SQL JOINs

Part 2

Self-Join

- A self JOIN is a regular join but the table is joined with itself.
- They are useful for comparisons within a table.

The general syntax is:

```
SELECT column-names
  FROM table-name T1 JOIN table-name T2
  WHERE condition
```

T1 and T2 are different table aliases for the same table

A self-join that returns vendors from cities in common with other vendors

```
SELECT DISTINCT Vendors1.VendorName, Vendors1.VendorCity,
    Vendors1.VendorState
FROM Vendors AS Vendors1 INNER JOIN Vendors AS Vendors2
    ON (Vendors1.VendorCity = Vendors2.VendorCity) AND
       (Vendors1.VendorState = Vendors2.VendorState) AND
       (Vendors1.VendorID <> Vendors2.VendorID)
ORDER BY Vendors1.VendorState, Vendors1.VendorCity;
```

The result set

	VendorName	VendorCity	VendorState
1	AT&T	Phoenix	AZ
2	Computer Library	Phoenix	AZ
3	Wells Fargo Bank	Phoenix	AZ
4	Aztek Label	Anaheim	CA
5	Blue Shield of California	Anaheim	CA
6	Abbey Office Furnishings	Fresno	CA
7	ASC Signs	Fresno	CA
8	BFI Industries	Fresno	CA

```
(84 rows)
```

(OUTER) JOINS

The explicit syntax for an outer join

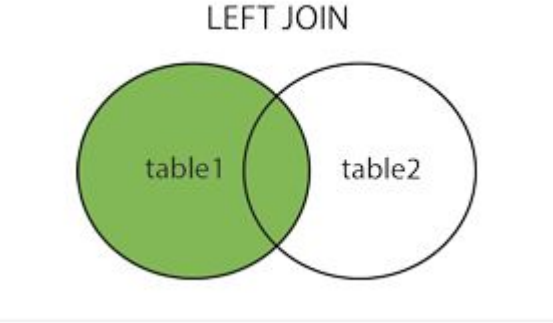
```
SELECT select_list
FROM table_1
    {LEFT|RIGHT|FULL} [OUTER] JOIN table_2
        ON join_condition_1
   [{LEFT|RIGHT|FULL} [OUTER] JOIN table_3
        ON join_condition_2]...
```

What outer joins do

Joins of this type	Keep unmatched rows from
Left outer join	The first (left) table
Right outer join	The second (right) table
Full outer join	Both tables

LEFT (OUTER) JOIN

- The INNER JOIN statement selects records that have matching values in both tables.
- The LEFT JOIN statement returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.



- **Note:** In some RDBMS LEFT JOIN is referred to as LEFT OUTER JOIN.

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

Note that (1) not every customer in our customers table has placed an order and (2) there are a few orders for which no customer record exists in our customers table.

- 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.

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

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

- 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.
- So why would this be useful? By adding the clause **WHERE order_date IS NULL** to our SQL query, it returns a list of all customers who have not placed an order

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

A SELECT statement that uses a left outer join

```
SELECT VendorName, InvoiceNumber, InvoiceTotal
FROM Vendors LEFT JOIN Invoices
    ON Vendors.VendorID = Invoices.VendorID
ORDER BY VendorName;
```

	VendorName	InvoiceNumber	InvoiceTotal
1	Abbey Office Furnishings	203339-13	17.50
2	American Booksellers Assoc	NULL	NULL
3	American Express	NULL	NULL
4	ASC Signs	NULL	NULL
5	Ascom Hasler Mailing Systems	NULL	NULL
6	AT&T	NULL	NULL

(202 rows)

The Departments table

	DeptName	DeptNo
1	Accounting	1
2	Payroll	2
3	Operations	3
4	Personnel	4
5	Maintenance	5

The Employees table

	EmployeeID	LastName	FirstName	DeptNo
1	1	Smith	Cindy	2
2	2	Jones	Elmer	4
3	3	Simonian	Ralph	2
4	4	Hernandez	Olivia	1
5	5	Aaronsen	Robert	2
6	6	Watson	Denise	6
7	7	Hardy	Thomas	5
8	8	O'Leary	Rhea	4
9	9	Locario	Paulo	6

