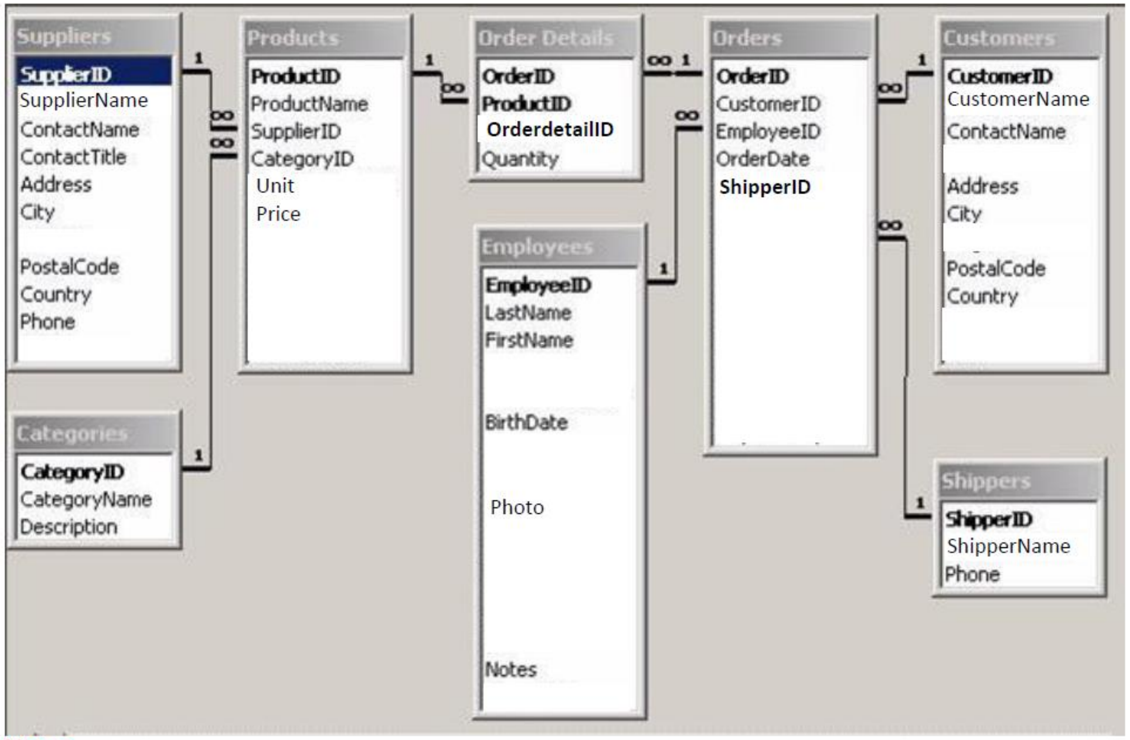
SQL Lab Tutorial

Objectives

- Practice writing SQL queries to retrieve required information from database
- Understand database concepts



Notes

- There is no space between the words “Order” and “Details” in the name of the OrderDetails table.

SELECT Statement

Used to retrieve data specified by the user

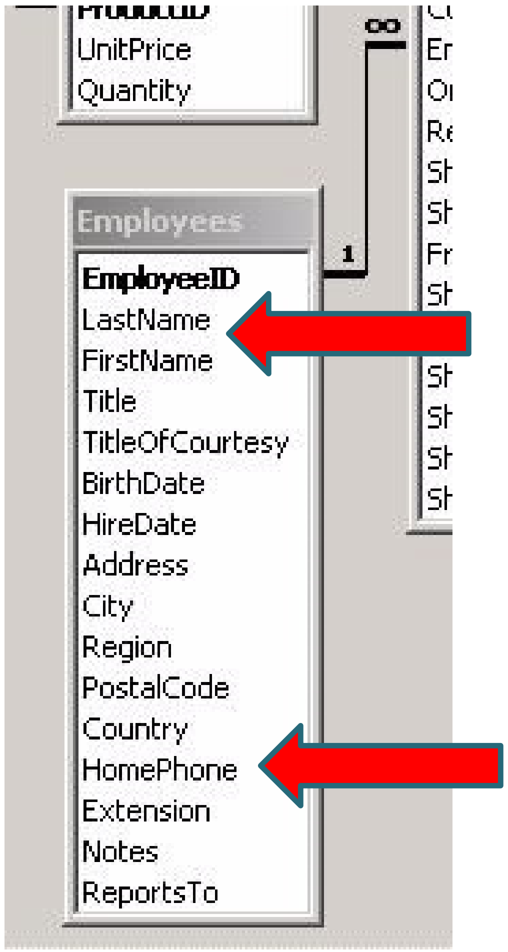
Example: You want to see first names and last names of employees in the Northwinds database.

