**Chapter-1**

**Lesson -1:**

**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.

**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. 2. Define a key in every table. Refer to attribute names—not their ordinal positions—in the ORDER BY clause.

**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

**Lesson1 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***

*Because attributes are supposed to be identified by name, ensuring that all attributes have names is relational, and hence not a violation of the relational model*

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.

**Lesson 2: understanding Logical Query processing**

**Quick Check**

■What is the difference between the WHERE and HAVING clauses?

**Quick Check Answer**

■ The WHERE clause is evaluated before rows are grouped, and therefore is evaluated per row. The HAVING clause is evaluated after rows are grouped, and therefore is evaluated per group.

**Quick Check**

1. Why are you not allowed to refer to a column alias defined by the SELECT clause in the WHERE clause?

2. Why are you not allowed to refer to a column alias defined by the SELECT clause in the same SELECT clause?

**Quick Check Answer**

1. Because the WHERE clause is logically evaluated in a phase earlier to the one that evaluates the SELECT clause.

2. Because all expressions that appear in the same logical query processing phase are evaluated conceptually at the same point in time.

**Lesson-2 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. Which of the following correctly represents the logical query processing order of the various query clauses?

a. SELECT > FROM > WHERE > GROUP BY > HAVING > ORDER BY

***B. FROM > WHERE > GROUP BY > HAVING > SELECT > ORDER BY***

c. FROM > WHERE > GROUP BY > HAVING > ORDER BY > SELECT

D. SELECT > ORDER BY > FROM > WHERE > GROUP BY > HAVING

2. Which of the following is invalid? (Choose all that apply.)

a. Referring to an attribute that you group by in the WHERE clause

B. Referring to an expression in the GROUP BY clause; for example, GROUP BY YEAR(orderdate)

***c. In a grouped query, referring in the SELECT list to an attribute that is not part of the GROUP BY list and not within an aggregate function***

***D. Referring to an alias defined in the SELECT clause in the HAVING clause***

3. What is true about the result of a query without an ORDER BY clause?

***a. It is relational as long as other relational requirements are met.***

*A query with an ORDER BY clause doesn’t return a relational result. For the result to be relational, the query must satisfy a number of requirements, including the following : the query must not have an ORDER BY clause, all attributes must have names, all attribute names must be unique, and duplicates must not appear in the result.*

B. It cannot have duplicates.

c. The order of the rows in the output is guaranteed to be the same as the insertion order.

D. The order of the rows in the output is guaranteed to be the same as that of the clustered index.

**case scenarios**

In the following case scenarios, you apply what you’ve learned about T-SQL querying. You can find the answers to these questions in the “Answers” section at the end of this chapter.

**Case Scenario 1: Importance of Theory**

You and a colleague on your team get into a discussion about the importance of understanding the theoretical foundations of T-SQL. Your colleague argues that there’s no point in understanding the foundations, and that it’s enough to just learn the technical aspects of T-SQL to be a good developer and to write correct code. Answer the following questions posed to you by your colleague:

1. Can you give an example for an element from set theory that can improve your understanding of T-SQL?

*One of most typical mistakes that T-SQL developers make is to assume that a query without an ORDER BY clause always returns the data in a certain order—for example, clustered index order. But if you understand that in set theory, a set has no particular order to its elements, you know that you shouldn’t make such assumptions. The only way in SQL to guarantee that the rows will be returned in a certain order is to add an ORDER BY clause. That’s just one of many examples for aspects of T-SQL that can be better understood if you understand the foundations of the language*

1. Can you explain why understanding the relational model is important for people who write T-SQL code?

*Even though T-SQL is based on the relational model, it deviates from it in a number of ways. But it gives you enough tools that if you understand the relational model, you can write in a relational way. Following the relational model helps you write code more correctly. Here are some examples :*

*■■ You shouldn’t rely on order of columns or rows.*

*■■ You should always name result columns.*

*■■ You should eliminate duplicates if they are possible in the result of your query*

**Case Scenario 2: Interviewing for a Code Reviewer position**

You are interviewed for a position as a code reviewer to help improve code quality. The organization’s application has queries written by untrained people. The queries have numerous problems, including logical bugs. Your interviewer poses a number of questions and asks for a concise answer of a few sentences to each question. Answer the following questions addressed to you by your interviewer:

1. Is it important to use standard code when possible, and why?

*It is important to use standard SQL code. This way, both the code and people’s knowledge is more portable. Especially in cases where there are both standard and nonstandard forms for a language element, it’s recommended to use the standard form.*

1. We have many queries that use ordinal positions in the ORDER BY clause. Is that a bad practice, and if so why?

*Using ordinal positions in the ORDER BY clause is a bad practice. From a relational perspective, you are supposed to refer to attributes by name, and not by ordinal position. Also, what if the SELECT list is revised in the future and the developer forgets to revise the ORDER BY list accordingly?*

1. If a query doesn’t have an ORDER BY clause, what is the order in which the records are returned?

*When the query doesn’t have an ORDER BY clause, there are no assurances for any particular order in the result. The order should be considered arbitrary. You also notice that the interviewer used the incorrect term record instead of row. You might want to mention something about this, because the interviewer may have done so on purpose to test you.*

1. Would you recommend putting a DISTINCT clause in every query?

*From a pure relational perspective, this actually could be valid, and perhaps even recommended. But from a practical perspective, there is the chance that SQL Server will try to remove duplicates even when there are none, and this will incur extra cost. Therefore, it is recommended that you add the DISTINCT clause only when duplicates are possible in the result and you’re not supposed to return the duplicates.*

**Chapter-2**

**Lesson-1**

**Quick Check**

1. What are the forms of aliasing an attribute in T-SQL?

2. What is an irregular identifier?

**Quick Check Answer**

1. The forms are <expression> AS <alias>, <expression> <alias>, and <alias> = <expression>.

2. An identifier that does not follow the rules for formatting identifiers; for example, it starts with a digit, has an embedded space, or is a reserved T-SQL keyword.

1. What is the importance of the ability to assign attribute aliases in T-SQL? (Choose all that apply.)

a. The ability to assign attribute aliases is just an aesthetic feature.

*B. An expression that is based on a computation results in no attribute name unless you assign one with an alias, and this is not relational.*

c. T-SQL requires all result attributes of a query to have names.

*D. Using attribute aliases, you can assign your own name to a result attribute if you need it to be different than the source attribute name.*

2. What are the mandatory clauses in a SELECT query, according to T-SQL?

a. The FROM and SELECT clauses B. The SELECT and WHERE clauses

*c. The SELECT clause*

D. The FROM and WHERE clauses

3. Which of the following practices are considered bad practices? (Choose all that apply.)

a. Aliasing columns by using the AS clause

B. Aliasing tables by using the AS clause

*c. Not assigning column aliases when the column is a result of a computation*

*D. Using \* in the SELECT list*

**Lesson 2**

The typical options people use to generate surrogate keys are:

■■ the identity column property A property that automatically generates keys in an attribute of a numeric type with a scale of 0; namely, any integer type (TINYINT, SMALLINT, INT, BIGINT) or NUMERIC/DECIMAL with a scale of 0.

■■ the sequence object An independent object in the database from which you can obtain new sequence values. Like identity, it supports any numeric type with a scale of 0. Unlike identity, it’s not tied to a particular column; instead, as mentioned, it is an independent object in the database. You can also request a new value from a sequence object before using it. There are a number of other advantages over identity that will be covered in Chapter 11.

■■ nonsequential GuiDs You can generate nonsequential global unique identifiers to be stored in an attribute of a UNIQUEIDENTIFIER type. You can use the T-SQL function NEWID to generate a new GUID, possibly invoking it with a default expression attached to the column. You can also generate one from anywhere—for example, the client— by using an application programming interface (API) that generates a new GUID. The GUIDs are guaranteed to be unique across space and time.

■■ sequential GuiDs You can generate sequential GUIDs within the machine by using the T-SQL function NEWSEQUENTIALID.

■■ custom solutions If you do not want to use the built-in tools that SQL Server provides to generate keys, you need to develop your own custom solution. The data type for the key then depends on your solution.

