

## 22.1 Introduction: Data access for decision makers

When you used the ACCESS report writer to create an invoice report in [Section 16.4](#), you may have made an important observation:

*Creating a report from a relational database requires a solid understanding of where the data is stored and how the tables are related.*

In this case, you are the person who implemented the database so you know that each **order** consists of many **orderDetails** and that **orderDetails.ActualPrice** is what the items sold for, not **Products.UnitPrice** and so on. But who else in your organization could realistically be expected to know all these details?

### 22.1.1 Database specialists versus business specialists

A real-world business application typically contains hundreds or thousands of tables and a mind-boggling web of relationships between tables. The good news is that within each organization, there is an individual who

understands how all the data fits together—the database administrator (DBA). The bad news is that the DBA is a database specialist, not a business specialist.

The decision-makers who most need the information locked up inside the database (marketing managers, executives) seldom have any training (or even interest) in the subtleties of third-normal form, concatenated keys, or referential integrity. Similarly, as a class of individuals, DBAs are not known for their marketing instincts or general business acumen.

### 22.1.2 Dimensional data modeling

In the early 90s, a new class of database application—the data warehouse—emerged to address the problems encountered by managers as they tried to access information locked up inside of transaction processing systems. At the simplest level, a data warehouse is simply a read-only copy of the data in a transaction processing system. However, instead of being optimized for transaction processing, a data warehouse is optimized for reporting and decision support. Specifically, a data warehouse is based on a “dimensional” data model rather than a “normalized” data model.



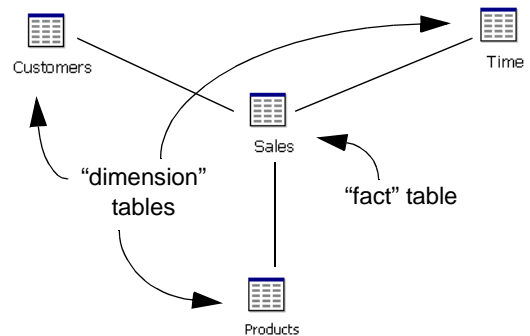
An example of a dimensional data model is shown in [Figure 22.1](#): the center table—**Sales**—contains the “fact” of primary importance to decision makers: the dollar amount sold. The other tables—**Customers**, **Time**, and **Products**—are the “dimensions” along which sales vary.

For example, a manager may want to know who her best customers are this quarter and what they are buying. Finding an answer using a normalized database would require some reasonably sophisticated knowledge of both a query language and the table structure of the database. However, as you will see, it is very easy to answer this type of question when the source of the data is organized into facts and dimensions. The ease with which business users can create complex queries is an important benefit of the data warehousing approach to decision support.

### 22.1.3 Building a data warehouse

In this lesson, you will build a very small-scale data warehouse. Although your warehouse will be implemented in **ACCESS** and will be a fraction of the size of a real-world warehouse, even a small warehouse is sufficient to illustrate the critical elements of data extraction and dimensional data modeling. In [Lesson 23](#) you will use your data warehouse to explore your data in greater detail and answer complex questions about your business.

FIGURE 22.1: A dimensional data model.



## 22.2 Learning objectives

- understand the difference between data models for transaction processing and data models for decision support
- denormalize data to create dimension tables
- extract data to create a fact table
- build a star schema
- use grouping to change the granularity of a fact table



## 22.3 Exercises

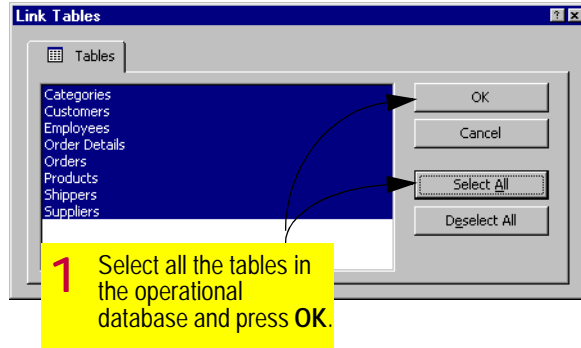
Since you may not have enough data in your order entry application to yield interesting query results, we will use the NORTHWIND TRADERS sample database (recall [Section 4.3.2](#)) as the source for our data warehouse. The NORTHWIND TRADERS database is also small by real-world standards; however, it contains enough orders (just over 800) to make the querying exercises in this lesson and [Lesson 23](#) worthwhile.

### 22.3.1 Preliminaries

Rather than alter the NORTHWIND TRADERS sample database, we are going to create links to its tables (recall [Section 8.3.3](#)) and **extract** the data into a new database file.

