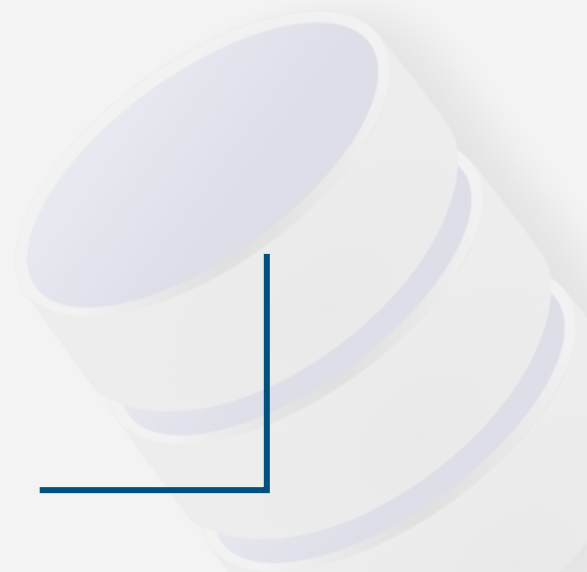




CS 3200 - Fall 2024

SQL - Part 1

Mark Fontenot, PhD
Northeastern University



Setting the Stage...

- **Relational Algebra** → query language for set of relations based on Relational Data Model
- The **relational data model** forms the foundation for modern **relational database management systems** (RDBMS) such as MySQL, Postgres, Oracle DB, MS SQL Server, etc.
- Each RDBMS provides a query language that is a “flavor” of the **Structured Query Language** (SQL) standard.

SQL: Structured Query Language

- SQL is a standardized language* used for managing and manipulating relational databases.
- SQL is mostly a declarative language → say what you want rather than how to get it
- but ... many RDBMS vendors add procedural extensions to their SQL implementations
 - PL/SQL from Oracle
 - T-SQL from Microsoft SQL Server
 - ...

* See the [Standards Overview](#) section of the SQL Wikipedia Article

Categories of SQL Commands

- Major Categories of SQL Commands:
 - **DQL** (Data Query Language)
 - SELECT Statements
 - **DDL** (Data Definition Language)
 - CREATE, ALTER, and DROP Statements
 - **DML** (Data Manipulation Language)
 - INSERT, UPDATE, and DELETE Statements
 - *some resources break it down even further*



