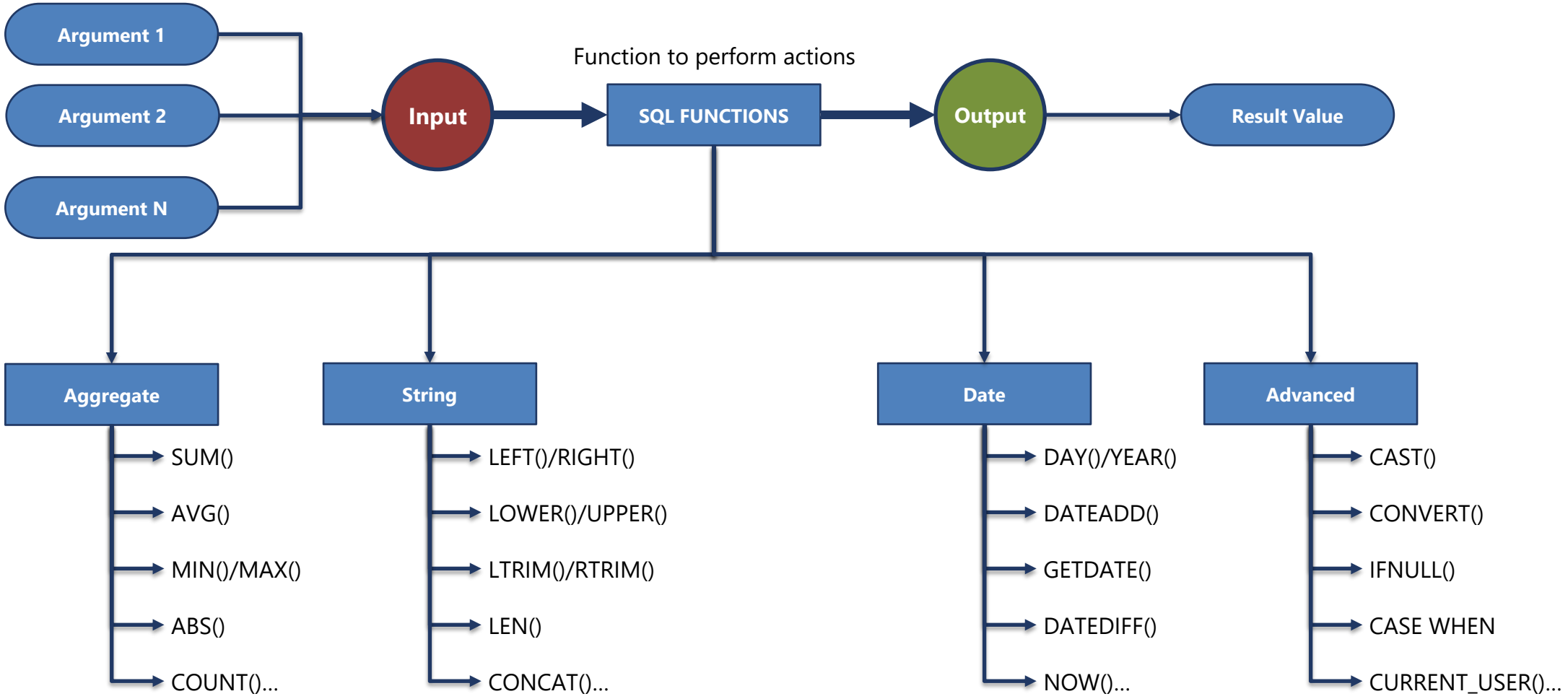
PART 4: Data Wrangling with SQL



PART 4: Data Wrangling with SQL

Aggregating

Aggregate functions allow you to summarize information about a group of rows.
It can be used to answer questions like “How much sales in 2020?” or “What’s the number of female employees?” or “What’s the highest salary?”

