Introduction

Business Intelligence has become a buzzword in recent years as a support to decision making. Today we can find several database systems which include data warehousing, online analytical processing (OLAP), and data mining technologies. Data warehousing provides an efficient storage, maintenance, and retrieval of data. OLAP is a service that provides a way to create ad hoc queries against the data warehouse in order to answer important business questions. Data mining is a disciple comprising of several algorithms for discovering knowledge in a large bulk of data.

In order to build a data warehouse solution, we need to model a consistent architecture where the operational data will fit well in an integrated and enterprise-wide view as well as to take into consideration a handful implementation strategies to provide a high quality application. The design and implementation of a data warehouse solution sometimes is a very complex challenge in theory and practice. In this article, I will cover the main principles and techniques to design and implement a data warehouse providing my own experience in such an overwhelming challenge.

This is the third article of a series of articles that I want to write in order to share my knowledge and experience in this subject matter. In this part and the next one, we will see how to implement the technical solution for a data warehouse.

Implementing the technical solution

In this section, we’ll build the relational tables which support the dimensional data model that we have designed in the previous section. Then we’ll see how to load these relational tables with data from the AdventureWorks database as the main data source and finally we’re going to build the Analysis Services cubes to present the information to end-users.

Let’s talk about the implementation strategies of the dimensional data model. Open the SQL Server Management Studio and create a new database; then specify a name such as TestDW, and click OK to create it.

In order to create the dimensions and the fact tables, run the following SQL DDL statements (see Listing 1). We have added a unique key constraint for each table’s business key as a clustered index and in order to improve the performance when we need to fetch the dimension records based on a the business key which is very common operation on data warehouse. The clustered indexes enable to physically arrange the data in the dimensions table based on the values of the business key.

```
CREATE TABLE ProductDim(
    ProductDim_ID int IDENTITY(1,1),
    ProductBusinessID int NOT NULL,
    ProductName nvarchar(50) NOT NULL,
    Color nvarchar(15) NULL,
    SubcategoryName nvarchar(50) NOT NULL,
    CONSTRAINT "ProductDim.PK" PRIMARY KEY NONCLUSTERED (ProductDim_ID)
)
GO
CREATE UNIQUE CLUSTERED INDEX "ProductDim.ProductBusinessID" ON ProductDim(ProductBusinessID)
```
CREATE TABLE CustomerDim(
  CustomerDim_ID int IDENTITY(1,1),
  CustomerBusinessId int NOT NULL,
  CustomerType char(10) NULL,
  CONSTRAINT "CustomerDim.PK" PRIMARY KEY NONCLUSTERED (CustomerDim_ID)
)
go

CREATE UNIQUE CLUSTERED INDEX "CustomerDim.CustomerBusinessID" ON CustomerDim(CustomerBusinessId)
go

CREATE TABLE RegionDim(
  RegionDim_ID int IDENTITY(1,1),
  RegionBusinessID int NOT NULL,
  RegionName nvarchar(50) NOT NULL,
  CONSTRAINT "RegionDim.PK" PRIMARY KEY NONCLUSTERED (RegionDim_ID)
)
go

CREATE UNIQUE CLUSTERED INDEX "RegionDim.RegionBusinessID" ON RegionDim(RegionBusinessID)
go

CREATE TABLE TimePeriodDim(
  TimePeriodDim_ID int IDENTITY(1,1),
  Calendar_Date datetime NOT NULL,
  Calendar_Year int NOT NULL,
  Calendar_Month int NOT NULL,
  Calendar_Quarter int NOT NULL,
  Calendar_Week int NOT NULL,
  CONSTRAINT "TimePeriodDim.PK" PRIMARY KEY NONCLUSTERED (TimePeriodDim_ID)
)
go

CREATE UNIQUE CLUSTERED INDEX "TimePeriodDim.Calendar_Date" ON TimePeriodDim(Calendar_Date)
go

