Минобрнауки России
федеральное государственное бюджетное образовательное
учреждение высшего образования
«Омский государственный университет им. Ф.М. Достоевского»
Кафедра компьютерных технологий и сетей

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Лавров Д. Н.
«___» _____ 2017 г.

РАЗРАБОТКА ПРОГРАММЫ ДЛЯ ПРЕОБРАЗОВАНИЯ ИЗ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ В МНОГОМЕРНУЮ ДЛЯ ИНТЕРНЕТ-МАГАЗИНА

Выпускная квалификационная работа
по направлению 09.03.01 Информатика и вычислительная техника

Научный руководитель:
ст. преподаватель
_____ Опарина Т.М.
«___» _____ 2017 г.
Выполнил:
студент группы СИБ301У
_____ Кузина С.А.
«___» _____ 2017 г.

Омск
2017

Содержание

Введение 3
Глава 1. Описание проекта 5
1.1.Анализ средств создания интернет-магазинов 5
1.2.Типичная аналитика интернет-магазина 7
1.3.Особенности базы данных интернет-магазина 9
1.4.Проектирование многомерной структуры интернет-магазина 12
1.5 Выбор средств разработки 16
Глава 2. Реализация проекта 17
2.1.Основные требования к оборудованию и программному обеспечению 17
2.2.Описание организации данных 18
2.3.Алгоритм работы программы 19
2.4.Анализ скорости выполнения запросов в реляционной базе данных и в многомерной системе 32
Заключение 34
Список литературы 35

Введение

В современном мире существует огромное множество различных предприятий, корпораций, организаций различного масштаба. В процессе своей деятельности финансовые учреждения, бизнес-корпорации, промышленные предприятия, органы государственной власти накапливают значительные объемы данных. Они хранят в себе большие потенциальные возможности по извлечению аналитической информации, на основе которой можно выявлять скрытые тенденции, строить стратегию развития, находить новые пути развития. Но для того чтобы существующие объемы данных действительно способствовали принятию управленческих решений, информация должна быть представлена пользователю-аналитику в определенной форме. Иными словами, необходимы развитые инструменты обработки данных. Главным критерием выбора инструмента является скорость обработки больших объёмов информации. Такую возможность предоставляют системы OLAP, работающие с многомерными базами данных.

В данной работе будет рассмотрена возможность интеграции базы данных интернет-магазина в многомерную структуру. Количество интернет-магазинов неуклонно растёт и для любого из них актуальной задачей является привлечение как можно большего числа покупателей.

Чтобы повысить свою привлекательность, интернет-магазинам необходимо постоянно изучать разного рода аналитику: анализ продаж, целевой аудитории, конкурентоспособности сайта [1]. Анализ – это самый качественный инструмент правильной работы интернет-магазина. А для удобства анализа имеет смысл использовать многомерные базы данных.

Однако, как показывает практика, зачастую интернет-магазины построены на реляционных базах данных (MySQL, PostgreSQL, SQLite). В реляционных базах данных удобно хранить большие объёмы информации, но

извлекать информацию для анализа проблематично, гораздо эффективнее с этим справляются многомерные базы данных.

Таким образом, цель работы – разработать приложение для интеграции реляционной базы данных интернет-магазина в многомерную систему.

Для достижения цели потребуется решить следующие задачи:

1. Проанализировать средства создания интернет-магазинов.
2. Проанализировать существующие способы аналитики продаж, определить необходимые данные для построения OLAP-куба.
3. Сформулировать требования к приложению.
4. Выбрать средства разработки.
5. Разработать приложение.
6. Проанализировать скорость выполнения запросов в реляционной и многомерной базах данных.

Глава 1. Описание проекта

1.1. Анализ средств создания интернет-магазинов

Следующая таблица демонстрирует основные существующие способы создания интернет-магазинов.

Таблица 1. Способы создания Интернет-магазина

Способ	База данных	Достоинства	Недостатки
Конструкторы сайтов (SaaS)	MySQL	- простота в использовании - скорость создания магазина	- шаблонный дизайн - возможности строго ограничены
CMS (Content management system), ориентированные на создание интернет-магазинов	MySQL	- индивидуальный дизайн - возможности не ограничены или почти не ограничены, в зависимости от выбора CMS	- требуется больше времени для создания магазина, по сравнению с SaaS
Универсальные CMS	MySQL/ SQLite/ PostgreSQL/ MS SQL	- индивидуальный дизайн - возможности не ограничены	- требуется больше времени для создания магазина, по сравнению с предыдущими вариантами

Исходя из проведённого анализа, можно сделать следующие выводы:

- для небольших интернет-магазинов (например, ориентированных на один бренд) лучше всего подходят SaaS-конструкторы;

- все интернет-магазины строятся на реляционных БД, чаще всего для интернет-магазина используют СУБД MySQL.

1.2. Типичная аналитика интернет-магазина

Основные виды анализа, а также желательная периодичность их осуществления представлены в таблице 2.

Таблица 2. Основные виды анализа продаж

Виды анализа	Цели анализа	Периодичность				
		Д	Н	М	К	Г
Динамика товарооборота, прибыли, удельных показателей продаж	Отслеживание тенденций продаж. Оперативная корректировка ассортимента, цен. Построение прогнозов продаж.	+	+	+	+	+
Анализ товарных запасов	Обнаружение пробелов в ассортименте. Оценка избыточности или недостатка товарных запасов.		+	+		
Сравнение продаж отчетного периода с предыдущими	Отслеживание динамики продаж по различным направлениям (точки продаж, товары, товарные группы, бренды и т. д.).			+	+	+
ABC-, XYZ-анализ	Определение приоритетных направлений развития. Перераспределение ресурсов [2].			+	+	+
Анализ по Бостонской матрице	Оценка вклада групп товаров в продажи магазина. Корректировка ассортимента и ценовой политики.				+	+

Где Д — ежедневно, Н — еженедельно, М — ежемесячно, К — ежеквартально, Г — ежегодно [3].

Таким образом, для проведения анализа необходимо получить названия товаров, их категории для создания иерархии, данные по полученной прибыли (суммы продаж) по каждому товару и категории, города, покупающие эти товары. Таким образом, измерениями куба будут товар, город и время (т.е. отчётный период). Суммы продаж – это меры куба. По этим данным можно анализировать динамику продаж относительно точек продаж (городов), товаров и товарных групп. Названия товаров можно получить из таблицы «Products», город доставки – из таблицы «Shipping», сумму продаж определённого товара можно получить так же из таблицы «Orders».

Для анализа товарных запасов достаточно взять в качестве меры количество проданного товара по каждому товару из таблицы «Orders».

1.3. Особенности базы данных интернет-магазина

Основные таблицы базы данных для интернет-магазина представлены на рисунке 1.

