

Creating a Data Warehouse using SQL Server

Jens Otto Sørensen
Department of Information Sciences
The Aarhus School of Business, Denmark
jos@hha.dk

Karl Alnor
Department of Information Sciences
The Aarhus School of Business, Denmark
kaa@hha.dk

Abstract:

In this paper we construct a Star Join Schema and show how this schema can be created using the basic tools delivered with SQL Server 7.0. Major objectives are to keep the operational database unchanged so that data loading can be done without disturbing the business logic of the operational database. The operational database is an expanded version of the Pubs database [Sol96].

1 Introduction

SQL Server 7.0 is at the time of writing the latest release of Microsoft's "large" database. This version of the database is a significant upgrade of the previous (version 6.5) both with respect to user friendliness as well as with respect to capabilities. There is therefore in our minds no question about the importance of this release to the database world. The pricing and the aggressive marketing push behind this release will soon make this one of the most used, if not the most used, database for small to medium sized companies. Microsoft will most likely also try to make companies use this database as the fundament for their data warehouse or data mart efforts [Mic98c].

We therefore think that it is of great importance to evaluate whether MS SQL Server is a suitable platform for Star Join Schema Data Warehouses.

In this paper we focus on how to create Star Join Schema Data Warehouses using the basic tools delivered with SQL Server 7.0. One major design objective has been to keep the operational database unchanged.

The Pubs database is a sample database distributed by Microsoft together with the SQL Server. The database

contains information about a fictitious book distribution company. An ER diagram of the Pubs database can be found in appendix A. There is information about publishers, authors, titles, stores, sales, and more. If we are to believe that this is the database of a book distribution company then some of the tables really do not belong in the database, e.g. the table "roysched" that shows how much royalty individual authors receive on their books. Likewise the many-to-many relationship between authors and titles has an attribute "royaltyper" indicating the royalty percentage. A book distribution company would normally not have access to that kind of information, however, these tables do not cause any trouble in our use of the database.

The Pubs database does not have a theoretically clean design, some tables are even not on second normal form. Therefore it does not lend itself easily to a Star Join Schema and as such it presents some interesting problems both in the design of the Star Join Schema and in the loading of data. In the book [Sol96] there is an expanded version (expanded in terms of tuples, not tables) of the Pubs database containing some 168,725 rows in the largest table.

2 Setting up a Star Join Schema on the Basis of the Pubs Database

We need to decide on what the attributes of the fact table should be and what the dimensions should be. Since this is a distributor database we do not have access to information about actual sales of single books in the stores, but we have information on "shipments to bookstores". In [Kim96] there is an ideal shipments fact table. We do not have access to all the attributes listed there but some of them.

2.1 Fact Table

2.1.1 The Grain of the Fact Table

The grain of the fact table should be the individual order *line*. This gives us some problems since the Sales table of the Pubs database is not a proper invoice with a heading and a number of invoice lines.

The Sales table is not even on second normal form (2NF) because the ord_date is not fully dependent on the whole primary key. Please note that the primary key

The copyright of this paper belongs to the paper's authors. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage.

Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'99)

Heidelberg, Germany, 14. - 15. 6. 1999

(S. Gatzju, M. Jeusfeld, M. Staudt, Y. Vassiliou, eds.)

<http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-19/>

of the Sales table is (stor_id, ord_num, title_id). Therefore ord_num is not a proper invoice line number. On the other hand there is also no proper invoice (or order) number in the usual sense of the word i.e. usually we would have a pair: Invoice_number and invoice_line number, but here we just have a rather messed up primary key. This primary key will actually only allow a store to reorder a particular title if the ord_num is different from the order number of the previous order. We must create a proper identifier for each order line *or* we must live with the fact that each orderline has to be identified with the set of the three attributes (stor_id, ord_num, title_id).

We chose here to create an invoice number and a line number, the primary reason being that the stor_id will also be a foreign key to one of the dimensional tables, and we don't want our design to be troubled by overloading the semantics of that key.

2.1.2 The Fact Table Attributes Excluding Foreign Keys to Dimensional Tables

Quantities shipped would be a good candidate for a fact table attribute, since it is a perfectly additive attribute that will roll up on several dimensions. One example could be individual stores, stores grouped by zip number, and stores grouped by state.

