



Primary and Foreign Key Constraints

Primary keys and foreign keys are two types of constraints that can be used to enforce data integrity in SQL Server tables. These are important database objects.



Primary Key Constraints:

A table typically has a column or combination of columns that contain values that uniquely identify each row in the table. This column, or columns, is called the primary key (PK) of the table and enforces the entity integrity of the table. Because primary key constraints guarantee unique data, they're frequently defined on an identity column.

When you specify a primary key constraint for a table, the Database Engine enforces data uniqueness by automatically creating a unique index for the primary key columns. This index also permits fast access to data when the primary key is used in queries. If a primary key constraint is defined on more than one column, values may be duplicated within one column, but each combination of values from all the columns in the primary key constraint definition must be unique.

As shown in the following illustration, the ProductID and VendorID columns in the Purchasing.ProductVendor table form a composite primary key constraint for this table. This makes sure that every row in the ProductVendor table has a unique combination of ProductID and VendorID. This prevents the insertion of duplicate rows.

Primary Key

ProductID	VendorID	AverageLeadTime	StandardPrice	LastReceiptCost
1	1	17	47.8700	50.2635
2	104	19	39.9200	41.9160
7	4	17	54.3100	57.0255
609	7	17	25.7700	27.0585
609	100	19	28.1700	29.5785

ProductVendor table

- A table can contain only one primary key constraint.
- A primary key can't exceed 16 columns and a total key length of 900 bytes.
- The index generated by a primary key constraint can't cause the number of indexes on the table to exceed 999 nonclustered indexes and 1 clustered index.
- If clustered or nonclustered isn't specified for a primary key constraint, clustered is used if there's no clustered index on the table.
- All columns defined within a primary key constraint must be defined as not null. If nullability isn't specified, all columns participating in a primary key constraint have their nullability set to not null.
- If a primary key is defined on a CLR user-defined type column, the implementation of the type must support binary ordering.

Foreign Key Constraints

A foreign key (FK) is a column or combination of columns that is used to establish and enforce a link between the data in two tables to control the data that can be stored in the foreign key table. In a foreign key reference, a link is created between two tables when the column or columns that hold the primary key value for one table are referenced by the column or columns in another table. This column becomes a foreign key in the second table.

For example, the Sales.SalesOrderHeader table has a foreign key link to the Sales.SalesPerson table because there's a logical relationship between sales orders and salespeople. The SalesPersonID column in the SalesOrderHeader table matches the primary key column of the SalesPerson table. The SalesPersonID column in the SalesOrderHeader table is the foreign key to the SalesPerson table. By creating this foreign key relationship, a value for SalesPersonID can't be inserted into the SalesOrderHeader table if it doesn't already exist in the SalesPerson table.

A table can reference a maximum of 253 other tables and columns as foreign keys (outgoing references). SQL Server 2016 (13.x) increases the limit for the number of other tables and columns that can reference columns in a single table (incoming references), from 253 to 10,000. (Requires at least 130 compatibility level.) The increase has the following restrictions:

- Greater than 253 foreign key references are only supported for DELETE DML operations. UPDATE and MERGE operations aren't supported.
- A table with a foreign key reference to itself is still limited to 253 foreign key references.
- Greater than 253 foreign key references aren't currently available for columnstore indexes, memory-optimized tables, Stretch Database, or partitioned foreign key tables.

Creating Tables with Keys:

1. Now when it comes to creating tables with primary keys and foreign keys. I have the SQL script in place that creates three tables. One is a customer table that has the customer ID and the customer's name.
2. You can get this Script with other scripts as well on GitHub.
3. Below is the Script which we are going to use in this lab.
4. First we are going to copy the create statements and create our 3 tables.

```
CREATE TABLE Customer (
    CustomerID varchar(100) NOT NULL,
    CustomerName varchar(1000),
    PRIMARY KEY (CustomerID)
);
```

```
CREATE TABLE Course (
    CourseID varchar(100) NOT NULL,
    CourseName varchar(1000),
```

```
    Price real,  
    PRIMARY KEY (CourseID)  
);
```

```
CREATE TABLE Orders (  
    OrderID varchar(100) NOT NULL,  
    CourseID varchar(100),  
    CustomerID varchar(100),  
    Discountpercent int,  
    PRIMARY KEY (OrderID),  
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),  
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)  
);
```

```
INSERT INTO Customer(CustomerID,CustomerName) VALUES ('C1','UserA');  
INSERT INTO Customer(CustomerID,CustomerName) VALUES ('C2','UserB');  
INSERT INTO Customer(CustomerID,CustomerName) VALUES ('C3','UserC');
```

```
SELECT * FROM Customer;
```

```
INSERT INTO COURSE(CourseID,CourseName,Price) VALUES ('D1','AZ-900',99.99);  
INSERT INTO COURSE(CourseID,CourseName,Price) VALUES ('D2','DP-900',100.99);  
INSERT INTO COURSE(CourseID,CourseName,Price) VALUES ('D3','AZ-104',89.99);
```

```
SELECT * FROM Course;
```

```
INSERT INTO Orders(OrderID,CourseID,CustomerID,Discountpercent) VALUES  
('O1','D2','C1',90);  
INSERT INTO Orders(OrderID,CourseID,CustomerID,Discountpercent) VALUES  
('O2','D1','C2',50);  
INSERT INTO Orders(OrderID,CourseID,CustomerID,Discountpercent) VALUES  
('O3','D3','C3',60);
```

5. Below you can see that we have all of the tables in place.

```

CREATE TABLE Customer (
    CustomerID varchar(100) NOT NULL,
    CustomerName varchar(1000),
    PRIMARY KEY (CustomerID)
);

CREATE TABLE Course (
    CourseID varchar(100) NOT NULL,
    CourseName varchar(1000),
    Price real,
    PRIMARY KEY (CourseID)
);

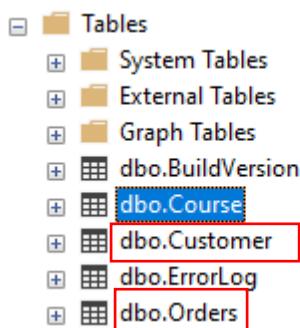
CREATE TABLE Orders (
    OrderID varchar(100) NOT NULL,
    CourseID varchar(100),
    CustomerID varchar(100),
    Discountpercent int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);

```

150 % Messages
Commands completed successfully.
Completion time: 2024-04-15T16:28:18.9926856+05:30

150 % demoazure1010.database.windows.net | sqldadmin (65) | demodb | 00:00:00 | 0 rows
Query executed successfully.

6. Now from the left pane if you select the table section hit the refresh button and wait then we can see our tables.



7. Currently there is no information in these tables. Now if you will try to select and view the data you will find them empty. You can see below nothing is there in our orders table same goes for the other tables too.

SQLQuery1.sql - de...odb (sqladmin (65))*

```
SELECT * FROM Orders
```

150 %

Results Messages

OrderID	CourselD	CustomerID	Discountpercent
---------	----------	------------	-----------------

8. Now we are going to insert some values in these tables. First is Customer table.

SQLQuery1.sql - de...odb (sqladmin (65))*

```
INSERT INTO Customer(CustomerID, CustomerName) VALUES ('C1', 'UserA');
INSERT INTO Customer(CustomerID, CustomerName) VALUES ('C2', 'UserB');
INSERT INTO Customer(CustomerID, CustomerName) VALUES ('C3', 'UserC');
```

150 %

Messages

```
(1 row affected)
(1 row affected)
(1 row affected)
```

Completion time: 2024-04-15T16:34:00.4659989+05:30

9. Now if do a Select statement so, you will be able to see the data.

SQLQuery1.sql - de...odb (sqladmin (65))*

```
SELECT * FROM Customer
```

150 %

Results Messages

	CustomerID	CustomerName
1	C1	UserA
2	C2	UserB
3	C3	UserC

10. Now we are going to do the same for rest of the tables too.

11. Below is the Course table and the output.

SQLQuery1.sql - de...odb (sqladmin (65))*

```
INSERT INTO COURSE(CourseID,CourseName,Price) VALUES ('D1','AZ-900',99.99);
INSERT INTO COURSE(CourseID,CourseName,Price) VALUES ('D2','DP-900',100.99);
INSERT INTO COURSE(CourseID,CourseName,Price) VALUES ('D3','AZ-104',89.99);
```