In the SELECT statement you would specify these fields by:

SELECT LastName, FirstName

This statement's attributes must match the attribute names exactly as shown in the model. (Capitalization not required.)

SELECT LastName, FirstName



Note: Each attribute from each table in Northwinds represents a column (heading) in the data that is retrieved as shown below.

EmployeeID	LastName	FirstName	Title
101	Jones	Aaron	Manager
102	Smith	Ann	Director
103	Edwards	Jim	Associate

**Example data only shown above*

FROM Clause

Now that you have chosen attributes, the next requirement is to specify which table the attributes come from by using the “FROM” clause.   The table(s) must contain the attributes listed in the SELECT statement.

Example (Q1):

Please do not copy and paste the query to W3Schols (it may not work because of formatting issues), but type the query there.

```
SELECT LastName, FirstName
FROM  Employees
```

The SELECT statement can also use an asterisk (*) to include all fields from tables specified in the FROM clause.

EXAMPLE (Q2)**:** You want to pull all of the attributes from the Employees table.

SELECT   *
FROM employees

Note: You can use this function with more than one table, but a <u>table join</u> is needed (which will be discussed later).

WHERE Clause

Though a query only requires a SELECT statement and a FROM clause, additional clauses are sometimes necessary for more complex requests.

When using the WHERE clause, you can set a specific condition by using an operator.

=	Equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<> or !=	Not equal to

Example (Q3):

```
SELECT CustomerName, ContactName, City
FROM  Customers
WHERE city = “London”
```

Please do not copy and paste the query to W3Schols (it may not work because of formatting issues), but type the query there.

Some other examples:

```
SELECT CustomerName, ContactName, City
FROM  Customers
WHERE city != “London”
```

```
SELECT CustomerName, ContactName, City
FROM  Customers
WHERE city <> “London”
```

AGGREGATE FUNCTIONS

These functions are used to return a single calculated value about specific attributes in the SELECT clause.

MIN (…)	returns the smallest value in a given column
MAX (…)	returns the largest value in a given column
SUM (…)	returns the sum of the numeric values in a given column
AVG (…)	returns the average value of a given column
COUNT (…)	returns the total number of values in a given column
COUNT(*)	returns the number of rows in a table

GROUP BY Clause

RULE: ALWAYS USE THE **GROUP BY** CLAUSE WHEN YOU HAVE AN AGGREGATE FUNCTION (i.e. min, max, sum, etc.) IN THE **SELECT STATEMENT** WITH OTHER NON-AGGREGATE ATTRIBUTES.

In the GROUP BY clause you must list all the attributes that are NOT aggregates in order for the data to process correctly.

Example (Q4)**:** Find the average price for each supplier.

```
SELECT SupplierID, AVG(Price)
FROM Products
GROUP BY SupplierID
```

HAVING Clause

The HAVING clause is best described as a way to specify conditions for the aggregate functions that are in the SELECT statement.

It follows the GROUP BY clause and can be viewed as the “WHERE” clause for attributes involved with aggregate functions.

Example (Q5)**:** Find supplier with an average price greater than $20.

```
SELECT SupplierID, AVG(Price)
FROM Products
GROUP BY SupplierID
HAVING AVG(Price)>20
```

ORDER BY Clause

ORDER BY allows the user to sort the results in a specific order (ascending or descending). If you do not specify to sort in either order, your results will automatically be sorted in **ascending** order.

