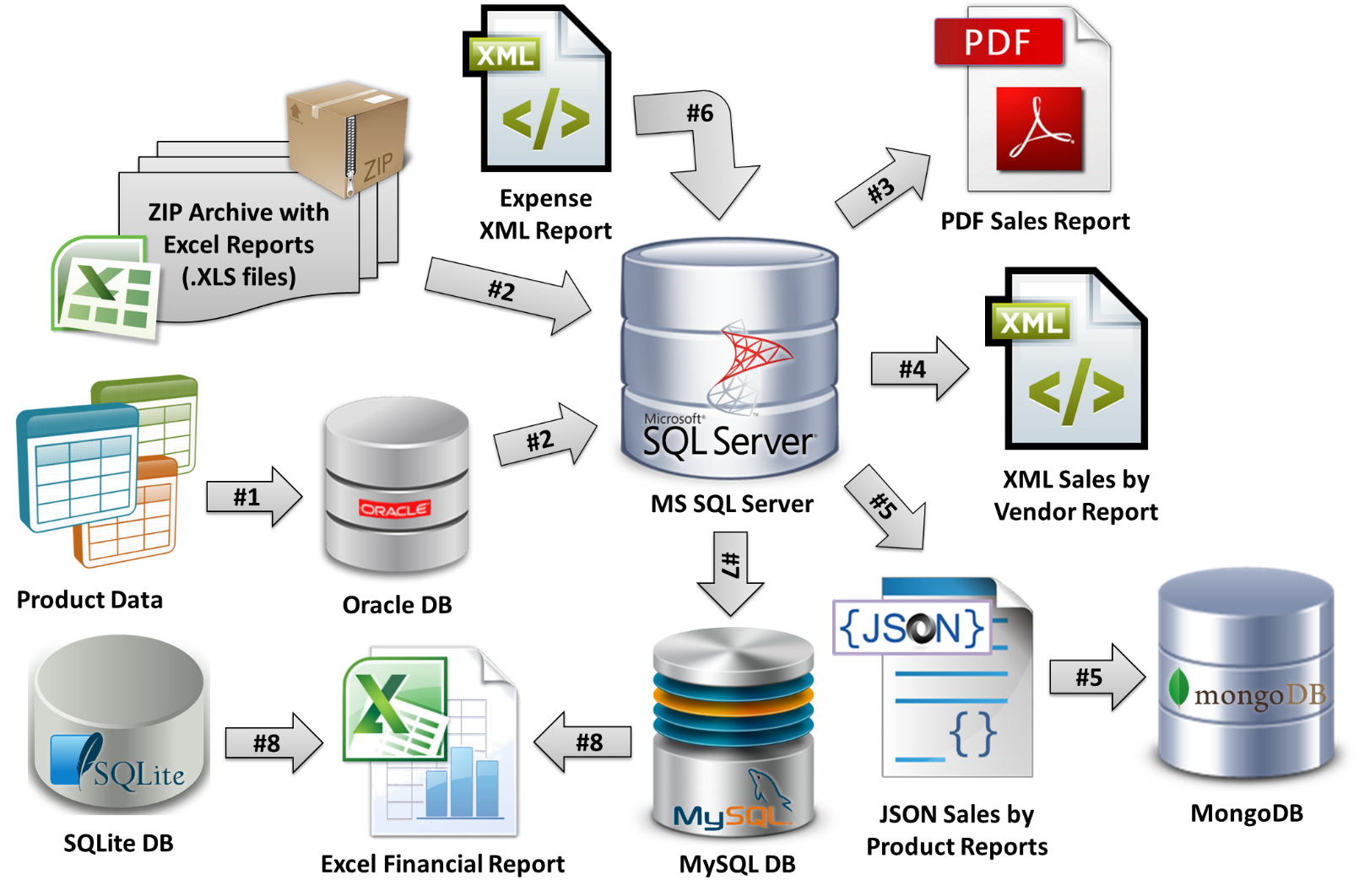
# Project Assignment for the [Database Applications Course @ SoftUni](https://softuni.bg/courses/database-applications/)

This project assignment is designed to develop skills for working with various **database systems** (like Oracle, SQL Server and MySQL), to access different **data** **sources** and process different **data formats** (like Excel, PDF, XML and JSON), to import data, process data and export data to different formats.

