

T-SQL supports two “not equal to” operators: <> and !=. The former is standard and the latter is not. 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.

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

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.

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;

Tatt: the order of rows is not guaranteed but order of columns is.

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;

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 ...

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

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(but set should not have duplicates). The query forces certain presentation ordering to the result and uses ordinal positions in the ORDER BY clause( but set should not have any ordering).

SELECT custid, YEAR(orderdate)

FROM Sales.Orders

ORDER BY 1, 2;

Removing the non-relational aspects from the query you get:

SELECT distinct custid, YEAR(orderdate) AS ORderYear

FROM Sales.Orders;

Following are the main query clauses specified in the order that you are supposed to type them (known as “keyed-in order”):

1. SELECT

2. FROM

3. WHERE

4. GROUP BY

5. HAVING

6. ORDER BY

Here is the logical query processing

order of the six main query clauses:

1. FROM

2. WHERE

3. GROUP BY

4. HAVING

5. SELECT

6. ORDER BY

Consider the following query as an example.

SELECT country, YEAR(hiredate) AS yearhired, COUNT(\*) AS numemployees

FROM HR.Employees

WHERE hiredate >= '20030101'

GROUP BY country, YEAR(hiredate)

HAVING COUNT(\*) > 1

ORDER BY country , yearhired DESC;

This query is issued against the HR.Employees table. It filters only employees that were hired in or after the year 2003. It groups the remaining employees by country and the hire year. It keeps only groups with more than one employee. For each qualifying group, the

query returns the hire year and count of employees, sorted by country and hire year, in descending order.