**Chap te r 1**

Foundations of Querying

Exam objectives in this chapter:

■■ Work with Data

■■ Query data by using SELECT statements.

Transact-SQL (T-SQL) is the main language used to manage

and manipulate data in Microsoft SQL Server. This chapter

lays the foundations for querying data by using T-SQL. The

chapter describes the roots of this language, terminology, and

the mindset you need to adopt when writing T-SQL code. It

then moves on to describe one of the most important concepts

you need to know about the language—logical query

processing.

Although this chapter doesn’t directly target specific exam

objectives other than discussing the design of the SELECT

statement, which is the main T-SQL statement used to query data, the rest of the chapters in

this Training Kit do. However, the information in this chapter is critical in order to correctly

understand the rest of the book.

Lessons in this chapter:

■■ Lesson 1: Understanding the Foundations of T-SQL

■■ Lesson 2: Understanding Logical Query Processing

**Before You Begin**

To complete the lessons in this chapter, you must have:

■■ An understanding of basic database concepts.

■■ Experience working with SQL Server Management Studio (SSMS).

■■ Some experience writing T-SQL code.

■■ Access to a SQL Server 2012 instance with the sample database TSQL2012 installed.

(Please see the book’s introduction for details on how to create the sample database.)

m p o r t a n t

***Have you read***

***page xxx?***

It contains valuable

information regarding

the skills you need to

pass the exam

**Lesson 1: Understanding the Foundations of T-SQL**

Many aspects of computing, like programming languages, evolve based on intuition and the

current trend. Without strong foundations, their lifespan can be very short, and if they do

survive, often the changes are very rapid due to changes in trends. T-SQL is different, mainly

because it has strong foundations—mathematics. You don’t need to be a mathematician to

write good SQL (though it certainly doesn’t hurt), but as long as you understand what those

foundations are, and some of their key principles, you will better understand the language

you are dealing with. Without those foundations, you can still write T-SQL code—even code

that runs successfully—but it will be like eating soup with a fork!

After this lesson, you will be able to:

■■ Describe the foundations that T-SQL is based on.

■■ Describe the importance of using T-SQL in a relational way.

■■ Use correct terminology when describing T-SQL–related elements.

Estimated lesson time: 40 minutes

Evolution of T-SQL

As mentioned, unlike many other aspects of computing, T-SQL is based on strong mathematical

foundations. Understanding some of the key principals from those foundations can help

you better understand the language you are dealing with. Then you will think in T-SQL terms

when coding in T-SQL, as opposed to coding with T-SQL while thinking in procedural terms.

Figure 1-1 illustrates the evolution of T-SQL from its core mathematical foundations.

T-SQL

SQL

Relational Model

Set Theory Predicate Logic

**Figure 1-1** Evolution of T-SQL.

T-SQL is the main language used to manage and manipulate data in Microsoft’s main relational

database management system (RDBMS), SQL Server—whether on premises or in the

cloud (Microsoft Windows Azure SQL Database). SQL Server also supports other languages,

like Microsoft Visual C# and Microsoft Visual Basic, but T-SQL is usually the preferred language

for data management and manipulation.

T-SQL is a dialect of standard SQL. SQL is a standard of both the International Organization

for Standards (ISO) and the American National Standards Institute (ANSI). The two standards

for SQL are basically the same. The SQL standard keeps evolving with time. Following is a list

of the major revisions of the standard so far:

■■ SQL-86

■■ SQL-89

■■ SQL-92

■■ SQL:1999

■■ SQL:2003

■■ SQL:2006

■■ SQL:2008

■■ SQL:2011

All leading database vendors, including Microsoft, implement a dialect of SQL as the main

language to manage and manipulate data in their database platforms. Therefore, the core

language elements look the same. However, each vendor decides which features to implement

and which not to. Also, the standard sometimes leaves some aspects as an implementation

choice. Each vendor also usually implements extensions to the standard in cases where

the vendor feels that an important feature isn’t covered by the standard.

Writing in a standard way is considered a best practice. When you do so, your code is

more portable. Your knowledge is more portable, too, because it is easy for you to start

working with new platforms. When the dialect you’re working with supports both a standard

and a nonstandard way to do something, you should always prefer the standard form as your

default choice. You should consider a nonstandard option only when it has some important

benefit to you that is not covered by the standard alternative.

As an example of when to choose the standard form, T-SQL supports two “not equal to”

operators: <> and !=. The former is standard and the latter is not. This case should be a nobrainer:

go for the standard one!

