SQL | Constraints

Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.

The available constraints in SQL are:

- **NOT NULL**: This constraint tells that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more.
- **UNIQUE**: This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.
- **PRIMARY KEY**: A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as primary key.
- **FOREIGN KEY**: A Foreign key is a field which can uniquely identify each row in a another table. And this constraint is used to specify a field as Foreign key.
- **CHECK**: This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.
- **DEFAULT**: This constraint specifies a default value for the column when no value is specified by the user.

How to specify constraints?

We can specify constraints at the time of creating the table using CREATE TABLE statement. We can also specify the constraints after creating a table using ALTER TABLE statement.

Syntax:

Below is the syntax to create constraints using CREATE TABLE statement at the time of creating the table.

```
CREATE TABLE sample_table
(
column1 data_type(size) constraint_name,
column2 data_type(size) constraint_name,
column3 data_type(size) constraint_name,
....
);
```

sample_table: Name of the table to be created.
data_type: Type of data that can be stored in the field.
constraint_name: Name of the constraint. for example- NOT NULL, UNIQUE, PRIMARY KEY etc.

Let us see each of the constraint in detail.

1. NOT NULL –
If we specify a field in a table to be NOT NULL. Then the field will never accept null value. That is, you will be not allowed to insert a new row in the table without specifying any value to this field.
For example, the below query creates a table Student with the fields ID and NAME as NOT NULL. That is, we are bound to specify values for these two fields every time we wish to insert a new row.

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
ADDRESS varchar(20)
);
```

2. UNIQUE –
This constraint helps to uniquely identify each row in the table. i.e. for a particular column, all the rows should have unique values. We can have more than one UNIQUE columns in a table.

For example, the below query creates a table Student where the field ID is specified as UNIQUE. i.e, no two students can have the same ID. Unique constraint in detail.

```
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20)
);
```

3. PRIMARY KEY –

Primary Key is a field which uniquely identifies each row in the table. If a field in a table as primary key, then the field will not be able to contain NULL values as well as all the rows should have unique values for this field. So, in other words we can say that this is combination of NOT NULL and UNIQUE constraints.

A table can have only one field as primary key. Below query will create a table named Student and specifies the field ID as primary key.

```
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20),
PRIMARY KEY(ID)
);
```

4. FOREIGN KEY –

Foreign Key is a field in a table which uniquely identifies each row of a another table. That is, this field points to primary key of another table. This usually creates a kind of link between the tables.

Consider the two tables as shown below:

Orders

O_ID	ORDER_NO	C_ID
1	2253	3
2	3325	3
3	4521	2

4	8532	1

Customers

C_ID	NAME	ADDRESS
1	RAMESH	DELHI
2	SURESH	NOIDA
3	DHARMESH	GURGAON

As we can see clearly that the field C_ID in Orders table is the primary key in Customers table, i.e. it uniquely identifies each row in the Customers table. Therefore, it is a Foreign Key in Orders table.
Syntax:

```
CREATE TABLE Orders
(
O_ID int NOT NULL,
ORDER_NO int NOT NULL,
C_ID int,
PRIMARY KEY (O_ID),
FOREIGN KEY (C_ID) REFERENCES Customers(C_ID)
)
```

(i) CHECK –
Using the CHECK constraint we can specify a condition for a field, which should be satisfied at the time of entering values for this field.
For example, the below query creates a table Student and specifies the condition for the field AGE as (AGE >= 18). That is, the user will not be allowed to enter any record in the table with AGE < 18. Check constraint in detail

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int NOT NULL CHECK (AGE >= 18)
);
```

(ii) DEFAULT –

This constraint is used to provide a default value for the fields. That is, if at the time of entering new records in the table if the user does not specify any value for these fields then the default value will be assigned to them.

For example, the below query will create a table named Student and specify the default value for the field AGE as 18.