- ➔ Create a new blank database in ACCESS called **OrderEntryWarehouse**.
- ➔ From the main menu, select **File** → **Get External Data** → **Link Tables**.
- ➔ Use the search feature of the “Link” dialog box to find the NORTHWIND TRADERS database (recall [Section 4.3.2](#)).
- ➔ When asked which tables you would like to create links to, select all the tables, as shown in [Figure 22.2](#).

FIGURE 22.2: Select tables in the operational system to link to.



Your data warehouse database should now contain links to all the tables in a transaction processing system application.

- ➔ Since you are still playing the role of the DBA at this point, bring up the relationships window to get a sense for the structure of the NORTHWIND TRADERS application.

### 22.3.2 Extraction, cleaning, and transformation

Data extraction is the process of copying the data from the **operational system** (i.e., the NORTHWIND TRADERS order entry system) to the



data warehouse. During extraction, two things normally happen to the data:

1. **Cleaning** – Data cleaning (or scrubbing) involves removing incorrect or inconsistent data, missing values, and so on. As you can imagine, this is costly and difficult process that involves a combination of specially written programs and manual intervention.
2. **Transformation** – Data from the transaction processing system must be transformed into a format suitable for reporting, analysis, and other decision support activities. Typically, transformation involves **de-normalizing** dimension tables and **pre-computing** fact tables. Both these processes are illustrated in the following sections.



To keep things simple, we are going to assume that the NORTHWIND TRADERS order entry application has been designed to minimize the possibility of errors entering the database. As such, the data cleaning stage is assumed to be unnecessary.

In the next few sections, you will use action queries to copy data from the linked tables you created in [Section 22.3.1](#) to new tables in your **OrderEntryWarehouse** database.

## 22.3.3 Creating dimension tables

Dimension tables are relatively static: they contain lists of products, customers, and so on. However, since the dimensions determine the types of questions you can ask of your data warehouse, it is important to put some thought into their design. There are two important questions to answer before you dive in:

1. What dimensions are important for making business decision?
2. What is the appropriate level of **granularity** for each dimension?

These questions are addressed on a case-by-case basis in the following sections.

### 22.3.3.1 The product dimension

Clearly, you are going to want to look at sales by product. The granularity issue is whether you need to look at individual SKUs (stock keeping units) or whether a coarser-grained approach (e.g., product category) is sufficient.

- ➔ Create a new query called **qryProductDimensionExtract** based on the **Products** table.
- ➔ From the **Query** menu, select **Make Table Query**.



➔ When prompted for the name of the new table, enter **dimProducts**.



Since you already have a linked table called **Products**, you cannot use the same name. In addition, the **dim** prefix is an easy way to indicate that the table contains denormalized dimension data.

In the NORTHWIND TRADERS database, each product is assigned to a category (beverages, condiments, and so on). If we analyze products by individual SKUs (in this case, **ProductID**), then the granularity is quite fine. However, if we lump all products within a particular category together and perform our analyses at the category level, then the granularity is more coarse.



If your dimension is too fine-grained, your data warehouse will be very large and may involve excessive processing to respond to user queries. However, if your dimension is too coarse-grained, you will be unable to ask certain kinds of questions.

In this case, we are going to include both product and category information. This will permit the user of the warehouse to drill-down to the appropriate level of granularity.

➔ Include the **Categories** table in the query. There should be a one-to-many relationship between **Categories** and **Products**.

➔ Project the **ProductName** and **CategoryName** fields into the query. These will be the values the user sees.

➔ Project the **ProductID** field into the query. This value will be used as a key to link the dimension table to the fact table.

➔ Finally, project the **UnitPrice** field into the query.



The rationale for including **UnitPrice** in the dimension table is that users may want to limit their analysis to high (or low) valued items. Having the **UnitPrice** field in the data warehouse allows users to apply price-related constraints to their queries. More generally, knowing what fields to include in a data warehouse requires a good understanding of how users make decisions. When in doubt, it is probably best to err on the side of including too much.

➔ Select **Query** → **Run** to execute the query. Examine the contents of **dimProducts**, as shown in [Figure 22.3](#).



FIGURE 22.3: The extraction query for the product dimension.