As an example of when the choice of standard or nonstandard depends on the circumstances,

consider the following: T-SQL supports multiple functions that convert a source value

to a target type. Among them are the CAST and CONVERT functions. The former is standard

and the latter isn’t. The nonstandard CONVERT function has a style argument that CAST

doesn’t support. Because CAST is standard, you should consider it your default choice for

conversions. You should consider using CONVERT only when you need to rely on the style

argument.

Yet another example of choosing the standard form is in the termination of T-SQL statements.

According to standard SQL, you should terminate your statements with a semicolon.

T-SQL currently doesn’t make this a requirement for all statements, only in cases where there

would otherwise be ambiguity of code elements, such as in the WITH clause of a common

table expression (CTE). You should still follow the standard and terminate all of your statements

even where it is currently not required.

Standard SQL is based on the *relational model*, which is a mathematical model for data

management and manipulation. The relational model was initially created and proposed by

Edgar F. Codd in 1969. Since then, it has been explained and developed by Chris Date, Hugh

Darwen, and others.

A common misconception is that the name “relational” has to do with relationships

between tables (that is, foreign keys). Actually, the true source for the model’s name is the

mathematical concept *relation*.

A relation in the relational model is what SQL calls a *table*. The two are not synonymous.

You could say that a table is an attempt by SQL to represent a relation (in addition to a relation

variable, but that’s not necessary to get into here). Some might say that it is not a very

successful attempt. Even though SQL is based on the relational model, it deviates from it in a

number of ways. But it’s important to note that as you understand the model’s principles, you

can use SQL—or more precisely, the dialect you are using—in a relational way. More on this,

including a further reading recommendation, is in the next section, “Using T-SQL in a Relational

Way.”

Getting back to a relation, which is what SQL attempts to represent with a table: a relation

has a heading and a body. The heading is a set of attributes (what SQL attempts to represent

with columns), each of a given type. An attribute is identified by name and type name. The

body is a set of tuples (what SQL attempts to represent with rows). Each tuple’s heading is the

heading of the relation. Each value of each tuple’s attribute is of its respective type.

Some of the most important principals to understand about T-SQL stem from the relational

model’s core foundations—set theory and predicate logic.

Remember that the heading of a relation is a set of attributes, and the body a set of tuples.

So what is a set? According to the creator of mathematical set theory, Georg Cantor, a *set* is

described as follows:

*By a “set” we mean any collection M into a whole of definite, distinct objects*

*m (which are called the “elements” of M) of our perception or of our thought.*

—George Cantor, in “Georg Cantor” by Joseph W. Dauben

(Princeton University Press, 1990)

There are a number of very important principles in this definition that, if understood,

should have direct implications on your T-SQL coding practices. For one, notice the term

*whole*. A set should be considered as a whole. This means that you do not interact with the

individual elements of the set, rather with the set as a whole.

Notice the term *distinct*—a set has no duplicates. Codd once remarked on the no duplicates

aspect: ”If something is true, then saying it twice won't make it any truer.“ For example,

the set {a, b, c} is considered equal to the set {a, a, b, c, c, c}.

Another critical aspect of a set doesn’t explicitly appear in the aforementioned definition

by Cantor, but rather is implied—there’s no relevance to the order of elements in a set. In

contrast, a sequence (which is an *ordered* set), for example, does have an order to its elements.

Combining the no duplicates and no relevance to order aspects means that the set

{a, b, c} is equal to the set {b, a, c, c, a, c}.

The other branch of mathematics that the relational model is based on is called predicate

logic. A *predicate* is an expression that when attributed to some object, makes a proposition

either true or false. For example, “salary greater than $50,000” is a predicate. You can evaluate

this predicate for a specific employee, in which case you have a proposition. For example,

suppose that for a particular employee, the salary is $60,000. When you evaluate the proposition

for that employee, you get a true proposition. In other words, a predicate is a parameterized

proposition.

The relational model uses predicates as one of its core elements. You can enforce data

integrity by using predicates. You can filter data by using predicates. You can even use predicates

to define the data model itself. You first identify propositions that need to be stored

in the database. Here’s an example proposition: an order with order ID 10248 was placed on

February 12, 2012 by the customer with ID 7, and handled by the employee with ID 3. You

then create predicates from the propositions by removing the data and keeping the heading.

Remember, the heading is a set of attributes, each identified by name and type name. In this

example, you have orderid INT, orderdate DATE, custid INT, and empid INT.

**Quick Check**

1. What are the mathematical branches that the relational model is based on?

2. What is the difference between T-SQL and SQL?

**Quick Check Answer**

1. Set theory and predicate logic.

2. SQL is standard; T-SQL is the dialect of and extension to SQL that Microsoft

implements in its RDBMS—SQL Server.

