SQL DISTINCT vs. UNIQUE

Duplicate entries imported from a SQL database skew numbers and analytics, so the Structured Query Language (SQL) offers a way to eliminate duplicates so that you only get the exact values that you want. The DISTINCT and UNIQUE SQL commands eliminate duplicates but have different use cases. DISTINCT works on data sets from SELECT queries, and UNIQUE is an attribute set on table columns.

What Is SQL DISTINCT?

Think of DISTINCT as the statement used to ensure that every record returned from a SELECT query is unique. Note that if the data set includes a table column set to have an incremented number, primary key, or other form of unique identifier, then every row will inherently be unique in the data set even with the DISTINCT clause.

For some queries, duplicate rows skew results and data analysis. As an example, a table containing shipment addresses might contain several of the same addresses where customers ordered items several times a month, but you might want to see a list of unique addresses to analyze the number of shipments made for the month. You can use the DISTINCT clause to return only a list of unique shipping addresses from the table and count the number of rows returned to calculate the total number of shipments made for the month.

An example shipment table is below:

id	address	date
345	34 Somewhere Dr	7/23/2023
548	34 Somewhere Dr	7/23/2023
895	98 Nowhere Dr	7/5/2023

In the table above, two shipments were sent to the same address. Without DISTINCT, three rows would be returned using the following SQL statement:

```
SELECT address, date WHERE MONTH(date) = 7;
```

Counting the number of addresses would result in three, but you've only shipped to two distinct addresses during the month on different dates. The DISTINCT clause changes the data set to only unique addresses:

```
SELECT DISTINCT address, date WHERE MONTH(date) = 7;
```

The id column in the example table contains a unique value, so the following query would return all three records even with the DISTINCT clause:

```
SELECT DISTINCT id, address, date WHERE MONTH(date) = 7;
```

Because the id column contains unique values, every row returned in this data set is unique. DISTINCT only works if every column in multiple rows in the data set has the same values.

Pros and Cons of Using SQL DISTINCT

Because the DISTINCT statement forces the database engine to compare rows in a data set, it reduces performance of the query. Using DISTINCT often could be a sign that tables aren't optimized for relational data. Always try to find better ways to return rows if you use DISTINCT often in SQL queries.

Even though DISTINCT reduces performance, it's not considered bad practice for every type of query. Performance monitoring tells you when a query isn't running at optimal speeds, so you can consider refactoring queries with DISTINCT in stored procedures running too slowly. DISTINCT is sometimes unavoidable, so do not consider it bad programming if it's used in some queries.

What Is SQL UNIQUE?

The UNIQUE statement in SQL adds a column constraint to a table. When a column is marked as UNIQUE, only unique values can be stored in the marked column for every record. The UNIQUE statement eliminates the possibility of duplicate records in a specific column. For example, social security numbers are unique values for every US citizen. Adding the UNIQUE constraint on a table's social security number column would restrict input on the column to only unique values.

Using the social security example, the following SQL statement creates a table of patients with the UNIQUE constraint on the SSN column:

```
CREATE TABLE Patients (

Id INTEGER PRIMARY KEY,

FirstName varchar(50),

LastName varchar(50),

SSN varchar(20) UNIQUE,
```

```
);
```

The Id column in the Patients table is marked as UNIQUE, so any application adding records to the table must use a unique value for the SSN column. If an application or user attempts to enter a duplicate value already stored in another record, the database will reject the entry and won't add the new record to the table.

Pros and Cons of Using UNIQUE

The Patients table is one example of the UNIQUE constraint, but mistakes in database table design could cause issues when UNIQUE is used incorrectly. A table design should take into account the front-end applications workflow and accommodate the data that must be stored. UNIQUE columns keep tables normalized so that data stays consistent across a database, but using UNIQUE in the wrong way can cause data integrity issues.

As an example, suppose that you have a table for storing patient data including phone numbers. You might initially think that a phone number only links to a single patient, but multiple patients could have the same household number. A mother and daughter living in the same household could have the same landline number connected to their accounts. When UNIQUE is placed on the phone number column, only one patient can have the household number. In this scenario, data integrity is affected.

SQL DISTINCT vs. UNIQUE

The major difference between the DISTINCT and UNIQUE statements is that DISTINCT works on queries and UNIQUE sets an attribute on a table column. They do not have the same functionality, but they serve the purpose of ensuring data is consistent across tables and queries. Database administrators use the UNIQUE statement to create tables, but any developer working with front-end applications might be asked to create queries using the DISTINCT clause.