SQL vs. Relational Algebra

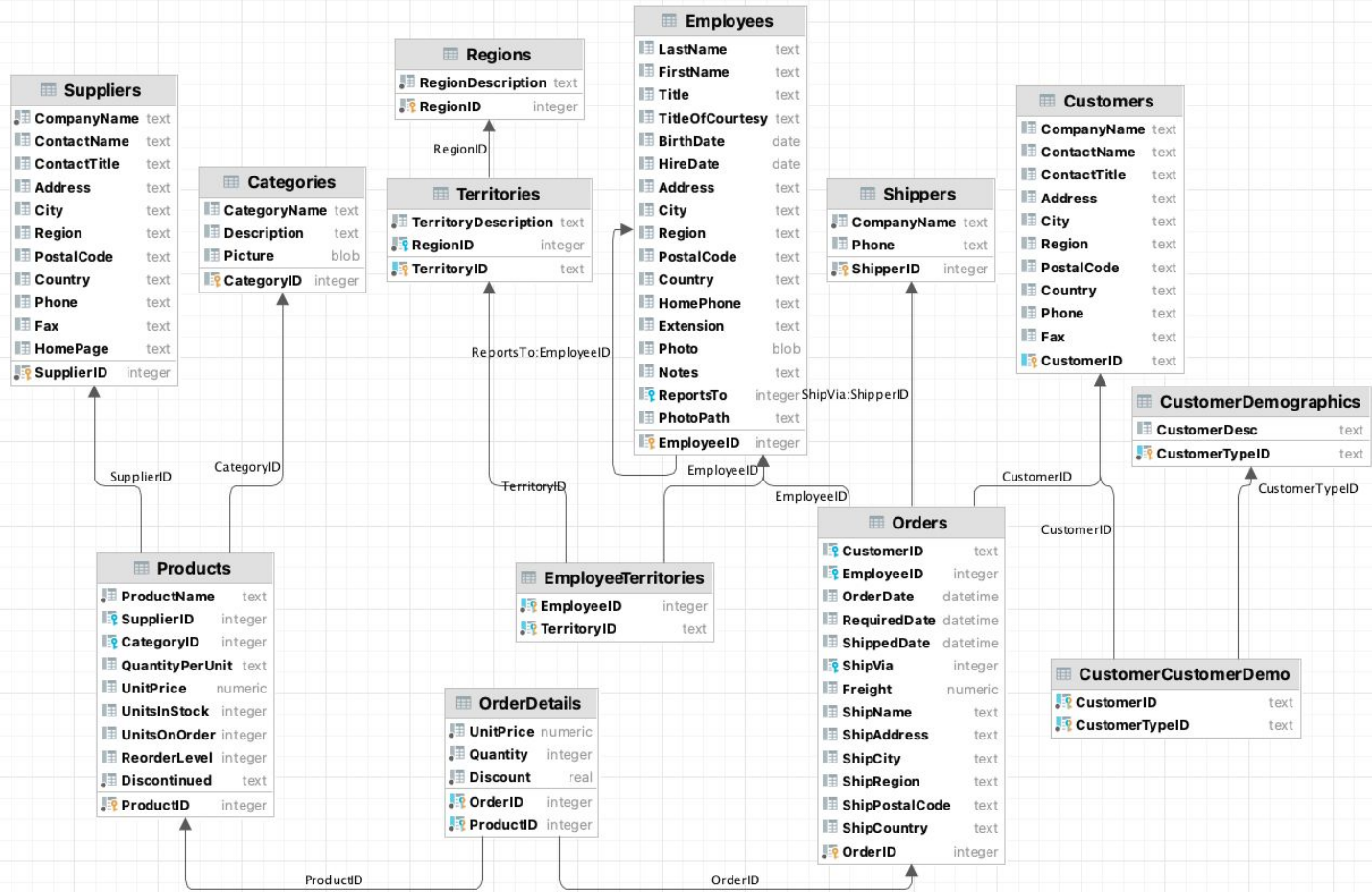
- SQL is based on ***multisets*** where RA is based on ***sets***.
- Data is always in some order in SQL where order doesn't matter in RA.
- SQL has implementation-specific limits on max size of a row, max size of a table, etc. where RA has no such limits
 - details are RDBMS-product specific

SQL vs. Relational Algebra

- Terminology Differences

Relational Algebra	SQL
Relation (Relation Instance)	Table
Relation Schema	Table Schema
Attribute	Attribute or Column
Tuple	Row
Primary Key	Primary Key
Foreign Key	Foreign Key

North Wind DB Schema



It's SQL Time: The SELECT Statement



Aside: Generic Syntax Statements

- Explanation of Generic Syntax in upcoming Slides
 - `<...>` - Placeholder
 - Indication that you should replace the placeholder (including the `<>`s) with something specific
 - Example: `SELECT <value>;` indicates that `<value>` should be replaced with an actual value, as in `SELECT 123;`
 - `[...]` - Optional
 - Indication that the component(s) enclosed in `[...]` are not always needed for a statement to be valid.
 - Example: `SELECT <value> [FROM <table>];` indicates either of the following are syntactically valid:
 - `SELECT 123;`
 - `SELECT 123 FROM categories;`

Intro to SELECTs

- **SELECT** statement is the querying “powerhouse” of SQL.
 - includes ways to filter horizontally and vertically, combine tables, aggregate by groups and more!
- The (almost) simplest SELECT statement

```
SELECT FirstName  
FROM Employees;
```

- Retrieves all FirstName values from table Employees.
LOW - it returns the FirstName column.