1 Create a query to extract product data from the operational database.

Field:	ProductID	ProductName	CategoryName	UnitPrice
Table:	Products	Products	Categories	
Sort:				
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:				
or:				

ProductID	ProductName	CategoryName	UnitPrice
1	Chai	Beverages	\$18.00
2	Chang	Beverages	\$19.00
3	Aniseed Syrup	Condiments	\$10.00
4	Chef Anton's Cajun Seasoning	Condiments	\$22.00
5	Chef Anton's Gumbo Mix	Condiments	\$21.35
6	Grandma's Boysenberry Spread	Condiments	\$25.00
7	Uncle Bob's Organic Dried Pears	Produce	\$30.00
8	Northwoods Cranberry Sauce	Condiments	\$40.00
9	Mishi Kobe Niku	Meat/Poultry	\$97.00
10	Ikura	Seafood	\$31.00
11	Queso Cabrales	Dairy Products	\$21.00
12	Queso Manchego La Pastora	Dairy Products	\$38.00
13	Konbu	Seafood	\$6.00
14	Tofu	Produce	\$23.25
15	Genen Shouyu	Condiments	\$15.50

2 Verify the resulting dimension table.

### 22.3.3.2 Taking a closer look at the products dimension table

There are a couple of things to notice about the `dimProducts` table:

1. **Denormalization** – When designing transaction processing systems, we make every effort to eliminate redundancy in our tables (recall the discussion of

normalization in [Section 7.1.1](#)). In data warehousing, database design logic is turned on its head: In order to save the computational effort of making a join when running queries against the data warehouse, the dimension table includes information from multiple entities (e.g., products and categories). Since data in the warehouse is



never changed or edited, this denormalized structure does not lead to the anomalies discussed in [Section 7.1](#).

2. **User-friendly values** – Since the extracted data is ultimately going to be used for creating reports, meaningful field values (such as **ProductName** and **CategoryName**) are used instead of key fields (like **CategoryID**). Key fields (such as **ProductID**) are only added when necessary for linking to a fact table.

### 22.3.3.3 The customer dimension

The customer dimension is similar to the product dimension in that a hierarchical relationship is implicit in the data. In this case, assume that users require data right down to the level of the individual customers, but may also want to aggregate across cities, regions, and countries.

- Create a new make-table query called **qryCustomerDimensionExtract**.
- Use **dimCustomers** as the name of the target table.
- Include the **Customers** table. In the NORTHWIND TRADERS database, the region information is included within the

**Customers** table so there is no need to create a join to another table.

- Project the following fields into the query: **CompanyName**, **City**, **Region**, and **Country**.
- Project **CustomerID** to enable the table to be linked to the fact table.
- Execute the query and verify the contents of **dimCustomers**, as shown in [Figure 22.4](#).

### 22.3.3.4 The time dimension

The time dimension is different from the products and customers dimension in that time exists independently of any particular data warehouse application. As a consequence, it is possible to create a *generic* time dimension table consisting of a date ID plus days of the week, months, quarters, years and so on. This dimension table could be used for all data warehouse applications.

In this lesson, we are going to take a different approach for two reasons.

1. **Event time vs. calendar time** – Although sales per day (or hour or minute) may be a meaningful piece of information, we are also interested in sales *per order*. One may not normally think of **OrderID** as a measure of time; however, it is important to remember that each order is an event.



FIGURE 22.4: The extraction query for the customer dimension.

1 Create a query to extract customer data from the operational database.

CustomerID	CompanyName	City	Region	Country
ALFKI	Alfreds Futterkiste	Berlin		Germany
ANATR	Ana Trujillo Emparedados y helados	México D.F.		Mexico
ANTON	Antonio Moreno Taquería	México D.F.		Mexico
AROUT	Around the Horn	London		UK
BERGS	Berglunds snabbköp	Luleå		Sweden
BLAUS	Blauer See Delikatessen	Mannheim		Germany
BLONP	Blondel père et fils	Strasbourg		France
BOLID	Bólido Comidas preparadas	Madrid		Spain
BONAP	Bon app'	Marseille		France
BOTTM	Bottom-Dollar Markets	Tsawassen	BC	Canada
BSBEV	B's Beverages	London		UK
CACTU	Cactus Comidas para llevar	Buenos Aires		Argentina
CENTC	Centro comercial Moctezuma	México D.F.		Mexico
CHOPJ	Chop-suey Chinese	Bern		Switzerland
COMMI	Comércio Mineiro	São Paulo	SP	Brazil

2 Verify the resulting dimension table.

Since many orders can occur per day, the granularity desired in this context is finer than the granularity of a generic date-based dimension table.

2. **Date manipulation functions**— Since each order has an order date, it is possible to derive the coarser-grained values of time

using specialized data manipulation functions.

- ➔ Create a new make-table query called `qryTimeDimensionExtract`.
- ➔ Use `dimTime` as the name of the target table.



- Include the **Orders** table and project the **OrderDate** field.
- Project the **OrderID** field so that a link can be made to the fact table.

### 22.3.3.5 Transforming the OrderDate field

The **OrderDate** field is defined as a Date/Time data type and contains all the information we require about day, month, year and so on. The trick is to extract this information and display it in a user-friendly format. To do this, we will use calculated fields and the built-in **DatePart()** function.