**Quick Check**

1. Would you use the type FLOAT to represent a product unit price?

2. What is the difference between NEWID and NEWSEQUENTIALID?

3. Which function returns the current date and time value as a DATETIME2 type?

4. When concatenating character strings, what is the difference between the plus (+) operator and the CONCAT function?

**Quick Check Answer**

1. No, because FLOAT is an approximate data type and cannot represent all values precisely.

2. The NEWID function generates GUID values in random order, whereas the NEWSEQUENTIAL ID function generates GUIDs that increase in a sequential order.

3. The SYSDATETIME function.

4. The + operator by default yields a NULL result on NULL input, whereas the CONCAT function treats NULLs as empty strings

1. Why is it important to use the appropriate type for attributes?

a. Because the type of your attribute enables you to control the formatting of the values

*B. Because the type constrains the values to a certain domain of supported values*

c. Because the type prevents duplicates

D. Because the type prevents NULLs

2. Which of the following functions would you consider using to generate surrogate keys? (Choose all that apply.)

a. NEWID

*correct: The NEWID function creates GUIDs in random order. You would consider it when the size overhead is not a major issue and the ability to generate a unique value across time and space, from anywhere, in random order is a higher priority.*

B. NEWSEQUENTIALID

*correct: The NEWSEQUENTIALID function generates GUIDs in increasing order within the machine. It helps reduce fragmentation and works well when a single session loads the data, and the number of drives is small. However, you should carefully consider an alternative using another key generator, like a sequence object, with a smaller type when possible.*

c. GETDATE

D. CURRENT\_TIMESTAMP

3. What is the difference between the simple CASE expression and the searched CASE expression?

a. The simple CASE expression is used when the database recovery model is simple, and the searched CASE expression is used when it’s full or bulk logged.

*B. The simple CASE expression compares an input expression to multiple possible expressions in the WHEN clauses, and the searched CASE expression uses independent predicates in the WHEN clauses*.

*correct: The difference between the two is that the simple form compares expressions and the searched form uses predicates*

c. The simple CASE expression can be used anywhere in a query, and the searched CASE expression can be used only in the WHERE clause.

D. The simple CASE expression can be used anywhere in a query, and the searched CASE expression can be used only in query filters (ON, WHERE, HAVING).

**case scenarios**

In the following case scenarios, you apply what you’ve learned about the SELECT statement. You can find the answers to these questions in the “Answers” section at the end of this chapter.

**Case Scenario 1: Reviewing the Use of Types**

You are hired as a consultant to help address performance issues in an existing system. The system was developed originally by using SQL Server 2005 and has recently been upgraded to SQL Server 2012. Write rates in the system are fairly low, and their performance is more than adequate. Also, write performance is not a priority. However, read performance is a priority, and currently it is not satisfactory. One of the main goals of the consulting engagement is to provide recommendations that will help improve read performance. You have a meeting with representatives of the customer, and they ask for your recommendations in different potential areas for improvement. One of the areas they inquire about is the use of data types. Your task is to respond to the following customer queries:

1. We have many attributes that represent a date, like order date, invoice date, and so on, and currently we use the DATETIME data type for those. Do you recommend sticking to the existing type or replacing it with another? Any other recommendations along similar lines?

*The DATETIME data type uses 8 bytes of storage. SQL Server 2012 supports the DATE data type, which uses 3 bytes of storage. In all those attributes that represent a date only, it is recommended to switch to using DATE. The lower the storage requirement, the better the reads can perform. As for other recommendations, the general rule “smaller is better, provided that you cover the needs of the attribute in the long run” is suitable for read performance. For example, if you have descriptions of varying lengths stored in a CHAR or NCHAR type, consider switching to VARCHAR or NVARCHAR, respectively. Also, if you’re currently using Unicode types but need to store strings of only one language—say, US English— consider using regular characters instead.*

1. We have our own custom table partitioning solution because we’re using the Standard edition of SQL Server. We use a surrogate key of a UNIQUEIDENTIFIER type with the NEWID function invoked by a default constraint expression as the primary key for the tables. We chose this approach because we do not want keys to conflict across the different tables. This primary key is also our clustered index key. Do you have any recommendations concerning our choice of a key?