Intro to SELECTs

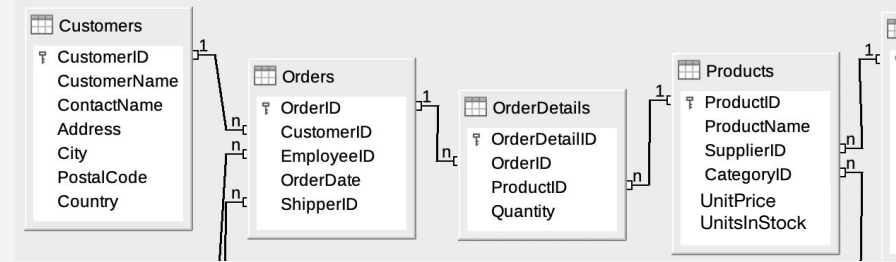
- **General Form** of a Simple **SELECT** Statement:

```
SELECT <col1>[, <col2>, ...]  
FROM <table name>;
```

What this does: retrieves a table with attributes <col1>, <col2>, ... from the identified table.

Note: I will capitalize keywords in SQL statements, but not strictly required by SQL.

SELECT Specific Columns



```
SELECT ProductID, ProductName
FROM Products;
```

Returns a table containing the **ProductID** and **ProductName** columns from the **Products** table.

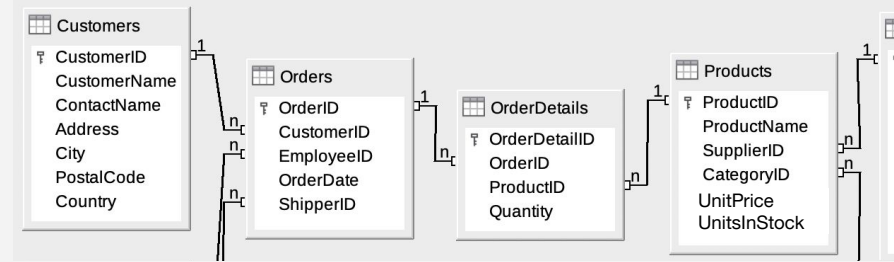
The screenshot shows a SQL playground interface with the following query and output:

```
1 ✓ SELECT ProductID, ProductName
2 FROM Products;
```

Output: main.Products x

ProductID	ProductName
1	Chai
2	Chang
3	Aniseed Syrup
4	Chef Anton's Cajun Seasoning
5	Chef Anton's Gumbo Mix
6	Grandma's Boysenberry Spread
7	Uncle Bob's Organic Dried Pears
8	Northwoods Cranberry Sauce
9	Mishi Kobe Niku
10	Ikura
11	Queso Cabrales
12	Queso Manchego La Pastora
13	Konbu
14	Tofu
15	Genen Shouyu

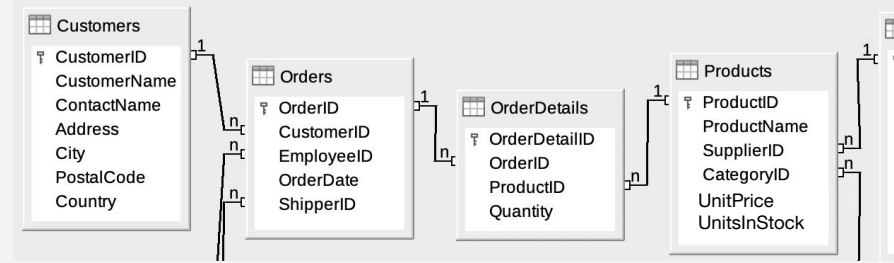
Retrieve All Columns



```
SELECT *  
FROM Products;
```

- Returns all columns and rows from the Products table
 - *Basically, returns the complete Products table*
- * is a **wildcard** character... in this case, it means “all columns”

Renaming/Aliasing Columns



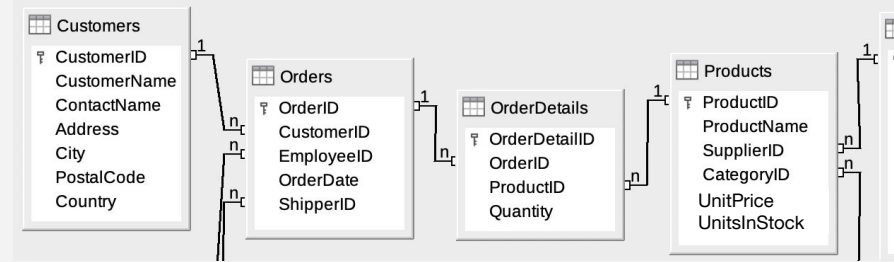
General
Form

```
SELECT <column> AS <newName>, ...  
FROM <table name>;
```