The UNIQUE statement does not return a value, but the DISTINCT clause could return millions of rows for a query. DISTINCT reduces performance of a query, so it could cause queries to take too long to return a data set if it's used to filter out duplicates in a large data set. A relational database management system (RDBMS) relies on good table design to keep queries optimized and avoid using DISTINCT, but any RDBMS has support for filtering out duplicate records and placing constraints on columns.

Portability & Best Practice: Why DISTINCT Is the Go-To Choice Today

While DISTINCT and the historic UNIQUE keyword may seem similar at first glance, **modern SQL practice strongly favors DISTINCT for query results** — and for good reason. The DISTINCT keyword is part of the official ANSI SQL standard and is supported across nearly all major database systems, including MySQL, PostgreSQL, SQL Server, Snowflake, BigQuery, and Oracle. In contrast, SELECT UNIQUE is a **legacy Oracle-specific synonym** that isn't recognized in many other systems and is generally discouraged in contemporary SQL code.

Using DISTINCT ensures your queries are **portable, future-proof, and clear to other developers**. It also avoids confusion with the **UNIQUE constraint**, which is a schema-level rule that enforces uniqueness at the database level and has nothing to do with filtering query results.

Conclusion

Most developers and database administrators work with both DISTINCT and UNIQUE statements in SQL. Monitor database performance with queries using DISTINCT, and ensure that columns with the UNIQUE constraint don't interfere with application storage. With the right database design, both SQL statements can be properly used without affecting data integrity and consistency.

Looking for ways to boost your SQL Server performance? Check out Everpure solutions for SQL Server today.

FAQ

What is the difference between DISTINCT and UNIQUE in SQL?
In most modern SQL dialects, DISTINCT is used in the SELECT list to remove duplicate rows from the query result, while UNIQUE is primarily used when defining a constraint on a table column to prevent duplicate values being stored in that column. Some databases accept SELECT UNIQUE as a synonym for SELECT DISTINCT, but in standards-aligned SQL, UNIQUE is about data integrity in the schema and DISTINCT is about de-duplicating rows in a query result.

Are DISTINCT and UNIQUE interchangeable in all SQL databases?
No, they are not fully interchangeable. In many systems SELECT UNIQUE simply behaves like SELECT DISTINCT, but that usage is legacy and not portable across

every database. The UNIQUE keyword is reliably supported as a constraint in CREATE TABLE and ALTER TABLE statements, while DISTINCT is the standard way to remove duplicates in SELECT queries. For portable code, it is safer to treat DISTINCT and UNIQUE as separate features and use each in its intended context.

How do DISTINCT and UNIQUE handle NULL values?
When you use DISTINCT in a SELECT query, most SQL implementations treat all NULL values in a column as equal for the purpose of de-duplication, so you will typically see at most one row with NULL in that column in the result set. For a UNIQUE constraint on a table column, behavior can vary by database. Some databases allow multiple NULL values in a UNIQUE column because NULL is considered "unknown" rather than equal to another NULL, while others restrict it to a single NULL depending on configuration.

Can I use DISTINCT or UNIQUE on multiple columns at once?
Yes. When you write `SELECT DISTINCT col1, col2` the database removes duplicate rows based on the combination of `col1` and `col2` together, not each column individually. A UNIQUE constraint can also cover multiple columns, often called a composite unique key. In that case, the database only enforces uniqueness on the combined set of column values, which is useful when no single column is unique by itself but the pair or group of columns should not repeat.

When should I use DISTINCT instead of GROUP BY to remove duplicates?
Use DISTINCT when your only goal is to eliminate duplicate rows from the result set without performing calculations. Use GROUP BY when you need to aggregate values, for example when you want counts, sums, or averages per group. Both can produce one row per unique combination of columns, but GROUP BY is designed for aggregation while DISTINCT is a simpler de-duplication tool for the selected columns.

Do DISTINCT and UNIQUE impact query performance?
Yes, they can. Removing duplicates requires the database engine to compare rows, which usually involves sorting or building a hash set. On large result sets this adds CPU and memory overhead. A UNIQUE constraint also affects write performance because the database must check for existing values before inserting or updating rows, although appropriate indexing is typically created automatically to make these checks efficient. In practice, good indexing on the columns you filter, join, or de-duplicate usually offsets most of the performance cost.

Is COUNT(DISTINCT …) the same as using DISTINCT in a subquery?
Logically they are similar, because `COUNT(DISTINCT col)` counts how many distinct values exist in that column, which is what you would get from `SELECT DISTINCT col` wrapped in an outer count. However, database optimizers often have specialized implementations for `COUNT(DISTINCT ...)` that can be more efficient than manually writing subqueries. When you only need the number of unique values rather than the values themselves, `COUNT(DISTINCT ...)` is usually the clearer and faster choice.