Other attributes to be included in the fact table are order date, list price, and actual price on invoice. If we had access to a more complete set of information we would have liked to include even more facts, such as on time count, damage free count etc. But we will content ourselves with these five attributes.

Our initial design for the Shipments Fact tables is as follows:

```
Shipments_Fact(Various foreign keys,  
invoice_number, invoice_line, order_date,  
qty_shipped, list_price, actual_price).
```

Later we describe how to load this fact table based on the tables in the Pubs database.

2.2 Dimensional Tables

We have not enough information to build some of the dimensional tables that would normally be included in a shipment star join schema. Most notable we do not have a ship date (!) thereby apparently eliminating the traditional time dimension, but it pops up again because we do have an order date attribute. However we have chosen not to construct a time dimension. We do not have ship-from and ship mode dimensions either. What we do have is described below.

2.2.1 Store Ship-to Dimension

We can use the stores table "as it is".

Attributes: stor_id, stor_name, stor_address, city, state, zip.

Primary key: stor_id.

2.2.2 Title Dimension

Likewise the title dimension is pretty straightforward but we need to de-normalize it to include relevant publisher information. Titles and publishers have a 1-to-many relationship making it easy and meaningful to de-normalize. We will then easily be able to roll up on the geographic hierarchy: city, state, and country. Another de-normalization along a slightly different line is to extract from the publication date the month, quarter, and year. In other applications it might be interesting to extract also the week-day and e.g. an indicator of whether the day is a weekend or not, but we will restrict ourselves to month, quarter, and year.

The titles and authors have a many-to-many relationship apparently posing a real problem, if we were to denormalize author information into the title dimension. However if we look at our fact table we see that there is really nothing to be gained by including author information, i. e. there is no meaningful way to restrict a query on the fact table with author information that will give us a meaningful answer.

Attributes: title_id, title, type, au_advance, pub_date, pub_month, pub_quarter, pub_year, pub_name, pub_city, pub_state, and pub_country.

Primary key: title_id.

2.2.3 Deal Dimension

The deal dimension is very small and it could be argued that it should be included as attributes in the stores' ship-to dimension, since they are really attributes of the store and not of the individual shipment. We have decided to include it as a dimension in its own right and thereby view it as an attribute of the individual shipment. Even though this is slightly violating the intended semantics of the Pubs database, it does give us one more dimension to use in our tests. We need the payterms attribute from the Sales table and the discounttype and discount attributes from the Discount table.

Attributes: deal_id, payterms, discounttype, and discount

Primary key: discount_id

2.2.4 Fact Table Primary and Foreign Keys

The Shipments fact table will have as its primary key, a composite key consisting of all the primary keys from the three dimension tables and also the degenerate dimension: The invoice and the invoice_line number. Remember that in this case as well as in most cases the invoice dimension, is a degenerate dimension and all the header information in an invoice really belongs in other dimensions. One example here is the payterms attribute that is part of the deal dimension.

Key: stor_id, title_id, discount_id, invoice_no, invoice_lineno

2.3 The Shipment Star Join Schema

Figure 1 is a diagram of the Pubs shipment star Join Schema.

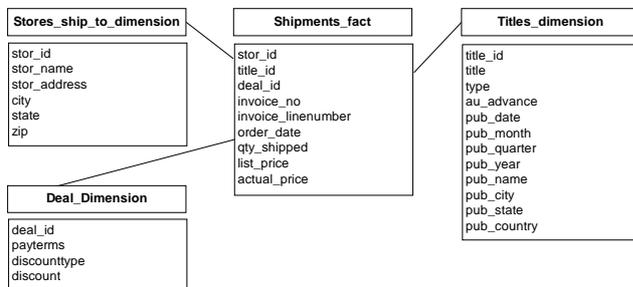


Figure 1: Diagram of the Pubs shipment star join schema

3 Loading the Shipment Star Join Schema Dimensions

There are several options when you want to load tables. The most appealing way to load the schema is to build a *Data Transformation Service Package* (DTS Package) using the DTS Designer (Microsoft, 1988b). The DTS Designer is a graphically oriented tool for building a number of *connections* and a number of *data transformation flows* to run between the connections.

Before creating the DTS Package you need to create the shipment star join schema database and all the tables in it including keys and other constraints. You then create connections and data transformation flows for each of the dimensions, and then finally a connection and a data transformation flow for the fact table. The data transformation flows can be expressed i.a. in SQL including transformations, but transformation can also be carried out using an ActiveX script, Jscript, PerlScript, or VBScript.