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int DEFAULT 18
);
```

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Last Updated : 09 Jun, 2021

124

Similar Reads

1. Difference between Entity constraints, Referential constraints and Semantic constraints
2. SQL | Checking Existing Constraints on a Table using Data Dictionaries
3. SQL Query to Drop Unique Key Constraints Using ALTER Command
4. SQL Query to Add Foreign Key Constraints Using ALTER Command
5. SQL Query to Add Unique key Constraints Using ALTER Command

Primary key constraint in SQL

dhatriganda07

A primary key constraint depicts a key comprising one or more columns that will help uniquely identify every tuple/record in a table.

Properties :

1. No duplicate values are allowed, i.e. Column assigned as primary key should have UNIQUE values only.
2. NO NULL values are present in column with Primary key. Hence there is Mandatory value in column having Primary key.
3. Only one primary key per table exist although Primary key may have multiple columns.
4. No new row can be inserted with the already existing primary key.
5. Classified as : a) Simple primary key that has a Single column 2) Composite primary key has Multiple column.
6. Defined in Create table / Alter table statement.

The primary key can be created in a table using PRIMARY KEY constraint. It can be created at two levels.

1. Column
2. Table.

SQL PRIMARY KEY at Column Level :
If Primary key contains just one column, it should be defined at column level. The following code creates the Primary key “ID” on the person table.

Syntax :

```
Create Table Person
(
Id int NOT NULL PRIMARY KEY,
Name varchar2(20),
Address varchar2(50)
);
```

Here NOT NULL is added to make sure ID should have unique values. SQL will automatically set null values to the primary key if it is not specified.

Example-1 :
To verify the working of Primary key :

```
Insert into Person values(1, 'Ajay', 'Mumbai');
```

Output :

```
1 row created
```

Example-2 :
Let's see if you will insert the same values again.

```
Insert into Person values(1, 'Ajay', 'Mumbai');
```

Output :

```
Error at line 1: unique constraint violated
```

Since we are inserting duplicate values, an error will be thrown since the Primary key "Id" can contain only unique values.

Example-3 :

```
Insert into Person values('', 'Ajay', 'Mumbai');
```

Output :

```
Error at line 1: cannot insert Null into<"user"."Person"."ID">
```

Primary Key cannot contain Null Values so That too will throw an error.

SQL PRIMARY KEY at Table Level :
Whenever the primary key contains multiple columns it has to be specified at Table level.

Syntax:

```
Create Table Person
(Id int NOT NULL,
Name varchar2(20),
Address varchar2(50),
PRIMARY KEY(Id, Name)
);
```

Here, you have only one Primary Key in a table but it consists of Multiple Columns(Id, Name). However, the Following are permissible.

```
Insert into Person values(1, 'Ajay', 'Mumbai');
Insert into Person values(2, 'Ajay', 'Mumbai');
```

Since multiple columns make up Primary Key so both the rows are considered different. SQL permits either of the two values can be duplicated but the combination must be unique.

SQL PRIMARY KEY with ALTER TABLE :
Most of the time, Primary Key is defined during the creation of the table but sometimes the Primary key may not be created in the already existing table. We can however add Primary Key using Alter Statement.

Syntax :

```
Alter Table Person add Primary Key(Id);
```

To add Primary key in multiple columns using the following query.

```
Alter Table Person add Primary Key(Id, Name);
```

It is necessary that the column added as primary key MUST contain unique values or else it will be violated. An id cannot be made Primary key if it contains duplicate values. It violates the basic rule of Primary Key. Altering the table to add Id as a primary key that may contain duplicate values generates an error.

Output :

```
Error at line 1: cannot validate- primary key violated
```

Also, A column added as primary key using alter statement should not have null values. Altering table to add Id as primary key that may contain null values generates an error.

Output :

```
Error at line 1: column contains NULL values; cannot alter to NOT NULL
```

DELETE PRIMARY KEY CONSTRAINT :

To remove Primary Key constraint on table use given SQL as follows.