CREATE TABLE SalesFact(
  SalesFact_ID int IDENTITY(1,1),
  ProductDim_ID int NOT NULL,
  CustomerDim_ID int NOT NULL,
  RegionDim_ID int NOT NULL,
  TimePeriodDim_ID int NOT NULL,
  SalesVolumes money NOT NULL,
  ShippedUnits int NOT NULL,
  CONSTRAINT "SalesFact.PK" PRIMARY KEY NONCLUSTERED (SalesFact_ID),
  CONSTRAINT "SalesFact.RefProductDim" FOREIGN KEY (ProductDim_ID)
  REFERENCES ProductDim(ProductDim_ID),
  CONSTRAINT "SalesFact.RefCustomerDim" FOREIGN KEY (CustomerDim_ID)
  REFERENCES CustomerDim(CustomerDim_ID),
  CONSTRAINT "SalesFact.RefRegionDim" FOREIGN KEY (RegionDim_ID)
  REFERENCES RegionDim(RegionDim_ID),
  CONSTRAINT "SalesFact.RefTimeDim" FOREIGN KEY (TimePeriodDim_ID)
  REFERENCES TimePeriodDim(TimePeriodDim_ID)
)
go

CREATE INDEX "SalesFact.SalesFact_ID" ON SalesFact (ProductDim_ID, CustomerDim_ID, RegionDim_ID, TimePeriodDim_ID)

Go

Listing 1
Now it's time to load the tables with data from the data sources. Although in the real-world, we can find multiple source systems and different ways of representing information within most business such as spreadsheet, text files and relational databases, we're going to use the AdventureWorks database shipped with SQL Server 2005 as our only data source for the data warehouse. This objective is achieved by an extract, transform and load (ETL) process using several technologies such as Data Transformation Services (DTS) packages in SQL Server 2000 and Integration Services in Microsoft SQL 2005 (SSIS).

In this case, we're going to populate the dimensions and fact tables using SELECT SQL statements.

The first steps are to load the data into the dimension tables from the Sales.Customer, Sales.SalesTerritory, Production.Product and Production.ProductSubcategory tables in the AdventureWorks database (see Listing 2).

```sql
INSERT INTO dbo.CustomerDim(CustomerBusinessId, CustomerType)
SELECT CustomerID AS CustomerBusinessID, CustomerType
FROM AdventureWorks.Sales.Customer;
go

INSERT INTO dbo.RegionDim(RegionBusinessID, RegionName)
SELECT TerritoryID AS RegionBusinessID, [Name]+'-'+CountryRegionCode AS RegionName
FROM AdventureWorks.Sales.SalesTerritory;
go

INSERT INTO dbo.ProductDim(ProductBusinessID, ProductName, Color, SubcategoryName)
FROM AdventureWorks.Production.Product p inner join AdventureWorks.Production.ProductSubcategory s
ON p.ProductSubcategoryID=s.ProductSubcategoryID;
go

DECLARE @dtStartDate datetime;
DECLARE @dtEndDate datetime;
SET @dtStartDate = '1/1/2000';
SET @dtEndDate = '1/1/2012';
WHILE (@dtStartDate <= @dtEndDate)
BEGIN
  INSERT INTO dbo.TimePeriodDim(Calendar_Date, Calendar_Year, Calendar_Month, Calendar_Quarter, Calendar_Week)
  SELECT @dtStartDate, DATEPART(YYYY,@dtStartDate) AS Calendar_Year,
  DATEPART(M,@dtStartDate) AS Calendar_Month,
  DATEPART(Q,@dtStartDate) AS Calendar_Quarter,
  DATEPART(WK,@dtStartDate) AS Calendar_Week;
  SET @dtStartDate = @dtStartDate + 1;
END;
go
```

**Listing 2**

Next step is to load the data into the fact table. One important point to analyze is that dimension tables use a surrogate keys, and these keys are only specific to the data warehouse; thus when we're loading data into the fact tables, a lookup step is always required to map business keys into surrogate keys.

