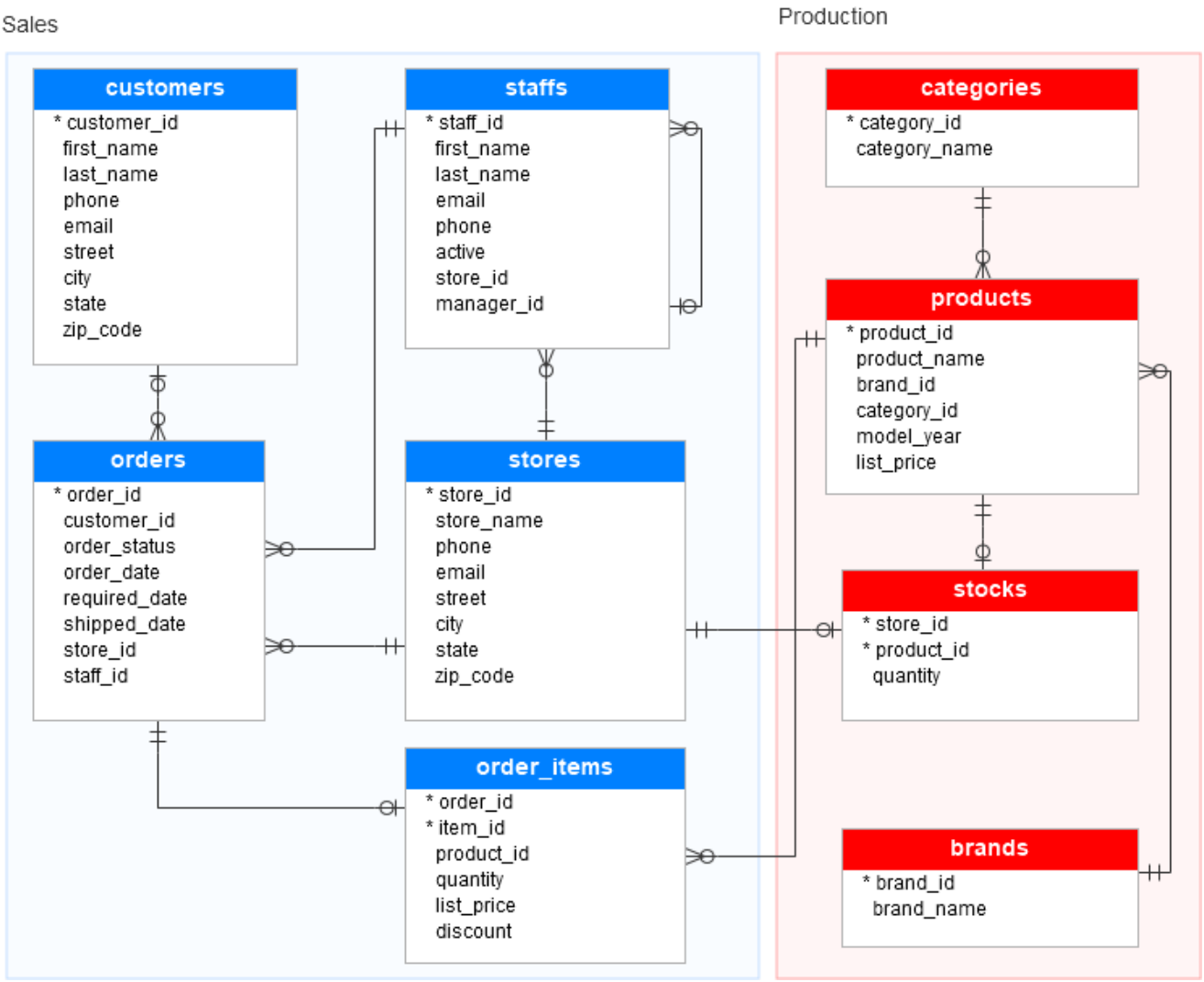
SQL SERVER SELF JOIN

The following illustrates the **BikeStores** database diagram:



As you can see from the diagram, the **BikeStores** sample database has two schemas sales and production, and these schemas have nine tables.

Database Tables

Table **sales.stores**

The **sales.stores** table includes the store's information. Each store has a store name, contact information such as phone and email, and an address including street, city, state, and zip code.