Add *asc* or *desc* at the end of the clause to specify.

Example (Q6): Find suppliers with an average price greater than $20 with supplier IDs greater than 5. Sort SupplierIDs in descending order.

```
SELECT SupplierID, AVG(price)
FROM Products
WHERE SupplierID > 5
GROUP BY SupplierID
HAVING AVG(price)>20
ORDER BY SupplierID desc
```

* Note: WHERE vs HAVING

Conditions

Compound conditions can be used to set even more parameters for the data. Many conditions can be used in both the WHERE and the HAVING clauses and are placed between the conditions.

AND : this operator joins two or more conditions, and displays a row only if that row’s data satisfies ALL conditions that are listed

Example (Q7):

```
SELECT  *
FROM products
WHERE  ProductID>5 AND CategoryID=4
        AND Price>10
```

OR *:* this operator joins two or more conditions, but returns a row if ANY of the conditions listed are true for that row’s data.

Example (Q8):

SELECT SupplierID, SupplierName
FROM Suppliers
WHERE (SupplierID>2 AND SupplierID<10) OR (Country= “USA”)

*Note that Supplier ID 2 is pulled because of its country *only*. (2 is not greater than 2.)

Example (Q9)

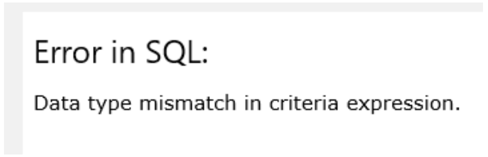
```
SELECT SupplierID, SupplierName
FROM Suppliers
WHERE SupplierID=2 OR SupplierID=5 OR
SupplierID=8
```

***IN**:* used to test whether or not a value (stated before the keyword IN) is "in" the list of values provided after the keyword IN

Example (Similar to Q10): Write a query to show SupplierID and SupplierName of suppliers with SupplierID equal to 1, 2, or 3.

```
SELECT SupplierID, SupplierName
FROM Suppliers
WHERE SupplierID IN ('1', '2', '3')
```

Just in case that you may still receive an error like



Error in SQL:

Data type mismatch in criteria expression.

Please try WHERE SupplierID IN (1, 2, 3)

***NOT IN**:* Excludes the items listed.

```
WHERE SupplierID NOT IN ('1', '2', '3')
```

BETWEEN*:* Used to return data within a range.

Example (Similar to Q11)

Write a query to show SupplierID and SupplierName of suppliers with SupplierID between 1 and 5

```
SELECT SupplierID, SupplierName
FROM Suppliers
WHERE SupplierID BETWEEN 1 AND 5
```

Note: using the BETWEEN clause includes the numbers in the range (1 and 5 in this example)

NOT BETWEEN*:* Used to return data NOT within the specified range.

```
WHERE SupplierID NOT BETWEEN 3 AND 5
```

LIKE: allows you to select only rows that are “like” what you specify

Example (Q12) : You want to pull all customers’ cities that start with “S” and postal codes that start with “9”.

```
SELECT CustomerName, City, PostalCode
FROM Customers
WHERE City LIKE ‘S%’ AND PostalCode LIKE ‘9%’
```

Note: When specifying specific data single quotations are used. The percent sign "%" can be used as a wild card to match any possible character that might appear *before* or *after* the characters specified. Capitalization does matter here.