Example

```
SELECT ProductID AS p_id,  
        ProductName AS p_name  
FROM Products;
```

Renaming/Aliasing Columns



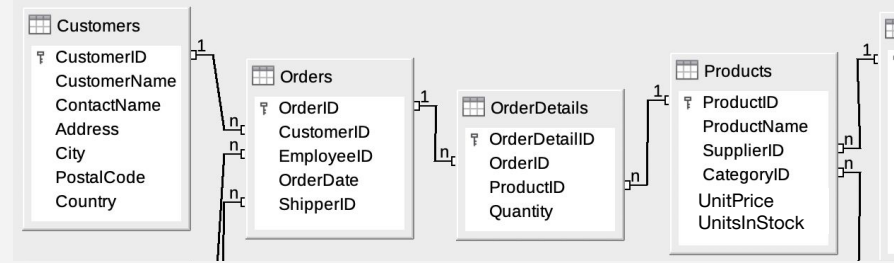
```
SELECT ProductID AS p_id,  
       ProductName AS p_name  
FROM Products;
```

Returns a table with
columns named **p_id** and
p_name.

The screenshot shows the SQL Server Output window with the following data:

	p_id	p_name
1	1	Chai
2	2	Chang
3	3	Aniseed Syrup
4	4	Chef Anton's Cajun Seasoning
5	5	Chef Anton's Gumbo Mix

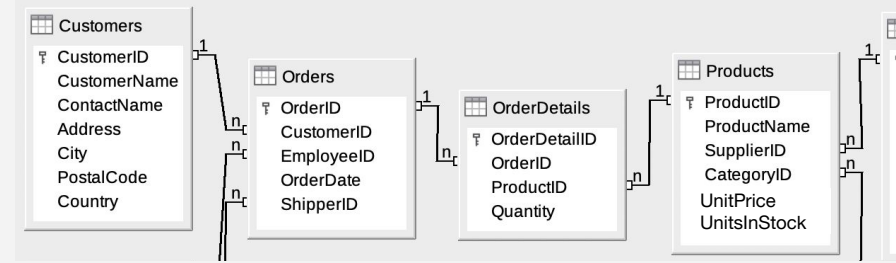
Basic Calculations



- You can perform mathematical operations between existing attributes in the table(s).
 - Note that functions available depend on the RDBMS
 - You should alias the resulting calculated attribute with **AS**.

```
SELECT OrderID, ProductID,  
        (UnitPrice * Quantity) AS cost  
FROM OrderDetails;
```

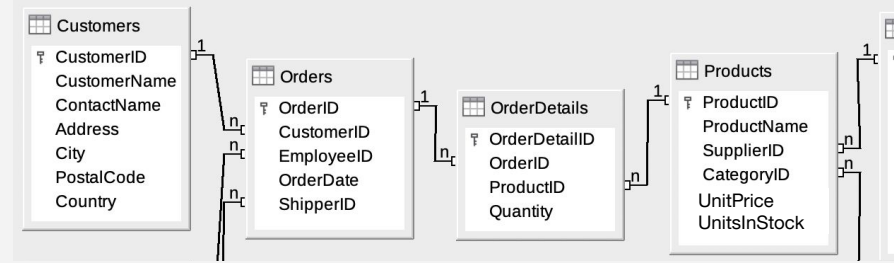

Sample Query 1



For all customers, provide a list containing the customer name, contact person's name, their city, and postal code.