Note

- Without a specified grouping, aggregate function runs on whole set of result rows.
- It is a good practice to give your aggregate functions an alias to make result easier to read.
- In addition to aggregating across all rows, you can apply it to individual groups by the **GROUP BY** clause.

```
SELECT Price, COUNT(ProductID) AS "ProductCount"
FROM Products
WHERE Price BETWEEN 10 AND 15
GROUP BY Price;
```

Price	ProductCount
10	3
12	1
12.5	2
12.75	1
13	1
13.25	1
14	4
15	2

Syntax

```
SELECT AGG_FUNC(column) AS aggregate_description, …
FROM table_name
WHERE constraint_expressions;
```

Example

```
SELECT COUNT(ProductID) AS "ProductCount"
FROM Products
WHERE Price BETWEEN 10 AND 30
```

Result

Number of Records: 77

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22

ProductCount
42

PART 4: Data Wrangling with SQL

String Functions

String function is used to perform on input string and return an output string or numeric value.

Note

We can use SQL string functions to

- Extract information: LEFT(), RIGHT(), CONCAT()…
- Convert format: CONVERT(), CAST(), STR() …
- Transform string: LOWER(), LTRIM()…

Syntax

```
SELECT AGG_FUNC(column) AS aggregate_description, …
FROM table_name
WHERE constraint_expressions;
```

Example

```
SELECT CustomerID, LEFT(CustomerName, 5) AS "LeftCustomerName"
FROM Customers
ORDER BY CustomerID;
```

Result

CustomerID	LeftCustomerName
1	Alfre
2	Ana T
3	Anton
4	Aroun

PART 4: Data Wrangling with SQL

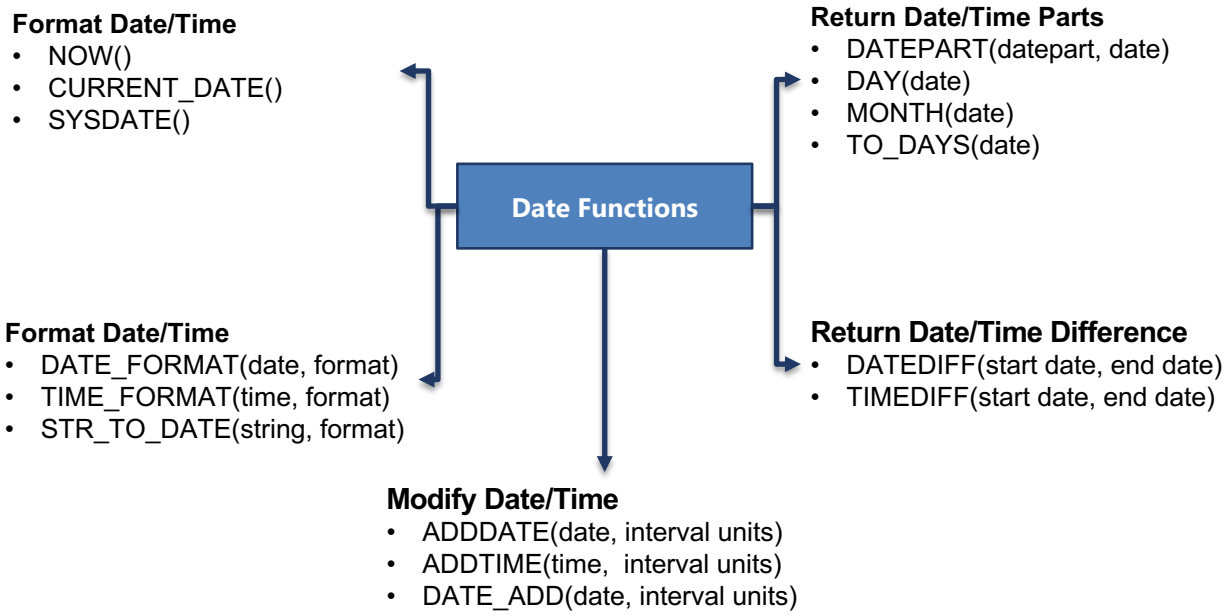
Date Functions

Date functions are used to deal with different date types. When working with date, make sure the date format matched the date column.