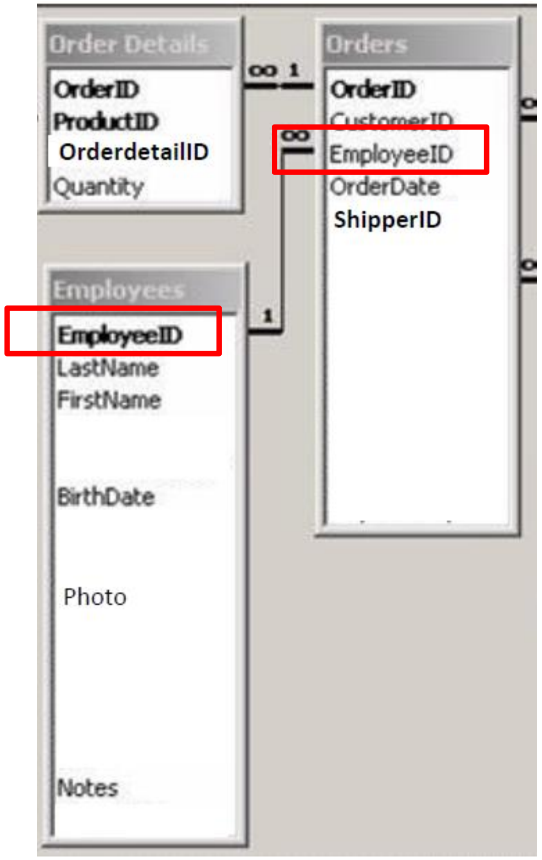
TABLE JOINS

Table joins are a must when you are wanting to incorporate attributes from more than one table into a single query.

A table join is inserted into the WHERE or JOIN ON clause, which involves the **attribute(s) that each of the tables share** (primary key of at least one table).

Example (Q13): You want to see a list of order IDs and which employees placed each order (organized alphabetically by employee last name).

```
SELECT orderID, LastName, FirstName
FROM orders, employees
WHERE employees.employeeID=orders.employeeID
ORDER BY LastName
```



Notice the join made in the WHERE clause because attributes in the SELECT statement are from two separate tables.
In the WHERE clause, we want to put the table name before the attribute name because both tables share the same attribute.

Or you can use JOIN Statement

```
SELECT orderID, LastName, FirstName
FROM orders
JOIN employees
ON employees.employeeID=orders.employeeID
ORDER BY LastName
```

A SQL query goes into a bar, walks up to two tables and asks, "Can I join you?"

How many orders does the first employee (LastName=Buchanan) processed? You can either manually count the number of orders or use the clause count(OrderID) in your query.

```
SELECT LastName, FirstName, count(OrderID)
FROM orders, employees
WHERE employees.employeeID=orders.employeeID
GROUP BY LastName
```

```
SELECT LastName, FirstName, count(OrderID)
FROM orders, employees
WHERE employees.employeeID=orders.employeeID AND
LastName='Buchanan'
```

JOIN Types

In SQL, a **join** is used to compare and combine — literally join — and return specific rows of data from two or more tables in a database. An **inner join** finds and returns matching data from tables, while an **outer join** finds and returns matching data *and* some dissimilar data from tables (Source diffen.com).

Inner Join is the default SQL join you get when you use the JOIN keyword by itself.

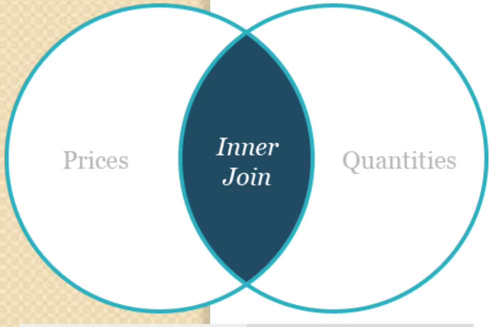


TABLE 1: PRICES

PRODUCT	PRICE
Potatoes	$3
Avocados	$4
Kiwis	$2
Onions	$1
Melons	$5
Oranges	$5
Tomatoes	$6

TABLE 2: QUANTITIES

PRODUCT	QUANTITY
Potatoes	45
Avocados	63
Kiwis	19
Onions	20
Melons	66
Broccoli	27
Squash	92

```
SELECT Prices.*, Quantities.Quantity
FROM Prices INNER JOIN Quantities
ON Prices.Product = Quantities.Product;
```

QUERY RESULT FOR INNER JOIN

PRODUCT	PRICE	QUANTITY
Potatoes	$3	45
Avocados	$4	63
Kiwis	$2	19
Onions	$1	20
Melons	$5	66