To load the dimensions we create a DTS package with four connections (three identical connections to the

Pubs database and one to the shipment star join schema).

These four connections must then be connected by three data transformation flows (also known as *data pumps*). All in all we get the following graphical representation of the DTS Package shown in Figure 2.

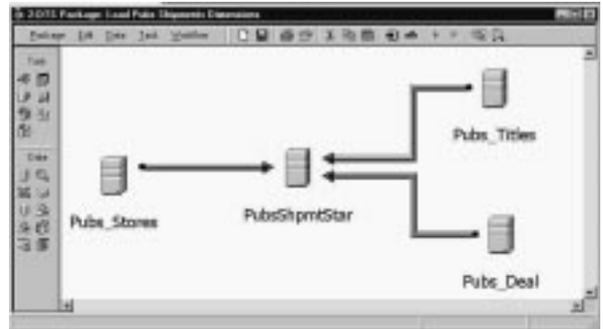


Figure 2: Graphical representation of DTS Package for loading dimensions

Later we will describe each of the three data transformation flows one at a time, but first we need the complete definition of the shipment star join schema.

3.1 The Shipment Star Join Schema

The SQL statements needed to create the star schema is easily derived from the preceding paragraphs and the definition of the Pubs database datatypes. An ER diagram for the shipment star join schema can be seen in appendix B.

3.2 The Store Ship-To Dimension Data Transformation Flow

This is the easiest table to load since it can be loaded without any modification from the Pubs database.

The SQL statement to be included in the DTS data pump between the connection Pubs_Stores and the connection PubsShpmtStar is:

```
SELECT
    s.stor_id,
    s.stor_name,
    s.stor_address,
    s.city,
    s.state,
    s.zip
FROM Stores s
```

The properties sheet for the data pump is shown in Figure 3. Please note that there are four tabs on the properties sheet: Source, Destination, Transformations, and Advanced.

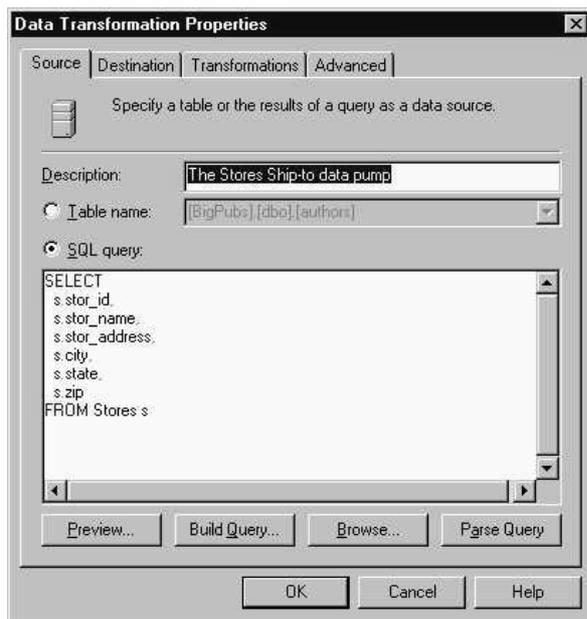


Figure 3: Properties sheet for the Store ship-to data pump

On the second tab, Destination, the destination table must be selected from a drop-down list. Because this is basically just a copy from one table to another we do not need to adjust anything on the third tab, Transformations. The correct default mapping between attributes based on the SELECT statement is already suggested.

3.3 The Title Dimension Data Transformation Flow

The SQL statement to be included in the DTS data pump between the connection Pubs_Titles and the connection PubsShpmtStar is:

```
SELECT
  t.title_id,
  t.title,
  t.type,
  t.advance,
  t.pubdate,
  DatePart(mm, t.pubdate) AS pub_month,
  DatePart(qq, t.pubdate) AS pub_quarter,
  DatePart(yy, t.pubdate) AS pub_year,
  p.pub_name,
  p.city,
  p.state,
  p.country
FROM Titles t, Publishers p
WHERE t.pub_id = p.pub_id
```

Please note the use of a non-standard SQL function to extract the month, quarter, and year information from the pubdate attribute. DatePart(yy, t.pubdate) should in correct SQL92 standard syntax be EXTRACT (YEAR FROM t.pubdate) [Can93, p. 166], but this standard SQL92 syntax is not supported by Microsoft SQL Server 7.0.