## Supermarkets Chain – Introduction

A chain of supermarkets has multiple decentralized information systems to hold its data (products, vendors, prices, sales, etc.) and works with different data sources. Data sometimes comes as Excel files or as XML documents or is accessed from a database. The management wants the IT department to produce daily reports in different formats so we need to **develop a set of tools** that do the data importing, processing, exporting and reports generation, following the steps from the figure below:



## Problem #1 – Design and Populate the Oracle Database

The chain of supermarkets holds information about its products in **Oracle database** consisting of at least three tables: **products**, **vendors** and **measures**.

For example, you may have the following tables with sample data (this is just an example):

### Products

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **VendorID** | **Product Name** | **MeasureID** | **Price** |
| 1 | 20 | Beer “Zagorka” | 100 | 0.86 |
| 2 | 30 | Vodka “Targovishte” | 100 | 7.56 |
| 3 | 20 | Beer “Beck’s” | 100 | 1.03 |
| 4 | 10 | Chocolate “Milka” | 200 | 2.80 |
| … | … | … | … | … |

### Vendors

|  |  |
| --- | --- |
| **ID** | **Vendor Name** |
| 10 | Nestle Sofia Corp. |
| 20 | Zagorka Corp. |
| 30 | Targovishte Bottling Company Ltd. |
| … | … |

### Measures

|  |  |
| --- | --- |
| **ID** | **Measure Name** |
| 100 | liters |
| 200 | pieces |
| 300 | kg |
| … | … |

You are free to use the above provided sample database schema and data or **design your own DB schema**. You may add more columns and new tables or redesign the existing tables. You may use sequential IDs for the primary keys (sequences in Oracle) or any other primary key notation (e.g. [UUID](http://oracle-base.com/articles/9i/uuid-9i.php) unique identifiers).

For testing purposes, please **fill between at least 20 records in each table**. Try to use real-world data.

You may use **any version of Oracle database** (including a cloud-based on-demand solution) and your favorite database management tools, libraries and frameworks. If you do not have experience, the following software and tools are recommended:

* [Oracle Database Express Edition 11g Release 2](http://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/index.html) – a free version of Oracle DB, running on Linux and Windows
* [Oracle SQL Developer 4.0](http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html) – a free Java-based database management tool, running on Linux and Windows

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Aggregated Sales Report** | | | | |
| Date: 20-Jul-2014 | | | | |
| **Product** | **Quantity** | **Unit Price** | **Location** | **Sum** |
| Beer “Beck’s” | 40 liters | 1.20 | Supermarket “Kaspichan – Center” | 48.00 |
| Beer “Zagorka” | 37 liters | 1.00 | Supermarket “Bourgas – Plaza” | 37.00 |
| Chocolate “Milka” | 7 pieces | 2.85 | Supermarket “Bay Ivan” – Zmeyovo | 19.95 |
| Vodka “Targovishte” | 14 liters | 8.50 | Supermarket “Bourgas – Plaza” | 119.00 |
| Chocolate “Milka” | 12 pieces | 2.90 | Supermarket “Kaspichan – Center” | 34.80 |
| Beer “Zagorka” | 65 liters | 0.92 | Supermarket “Kaspichan – Center” | 59.80 |
| Vodka “Targovishte” | 4 liters | 7.80 | Supermarket “Bay Ivan” – Zmeyovo | 31.20 |
| … | … | … | … | … |
| Total sum for 20-Jul-2014: | | | | **349.75** |
| Date: 21-Jul-2014 | | | | |
| **Product** | **Quantity** | **Unit Price** | **Location** | **Sum** |
| Beer “Zagorka” | 11 liters | 1.00 | Supermarket “Bourgas – Plaza” | 11.00 |
| Beer “Zagorka” | 78 liters | 0.92 | Supermarket “Kaspichan – Center” | 71.76 |
| Beer “Zagorka” | 146 liters | 0.88 | Supermarket “Plovdiv – Stolipinovo” | 128.48 |
| Vodka “Targovishte” | 20 liters | 8.50 | Supermarket “Bourgas – Plaza” | 170.00 |
| Vodka “Targovishte” | 67 liters | 7.70 | Supermarket “Plovdiv – Stolipinovo” | 515.90 |
| Vodka “Targovishte” | 3 liters | 7.80 | Supermarket “Bay Ivan” – Zmeyovo | 23.40 |
| Beer “Beck’s” | 43 liters | 1.20 | Supermarket “Kaspichan – Center” | 51.60 |
| Beer “Beck’s” | 75 liters | 1.05 | Supermarket “Plovdiv – Stolipinovo” | 78.75 |
| Chocolate “Milka” | 9 pieces | 2.90 | Supermarket “Kaspichan – Center” | 26.10 |
| Chocolate “Milka” | 5 pieces | 2.85 | Supermarket “Bay Ivan” – Zmeyovo | 14.25 |
| … | … | … | … | … |
| Total sum for 21-Jul-2014: | | | | **1091.24** |
| Date: 22-Jul-2014 | | | | |
| **Product** | **Quantity** | **Unit Price** | **Location** | **Sum** |
| Beer “Zagorka” | 16.00 | 1.00 | Supermarket “Bourgas – Plaza” | 16.00 |
| Beer “Zagorka” | 90.00 | 0.92 | Supermarket “Kaspichan – Center” | 82.80 |
| Beer “Zagorka” | 230.00 | 0.88 | Supermarket “Plovdiv – Stolipinovo” | 202.40 |
| Vodka “Targovishte” | 24.00 | 8.50 | Supermarket “Bourgas – Plaza” | 204.00 |
| Vodka “Targovishte” | 12.00 | 7.70 | Supermarket “Plovdiv – Stolipinovo” | 92.40 |
| Beer “Beck’s” | 18.00 | 1.20 | Supermarket “Kaspichan – Center” | 21.60 |
| Beer “Beck’s” | 60.00 | 1.05 | Supermarket “Plovdiv – Stolipinovo” | 63.00 |
| Chocolate “Milka” | 14.00 | 2.90 | Supermarket “Kaspichan – Center” | 40.60 |
| … | … | … | … | … |
| Total sum for 22-Jul-2014: | | | | **722.80** |
| Grand total: | | | | **2163.79** |