150 %

Messages

(1 row affected)
(1 row affected)
(1 row affected)

Completion time: 2024-04-15T16:35:57.7448933+05:30

SQLQuery1.sql - de...odb (sqladmin (65))*

```
SELECT * FROM Course;
```

150 %

Results Messages

	CourseID	CourseName	Price
1	D1	AZ-900	99.99
2	D2	DP-900	100.99
3	D3	AZ-104	89.99

12. Below is the Orders table and the output as well.

SQLQuery1.sql - de...odb (sqladmin (65))*

```
INSERT INTO Orders(OrderID, CourseID, CustomerID, Discountpercent) VALUES ('01', 'D2', 'C1', 90);
INSERT INTO Orders(OrderID, CourseID, CustomerID, Discountpercent) VALUES ('02', 'D1', 'C2', 50);
INSERT INTO Orders(OrderID, CourseID, CustomerID, Discountpercent) VALUES ('03', 'D3', 'C3', 60);
```

150 %

Messages

(1 row affected)

(1 row affected)

(1 row affected)

Completion time: 2024-04-15T16:36:38.7417289+05:30

The screenshot shows a SQL Server Management Studio (SSMS) interface. At the top, the title bar reads "SQLQuery1.sql - de...odb (sqladmin (65))". Below the title bar is a query window containing the following SQL code:

```
SELECT * FROM Orders;
```

Below the query window is a results grid. The grid has four columns: OrderID, CourseID, CustomerID, and Discountpercent. There are three rows of data:

	OrderID	CourseID	CustomerID	Discountpercent
1	O1	D2	C1	90
2	O2	D1	C2	50
3	O3	D3	C3	60

😊 JOIN Clause

1. The JOIN clause in SQL is used to combine rows from two or more tables based on a related column between them. It allows you to retrieve data from multiple tables in a single query, enabling you to create more complex queries that involve data from different parts of a database.
2. Below you can see a command that is used to join 2 tables, one is the Product table, and the other is the Sales order details table.
3. Below you can see that we get the output accordingly.

```
SQLQuery1.sql - de...odb [sqladmin (65)]* 4
SELECT SalesLT.Product.ProductID, ListPrice, OrderQty, LineTotal
FROM SalesLT.Product JOIN SalesLT.SalesOrderDetail
ON SalesLT.Product.ProductID=SalesLT.SalesOrderDetail.ProductID
```

Results

ProductID	ListPrice	OrderQty	LineTotal	
1	594.83	1	356.890000	
2	822	594.83	1	356.890000
3	907	106.50	1	63.900000
4	905	364.09	4	873.816000
5	983	769.49	2	923.380000
6	988	564.99	6	406.792000
7	748	1364.50	2	1637.400000
8	999	539.99	1	323.994000
9	926	249.79	1	149.874000
10	743	1349.60	1	809.760000
11	782	2294.99	4	5507.976000
12	918	264.05	2	516.860000
13	780	2319.99	4	5567.976000
14	937	80.99	1	48.594000
15	867	69.99	6	251.964000
16	988	564.99	1	67.798000
17	989	539.99	2	647.988000
18	991	539.99	3	971.982000
19	992	539.99	1	323.994000
20	993	539.99	2	647.988000
21	984	564.99	2	135.597800
22	986	564.99	3	203.396400
23	987	564.99	3	203.396400
24	961	769.49	2	923.380000
25	982	769.49	3	1385.082000
26	783	2294.99	5	6884.970000
27	809	61.92	3	111.456000
28	810	120.27	1	72.162000
29	935	40.49	2	48.588000
30	925	249.79	1	149.874000
31	869	69.99	7	293.958000
32	880	54.99	1	32.994000
33	714	49.99	3	69.982000
34	956	2384.07	3	4291.326000
35	954	2384.07	1	1430.442000
36	712	8.99	10	53.940000
37	877	7.95	10	47.700000
38	924	12.40	4	50.147600

Query executed successfully.