SQL SERVER SELF JOIN

```
CREATE TABLE sales.stores

(

        store_id INT IDENTITY (1, 1) PRIMARY KEY,

        store_name VARCHAR (255) NOT NULL,

        phone VARCHAR (25),

        email VARCHAR (255),

        street VARCHAR (255),

        city VARCHAR (255),

        state VARCHAR (10),

        zip_code VARCHAR (5)

);
```

Table **sales.staffs**

The **sales.staffs** table stores the essential information of staffs including first name, last name. It also contains the communication information such as email and phone.

A staff works at a store specified by the value in the **store_id** column. A store can have one or more staffs.

A staff reports to a store manager specified by the value in the **manager_id** column. If the value in the **manager_id** is null, then the staff is the top manager.

If a staff no longer works for any stores, the value in the active column is set to zero.

```
CREATE TABLE sales.staffs (

        staff_id INT IDENTITY (1, 1) PRIMARY KEY,

        first_name VARCHAR (50) NOT NULL,

        last_name VARCHAR (50) NOT NULL,

        email VARCHAR (255) NOT NULL UNIQUE,

        phone VARCHAR (25),

        active tinyint NOT NULL,
```

```
    store_id INT NOT NULL,
    manager_id INT,
    FOREIGN KEY (store_id)
REFERENCES sales.stores (store_id)
ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (manager_id)
REFERENCES sales.staffs (staff_id)
ON DELETE NO ACTION ON UPDATE NO ACTION
);
```

Table **production.categories**

The **production.categories** table stores the bike's categories such as children bicycles, comfort bicycles, and electric bikes.

```
CREATE TABLE production.categories (
    category_id INT IDENTITY (1, 1) PRIMARY KEY,
    category_name VARCHAR (255) NOT NULL
);
```

Table **production.brands**

The **production.brands** table stores the brand's information of bikes, for example, Electra, Haro, and Heller.

```
CREATE TABLE production.brands (
    brand_id INT IDENTITY (1, 1) PRIMARY KEY,
    brand_name VARCHAR (255) NOT NULL
);
```

SQL SERVER SELF JOIN

Table **production.products**

The **production.products** table stores the product's information such as name, brand, category, model year, and list price.

Each product belongs to a brand specified by the **brand_id** column. Hence, a brand may have zero or many products.

Each product also belongs a category specified by the **category_id** column. Also, each category may have zero or many products.

```
CREATE TABLE production.products (
        product_id INT IDENTITY (1, 1) PRIMARY KEY,
        product_name VARCHAR (255) NOT NULL,
        brand_id INT NOT NULL,
        category_id INT NOT NULL,
        model_year SMALLINT NOT NULL,
        list_price DECIMAL (10, 2) NOT NULL,
        FOREIGN KEY (category_id)
    REFERENCES production.categories (category_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (brand_id)
    REFERENCES production.brands (brand_id)
    ON DELETE CASCADE ON UPDATE CASCADE
);
```

Table **sales.customers**

The **sales.customers** table stores customer's information including first name, last name, phone, email, street, city, state and zip code.

```
CREATE TABLE sales.customers (
        customer_id INT IDENTITY (1, 1) PRIMARY KEY,
        first_name VARCHAR (255) NOT NULL,
        last_name VARCHAR (255) NOT NULL,
        phone VARCHAR (25),
```

```
    email VARCHAR (255) NOT NULL,
    street VARCHAR (255),
    city VARCHAR (50),
    state VARCHAR (25),
    zip_code VARCHAR (5)
);
```

Table **sales.orders**

The **sales.orders** table stores the sales order's header information including customer, order status, order date, required date, shipped date.

It also stores the information on where the sales transaction was created (store) and who created it (staff).

Each sales order has a row in the **sales_orders** table. A sales order has one or many line items stored in the **sales.order_items** table.

```
CREATE TABLE sales.orders (
    order_id INT IDENTITY (1, 1) PRIMARY KEY,
    customer_id INT,
    order_status tinyint NOT NULL,
    -- Order status: 1 = Pending; 2 = Processing; 3 = Rejected; 4 = Completed
    order_date DATE NOT NULL,
    required_date DATE NOT NULL,
    shipped_date DATE,
    store_id INT NOT NULL,
    staff_id INT NOT NULL,
    FOREIGN KEY (customer_id)
REFERENCES sales.customers (customer_id)
ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (store_id)
REFERENCES sales.stores (store_id)
ON DELETE CASCADE ON UPDATE CASCADE,
```

```
    FOREIGN KEY (staff_id)
REFERENCES sales.staffs (staff_id)
ON DELETE NO ACTION ON UPDATE NO ACTION
);
```

Table **sales.order_items**

The **sales.order_items** table stores the line items of a sales order. Each line item belongs to a sales order specified by the **order_id** column.