*For one, the UNIQUEIDENTIFIER type is large—16 bytes. And because it’s also the clustered index key, it is copied to all nonclustered indexes. Also, due to the random order in which the NEWID function generates values, there’s probably a high level of fragmentation in the index. A different approach to consider (and test!) is switching to an integer type and using the sequence object to generate keys that do not conflict across tables. Due to the reduced size of the type, with the multiplied effect on nonclustered indexes, performance of reads will likely improve. The values will be increasing, and as a result, there will be less fragmentation, which will also likely have a positive effect on reads.*

**Case Scenario 2: Reviewing the Use of Functions**

The same company who hired you to review their use of data types would like you to also review their use of functions. They pose the following question to you:

■■ Our application has worked with SQL Server so far, but due to a recent merger with another company, we need to support other database platforms as well. What can you recommend in terms of use of functions?

*To improve the portability of the code, it’s important to use standard code when possible, and this of course applies more specifically to the use of built-in functions. For example, use COALESCE and not ISNULL, use CURRENT\_TIMESTAMP and not GETDATE, and use CASE and not IIF*

**Chapter-3** **Filtering and Sorting Data**

**Lesson 1: filtering Data with predicates**

**Quick Check**

1. What are the performance benefits in using the WHERE filter?

2. What is the form of a filter predicate that can rely on index ordering called?

**Quick Check Answer**

1. You reduce network traffic by filtering in the database server instead of in the client, and you can potentially use indexes to avoid full scans of the tables involved.

2. A search argument, or SARG, for short.

1. What does the term three-valued logic refer to in T-SQL?

a. The three possible logical result values of a predicate : true, false, and NULL

*B. The three possible logical result values of a predicate : true, false, and unknown*

c. The three possible logical result values of a predicate : 1, 0, and NULL

D. The three possible logical result values of a predicate : -1, 0, and 1

2. Which of the following literals are language-dependent for the DATETIME data type? (Choose all that apply.)

*a. '2012-02-12'*

*The form '2012-02-12' is language-neutral for the data types DATE, DATETIME2, and DATETIMEOFFSET, but language-dependent for DATETIME and SMALLDATETIME.*

*B. '02/12/2012'*

*c. '12/02/2012'*

D. '20120212'

3. Which of the following predicates are search arguments? (Choose all that apply.)

a. DAY(orderdate) = 1

*B. companyname LIKE 'A%'*

*correct: The LIKE predicate is a search argument when the pattern starts with a known prefix.*

c. companyname LIKE '%A%'

D. companyname LIKE '%A'

*e. orderdate > = '20120212' AND orderdate < '20120213'*

*correct: Because no manipulation is applied to the filtered column, the predicate is a search argument*

**Lesson 2: sorting Data**

**Quick Check**

1. How do you guarantee the order of the rows in the result of a query?

2. What is the difference between the result of a query with and one without an ORDER BY clause?

**Quick Check Answer**

1. The only way to do so is by adding an ORDER BY clause.

2. Without an ORDER BY clause, the result is relational (from an ordering perspective); with an ORDER BY clause, the result is conceptually what the standard calls a cursor.

1. When a query doesn’t have an ORDER BY clause, what is the order in which the rows are returned?

*a. Arbitrary order*

*correct: Without an ORDER BY clause, ordering isn’t guaranteed and is said to be arbitrary—it’s optimization-dependent.*

B. Primary key order

c. Clustered index order

D. Insertion order

2. You want result rows to be sorted by orderdate descending, and then by orderid, descending. Which of the following clauses gives you what you want?

a. ORDER BY orderdate, orderid DESC

B. ORDER BY DESC orderdate, DESC orderid

*c. ORDER BY orderdate DESC, orderid DESC*

*correct: The correct syntax is to specify DESC after each expression whose ordering direction needs to be descending*

D. DESC ORDER BY orderdate, orderid

3. You want result rows to be sorted by orderdate ascending, and then by orderid, ascending. Which of the following clauses gives you what you want? (Choose all that apply.)

a. ORDER BY ASC(orderdate, orderid)

