Foreign Key

SQL FOREIGN KEY Constraint

A FOREIGN KEY is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Look at the following two tables:

"Persons" table:

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

"Orders" table:

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

MySQL:

```
CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
   FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders (
   OrderID int NOT NULL PRIMARY KEY,
   OrderNumber int NOT NULL,
   PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
   CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
   REFERENCES Persons(PersonID)
);
```

SQL FOREIGN KEY on ALTER TABLE

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL:

MySQL:

```
ALTER TABLE Orders
DROP FOREIGN KEY FK_PersonOrder;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
DROP CONSTRAINT FK_PersonOrder;
```

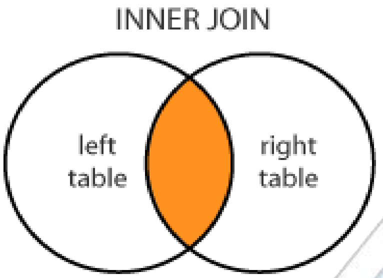
SQL JOIN

A **JOIN** clause is used to combine a rows from two or more tables, based on related column between them.

Four Basic types:

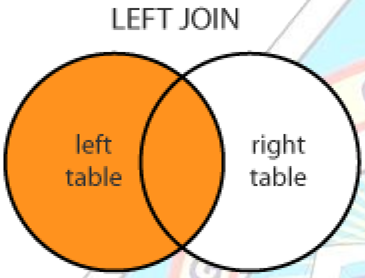
1. **Inner Join**

   Select all records from Table A and Table B where the join condition is **met**



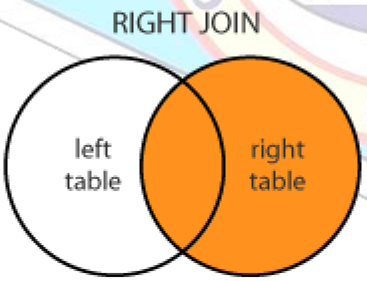
2. **Left Join**

   Select **all** records from **Table A**, along with records from Table B for which the join condition is met (if at all).



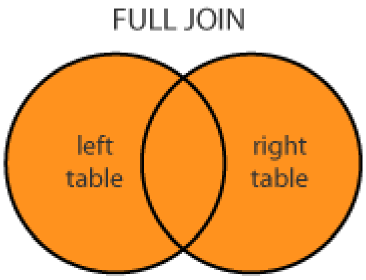
3. Right Join

   Select **all** records from **Table B,** along with records from Table A for which the join condition is met (if at all).



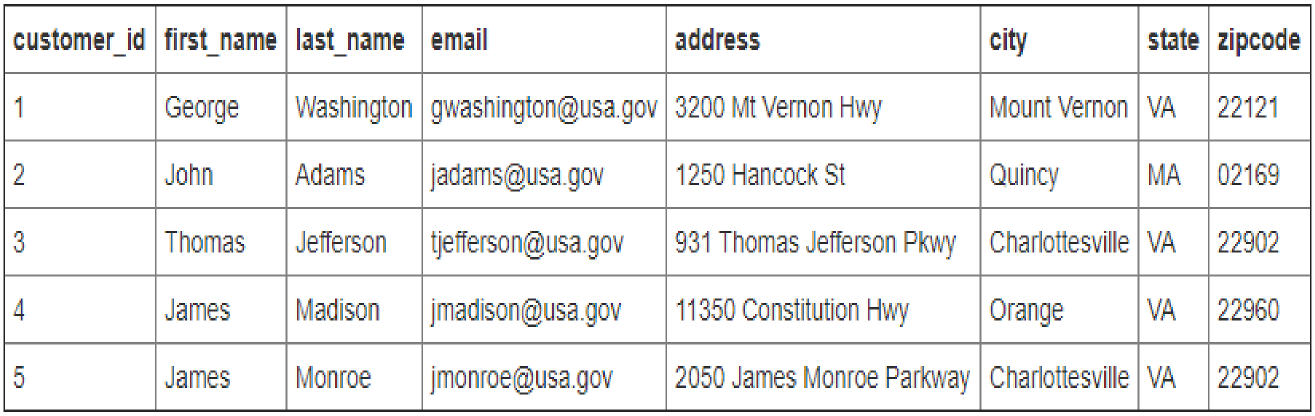
4. Full Join

   Select **all** records from Table A and Table B, regardless of whether the join condition is **met** or **not**

