on the more profitable products.

### 4.2. Task 2) Development of a Dimension Model for the HR Database

The HR department mainly deals with the employees and all related affairs. One of these affairs is the annual personnel talks with corresponding requests for salary adjustments. It is required to support the decision, if an employee will earn more money, by retrieving valuable information from the database. For example, this information can be the development of salary or the current average salary for the job.

First step for to design a model is to analyse the requirements in order to define the dimensions for this cube:

- Employee information is required to identify the employee (employee\_id)
- Furthermore, the salary of the employee needs to be recorded steadily (salary)
- Development needs to be illustrated, therefore a time dimension is necessary (time\_id, month, year)
- Similar jobs should be compared which each other, therefore the title of the job needs to be logged (job\_title)

According to these requirements, a table for the time dimension needs to be added to the database model. This example is designed for to record the information in monthly intervals, therefore the time dimension contains the month and the year and of course, an identifier for accessing the time. The next table that needs to be created is the table that actually contains the records. Beside the reference to the time, it contains the identifier for the employee, the job and the salary at the time of recording. Furthermore, a column that contains the term of employment is added. The following model has been designed:

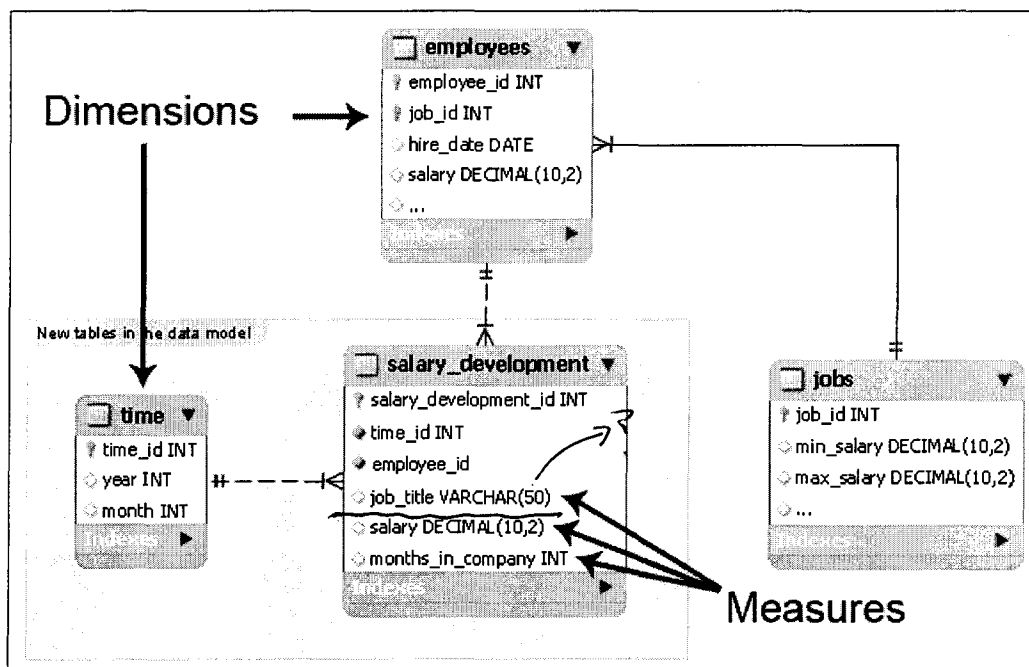


Figure 18: Multi-Dimensional Database Model for to provide recording of Salary Development



Implementation of it can be performed by creating the new tables and a transformation script.

The model realises the characteristics for to differ between fact data and dimension data that is mentioned by Mallach (2000, p.496):

Fact Data	Dimension Data
Millions or billions of rows	Tens to a few million rows
Multiple foreign keys	One primary key
Numeric	Textual Descriptions
Don't change	Frequently modified

**Table 4: Differences between Fact Data and Dimension Data**

In order to fill the new table with data, a transformation processing is required. It is recommended to perform the following steps each month:

- Add a new record in the table "time" that contains the current year and month
- For each employee the following information are collected and inserted into the "salary\_development" table together with the corresponding "time" identifier:
  - o Current job
  - o Current salary
  - o Current number of months of employment

An example for this processing can be simulated by creating inserting SQL statements:

```

/* Records for April 2010 */
INSERT INTO time (time_id, year, month)
VALUES      (1, 4, 2010);
INSERT INTO salary_development (salary_development_id, time_id,
employee_id, job_title, salary, months_in_company)
VALUES      (1, 1, 1, 'Stock Clerk', 2000, 120);
INSERT INTO salary_development (salary_development_id, time_id,
employee_id, job_title, salary, months_in_company)
VALUES      (2, 1, 2, 'Stock Clerk', 2200, 125);
INSERT INTO salary_development (salary_development_id, time_id,
employee_id, job_title, salary, months_in_company)
VALUES      (3, 1, 3, 'Stock Clerk', 2000, 70);
...

/* Records for May 2010 */
INSERT INTO time (time_id, year, month)
VALUES      (2, 5, 2010);
INSERT INTO salary_development (salary_development_id, time_id,
employee_id, job_title, salary, months_in_company)
VALUES      (4, 2, 1, 'Administration Assistant', 3100, 121);

```

```

INSERT INTO salary_development (salary_development_id, time_id,
    employee_id, job_title, salary, months_in_company)
VALUES (5, 2, 2, 'Stock Clerk', 2200, 126);
INSERT INTO salary_development (salary_development_id, time_id,
    employee_id, job_title, salary, months_in_company)
VALUES (6, 2, 3, 'Stock Clerk', 2200, 71);
...
    
```

Listing 25: Test Data for to Record Salary Development on a monthly base

Referring to the analysis techniques described in Task 1, a cube can be produced for accessing and evaluating the information:

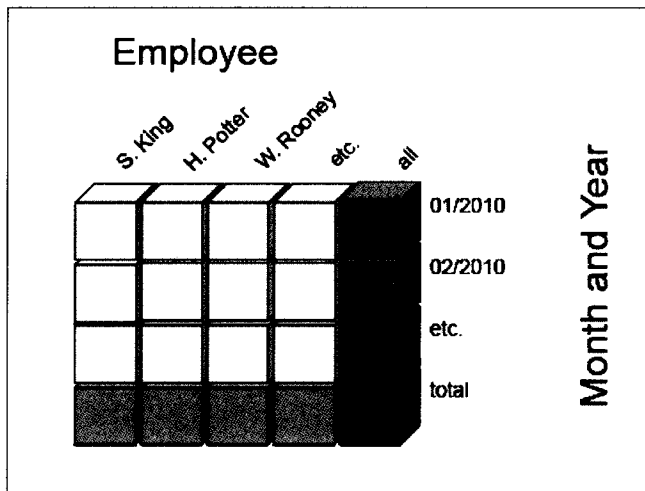


Figure 19: Two-Dimensional Model for Recording the Development of Salary

An example report for decision support might monitor the changes in an employee’s salary by using a curve chart with time dimension on the X-axis and the salary on the Y-axis:

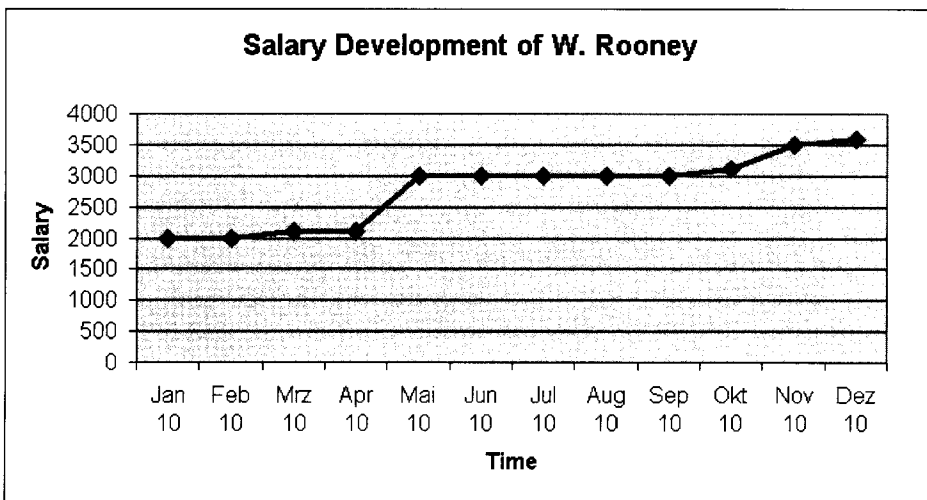


Figure 20: Example Diagram for Monitoring Development of Salary based on Collected Information

### **4.3. Evaluation of Technologies, Tools and Methods**

#### **4.3.1. Online Analytical Processing (OLAP)**

Firstly, it has to be said that the definition of OLAP is varying and the outcome of this topic is that OLAP is understood as a methodology for to explore databases. Beside Codd's definition of 18 rules for OLAP compliance (Smith 2004, p.403), there is another, simpler approach which is called FASMI. The abbreviation stands for the key requirements of OLAP applications (Smith 2004, p.406):

- Fast → immediate delivery of results, even if large cubes are involved
- Analysis → must be able to perform all required reports
- Shared → high application security, especially when dealing with concurrent updates. However, most OLAP applications use read-only access.
- Multi-Dimensional → key requirement is to analyse data in multiple hierarchies
- Information → refers to the data that an OLAP application can hold and how relevant it is for decision support

OLAP methods are regarded as highly valuable for companies in order to exploit e.g. data warehouse system and "make informed tactical and strategic decisions" (Smith 2004, p.376). In the sector of reports, there are different fields of application but all rely on multi-dimensional conceptual views.

#### **4.3.2. OLAP with Microsoft Excel and Oracle SQL**

Performing a multi-dimensional analysis can be done in many ways. There are different tools that relieve this particularly complicated process. Microsoft Excel has been used to analyse records of a three-dimensional cube. Unfortunately, the pivot table of Excel is not well-arranged enough to display the aggregated information. However, the handling is quite easy and Microsoft Excel is definitely a good choice for to analyse simple multi-dimensional models.

The other approach to analyse information is to insert aggregations into the query. Oracle contains a range of commands to retrieve processed data from multi-dimensional cubes (Smith 2004, p.393). Although they are saving time and might be valuable for fast results, they need to be reworked, because they contain unlabeled rows that might not be understandable for uninformed users. Therefore, a direct distribution of database results, e.g. for management, is not recommended.