A sales order line item includes product, order quantity, list price, and discount.

```
CREATE TABLE sales.order_items(
    order_id INT,
    item_id INT,
    product_id INT NOT NULL,
    quantity INT NOT NULL,
    list_price DECIMAL (10, 2) NOT NULL,
    discount DECIMAL (4, 2) NOT NULL DEFAULT 0,
    PRIMARY KEY (order_id, item_id),
    FOREIGN KEY (order_id)
REFERENCES sales.orders (order_id)
ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (product_id)
REFERENCES production.products (product_id)
ON DELETE CASCADE ON UPDATE CASCADE
);
```

Table **production.stocks**

The **production.stocks** table stores the inventory information i.e. the quantity of a particular product in a specific store.

SQL SERVER SELF JOIN

```
CREATE TABLE production.stocks (
        store_id INT,
        product_id INT,
        quantity INT,
        PRIMARY KEY (store_id, product_id),
        FOREIGN KEY (store_id)
    REFERENCES sales.stores (store_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (product_id)
    REFERENCES production.products (product_id)
    ON DELETE CASCADE ON UPDATE CASCADE
);
```

SQL Server self join syntax

A self-join allows you to join a table to itself. It helps query hierarchical data or compare rows within the same table.

A self-join uses the inner join or left join clause. Because the query that uses the self-join references the same table, the table alias is used to assign different names to the same table within the query.

Note that referencing the same table more than one in a query without using table aliases will result in an error.

The following shows the syntax of joining the table `T` to itself:

```
SELECT
   select_list
FROM
   T t1
[INNER | LEFT]  JOIN T t2 ON
   join_predicate;
```

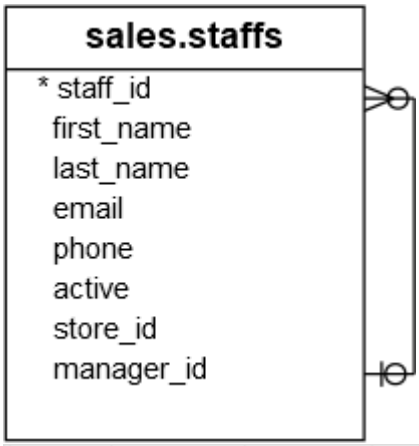
SQL SERVER SELF JOIN

SQL Server self join examples

Let's take some examples to understand how the self join works.

1) Using self join to query hierarchical data

Consider the following staffs table from the BIKESTORE DATABASE



staff_id	first_name	last_name	email	phone	active	store_id	manager_id
1	Fabiola	Jackson	fabiola.jackson@bikes.shop	(831) 555-5554	1	1	NULL
2	Mireya	Copeland	mireya.copeland@bikes.shop	(831) 555-5555	1	1	1
3	Genna	Serrano	genna.serrano@bikes.shop	(831) 555-5556	1	1	2
4	Virgie	Wiggins	virgie.wiggins@bikes.shop	(831) 555-5557	1	1	2
5	Jannette	David	jannette.david@bikes.shop	(516) 379-4444	1	2	1
6	Marcelene	Boyer	marcelene.boyer@bikes.shop	(516) 379-4445	1	2	5
7	Venita	Daniel	venita.daniel@bikes.shop	(516) 379-4446	1	2	5
8	Kali	Vargas	kali.vargas@bikes.shop	(972) 530-5555	1	3	1
9	Layla	Terrell	layla.terrell@bikes.shop	(972) 530-5556	1	3	7
10	Bernardine	Houston	bernardine.houston@bikes.shop	(972) 530-5557	1	3	7

The staffs table stores the staff information such as id, first name, last name, and email. It also has a column named manager_id that specifies the direct manager. For example, Mireya reports to Fabiola because the value in the manager_id of Mireya is Fabiola.

Fabiola has no manager, so the manager id column has a NULL.