Note

Date types commonly used are

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY



Example

```
SELECT DATE_FORMAT(NOW(), '%d-%M-%Y')

SELECT DATEDIFF('2020-01-01 23:59:59','2020-12-31')
```

Result

DATE_FORMAT(NOW(), '%d-%M-%Y')
28-July-2020

DATEDIFF('2020-01-01 23:59:59','2020-12-31')
-365

PART 4: Data Wrangling with SQL

More …

Advance functions, such as CASE WHEN, IFNULL, CAST, CURRENT_USER, etc.

Note

- CASE statement: it goes through conditions and returns a value when the first condition is met. It is similar to IF-THEN-ELSE statement.

- IFNULL statement: it returns a specified value if NULL exists, otherwise, it returns the expression. Below query replace NULL with 500.

```
SELECT IFNULL(NULL, 500);
```

- CAST statement: it is used to convert a value to a specified datatype. Below query converts a value to DATE datatype.

```
SELECT CAST("2017-08-29" AS DATE);
```

Syntax

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

Example

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'
    WHEN Quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

Result

OrderID	Quantity	QuantityText
10248	12	The quantity is under 30
10248	10	The quantity is under 30
10248	5	The quantity is under 30

PART 4: Data Wrangling with SQL

A Database can have one or more tables.

Order Table

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2
10251	84	3	1996-07-08	1
10252	76	4	1996-07-09	2

Shipper Table

ShipperID	ShipperName	Phone
1	Speedy Express	(503) 555-9831
2	United Package	(503) 555-3199
3	Federal Shipping	(503) 555-9931

Customer Table

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

PART 4: Data Wrangling with SQL

SQL Joins

Up to now, we've been working on a single table. In real world, data is broken down into pieces and stored in multiple tables.
JOIN clause is used to link columns between two or more tables. There are mainly 4 types of JOIN in SQL:

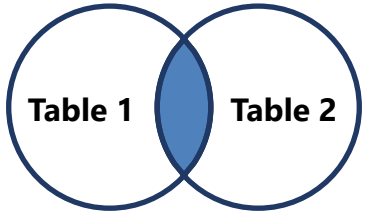
- INNER JOIN: Returns records that have matching values in both tables.
- LEFT JOIN: Returns all records from the left table, and the matched records from the right table.
- RIGHT JOIN: Returns all records from the right table, and the matched records from the left table.
- FULL OUTER JOIN: Returns all records when there is a match in either left or right table.