4. Now in the next command we are doing some computation.

```
SQLQuery1.sql - de...odb [sqladmin (65)]* 4
SELECT SalesLT.Product.ProductID, ListPrice, OrderQty, LineTotal, ((ListPrice*OrderQty)-LineTotal) AS 'Discount Amount'
FROM SalesLT.Product JOIN SalesLT.SalesOrderDetail
ON SalesLT.Product.ProductID=SalesLT.SalesOrderDetail.ProductID
```

Results

ProductID	ListPrice	OrderQty	LineTotal	Discount Amount	
1	594.83	1	356.890000	237.932000	
2	822	594.83	1	356.890000	237.932000
3	907	106.50	1	63.900000	42.600000
4	905	364.09	4	873.816000	582.544000
5	983	769.49	2	923.380000	615.920000
6	988	564.99	6	406.792000	2983.147200
7	748	1364.50	2	1637.400000	1091.600000
8	990	539.99	1	323.994000	215.996000
9	926	249.79	1	149.874000	99.916000
10	743	1349.60	1	809.760000	539.840000
11	782	2294.99	4	5507.976000	3671.984000
12	918	264.05	2	516.860000	211.240000
13	780	2319.99	4	5567.976000	3711.984000
14	937	80.99	1	48.594000	32.396000
15	867	69.99	6	251.964000	167.976000
16	985	564.99	1	67.798000	497.191200
17	989	539.99	2	647.988000	431.992000
18	991	539.99	3	971.982000	647.988000
19	992	539.99	1	323.994000	215.996000
20	993	539.99	2	647.988000	431.992000
21	984	564.99	2	135.597800	994.382400
22	986	564.99	3	203.396400	1491.573600
23	987	564.99	3	203.396400	1491.573600
24	981	769.49	2	923.380000	615.920000
25	982	769.49	3	1385.082000	923.380000
26	783	2294.99	5	6884.970000	4589.980000
27	809	61.92	3	111.456000	74.304000
28	810	120.27	1	72.162000	48.108000
29	935	40.49	2	48.588000	32.392000
30	924	12.40	4	50.147600	36.061600

Query executed successfully.

```
SQLQuery1.sql - de...odb (sqladmin (65))* < X

SELECT SalesLT.Product.ProductID, ListPrice, OrderQty, LineTotal, ((ListPrice*OrderQty)-LineTotal) AS ' Discount Amount'
FROM SalesLT.Product JOIN SalesLT.SalesOrderDetail
ON SalesLT.Product.ProductID=SalesLT.SalesOrderDetail.ProductID
WHERE OrderQty>10
```

Results Messages

	ProductID	ListPrice	OrderQty	LineTotal	Discount Amount
1	976	1700.99	25	19136.137500	23388.612500
2	864	63.50	15	497.681250	454.818750
3	883	53.99	11	537.567075	256.322924
4	974	1700.99	13	22112.855308	954.314692
5	715	49.99	17	844.036175	405.793825
6	712	8.99	11	56.209076	42.680924
7	711	34.99	15	274.234125	250.615875
8	708	34.99	11	218.771476	166.118524
9	870	4.99	11	31.199476	23.690524
10	965	742.35	11	8165.776500	3524.380860
11	957	2384.07	12	28608.445658	12347.575344
12	864	63.50	23	1458.562500	697.388750
13	708	34.99	12	238.659792	181.220208
14	883	53.99	16	863.064000	412.483600
15	884	53.99	15	809.846250	396.703375
16	864	63.50	23	1458.562500	697.388750
17	877	7.95	17	138.142250	64.534125
18	870	4.99	11	31.199476	23.690524
19	708	34.99	12	238.659792	181.220208
20	867	69.99	13	902.898892	392.698892
21	875	8.99	11	56.209076	42.680924
22	875	8.99	14	62.886024	54.321176
23	875	8.99	13	68.428908	50.441092
24	867	69.99	14	556.952424	422.907576

Query executed successfully.

demoazure1010.database.windows.net | sqladmin (65) | demodb | 00:00:00 | 24 rows