- Create a new calculated field in **qryTimeDimensionExtract** called **Year** and define it as follows:

**NL Year: DatePart("yyyy", OrderDate)**

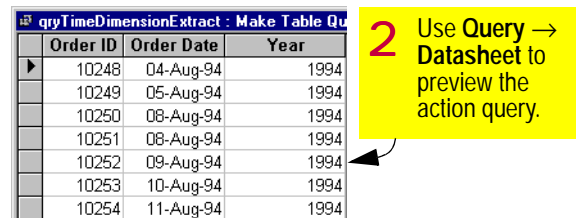
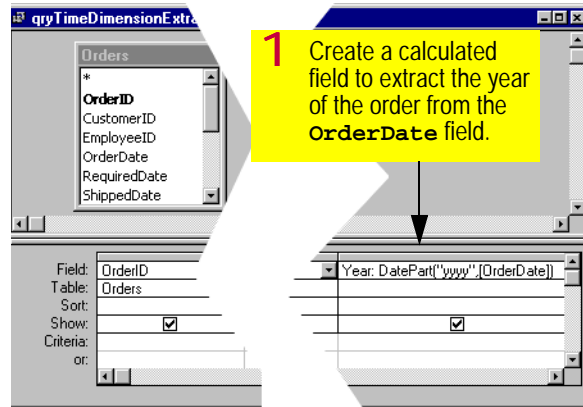
The **DatePart()** function takes two arguments: a special set of characters that determines what part of the date is returned and a valid date. For example, the argument "yyyy" tells the function to return a four-digit year.



Use the on-line help system and search under "datepart" to learn more about the function and its arguments.

- Select **View** → **Datasheet View** to preview new table and verify the **DatePart()** function, as shown in [Figure 22.5](#).

FIGURE 22.5: Use the **DatePart()** function to show the year of an order.



MICROSOFT updates the NORTHWIND TRADERS database from time to time. As a



consequence, the dates return by your queries may not correspond exactly to those shown in [Figure 22.5](#).

The procedure for transforming the month is the same except that `DatePart()` only returns the ordinal number of the month, not the name. Since the whole purpose of this exercise is to provide user-friendly, readable values for each dimension, a means of mapping month numbers to month names is required.

One approach is to create a lookup table of month numbers and months. A somewhat simpler approach (which we will use here) is to use the `Choose()` function within `ACCESS`.

➔ Create a new calculated field called month and define it as follows:

```
NL Month: DatePart("m", OrderDate)
```

➔ Preview the results and verify that the result is a number from 1 to 12.

➔ Modify the `Month` field so that the `DatePart()` function provides the first argument for the `Choose()` function:

```
NL Month: Choose(DatePart("m",  
OrderDate), "Jan",  
"Feb", "Mar", "Apr", "May", "Jun", "Jul",  
"Aug", "Sep", "Oct", "Nov", "Dec")
```



The `Choose()` function maps an index number (1, 2, ...) to the corresponding value in a list of choices. For example, the index number 2 maps to the second choice, and so on. See on-line help for more information about the `Choose()` function.

➔ Create a new calculated field called `Quarter` as follows:

```
NL Quarter: "Q" & DatePart("q",  
OrderDate)
```



When the first argument for the `DatePart()` function is "q", the function returns a value 1 to 4 corresponding to the quarter. To make it more readable, the letter "Q" is added to the front of each value:

➔ Include a final calculated field called `Holiday`:

```
NL Holiday: False
```



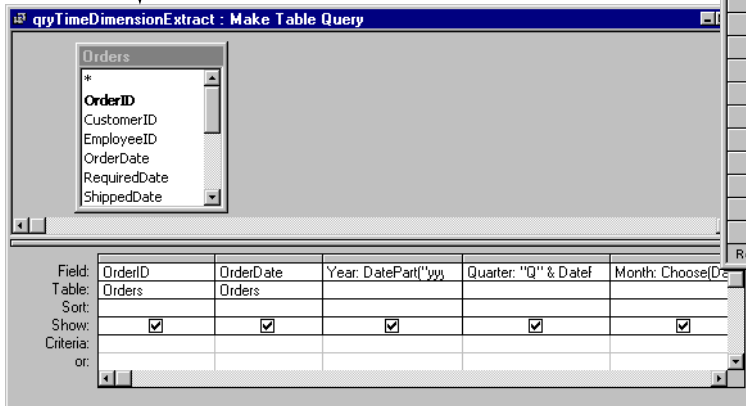
The `Holiday` field creates a new field in the `dimTime` table to indicate whether a particular date is a holiday. This type of information is often useful in a retail context for interpreting spikes in demand. Of course, someone has to go through the `dimTime` table and change `Holiday = True` where appropriate.



- Execute the query and verify the results, as shown in Figure 22.6.

FIGURE 22.6: The extraction query for the time dimension.