Table 1

Name	Gender
Andy	M
Bob	M
Charlie	M
Elsa	F
Felicia	F

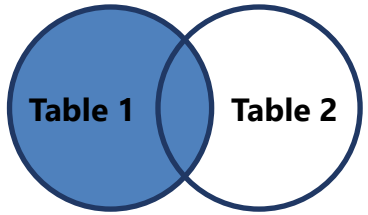
Table 2

Name	Age
Andy	15
Bob	23
Damian	27
Elsa	18
Gertrude	46



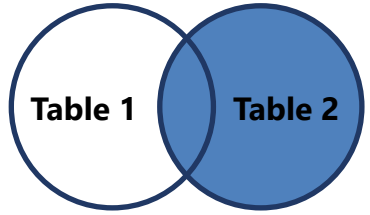
Inner Join

Name	Gender	Age
Andy	M	15
Bob	M	23
Elsa	F	18



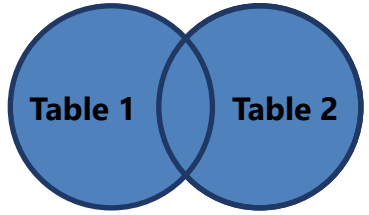
Left Join

Name	Gender	Age
Andy	M	15
Bob	M	23
Charlie	M	N/A
Elsa	F	18
Felicia	F	N/A



Right Join

Name	Age	Gender
Andy	15	M
Bob	23	M
Damien	27	N/A
Elsa	18	18
Gertrude	46	N/A

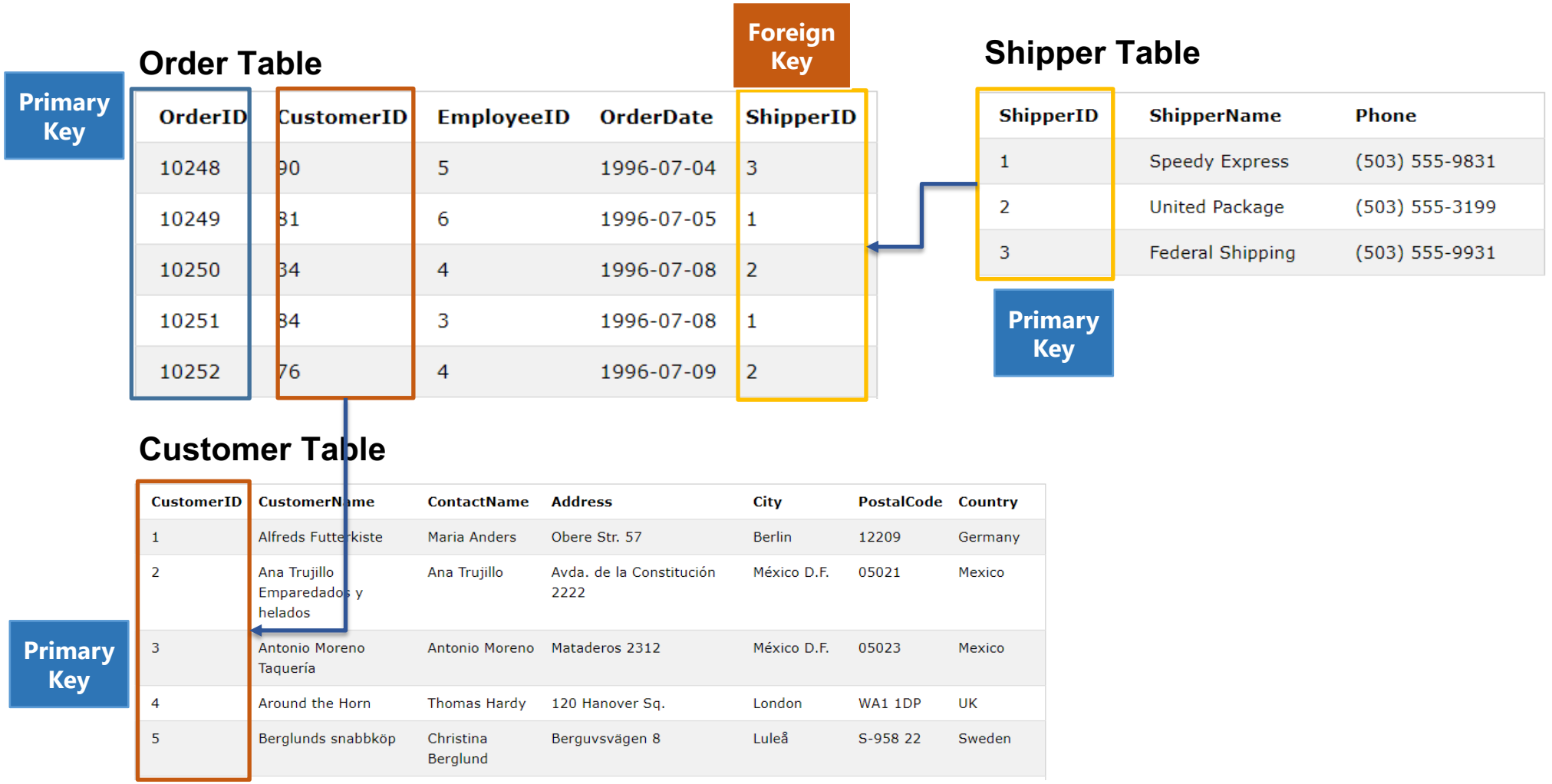


Full Outer Join

Name	Gender	Age
Andy	M	15
Bob	M	23
Charlie	M	N/A
Damien	N/A	27
Elsa	F	18
Felicia	F	N/A
Gertrude	N/A	46

PART 4: Data Wrangling with SQL

A Database can have one or more tables.