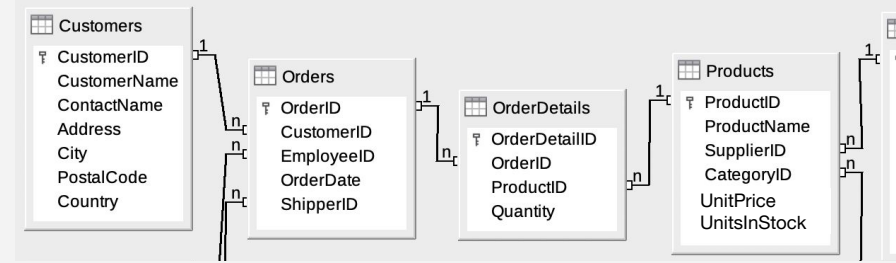
Sample Query 2



Your boss sends you the following email... *“Hi! Could you get me a list of all products with our current stock levels? Thanks!”*



Sample Query 3



The stockroom manager needs to know the current value of each product we currently have in stock. So, if we have 10 units of Sharp Cheddar, what is the total value for those 10 units. Can you help them?

Selecting Specific Rows with WHERE



WHERE -

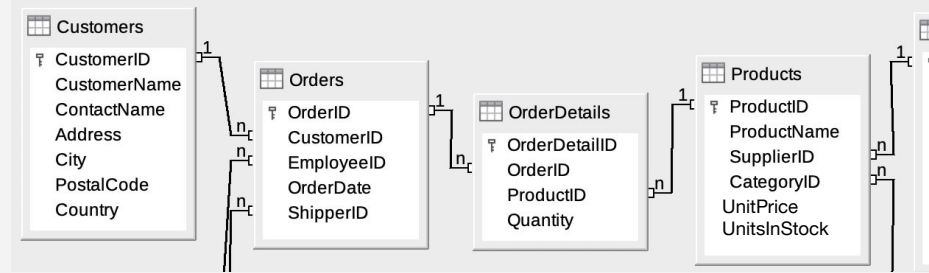
- **WHERE clause** - includes predicates applied to every row to determine if it is returned or not.
 - Each condition should evaluate to *true* or *false*.

General
Form

```
SELECT <list of columns>  
FROM <table name>  
[WHERE <condition list>]
```

Why condition list?

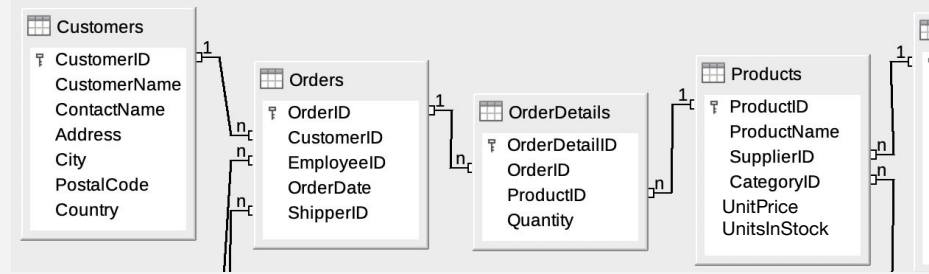
WHERE Examples



- Give the names and unit prices of products that are less than \$15.

```
SELECT ProductName, UnitPrice
FROM Products
WHERE UnitPrice < 15;
```

WHERE Examples



- Give the names of any employees hired on or after Jan 1 2014.

```
SELECT LastName, FirstName, HireDate  
FROM Employees  
WHERE HireDate >= '2014-01-01';
```

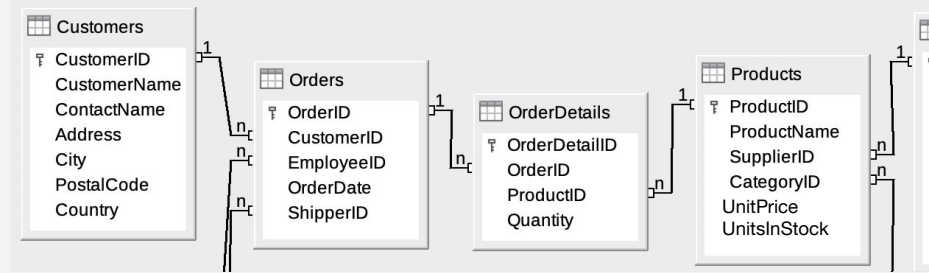
Notice the way we format a date literal.