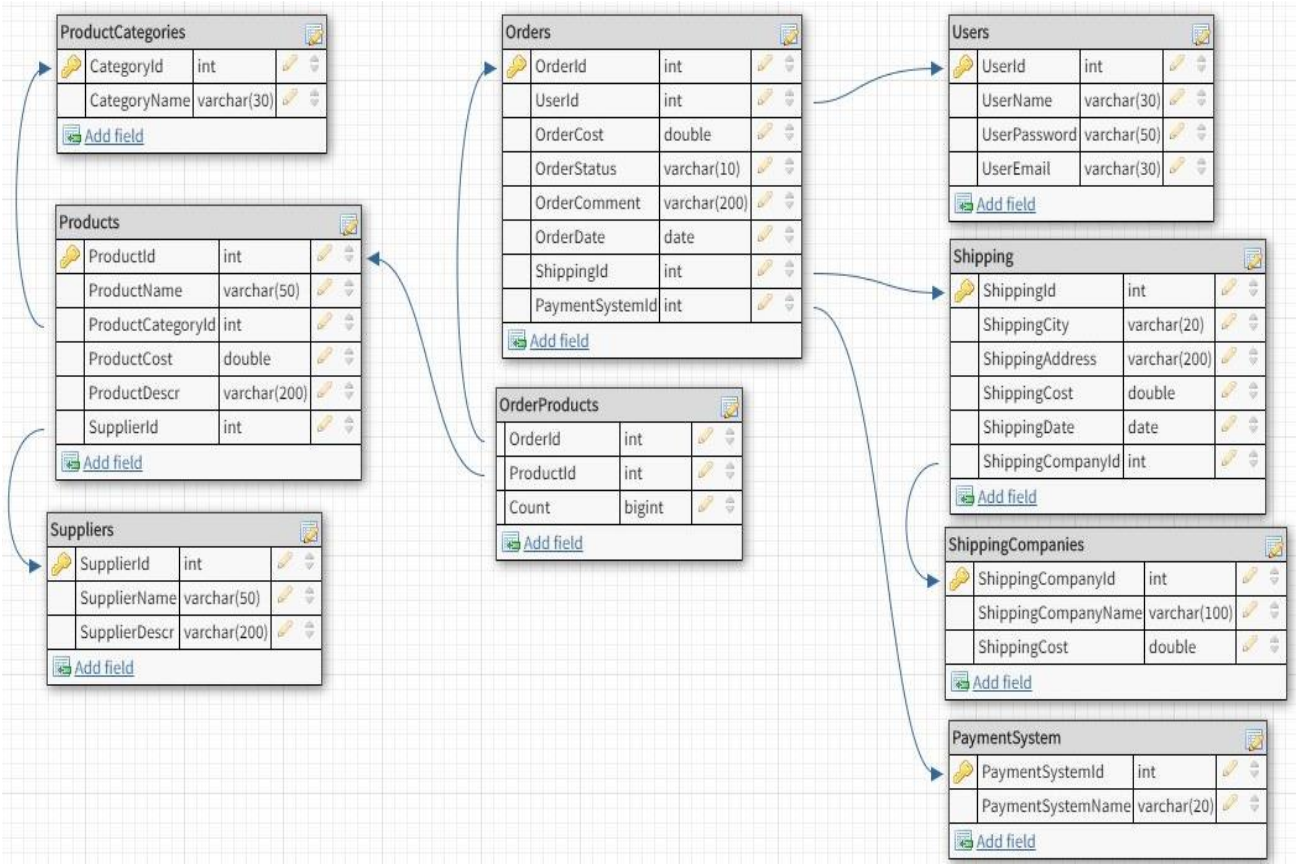

Рис. 1. Схема реляционной БД интернет-магазина

Поля таблицы «Товар»:

1. Наименование товара.
2. Описание.
3. Цена.
4. Количество.

Поля таблицы «Производитель»:

1. Наименование.
2. Описание.

Поля таблицы «Категория»:

Главным назначение данной сущности базы данных Интернет-магазина является группировка товаров. У категорий существует иерархия (категория-подкатегория и т.д.), что необходимо учитывать в виде атрибутов:

1. Название.
2. Описание.
3. Иерархия – номер ступени категории в иерархии.

Поля таблицы «Покупатель»:

1. Имя пользователя.
2. Пароль.
3. Контакты – телефон пользователя, e-mail, факс, почтовый индекс, адрес и др. – каждый атрибут хранится в отдельном поле.

Поля таблицы «Заказ»:

1. Покупатель – идентификатор покупателя из одноимённой таблицы, который сделал заказ.
2. Товар – идентификатор товара из соответствующей таблицы. Поскольку в одном заказе может быть несколько товаров, есть смысл сделать отдельную таблицу для фиксации этой связи, хранящей идентификатор заказа и товара.
3. Количество товаров.
4. Цена товара.
5. Сумма заказа – суммарная цена всех товаров, значащихся в заказе.
6. Способ оплаты – здесь в большинстве случаев хранится название платёжной системы или её идентификатор из одноимённой таблицы.
7. Статус – состояние заказа: «В обработке», «Выполнен», «Отклонён» и др.

8. Комментарий – поле, которое заполняет пользователь при оформлении. Не является обязательным.

Поля таблицы «Доставка»:

1. Заказ – идентификатор заказа.
2. Город доставки.
3. Способ доставки – название транспортной компании, либо её идентификатор.
4. Стоимость доставки.
5. Адрес места назначения.

1.4. Проектирование многомерной структуры интернет-магазина

Многомерная структура интернет-магазина будет состоять из измерений:

- {Поставщик | Группа товаров | Товар}

- {Страна | Регион | Город}

- {Год | Квартал | Месяц | Неделя | День | Сумма продаж}

Соответственно, будут присутствовать таблицы измерений Suppliers («Поставщик»), Countries («Страна»), Times («Время»). А так же таблица фактов Sales («Продажи»).

Таблица измерений Suppliers:

- SuppliersKey – ключ таблицы.

- SupplierID – код поставщика.

- SupplierName – имя поставщика.

- ProductID – код продукта.

- ProductName – название продукта.

- CategoryID – код группы товаров.

- CategoryName – название группы товаров.

Имеется иерхархия «ProductCategories»

- CategoryName – название группы.

- ProductName – название товара.

Таблица измерений Times:

- TimeKey – ключ таблицы.

- YearID – код года.

- Year – год.

- MonthID – код месяца.

- Month – месяц.

- DayID – код дня.

- Day – день.

Имеется иерархия «Times»:

- Year.

- Month.

- Day.

Таблица измерений Countries:

- CountriesKey – ключ таблицы.

- Country – страна.

- City – город.

Иерархия:

- Country – страна.

- City – город.

Таблица фактов Sales:

- SuppliersKey – ключ из таблицы «Suppliers».

- CountriesKey – ключ из таблицы «Countries».

- TimeKey – ключ из таблицы «Times».

- OrderCost – сумма продаж (мера).

Осями OLAP-куба служат основные атрибуты анализируемого бизнес-процесса. Для продаж это могут быть товар, регион, тип покупателя. В качестве одного из измерений используется время.

На пересечениях осей измерений (Dimensions) находятся данные, количественно характеризующие процесс – меры (Measures). Это могут быть объемы продаж в количестве или в денежном выражении, остатки на складе, издержки и т. п. [4].

На рисунке 2 представлен OLAP-куб для интернет-магазина с измерениями «Месяц», «Группа товаров», «Город» и мерой «Сумма продаж».