1 Create a query to extract time data from the operational database.



OrderID	OrderDate	Year	Quarter	Month	Holiday
10330	16/11/94	1994	Q4	Nov	0
10331	16/11/94	1994	Q4	Nov	0
10332	17/11/94	1994	Q4	Nov	0
10333	18/11/94	1994	Q4	Nov	0
10334	21/11/94	1994	Q4	Nov	0
10335	22/11/94	1994	Q4	Nov	0
10336	23/11/94	1994	Q4	Nov	0
10337	24/11/94	1994	Q4	Nov	0
10338	25/11/94	1994	Q4	Nov	0
10339	28/11/94	1994	Q4	Nov	0
10340	29/11/94	1994	Q4	Nov	0
10341	29/11/94	1994	Q4	Nov	0
10342	30/11/94	1994	Q4	Nov	0
10343	01/12/94	1994	Q4	Dec	0
10344	02/12/94	1994	Q4	Dec	0
10345	05/12/94	1994	Q4	Dec	0

2 Verify the resulting dimension table.

### 22.3.4 Creating a fact table

A fact table contains one or more results of interest for each unique combination of dimensions. For example, if we were interested in the value of sales of products to customers

per day, then we would compute this value for each product  $\times$  customer  $\times$  day and store it in a table. In the case of NORTHWIND TRADERS, the results would be  $77 \times 91 \times 365 = 2.5$  million facts per year.



The number of non-zero facts is be much smaller than 2.5 million since all customers do not order all products every day of the year. Despite this sparseness however, fact tables tend to be very large.

## 22.3.4.1 Determining foreign keys

Through the selection of keys in the dimension tables, we have already determined the foreign keys that must be included in the fact table: **ProductID**, **CustomerID**, and **OrderID**.

- ➔ Create a new make-table query called **qryFactExtract**.
- ➔ Use **factSales** as the name of the target table.
- ➔ Add the **Order** and **OrderDetails** tables to the query.
- ➔ Note that the **Order** and **OrderDetails** tables contain all the fields we need to create joins to the dimension tables: **ProductID**, **CustomerID**, and **OrderID**.
- ➔ Project the foreign keys into the query definition.

## 22.3.4.2 Calculating sales

What remains to be determined is the dollar value of sales for each combination of the dimensional values. To calculate the total sale for each product × customer × order combination, we have to know a couple of things about the data:

1. The total value of an order is the sum of extended prices of the line items in the order.
  2. The **OrderDetail.UnitPrice** value can be discounted by the amount in **OrderDetails.Discount**. The extended price calculation must therefore include the discount.
- ➔ Create a calculated field called total sale as follows:  

```
NL TotalSale: Quantity * (1-Discount)* UnitPrice
```
  - ➔ Preview the results and examine the results as shown in [Figure 22.7](#).

## 22.3.5 Refresh intervals

Let's summarize what you have done to this point: You have extracted and transformed data from one database and stored it in another database. The new database (**OrderEntryWarehouse.mdb**) is simply a static



FIGURE 22.7: The fact table for order-level analyses of sales.

**1** Project the necessary foreign keys.

**2** Calculate the total sale as a function of information in the **Order Details** table.

**3** Verify the resulting fact table.

**?** NORTHWIND uses a short textual code for **CustomerID** rather than an AutoNumber.

OrderID	CustomerID	ProductID	TotalSale
10248	VINET	11	168
10248	VINET	42	98
10248	VINET	72	174
10249	TOMSP	14	167.4
10249	TOMSP	51	1696
10250	HANAR	41	77
10250	HANAR	51	1261.3999912
10250	HANAR	65	214.1999985
10251	VICTE	22	95.75999925
10251	VICTE	57	222.29999983
10251	VICTE	65	336
10252	SUPRD	20	2462.3999981
10252	SUPRD	33	47.49999963
10252	SUPRD	60	1088
10253	HANAR	31	200
10253	HANAR	39	604.8

Field:	OrderID	CustomerID	ProductID	TotalSale: [Quantity]*[1-[Discount]]*[UnitPrice]
Table:	Orders	Orders	Order Details	
Sort:				
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:				
or:				

copy (or “snapshot”) of the NORTHWIND TRADERS application.

Clearly, your warehouse data is out-of-date as soon as a new transaction is added to the operational system. But since we plan to use the data warehouse to see the big picture (e.g., sales trends over the last four quarters), an

order here or there does not make that much difference. You can refresh the data warehouse daily, weekly, monthly, or according to whatever schedule makes sense.



## 22.3.6 Creating a star schema

A star schema is a set of relationships between a fact and several dimension tables. Since the fact table is at the center and many dimension tables are around the perimeter (recall [Figure 22.1](#)) the configuration resembles a star—hence the name.