```
ALTER table Person DROP PRIMARY KEY;
```

Last Updated : 11 Nov, 2020

6

Similar Reads

1. Foreign Key constraint in SQL
2. SQL Query to Drop Foreign Key Constraint Using ALTER Command
3. Difference between Primary key and Unique key
4. Primary key in MS SQL Server
5. SQL Query to Check or Find the Column Name Which Is Primary Key Column
6. SQL Query to Create Table With a Primary Key
7. Change Primary Key Column in SQL Server
8. SQL Query to Remove Primary Key
9. How to Create a Composite Primary Key in SQL Server?
10. SQL | CHECK Constraint

Previous Next

Article Contributed By :

dhatriganda07
dhatriganda07

Vote for difficulty

Current difficulty : Basic

Easy | Normal | Medium | Hard | Expert

Article Tags : DBMS-SQL, SQL

SQL NOT NULL Constraint

sarithaakshaj

Pre-requisite: NULL Values in SQL

The SQL NOT NULL forces particular values or records should not to hold a null value. It is somewhat similar to the primary key condition as the primary key can't have null values in the table although both are completely different things.

In SQL, constraints are some set of rules that are applied to the data type of the specified table, Or we can say that using constraints we can apply limits on the type of data that can be stored in the particular column of the table.

Constraints are typically placed specified along with CREATE statement. By default, a column can hold null values.

Query:

```
CREATE TABLE Emp(
    EmpID INT PRIMARY KEY,
    Name VARCHAR(50),
    Country VARCHAR(50),
    Age int(2),
  Salary int(10)
);
```

Output:

Emp

EmpID	Name	Country	Age	Salary
empty				

If you don't want to have a null column or a null value you need to define constraints like NOT NULL. **NOT NULL** constraints make sure that a column does not hold null values, or in other words, NOT NULL constraints make sure that you cannot insert a new record or update a record without entering a value to the specified column(i.e., NOT NULL column).

It prevents for acceptance of NULL values. It can be applied to column-level constraints.

Syntax:

CREATE TABLE table_Name

(

column1 data_type(size) NOT NULL,

column2 data_type(size) NOT NULL,

....

);

SQL NOT NULL on CREATE a Table

In SQL, we can add NOT NULL constraints while creating a table.

For example, the "EMPID" will not accept NULL values when the EMPLOYEES table is created because NOT NULL constraints are used with these columns.

Query:

```
CREATE TABLE Emp(
    EmpID INT NOT NULL PRIMARY KEY,
    Name VARCHAR (50),
    Country VARCHAR(50),
    Age int(2),
  Salary int(10));
```

Output:

```
Field   Type     Null    Key     Default Extra
EmpID   int      NO      PRI     NULL
Name    varchar(50)      YES             NULL
Country varchar(50)      YES             NULL
Age     int      YES             NULL
Salary  int      YES             NULL
```

SQL NOT NULL on ALTER Table

We can also add a NOT NULL constraint in the existing table using the ALTER statement. For example, if the EMPLOYEES table has already been created then add NOT NULL constraints to the “Name” column using ALTER statements in SQL as follows:

Query:

```
ALTER TABLE Emp modify Name Varchar(50) NOT NULL;
```

Last Updated : 06 Apr, 2023

3

Similar Reads

1. Dangling, Void , Null and Wild Pointers
2. Factorial of a number in PL/SQL
3. SQL HAVING Clause with Examples
4. SQL Drop Table Statement
5. SQL USE Database Statement
6. How to Connect Python with SQL Database?
7. SET ROWCOUNT Function in SQL Server
8. Interface Python with an SQL Database
9. List Methods in Python | Set 1 (in, not in, len(), min(), max()...)
10. Check if a number is Full Fibonacci or not

Related Tutorials

SQL | UNIQUE Constraint

SQL Constraints Unique constraint in SQL is used to check whether the sub-query has duplicate tuples in its result. It returns a boolean value indicating the presence/absence of duplicate tuples. Unique construct returns true only if the subquery has no duplicate tuples, else it returns false.

Important Points:

- Evaluates to true on an empty subquery.
- Returns true only if there are unique tuples present as the output of the sub-query (two tuples are unique if the value of any attribute of the two tuples differs).
- Returns true if the sub-query has two duplicate rows with at least one attribute as NULL.

Syntax:

```
CREATE TABLE table_name (
 column1 datatype UNIQUE,
 column2 datatype,
 ...
);
```


Parameter Explanation:

1. **UNIQUE**: It means that in that particular column, the data should be unique.

We can directly calculate the unique data in the column without using unique data words in SQL but the UNIQUE keyword ease the complexity.Suppose that we have one table in which we have to calculate the unique data in the ID column.

Query:

```
SELECT table.ID
FROM  table
WHERE UNIQUE (SELECT table2.ID
              FROM table2
              WHERE table.ID = table2.ID);
```

Note: During the execution, first the outer query is evaluated to obtain the table. ID. Following this, the inner subquery is processed which produces a new relation that contains the output of the inner query such as the table.ID == table2.ID. If every row in the new relation is unique then unique returns true and the corresponding table.ID is added as a tuple in the output relation produced. However, if every row in the new relationship is not unique then unique evaluates to false and the corresponding table.ID is not added to the output relation. Unique applied to a subquery return false if and only if there are two tuples t1 and t2 such that t1 = t2. It considers t1 and t2 to be two different tuples when unique is applied to a subquery that contains t1 and t2 such that t1 = t2 and at least one of the attributes of these tuples contains a NULL value. The unique predicate in this case evaluates to true. This is because, a NULL value in SQL is treated as an unknown value therefore, two NULL values are considered to be distinct.

The SQL statement without a UNIQUE clause can also be written as:

Query:

```
SELECT table.ID
FROM  table
WHERE 1 <= (SELECT count(table2.ID)
              FROM table2
              WHERE table.ID = table2.ID);
```

Rules for the data in a table can be specified using SQL constraints.

The kinds of data that can be entered into a table are restricted by constraints. This guarantees the reliability and accuracy of the data in the table. The action is stopped if there is a violation between the constraint and the data action column-level or table-level constraints are both possible. Table-level constraints apply to the entire table, while column-level constraints only affect the specified column.

In SQL, the following restrictions are frequently applied:

- **NULL VALUE:** A column cannot have a NULL value by using the NOT NULL flag.

- **UNIQUE KEY:** A unique value makes sure that each value in a column is distinct.
- **PRIMARY KEY:** A NOT NULL and UNIQUE combination. Identifies each row in a table individually.
- **A FOREIGN KEY:** Guards against actions that would break links between tables.
- **CHECK:** Verifies that the values in a column meet a particular requirement.

SQL Unique Constraint on ALTER Table

We can add a unique column in a table using ALTER.Suppose that we have one table with the named instructor and we want to insert one unique column in the instructor.

Syntax:

ALTER TABLE Instructor

ADD UNIQUE(ID);

To DROP a Unique Constraint

Suppose we have to DROP that column, particularly in the table.

Syntax:

ALTER TABLE Instructor

DROP INDEX ID;

Queries:

Find all the instructors that taught at most one course in the year 2017.

Instructor relation:

EmployeeID	Name	CourseID	Year
77505	Alan	SC110	2017
77815	Will	CSE774	2017
85019	Smith	EE457	2017
92701	Sam	PYS504	2017

EmployeeID	Name	CourseID	Year
60215	Harold	HSS103	2016
77505	Alan	BIO775	2017
92701	Sam	ECO980	2017

Without Using Unique Constraint

Query:

```
SELECT I.EMPLOYEEID, I.NAME
FROM Instructor as I
WHERE UNIQUE (SELECT Inst.EMPLOYEEID
              FROM Instructor as Inst
              WHERE I.EMPLOYEEID = Inst.EMPLOYEEID
                          and Inst.YEAR = 2017);
```

By Using Unique Constraint

Query:

```
SELECT Name
FROM Instructor
WHERE Year = 2017
GROUP BY Name
HAVING COUNT(DISTINCT CourseID) = 1;
```

Output:

EmployeeID	Name
77815	Will
85019	Smith

Explanation: In the Instructor relation, only instructors Will and Smith teach a single course during the year 2017. The sub-query corresponding to these instructors contains only a single tuple and therefore the unique clause corresponding to these instructors evaluates to true thereby producing these two instructors in the output relation.

Example 2

Find all the courses in the Computer Science department that has only a single instructor allotted to that course.

Course relation:

CourseID	Name	Department	InstructorID
CSE505	Computer Network	Computer Science	11071
CSE245	Operating System	Computer Science	74505
CSE101	Programming	Computer Science	12715
HSS505	Psychology	Social Science	85017
EE475	Signals & Systems	Electrical	22150
CSE314	DBMS	Computer Science	44704
CSE505	Computer Network	Computer Science	11747
CSE314	DBMS	Computer Science	44715

Without Unique Constraint

Query:

```
SELECT C.COURSEID, C.NAME
FROM Course as C
WHERE UNIQUE (SELECT T.INSTRUCTORID
              FROM Course as T
              WHERE T.COURSEID = C.COURSEID
                          and C.DEPARTMENT = 'Computer Science');
```

By using UNIQUE Constraint

Query:

```
SELECT CourseID
FROM Instructor
```

```
WHERE CourseID LIKE 'CSE%'
GROUP BY CourseID
HAVING COUNT(DISTINCT Name) = 1;
```

Output:

COURSE_ID	Name
CSE245	Operating System
CSE101	Programming

Explanation: In the course relation, the only courses in the computer science department that have a single instructor allotted are Operating Systems and Programming. The unique constraint corresponding to these two courses has only a single tuple consisting of the corresponding instructors. So, the unique clause for these two courses evaluates to true and these courses are displayed in output relation. Other courses in the Course relation either have two or more instructors or they do not belong to the computer science department and therefore, those courses aren't displayed in the output relation.

Frequently Asked Question(FAQ)

Q.1 What is a unique constraint in SQL?

Ans. A UNIQUE constraint ensures that all values in a column are unique. UNIQUE and PRIMARY KEY constraints provide uniqueness guarantees for a column or set of columns. PRIMARY KEY constraints automatically have UNIQUE constraints.

This article is contributed by **Mayank Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 14 Apr, 2023

15

Similar Reads

1. Unique Constraint in MS SQL Server
2. SQL | CHECK Constraint

SQL | DEFAULT Constraint

khushboogoyal499

The **DEFAULT Constraint** is used to fill a column with a default and fixed value. The value will be added to all new records when no other value is provided.

Syntax

*CREATE TABLE tablename (Columnname **DEFAULT** 'defaultvalue');*

Using DEFAULT on CREATE TABLE

To set a DEFAULT value for the "Location" column when the "Geeks" table is created.

Query:

```
CREATE TABLE Geeks (
ID int NOT NULL,
Name varchar(255),
Age int,
Location varchar(255) DEFAULT 'Noida');
INSERT INTO Geeks VALUES (4, 'Mira', 23, 'Delhi');
INSERT INTO Geeks VALUES (5, 'Hema', 27,DEFAULT);
INSERT INTO Geeks VALUES (6, 'Neha', 25, 'Delhi');
INSERT INTO Geeks VALUES (7, 'Khushi', 26,DEFAULT);
Select * from Geeks;
```

Output:

ID	Name	Age	Location
4	Mira	23	Delhi
5	Hema	27	Noida
6	Neha	25	Delhi
7	Khushi	26	Noida

DROP a DEFAULT Constraint

Syntax

ALTER TABLE tablename

ALTER COLUMN columnname

DROP DEFAULT;

Query:

```
ALTER TABLE Geeks
ALTER COLUMN Location
DROP DEFAULT;
```

Let us add 2 new rows in the Geeks table :

Query:

```
INSERT INTO Geeks VALUES (8, 'Komal', 24, 'Delhi');
INSERT INTO Geeks VALUES (9, 'Payal', 26,NULL);
```

Note – Dropping the default constraint will not affect the current data in the table, it will only apply to new rows.