Boolean Expression Operators

Operator	Meaning	Example
=	Equal to	InvoiceId = 2
<>	Not equal to	Name <> 'U2'
< or >	Less/Greater than	UnitPrice < 5
<= or >=	Less/Greater than or equal to	UnitPrice >= 0.99
LIKE	Matches pattern	PostalCode LIKE 'T2%'
IN	Within a set	City IN ('Calgary', 'Edmonton')
IS or IS NOT	Compare to NULL	ReportsTo IS NULL
BETWEEN	Inclusive range (esp. dates)	UnitPrice BETWEEN 0.99 AND 1.99

- Boolean expressions can be combined with *logical* **AND** or **OR**.

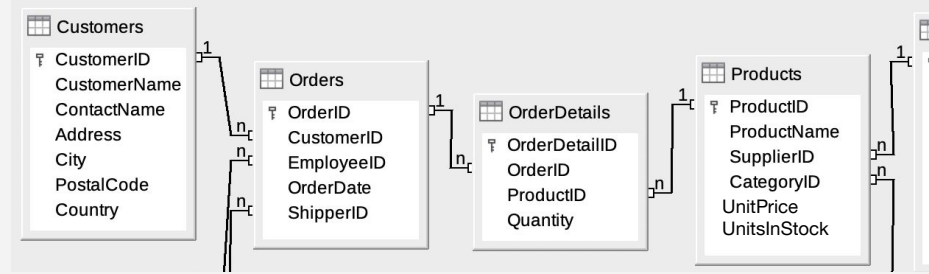
WHERE Examples



- Give the names and unit prices of products from supplier with id 7 that are less than \$15..

```
SELECT ProductName, UnitPrice
FROM Products
WHERE SupplierID = 7 AND UnitPrice < 15 ;
```

WHERE Examples



- Give a list of all orders placed by customer with id 5 or customer with id 12. Only return OrderID.

```
SELECT OrderID
FROM Orders
WHERE CustomerID = 5 OR CustomerID = 12;
```

WHERE - IN operator

- Check if attribute value is within an enumerated list of values

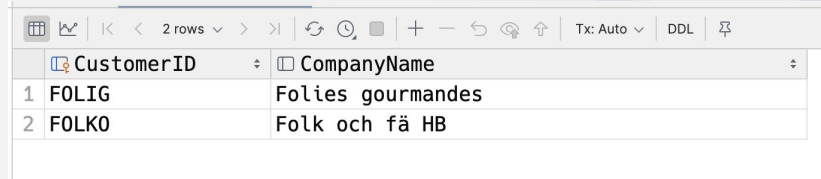
- The boss wants a list of our customers' company names and contact names for all customers in the British Isles or Central America.

```
SELECT CompanyName, ContactName  
FROM Customers  
WHERE Region IN ('British Isles', 'Central America');
```

WHERE - LIKE Operator

- LIKE operator is used for pattern matching in strings
 - **<column|expression> LIKE '<pattern>'**
 - **<pattern>** may contain wildcard characters
 - **%** - matches any sequence of zero or more characters in the string
 - **_** (underscore) - matches any single character in the string
 - Any other character in **<pattern>** matches itself or its lower/upper case equivalent (i.e. case-insensitive matching)

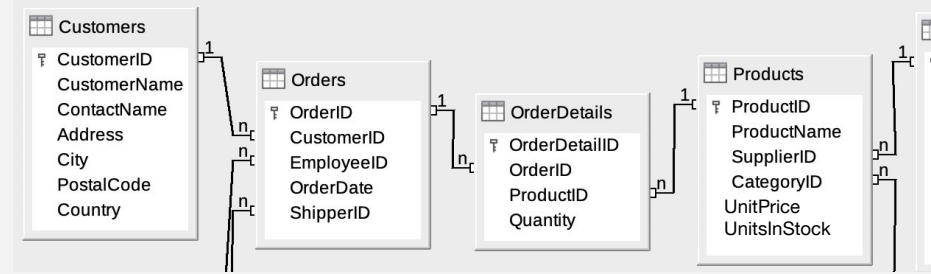
```
SELECT CustomerID, CompanyName
FROM Customers
WHERE CompanyName LIKE 'fo%';
```



The screenshot shows a database query result with two columns: CustomerID and CompanyName. The first row has CustomerID 1 and CompanyName 'FOLIG', with a corresponding result of 'Folies gourmandes'. The second row has CustomerID 2 and CompanyName 'FOLKO', with a corresponding result of 'Folk och få HB'. The interface includes navigation buttons and a status bar at the top.