- Create a new select query called **qryStarSchema**.
- Add the **factSales**, **dimCustomers**, **dimTime**, and **dimProducts** table to the queries.
- Drag the primary keys onto the corresponding foreign keys in the fact table to create query-level relationships, as shown in [Figure 22.8](#).



Since the data warehouse is read-only, there is no need to create relationships in the relationship window or specify referential integrity.

- Project the finest-grained field from each dimension table into the query (specifically: **OrderID**, **CompanyName**, **ProductName**).
- Project the **TotalSale** field.

Since the **TotalSale** field has a numeric data type, you may want to show it in the query formatted as currency.

- Right-click anywhere on the **TotalSale** field, bring up the properties sheet, and enter “Currency” in the **Format** property.



Changing the format of a query field is merely an aesthetic enhancement—the underlying representation of the number remains the same. You may also change the data type of the **TotalSale** field in the **factSales** table—the result is the same.

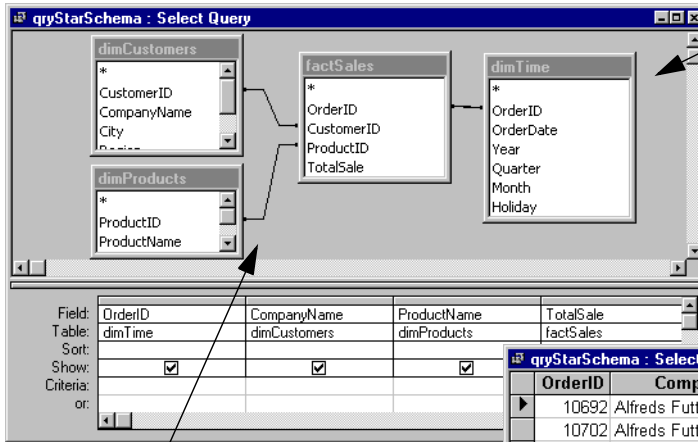
- View the star schema query in data sheet mode, as shown in [Figure 22.8](#).
- Ensure you understand the meaning of each row: the **TotalSale** represents the amount per product per order per customer. This is identical to the granularity of the **OrderDetails** table.

## 22.3.7 Aggregating data using the GroupBy operator

The level of granularity in [Figure 22.8](#) is probably too fine to be useful for many decision making purposes. In this section, you are going to use the “totals” feature in QBE (which is



FIGURE 22.8: Create a star schema to join the fact table with several dimension tables.



1 Create a star schema query based on the fact table and the dimension tables.

3 Note the result: total sale for each order, customer, and product.

2 Link the tables in the usual way (drag the primary key for each dimension onto the corresponding foreign key of the fact table).

OrderID	CompanyName	ProductName	TotalSale
10692	Alfreds Futterkiste	Vegie-spread	\$878.00
10702	Alfreds Futterkiste	Lakkalikööri	\$270.00
10835	Alfreds Futterkiste	Raclette Courdavault	\$825.00
10835	Alfreds Futterkiste	Original Frankfurter grüne Soße	\$20.80
11011	Alfreds Futterkiste	Flötensost	\$430.00
11011	Alfreds Futterkiste	Escargots de Bourgogne	\$503.50
10643	Alfreds Futterkiste	Rössle Sauerkraut	\$513.00
10702	Alfreds Futterkiste	Aniseed Syrup	\$60.00
10643	Alfreds Futterkiste	Spegesild	\$18.00
10952	Alfreds Futterkiste	Rössle Sauerkraut	\$91.20
10643	Alfreds Futterkiste	Chartreuse verte	\$283.50
10952	Alfreds Futterkiste	Grandma's Boysenberry Spread	\$380.00
10759	Ana Trujillo Emparedados y	Mascarpone Fabioli	\$320.00
10625	Ana Trujillo Emparedados y	Camembert Pierrot	\$340.00
10625	Ana Trujillo Emparedados y	Tofu	\$69.75
10308	Ana Trujillo Emparedados y	Gudbrandsdalsost	\$78.80

identical to the **GroupBy** operator in SQL) to aggregate the data.

### 22.3.7.1 Setting up grouping and totals

➔ Switch back to the design view of **qryStarSchema**.



- Select **View** → **Totals** from the main menu. Alternatively, press the sigma ( $\Sigma$ ) button on the toolbar.
- Notice that a “Total” row is added to the query definition grid and that term “Group By” appears in the row for every field.
- Leave the “Group By” entry for all the foreign keys, but change it to “Sum” for the **TotalSale** field, as shown in [Figure 22.9](#).
- Preview the results. You will note no change from the previous result since grouping on unique values of **OrderID**, **CompanyName**, and **ProductName** results in individual order details.