PART 4: Data Wrangling with SQL

Joins

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

Note

Tables that share the information need to have a **primary key** that identifies the records **uniquely** across tables. JOIN clause in query can combine records across tables using this **key**.

- PRIMARY KEY is the column which uniquely identifies each record.
- PRIMARY KEY must contain unique values.
- One table can only have one primary key. This primary key can consist of one or multiple columns.

Syntax

```
SELECT column1, column2, … FROM table1
LEFT JOIN table2
    ON table1.id = table2.id
```

Example

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;
```

Result

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

OrderID	CustomerName	OrderDate
10248	Wilman Kala	1996-07-04
10249	Tradição Hipermercados	1996-07-05
10250	Hanari Carnes	1996-07-08

PART 4: Data Wrangling with SQL

SQL Set (Union, Union All, Intersect, Except)

Joins are used to add more columns. When working with multiple tables, we may need to add more rows by combining results into one table. There are 4 commonly used set operations: UNION, UNION ALL, INTERCEPT and EXCEPT.

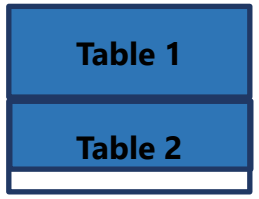
- When using UNION, the tables should have same column count, order and data type.
- UNION ALL will keep the duplicate rows. UNION without ALL will remove the duplicate rows.
- INTERSECT will only keep rows that are identical in both tables.
- EXCEPT will ensure only rows in the first table set that aren't in the second are returned.

Table 1

Name	Gender
Andy	M
Bob	M
Charlie	M
Elsa	F
Felicia	F

Table 2

Name	Gender
Andy	M
Bob	M
Damian	M
Elsa	F
Gertrude	F

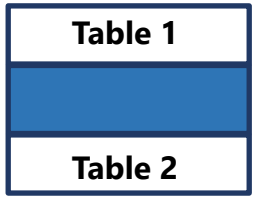


UNION

Name	Gender
Andy	M
Bob	M
Charlie	M
Elsa	F
Felicia	F
Damian	M
Gertrude	F

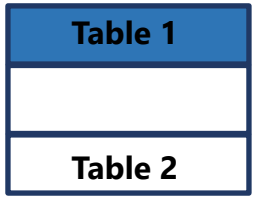
UNION ALL

Name	Gender
Andy	M
Bob	M
Charlie	M
Elsa	F
Felicia	F
Andy	M
Bob	M
Damian	M
Elsa	F
Gertrude	F



INTERSECT

Name	Gender
Andy	M
Bob	M
Elsa	F



EXCEPT

Name	Gender
Charlie	M
Felicia	F

PART 4: Data Wrangling with SQL

Sets

UNION/UNION ALL/INTERSECT/EXCEPT are the commonly used set operators.

Note

- When appending tables, they should have same column count, order and data type.
- In order of operations, UNION happens before ORDER BY and LIMIT.

Syntax

```
SELECT column1, column2,...   FROM table1
UNION / UNION ALL / INTERSECT / EXCEPT
SELECT column1, column2,...  FROM table2
```

Example