A more advanced approach, which is recommended for usage in companies, is the Mondrian OLAP Tool (see Chapter 6.).

### 4.3.3. MySQL Workbench 5.1.

The MySQL Workbench is an assisting tool for database modelling. Although the product name contains the word MySQL, it is possible to produce Entity Relationship Diagrams for various relational database management systems. However, the software offers more advanced functionality as well. For example, it is possible to perform Reverse Engineering with MySQL databases. That means that the software connects to an existing database, retrieves the data structure and transforms it into an Entity Relationship Diagram. This visual diagram might be edited or extended and the changes can be applied to the original database as well. Creating and editing tables, columns and relationships is regarded as very simple, because it is supported by wizards.

Alternatively, the free tool “DBDesigner 4”, which is manufactured by fabFORCE, would be a good choice as well. It offers similar functionality (including synchronisation) and is optimised for MySQL but can also handle different database management systems (FabForce.net 2010). These DBMS can be connected using ODBC or specific drivers. The graphical user interface of the diagram designer looks nearly equal to the MySQL Workbench.

### 4.4. References

FABFORCE.NET, 2010. *fabFORCE.net – DB Designer 4* [online]. Available: <http://www.fabforce.net/dbdesigner4/> [accessed 18<sup>th</sup> April 2010]

MALLACH, E. G., 2000. *Decision Support and Data Warehouse Systems*. Boston: McGraw-Hill

MONGER, A. 2010 ? (TODO). *Analyse Multi-Dimensional Data Practical* [online]. Available: <http://mycourse.solent.ac.uk/file.php/1795/DatabaseSystems/Topics/OLAP/OLAP%20Practical.ppt> [accessed 16<sup>th</sup> April 2010]

SMITH, W., 2004. *Decision Support Systems*. Basingstoke: Palgrave Macmillan

## 5. Mining Databases for Decision Support

### 5.1. *Task 1) Design and Implementation of an example Market Basket Analysis*

#### 5.1.1. Choice of Data Mining Tools and Technique

This topic deals with a market basket analysis; therefore a methodology needs to be determined in order to achieve a result. The CRISP methodology (Cross Industry Standard Process for Data Mining) is a promising approach for to perform this analysis. It consists of six steps which are referenced by the arrows in brackets (→ [name of activity]). The result of this analysis should be a rule which supports the management with their decisions.

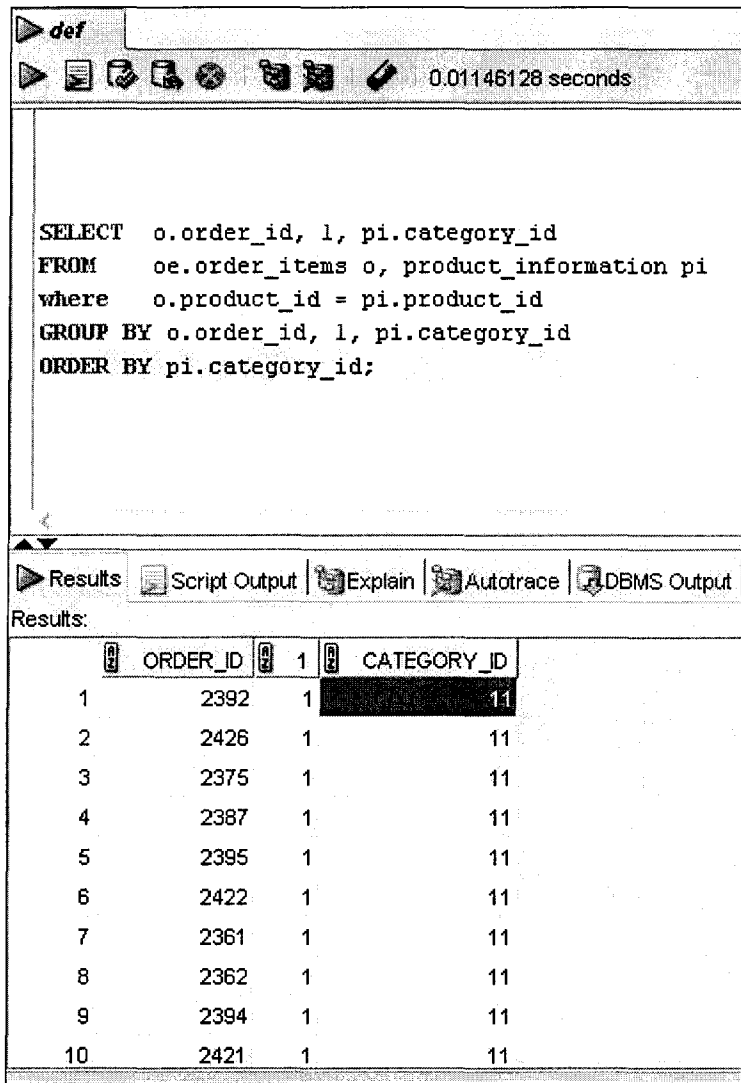
#### 5.1.2. Background Discussion (→ Business / Data Understanding)

The existing OE database contains a vast range of information which might be used for decision support. However, only a small part can be considered for this activity. In order to receive a valuable result, a clearly laid out set of data will be analyzed. Before determining this small area, a detailed knowledge of the data model and therefore the business has to be gained (→ Business Understanding).

An appropriate Assumption, which regards not too many different records, might be to determine, which categories of products are sold together and how they relate to each other (→ Data Understanding).

### 5.1.3. Preparation of required Data (→ Data Preparation)

The first step in order to perform a market basket analysis is to prepare the data for further processing. An example approach is to query a list of orders and the product categories of contained items from the database:



The screenshot shows a database query tool interface. At the top, there is a toolbar with various icons and a timer showing "0.01146128 seconds". Below the toolbar, the SQL query is displayed in a text area:

```
SELECT o.order_id, l, pi.category_id
FROM   oe.order_items o, product_information pi
where  o.product_id = pi.product_id
GROUP BY o.order_id, l, pi.category_id
ORDER BY pi.category_id;
```

Below the query, there is a navigation bar with buttons for "Results", "Script Output", "Explain", "Autotrace", and "DBMS Output". The "Results" button is selected, and the results are displayed in a table below the navigation bar. The table has four columns: "ORDER\_ID", "l", and "CATEGORY\_ID". The first row has a value of "11" in the "CATEGORY\_ID" column, which is highlighted in black.

	ORDER_ID	l	CATEGORY_ID
1	2392	1	11
2	2426	1	11
3	2375	1	11
4	2387	1	11
5	2395	1	11
6	2422	1	11
7	2361	1	11
8	2362	1	11
9	2394	1	11
10	2421	1	11

Figure 21: Retrieving raw Data from the Database

This table can be transferred to Microsoft Excel and transformed into a Pivot Table according to Berry and Linoff's recommendation (2004, p.295):

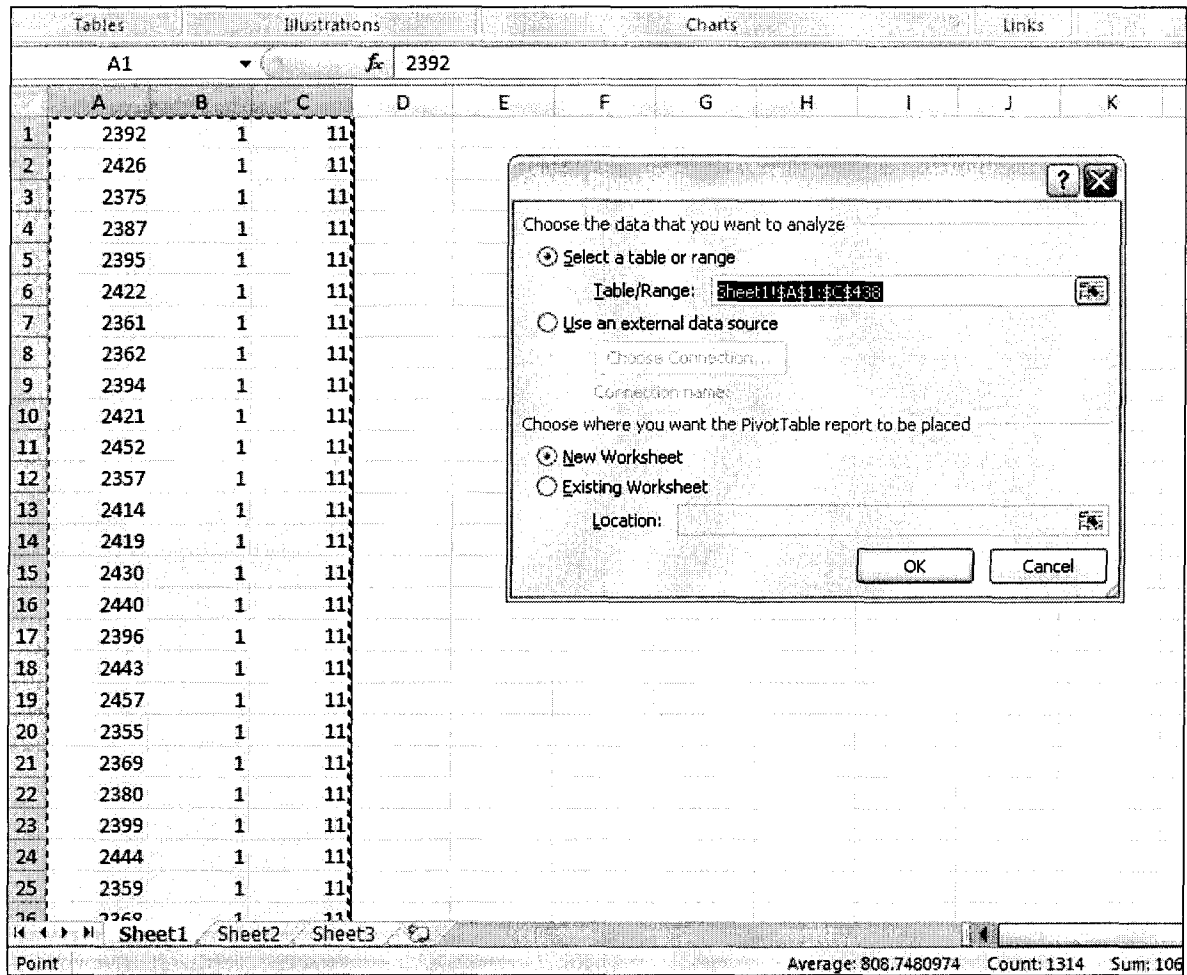


Figure 22: Inserting it to Excel and Creating a Pivot Table

The result of this transformation contains a table which shows the product category IDs as columns and the order IDs on the rows. The number "1" indicates that a certain product category is contained in the corresponding order:

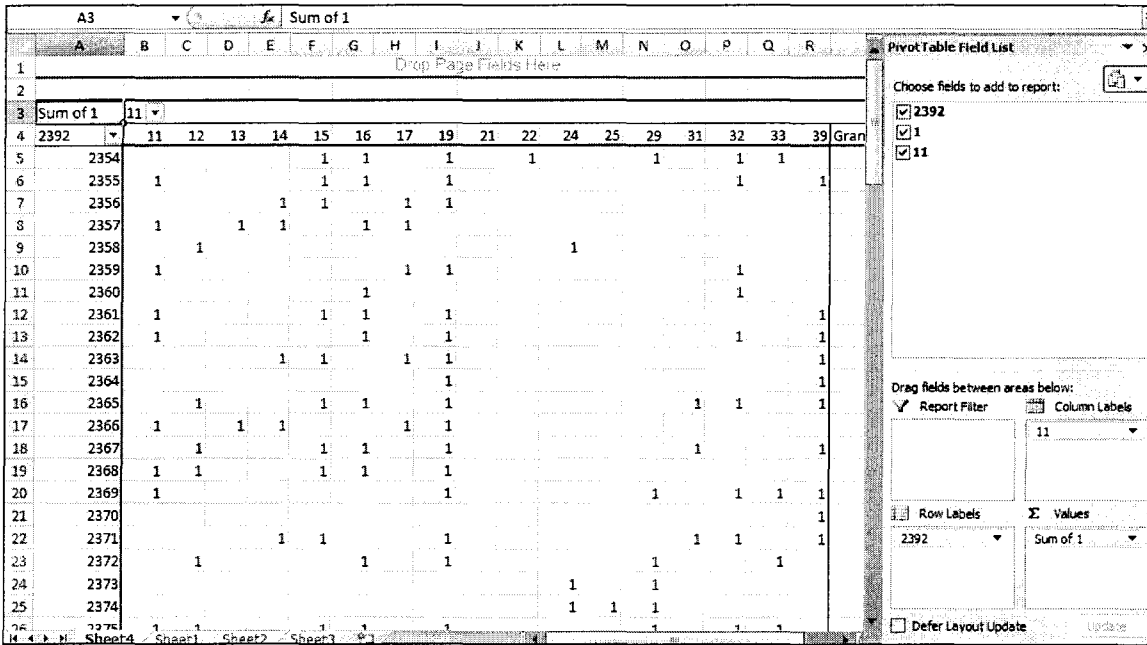


Figure 23: View of the Pivot Table

The final steps in Microsoft Excel are on the one hand to replace all empty cells with quotation marks and the "1" with the string "yes". On the other hand, the produced result needs to be stored in CSV format in order to use it with the WEKA Data Mining Tool (see Chapter 5.2.2.).

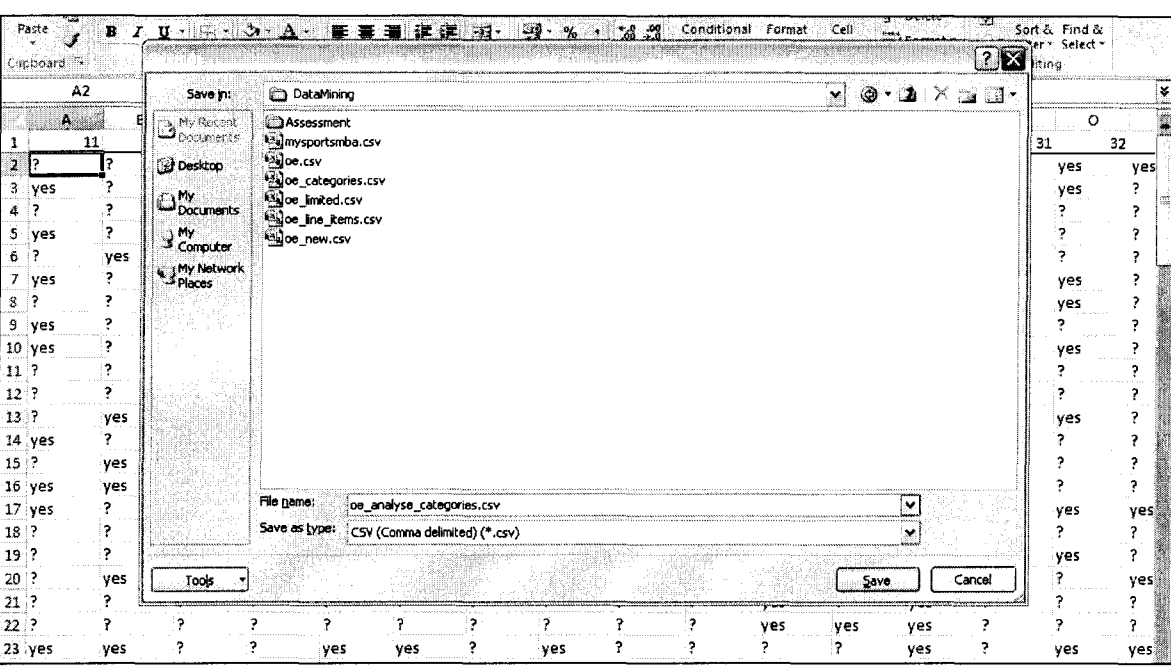


Figure 24: Saving the created Document as a CSV File (Comma Separated Values)



### 5.1.4. Performing the Analysis using WEKA Data Mining Tool (→ Modelling)

The WEKA Data Mining Tool is used for automatic search for rules, because manual research would consume a huge effort, especially when dealing with huge sets of data.

Searching for rules can be done by using the “WEKA Explorer”. At first, the generated CSV file needs to be imported:

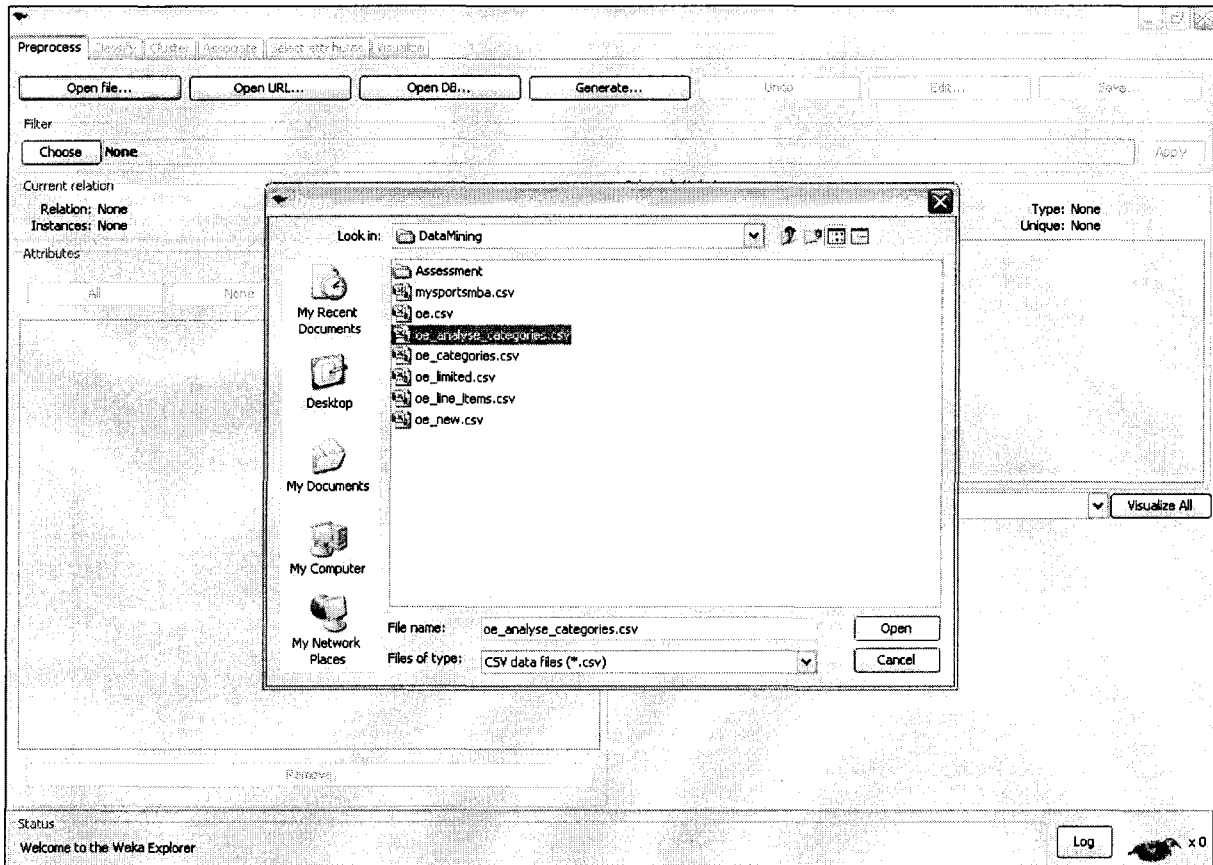
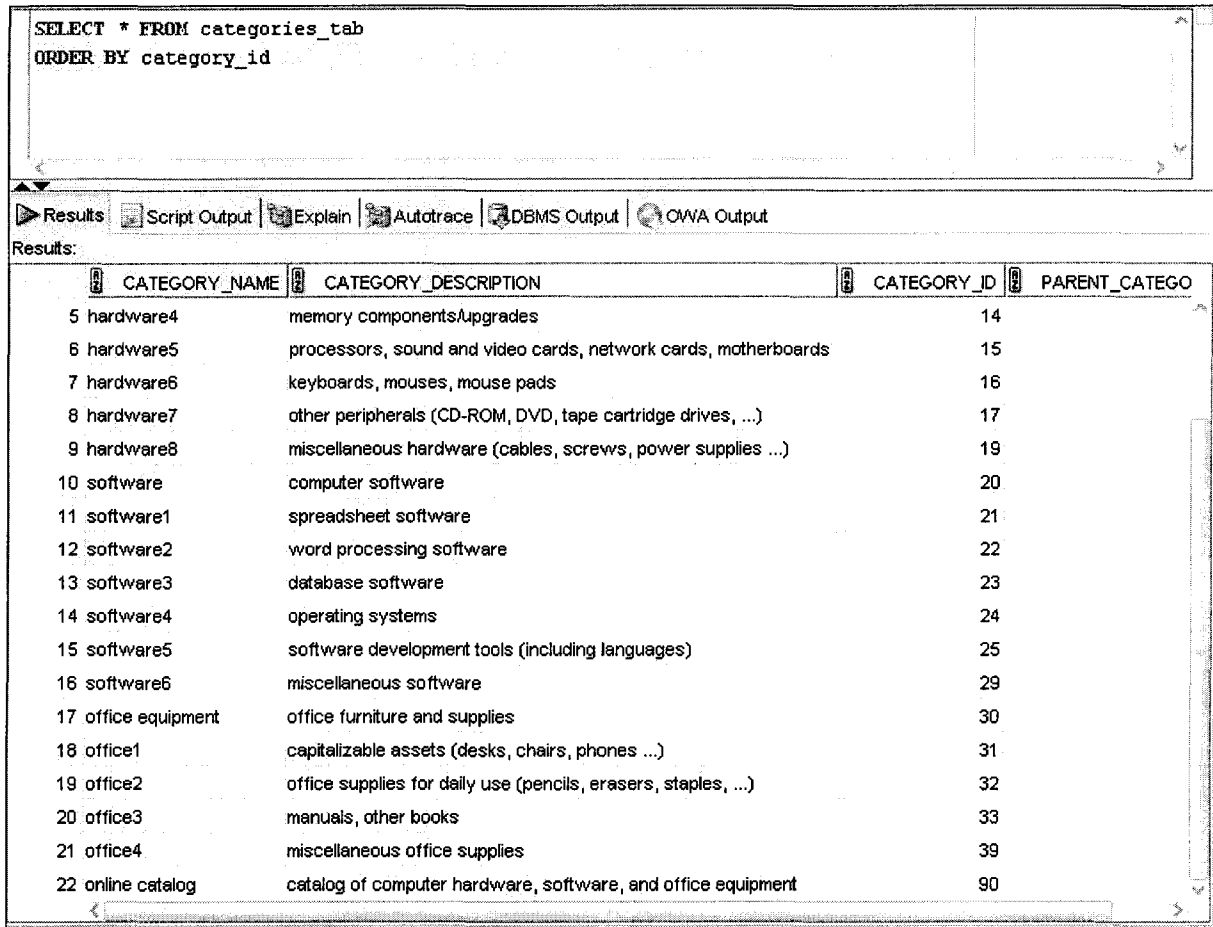