CustomerID	CompanyName	
1	FOLIG	Folies gourmandes
2	FOLKO	Folk och få HB

WHERE - Your Turn



1. Provide a list of products from the supplier with id of 12 that cost more than \$25.



WHERE - Your Turn

2. Customer relations wants a list of all customer info for any customer in Boston or New York (cities).



Adding the ORDER BY Clause



Sorting the Results

Specify the ordering of the rows in the output.

General Form

```
SELECT <list of columns>  
FROM <table name>  
[WHERE <condition list>]  
[ORDER BY <column-order list>];
```

For each attribute in order by clause, **ASC** (default) or **DESC** can be specified.

Sorting the Results

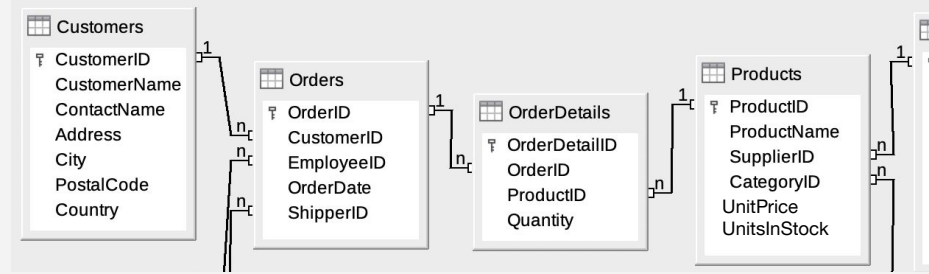
Specify the ordering of the rows in the output.

General Form

```
SELECT <list of columns>  
FROM <table name>  
[WHERE <condition list>]  
[ORDER BY <column-order list>];
```

If **<column-order list>** contains more than one column, the first column listed is the primary, and the next column is used to break ties when the value in the primary column are equal.

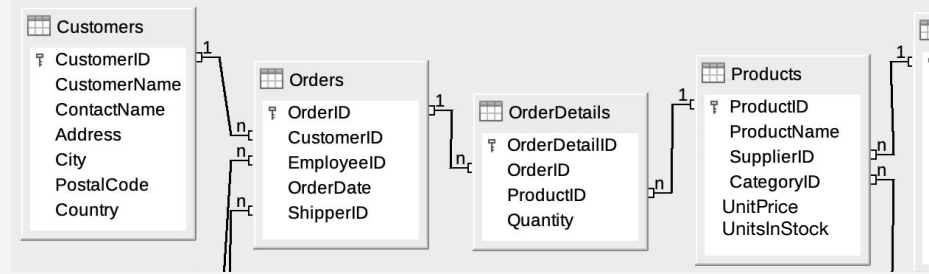
Output Order Example



- Give the names, unit prices, and number of units in stock for any products we are currently waiting to receive. Sort the list by product name in reverse alphabetical order.

```
SELECT ProductName, UnitPrice, UnitsOnOrder
FROM Products
WHERE UnitsOnOrder > 0
ORDER BY ProductName DESC;
```