```
SELECT 'Customer' AS Type, ContactName, City, Country
FROM Customers
UNION
SELECT 'Supplier', ContactName, City, Country
FROM Suppliers
```

Result

Number of Records: 120

Type	ContactName	City	Country
Customer	Alejandra Camino	Madrid	Spain
Customer	Alexander Feuer	Leipzig	Germany
Customer	Ana Trujillo	México D.F.	Mexico

PART 4: Data Wrangling with SQL

Nested Queries

You may notice that there are many questions we cannot answer without additional processing. You can either make multiple queries and process the data on your own, or you can use nested queries to build a more complex query and answer more questions.

Note

- Subquery must be enclosed within parentheses.
- A subquery can be referenced anywhere a normal table can be referenced. Subqueries can be used with SELECT, INSERT, UPDATE, DELETE statement etc.

Example

Table below shows sales revenue each employee brings in and their salary. You need to find out who are the employees costing more than the average sales revenue brought?

First, you need to calculate the average sales revenue.

Then using the result from 1st step, you compare the costs of each employee against the value.

EmployeeSales

employeeid	name	salary	sales
1	Mark	2500	3000
2	Daniel	2000	2500
3	Helen	2850	2600
4	Steve	3000	3100
5	Sheryl	2350	2400

Syntax

```
SELECT column_name1, column_name2,...
FROM table_name1
WHERE column1 OPERATOR
    (SELECT column_name
     FROM table_name2
     WHERE condition)
```

Example

```
SELECT name
FROM EmployeeSales
WHERE salary >
   (SELECT AVG(sales)
    FROM EmployeeSales)
```

Result

Helen

Steve

PART 4: Data Wrangling with SQL

Nested Queries

Subqueries are not limited to use in WHERE clause. You can use it in FROM clause.

More Example

Table below shows sales revenue each employee brings in and their department. You need to find out which department corresponds to the highest average salary?
First, you need to get the average salary by department.
Then using the result from 1st step, you find the maximum average salary.

DepartmentSales

employeeid	name	department	salary
1	Mark	Finance	2500
2	Daniel	IT	2000
3	Helen	Marketing	2850
4	Steve	IT	3000
5	Sheryl	Marketing	2350

The inner query returns a table with two columns: **department**, **avg_salary** to show the average salary of each department.
Then outer query simply calculates the maximum average salary based on the **salary_by_department** table.

Syntax

```
SELECT column_name1, column_name2,...
FROM table_name1
WHERE column1 OPERATOR
    (SELECT column_name
     FROM table_name2
     WHERE condition)
```

Example

```
SELECT department, MAX(salary_by_department.avg_salary) AS max_salary
FROM (
    SELECT department, AVG(salary) AS avg_salary
    FROM DepartmentSales
    GROUP BY department) salary_by_department;
```

Result

department	max_salary
Finance	2600

PART 4: Data Wrangling with SQL

Nested Query – INSERT/UPDATE/DELETE

INSERT statement can use the returned result from the subquery and insert into another table.
UPDATE statement can update either single or multiple columns in a table.

Example

Consider a table **Customer_New** with similar structure as **Customer** table. You want to copy the complete **Customer_New** into the Customer table.

Customer

CustomerID	Name	City
1	Mark	Berlin
2	Daniel	London
3	Helen	Chicago
4	Steve	New York
5	Sheryl	Singapore

Customer_New

CustomerID	Name	City
6	Dave	London
7	Mark	Tokyo
8	Karren	Sydney

```
INSERT INTO Customer
SELECT * FROM Customer_New
```

Customer

CustomerID	Name	City
1	Mark	Berlin
2	Daniel	London
3	Helen	Chicago
4	Steve	New York
5	Sheryl	Singapore
6	Dave	London
7	Mark	Tokyo
8	Karren	Sydney

```
DELETE FROM Customer
WHERE City IN (
    SELECT City FROM Customer_New)
```

Customer

CustomerID	Name	City
1	Mark	Berlin
3	Helen	Chicago
4	Steve	New York
5	Sheryl	Singapore

Syntax - INSERT

```
INSERT INTO table_name1 (column1, column2,...)
SELECT column1, column2,...
  FROM table_name2
 WHERE condition
```

Syntax - UPDATE

```
UPDATE table_name1
SET column1 = new_value
WHERE operator
   (SELECT column_name
   FROM table_name
   WHERE condition)
```

Syntax - DELETE

```
DELETE FROM table_name1
WHERE operator
   (SELECT column_name
   FROM table_name
   WHERE condition)
```