Using T-SQL in a Relational Way

As mentioned in the previous section, T-SQL is based on SQL, which in turn is based on the

relational model. However, there are a number of ways in which SQL—and therefore, T-SQL—

deviates from the relational model. But the language gives you enough tools so that if you

understand the relational model, you can use the language in a relational manner, and thus

write more-correct code.

*more info* **SQL and Relational Theory**

For detailed information about the differences between SQL and the relational model and

how to use SQL in a relational way, see *SQL and Relational Theory,* Second Edition by C. J.

Date (O’Reilly Media, 2011). It’s an excellent book that all database practitioners should

read.

Remember that a relation has a heading and a body. The heading is a set of attributes and

the body is a set of tuples. Remember from the definition of a set that a set is supposed to be

considered as a whole. What this translates to in T-SQL is that you’re supposed to write queries

that interact with the tables as a whole. You should try to avoid using iterative constructs

like cursors and loops that iterate through the rows one at a time. You should also try to avoid

thinking in iterative terms because this kind of thinking is what leads to iterative solutions.

For people with a procedural programming background, the natural way to interact with

data (in a file, record set, or data reader) is with iterations. So using cursors and other iterative

constructs in T-SQL is, in a way, an extension to what they already know. However, the correct

way from the relational model’s perspective is not to interact with the rows one at a time;

rather, use relational operations and return a relational result. This, in T-SQL, translates to

writing queries.

Remember also that a set has no duplicates. T-SQL doesn’t always enforce this rule. For example,

you can create a table without a key. In such a case, you are allowed to have duplicate

rows in the table. To follow relational theory, you need to enforce uniqueness in your tables—

for example, by using a primary key or a unique constraint.

Even when the table doesn’t allow duplicate rows, a query against the table can still return

duplicate rows in its result. You'll find further discussion about duplicates in subsequent chapters,

but here is an example for illustration purposes. Consider the following query.

USE TSQL2012;

SELECT country

FROM HR.Employees;

The query is issued against the TSQL2012 sample database. It returns the country attribute

of the employees stored in the HR.Employees table. According to the relational model, a

relational operation against a relation is supposed to return a relation. In this case, this should

translate to returning the set of countries where there are employees, with an emphasis on

set, as in no duplicates. However, T-SQL doesn’t attempt to remove duplicates by default.

Here’s the output of this query.

Country

---------------

USA

USA

USA

USA

UK

UK

UK

USA

UK

In fact, T-SQL is based more on multiset theory than on set theory. A *multiset* (also known

as a bag or a superset) in many respects is similar to a set, but can have duplicates. As mentioned,

the T-SQL language does give you enough tools so that if you want to follow relational

theory, you can do so. For example, the language provides you with a DISTINCT clause

to remove duplicates. Here’s the revised query.

SELECT DISTINCT country

FROM HR.Employees;

Here’s the revised query’s output.

Country

---------------

UK

USA

Another fundamental aspect of a set is that there’s no relevance to the order of the elements.

For this reason, rows in a table have no particular order, conceptually. So when you

issue a query against a table and don’t indicate explicitly that you want to return the rows in

particular presentation order, the result is supposed to be relational. Therefore, you shouldn’t

assume any specific order to the rows in the result, never mind what you know about the

physical representation of the data, for example, when the data is indexed.

As an example, consider the following query.

SELECT empid, lastname

FROM HR.Employees;

When this query was run on one system, it returned the following output, which looks like

it is sorted by the column lastname.

empid lastname

------ -------------

5 Buck

8 Cameron

1 Davis

9 Dolgopyatova

2 Funk

7 King

3 Lew

4 Peled

6 Suurs

Even if the rows were returned in a different order, the result would have still been considered

correct. SQL Server can choose between different physical access methods to process

the query, knowing that it doesn’t need to guarantee the order in the result. For example, SQL

Server could decide to parallelize the query or scan the data in file order (as opposed to index

order).

If you do need to guarantee a specific presentation order to the rows in the result, you

need to add an ORDER BY clause to the query, as follows.

SELECT empid, lastname

FROM HR.Employees

ORDER BY empid;

This time, the result isn’t relational—it’s what standard SQL calls a *cursor*. The order of the

rows in the output is guaranteed based on the empid attribute. Here’s the output of this query.

empid lastname

------ -------------

1 Davis

2 Funk

3 Lew

4 Peled

5 Buck

6 Suurs

7 King

8 Cameron

9 Dolgopyatova

The heading of a relation is a set of attributes that are supposed to be identified by name

and type name. There’s no order to the attributes. Conversely, T-SQL does keep track of

ordinal positions of columns based on their order of appearance in the table definition. When

you issue a query with SELECT \*, you are guaranteed to get the columns in the result based on

definition order. Also, T-SQL allows referring to ordinal positions of columns from the result in

the ORDER BY clause, as follows.