Figure 25: Importing the generated File into WEKA Data Mining Tool

After loading the file, some categories need to be identified for comparison; therefore the IDs have to be looked up for the related text values:



The screenshot shows a database query tool interface. At the top, a text box contains the SQL query: `SELECT * FROM categories_tab ORDER BY category_id`. Below the query box is a toolbar with buttons for **Results**, **Script Output**, **Explain**, **Autotrace**, **DBMS Output**, and **OWA Output**. The **Results** button is selected, and the results are displayed in a table below. The table has four columns: **CATEGORY\_NAME**, **CATEGORY\_DESCRIPTION**, **CATEGORY\_ID**, and **PARENT\_CATEGO**. The results are as follows:

CATEGORY_NAME	CATEGORY_DESCRIPTION	CATEGORY_ID	PARENT_CATEGO
5 hardware4	memory components/upgrades	14	
6 hardware5	processors, sound and video cards, network cards, motherboards	15	
7 hardware6	keyboards, mice, mouse pads	16	
8 hardware7	other peripherals (CD-ROM, DVD, tape cartridge drives, ...)	17	
9 hardware8	miscellaneous hardware (cables, screws, power supplies ...)	19	
10 software	computer software	20	
11 software1	spreadsheet software	21	
12 software2	word processing software	22	
13 software3	database software	23	
14 software4	operating systems	24	
15 software5	software development tools (including languages)	25	
16 software6	miscellaneous software	29	
17 office equipment	office furniture and supplies	30	
18 office1	capitalizable assets (desks, chairs, phones ...)	31	
19 office2	office supplies for daily use (pencils, erasers, staples, ...)	32	
20 office3	manuals, other books	33	
21 office4	miscellaneous office supplies	39	
22 online catalog	catalog of computer hardware, software, and office equipment	90	

Figure 26: Looking Up the Database for Product Categories and their Descriptions

One optional step is to exclude product categories of cheaper value, which might be so called items of category C (Wan 2008). This step is performed by sorting the average price of items of one category and deciding to exclude the ones that are below a defined price:

```
SELECT category_id, Round( avg(min_price),2)
FROM product_information
GROUP BY category_id
ORDER BY avg(min_price)
```

Results: Script Output Explain Autotrace DBMS

	CATEGORY_ID	ROUND(AVG(MIN_PRICE),2)
1	39	23.95
2	16	26.9
3	32	35.45
4	21	36.57
5	22	45
6	33	47.17
7	29	84.71
8	15	102.25
9	17	109.05
10	25	161.88
11	24	169.77
12	14	202.92
13	12	251.67
14	19	254.96
15	13	441.5
16	11	469.81
17	31	1722.2

Figure 27: Analysing the Average Prices of Products from each Category

In this case, all products with an average price of less than "50.00" are excluded. The WEKA tool offers the possibility to remove certain records from the list of categories:

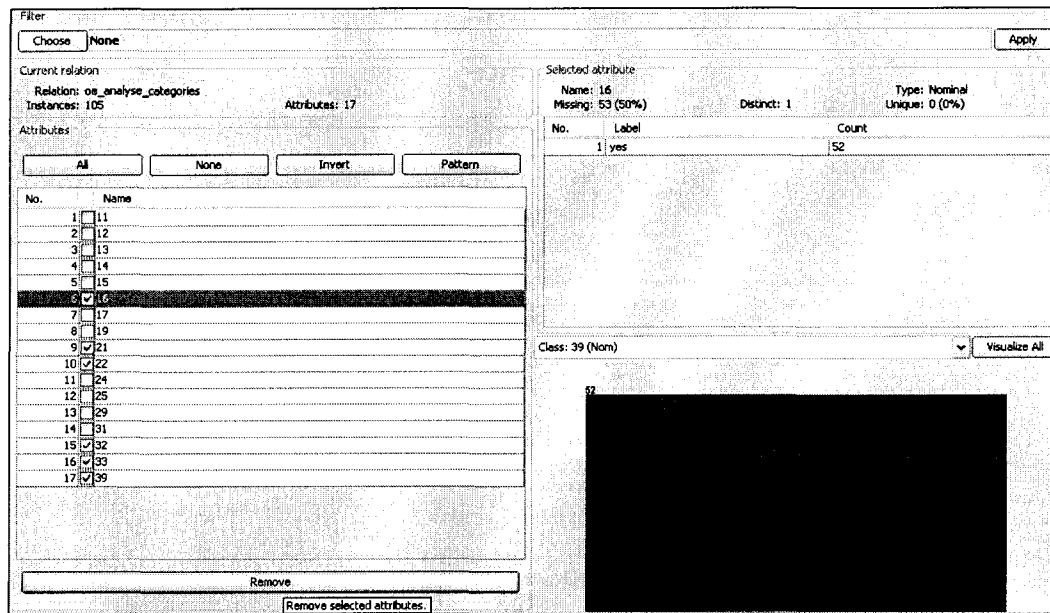


Figure 28: Removing irrelevant Categories

After this step, the actual analysis can be performed by choosing the "Associate" tab pane and starting the analysis. The result looks like this and shows two found rules with a quite high confidence:

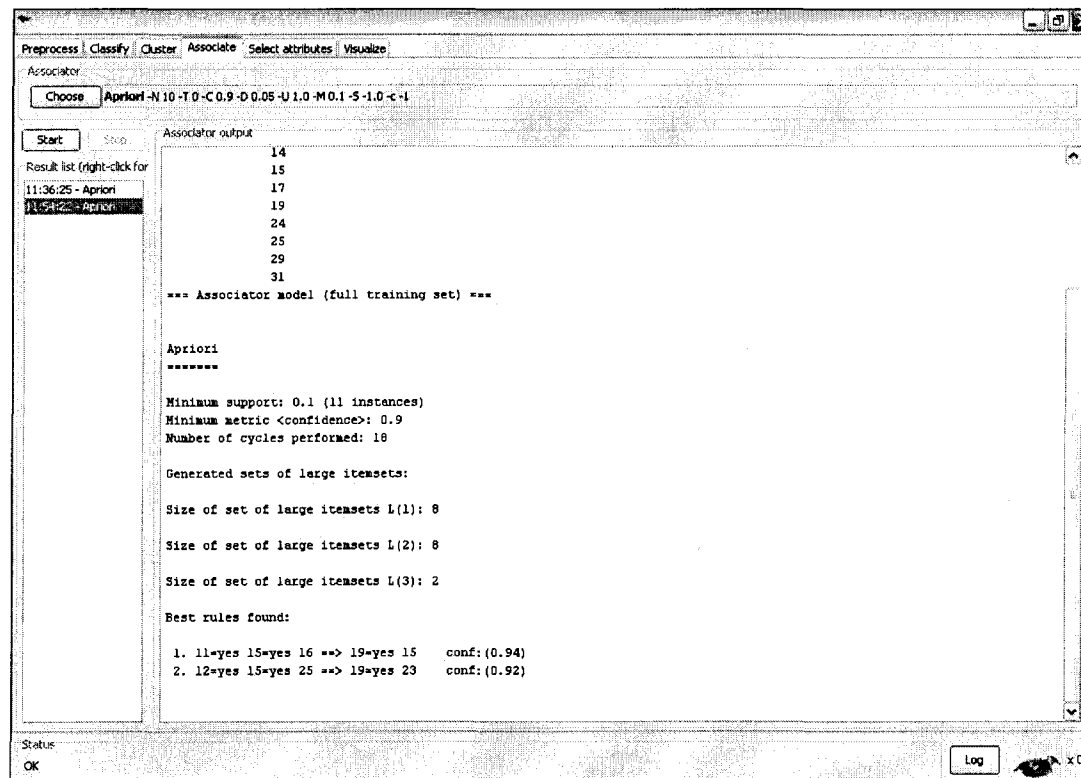


Figure 29: Result of an Example Search for Association Rules

These rules state that:

- Whenever products of category 11 and category 15 are sold together, there is a 94% chance, that products of category 19 are sold as well in this order
- Whenever products of category 12 and category 15 are sold together, there is a 92% chance, that products of category 19 are sold as well in this order

However, there is the possibility to explore even more rules by adjusting the settings for analysis. The value of minimum confidence can be decreased to e.g. 70% (0.7) – by default, it is set to 90%:

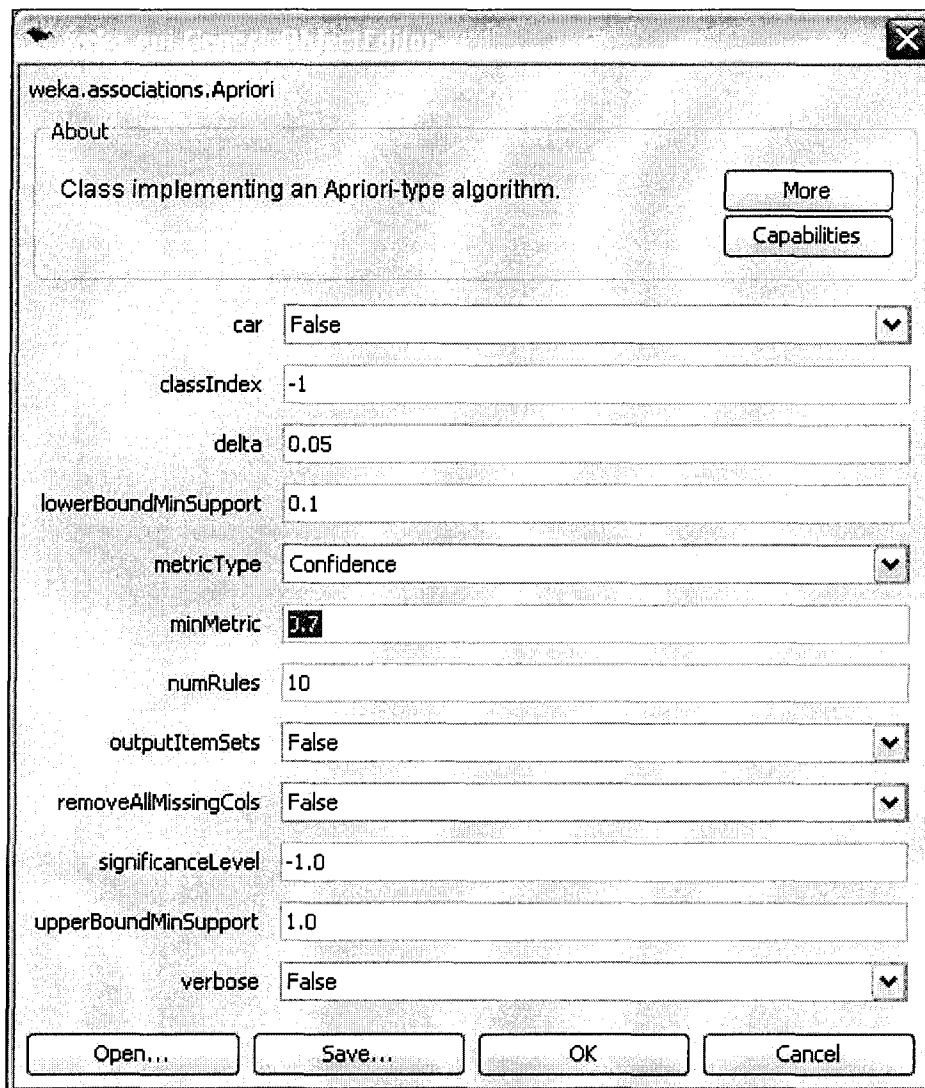


Figure 30: Adjustment of Settings for Finding Association Rules

Therefore, the result of a new analysis shows more rules:

```
Best rules found:

1. 11=yes 15=yes 16 ==> 19=yes 15    conf: (0.94)
2. 12=yes 15=yes 25 ==> 19=yes 23    conf: (0.92)
3. 12=yes 33 ==> 19=yes 29    conf: (0.88)
4. 15=yes 49 ==> 19=yes 41    conf: (0.84)
5. 12=yes 19=yes 29 ==> 15=yes 23    conf: (0.79)
6. 14=yes 14 ==> 19=yes 11    conf: (0.79)
7. 17=yes 17 ==> 19=yes 13    conf: (0.76)
8. 12=yes 33 ==> 15=yes 25    conf: (0.76)
9. 11=yes 32 ==> 19=yes 23    conf: (0.72)
```

Figure 31: Example Result for an Analysis with lower Confidence

Data quality is another important aspect of meaningful data mining. Usually, data quality in market basket analysis is not really high (Berry and Linoff 2004, p.308). An approach to increase quality has been undertaken as follows:

In the figure above, the product category 19 is contained in nearly every rule. A look up for the name of the category shows why: it is called “miscellaneous hardware (cables, screws, power supplies...)”, which means it contains mainly mass products. It is seen as necessary to remove category 19 in order to achieve meaningful rules.

The final result of the WEKA Data Mining Tool analysis looks like this:

```
Apriori -N 10 -T 0 -C 0.5 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

Start Stop
Result list (right-click for)
11:36:25 - Apriori
11:54:22 - Apriori
12:02:44 - Apriori
12:03:43 - Apriori

13
14
15
17
24
25
29
31

=== Apriori model (full training set) ===

Apriori
=====

Minimum support: 0.1 (11 instances)
Minimum metric <confidence>: 0.5
Number of cycles performed: 18

Generated sets of large itemsets:

Size of set of large itemsets L(1): 7
Size of set of large itemsets L(2): 2

Best rules found:

1. 12=yes 33 ==> 15=yes 25    conf: (0.76)
2. 15=yes 49 ==> 12=yes 25    conf: (0.51)
3. 11=yes 32 ==> 15=yes 16    conf: (0.5)
```

Figure 32: Result after excluding Category ID 19

### 5.1.5. Significance of discovered Rules for Decision Support (→ Evaluation)

The found rules state, that:

- On the one hand, in 76% of the orders where a product of category 12 has been sold, a product of category 15 has been sold as well.
- The other way round, in just 51% of the orders where a product of category 15 has been sold, a product of category 12 has been sold as well.

Nevertheless, there must be a relationship between these products, and a decision for the business could for example be:

- Place items of these categories next to each other in the shop in order to allow easier access for the customer
- Spend further effort in investigating which particular items are related to each other

However, it needs to be said that there are many more possible rules and combinations, or even other data mining problems to be solved with this methodology.

## 5.2. Evaluation of Technologies, Tools and Methods

### 5.2.1. CRISP Methodology

The CRISP methodology has been conceived in 1996 by a group of three industrial companies: Daimler-Benz, ISL and NCR. The first quality draft of the process model has been published in 1999 (CRISP-DM Consortium 2000). Therefore, this process model is used in different independent industries for over 10 years now. A small (and therefore not really representative) poll within the data mining web community "KDnuggets.com" asked the users for their preferred data mining methodologies. The result shows that 42% of them are using the CRISP-DM; followed by individual approaches and the SEMMA process (KDnuggets 2007).

Using the CRISP methodology is regarded as easy on the one hand, because the six steps are clearly described and documented. On the other hand, it could turn into very complex activities, especially when dealing with more complicated techniques than the market basket analysis by finding association rules. According to Thearling (2009), some of these techniques might be:

- Classification and Clustering → splits data sets into groups; measures and evaluates if they are "close to" or "far from" each other
- Retrospective Data Analysis → "Data analysis that provides insights into trends, behaviours, or events that have already occurred." (Thearling 2009)

In summary, CRISP is recommended for usage, because it is accepted by the industry (CRISP-DM Consortium 2000) and data mining specialists as well (KDnuggets 2007).

### 5.2.2. WEKA Data Mining Tool

The WEKA Data Mining Tool is available in two ways. There is a cost free standalone solution, but it is also included in the Pentaho Open Source Business Intelligence Suite (Pentaho Corporation 2010).

During the market basket analysis, it has been used for automatic search of association rules. After a short introduction phase, the basic steps in order to achieve a result have been performed without major problems. Nevertheless, the scope of the software is regarded as vast, because there are many other features and options which might be valuable for different kinds of analysis. Therefore, the handling of this software could be complicated and the first impression of the interface might scare unaware users.

Beside the semi-manual approach by using Excel as an intermediate application for preparation, the WEKA Data Mining Tool is able to directly connect to databases as well. However, this is not regarded as part of the performed tasks and might belong to the advanced topic research.

In summary, the WEKA Data Mining Tool meets all requirements for performing this market basket analysis. Assumed that all features of the program are explored, it is providing great facilities for to analyse business data and therefore derive decisions. Especially larger companies can profit from this powerful software by using it for to effectively use their data warehouses.