The Projects table

	ProjectNo	EmployeeID
1	P1011	8
2	P1011	4
3	P1012	3
4	P1012	1
5	P1012	5
6	P1013	6
7	P1013	9
8	P1014	10

A left outer join

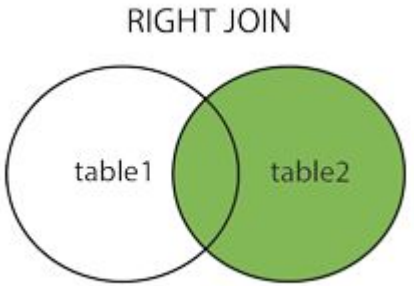
```
SELECT DeptName, Departments.DeptNo, LastName
FROM Departments LEFT JOIN Employees
    ON Departments.DeptNo = Employees.DeptNo;
```

The result set

	DeptName	DeptNo	LastName
1	Accounting	1	Hernandez
2	Payroll	2	Smith
3	Payroll	2	Simonian
4	Payroll	2	Aaronsen
5	Operations	3	NULL
6	Personnel	4	Jones
7	Personnel	4	O'Leary
8	Maintenance	5	Hardy

RIGHT (OUTER) JOIN

- The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match



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

Note that (1) not every customer in our customers table has placed an order and (2) there are a few orders for which no customer record exists in our customers table.

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

```
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:

```
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
```

A right outer join

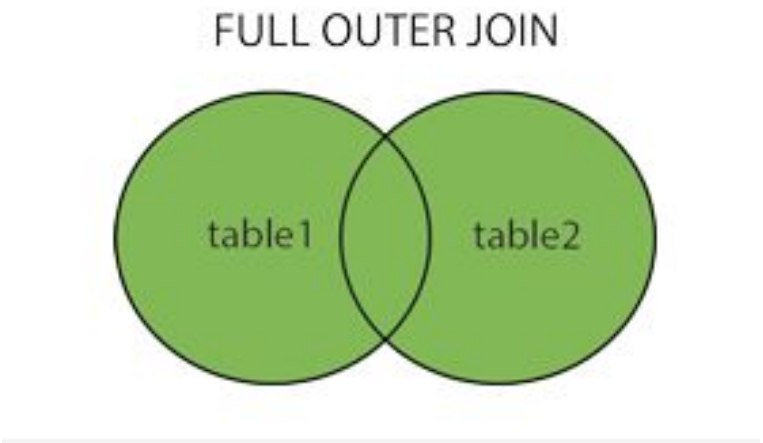
```
SELECT DeptName, Employees.DeptNo, LastName
FROM Departments RIGHT JOIN Employees
    ON Departments.DeptNo = Employees.DeptNo;
```

The result set

	DeptName	DeptNo	LastName
1	Payroll	2	Smith
2	Personnel	4	Jones
3	Payroll	2	Simonian
4	Accounting	1	Hernandez
5	Payroll	2	Aaronsen
6	NULL	6	Watson
7	Maintenance	5	Hardy
8	Personnel	4	O'Leary
9	NULL	6	Locario

FULL (OUTER) JOIN

- The FULL OUTER JOIN keyword return all records when there is a match in either left (table1) or right (table2) table records.
- **Note:** FULL OUTER JOIN can potentially return very large result-sets!



A full outer join

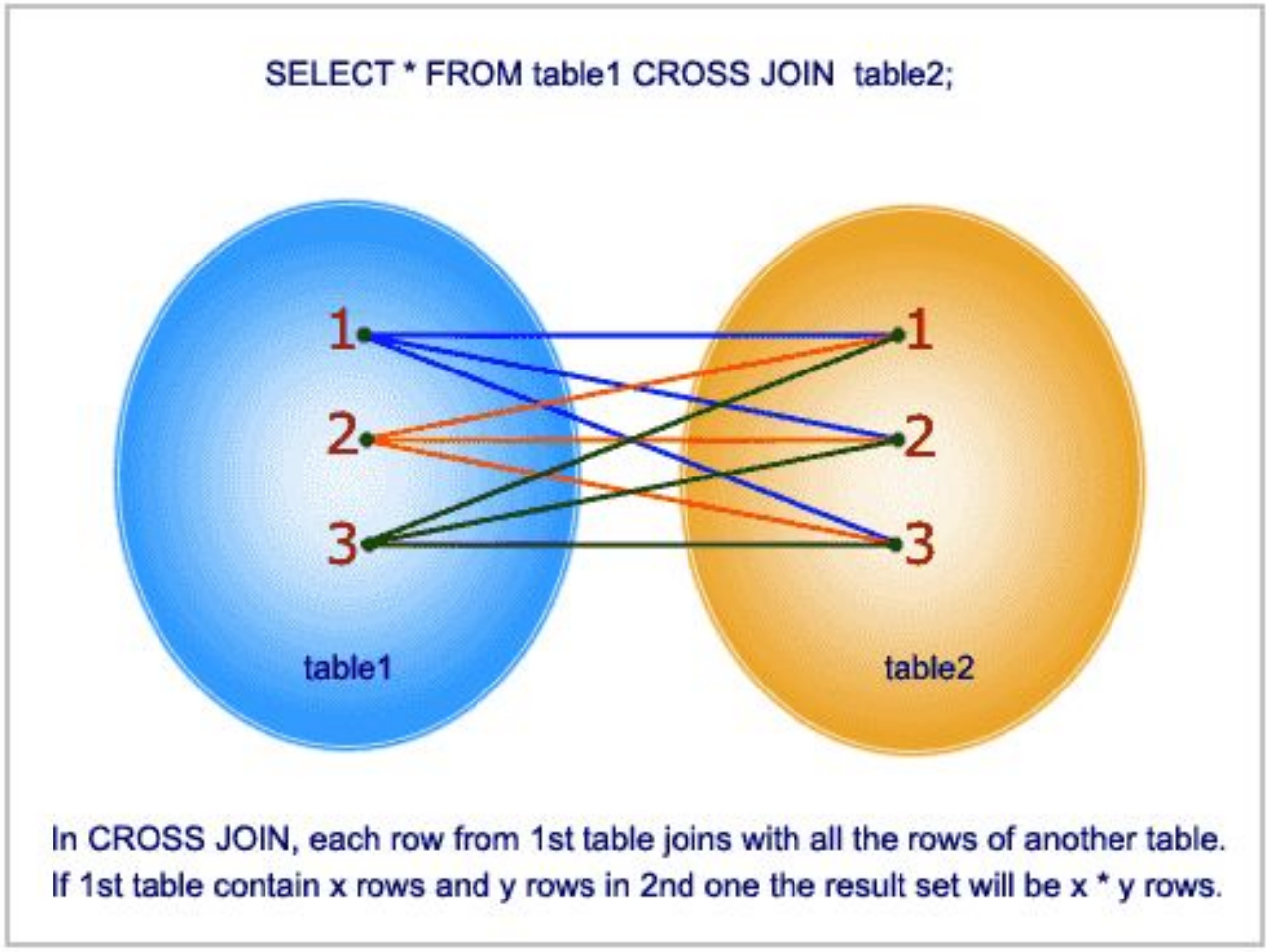
```
SELECT DeptName, Departments.DeptNo,
    Employees.DeptNo, LastName
FROM Departments FULL JOIN Employees
       ON Departments.DeptNo = Employees.DeptNo;
```

The result set

	DeptName	DeptNo	DeptNo	LastName
1	Accounting	1	1	Hernandez
2	Payroll	2	2	Smith
3	Payroll	2	2	Simonian
4	Payroll	2	2	Aaronsen
5	Operations	3	NULL	NULL
6	Personnel	4	4	Jones
7	Personnel	4	4	O'Leary
8	Maintenance	5	5	Hardy
9	NULL	NULL	6	Watson
10	NULL	NULL	6	Locario

CROSS JOIN

- A cross join is used when you wish to create combination of every row from two tables.
- All row combinations are included in the result; this is commonly called cross product join or Cartesian product. One use for a cross join is to obtain all combinations of items, such as colors and sizes.
- Each row in the first table is paired with all the rows in the second table.
- DANGER: if each table has 1000 rows, you will get 1,000,000 rows in the result set which is huuuuge



How to code a cross join using the explicit syntax

The explicit syntax for a cross join