## 22.3.7.2 Different levels of aggregation

There are two ways to change the level of aggregation in a star schema: change the level of granularity for a dimension or drop the dimension from the results set altogether.

- Before switching back to design view, make a mental note of the number of records in the results set (2155 records are shown in [Figure 22.9](#); however, the number you see may vary depending on the version of the NORTHWIND TRADERS database you are using).

- Switch to design view, click on the grey bar above the **OrderID** field in the query definition grid, and press **Delete**.
- Preview the results and make a mental note of the number of records in the results set.

In this modified query, you are grouping on **CustomerID** and **ProductID** and summing extended price. What this means is that the value of **TotalSale** reflects the sum of sales for each unique combination of product and customer regardless of when (i.e., in which order) the products were ordered.

- Return to design mode and delete the **ProductID** field from the query definition grid.
- Preview the results and make a mental note of the number of records in the results set.

In this case, you are grouping on **CustomerID** only. The **TotalSale** field represents the total value of all products in all orders to the customer in question.

- Return to query design mode and delete the **CustomerID** field.
- Preview the results.



FIGURE 22.9: Use the **groupBy** operator to aggregate the **TotalSale** measure across dimension values.

1 Select **View** → **Totals** or click the sigma on the tool bar to show the "total" row.

Field:	Year	City	CategoryNam	TotalSale
Table:	dimTime	dimCustomer	dimProducts	factSales
Total:	Group By	Group By	Group By	Sum
Sort:				
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:				
or:				

OrderID	CompanyName	ProductName	SumOfTotalSa
10248	Vins et alcools Ch	Mozzarella di Giovan	\$174.00
10248	Vins et alcools Ch	Queso Cabrales	\$168.00
10248	Vins et alcools Ch	Singaporean Hokkier	\$98.00
10249	Toms Spezialitäten	Manjimup Dried Appl	\$1,696.00
10249	Toms Spezialitäten	Tofu	\$167.40
10250	Hanari Carnes	Jack's New England	\$77.00
10250	Hanari Carnes	Louisiana Fiery Hot F	\$214.20
10250	Hanari Carnes	Manjimup Dried Appl	\$1,261.40
10251	Victuailles en stoc	Gustaf's Knäckebröd	\$95.76
10251	Victuailles en stoc	Louisiana Fiery Hot F	\$336.00
10251	Victuailles en stoc	Ravioli Angelo	\$222.30
10252	Suprêmes délices	Camembert Pierrot	\$1,088.00
10252	Suprêmes délices	Geitost	\$47.50
10252	Suprêmes délices	Sir Rodney's Marmal	\$2,462.40
10253	Hanari Carnes	Chartreuse verte	\$501.00
10253	Hanari Carnes	Gorgonzola Telino	\$100.00

2 Group on unique combinations of **OrderID**, **CompanyName**, and **ProductName**.

3 Calculate the sum of **TotalSale** for each unique group.

⚠ No aggregation occurs in this example since each order detail has a unique combination of **OrderID** and **ProductName**.

Now, the total sales for all customers, products, and orders is collapsed into a single value. To



get subtotals for particular values of one or more fields, reverse the process by adding them to the query and using the **GroupBy** operator.

- Project **Year**, **City**, and **CategoryName** into the query definition grid.
- Verify the results as shown in [Figure 22.10](#).

FIGURE 22.10: Total sales by year, city, and category.

**1** Project coarser-grained dimension fields into the query definition.

**2** Use the **GroupBy** operator to aggregate the results for each unique combination of year, city, and category.

**3** All values of **TotalSale** for a particular year, city, and category are summed into a single amount.

Year	City	CategoryName	SumOfTotalSale
1994	Aachen	Beverages	\$247.20
1994	Aachen	Dairy Products	\$200.00
1994	Albuquerque	Beverages	\$848.40
1994	Albuquerque	Condiments	\$163.20
1994	Albuquerque	Confections	\$4,033.30
1994	Albuquerque	Dairy Products	\$2,560.00
1994	Albuquerque	Grains/Cereals	\$668.80
1994	Albuquerque	Meat/Poultry	\$1,478.88
1994	Albuquerque	Produce	\$360.00
1994	Albuquerque	Seafood	\$363.20
1994	Anchorage	Beverages	\$388.80
1994	Anchorage	Meat/Poultry	\$2,851.50
1994	Anchorage	Seafood	\$1,435.50
		Beverages	\$403.20
		Condiments	\$324.80
		Meat/Poultry	\$106.20
		Seafood	\$136.00

### 22.3.8 Using aggregation and a star schema to answer a business question

The top part of the query in [Figure 22.10](#) is a

dimensional data model. To test the hypothesis that it is easier for decision makers to create their own queries using dimensional data



models, we can start by considering a business question:

- *What were the total sales of each product in each city in Canada for 1994? Break the results down by quarter and sort the results in descending order of importance.*

