SQL Standard

SQL Identity Columns

7 months ago • by John Otieno

In this tutorial, we will learn about the identity columns in SQL. We start with the basics and discuss what an identity column is, the characteristics of an identity column, and how to create and use them.

For demonstration purposes, we will use the MySQL database to show how to create and use an identity column. Keep in mind that the definition and implementation of identity columns vary depending on the database engine.

What Is an Identity Column?

Let us start with the basics and discuss what an identity column is. In SQL, an identity column is a special type of column that allows the database engine to automatically generate the unique and sequential numerical values for every record that is added to the table.

Although this is not a requirement, we mainly use an identity column to serve as the primary key of a table to ensure that each row has a unique identifier. You may also hear the identity column under the name “auto increment column” (MySQL) or “serial” (PostgreSQL).

Characteristics of Identity Columns

The following are some properties of identity columns. Some of these characteristics persist across the database engine:

1. Auto Increment – The values of an identity column are automatically generated and incremented by the database engine. Unless specified, the values start from 1 and increase by 1 for each new row added.
2. Unique – Each value of an identity column is unique within the table. This helps to ensure the data integrity, especially when used as the primary key.
3. Sequential – The values are generated sequentially by the database engine. As mentioned, unless explicitly specified, the database starts from 1 and increases by 1

for each new row without a gap or overlap.

4. Immutability – The value of an identity column is immutable. Once assigned, you cannot change it even with an update table clause. This is because modifying the value leads to data inconsistencies which removes the sole purpose of a primary key.

These are some of the characteristics of identity columns in SQL.

Syntax:

The SQL syntax to define an identity column may vary slightly between different database management systems. However, the concept remains the same.

```
column_name data_type GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY[ ( sequence_option
) ]
```

In the given syntax:

1. The “data_type” represent the data type of the identity column. This can be any integer data type.
2. The GENERATED ALWAYS clause tells the database engine to generate the sequential integers for the identity column. This automatically prevents you from manually inserting or updating the value of the identity column.
3. The GENERATED BY DEFAULT tells the database engine that it can automatically generate the value of the identity column but we can manually insert or update its value.

Examples:

Let us look at some examples on how to implement the identity columns in SQL.

Example 1: GENERATED ALWAYS

The first is using the GENERATED ALWAYS clause which leaves the functionality of the identity column to the database engine.

Consider the following example:

```
CREATE TABLE vpc_mapping (
    id INT GENERATED ALWAYS AS IDENTITY,
    vpc_name VARCHAR(255),
    mapping_description TEXT,
    vpc_id VARCHAR(50),
    subnet_id VARCHAR(50),
    status VARCHAR(20) NOT NULL,
    region VARCHAR(50),
    PRIMARY KEY (id)
);
```

In the given example, we create a table called “vpc_mapping” which contains the details about vpc network.

However, pay attention to the ID column. We set the data type of the column as int and use the GENERATED ALWAYS AS IDENTITY to tell the database engine that we wish to create this column as an identity column that is managed by the database engine.

We can then insert the rows into the table without providing the value of the column as follows:

```
INSERT
    INTO
    vpc_mapping (vpc_name,
    mapping_description,
    vpc_id,
    subnet_id,
    status,
    region)
VALUES ('Production VPC',
'Main VPC for production environment',
'vpc-123456',
'subnet-789012',
'Active',
'us-east-1');

INSERT
    INTO
    vpc_mapping (vpc_name,
    mapping_description,
    vpc_id,
    subnet_id,
    status,
    region)
VALUES ('Development VPC',
'Development and testing VPC',
'vpc-789012',
'subnet-345678',
'Inactive',
'us-west-2');

INSERT
    INTO
    vpc_mapping (vpc_name,
    mapping_description,
    vpc_id,
    subnet_id,
    status,
    region)
VALUES ('Backup VPC',
'VPC for data backup and recovery',
'vpc-456789',
'subnet-567890',
'Active',
'eu-west-1');
```

In the previous example insert statements, we make sure not to include the value of the ID column. Doing otherwise causes the database to error.

Example:

```
[Err] ERROR: cannot insert into column " id"
```

DETAIL: **Column** “id” **is** an **identity column** defined **as GENERATED ALWAYS**.

This is because the GENERATED ALWAYS does not allow us to manually specify the value.

Example 2: GENERATED BY DEFAULT

Defining the identity column with GENERATED BY DEFAULT clause allows us to manually insert or update the value of the column.

```
DROP TABLE IF EXISTS vpc_mapping;
CREATE TABLE vpc_mapping (
    id INT GENERATED BY DEFAULT AS IDENTITY,
    vpc_name VARCHAR(255),
    mapping_description TEXT,
    vpc_id VARCHAR(50),
    subnet_id VARCHAR(50),
    status VARCHAR(20) NOT NULL,
    region VARCHAR(50),
    PRIMARY KEY (id)
);
```

Once created, we can specify the value of the column as follows:

```
INSERT
    INTO
    vpc_mapping (id,
vpc_name,
    mapping_description,
    vpc_id,
    subnet_id,
    status,
    region)
VALUES (1, 'Production VPC',
'Main VPC for production environment',
'vpc-123456',
'subnet-789012',
'Active',
'us-east-1');
```

As you can see, we specify the value of the ID column without getting an error.

Conclusion

In this tutorial, we learned about identity columns in SQL. We learned what they are, their characteristics, how they work, and how to use them in SQL database. It is good to reference the docs for your engine to learn more about the identity column. For example: in PostgreSQL, check SERIAL, MySQL, check AUTO_INCREMENT, SQL Server, AUTO INCREMENT.

ABOUT THE AUTHOR

John Otieno

My name is John and am a fellow geek like you. I am passionate about all things computers from Hardware, Operating systems to Programming. My dream is to share my knowledge with the world and help out fellow geeks. Follow my content by subscribing to LinuxHint mailing list

View all posts

RELATED LINUX HINT POSTS

Add a Column to the Table in SQL

Compare Two Tables in SQL

Combine Two Columns in SQL

SQL Having Clause

SQL Subtract

SQL Select All Except

SQL Outer Join

Linux Hint LLC, editor@linuxhint.com
1210 Kelly Park Circle, Morgan Hill, CA 95037