```
SELECT select_list
FROM table_1 CROSS JOIN table_2
```

A cross join that uses the explicit syntax

```
SELECT Departments.DeptNo, DeptName,
    EmployeeID, LastName
FROM Departments CROSS JOIN Employees
ORDER BY Departments.DeptNo;
```

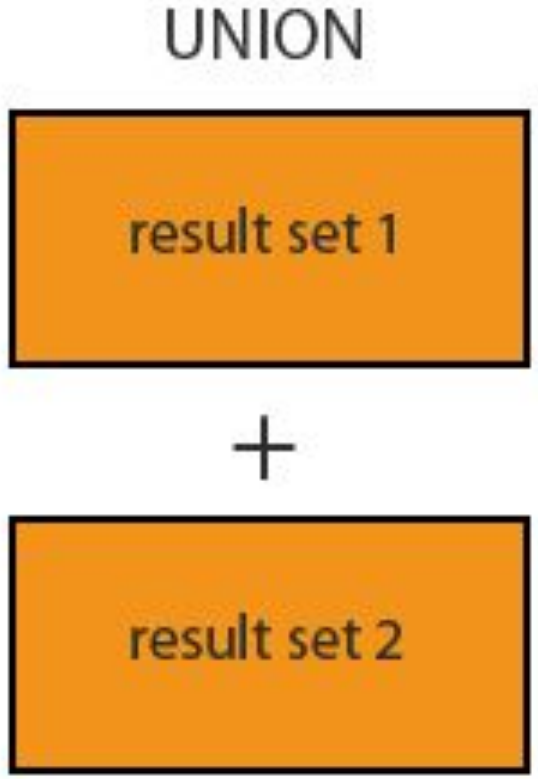
The result set created by the cross joins

	DeptNo	DeptName	EmployeeID	LastName
1	1	Accounting	1	Smith
2	1	Accounting	2	Jones
3	1	Accounting	3	Simonian
4	1	Accounting	4	Hernandez
5	1	Accounting	5	Aaronsen
6	1	Accounting	6	Watson
7	1	Accounting	7	Hardy