Рис. 2. OLAP-куб Интернет-магазина

Алгоритм работы программы в виде блок-схемы

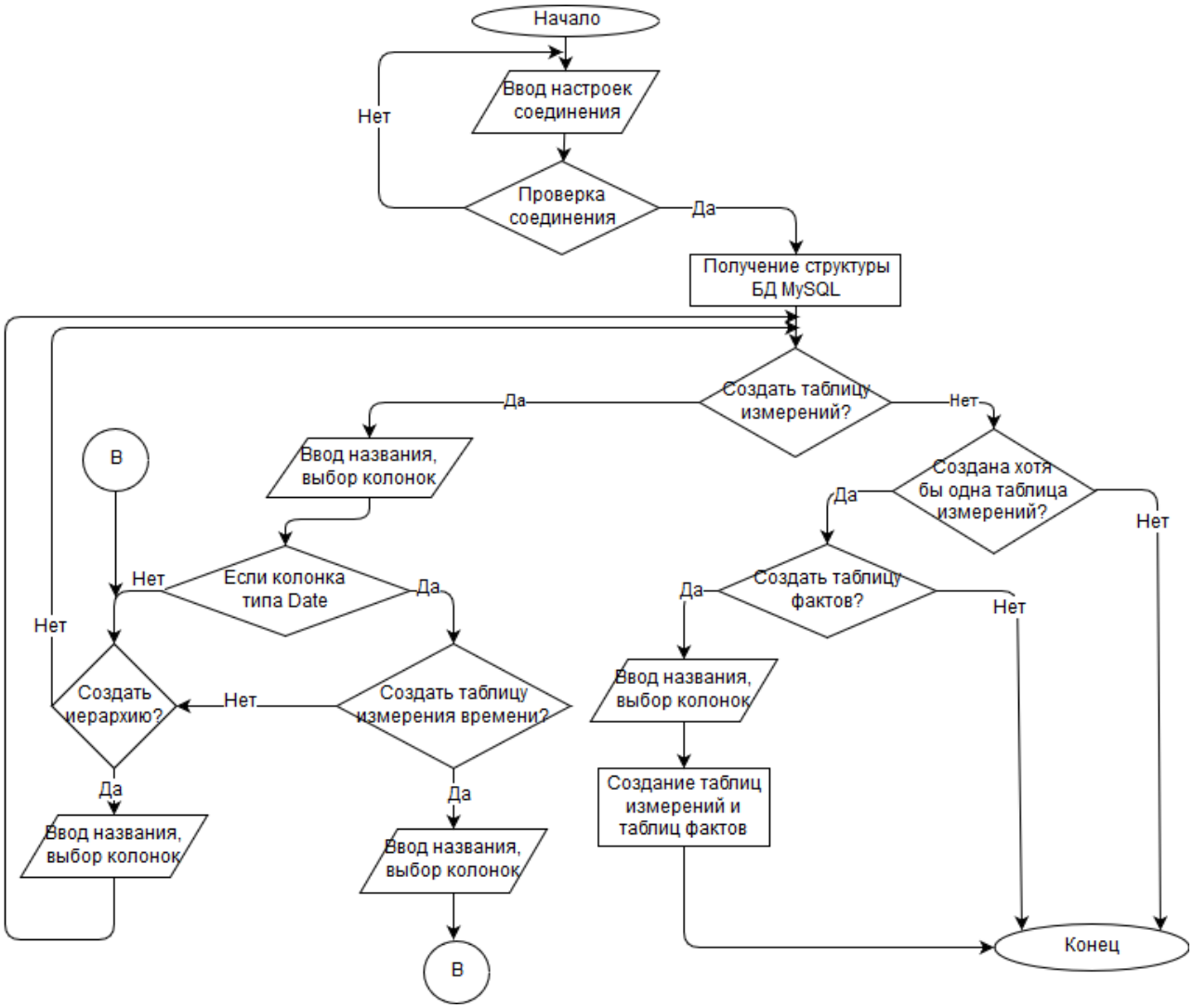

Рис. 3. Блок-схема алгоритма

1.5 Выбор средств разработки

Для разработки данного приложения был выбран язык Java 1.8. Это современный объектно-ориентированный язык, имеющий строгую типизацию данных, что уменьшает вероятность ошибок. Кроме того, приложение, написанное на Java, будет кроссплатформенным.

Для создания интерфейса приложения был использован Java Scene Builder, а сам интерфейс построен с использованием библиотеки JavaFX.

Для сборки проекта был выбран фреймворк Maven.

В качестве реляционной базы данных выбрана СУБД MySQL.

Для разработки OLAP-системы выбрана СУБД Oracle Database 12g.

В качестве аналитического инструмента для работы с OLAP выбран Analytic Workspace Manager.

Глава 2. Реализация проекта

2.1. Основные требования к оборудованию и программному обеспечению

Минимальные системные требования:

- процессор – Intel Core 2 Dual 2MHZ;
- оперативная память - 1GB RAM;
- жесткий диск - 5 GB свободной памяти жесткого диска;
- видео – SVGA 800 x 600;
- операционная система: Windows XP и выше, либо ОС Linux.

Языки и среды:

- Java 1.8 JRE;
- MySQL;
- Oracle Database 12g;
- Analytic Workspace Manager.

2.2. Описание организации данных

Изначально данные интернет-магазина хранятся в СУБД MySQL. Для построения OLAP-системы на основе этих данных необходимо использовать Oracle Database.

С помощью Database Configuration Assistant будет создана база данных TEST.

От имени SYS в SQL Developer необходимо создать пользователя и дать ему привилегии [5]:

```
CREATE USER test IDENTIFIED BY 123

GRANT all privileges TO test
```

Созданного пользователя в дальнейшем нужно будет использовать для доступа к Oracle Database.

2.3. Алгоритм работы программы

2.3.1 Установка соединения с базами данных

Устанавливаем соединение с базой данных MySQL:

```
try {
    Class.forName("com.mysql.jdbc.Driver");
} catch (ClassNotFoundException e) {
    throw new IllegalStateException("Cannot find the driver in the classpath!",
e);
}

// get properties
String url = "jdbc:mysql://" + servername.getText() + ":" + port.getText() + "/"
+ dbname.getText();
String username = username.getText();
String password = password.getText();
Connection connection = DriverManager.getConnection(url, username, password);

System.out.println("Connecting database...");
```

Устанавливаем соединение с базой данных Oracle Database:

```
OracleConnection conn = null;

// read properties
OracleDataSource ods = new OracleDataSource();
ods.setURL("jdbc:oracle:thin:@" + orservname.getText() + ":" + orport.getText()
+ ":" + ordbname.getText());
ods.setUser(props.getProperty(orusername.getText()));
ods.setPassword(props.getProperty(orpassword.getText()));

conn = (OracleConnection) ods.getConnection();
```

Необходимо указать настройки для подключения к СУБД MySQL и Oracle Database. Сделать это можно, нажав на пункт меню «Настройки» в главном окне программы.