### 5.3. References

BERRY, M.J.A. and G. LINOFF, 2004. *Market Basket Analysis and Association Rules*. London: Wiley

KDNUGGETS, 2007. *Poll: Data Mining Methodology* [online]. Available: [http://www.kdnuggets.com/polls/2007/data\\_mining\\_methodology.htm](http://www.kdnuggets.com/polls/2007/data_mining_methodology.htm) [accessed 26th April 2010]

THEARLING, K., 2009. *An introduction to Data Mining* [online]. Available: <http://www.thearling.com/text/dmwhite/dmwhite.htm> [accessed 26th April 2010]



WAN, D., 2008. *ABC Analysis in Inventory Management* [online]. Available:  
<http://dylanwan.wordpress.com/2008/03/11/abc-analysis-in-inventory-management/>  
[accessed 25<sup>th</sup> April 2010]

## 6. Mondrian OLAP Tool

### 6.1. Introduction to Mondrian

Mondrian OLAP Tool is the name of a web application that is used for multidimensional exploration of databases. Although it is available as an open source product, it is part of the Pentaho Open Source Business Intelligence suite as well (Pentaho Corporation 2010, "Pentaho Reporting Enterprise Edition"). This commercial suite contains a range of analysis; data integration and reporting software like the WEKA Data Mining Tool (see Chapter 5.2.2.). Mondrian has the ability to deal with large datasets, for example data warehouses. Instead of common SQL statements, it mainly uses Microsoft's MDX (Multi Dimensional eXpression) technology to query databases in various dimensions (Microsoft Corporation 2010, "Multidimensional Expressions (MDX) Reference"). Primarily, it is used for reporting; therefore it accesses the target database for read-only queries. Mondrian makes meaningful usage of standard DBMS aggregation functionality like "GROUP BY" statements or materialised views (Hyde 2006, "Architecture").

Due to the fact that Mondrian OLAP Tool is a web application, the usage is completely browser-based. All menus, which actually are pages, need to be created as Java Server Pages (JSP). The contents of these pages can be customised and may range from simple MDX query interfaces to pivot tables or chart views. Depending on the setup, the start screen shows all available menus.

In this case study, the pivot table component "JPivot" is used because it provides comfortable exploration options and is regarded as easy-to-use. The user interface consists of graphical buttons that includes so called "tooltip" descriptions as well. Exploration of the different dimensions behaves like a tree navigation that means the levels can be expanded step-by-step:

Product	Promotion Media	Unit Sales
-All Products	+All Media	266.773
+Drink	+All Media	24.597
-Food	+All Media	191.940
-Baked Goods	+All Media	7.870
-Bread	+All Media	7.870
+Bagels	+All Media	815
-Muffins	+All Media	3.497
+Colony	+All Media	740
+Fantastic	+All Media	798
-Great	+All Media	605
Great Blueberry Muffins	+All Media	127
Great Cranberry Muffins	+All Media	165
Great English Muffins	+All Media	164
Great Muffins	+All Media	149
+Modell	+All Media	719
+Sphinx	+All Media	635
+Sliced Bread	+All Media	3.558
+Baking Goods	+All Media	20.245
+Breakfast Foods	+All Media	3.317
+Canned Foods	+All Media	19.026
+Canned Products	+All Media	1.812

Figure 33: Exploring the dimensions in a JPivot table

Information can be displayed in different ways, e.g. chart views:

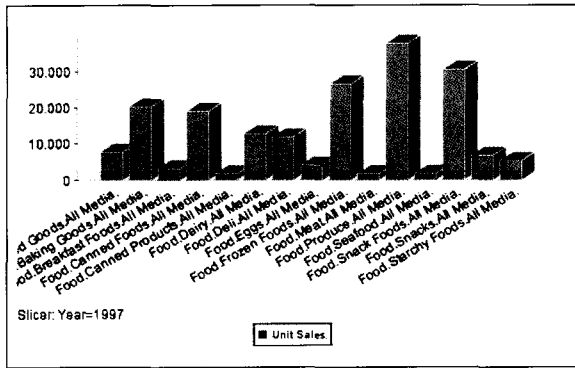


Figure 34: Example Vertical Bar Chart

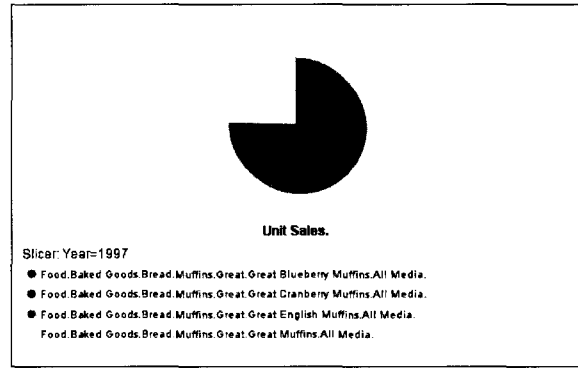


Figure 35: Example Pie Chart

An advantage of the web application concept is that a range of different users can access the same application and related data pool. However, it is not always intended to distribute data for everyone, especially when dealing with sensitive data like customer behaviour or even employee data. Therefore, security can be increased by using roles. These roles might be regional or hierarchical limited, e.g. the managers for Europe can only access sales statistics for Europe or an HR manager just sees information of his employees but not about sales.

### 6.2. Scenario

A company decided to establish a data warehouse in order to generate sales and turnover reports for the management and major shareholders. Originally, it was intended to record only sales data by product and area, but the database developers included all available data in this data warehouse. One of these dimensions covers the promotion media that advertised the products. This information has been elicited by surveying customers at point-of-service.

The management intends to start a new marketing campaign for to increase sales of Dairy products. The database expert is asked to provide a report that contains Dairy product sales statistics of the year 1997 grouped by promotion media types that caused selling in order to determine the most effective one. Furthermore, the manager mentions that this kind of information will be frequently requested in future with different products or product groups. The database expert suggests trying the Mondrian OLAP Tool.

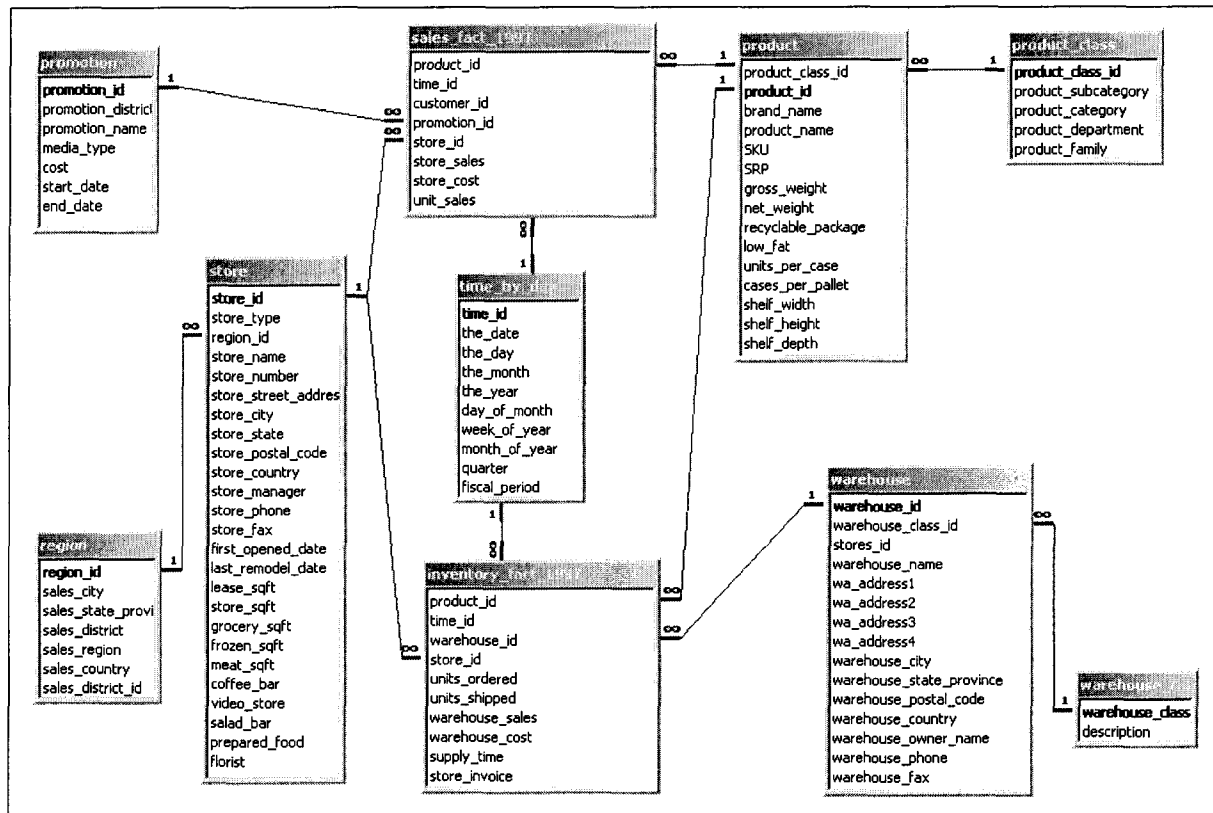


Figure 36: Entity Relationship Diagram of Company's Data Warehouse

### 6.3. Analysis, Design, Implementation and Test

#### 6.3.1. Analysis

The first step when using the Mondrian OLAP Tool is to analyse the required data and data sources:

Which data dimensions are needed?

- Time (In fact, it is determined to 1997 because the demo database only contains complete sample data for this year)
- Products
- Promotion Media
- Sales Data

Which tables hold this information and which related tables are involved?

- The "sales\_fact\_1997" table is similar to a materialised view and contains all relevant sales data
- Foreign key "time\_id" links to the table "time\_by\_day" in which the year is stored
- Foreign key "product\_id" links to the "products" table and contains detailed product data like the product and brand name. Each product is categorised into product classes; therefore the table "product\_class" is involved as well
- Foreign key "time\_id" is linking to the "time" table and is required as well, because it contains the information about the year

#### 6.3.2. Design

Generally, the structured query for this report can be described as follows:

"Select sales data of dairy products of the year 1997 and group them by the promotion media which animated the customer to buy it."