```
(45 rows)
```

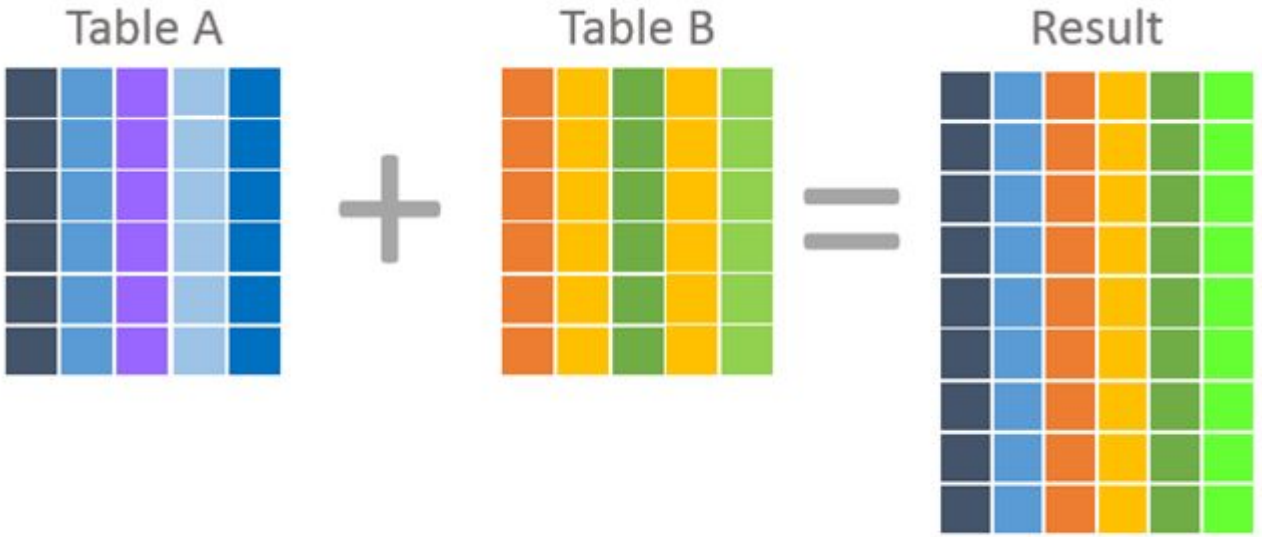
UNION

- The UNION operator is used to combine the result-set of two or more SELECT statements.
- Each SELECT statement within UNION must have the same number of columns
- The corresponding columns in each result set must have compatible data types
- The columns in each SELECT statement must also be in the same order



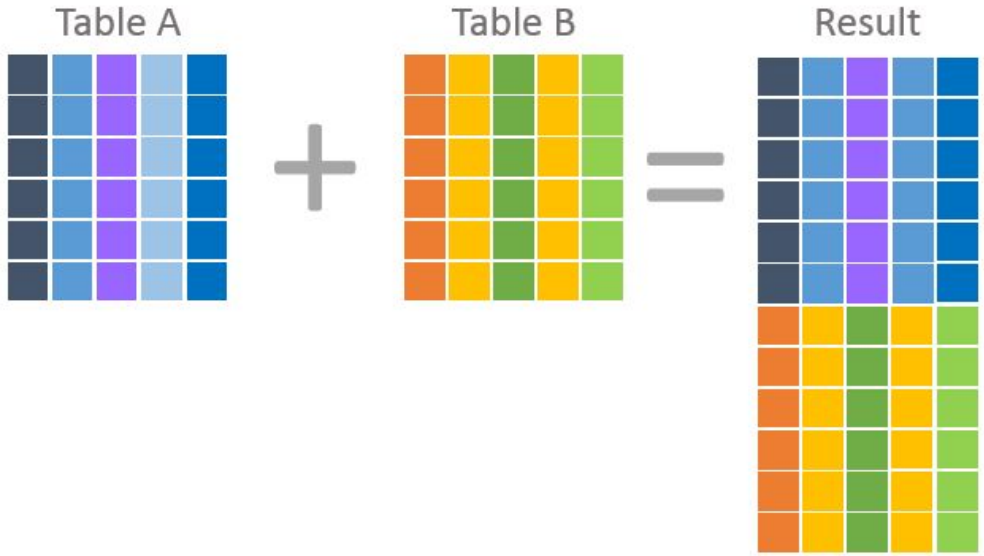
UNION vs JOIN

- JOINs and UNIONs can be used to combine data from one or more tables. The difference lies in how the data is combined.
- In simple terms, **joins combine data into new columns**. If two tables are joined together, then the data from the first table is shown in one set of column alongside the second table's column in the same row.
- Each row in the result contains columns from BOTH table A and B. Rows are created when columns from one table match columns from another. This match is called the join condition.



Joins Combine Columns

- **Unions combine data into new rows**. If two tables are “unioned” together, then the data from the first table is in one set of rows, and the data from the second table in another set. The rows are in the same result.



Unions Combine Rows

- In a union each row within the result is from one table OR the other. In a union, columns aren’t combined to create results, rows are combined.
- Unions are typically used where you have two results whose rows you want to include in the same result.

The syntax for a union operation

```
    SELECT_statement_1
UNION [ALL]
    SELECT_statement_2
[UNION [ALL]
    SELECT_statement_3]...
[ORDER BY order_by_list]
```

- The column names in the final result set are taken from the first SELECT clause
- When rows are combined duplicate rows are eliminated. If you want to keep all rows from both select statement's results use the **ALL** keyword.

A union that combines data from two different tables