3.4 The Deal Dimension Data Transformation Flow

The deal dimension is a bit more difficult as we need to generate values for the deal_id. This will be done automatically since we have defined deal_id to be of type "int IDENTITY". This means that each subsequent row is assigned the next identity value, which is equal to the last IDENTITY value plus 1.

According to our analysis in the Deal Dimension section on page 2, the deal dimension should have one row for each combination of a unique payterms and discounttype.

If we let

```
p = SELECT COUNT(DISTINCT s.payterms)
      FROM Sales s
n = SELECT COUNT(d.discounttype)
      FROM Discounts d
```

Then the cardinality of the deal dimension will be p*n. The SQL statement to be included in the DTS data pump between the connection Pubs_Titles and the connection PubsShpmtStar is.

```
SELECT
  payterms           = p_sales.payterms,
  discounttype       = d.discounttype,
  discount           = d.discount
FROM (select distinct payterms from
sales) AS p_sales, discounts d
```

Please note that we must not have a join condition in the SQL statement, because we want the Cartesian product of the two tables. Also please note that the suggested default mapping between attributes is wrong, therefore the Transformations section of the properties sheet for the deal dimension must be changed to what is shown in Figure 4. The DTS Package can now be executed, and all three dimension tables will be loaded.

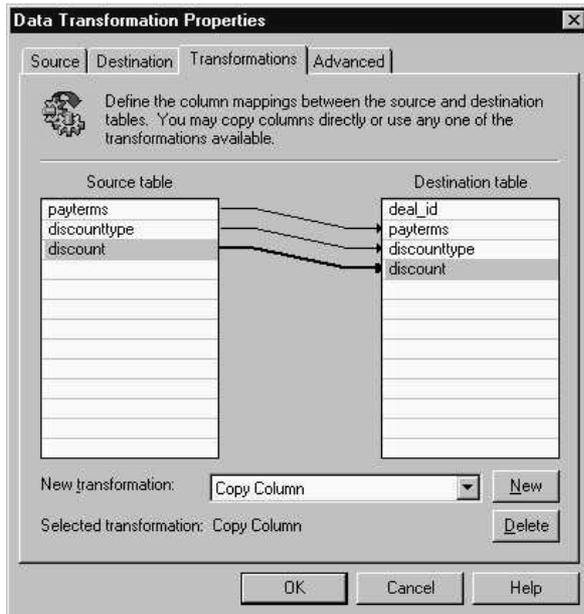


Figure 4: Transformations for the Deal Dimension data pump

4 Loading the Shipment Star Join Schema Fact Table

There are a number of problems here. The most severe is the assignment of invoice numbers and invoice line numbers to each sale. A preliminary analysis of the sales table indicates that most invoices only have one invoice line, but there are also invoices with a number of invoice lines ranging from 2 to 4. That is, the Sales table in the operational database is already denormalised in a way that supports the dimensional design.

Please remember that we have changed the semantics of how a discount is given. In the original Pubs database a discount is something a given store receives, and not all stores are given discounts. However, payterms are assigned, apparently at random. In our interpretation a certain combination of payterms and a discounttype (with associated discount percentage) is given on every sale (e.g. invoice number) posing us with the problem of assigning deal_id's to the sales fact table. The best solution would appear to be assigning a random deal_id to every sale.

4.1 Assigning Invoice - and Invoice Line Numbers to the Fact Table

This has to be done via a number of steps and will involve some intermediary tables as well as a step using an updateable cursor.

4.1.1 Step 1: Creating Intermediary Tables

The first step is to create two intermediary tables, Invoice_Head and Extended_Sales.

Invoice_Head has four attributes, invoice_no (no line numbers yet), stor_id, ord_num, and deal_id. The invoice_no attribute is declared as type INT IDENTITY.

```
CREATE TABLE Invoice_Head (
    stor_id    VARCHAR(4) NOT NULL,
    ord_num    VARCHAR(20) NOT NULL,
    invoice_no INT IDENTITY,
    deal_id    INT,
    CONSTRAINT PK_Invoice_Head
    PRIMARY KEY (stor_id, ord_num))
```