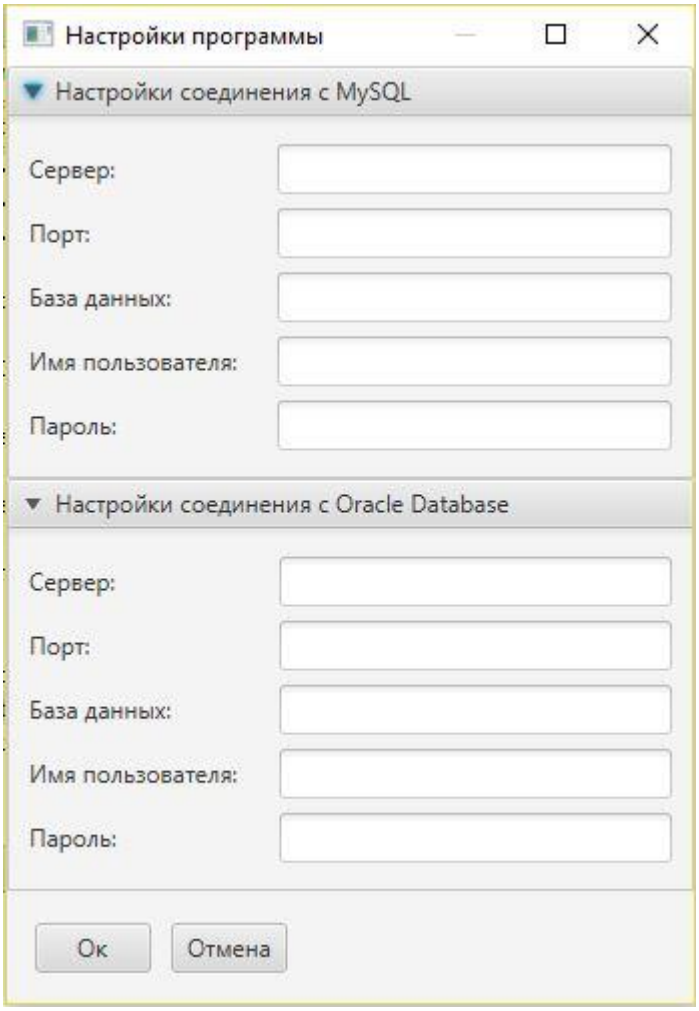

Рис. 4. Окно настроек программы

Настройки для MySQL:

- Имя сервера (по умолчанию localhost).
- Порт (по умолчанию 3306).
- Имя базы данных.
- Имя пользователя.
- Пароль.

Настройки для Oracle Database:

- Имя сервера (по умолчанию localhost).
- Порт (по умолчанию 1521).
- SID.
- Имя пользователя (пользователь test).

- Пароль (пароль 123).

2.3.2. Получение структуры MySQL

Далее получаем структуру базы данных из MySQL. Сложность в том, что не все типы данных взаимозаменяемы, а так же многие типы данных позволяют ограничить количество символов в поле (длину). Значит, длину поля тоже нужно учитывать.

```
// сначала получаем имена таблиц
public static ObservableList<String> getTableNames() {
    // get all table names
    Connection connection = connect();
    DatabaseMetaData md = null;
    ObservableList<String> tables = FXCollections.observableArrayList();
    try {
        md = connection.getMetaData();
        ResultSet rs = md.getTables(null, null, "%", null);
        while (rs.next()) {
            tables.add(rs.getString(3));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return tables;
}

// затем получаем имена полей
public static ArrayList<TreeItem> getFieldNames(String table) {
    Connection connection = connect();
    String query = "SELECT * FROM " + table;
    ArrayList<TreeItem> iFields = new ArrayList<TreeItem>();

    try {
        PreparedStatement ps = connection.prepareStatement(query);

        ResultSet resultSet = ps.executeQuery();
        ResultSetMetaData metaData = resultSet.getMetaData();
        int count = metaData.getColumnCount(); //count of columns
        for (int i = 1; i <= count; i++) {
            try {
                col = metaData.getColumnLabel(i);
                TreeItem iTable = new TreeItem(col);
                iFields.add(iTable);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return iFields;
}

// функция для получения типа и длины поля
```

```
public static Map<Integer, Integer> getFieldType(String fieldName, String table)
{
    Connection conn = connect();
    String query = "SELECT " + fieldName + " FROM " + table;
    Map<Integer, Integer> fieldsAndTypes = new HashMap<>();
    try {
        PreparedStatement ps = conn.prepareStatement(query);
        ResultSet resultSet = ps.executeQuery();
        ResultSetMetaData metaData = resultSet.getMetaData();
        int count = metaData.getColumnCount(); //count of columns
        String columnName[] = new String[count];
        for (int i = 1; i <= count; i++) {
            String col = metaData.getColumnLabel(i);
            columnName[i-1] = col;
            if (metaData.getColumnTypeName(i).equals("INT")){
                fieldsAndTypes.put(1, 0));
            } else if (metaData.getColumnTypeName(i).equals("VARCHAR")) {
                fieldsAndTypes.put(2, metaData.getColumnDisplaySize(i));
            } else if (metaData.getColumnTypeName(i).equals("DATE")) {
                fieldsAndTypes.put(3, 0);
            } else if (metaData.getColumnTypeName(i).equals("DOUBLE")) {
                fieldsAndTypes.put(4, 0);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return fieldsAndTypes;
}
```

2.3.3. Создание таблиц измерений и таблиц фактов

Далее для создания таблиц измерений пользователю нужно ввести имя измерения и выбрать поля будущей таблицы. Интерфейс позволяет выбрать сразу все необходимые поля для измерения, для этого нужно зажать клавишу Ctrl.