The developers of the Mondrian OLAP Tool are quoting as a key feature that no knowledge about SQL is needed because it uses MDX. That is why a transcription of this query into MDX is needed for to implement this scenario:

```
SELECT
  {[Measures].[Unit Sales]} ON columns,
  {[([Product].[All Products], [Promotion Media].[All Media])}] ON rows
FROM Sales
WHERE ([Time].[1997])
```

This MDX query can not really be compared to SQL because it is able to define different dimensions (Microsoft Corporation 2010, "Comparison of SQL and MDX"):

- The values of "Unit Sales" are listed on the columns
- The Dimensions Products and Promotion Media are listed on the rows

### 6.3.3. Implementation

The first step to achieve a result is to install the Mondrian OLAP Tool. For this project, the Mondrian Demo database “FoodMart” is used. Subject to the condition that a Java Application Server is already installed, configured and properly working, the setup of Mondrian is done in the following way:

1. Installation of the web application on an Application Server for Java, e.g. Tomcat. Mondrian is delivered as a WAR-File (Web Application Archive) and can be deployed in Tomcat using the Tomcat Web Application Manager:

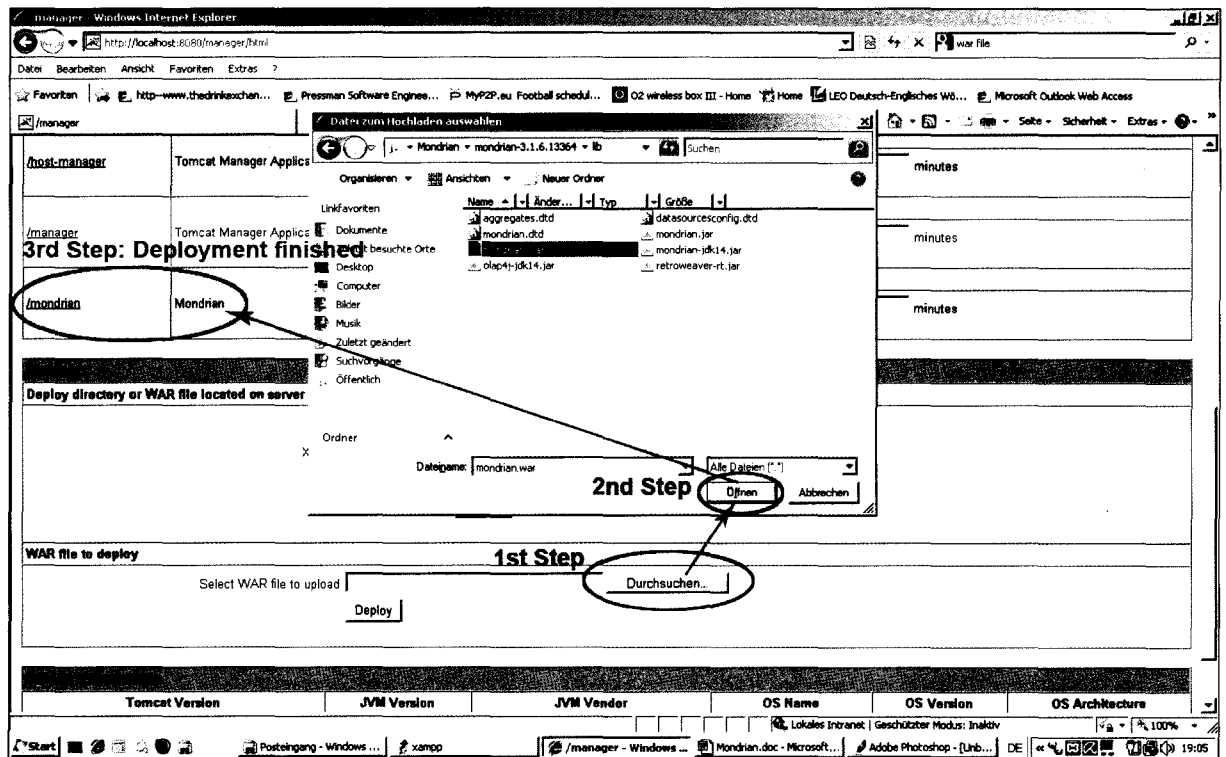


Figure 37: Deployment of Mondrian OLAP Tool on a Tomcat Web Server

2. Connecting to the database can be configured in an XML file called “datasources.xml”. The technical recommendation for database connections is JDBC, e.g. a JDBC ODBC. An ODBC data source with the name “FoodMart” has been created as a “System DSN”. The connection string is inserted into the configuration XML file:

```
<DataSource>
<DataSourceName>Provider=Mondrian;DataSource=MondrianFoodMart;</DataS
ourceName>
[... ]
<DataSourceInfo>Provider=mondrian;Jdbc=jdbc:odbc:FoodMart;JdbcDrivers
=sun.jdbc.odbc.JdbcOdbcDriver;Catalog=/WEB-
INF/queries/FoodMart.xml</DataSourceInfo>
[... ]
</DataSource>
```

The implementation progresses by defining the desired data in the “FoodMart.xml” file. Dimensions and their links to the database tables are configured as a so called “schema”. It has been created using the Mondrian online documentation (Hyde 2009, “How to Design a Mondrian Schema”) and elements from the demo project (Hyde 2009, “Set up test data in a non-embedded database”).

The “Time” Dimension is defined like this:

```
<Dimension name="Time" type="TimeDimension">
  <Hierarchy hasAll="false" primaryKey="time_id">
    <Table name="time_by_day"/>
    <Level name="Year" column="the_year" type="Numeric"
      uniqueMembers="true" levelType="TimeYears"/>
  </Hierarchy>
</Dimension>
```

**Listing 26: Definition for the "Time Dimension"**

The “Product” dimension is more complex because it has relations to the “product\_class” table and contains so called levels. Each level aggregates the values below and can be regarded a category. The configuration works as follows:

```
<Dimension name="Product">
  <Hierarchy hasAll="true" primaryKey="product_id"
    primaryKeyTable="product">
    <Join leftKey="product_class_id" rightKey="product_class_id">
      <Table name="product"/>
      <Table name="product_class"/>
    </Join>
    <Level name="Product Family" table="product_class"
      column="product_family" uniqueMembers="true"/>
    <Level name="Product Department" table="product_class"
      column="product_department" uniqueMembers="false"/>
    <Level name="Product Category" table="product_class"
      column="product_category" uniqueMembers="false"/>
    <Level name="Product Subcategory" table="product_class"
      column="product_subcategory" uniqueMembers="false"/>
    <Level name="Brand Name" table="product" column="brand_name"
      uniqueMembers="false"/>
    <Level name="Product Name" table="product" column="product_name"
      uniqueMembers="true"/>
  </Hierarchy>
</Dimension>
```

**Listing 27: Definition of the "Product Dimension"**

The “Promotion Media” contains only one level that lists all types of media:

```
<Dimension name="Promotion Media">
  <Hierarchy hasAll="true" allMemberName="All Media"
    primaryKey="promotion_id" defaultMember="All Media">
```

```

    <Table name="promotion"/>
    <Level name="Media Type" column="media_type" uniqueMembers="true"/>
  </Hierarchy>
</Dimension>

```

**Listing 28: Definition of the "Promotion Media Dimension"**

The Sales data are configured as a cube. Beside the source table, the predefined dimensions are connected to the foreign key columns. The numbers of sold products are registered in the "Measure" tag with the aggregation function to summarise the sold numbers:

```

<Cube name="Sales" defaultMeasure="Unit Sales">
  <Table name="sales_fact_1997">
    <AggName name="agg_c_special_sales_fact_1997">
      <AggFactCount column="FACT_COUNT"/>
      <AggIgnoreColumn column="foo"/>
      <AggIgnoreColumn column="bar"/>
      <AggForeignKey factColumn="product_id" aggColumn="PRODUCT_ID" />
      <AggForeignKey factColumn="customer_id" aggColumn="CUSTOMER_ID" />
      <AggForeignKey factColumn="promotion_id" aggColumn="PROMOTION_ID" />
      <AggMeasure name=" [Measures].[Unit Sales]" column="UNIT_SALES_SUM" >
      <AggLevel name=" [Time].[Year]" column="TIME_YEAR" />
    </AggName>
  </Table>
  <DimensionUsage name="Time" source="Time" foreignKey="time_id"/>
  <DimensionUsage name="Product" source="Product" foreignKey="product_id"/>
  <DimensionUsage name="Promotion Media" source="Promotion Media"
    foreignKey="promotion_id"/>
  <Measure name="Unit Sales" column="unit_sales" aggregator="sum"
    formatString="Standard"/>
</Cube>

```

**Listing 29: Configuration of the Cube**

The final step in order to display a pivot table is to create a JSP that contains a JPivot web control:

```

<%@ page session="true" contentType="text/html; charset=ISO-8859-1" %>
<%@ taglib uri="http://www.tonbeller.com/jpivot" prefix="jp" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>

<jp:mondrianQuery id="query01" jdbcDriver="sun.jdbc.odbc.JdbcOdbcDriver"
jdbcUrl="jdbc:odbc:MondrianFoodMart" catalogUri="/WEB-INF/queries/FoodMart.xml">
select
  {[Measures].[Unit Sales]} ON columns,
  {[([Product].[All Products], [Promotion Media].[All Media])}] ON rows
from Sales
where ([Time].[1997])
</jp:mondrianQuery>

```

**Listing 30: Java Server Page that displays a pivot table for the example database**



### 6.3.4. Test

The example shows the requested data in a JPivot view. JPivot is another open source project that can be used within Mondrian (Tonbeller AG 2010). It is a comfortable way to display multidimensional data in a pivot table and contains several features like a chart generator.