*B. ORDER BY orderdate, orderid ASC*

*c. ORDER BY orderdate ASC, orderid ASC*

*D. ORDER BY orderdate, ordered*

**Lesson 3: filtering Data with tOp and Offset-fetch**

**Quick Check**

1. How do you guarantee deterministic results with TOP?

2. What are the benefits of using OFFSET-FETCH over TOP?

**Quick Check Answer**

1. By either returning all ties by using the WITH TIES option or by defining unique ordering to break ties.

2. OFFSET-FETCH is standard and TOP isn’t; also, OFFSET-FETCH supports a skipping capability that TOP doesn’t.

1. You execute a query with a TOP (3) option. Which of the following options most accurately describes how many rows will be returned?

a. Fewer than three rows

*B. Three rows or fewer*

*correct: If there are fewer rows than three in the query result without TOP, the query will return only those rows. If there are three rows or more without TOP, the query will return three rows*

c. Three rows

D. Three rows or more

e. More than three rows

F. Fewer than three, three, or more than three rows

2. You execute a query with TOP (3) WITH TIES and nonunique ordering. Which of the following options most accurately describes how many rows will be returned?

a. Fewer than three rows

B. Three rows or fewer

c. Three rows

D. Three rows or more

e. More than three rows

*F. Fewer than three, three, or more than three rows*

*correct: If there are fewer rows than three in the query result without TOP, the query will return only those rows. If there are at least three rows in the result and no ties with the third, the query will return three rows. If there are more than three rows in the result, as well as ties with the third row, the query will return more than three rows.*

3. Which of the following OFFSET-FETCH options are valid in T-SQL? (Choose all that apply.)

*a. SELECT … ORDER BY orderid OFFSET 25 ROWS*

*correct: T-SQL supports indicating an OFFSET clause without a FETCH clause.*

B. SELECT … ORDER BY orderid FETCH NEXT 25 ROWS ONLY

*c. SELECT … ORDER BY orderid OFFSET 25 ROWS FETCH NEXT 25 ROWS ONLY*

*correct: T-SQL supports indicating both OFFSET and FETCH clauses.*

D. SELECT … <no ORDER BY> OFFSET 0 ROWS FETCH FIRST 25 ROWS ONLY

**Case Scenario 1: Filtering and Sorting performance Recommendations**

You are hired as a consultant to help address query performance problems in a beer factory running SQL Server 2012. You trace a typical workload submitted to the system and observe very slow query run times. You see a lot of network traffic. You see that many queries return all rows to the client and then the client handles the filtering. Queries that do filter data often manipulate the filtered columns. All queries have ORDER BY clauses, and when you inquire about this, you are told that it’s not really needed, but the developers got accustomed to doing so—just in case. You identify a lot of expensive sort operations. The customer is looking for recommendations to improve performance and asks you the following questions:

1. Can anything be done to improve the way filtering is handled?

*For one thing, as much filtering as possible should be done in the database. Doing most of the filtering in the client means that you’re scanning more data, which increases the stress on the storage subsystem, and also that you cause unnecessary network traffic. When you do filter in the databases, for example by using the WHERE clause, you should use search arguments that increase the likelihood for efficient use of indexes. You should try as much as possible to avoid manipulating the filtered columns.*

1. Is there any harm in specifying ORDER BY even when the data doesn’t need to be returned ordered?

*Adding an ORDER BY clause means that SQL Server needs to guarantee returning the rows in the requested order. If there are no existing indexes to support the ordering requirements, SQL Server will have no choice but to sort the data. Sorting is expensive with large sets. So the general recommendation is to avoid adding ORDER BY clauses to queries when there are no ordering requirements. And when you do need to return the rows in a particular order, consider arranging supporting indexes that can prevent SQL Server from needing to perform expensive sort operations.*

1. Any recommendations related to queries with TOP and OFFSET-FETCH?

*The main way to help queries with TOP and OFFSET-FETCH perform well is by arranging indexes to support the ordering elements. This can prevent scanning all data, in addition to sorting.*

**Case Scenario 2: Tutoring a Junior Developer**

You are tutoring a junior developer regarding filtering and sorting data with T-SQL. The developer seems to be confused about certain topics and poses some questions to you. Answer the following to the best of your knowledge:

1. When I try to refer to a column alias that I defined in the SELECT list in the WHERE clause, I get an error. Can you explain why this isn’t allowed and what the workarounds are?