Extended_Sales have the same attributes as the sales table plus four additional attributes, deal_id, price, invoice_no and invoice_lineno and *must* have the same primary key (stor_id, ord_num, title_id) as sales, otherwise the cursor-based updates below will fail.

```
CREATE TABLE Extended_sales (
    stor_id    VARCHAR(4) NOT NULL,
    ord_num    VARCHAR(20) NOT NULL,
    ord_date    DATETIME NOT NULL,
    qty        SMALLINT NOT NULL,
    payterms    VARCHAR(12) NOT NULL,
    title_id    VARCHAR(6) NOT NULL,
    deal_id    INT,
    price      MONEY NOT NULL,
    invoice_no INT NULL,
    invoice_lineno INT NULL,
    CONSTRAINT PK_Extended_Sales
    PRIMARY KEY (stor_id, ord_num, title_id))
```

4.1.2 Step 2: Populating the Invoice_Head Table

Because Invoice_Head.invoice_no has been declared as INT IDENTITY, SQL Server will generate a new value, which is equal to the last IDENTITY value plus 1, for each inserted row, starting with the value one. Therefore the SELECT statement below has no reference whatsoever to the invoice_no attribute. This SELECT statement is part of the data pump between the connections Pubs and PubsShpmtStar in figure 5 and with the proper selections on the Destination and Transformations tabs (not shown here) Invoice_Head will be populated.

```
SELECT
    s.stor_id,
    s.ord_num,
    ROUND(RAND()*9+1, 0, 1) AS deal_id
FROM sales s
GROUP BY s.stor_id, s.ord_num
```

Now we have a table with unique invoice numbers that through joining over (stor_id, ord_num) can be

connected to each sale. The table's `invoice_no` attribute will be populated with a sequence starting with one, and the `deal_id` will be populated with random numbers in the range 1 to $n * p$ (= 9 in our particular case) as discussed in the introduction to this section. Care must be taken when using the `RAND` function without a seed. The SQL Server books online [Mic98a] indicates that repeated calls to `RAND` will generate a sequence of random numbers, but *only* if the calls are a millisecond or more apart. Even on moderately fast machines many, many rows will be populated within a millisecond.

4.1.3 Step 3: Populating the Extended_Sales Table

Step 3 is most conveniently done by dividing it into 3 substeps.

Step 3.1: Populating the `Extended_Sales` table with sales and prices.

A simple DTS data pump will populate `Extended_Sales` with values from the operational database by joining `Sales` and `Titles`. The involved SQL statement is:

```
SELECT
    s.stor_id,
    s.ord_num,
    s.ord_date,
    s.qty,
    s.payterms,
    s.title_id,
    t.price
FROM Sales s JOIN Titles t ON s.title_id
= t.title_id
```

Furthermore the default Transformations will not do, it has to be adjusted. This is very simple and is not shown here.

Please note that step 2 and step 3.1 can be done in parallel.

Step 3.2: Update `Extended_Sales` with invoice numbers.

The `Invoice_head` table will be used to insert the correct new `invoice_numbers` into the expanded copy of the sales table.

```
UPDATE extended_Sales
SET invoice_no = i.invoice_no
FROM Invoice_Head i
WHERE Extended_Sales.ord_num = i.ord_num
AND Extended_sales.stor_id =i.stor_id
```

Please note that this time we do not need a data pump between the two connections, because we are operating entirely within the `PubsShipmStar` database. Therefore we use an "Execute SQL Task" [Mic98b]. See also figure 5 on page 10-7.

Step 3.3: Insert invoice line numbers into `Extended_Sales`.