In our case, we're integrating data within the same database system, thus we're going to map business keys to surrogate keys by using inner join operations as well as to extract the sales related data from the Sales.SalesOrderHeader and Sales.SalesOrderDetail tables in the AdventureWorks database. The SELECT SQL statement to load the fact table SalesFact is shown in Listing 3.

```sql
INSERT INTO SalesFact(ProductDim_ID, CustomerDim_ID, RegionDim_ID, TimePeriodDim_ID, SalesVolumes, ShippedUnits)
FROM AdventureWorks.Sales.SalesOrderHeader sh inner join AdventureWorks.Sales.SalesOrderDetail sd
ON sh.SalesOrderID=sd.SalesOrderID inner join dbo.CustomerDim c
ON sh.CustomerID=c.CustomerBusinessId inner join dbo.RegionDim r
ON sh.TerritoryID=r.RegionBusinessID inner join dbo.ProductDim p
```

**Listing 3**
ON sh.OrderDate = t.Calendar_Date
GROUP BY p.ProductDim_ID, c.CustomerDim_ID, r.RegionDim_ID, t.TimePeriodDim_ID, TotalDue;

Go

Listing 3

Now that we have built the data warehouse and load the required data by the integration process, then we need to add the final component of the architecture: the multidimensional database. The relational data warehouse consolidates the underlying data sources, but the end-users need a flexible analytical capability easy to use. The solution is to build an Analysis Services database from the relational data warehouse. Next we’ll adjust the dimensional model created before to better support cubes.

Let’s start the solution by using the BI Development Studio in order to create a new Analysis Services project (see Figure 1). This will create a new local working directory which contains the definition of the underlying objects. Later when we have finished designing the objects, we can deploy the project to a server running an Analysis Services instance.

![Figure 1](image)

The first step is to add a data source and specify the type of database and connection information. In the Solution Explorer window, navigate to the Data Sources node in the tree, then right-click and select New Data Source option in order to launch the Data Source Wizard. In the Select how to define the connection page, click on New button and the Connection Manager windows appears. Set the server name and select the data warehouse database and click on the OK button (see Figure 2).
Next in the Data Source Wizard, click on the Next button, and in the Impersonation Wizard page, select the Use the credentials of the current users option. Click on the Next button and finish the wizard (see Figure 3).

Now we’ll select the parts of the data source to be analyzed by defining a logical view called a data source view (DSV). Using data source views, you can also define relationships between the objects as well as add calculated columns. This is an abstraction layer between the dimensional database and the data source when you’re building the cubes.

Let’s define five data source views to select the dimensions and fact tables from the data source. In the Solution Explorer window, navigate to the Data Source Views node in the tree, then right-click and select New Data Source View option in order to launch the Data Source View Wizard. Click on Next button, in the Select Data Source page,
choose the previously created data source, and click on the Next button and the Select Tables and Views page appears. You select all the tables from the data source by clicking on the >> button (see Figure 4).

Figure 4

Then click on the Next and Finish buttons. You can see in the data source view (see Figure 5) that we don’t need to create the relationships between objects, because the wizard detects these relationships based on the primary key and any referential integrity constraints. In the case, we need more relationships, it’s very simple the creation process. You drag and drop a foreign key column from the source table to the corresponding primary key in the destination table.

Figure 5

Now it’s the time for the creation of the cube. In the Solution Explorer window, navigate to the Cubes node in the tree, then right-click and select New Cube option in order to launch the Cube Wizard. Click on the Next button, in the Select Build Method page, select the Build the cube using a data source option and click on the Next button (see Figure 6).
In the Select Data Source View page, select the previously defined data source view and click on the Next button. The next two pages will allow you to choose a data source view and analyze it in order to identify potential dimensions and facts tables. Click on the Next button, and the Identify Fact and Dimension Tables page, you can see that the wizard detects the CustomerDim, ProductDim, RegionDim and TimePeriodDim as the dimension tables as well as the SalesFact and TimePeriodDim as the fact tables. You must uncheck the fact option for TimePeriodDim table. You must also configure manually the TimePeriodDim table as time dimension (see Figure 7).

Click on Next button, and the Select Time Periods page will appear which enable map the property names of time dimension in the cube with the property names of time dimension table (see Figure 8).
Conclusion

In this article, I've started to show how to build the data warehouse using Microsoft technologies. In next article, I will finish on the solution.

Leave a Reply

Enter your comment here...