Output Order Example

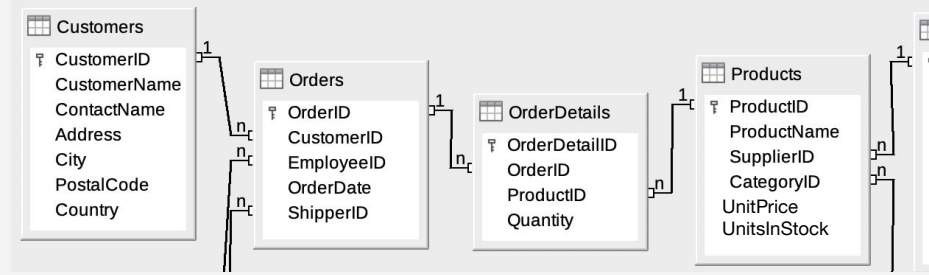


- Provide a list of our customers' company names with city, country, and region sorted by company name within each region.

```
SELECT CompanyName, City, Country, Region  
FROM Customers  
ORDER BY Region, CompanyName;
```

Notice.... you aren't required to have a WHERE clause.

Removing Duplicates



- Use the **SELECT DISTINCT** to remove duplicate **ROWS** from the result
 - Note: Only put DISTINCT once, not for each attribute
- **Provide a list of all Countries where Northwind currently has customers. Sort the result alphabetically.**

```
SELECT DISTINCT Country  
FROM Customers  
ORDER BY Country ASC;
```

What do you think?

What do you think the potential difference between the output of these two queries?

```
SELECT ShipCity, ShipCountry
from Orders
where shipregion = 'Western Europe'
order by shipcity, shipcountry;
```

versus

```
SELECT DISTINCT ShipCity, ShipCountry
FROM Orders
WHERE shipregion = 'Western Europe'
ORDER BY shipcity, shipcountry;
```

Aggregate Functions



Aggregate Functions

- Allows us to apply some function to all rows in the result set of a query
- When used alone, resultset will be a single row
- Common aggregate functions:

- **MAX**
- **MIN**
- **SUM**
- **AVG**
- **COUNT**

```
SELECT MAX(UnitPrice)  
FROM Products;
```

```
SELECT COUNT(*)  
FROM Products;
```

Don't forget... you can alias these to more readable column names for the output.

Aggregate Function Practice

What's the average price of all products we sell?

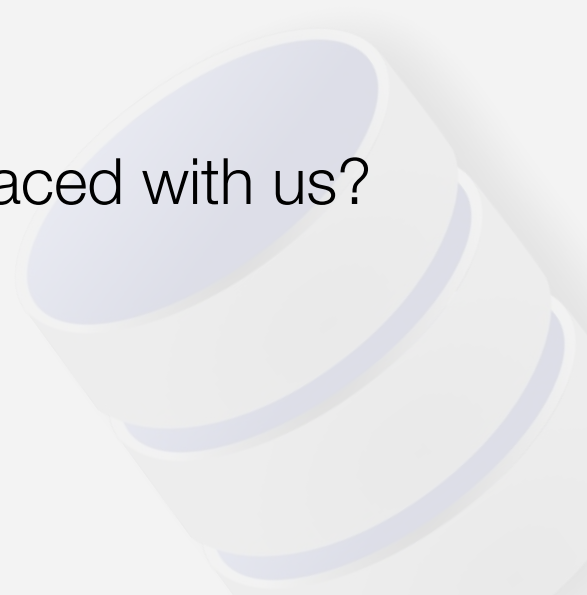
What's the average price of all non-discontinued products we sell?



Aggregate Function Practice

What is the date of the first order we ever processed?

How many orders has “Alfreds Futterkiste” placed with us?



SET Operations



Union

Provide a list of all cities in Western Europe where either employees live or customers are based.

```
SELECT City
FROM Employees
WHERE Region = 'Western Europe'
UNION ALL
SELECT City
FROM Customers
WHERE Region = 'Western Europe';
```

Includes duplicates

Use **UNION** (without ALL) to remove dupes

Intersection

Provide a list of all cities in the USA where we have **BOTH** employees and customers.

```
SELECT City
FROM Employees
WHERE Country = 'USA'
  INTERSECT
SELECT City
FROM Customers
WHERE Country = 'USA';
```



There's no ALL version
with Intersect

Except

Provide a list of all cities in the USA where Employees live but where we have no customers.

```
SELECT City
FROM Employees
WHERE Country = 'USA'
  EXCEPT
SELECT City
FROM Customers
WHERE Country = 'USA';
```

