# LAB - Foreign Key Constraint

In this lab, you will learn how to use SQL Server foreign key constraint to enforce a link between the data in two tables.

Consider the following `vendor_groups` and `vendors` tables:

```
CREATE TABLE procurement.vendor_groups (
    group_id INT IDENTITY PRIMARY KEY,
    group_name VARCHAR (100) NOT NULL
);

CREATE TABLE procurement.vendors (
        vendor_id INT IDENTITY PRIMARY KEY,
        vendor_name VARCHAR(100) NOT NULL,
        group_id INT NOT NULL,
);
```

Each vendor belongs to a vendor group and each vendor group may have zero or more vendors. The relationship between the `vendor_groups` and `vendors` tables is one-to-many.

For each row in the `vendors` table, you can always find a corresponding row in the `vendor_groups` table.

However, with the current tables setup, you can insert a row into the `vendors` table without a corresponding row in the `vendor_groups` table. Similarly, you can also delete a row in the `vendor_groups` table without updating or deleting the corresponding rows in the `vendors` table that results in orphaned rows in `vendors` table.

To enforce the link between data in the `vendor_groups` and `vendors` tables, you need to establish a foreign key relationship in the `vendors` table.

A foreign key is a column or a group of columns in one table that uniquely identifies a row of another table (or the same table in case of self-reference).

To create a foreign key, you use the `FOREIGN KEY` constraint.

The following statements drop the `vendors` table and recreates it with a `FOREIGN KEY` constraint:

```
DROP TABLE vendors;

CREATE TABLE procurement.vendors (
        vendor_id INT IDENTITY PRIMARY KEY,
        vendor_name VARCHAR(100) NOT NULL,
        group_id INT NOT NULL,
        CONSTRAINT fk_group FOREIGN KEY (group_id)
        REFERENCES procurement.vendor_groups(group_id)
);
```

The `vendor_groups` table now is called the **parent table** that is the table to which the foreign key constraint references. The `vendors` table is called the **child table** that is the table to which the foreign key constraint is applied.

In the statement above, the following clause creates a `FOREIGN KEY` constraint named `fk_group` that links the `group_id` in the `vendors` table to the `group_id` in the `vendor_groups` table:

```
CONSTRAINT fk_group FOREIGN KEY (group_id) REFERENCES
procurement.vendor_groups(group_id)
```

## Syntax

The general syntax for creating a `FOREIGN KEY` constraint is as follows:

```
CONSTRAINT fk_constraint_name
FOREIGN KEY (column_1, column2,...)
REFERENCES parent_table_name(column1,column2,..)
```

Let's examine this syntax in detail.

First, specify the `FOREIGN KEY` constraint name after the `CONSTRAINT` keyword. The constraint name is optional therefore it is possible to define a `FOREIGN KEY` constraint as follows:

```
FOREIGN KEY (column_1, column2,...)
REFERENCES parent_table_name(column1,column2,..)
Code language: SQL (Structured Query Language) (sql)
```

In this case, SQL Server will automatically generate a name for the `FOREIGN KEY` constraint.

Second, specify a list of comma-separated foreign key columns enclosed by parentheses after the `FOREIGN KEY` keyword.

Third, specify the name of the parent table to which the foreign key references and a list of comma-separated columns that has a link with the column in the child table.

## Example

First, insert some rows into the `vendor_groups` table:

```
INSERT INTO procurement.vendor_groups(group_name)
VALUES('Third-Party Vendors'),
      ('Interco Vendors'),
      ('One-time Vendors');
```

Second, insert a new vendor with a vendor group into the `vendors` table:

```
INSERT INTO procurement.vendors(vendor_name, group_id)
VALUES('ABC Corp',1);
```

The statement worked as expected.

Third, try to insert a new vendor whose vendor group does not exist in the `vendor_groups` table:

```
INSERT INTO procurement.vendors(vendor_name, group_id)
VALUES('XYZ Corp',4);
```

SQL Server issued the following error:

```
The INSERT statement conflicted with the FOREIGN KEY constraint "fk_group". The
conflict occurred in database "BikeStores", table "procurement.vendor_groups",
column 'group_id'.
```

In this example, because of the `FOREIGN KEY` constraint, SQL Server rejected the insert and issued an error.

## Referential actions

The foreign key constraint ensures referential integrity. It means that you can only insert a row into the child table if there is a corresponding row in the parent table.

Besides, the foreign key constraint allows you to define the referential actions when the row in the parent table is updated or deleted as follows:

```
FOREIGN KEY (foreign_key_columns)
    REFERENCES parent_table(parent_key_columns)
    ON UPDATE action
    ON DELETE action;
```

The `ON UPDATE` and `ON DELETE` specify which action will execute when a row in the parent table is updated or deleted. The following are permitted actions: `NO ACTION`, `CASCADE`, `SET NULL`, and `SET DEFAULT`

## Delete actions of rows in the parent table

If you delete one or more rows in the parent table, you can set one of the following actions:

- `ON DELETE NO ACTION`: SQL Server raises an error and rolls back the delete action on the row in the parent table.
- `ON DELETE CASCADE`: SQL Server deletes the rows in the child table that is corresponding to the row deleted from the parent table.
- `ON DELETE SET NULL`: SQL Server sets the rows in the child table to `NULL` if the corresponding rows in the parent table are deleted. To execute this action, the foreign key columns must be nullable in the child table.
- `ON DELETE SET DEFAULT` SQL Server sets the rows in the child table to their default values if the corresponding rows in the parent table are deleted. To execute this action, the foreign key columns must have default definitions in the child table. Note that a nullable column has a default value of `NULL` if no default value specified.

By default, SQL Server applies `ON DELETE NO ACTION` if you don't explicitly specify any action.

## Update action of rows in the parent table

If you update one or more rows in the parent table, you can set one of the following actions:

- `ON UPDATE NO ACTION`: SQL Server raises an error and rolls back the update action on the row in the parent table.
- `ON UPDATE CASCADE`: SQL Server updates the corresponding rows in the child table when the rows in the parent table are updated.
- `ON UPDATE SET NULL`: SQL Server sets the rows in the child table to `NULL` when the corresponding row in the parent table is updated. Note that the foreign key columns must be nullable for this action to execute.
- `ON UPDATE SET DEFAULT`: SQL Server sets the default values for the rows in the child table that have the corresponding rows in the parent table updated.

In this lab, you have learned how to use the SQL Server foreign key constraint to enforce the referential integrity between tables.