```
    SELECT 'Active' AS Source, InvoiceNumber,
        InvoiceDate, InvoiceTotal
    FROM ActiveInvoices
    WHERE InvoiceDate >= '02/01/2016'
UNION
    SELECT 'Paid' AS Source, InvoiceNumber,
        InvoiceDate, InvoiceTotal
    FROM PaidInvoices
    WHERE InvoiceDate >= '02/01/2016'
ORDER BY InvoiceTotal DESC;
```

	Source	InvoiceNumber	InvoiceDate	InvoiceTotal
1	Paid	P-0259	2016-03-19 00:00:00	26881.40
2	Paid	0-2060	2016-03-24 00:00:00	23517.58
3	Paid	40318	2016-02-01 00:00:00	21842.00
4	Active	P-0608	2016-03-23 00:00:00	20551.18
5	Active	0-2436	2016-03-31 00:00:00	10976.06
6	Paid	509786	2016-02-18 00:00:00	6940.25
7	Paid	989319-447	2016-03-24 00:00:00	3689.99
8	Paid	989319-437	2016-02-01 00:00:00	2765.36
9	Paid	367447	2016-02-11 00:00:00	2433.00

```
(72 rows)
```

A union that combines data from the same table

```
    SELECT 'Active' AS Source, InvoiceNumber,
        InvoiceDate, InvoiceTotal
    FROM Invoices
    WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0
UNION
    SELECT 'Paid' AS Source, InvoiceNumber,
        InvoiceDate, InvoiceTotal
    FROM Invoices
    WHERE InvoiceTotal - PaymentTotal - CreditTotal <= 0
ORDER BY InvoiceTotal DESC;
```

The result set

	Source	InvoiceNumber	InvoiceDate	InvoiceTotal
1	Paid	0-2058	2016-01-28 00:00:00	37966.19
2	Paid	P-0259	2016-03-19 00:00:00	26881.40
3	Paid	0-2060	2016-03-24 00:00:00	23517.58
4	Paid	40318	2016-02-01 00:00:00	21842.00
5	Active	P-0608	2016-03-23 00:00:00	20551.18

(114 rows)

A union that combines payment data from the same joined tables