```
Select * from Geeks;
```

Output:

ID	Name	Age	Location
4	Mira	23	Delhi
5	Hema	27	Noida
6	Neha	25	Delhi
7	Khushi	26	Noida
8	Komal	24	Delhi
9	Payal	26	NULL

Foreign Key constraint in SQL

dhatriganda07

Foreign Key is a column that refers to the primary key/unique key of other table. So it demonstrates relationship between tables and act as cross reference among them. Table in which foreign key is defined is called Foreign table/Referencing table. Table that defines primary/unique key and is referenced by foreign key is called primary table/master table/ Referenced Table. It is Defined in Create table/Alter table statement.

For the table that contains Foreign key, it should match the primary key in referenced table for every row. This is called Referential Integrity. Foreign key ensures referential integrity.

Properties :

- Parent that is being referenced has to be unique/Primary Key.
- Child may have duplicates and nulls.
- Parent record can be deleted if no child exists.
- Master table cannot be updated if child exists.
- Must reference PRIMARY KEY in primary table.
- Foreign key column and constraint column should have matching data types.
- Records cannot be inserted in child table if corresponding record in master table do not exist.
- Records of master table cannot be deleted if corresponding records in child table exits.

1. **SQL Foreign key At column level :**
   Syntax –


```
Create table people (no int references person,
                            Fname varchar2(20));
                OR
```

```
Create table people (no int references person(id),
                                Fname varchar2(20));
```

Here Person table should have primary key with type int. If there is single columnar Primary key in table, column name in syntax can be omitted. So both the above syntax works correctly.

To check the constraint,

- If Parent table doesn't have primary key.

  OUTPUT :
  ```
  Error at line 1 : referenced table does not have a primary key.
  ```

- If Parent table has Primary Key of different datatype.

  OUTPUT :
  ```
  Error at line 1 : column type incompatible with referenced column type.
  ```

2. **SQL Foreign key At table level :**
   Syntax –

```
create table people(no varchar2(10),
                    fname varchar2(20),
                    foreign key(no) references person);
                       OR
create table people(no varchar2(10),
                    fname varchar2(20),
                    foreign key(no) references person(id));
```

   Column name of referenced table can be ignored.

3. **Insert Operation in Foreign Key Table :**
   If corresponding value in foreign table doesn't exists, a record in child table cannot be inserted.

   OUTPUT :
   ```
   Error at line 1 : integrity constraint violated - parent key not found.
   ```

4. **Delete Operation in Foreign Key Table :**
   When a record in master table is deleted and corresponding record in child table exists, an error message is displayed and prevents delete operation from going through.

   OUTPUT :
   ```
   Error at line 1 : integrity constraint violated - child record found.
   ```

5. **Foreign Key with ON DELETE CASCADE :**
   The default behavior of foreign key can be changed using ON DELETE CASCADE. When

this option is specified in foreign key definition, if a record is deleted in master table, all corresponding record in detail table will be deleted.

Syntax –

```
create table people(no varchar2(10),
                    fname varchar2(20),
                    foreign key(no)
references person on delete cascade);
```

Now deleting records from person will delete all corresponding records from child table.