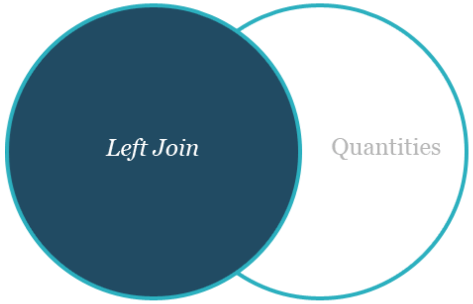


TABLE 1: PRICES

PRODUCT	PRICE
Potatoes	$3
Avocados	$4
Kiwis	$2
Onions	$1
Melons	$5
Oranges	$5
Tomatoes	$6

TABLE 2: QUANTITIES

PRODUCT	QUANTITY
Potatoes	45
Avocados	63
Kiwis	19
Onions	20
Melons	66
Broccoli	27
Squash	92

```
SELECT Prices.*, Quantities.Quantity
FROM Prices LEFT OUTER JOIN Quantities
ON Prices.Product = Quantities.Product;
```

QUERY RESULT FOR LEFT OUTER JOIN

PRODUCT	PRICE	QUANTITY
Potatoes	$3	45
Avocados	$4	63
Kiwis	$2	19
Onions	$1	20
Melons	$5	66
Oranges	$5	NULL
Tomatoes	$6	NULL

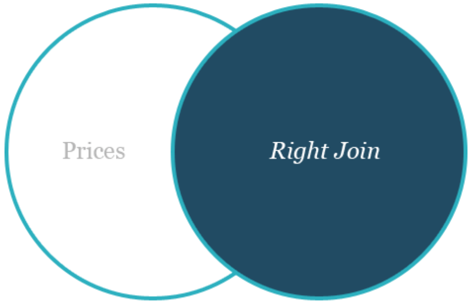


TABLE 1: PRICES

PRODUCT	PRICE
Potatoes	$3
Avocados	$4
Kiwis	$2
Onions	$1
Melons	$5
Oranges	$5
Tomatoes	$6

TABLE 2: QUANTITIES

PRODUCT	QUANTITY
Potatoes	45
Avocados	63
Kiwis	19
Onions	20
Melons	66
Broccoli	27
Squash	92

```
SELECT Prices.*, Quantities.Quantity
FROM Prices RIGHT OUTER JOIN Quantities
ON Prices.Product = Quantities.Product;
```

QUERY RESULT FOR RIGHT OUTER JOIN

PRICE	PRODUCT	QUANTITY
$3	Potatoes	45
$4	Avocados	63
$2	Kiwis	19
$1	Onions	20
$5	Melons	66
NULL	Broccoli	27
NULL	Squash	92

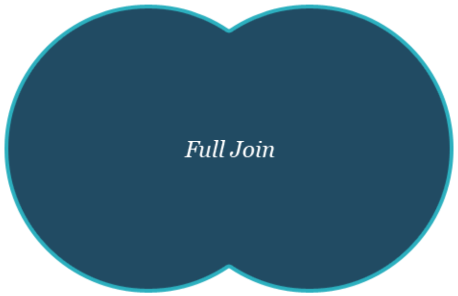


TABLE 1: PRICES

PRODUCT	PRICE
Potatoes	$3
Avocados	$4
Kiwis	$2
Onions	$1
Melons	$5
Oranges	$5
Tomatoes	$6

TABLE 2: QUANTITIES

PRODUCT	QUANTITY
Potatoes	45
Avocados	63
Kiwis	19
Onions	20
Melons	66
Broccoli	27
Squash	92

```
SELECT Prices.*, Quantities.Quantity
FROM Prices FULL OUTER JOIN Quantities
ON Prices.Product = Quantities.Product;
```

QUERY RESULT FOR FULL OUTER JOIN

PRICES.PRODUCT	PRICE	QUANTITIES.PRODUCT	QUANTITY
Potatoes	$3	Potatoes	45
Avocados	$4	Avocados	63
Kiwis	$2	Kiwis	19
Onions	$1	Onions	20
Melons	$5	Melons	66
Oranges	$5	NULL	NULL
Tomatoes	$6	NULL	NULL
NULL	NULL	Broccoli	27
NULL	NULL	Squash	92

Thanks!
I hope you have a better understanding of Basic SQL Querying