```
    SELECT InvoiceNumber, VendorName,
        '33% Payment' AS PaymentType,
        InvoiceTotal AS Total,
       (InvoiceTotal * 0.333) AS Payment
    FROM Invoices JOIN Vendors
        ON Invoices.VendorID = Vendors.VendorID
    WHERE InvoiceTotal > 10000
UNION
    SELECT InvoiceNumber, VendorName,
        '50% Payment' AS PaymentType,
        InvoiceTotal AS Total,
       (InvoiceTotal * 0.5) AS Payment
    FROM Invoices JOIN Vendors
        ON Invoices.VendorID = Vendors.VendorID
    WHERE InvoiceTotal BETWEEN 500 AND 10000
```

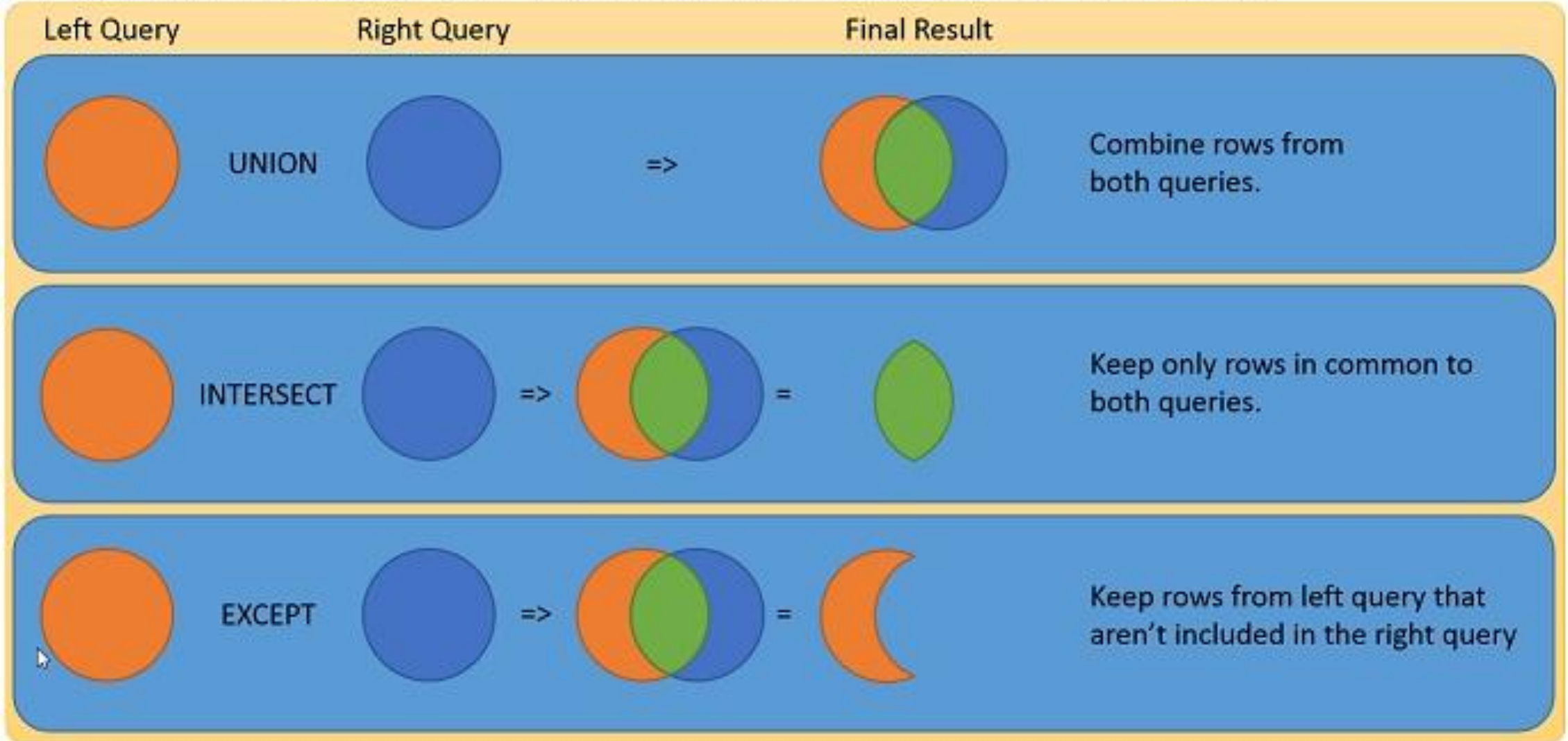
A union that combines payment data from the same joined tables (continued)

```
UNION
    SELECT InvoiceNumber, VendorName,
        'Full amount' AS PaymentType,
        InvoiceTotal AS Total,
        InvoiceTotal AS Payment
    FROM Invoices JOIN Vendors
        ON Invoices.VendorID = Vendors.VendorID
    WHERE InvoiceTotal < 500
ORDER BY PaymentType, VendorName, InvoiceNumber;
```

	InvoiceNumber	VendorName	PaymentType	Total	Payment
6	P-0608	Malloy Lithographing Inc	33% Payment	20551.18	6843.5429400
7	509786	Bertelsmann Industry S...	50% Payment	6940.25	3470.1250000
8	587056	Cahners Publishing Co...	50% Payment	2184.50	1092.2500000
9	367447	Computerworld	50% Payment	2433.00	1216.5000000