To get who reports to whom, you use the self join as shown in the following query:

```
SELECT
    e.first_name + ' ' + e.last_name employee,
    m.first_name + ' ' + m.last_name manager
FROM
    sales.staffs e
INNER JOIN sales.staffs m ON m.staff_id = e.manager_id
ORDER BY
```

SQL SERVER SELF JOIN

```
    manager;
```
Code language: SQL (Structured Query Language) (sql)

employee	manager
Mireya Copeland	Fabiola Jackson
Jannette David	Fabiola Jackson
Kali Vargas	Fabiola Jackson
Marcelene Boyer	Jannette David
Venita Daniel	Jannette David
Genna Serrano	Mireya Copeland
Virgie Wiggins	Mireya Copeland
Layla Terrell	Venita Daniel
Bernardine Houston	Venita Daniel

In this example, we referenced to the staffs table twice: one as e for the employees and the other as m for the managers. The join predicate matches employee and manager relationship using the values in the e.manager_id and m.staff_id columns.

The employee column does not have Fabiola Jackson because of the INNER JOIN effect. If you replace the INNER JOIN clause by the LEFT JOIN clause as shown in the following query, you will get the result set that includes Fabiola Jackson in the employee column:

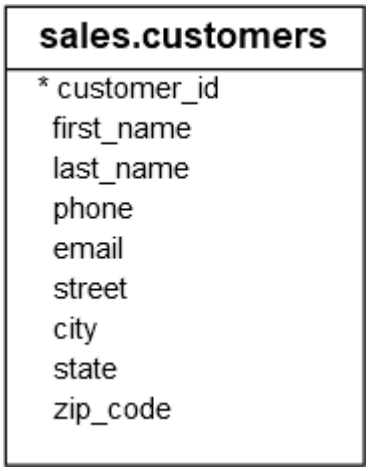
```
SELECT
    e.first_name + ' ' + e.last_name employee,
    m.first_name + ' ' + m.last_name manager
FROM
    sales.staffs e
LEFT JOIN sales.staffs m ON m.staff_id = e.manager_id
ORDER BY
    manager;
```
Code language: SQL (Structured Query Language) (sql)

employee	manager
Fabiola Jackson	NULL
Mireya Copeland	Fabiola Jackson
Jannette David	Fabiola Jackson
Kali Vargas	Fabiola Jackson
Marcelene Boyer	Jannette David
Venita Daniel	Jannette David
Genna Serrano	Mireya Copeland
Virgie Wiggins	Mireya Copeland
Layla Terrell	Venita Daniel
Bernardine Houston	Venita Daniel

2) Using self join to compare rows within a table

See the following customers table:

SQL SERVER SELF JOIN



The following statement uses the self join to find the customers located in the same city.

```
SELECT
    c1.city,
    c1.first_name + ' ' + c1.last_name customer_1,
    c2.first_name + ' ' + c2.last_name customer_2
FROM
    sales.customers c1
INNER JOIN sales.customers c2 ON c1.customer_id >
c2.customer_id
AND c1.city = c2.city
ORDER BY
    city,
    customer_1,
    customer_2;
```

Code language: SQL (Structured Query Language) (sql)

SQL SERVER SELF JOIN

city	customer_1	customer_2
Albany	Douglass Blankenship	Mi Gray
Albany	Douglass Blankenship	Priscilla Wilkins
Albany	Mi Gray	Priscilla Wilkins
Amarillo	Andria Rivers	Delaine Estes
Amarillo	Andria Rivers	Jonell Rivas
Amarillo	Andria Rivers	Luis Tyler
Amarillo	Andria Rivers	Narcisa Knapp
Amarillo	Delaine Estes	Jonell Rivas
Amarillo	Delaine Estes	Luis Tyler
Amarillo	Delaine Estes	Narcisa Knapp
Amarillo	Jonell Rivas	Luis Tyler
Amarillo	Jonell Rivas	Narcisa Knapp
Amarillo	Narcisa Knapp	Luis Tyler
Amityville	Abby Gamble	Tenisha Lyons
Amityville	Barton Cox	Abby Gamble
Amityville	Barton Cox	Kylee Dickson
Amityville	Barton Cox	Marisa Chambers
Amityville	Barton Cox	Myron Ruiz
Amityville	Barton Cox	Tenisha Lyons
Amityville	Barton Cox	Thalia Horne