SELECT empid, lastname

FROM HR.Employees

ORDER BY 1;

Beyond the fact that this practice is not relational, think about the potential for error if at

some point you change the SELECT list and forget to change the ORDER BY list accordingly.

Therefore, the recommendation is to always indicate the names of the attributes that you

need to order by.

T-SQL has another deviation from the relational model in that it allows defining result

columns based on an expression without assigning a name to the target column. For example,

the following query is valid in T-SQL.

SELECT empid, firstname + ' ' + lastname

FROM HR.Employees;

This query generates the following output.

empid

------ ------------------

1 Sara Davis

2 Don Funk

3 Judy Lew

4 Yael Peled

5 Sven Buck

6 Paul Suurs

7 Russell King

8 Maria Cameron

9 Zoya Dolgopyatova

But according to the relational model, all attributes must have names. In order for the

query to be relational, you need to assign an alias to the target attribute. You can do so by

using the AS clause, as follows.

SELECT empid, firstname + ' ' + lastname AS fullname

FROM HR.Employees;

Also, T-SQL allows a query to return multiple result columns with the same name. For

example, consider a join between two tables, T1 and T2, both with a column called keycol.

T-SQL allows a SELECT list that looks like the following.

SELECT T1.keycol, T2.keycol ...

For the result to be relational, all attributes must have unique names, so you would need

to use different aliases for the result attributes, as in the following.

SELECT T1.keycol AS key1, T2.keycol AS key2 ...

As for predicates, following the *law of excluded middle* in mathematical logic, a predicate

can evaluate to true or false. In other words, predicates are supposed to use two-valued logic.

However, Codd wanted to reflect the possibility for values to be missing in his model. He

referred to two kinds of missing values: missing but applicable and missing but inapplicable.

Take a mobilephone attribute of an employee as an example. A missing but applicable value

would be if an employee has a mobile phone but did not want to provide this information, for

example, for privacy reasons. A missing but inapplicable value would be when the employee

simply doesn’t have a mobile phone. According to Codd, a language based on his model

should provide two different marks for the two cases. T-SQL—again, based on standard

SQL—implements only one general purpose mark called NULL for any kind of missing value.

This leads to three-valued predicate logic. Namely, when a predicate compares two values,

for example, mobilephone = '(425) 555-0136', if both are present, the result evaluates to

either true or false. But if one of them is NULL, the result evaluates to a third logical value—

unknown.

Note that some believe that a valid relational model should follow two-valued logic, and

strongly object to the concept of NULLs in SQL. But as mentioned, the creator of the relational

model believed in the idea of supporting missing values, and predicates that extend

beyond two-valued logic. What’s important from a perspective of coding with T-SQL is to

realize that if the database you are querying supports NULLs, their treatment is far from being

trivial. That is, you need to carefully understand what happens when NULLs are involved

in the data you’re manipulating with various query constructs, like filtering, sorting, grouping,

joining, or intersecting. Hence, with every piece of code you write with T-SQL, you want to ask

yourself whether NULLs are possible in the data you’re interacting with. If the answer is yes,

you want to make sure that you understand the treatment of NULLs in your query, and ensure

that your tests address treatment of NULLs specifically.

**Quick Check**

1. Name two aspects in which T-SQL deviates from the relational model.

2. Explain how you can address the two items in question 1 and use T-SQL in a

relational way.

**Quick Check Answer**

1. A relation has a body with a distinct set of tuples. A table doesn’t have to have

a key. T-SQL allows referring to ordinal positions of columns in the ORDER BY

clause.

2. Define a key in every table. Refer to attribute names—not their ordinal

positions—in the ORDER BY clause.

Using Correct Terminology

Your use of terminology reflects on your knowledge. Therefore, you should make an effort to

understand and use correct terminology. When discussing T-SQL–related topics, people often

use incorrect terms. And if that’s not enough, even when you do realize what the correct

terms are, you also need to understand the differences between the terms in T-SQL and those

in the relational model.

As an example of incorrect terms in T-SQL, people often use the terms “field” and “record”

to refer to what T-SQL calls “column” and “row,” respectively. Fields and records are physical.