```
(114 rows)
```

Visual Explanation of UNION, INTERSECT, and EXCEPT operators



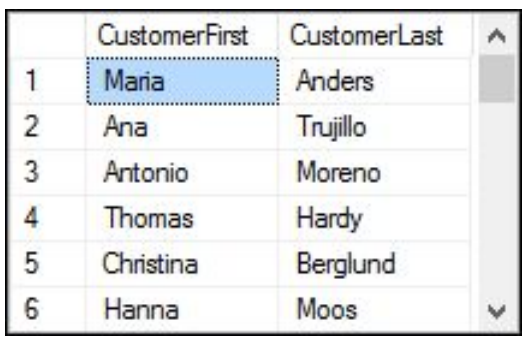
EXCEPT and INTERSECT

- Returns distinct rows by comparing the results of two queries.
  - EXCEPT returns distinct rows from the left input query that aren't output by the right input query.
  - INTERSECT returns distinct rows that are output by both the left and right input queries operator.
- The basic rules for combining the result sets of two queries that use EXCEPT or INTERSECT are the following:
  - The number and the order of the columns must be the same in all queries.
  - The data types must be compatible.

The syntax for the EXCEPT and INTERSECT operators

```
    SELECT_statement_1
{EXCEPT|INTERSECT}
    SELECT_statement_2
[ORDER BY order_by_list]
```

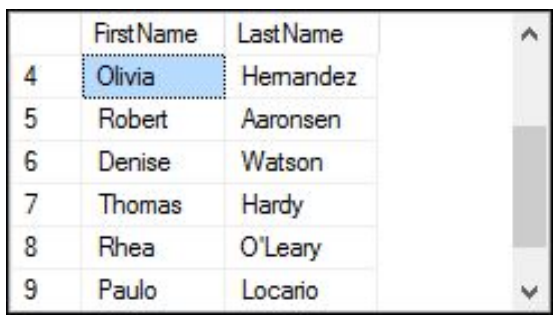
The Customers table



	CustomerFirst	CustomerLast
1	Maria	Anders
2	Ana	Trujillo
3	Antonio	Moreno
4	Thomas	Hardy
5	Christina	Berglund
6	Hanna	Moos

(24 rows)

The Employees table



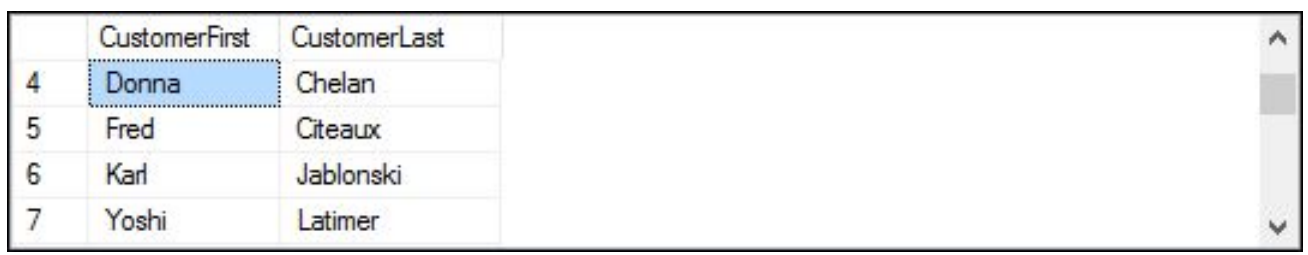
	FirstName	LastName
4	Olivia	Hernandez
5	Robert	Aaronsen
6	Denise	Watson
7	Thomas	Hardy
8	Rhea	O'Leary
9	Paulo	Locario

(9 rows)

Exclude rows from the first query if they also occur in the second query

```
    SELECT CustomerFirst, CustomerLast
    FROM Customers
EXCEPT
    SELECT FirstName, LastName
    FROM Employees
ORDER BY CustomerLast;
```

The result set



	CustomerFirst	CustomerLast
4	Donna	Chelan
5	Fred	Citeaux
6	Karl	Jablonski
7	Yoshi	Latimer

(23 rows)

Only include rows that occur in both queries

```
    SELECT CustomerFirst, CustomerLast
    FROM Customers
INTERSECT
    SELECT FirstName, LastName
    FROM Employees;
```

The result set

	CustomerFirst	CustomerLast
1	Thomas	Hardy

(1 row)