Inserting the correct line numbers will involve a cursor, because we need to go down the rows of the table one by one and compare the previous `invoice_no` with the current, and thereby decide whether the line number attribute should be incremented by one or reset to one. Please note the `ORDER BY` clause in the cursor, this is essential, ensuring that when a break is encountered it is guaranteed that we will not meet that invoice number again further down the table.

```
/* auxillary variables */
DECLARE
    @prev_inv_no INT,
    @cur_inv_no INT,
    @inv_lineno INT,
/* Extended_Sales attribute variables */
    @stor_id VARCHAR (4),
    @ord_num VARCHAR(20),
    @title_id VARCHAR (6),
    @invoice_no INT,
    @invoice_lineno INT

DECLARE Extended_Sales_cursor
CURSOR FOR
    SELECT stor_id, ord_num, title_id,
invoice_no, invoice_lineno
FROM Extended_Sales
ORDER BY invoice_no
FOR UPDATE OF invoice_lineno

OPEN Extended_Sales_cursor

/* Initialize and fetch first before
entering loop */
FETCH NEXT FROM Extended_Sales_cursor
INTO @stor_id, @ord_num, @title_id,
@invoice_no, @invoice_lineno

SET @prev_inv_no = @invoice_no
SET @inv_lineno = 1

WHILE (@@fetch_status = 0)
/* equals not.eof) in 3th gen. languages*/
BEGIN
    UPDATE Extended_Sales
    SET Extended_Sales.invoice_lineno =
@inv_lineno
WHERE CURRENT OF
Extended_Sales_cursor

    FETCH NEXT
FROM Extended_Sales_cursor
INTO @stor_id, @ord_num, @title_id,
@invoice_no, @invoice_lineno

SET @cur_inv_no = @invoice_no
IF @cur_inv_no = @prev_inv_no
SET @inv_lineno = @inv_lineno +1
ELSE
SET @inv_lineno = 1
```

```

SET @prev_inv_no = @cur_inv_no
END /* WHILE */

```

```

CLOSE Extended_Sales_cursor
DEALLOCATE Extended_Sales_cursor

```

This code is contained in the Execute SQL Task named Step 3.3 in figure 5.

4.1.4 Step 4: Populate Shipments_Fact Table

We have both the Deal_Dimension and the Extended_Sales tables making it relatively easy to insert the relevant tuples into the Shipment_Fact table. Once again we will use an Execute SQL Task, see figure 5 below. The SQL statement is:

```

INSERT INTO Shipments_Fact
(stor_id, title_id, deal_id,
invoice_no, invoice_lineno, order_date,
qty_shipped, list_price, actual_price)
(SELECT e.stor_id, e.title_id, e.deal_id,
e.invoice_no, e.invoice_lineno,
e.ord_date, e.qty, e.price,
(1 - d.discount/100)*e.price
FROM
Extended_Sales e JOIN Deal_Dimension d
ON e.deal_id = d.deal_id)

```

These four steps conclude the loading of the fact table. Figure 5 below is a picture of the whole process. Please note that the dotted arrows indicate that the transformation step is dependent on the succes of the previous step.

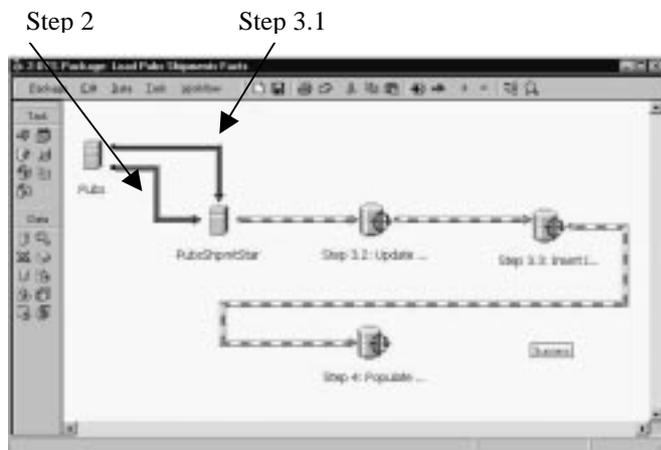


Figure 5: DTS Package for loading the fact table

5 Conclusion

In this paper we have concentrated on the logical design of a Star Join Schema and how easy this Schema can be created using the basic tools delivered with SQL Server 7.0. In particular we have demonstrated that all transformations have been done without changing the operational database at all.

We have not compared with other databases and we have not done extensive timings, since it is very dependent on the disk subsystem, the processor power available and the amount of physical RAM. Also equally important are the choices of indexing and size of key attributes. In a short paper like this we can not possibly cover all aspects of traditional Star Join Schema data warehousing, and we have not looked into other very interesting issues such as programming support in traditional third generation languages, end user tools etc.

In particular we are impressed with the relatively ease of use compared to the complexity of the tasks that SQL Server 7.0 offers. Microsoft SQL Server 7.0 has many of the tools required to build a traditional Star Join Schema Data Warehouse from the ground based on an operational database with transaction oriented tables.