Fields are what you have in user interfaces in client applications, and records are what you

have in files and cursors. Tables are logical, and they have logical rows and columns.

Another example of an incorrect term is referring to “NULL values.” A NULL is a mark for a

missing value—not a value itself. Hence, the correct usage of the term is either “NULL mark”

or just “NULL.”

Besides using correct T-SQL terminology, it’s also important to understand the differences

between T-SQL terms and their relational counterparts. Remember from the previous section

that T-SQL attempts to represent a relation with a table, a tuple with a row, and an attribute

with a column; but the T-SQL concepts and their relational counterparts differ in a number

of ways. As long as you are conscious of those differences, you can, and should, strive to use

T-SQL in a relational way.

**Quick Check**

1. Why are the terms “field” and “record” incorrect when referring to column and

row?

2. Why is the term “NULL value” incorrect?

**Quick Check Answer**

1. Because “field” and “record” describe physical things, whereas columns and

rows are logical elements of a table.

2. Because NULL isn’t a value; rather, it’s a mark for a missing value.

Practice **Using T-SQL in a Relational Way**

In this practice, you exercise your knowledge of using T-SQL in a relational way.

If you encounter a problem completing an exercise, you can install the completed projects

from the Solution folder that is provided with the companion content for this chapter and

lesson.

Exercise 1 Identify Nonrelational Elements in a Query

In this exercise, you are given a query. Your task is to identify the nonrelational elements in

the query.

1. Open SQL Server management Studio (SSMS) and connect to the sample database

TSQL2012. (See the book’s introduction for instructions on how to create the sample

database and how to work with SSMS.)

2. Type the following query in the query window and execute it.

SELECT custid, YEAR(orderdate)

FROM Sales.Orders

ORDER BY 1, 2;

You get the following output shown here in abbreviated form.

custid

----------- -----------

1 2007

1 2007

1 2007

1 2008

1 2008

1 2008

2 2006

2 2007

2 2007

2 2008

...

3. Review the code and its output. The query is supposed to return for each customer

and order year the customer ID (custid) and order year (YEAR(orderdate)). Note that

there’s no presentation ordering requirement from the query. Can you identify what

the nonrelational aspects of the query are?

Answer: The query doesn’t alias the expression YEAR(orderdate), so there’s no name for

the result attribute. The query can return duplicates. The query forces certain presentation

ordering to the result and uses ordinal positions in the ORDER BY clause.

Exercise 2 Make the Nonrelational Query Relational

In this exercise, you work with the query provided in Exercise 1 as your starting point. After

you identify the nonrelational elements in the query, you need to apply the appropriate revisions

to make it relational.

■■ In step 3 of Exercise 1, you identified the nonrelational elements in the last query. Apply

revisions to the query to make it relational.

A number of revisions are required to make the query relational.

■■ Define an attribute name by assigning an alias to the expression YEAR(orderdate).

■■ Add a DISTINCT clause to remove duplicates.

■■ Also, remove the ORDER BY clause to return a relational result.

■■ Even if there was a presentation ordering requirement (not in this case), you should

not use ordinal positions; instead, use attribute names. Your code should look like

the following.

SELECT DISTINCT custid, YEAR(orderdate) AS orderyear

FROM Sales.Orders;

Lesson Summary

■■ T-SQL is based on strong mathematical foundations. It is based on standard SQL,

which in turn is based on the relational model, which in turn is based on set theory and

predicate logic.

■■ It is important to understand the relational model and apply its principals when writing

T-SQL code.

■■ When describing concepts in T-SQL, you should use correct terminology because it

reflects on your knowledge.

Lesson Review

Answer the following questions to test your knowledge of the information in this lesson. You

can find the answers to these questions and explanations of why each answer choice is correct

or incorrect in the “Answers” section at the end of this chapter.

1. Why is it important to use standard SQL code when possible and know what is standard

and what isn’t? (Choose all that apply.)

A. It is not important to code using standard SQL.

B. Standard SQL code is more portable between platforms.

C. Standard SQL code is more efficient.

D. Knowing what standard SQL code is makes your knowledge more portable.

2. Which of the following is not a violation of the relational model?

A. Using ordinal positions for columns

B. Returning duplicate rows

C. Not defining a key in a table

D. Ensuring that all attributes in the result of a query have names

3. What is the relationship between SQL and T-SQL?

A. T-SQL is the standard language and SQL is the dialect in Microsoft SQL Server.

B. SQL is the standard language and T-SQL is the dialect in Microsoft SQL Server.

C. Both SQL and T-SQL are standard languages.

D. Both SQL and T-SQL are dialects in Microsoft SQL Server.