*To be able to understand why you can’t refer to an alias that was defined in the SELECT list in the WHERE clause, you need to understand logical query processing. Even though the keyed-in order of the clauses is SELECT-FROM-WHERE-GROUP BY-HAVING-ORDER BY, the logical query processing order is FROM-WHERE-GROUP BY-HAVING-SELECT-ORDER BY. As you can see, the WHERE clause is evaluated prior to the SELECT clause, and therefore aliases defined in the SELECT clause aren’t visible to the WHERE clause.*

1. Referring to a column alias in the ORDER BY clause seems to be supported. Why is that?

*Logical query processing order explains why the ORDER BY clause can refer to aliases defined in the SELECT clause. That’s because the ORDER BY clause is logically evaluated after the SELECT clause.*

1. Why is it that Microsoft made it mandatory to specify an ORDER BY clause when using OFFSET-FETCH but not when using TOP? Does this mean that only TOP queries can have nondeterministic ordering?

*The ORDER BY clause is mandatory when using OFFSET-FETCH because this clause is standard, and standard SQL decided to make it mandatory. Microsoft simply followed the standard. As for TOP, this feature is proprietary, and when Microsoft designed it, they chose to allow using TOP in a completely nondeterministic manner—without an ORDER BY clause. Note that the fact that OFFSET-FETCH requires an ORDER BY clause doesn’t mean that you must use deterministic ordering. For example, if your ORDER BY list isn’t unique, the ordering isn’t deterministic. And if you want the ordering to be completely nondeterministic, you can specify ORDER BY (SELECT NULL) and then it’s equivalent to not specifying an ORDER BY clause at all.*

**Chapter-4 Combining Sets**

**Lesson 1: using joins**

**Quick Check**

1. What is the difference between the old and new syntax for cross joins?

2. What are the different types of outer joins?

**Quick Check Answer**

1. The new syntax has the CROSS JOIN keywords between the table names and the old syntax has a comma.

2. Left, right, and full.

1. What is the difference between the ON clause and the WHERE clause?

a. The ON clause uses two-valued logic and the WHERE clause uses three-valued logic.

B. The ON clause uses three-valued logic and the WHERE clause uses two-valued logic.

c. In outer joins, the ON clause determines filtering and the WHERE clause determines matching.

*D. In outer joins, the ON clause determines matching and the WHERE clause determines filtering.*

*correct: ON determines matching and WHERE determines filtering*

2. Which keywords can be omitted in the new standard join syntax without changing the meaning of the join? (Choose all that apply.)

a. JOIN B. CROSS

*c. INNER*

*D. OUTER*

3. Which syntax is recommended to use for cross joins and inner joins, and why?

*a. The syntax with the JOIN keyword because it’s consistent with outer join syntax and is less prone to errors.*

B. The syntax with the comma between the table names because it’s consistent with outer join syntax and is less prone to errors.

c. It is recommended to avoid using cross and inner joins.

D. It is recommended to use only lowercase characters and omit default keywords, as in join instead of INNER JOIN because it increases energy consumption

**Lesson 2: using subqueries, table expressions, and the appLy Operator**

**Quick Check**

1. What is the difference between self-contained and correlated subqueries?

2. What is the difference between the APPLY and JOIN operators?

**Quick Check Answer**

1. Self-contained subqueries are independent of the outer query, whereas correlated subqueries have a reference to an element from the table in the outer query.

2. With a JOIN operator, both inputs represent static relations. With APPLY, the left side is a static relation, but the right side can be a table expression with correlations to elements from the left table.

1. What happens when a scalar subquery returns more than one value?

*a. The query fails at run time.*

*correct: The query fails at run time, indicating that more than one value is returned.*

B. The first value is returned.

c. The last value is returned.

D. The result is converted to a NULL.

2. What are the benefits of using a CTE over derived tables? (Choose all that apply.)

a. CTEs are better performing than derived tables.