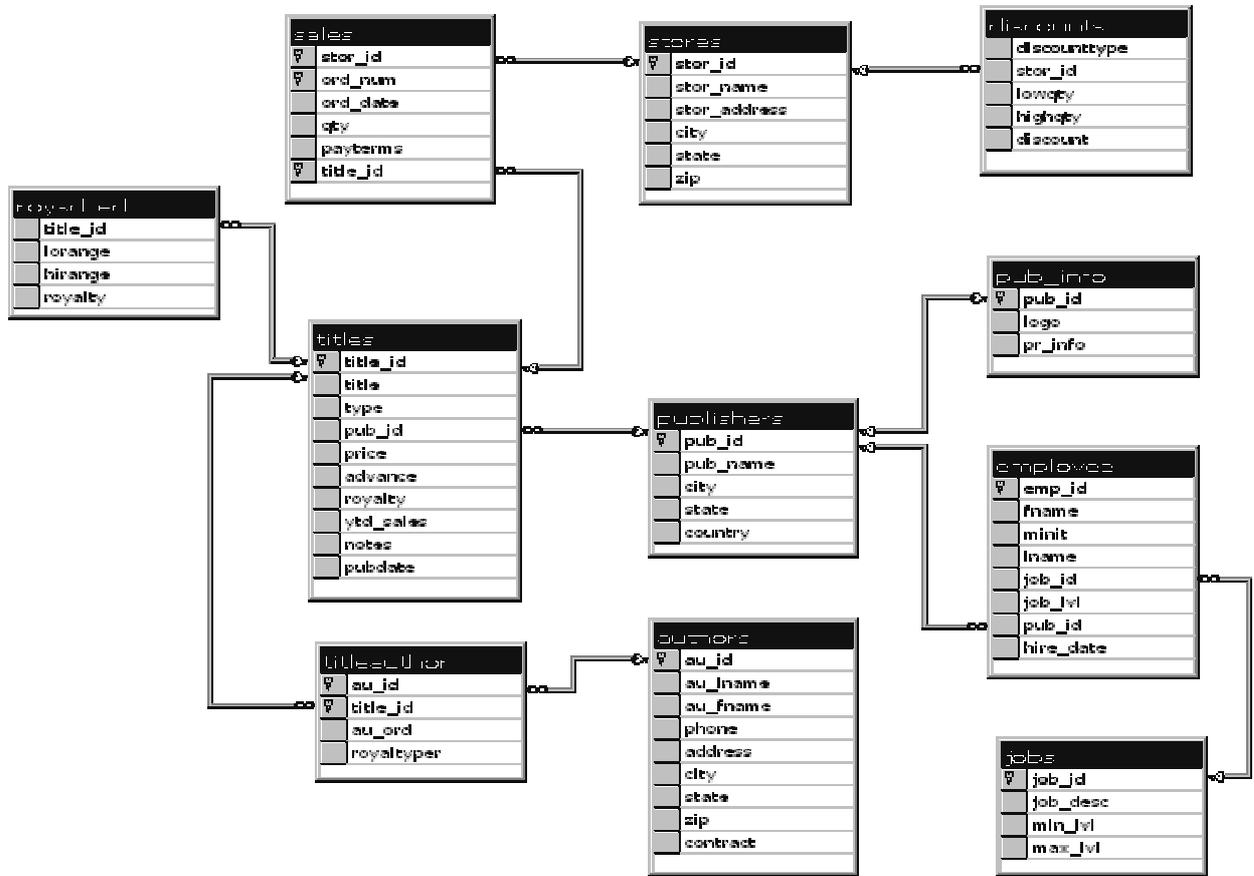
Notably, the DTS Packages allow for an integrated approach to loading and updating the data Warehouse.

Although more ways to get data into a datawarehouse exist, we believe that the easiness offered by the DTS Packages will be a very strong selling point.