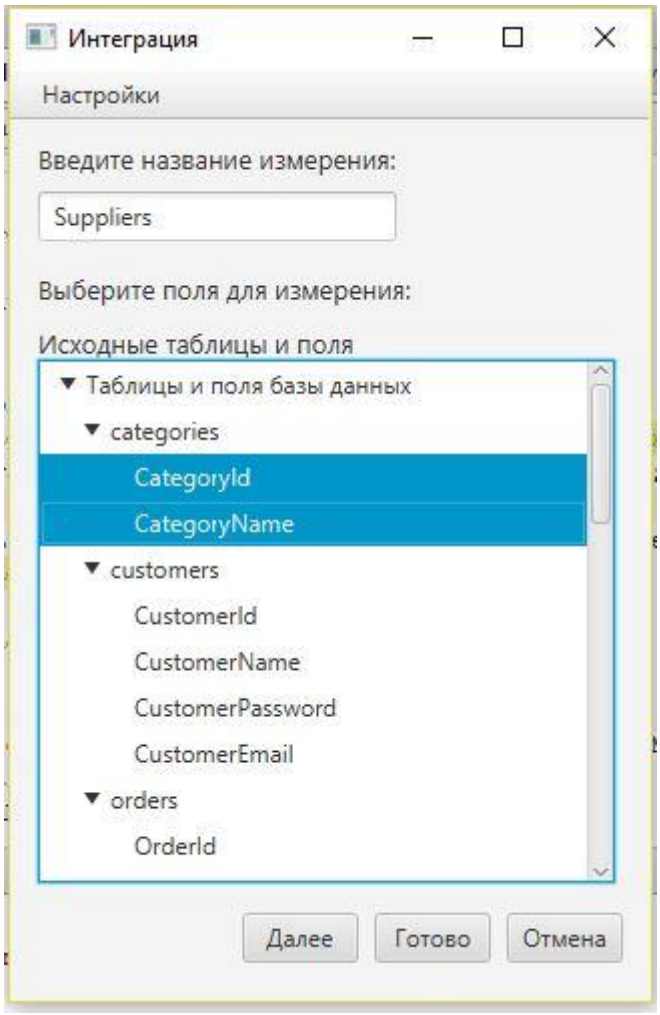

Рис. 5. Создание таблицы измерения

Если после этого нажать «Готово», то создастся одна таблица Измерений с именем «Suppliers» и выбранными колонками. Типы колонок и их длина взяты из MySQL. Для каждой таблицы измерений будет создан первичный ключ вида <<TableName>KEY>.

```
// функция для создания таблиц в Oracle Database
public static void createTable(Map<String, Map<Integer, Integer>>
fieldsAndTypes, String tableName) {
    Connection conn = connectOracle();
    String fieldsTypes = "";
```

```
    int size = fieldsAndTypes.size();
    int i = 0;
    for (Map.Entry entry : fieldsAndTypes.entrySet()) {
        i++;
        String field = (String) entry.getKey();
        Map<Integer, Integer> typeLength = (Map<Integer, Integer>)
entry.getValue();

        //действия с ключом
        Set keys = typeLength.keySet();
        Integer type = 0;
        for (Object key: keys) {
            type = (Integer) key;
        }

        // Для типа Varchar важно указать длину
        String fieldType = types.get(type);
        String fieldLength = "";
        if (type == 2) {
            int length = typeLength.get(type);
            fieldLength = "(" + length + ")";
        }
        if (i == size) {
            fieldsTypes += field + " " + fieldType + fieldLength;
        }
        else {
            fieldsTypes += field + " " + fieldType + fieldLength + ", ";
        }
    }

    if (!fieldsTypes.equals("")) {
        String tableCreateQuery = String tableCreateQuery = "create table " +
tableName + "(" + fieldsTypes + " PRIMARY KEY(" + tableName + "Key))";
        Statement statement = null;
        try {
            statement = conn.createStatement();
            statement.execute(tableCreateQuery);

            System.out.println("Table Created Successfully");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Если нажать кнопку «Далее», то откроется интерфейс для создания иерархии измерения.

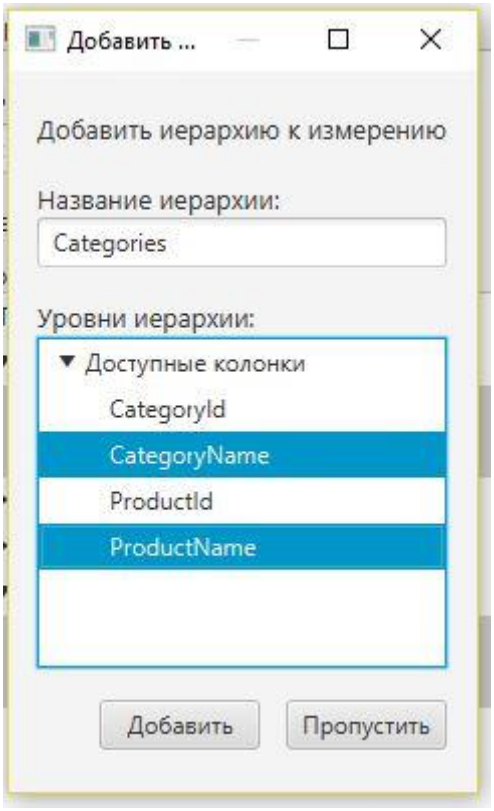

Рис. 6. Создание иерархии

Создание иерархии можно пропустить. После нажатия кнопки «Добавить» создастся иерархия с выбранными колонками, а так же будут созданы уровни измерения.

Далее пользователю будет предложено создать ещё таблицу измерений.

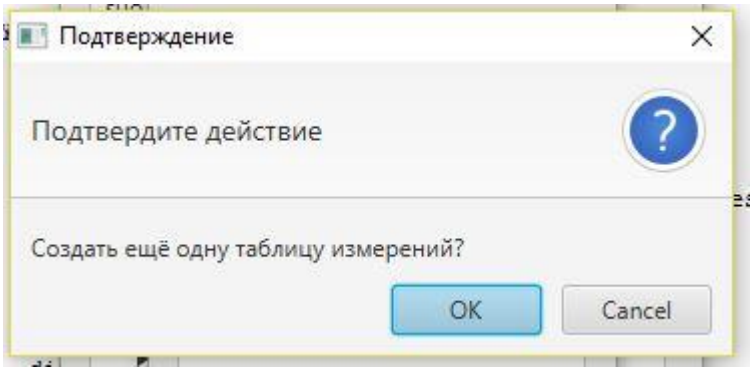

Рис. 7. Выбор создания следующей таблицы

Если пользователь откажется, то программа предложит создать таблицу фактов.

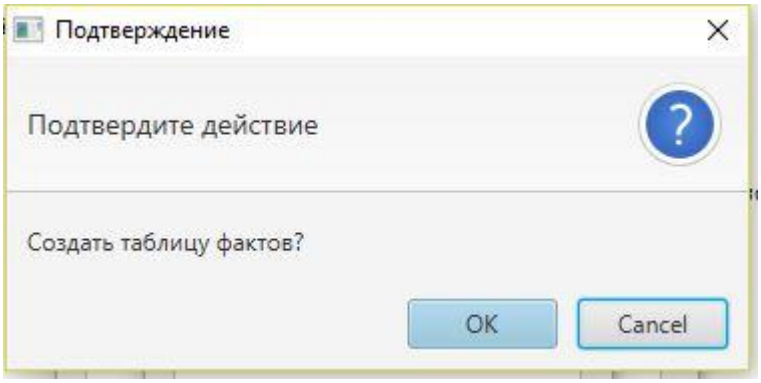

Рис. 8. Выбор создания таблицы фактов

Таблица фактов, как правило, содержит уникальный составной ключ, объединяющий первичные ключи таблиц измерений. Поэтому в последнем шаге нужно выбрать только меры, которые будут в таблице фактов [6].

Мера (факты) — это численное значение (числовой показатель), выражающее определенный аспект деятельности организации и используемое в базовой и возвратной информации [7].

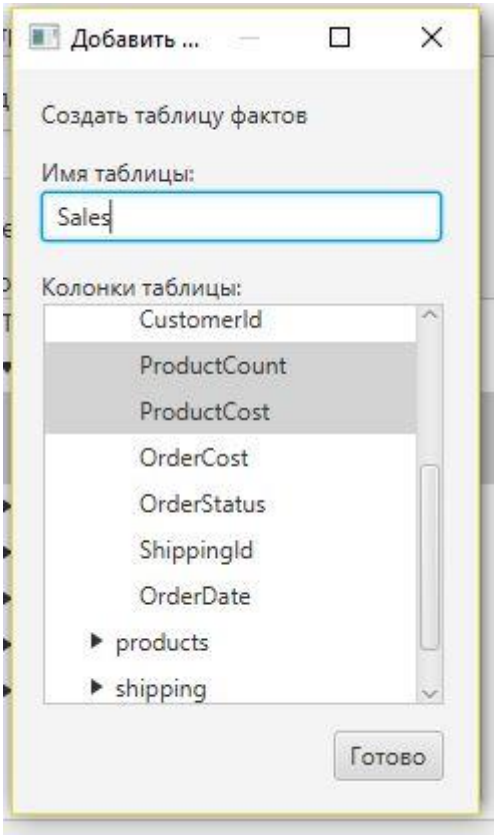

Рис. 9. Создание таблицы фактов. Выбор мер

Когда созданы таблицы измерений и таблицы фактов, приложение создаст аналитическое пространство с измерениями, уровнями измерений, иерархиями и кубом, построенным на основе этих измерений.

Первыми создаются измерения и уровни измерений.