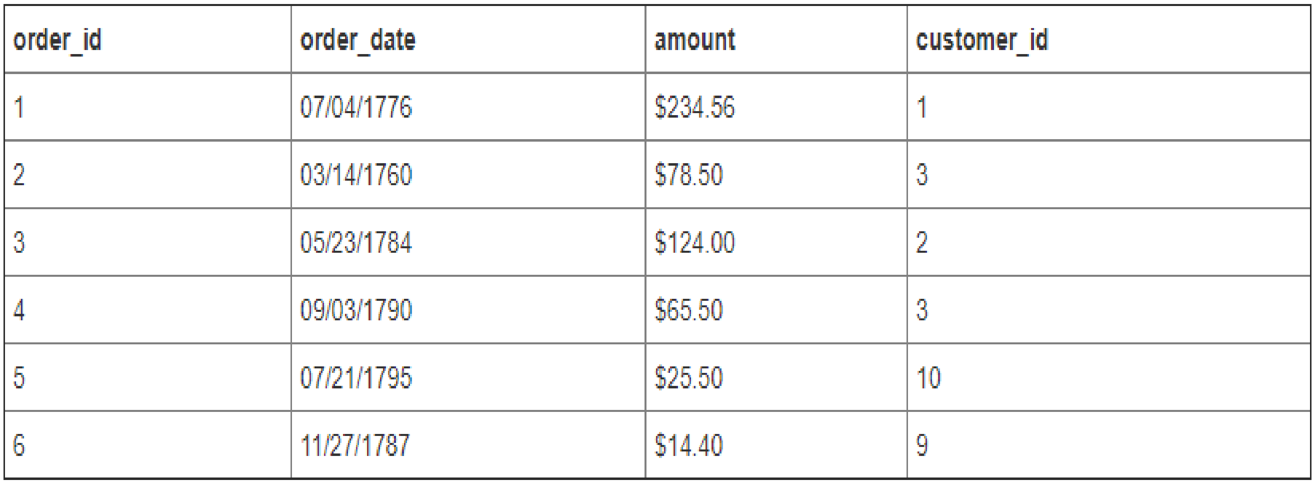


Examples of SQL Join Types

Let's use the tables we introduced in the “What is a SQL join?” section to show examples of these joins in action. The relationship between the two tables is specified by the customer_id key, which is the "primary key" in customers table and a "foreign key" in the orders table:



customer_id	first_name	last_name	email	address	city	state	zipcode
1	George	Washington	gwashington@usa.gov	3200 Mt Vernon Hwy	Mount Vernon	VA	22121
2	John	Adams	jadams@usa.gov	1250 Hancock St	Quincy	MA	02169
3	Thomas	Jefferson	tjefferson@usa.gov	931 Thomas Jefferson Pkwy	Charlottesville	VA	22902
4	James	Madison	jmadison@usa.gov	11350 Constitution Hwy	Orange	VA	22960
5	James	Monroe	jmonroe@usa.gov	2050 James Monroe Parkway	Charlottesville	VA	22902



order_id	order_date	amount	customer_id
1	07/04/1776	$234.56	1
2	03/14/1760	$78.50	3
3	05/23/1784	$124.00	2
4	09/03/1790	$65.50	3
5	07/21/1795	$25.50	10
6	11/27/1787	$14.40	9

Inner Join

Let’s say we wanted to get a list of those customers who placed an order and the details of the order they placed. This would be a perfect fit for an inner join, since an inner join returns records at the intersection of the two tables.

Syntax

```
select first_name, last_name, order_date, order_amount
from customers c
inner join orders o
on c.customer_id = o.customer_id;
```

first_name	last_name	order_date	order_amount
George	Washington	07/4/1776	$234.56
John	Adams	05/23/1784	$124.00
Thomas	Jefferson	03/14/1760	$78.50
Thomas	Jefferson	09/03/1790	$65.50

Note that only George Washington, John Adams and Thomas Jefferson placed orders, with Thomas Jefferson placing two separate orders on 3/14/1760 and 9/03/1790.

Left Join

If we wanted to simply append information about orders to our customers table, regardless of whether a customer placed an order or not, we would use a left join. A left join returns all records from table A and any matching records from table B.

Syntax

```
select first_name, last_name, order_date, order_amount
from customers c
left join orders o
on c.customer_id = o.customer_id
```

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	$234.56
John	Adams	05/23/1784	$124.00
Thomas	Jefferson	03/14/1760	$78.50
Thomas	Jefferson	09/03/1790	$65.50
James	Madison	NULL	NULL
James	Monroe	NULL	NULL

Note that since there were no matching records for James Madison and James Monroe in our orders table, the order_date and order_amount are NULL, which simply means there is no data for these fields.

Or

Add null like the query below:

```
select first_name, last_name, order_date, order_amount
from customers c
left join orders o
on c.customer_id = o.customer_id
where order_date is NULL
```

Right Join

Right join is a mirror version of the left join and allows to get a list of all orders, appended with customer information.

Syntax

```
select first_name, last_name, order_date, order_amount
from customers c
right join orders o
on c.customer_id = o.customer_id
```

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	$234.56
Thomas	Jefferson	03/14/1760	$78.50
John	Adams	05/23/1784	$124.00
Thomas	Jefferson	09/03/1790	$65.50
NULL	NULL	07/21/1795	$25.50
NULL	NULL	11/27/1787	$14.40

Note that since there were no matching customer records for orders placed in 1795 and 1787, the first_name and last_name fields are NULL in the resulting set.

Also note that the order in which the tables are joined is important. We are right joining the orders table to the customers table. If we were to right join the customers table to the orders table, the result would be the same as left joining the orders table to the customers table.

Why is this useful? Simply adding a “where first_name is NULL” line to our SQL query returns a list of all orders for which we failed to record information about the customers who placed them:

OR this syntax will help

```
select first_name, last_name, order_date, order_amount
from customers c
right join orders o
on c.customer_id = o.customer_id
where first_name is NULL
```

Full Join

Finally, for a list of all records from both tables, we can use a full join.

Syntax

```
select first_name, last_name, order_date, order_amount
from customers c
full join orders o
on c.customer_id = o.customer_id
```

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	$234.56
Thomas	Jefferson	03/14/1760	$78.50
John	Adams	05/23/1784	$124.00
Thomas	Jefferson	09/03/1790	$65.50
NULL	NULL	07/21/1795	$25.50
NULL	NULL	11/27/1787	$14.40
James	Madison	NULL	NULL
James	Monroe	NULL	NULL