The following condition makes sure that the statement doesn't compare the same customer:

```
c1.customer_id > c2.customer_id
```
Code language: SQL (Structured Query Language) (sql)

And the following condition matches the city of the two customers:

```
AND c1.city = c2.city
```
Code language: SQL (Structured Query Language) (sql)

Note that if you change the greater than (>) operator by the not equal to (<>) operator, you will get more rows:

```
SELECT
    c1.city,
    c1.first_name + ' ' + c1.last_name customer_1,
    c2.first_name + ' ' + c2.last_name customer_2
FROM
    sales.customers c1
INNER JOIN sales.customers c2 ON c1.customer_id <>
c2.customer_id
AND c1.city = c2.city
ORDER BY
    city,
    customer_1,
    customer_2;
```

SQL SERVER SELF JOIN

Code language: SQL (Structured Query Language) (sql)

city	customer_1	customer_2
Albany	Douglass Blankenship	Mi Gray
Albany	Douglass Blankenship	Priscilla Wilkins
Albany	Mi Gray	Douglass Blankenship
Albany	Mi Gray	Priscilla Wilkins
Albany	Priscilla Wilkins	Douglass Blankenship
Albany	Priscilla Wilkins	Mi Gray
Amarillo	Andria Rivers	Delaine Estes
Amarillo	Andria Rivers	Jonell Rivas
Amarillo	Andria Rivers	Luis Tyler
Amarillo	Andria Rivers	Narcisa Knapp
Amarillo	Delaine Estes	Andria Rivers
Amarillo	Delaine Estes	Jonell Rivas
Amarillo	Delaine Estes	Luis Tyler
Amarillo	Delaine Estes	Narcisa Knapp
Amarillo	Jonell Rivas	Andria Rivers
Amarillo	Jonell Rivas	Delaine Estes
Amarillo	Jonell Rivas	Luis Tyler
Amarillo	Jonell Rivas	Narcisa Knapp
Amarillo	Luis Tyler	Andria Rivers
Amarillo	Luis Tyler	Delaine Estes
Amarillo	Luis Tyler	Jonell Rivas
Amarillo	Luis Tyler	Narcisa Knapp

Let's see the difference between > and <> in the ON clause by limiting to one city to make it easier for comparison.

The following query returns the customers located in Albany:

```
SELECT
    customer_id, first_name + ' ' + last_name c,
    city
FROM
    sales.customers
WHERE
    city = 'Albany'
ORDER BY
    c;
```

Code language: SQL (Structured Query Language) (sql)

customer_id	c	city
1114	Douglass Blankenship	Albany
345	Mi Gray	Albany
210	Priscilla Wilkins	Albany

This query uses (>) operator in the ON clause:

```
SELECT
    c1.city,
```

SQL SERVER SELF JOIN

```
    c1.first_name + ' ' + c1.last_name customer_1,
    c2.first_name + ' ' + c2.last_name customer_2
FROM
    sales.customers c1
INNER JOIN sales.customers c2 ON c1.customer_id >
c2.customer_id
AND c1.city = c2.city
WHERE c1.city = 'Albany'
ORDER BY
    c1.city,
    customer_1,
    customer_2;
```

Code language: SQL (Structured Query Language) (sql)

The output is:

city	customer_1	customer_2
Albany	Douglass Blankenship	Mi Gray
Albany	Douglass Blankenship	Priscilla Wilkins
Albany	Mi Gray	Priscilla Wilkins

This query uses (<>) operator in the ON clause:

```
SELECT
    c1.city,
      c1.first_name + ' ' + c1.last_name customer_1,
    c2.first_name + ' ' + c2.last_name customer_2
FROM
    sales.customers c1
INNER JOIN sales.customers c2 ON c1.customer_id <>
c2.customer_id
AND c1.city = c2.city
WHERE c1.city = 'Albany'
ORDER BY
      c1.city,
    customer_1,
    customer_2;
```

Code language: SQL (Structured Query Language) (sql)

Here is the output:

city	customer_1	customer_2
Albany	Douglass Blankenship	Mi Gray
Albany	Douglass Blankenship	Priscilla Wilkins
Albany	Mi Gray	Douglass Blankenship
Albany	Mi Gray	Priscilla Wilkins
Albany	Priscilla Wilkins	Douglass Blankenship
Albany	Priscilla Wilkins	Mi Gray