LAB - Unique Constraint

In this lab, you will learn how to use SQL Server `UNIQUE` constraint to ensure the uniqueness of data contained in a column or a group of columns.

SQL Server `UNIQUE` constraints allow you to ensure that the data stored in a column, or a group of columns, is unique among the rows in a table.

The following statement creates a table whose data in the `email` column is unique among the rows in the `hr.persons` table:

```
CREATE SCHEMA hr;
GO

CREATE TABLE hr.persons(
    person_id INT IDENTITY PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE
);
```

In this syntax, you define the `UNIQUE` constraint as a column constraint. You can also define the `UNIQUE` constraint as a table constraint, like this:

```
CREATE TABLE hr.persons(
    person_id INT IDENTITY PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    email VARCHAR(255),
    UNIQUE(email)
);
```

Behind the scenes, SQL Server automatically creates a `UNIQUE` index to enforce the uniqueness of data stored in the columns that participate in the `UNIQUE` constraint. Therefore, if you attempt to insert a duplicate row, SQL Server rejects the change and returns an error message stating that the `UNIQUE` constraint has been violated.

The following statement inserts a new row into the `hr.persons` table:

```
INSERT INTO hr.persons(first_name, last_name, email)
VALUES('John','Doe','j.doe@bike.stores');
```

The statement works as expected. However, the following statement fails due to the duplicate email:

```
INSERT INTO hr.persons(first_name, last_name, email)
VALUES('Jane','Doe','j.doe@bike.stores');
```

SQL Server issued the following error message:

```
Violation of UNIQUE KEY constraint 'UQ__persons__AB6E616417240E4E'. Cannot insert
duplicate key in object 'hr.persons'. The duplicate key value is
(j.doe@bike.stores).
```

If you don't specify a separate name for the `UNIQUE` constraint, SQL Server will automatically generate a name for it. In this example, the constraint name is `UQ__persons__AB6E616417240E4E`, which is not quite readable.

To assign a particular name to a `UNIQUE` constraint, you use the `CONSTRAINT` keyword as follows:

```
CREATE TABLE hr.persons (
    person_id INT IDENTITY PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    email VARCHAR(255),
    CONSTRAINT unique_email UNIQUE(email)
);
```

The following are the benefits of assigning a `UNIQUE` constraint a specific name:

- It easier to classify the error message.
- You can reference the constraint name when you want to modify it.

UNIQUE constraint vs. PRIMARY KEY constraint

Although both `UNIQUE` and `PRIMARY KEY` constraints enforce the uniqueness of data, you should use the `UNIQUE` constraint instead of `PRIMARY KEY` constraint when you want to enforce the uniqueness of a column, or a group of columns, that are not the primary key columns.

UNIQUE constraint allows `NULL`, however a PRIMARY KEY doesn't. Moreover, `UNIQUE` constraint treats the `NULL` as a regular value, therefore, it only allows one `NULL` value per column.

The following statement inserts a row whose value in the `email` column is `NULL`:

```
INSERT INTO hr.persons(first_name, last_name)
VALUES('John','Smith');
```

Now, if you try to insert one more `NULL` into the `email` column, you will get an error:

```
INSERT INTO hr.persons(first_name, last_name)
VALUES('Lily','Bush');
```

Here is the output:

```
Violation of UNIQUE KEY constraint 'UQ__persons__AB6E616417240E4E'. Cannot insert
duplicate key in object 'hr.persons'. The duplicate key value is (<NULL>).
```

`UNIQUE` constraint for a group of columns

To define a `UNIQUE` constraint for a group of columns, you write it as a table constraint with column names separated by commas as follows:

```
CREATE TABLE table_name (
    key_column data_type PRIMARY KEY,
    column1 data_type,
    column2 data_type,
    column3 data_type,
    ...,
    UNIQUE (column1,column2)
);
```

The following example creates a `UNIQUE` constraint that consists of two columns `person_id` and `skill_id`:

```
CREATE TABLE hr.person_skills (
    id INT IDENTITY PRIMARY KEY,
    person_id int,
    skill_id int,
    updated_at DATETIME,
    UNIQUE (person_id, skill_id)
);
```

Add `UNIQUE` constraints to existing columns

When you add a `UNIQUE` constraint to an existing column or a group of columns in a table, SQL Server first examines the existing data in these columns to ensure that all values are unique. If SQL Server finds the duplicate values, then it returns an error and does not add the `UNIQUE` constraint.

The following shows the syntax of adding a `UNIQUE` constraint to a table:

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
UNIQUE(column1, column2,...);
```

Suppose you have the following `hr.persons` table:

```
CREATE TABLE hr.persons (
    person_id INT IDENTITY PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    email VARCHAR(255),
    phone VARCHAR(20),
);
```

The following statement adds a `UNIQUE` constraint to the `email` column:

```
ALTER TABLE hr.persons
ADD CONSTRAINT unique_email UNIQUE(email);
```

Similarly, the following statement adds a `UNIQUE` constraint to the phone column:

```
ALTER TABLE hr.persons
ADD CONSTRAINT unique_phone UNIQUE(phone);
```

Delete `UNIQUE` constraints

To define a `UNIQUE` constraint, you use the `ALTER TABLE DROP CONSTRAINT` statement as follows:

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

The following statement removes the `unique_phone` constraint from the hr.person table:

```
ALTER TABLE hr.persons
DROP CONSTRAINT unique_phone;
```

Modify `UNIQUE` constraints

SQL Server does not have any direct statement to modify a `UNIQUE` constraint, therefore, you need to drop the constraint first and recreate it if you want to change the constraint.

In this lab, you have learned how to use the SQL Server `UNIQUE` constraint to make sure that the data contained in a column or a group of columns is unique.