The pivot table needs to be explored in order to separate and sort the data according to the management's request:

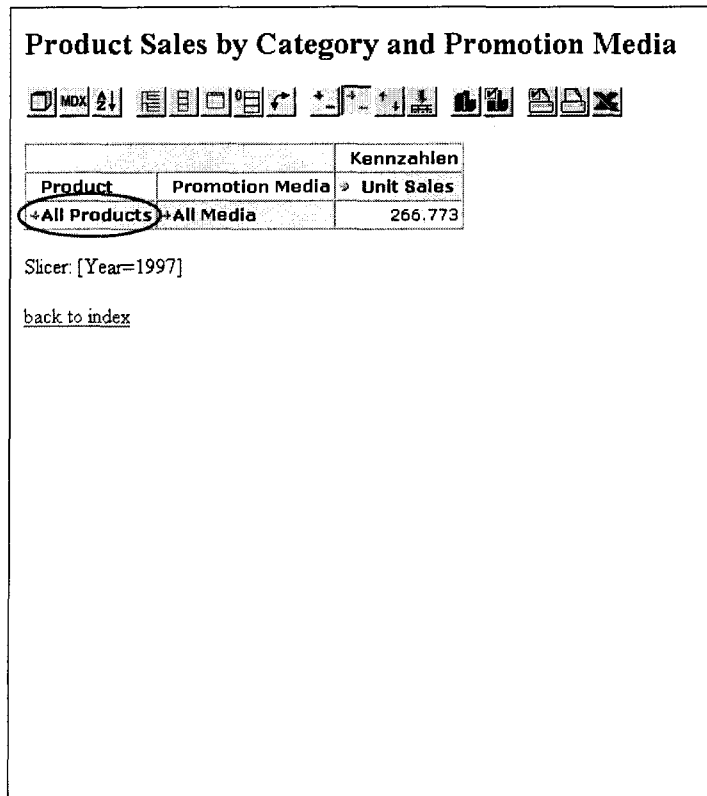


Figure 38: Default view of pivot table; Expand the product tree until "Dairy Products" are visible

### Product Sales by Category and Promotion Media

Product	Promotion Media	Kennzahlen Unit Sales
-All Products	+All Media	266.773
+Drink	+All Media	24.597
-Food	+All Media	191.940
+Baked Goods	+All Media	7.870
+Baking Goods	+All Media	20.245
+Breakfast Foods	+All Media	3.317
+Canned Foods	+All Media	19.026
+Canned Products	+All Media	1.812
+Dairy	+All Media	12.885
+Deli	+All Media	12.037
+Eggs	+All Media	4.132
+Frozen Foods	+All Media	26.655
+Meat	+All Media	1.714
+Produce	+All Media	37.792
+Seafood	+All Media	1.764
+Snack Foods	+All Media	30.545
+Snacks	+All Media	6.884
+Starchy Foods	+All Media	5.262
+Non-Consumable	+All Media	50.236

Slicer: [Year=1997]

Figure 39: Detailed Product Class view; Expand Media in order to get a detailed list of all Promotion Media Types

### Product Sales by Category and Promotion Media

Product	Promotion Media	Kennzahlen Unit Sales
-All Products	+All Media	266.773
+Drink	+All Media	24.597
-Food	+All Media	191.940
+Baked Goods	+All Media	7.870
+Baking Goods	+All Media	20.245
+Breakfast Foods	+All Media	3.317
+Canned Foods	+All Media	19.026
+Canned Products	+All Media	1.812
+Dairy	-All Media	12.885
	Bulk Mail	189
	Cash Register Handout	339
	Daily Paper	349
	Daily Paper, Radio	327
	Daily Paper, Radio, TV	454
	In-Store Coupon	166
	No Media	9.510
	Product Attachment	355
	Radio	112
	Street Handout	301
	Sunday Paper	200
	Sunday Paper, Radio	283

Figure 40: Detailed view for Dairy Products and Promotion Media Types; Change view to only display Dairy Products

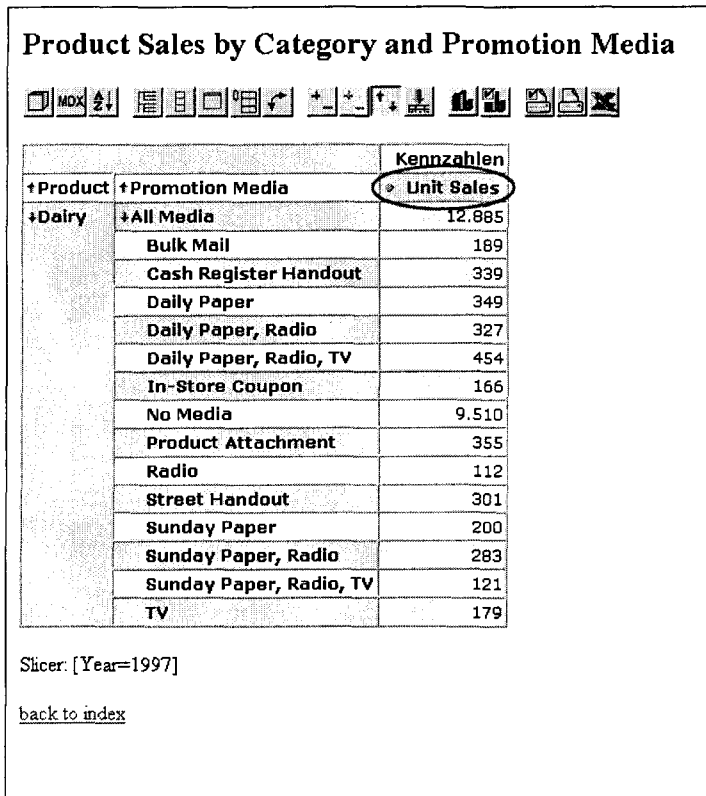


Figure 41: Drilled view with detailed data; Sort Unit Sales descending

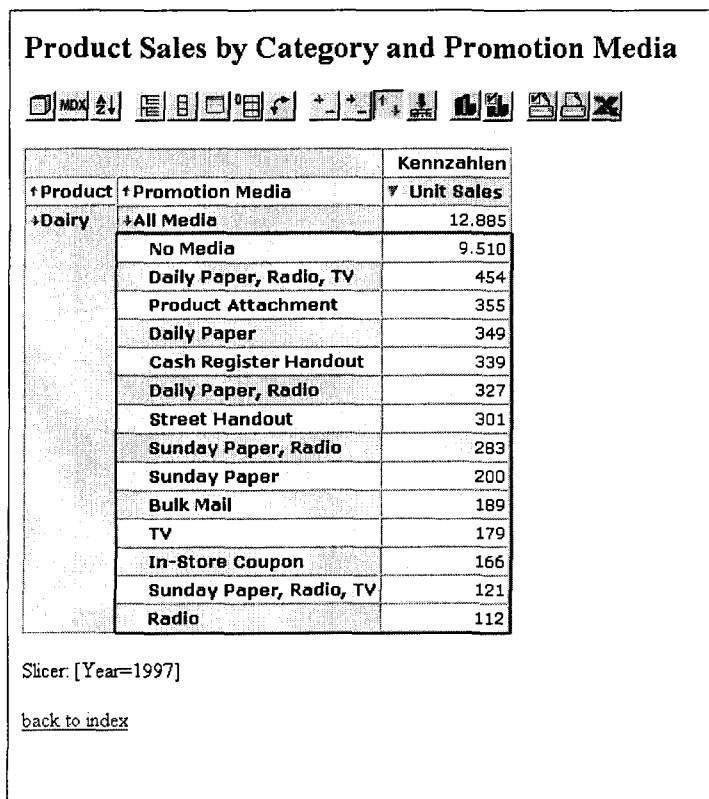


Figure 42: Final view that contains the desired data

#### **6.4. Evaluation of Technologies, Tools and Methods**

Basically, Mondrian OLAP Tool is regarded as helpful software for analysis of data. Using JDBC as the driver technology behind makes it quite flexible in terms of the underlying database management system. Installation and basic configuration of the core system is practicable for experienced developers with skills on web applications and web server administration.

Besides the fact, that distinctive knowledge of the data model or data warehouse is essential for all OLAP activities, a key task for working with Mondrian is the specification of the cube. Configuration of the so called "schema" might become difficult, especially when dealing with more complex data structures. Nevertheless, a comprehensively written documentation and a demonstration for Mondrian are available (Hyde 2009, "Documentation"). It is worth mentioning that basic knowledge of the MDX query language is indispensable, but Microsoft provides accompanying documentation in the Microsoft Developers Network (Microsoft Corporation 2010, "Multidimensional Expressions (MDX) Reference").

Although being open source software, Mondrian seems to be technically-matured and reliable. No bugs or compatibility issues have occurred during implementation of the scenario. Mondrian emphasises that system performance was one of the main development objectives (Hyde 2007, "Performance"):

- Meaningful usage of existing DBMS functionality
- Extensive usage of caching techniques and garbage collection
- Developers are giving further advice on how to optimise the database for usage with Mondrian

Mondrian fully meets the needs for the given scenario and is likely to meet most business requirements for online analytical processing. However, it might concern the management that the software is an open source project. The reasons could be concerns about reliability, missing responsibility or nonexistent service and support. Therefore, the Pentaho Open Source Business Intelligence suite would be an alternative, which includes an extended version of Mondrian OLAP Tool for commercial usage. It goes without saying that this software is not available for free. Licenses for the "Pentaho Analysis" software start at around USD 4,000 and the whole Pentaho BI Suite license starts at USD 10.000 (Pentaho Corporation 2010, "Pentaho Enterprise Edition Pricing"). Pentaho is highly accepted within the industry and is used by companies and organisations like Ericsson, NHS and T-Mobile (Pentaho Corporation 2010).

Another open-source alternative is the BIRT (Business Intelligence and Reporting Tools) project which is a reporting system that is based on Eclipse (see Chapter 2.2.3.). The intended usage of BIRT is to integrate reports in web applications. The steps to achieve a result are (The Eclipse Foundation 2010, "BIRT Overview"):

- Access the data source
- Transform data into data sets
- Add business logic if required
- Present the data as a report

Although these key tasks are quite similar to Mondrian, the focus of BIRT rather seems to be the optical design of the report. Handling of BIRT is a slightly more intuitive than Mondrian because BIRT delivers a fully developed GUI with corresponding wizards (The Eclipse Foundation 2010, "BIRT Demo").

*good evaluation*

## 6.5. References

HYDE, J., 2006-2009. *Pentaho Commercial Open Source Business Intelligence* [online]. Available: <http://mondrian.pentaho.org/documentation/> [accessed 4<sup>th</sup> April 2010]

MICROSOFT CORPORATION, 2010. *Multidimensional Expressions (MDX) Reference* [online]. Available: <http://msdn.microsoft.com/en-us/library/ms145506.aspx> [accessed 4th April 2010]

MICROSOFT CORPORATION, 2010. *Comparison of SQL and MDX* [online]. Available: [http://msdn.microsoft.com/en-us/library/aa216779\(SQL.80\).aspx](http://msdn.microsoft.com/en-us/library/aa216779(SQL.80).aspx) [accessed 4<sup>th</sup> April 2010]

PENTAHO CORPORATION, 2010. *Open Source Business Intelligence* [online]. Available: <http://www.pentaho.com/> [accessed 4<sup>th</sup> April 2010]

THE ECLIPSE FOUNDATION, 2010. *Eclipse BIRT Home* [online]. Available: <http://www.eclipse.org/birt/phoenix/> [accessed 6<sup>th</sup> April 2010]

TONBELLER AG, 2010. *JPivot – Home* [online]. Available: <http://jpivot.sourceforge.net/index.html> [accessed 8th April 2010]