**Output:** create a **SQL script** that creates the database schema in the Oracle DB and fills the sample data (20-30 products, vendors, measures and any other data you have designed).

## Problem #2 – Replicate the Tables from Oracle DB and Load Excel Reports from ZIP File into SQL Server

Your task is to write a C# / Java / other application to **replicate the Oracle database tables** to SQL Server database and **load Excel reports into** **MS SQL Server** database. You should preliminary **design a database schema** in SQL Server to hold all data about the products (data from the Oracle database like products, vendors and measures + data from the Excel files like supermarkets and sales by date) or **use ORM framework and "code-first" approach** to populate the database schema in SQL Server at runtime.

Your app should first **replicate the products data** from Oracle to SQL Server. This should be done on demand, ignoring the products that are already in the SQL Server. Existing products are not changed, but any new products are added in the SQL Server DB on demand.

The **Excel files** are packed inside a **ZIP archive** holding subfolders named as the **dates of the report** in format **dd-MMM-yyyy** (see the example reports archive <Sample-Sales-Reports.zip>). Your app should read the ZIP files without unpacking them. Each Excel file holds data similar to the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Supermarket “Plovdiv – Center”** | | | |
| **Product** | **Quantity** | **Unit Price** | **Sum** |
| Beer “Beck’s” | 75 | 1.05 | 78.75 |
| Beer “Zagorka” | 146 | 0.88 | 128.48 |
| Vodka “Targovishte” | 67 | 7.7 | 515.90 |
| Total sum: | | | **723.13** |

Note that the ZIP file could contain few hundred dates (folders), each holding few hundreds Excel files, each holding thousands of data rows, so think about the performance. Use **transactions** to avoid inconsistent data in case of crashing during the load process.

**Input**: Oracle database holding products; ZIP file with Excel 2003 reports (see the example).

**Output**: C# / Java or other application that loads the Excel files into the SQL Server DB.

## Problem #3 – Generate PDF Sales Reports

Create a C# / Java / other program to **generate PDF reports** summarizing the sales information for given period from the SQL Server database.

For example, the **PDF report** may hold a table like the sample below:

A sample PDF report is also available: <Sample-Aggregated-Sales-Report.pdf>.

**Input**: SQL Server database; start date; end date.

**Output**: C# / Java / other program that creates a PDF report based on the sales data.

## Problem #4 – Generate XML Sales by Vendor Report

Create a C# / Java / other program to **generate report in XML format** holding the sales by vendor for given period, like the sample below:

|  |
| --- |
| **Sales-by-Vendors-Report.xml** |
| <?xml version="1.0" encoding="utf-8">  <sales>  <sale vendor="Nestle Sofia Corp.">  <summary date="20-Jul-2014" total-sum="54.75" />  <summary date="21-Jul-2014" total-sum="40.35" />  <summary date="22-Jul-2014" total-sum="40.60" />  </sale>  <sale vendor="Targovishte Bottling Company Ltd.">  <summary date="20-Jul-2014" total-sum="150.20" />  <summary date="21-Jul-2014" total-sum="709.30" />  <summary date="22-Jul-2014" total-sum="249.40" />  </sale>  <sale vendor="Zagorka Corp.">  <summary date="20-Jul-2014" total-sum="144.80" />  <summary date="21-Jul-2014" total-sum="341.59" />  <summary date="22-Jul-2014" total-sum="385.80" />  </sale>  <sales> |

Save the report in a local XML file.