```
public static void createAndMapDimensionLevels(MdmPrimaryDimension mdmChanDim,
MdmMetadataProvider metadataProvider, ArrayList<String> dimLevels, String
schemaName, Map<String, String> tableAndKey, Map<String, String> tableAndDsc) {
    // get dimension level names
    for (String dimLevel : dimLevels) {
        dimLevelNames.add(dimLevel.toUpperCase());
    }

    // get dimension keys
    for (Map.Entry entry : tableAndKey.entrySet()) {
        String key = schemaName + "." + entry.getKey() + "." + entry.getValue();
        keyColumns.add(key.toUpperCase());
    }

    // get dimension descriptions
    for (Map.Entry entry : tableAndDsc.entrySet()) {
        String key = schemaName + "." + entry.getKey() + "." + entry.getValue();
        lDescColNames.add(key.toUpperCase());
    }

    // Create the MdmDimensionLevel and MemberListMap objects.
    int i = 0;
    for (String dimLevelName : dimLevelNames) {
        MdmDimensionLevel mdmDimLevel =
mdmChanDim.findOrCreateDimensionLevel(dimLevelNames.get(i));
        dimLevelList.add(mdmDimLevel);

        // Create a MemberListMap for the dimension level.
        MemberListMap mdmDimLevelMemListMap =
mdmDimLevel.findOrCreateMemberListMap();
        ColumnExpression keyColExp = (ColumnExpression)
SyntaxObject.fromSyntax(keyColumns.get(i), metadataProvider);
        mdmDimLevelMemListMap.setKeyExpression(keyColExp);
        mdmDimLevelMemListMap.setQuery(keyColExp.getQuery());

        // Create an attribute map for the Long Description attribute.
        createLongDesciptionAttribute(mdmChanDim);
        AttributeMap attrMapLong =
mdmDimLevelMemListMap.findOrCreateAttributeMap(chanLongDescAttr);

        // Create an expression for the attribute map.
        Expression lDescColExp = (Expression)
SyntaxObject.fromSyntax(lDescColNames.get(i), metadataProvider);
        attrMapLong.setExpression(lDescColExp);
        i++;
    }
}
```

Затем создаются иерархии.

```
public static void createAndMapHierarchies(MdmPrimaryDimension mdmChanDim,
```

```
MdmMetadataProvider metadataProvider, String hierarchyName) {
    MdmLevelHierarchy mdmLevelHier =
mdmChanDim.findOrCreateLevelHierarchy(hierarchyName.toUpperCase());

    // Create the MdmHierarchyLevel and HierarchyLevelMap objects.
    int i = 0;
    for(String hierLevelName : dimLevelNames) {
        MdmHierarchyLevel mdmHierLevel =
mdmLevelHier.findOrCreateHierarchyLevel(mdmLevelHier.getPrimaryDimension()
                .findOrCreateDimensionLevel(hierLevelName));
        HierarchyLevelMap hierLevelMap =
mdmHierLevel.findOrCreateHierarchyLevelMap();
        ColumnExpression keyColExp = (ColumnExpression)
SyntaxObject.fromSyntax(keyColumns.get(i), metadataProvider);
        hierLevelMap.setKeyExpression(keyColExp);
        hierLevelMap.setQuery(keyColExp.getQuery());

        //Set the MdmDimensionLevel for the MdmHierarchyLevel.
        mdmHierLevel.setDimensionLevel(dimLevelList.get(i));
        i++;
    }
}
```

После иерархий создаётся куб и меры.

```
private MdmCube createAndMapCube(ArrayList<MdmPrimaryDimension> mdmDimentions,
MdmMetadataProvider metadataProvider, DataProvider dp, MdmDatabaseSchema
mdmDBSchema, AW aw, String cubeName, ArrayList<String> measureNames, String
tFactName) {
    MdmCube mdmCube = mdmDBSchema.findOrCreateCube(cubeName.toUpperCase());
    // Add dimensions to the cube.
    for (MdmPrimaryDimension dim : mdmDimentions) {
        mdmCube.addDimension(dim);
    }

    AWCubeOrganization awCubeOrg = mdmCube.createAWOrganization(aw, true);
    awCubeOrg.setMVOption(AWCubeOrganization.NONE_MV_OPTION);
    awCubeOrg.setMeasureStorage(AWCubeOrganization.SHARED_MEASURE_STORAGE);
    awCubeOrg.setCubeStorageType("NUMBER");

    AggregationCommand aggCommand = new AggregationCommand("AVG");
    ArrayList<ConsistentSolveCommand> solveCommands = new ArrayList();
    solveCommands.add(aggCommand);
    ConsistentSolveSpecification conSolveSpec =
            new ConsistentSolveSpecification(solveCommands);
    mdmCube.setConsistentSolveSpecification(conSolveSpec);

    // Create and map the measures of the cube.
    createAndMapMeasures(mdmCube, metadataProvider, mdmDBSchema, measureNames,
tFactName);
    // Commit the Transaction.
    commit(mdmCube, dp);
    return mdmCube;
}

private void createAndMapMeasures(MdmCube mdmCube, MdmMetadataProvider
metadataProvider, MdmDatabaseSchema mdmDBSchema, ArrayList<String> measureNames,
String tableName) {
    ArrayList<MdmBaseMeasure> measures = new ArrayList();

    for (String measureName : measureNames) {
```

```
        MdmBaseMeasure mdmNewMeasure =
mdmCube.findOrCreateBaseMeasure(measureName.toUpperCase());
        SQLDataType sdt = new SQLDataType("NUMBER");
        mdmNewMeasure.setSQLDataType(sdt);
        measures.add(mdmNewMeasure);
    }

    String schemaName = mdmDBSchema.getName();

    MdmTable mdmTable = (MdmTable)
mdmDBSchema.getTopLevelObject(tableName.toUpperCase());
    Query cubeQuery = mdmTable.getQuery();
    ArrayList<String> measureColumns = new ArrayList();

    for (MdmBaseMeasure measure : measures) {
        String key = schemaName + "." + tableName.toUpperCase() + "." +
measure.getName();
        measureColumns.add(key.toUpperCase());
    }