- ➔ Create a new query called **qryQuestion**.
- ➔ Repeat the steps in [Section 22.3.6](#) to create a star schema query.
- ➔ Project **City**, **Quarter**, **ProductName**, and **TotalSale** into the query.
- ➔ Ensure the “Totals” feature is on and that you are grouping by unique combinations of **City**, **Quarter**, and **ProductName**.
- ➔ Sum the **TotalSale** field.
- ➔ Set the query to sort on **TotalSale** in descending order.
- ➔ To constrain the results to Canada in 1994, project the **Country** and **Year** fields into the query. For both fields, ensure the “Show” box is unchecked and that the “Group By” entry is replaced by “Where”.
- ➔ View the results as shown in [Figure 22.11](#).



Based on the results of the query, you may re-evaluate the effectiveness of you sales programs in Canadian cities other than Montreal.

Hopefully you agree that a reasonably query-literate individual could construct this type of query and interpret its results. In [Lesson 23](#) you will perform more sophisticated queries and analysis using your new data warehouse.

## 22.4 Discussion

### 22.4.1 Rationale for data warehousing

Data warehousing is based on three basic observations:

1. Normalized data models are difficult for business users to understand and navigate. There are better methods of storing and representing data for decision support applications.
2. The computational load placed on an operational system by a decision support application may be considerable. In other words, it is conceivable that a middle manager tucked away in a cubical somewhere could bring an organization’s primary operational system to its knees with a well-intentioned but poorly designed query. It is better to isolate mission-critical



The screenshot shows a query builder window titled 'qryQuestion : Select Query'. It displays a dimensional data model with four tables: dimCustomers, dimProducts, factSales, and dimTime. dimCustomers and dimProducts are connected to factSales, which is connected to dimTime. Below the model is a table for query criteria:

Field:	City	ProductName	Quarter	TotalSale	Country	Year
Table:	dimCustomers	dimProducts	dimTime	factSales	dimCustomers	dimTime
Total:	Group By	Group By	Group By	Sum	Where	Where
Sort:				Descending		
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Criteria:					"Canada"	1994
or:						

FIGURE 22.11: Answering a business question using a dimensional data model.

1 Select the appropriate dimension fields to answer the question.

2 Constrain the results by using the **Where** operator.

3 Uncheck the **Show** box to ensure that the constraint fields are not shown in the results set.

4 Verify the results of the query.

City	ProductName	Quarter	SumOfTotalSale
Montréal	Alice Mutton	Q4	\$2,074.80
Montréal	Carnarvon Tigers	Q4	\$1,600.00
Montréal	Tarte au sucre	Q4	\$1,103.20
Montréal	Chef Anton's Cajun Seasoning	Q4	\$176.00
Montréal	Zaanse koeken	Q4	\$97.28
Montréal	Singaporean Hokkien Fried Mee	Q4	\$89.60

transaction processing systems from the end-user computing revolution.

- 3. Many aspects of an organization's operations have an implicit time element that is ignored by the transaction processing system. An example of this in the order entry system is the inventory level (**QtyOnHand**) of each product.

Up until the early 1990s, vendors of databases and transaction processing applications were insisting that their transaction processing systems could do both. This is slowly changing as users and vendors adopt a more pragmatic stance.



## 22.4.2 The first law of data warehousing

There is one simple design rule that dominates the design and implementation of data warehouses: *disk space is cheap; time is expensive.*

Time, as it is used here, does not mean computational processing time. It means the time of the decision maker who is waiting for a query to return a result. One of the implications is that hardware vendors sell a lot of expensive gear. Very large arrays of hard drives and parallel processing machines are all the rage in data warehousing.

## 22.4.3 Multiple fact tables

Assume that your firm processes several thousand orders per day and that many of the decision makers in the organization are concerned with monthly measures of performance broken down by region. Although it is certainly possible to get this information by projecting a coarse-grained measure of time (e.g., month) into the query and using the **GroupBy** operator to calculate monthly sales totals, this approach requires a considerable amount of processing.

Given the first law of data warehousing, a better approach involves a straight trade-off between disk space and query performance: create a second fact table with pre-computed

monthly totals. In practice, it is not uncommon to see multiple fact tables containing the same “fact” but with different levels of aggregation precomputed. This is one reason that firms often have multiple terabyte data warehouses.

## 22.5 Application to the assignment

- ➔ Ensure you have implemented all the extraction queries discussed in this lesson.
- ➔ Set the primary key for each dimension table and the fact table.



ACCESS automatically creates indexes for primary keys so you do not need to worry about indexing your tables manually. Although indexes increase the size of your database, the retrieval of records in an indexed table is orders of magnitude faster than in an un-indexed table.