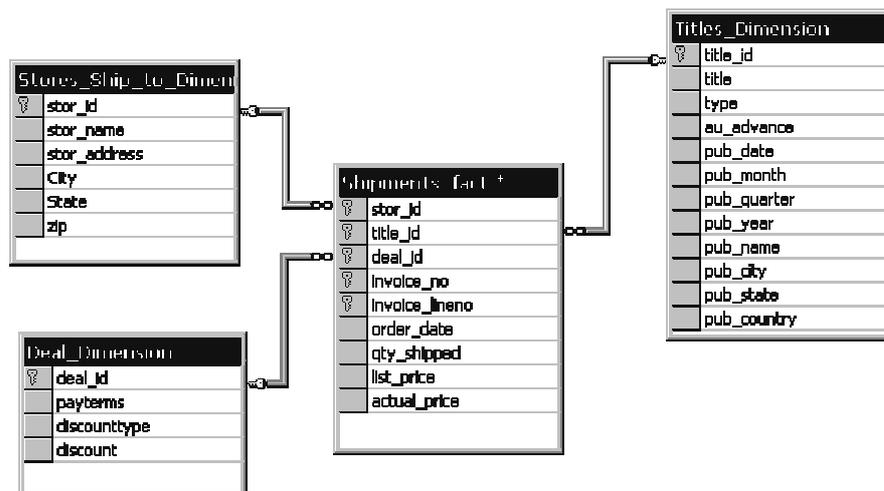
In our opinion the best procedure is to build a prototype of the data warehouse along the lines given in this paper and then test it either on the actual hardware, or if that is not possible on hardware that have relatively well understood scaling capabilities. This way building a traditional data warehouse could be a process very much like developing systems with prototypes and substantial user involvement.

Future work along these lines could include a closer look at the hardware required to support a large data warehouse and various performance issues. It would also be interesting to look closer at the OLAP Server and the PivotTable Service, both middle-tier tools for analyzing and presenting the data included with SQL Server.

6 Appendix A: ER Diagram for the Pubs Database



7 Appendix B: ER Diagram for the Pubs Shipment Star Join Schema



8 Appendix C: Commands Needed to Create BigPubs

8.1 Creation of the schema of the operational database BigPubs.

Please note that all testing has been done on SQL Server 7.0 released version.

Please also note that the script used to create the sample database schema comes from [Sol96] and is originally intended to be used with SQL Server 6.5, but version 7.0 is capable of running it unmodified.

8.2 Bulk Loading of Data Into the Operational Database

These commands have been executed from SQL Query Analyser.

```
BULK INSERT authors      FROM
"D:\Source\authors.bcp"
BULK INSERT discounts   FROM
"D:\Source\discounts.bcp"
BULK INSERT employee    FROM
"D:\Source\employee.bcp"
BULK INSERT jobs        FROM
"D:\Source\jobs.bcp"
BULK INSERT pub_info    FROM
"D:\Source\pub_info.bcp"
BULK INSERT publishers  FROM
"D:\Source\publishers.bcp"
BULK INSERT roysched    FROM
"D:\Source\roysched.bcp"
BULK INSERT sales       FROM
"D:\Source\sales.bcp"
BULK INSERT stores      FROM
"D:\Source\stores.bcp"
BULK INSERT titleauthor FROM
"D:\Source\titleauthor.bcp"
BULK INSERT titles      FROM
"D:\Source\titles.bcp"
```

It turns out that one book does not have a price, which is unacceptable in our use of the database. Therefore the following SQL update has been performed on BigPubs.

```
UPDATE Titles SET price = 12.0 WHERE
title_id = "PC9999"
```

References

- [Sol96] D. Solomon, R. Rankins, K. Delaney, and T. Dhawan, *Microsoft SQL Server 6.5 Unleashed*, 2 ed. Indianapolis: SAMS Publishing, 1996.
- [Mic98a] Microsoft Corporation, "Using Mathematical functions," in *Accessing and Changing Data. Included in the SQL Server Books Online. SQL Server v. 7.0*: Microsoft Corporation, 1998.
- [Mic98b] Microsoft Corporation, "DTS Designer," in *Data Transformation Services. Included in SQL Server Books Online. SQL Server v. 7.0*: Microsoft Corporation, 1998.
- [Mic98c] Microsoft Corporation, "Microsoft Back-Office - SQL Server 7.0 Data Warehousing," 1998.
<http://www.microsoft.com/backoffice/sql/70/gen/dw.htm>. Read on 01-Dec-1998.
- [Kim96] R. Kimball, *The Data Warehouse Toolkit*. New York: Wiley Computer Publishing, 1996.
- [Can93] S. J. Cannan and G. A. M. Otten, *SQL - The Standard Handbook*. Maidenhead, UK: McGraw-Hill Book Company Europe, 1993.

Acknowledgements

This work is supported in part by the Danish Social Science Research Council, Project no. 9701783.

Part of the work was done while the first author was visiting the Nykredit Center for Database Research, Aalborg University in the winter of 1998/99.