    CubeMap cubeMap = mdmCube.createCubeMap();
    cubeMap.setQuery(cubeQuery);

    // Create MeasureMap objects for the measures of the cube and
    // set the expressions for the measures. The expressions specify the
    // columns of the fact table for the measures.
    int i = 0;
    for(MdmBaseMeasure mdmBaseMeasure : measures) {
        MeasureMap measureMap = cubeMap.findOrCreateMeasureMap(mdmBaseMeasure);
        Expression expr = (Expression)
SyntaxObject.fromSyntax(measureColumns.get(i), metadataProvider);
        measureMap.setExpression(expr);
        i++;
    }

    // Create CubeDimensionalityMap objects for the dimensions of the cube and
    // set the expressions for the dimensions. The expressions specify the
    // columns of the fact table for the dimensions.
    // соединить ключи измерений с ключами таблицы фактов
    ArrayList<String> fieldKeyNames = getFactFieldKeyNames(tableName);

    List<MdmDimensionality> mdmDimltys = mdmCube.getDimensionality();
    for (MdmDimensionality mdmDimlty : mdmDimltys) {
        CubeDimensionalityMap cubeDimMap =
cubeMap.findOrCreateCubeDimensionalityMap(mdmDimlty);
        MdmPrimaryDimension mdmPrimDim = (MdmPrimaryDimension)
mdmDimlty.getDimension();
        String columnMap = "";
        String dimName = mdmPrimDim.getName();
        for (String key : fieldKeyNames) {
            String newKey = mdmDBSchema.getName() + "." + tableName + "." + key;
            if (key.endsWith("KEY")) {
                key = key.substring(0, key.length() - 3);
                if (dimName.equals(key)) {
                    columnMap = newKey;
                }
            }
        }

        if (!columnMap.equals("")) {
            Expression expr = (Expression) SyntaxObject.fromSyntax(columnMap,
metadataProvider);
            cubeDimMap.setExpression(expr);
```