```
OUTPUT :
select * from person;
no rows selected

select * from people;
no rows selected
```

6. **Foreign Key with ON DELETE SET NULL :**
   A Foreign key with SET NULL ON DELETE means if record in parent table is deleted, corresponding records in child table will have foreign key fields set to null. Records in child table will not be deleted.
   Syntax –

```
create table people(no varchar2(10),
                 fname varchar2(20),
                    foreign key(no)
references person on delete set null);


OUTPUT :
select * from person;
no rows selected

select * from people;
NO Fname
pqr
```

Notice the field “No” in people table that was referencing Primary key of Person table. On deleting person data, it will set null in child table people. But the record will not be deleted.

Last Updated : 19 Oct, 2020

8

Similar Reads

1. SQL Query to Drop Foreign Key Constraint Using ALTER Command

SQL | CHECK Constraint

SQL Constraints Check Constraint is used to specify a predicate that every tuple must satisfy in a given relation. It limits the values that a column can hold in a relation.

- The predicate in check constraint can hold a sub query.
- Check constraint defined on an attribute restricts the range of values for that attribute.
- If the value being added to an attribute of a tuple violates the check constraint, the check constraint evaluates to false and the corresponding update is aborted.
- Check constraint is generally specified with the CREATE TABLE command in SQL.

Syntax:

```
CREATE TABLE pets(
        ID INT NOT NULL,
        Name VARCHAR(30) NOT NULL,
        Breed VARCHAR(20) NOT NULL,
        Age INT,
        GENDER VARCHAR(9),
        PRIMARY KEY(ID),
        check(GENDER in ('Male', 'Female', 'Unknown'))
        );
```

Note: The check constraint in the above SQL command restricts the GENDER to belong to only the categories specified. If a new tuple is added or an existing tuple in the relation is updated with a GENDER that doesn't belong to any of the three categories mentioned, then the corresponding database update is aborted.

Query

Constraint: Only students with age >= 17 are can enroll themselves in a university. **Schema for student database in university:**

```
CREATE TABLE student(
        StudentID INT NOT NULL,
        Name VARCHAR(30) NOT NULL,
        Age INT NOT NULL,
        GENDER VARCHAR(9),
        PRIMARY KEY(ID),
        check(Age >= 17)
        );
```

Student relation:

StudentID	Name	Age	Gender
1001	Ron	18	Male
1002	Sam	17	Male
1003	Georgia	17	Female
1004	Erik	19	Unknown
1005	Christine	17	Female

Explanation: In the above relation, the age of all students is greater than equal to 17 years, according to the constraint mentioned in the check statement in the schema of the relation. If, however following SQL statement is executed:

```
INSERT INTO student(STUDENTID, NAME, AGE, GENDER)
VALUES (1006, 'Emma', 16, 'Female');
```

There won't be any database update and as the age < 17 years. **Different options to use Check constraint:**

- **With alter:** Check constraint can also be added to an already created relation using the syntax:

```
alter table TABLE_NAME modify COLUMN_NAME check(Predicate);
```

- **Giving variable name to check constraint:**Check constraints can be given a variable name using the syntax:

```
alter table TABLE_NAME add constraint CHECK_CONST check (Predicate);
```

- **Remove check constraint:** Check constraint can be removed from the relation in the database from SQL server using the syntax:

```
alter table TABLE_NAME drop constraint CHECK_CONSTRAINT_NAME;
```

- **Drop check constraint:** Check constraint can be dropped from the relation in the database in MySQL using the syntax:

```
alter table TABLE_NAME drop check CHECK_CONSTRAINT_NAME;
```

View existing constraints on a particular table

If you want to check if a constraint or any constraint exists within the table in mysql then you can use the following command. This command will show a tabular output of all the constraint-related data for the table name you've passed in the statement, in our case we'll use the employee table.

```
SELECT *
FROM information_schema.table_constraints
WHERE table_schema = schema()
AND table_name = 'employee';
```

This article is contributed by **Mayank Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 07 Feb, 2023

18

Similar Reads

1. Check Constraint in MS SQL Server
2. SQL | UNIQUE Constraint