*B. CTEs don’t nest; the code is more modular, making it easier to follow the logic.*

*correct: If you want to refer to one derived table from another, you need to nest them. With CTEs, you separate those by commas, so the code is more modular and easier to follow.*

*c. Unlike with derived tables, you can refer to multiple instances of the same CTE name, avoiding repetition of code.*

*correct: Because the CTE name is defined before the outer query that uses it, the outer query is allowed to refer to multiple instances of the same CTE name.*

D. Unlike derived tables, CTEs can be used by all statements in the session, and not just the statement defining them.

3. What is the difference between the result of T1 CROSS APPLY T2 and T1 CROSS JOIN T2 (the right table expression isn’t correlated to the left)?

a. CROSS APPLY filters only rows where the values of columns with the same name are equal; CROSS JOIN just returns all combinations.

B. If T1 has rows and T2 doesn’t, CROSS APPLY returns an empty set and CROSS JOIN still returns the rows from T1.

c. If T1 has rows and T2 doesn’t, CROSS APPLY still returns the rows from T1 and CROSS join returns an empty set.

*D. There is no difference.*

*correct: Both return the same result when there’s no correlation because CROSS APPLY applies all rows from T2 to each row from T1.*

**Lesson 3: using set Operators**

**Quick Check**

1. Which set operators does T-SQL support?

2. Name two requirements for the queries involved in a set operator.

**Quick Check Answer**

1. The UNION, INTERSECT, and EXCEPT set operators, as well as the UNION ALL multiset operator.

2. The number of columns in the two queries needs to be the same, and corresponding columns need to have compatible types.

1. Which of the following operators removes duplicates from the result? (Choose all that apply.)

*a. UNION*

B. UNION ALL

*c. INTERSECT*

*D. EXCEPT*

2. In which operator does the order of the input queries matter?

a. UNION

B. UNION ALL

c. INTERSECT

*D. EXCEPT*

3. Which of the following is the equivalent of <query 1> UNION <query 2> INTERSECT <query 3> EXCEPT <query 4>?

a. (<query 1> UNION <query 2>) INTERSECT (<query 3> EXCEPT <query 4>)

*B. <query 1> UNION (<query 2> INTERSECT <query 3>) EXCEPT <query 4>*

*correct: Without the parentheses, the INTERSECT precedes the other operators, and with the specified parentheses, it’s the same.*

c. <query 1> UNION <query 2> INTERSECT (<query 3> EXCEPT <query 4>)

**Case Scenario 1: Code Review**

You are asked to review the code in a system that suffers from both code maintainability problems and performance problems. You come up with the following findings and need to determine what to recommend to the customer:

1. You find many queries that use a number of nesting levels of derived tables, making it very hard to follow the logic. You also find a lot of queries that join multiple derived tables that are based on the same query, and you find that some queries are repeated in a number of places in the code. What can you recommend to the customer to reduce the complexity and improve maintainability?

*To address the nesting complexity of derived tables, in addition to the duplication of derived table code, you can use CTEs. CTEs don’t nest; instead, they are more modular. Also, you can define a CTE once and refer to it multiple times in the outer query. As for queries that are repeated in different places in your code for reusability you can use views and inline table-valued functions. Use the former if you don’t need to pass parameters and the latter if you do.*

1. During your review, you identify a number of cases where cursors are used to access the instances of a certain entity (like customer, employee, shipper) one at a time; next the code invokes a query per each of those instances, storing the result in a temporary table; then the code just returns all the rows from the temporary tables. The customer has both code maintainability and performance problems with the existing code. What can you recommend?

*The customer should evaluate the use of the APPLY operator instead of the cursor plus the query per row. The APPLY operator involves less code and therefore improves the maintainability, and it does not incur the performance hit that cursors usually do.*

1. You identify performance issues with joins. You realize that there are no indexes created explicitly in the system; there are only the ones created by default through primary key and unique constraints. What can you recommend?

*The customer should examine foreign key relationships and evaluate creating indexes on the foreign key columns.*

**Case Scenario 2: Explaining Set Operators**

You are presenting a session about set operators in a conference. At the end of the session, you give the audience an opportunity to ask questions. Answer the following questions presented to you by attendees:

1. In our system, we have a number of views that use a UNION operator to combine disjoint sets from different tables. We see performance problems when querying the views. Do you have any suggestions to try and improve the performance?