```
            // Associate the leaf level of the hierarchy with the cube.
            MdmHierarchy mdmDefHier = mdmPrimDim.getDefaultHierarchy();
            MdmLevelHierarchy mdmLevHier = (MdmLevelHierarchy) mdmDefHier;
            List<MdmHierarchyLevel> levHierList =
mdmLevHier.getHierarchyLevels();
            // The last element in the list must be the leaf level of the
hierarchy.
            MdmHierarchyLevel leafLevel = levHierList.get(levHierList.size() -
1);
            cubeDimMap.setMappedDimension(leafLevel);
        }
    }
}
```

Таблицы измерений и таблицы фактов можно посмотреть с помощью SQL Developer.

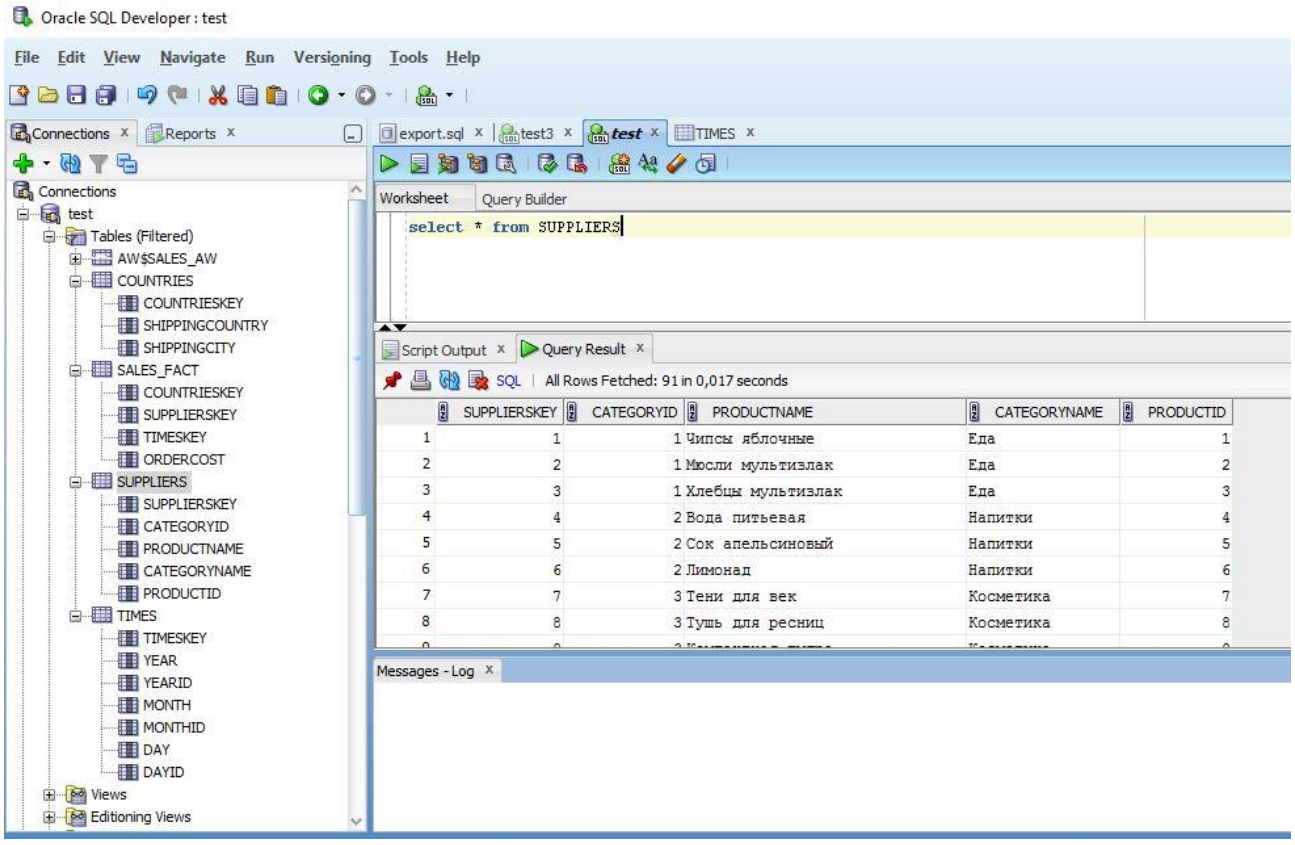

Рис. 10. Созданные таблицы в SQL Developer

Созданную OLAP-систему (куб, измерения, иерархии) можно посмотреть с помощью Analytic Workspace Manager.

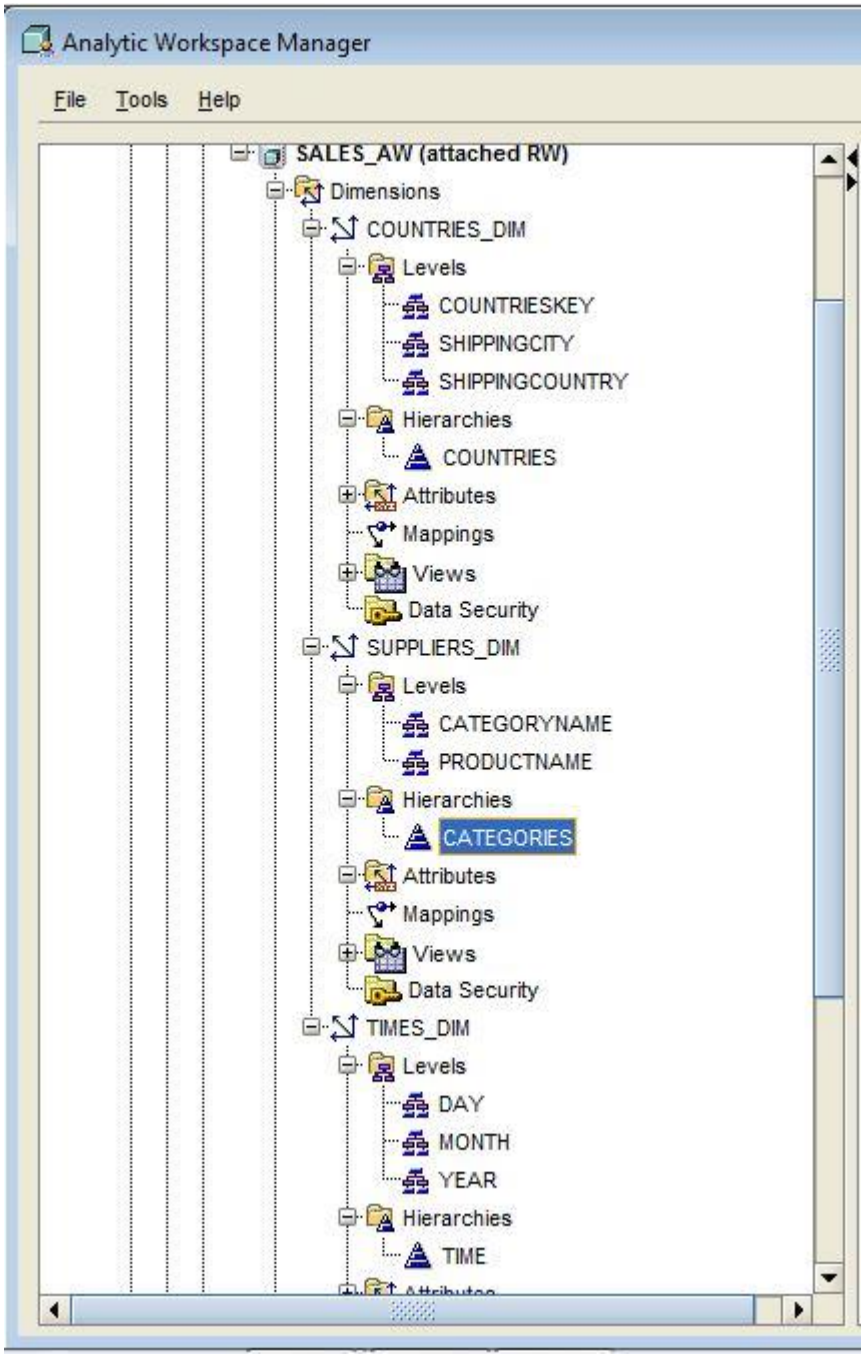

Рис. 11. Analytic Workspace Manager

2.4. Анализ скорости выполнения запросов в реляционной базе данных и в многомерной системе

Запрос для реляционной базы данных:

SELECT p.ProductName, s.ShippingCity, o.OrderCost, o.OrderDate FROM orders AS o INNER JOIN shipping AS s ON o.ShippingId = s.ShippingId INNER JOIN productorder AS po ON o.OrderId = po.OrderId INNER JOIN products AS p ON po.ProductId = p.ProductId WHERE YEAR(o.OrderDate) = 2014;

Запрос для многомерной базы данных:

SELECT countries.shippingcity, times.year, suppliers.productname, sales_fact.ordercost FROM sales_fact INNER JOIN times on sales_fact.timeskey = times.timeskey INNER JOIN suppliers on sales_fact.supplierskey = suppliers.supplierskey INNER JOIN countries on sales_fact.countrieskey = countries.countrieskey;

Таблица 3. Анализ скорости выполнения запроса

Количество данных, в строках	Скорость выполнения запроса	
	Реляционная БД, мс	Многомерная БД, мс
10	9	9
20	31	31
30	55	54
40	91	89
50	130	128
60	178	175
70	247	243
80	279	275
90	365	359
100	378	374
200	680	674
300	1152	1146
400	1224	1218
500	1328	1321
600	1465	1456
700	1539	1530

Продолжение таблицы 3

800	1570	1560
900	1684	1674
1000	1790	1779

Данный анализ показал, что наблюдается незначительное увеличение скорости выполнения запроса для получения одних и тех же данных из реляционной базы данных и из многомерной базы данных. Но при этом, чем больше увеличивается нагрузка на базу данных, тем больше разница в скорости выполнения и многомерная база данных справляется быстрее. Соответственно, для интернет-магазинов с большими объёмами данных для анализа продаж или других показателей лучше подойдёт многомерная база данных.

Заключение

В результате выполнения дипломной работы была разработана программа для преобразования из реляционной базы данных в многомерную для интернет-магазина.

В полной мере были решены поставленные задачи:

1. Проанализированы средства создания интернет-магазинов.
2. Проанализированы существующие способы аналитики продаж, определены необходимые данные для построения OLAP-куба.
3. Сформулированы требования к приложению.
4. Выбраны средства разработки.
5. Приложение разработано.
6. Проанализирована скорость выполнения запросов в реляционной и многомерной БД.

Программа подходит для любых баз данных, работающих в MySQL.

Программа является кроссплатформенной и предоставляется пользователям бесплатно по лицензии GPL, имеет открытый исходный код.

Список литературы

1. Айзенберг Б., Айзенберг Д. Добавьте в корзину. Ключевые принципы повышения конверсии веб-сайта М.: Издательство «Манн, Иванов и Фербер», 2011. 296 с.

2. Шкляревский Ю.Е. TRADEMARK: как бренд-менеджеры делают это СПб.: Питер, 2016. 368 с.

3. Гарифулин А. Ф. Эффективное управление ассортиментом организации URL: http://www.profiz.ru/peo/6_2011/effek_uprav_assortimentom/ (дата обращения: 22.12.2016).

4. Туманов В. Проектирование кубов данных URL: http://www.intuit.ru/studies/courses/599/455/lecture/10190 (дата обращения: 22.12.2016).

5. Гринвальд Р., Стаковьяк Р., Додж Г., Кляйн Д., Шапиро Б., Челья К.Д. Программирование баз данных Oracle для профессионалов М.: Издательский дом «Вильямс», 2007. 784 с.

6. Барсегян А.А. Технологии анализа данных: Data Mining, Visual Mining, Text Mining, OLAP СПб.: БХВ-Петербург, 2007. 384 с.

7. Пирогов В.Ю. Информационные системы и базы данных: организация и проектирование СПб.: БХВ-Петербург, 2009. 529 с.