**Input**: SQL Server database; start date; end date.

**Output**: C# / Java / other program that generates an XML file holding the sales by vendor report.

## Problem #5 – JSON Reports in MongoDB

Write a C# / Java / other program to create **reports** for the sales of each product for given period in **JSON format** and save all reports in **MongoDB** andas JSON files. All reports should look like the samples below. The reports should be saved in the **MongoDB** database in a collection called **SalesByProductReports** as well as in the file system in a folder called “**Json-Reports**”, in files named “**XX.json**” where **XX** is the product ID.

Sample product reports in JSON format:

|  |
| --- |
| **3.json** |
| {  "product-id" : 3,  "product-name" : "Beer “Beck’s”",  "vendor-name" : "Zagorka Corp.",  "total-quantity-sold" : 236,  "total-incomes" : 262.95,  } |
| **1.json** |
| {  "product-id" : 1,  "product-name" : "Beer “Zagorka”",  "vendor-name" : "Zagorka Corp.",  "total-quantity-sold" : 673,  "total-incomes" : 609.24,  } |
| **4.json** |
| {  "product-id" : 4,  "product-name" : "Chocolate “Milka”",  "vendor-name" : "Nestle Sofia Corp.",  "total-quantity-sold" : 47,  "total-incomes" : 135.70,  } |
| **2.json** |
| {  "product-id" : 2,  "product-name" : "Vodka “Targovishte”",  "vendor-name" : "Targovishte Bottling Company Ltd.",  "total-quantity-sold" : 144,  "total-incomes" : 1155.90,  } |

**Input**: SQL Server database; start date; end date.

**Output**: a set of JSON files in the file system; data loaded in MongoDB database.

## Problem #6 – Load Expense Data from XML

Create an **XML** file holding **expenses by vendor and month**. For example, see the file <Sample-Vendor-Expenses.xml>:

|  |
| --- |
| **Vendor-Expenses.xml** |
| <?xml version="1.0" encoding="utf-8">  <expenses-by-month>  <vendor name="Nestle Sofia Corp.">  <expenses month="Jul-2013">30.00</expenses>  <expenses month="Aug-2013">40.00</expenses>  </vendor>  <vendor name="Targovishte Bottling Company Ltd.">  <expenses month="Jul-2013">200.00</expenses>  <expenses month="Aug-2013">180.00</expenses>  </vendor>  <vendor name="Zagorka Corp.">  <expenses month="Jul-2013">120.00</expenses>  <expenses month="Aug-2013">180.00</expenses>  </vendor>  <expenses-by-month> |

**Create new tables and relationships** to keep the expenses by period in your SQL Server database.

Write a C# / Java / other program to **read the** XML file, parse it and **save the data** in the **SQL Server** database.

**Input**: XML file holding expenses by vendors by months.

**Output**: program for loading an XML expense report in SQL Server; data loaded in the SQL Server database.

## Problem #7 – Load Data to MySQL

Create a C# / Java / other program to transfer the expense and incomes data from SQL Server to MySQL. This should transfer the data holding the **vendors**, their **products**, their **incomes by product** and their **expenses**. The data transfer logic should keep all existing data in the MySQL database and insert only the non-existing table rows from the SQL Server database. You are free to design the MySQL DB schema in a way you think will work well.

**Input**: SQL Server database holding the vendors, products, incomes and expenses.

**Output**: program for loading vendors, products, incomes and expenses in MySQL database.

## Additional Requirements

* Your main program logic should be written in C#, Java or other language.
  + It should provide a way to execute each of the problems from the project assignment.
  + It can be console-based or might provide GUI of your choice.
* Use non-commercial libraries to read ZIP files, read / write Excel documents, connect to various databases, generate PDF documents, read / write XML and JSON documents.
* You are free to use OLEDB, ODBC, ADO.NET or other low-level database access libraries / protocols or modern ORM framework with "code first" or "database first" approach.
* MongoDB should be accessed through its official MongoDB C# / Java / other driver. The database instance may reside locally or in a cloud environment.
* The SQLite embedded database should be accesses though its Entity Framework provider.