*The UNION operator returns distinct rows. When the unified sets are disjoint, there are no duplicates to remove, but the SQL Server Query Optimizer may not realize it. Trying to remove duplicates even when there are none involves extra cost. So when the sets are disjoint, it’s important to use the UNION ALL operator and not UNION. Also, adding CHECK constraints that define the ranges supported by each table can help the optimizer realize that the sets are disjoint. Then, even when using UNION, the optimizer can realize it doesn’t need to remove duplicates*

1. Can you point out the advantages of using set operators like INTERSECT and EXCEPT compared to the use of inner and outer joins?

*Set operators have a number of benefits. They allow simpler code because you don’t explicitly compare the columns from the two inputs like you do with joins. Also, when set operators compare two NULLs, they consider them the same, which is not the case with joins. When this is the desired behavior, it is easier to use set operators. With join, you have to add predicates to get such behavior.*

**Chapter-5 Grouping and Windowing**

**Lesson 1: Writing Grouped Queries**

**Quick Check**

1. What makes a query a grouped query?

2. What are the clauses that you can use to define multiple grouping sets in the same query?

**Quick Check Answer**

1. When you use an aggregate function, a GROUP BY clause, or both.

2. GROUPING SETS, CUBE, and ROLLUP.

1. What is the restriction that grouped queries impose on your expressions?

a. If the query is a grouped query, you must invoke an aggregate function.

B. If the query has an aggregate function, it must have a GROUP BY clause.

c. The elements in the GROUP BY clause must also be specified in the SELECT clause.

D. If you refer to an element from the queried tables in the HAVING, SELECT, or ORDER BY clauses, it must either appear in the GROUP BY list or be contained by an aggregate function.

2. What is the purpose of the GROUPING and GROUPING\_ID functions? (Choose all that apply.)

a. You can use these functions in the GROUP BY clause to group data.

B. You can use these functions to tell whether a NULL in the result represents a placeholder for an element that is not part of the grouping set or an original NULL from the table.

c. You can use these functions to uniquely identify the grouping set that the result row is associated with.

D. These functions can be used to sort data based on grouping set association—that is, first detail, and then aggregates.

3. What is the difference between the COUNT(\*) aggregate function and the COUNT(<expression>) general set function?

a. COUNT(\*) counts rows; COUNT(<expression>) counts rows where <expression> is not NULL.

B. COUNT(\*) counts columns; COUNT(<expression>) counts rows.

c. COUNT(\*) returns a BIGINT; COUNT(<expression>) returns an INT.

D. There’s no difference between the functions.

**Lesson 2: pivoting and unpivoting Data**

**Quick Check**

1. What is the difference between PIVOT and UNPIVOT?

2. What type of language constructs are PIVOT and UNPIVOT implemented as?

**Quick Check Answer**

1. PIVOT rotates data from a state of rows to a state of columns; UNPIVOT rotates the data from columns to rows.

2. PIVOT and UNPIVOT are implemented as table operators.

1. How does the PIVOT operator determine what the grouping element is?

a. It’s the element specified as input to the GROUPING function.

B. It’s determined by elimination—the element(s) from the queried table that were not specified as the spreading or aggregation elements.

c. It’s the element specified in the GROUP BY clause.

D. It’s the primary key.

2. Which of the following are not allowed in the PIVOT operator’s specification? (Choose all that apply.)

a. Specifying a computation as input to the aggregate function

B. Specifying a computation as the spreading element

c. Specifying a subquery in the IN clause

D. Specifying multiple aggregate functions

3. What is the data type of the target values column in the result of an UNPIVOT operator?

a. INT

B. NVARCHAR(128)

c. SQL\_VARIANT

D. The data type of the source columns that you unpivot

**Lesson 3: using Window functions**

**Quick Check**

1. What are the clauses that the different types of window functions support?

2. What do the delimiters UNBOUNDED PRECEDING and UNBOUNDED FOLLOWING

represent?

**Quick Check Answer**

1. Partitioning, ordering, and framing clauses.

2. The beginning and end of the partition, respectively.

**Chapter-6**

**Chapter-7**

**Chapter-8**

**Chapter-9**

**Chapter-10**

**Chapter-11**

**Chapter-12**