SQL Basics

“Structured Query Language”

- It is NOT much like other programming languages
- It is NOT PROCEDURAL
- It does not process one record at a time, rather, it is a SET processing language
- All inputs to SQL are tables
- The output from a query is a table
- Output from a query referred to as the “Answer Set”
- Some queries may produce “interim” temporary answer sets

SQL Basics

"Structured Query Language"

- It is a relatively simple language – brief syntax, few commands
- It is a relatively powerful language – a FEW lines of code can accomplish a LOT of work
- ANSI (American National Standards Institute) maintains a specification for "standard" SQL
- Each DBMS manufacturer follows the ANSI standard, but also adds extended features unique to their SQL

SQL Basics

Useful Tool for Managing your Databases

- “SQLYog” from WebYog
- The community edition is free

- https://github.com/webyog/sqlyog-community/wiki/Downloads

SQL Basics

USE statement

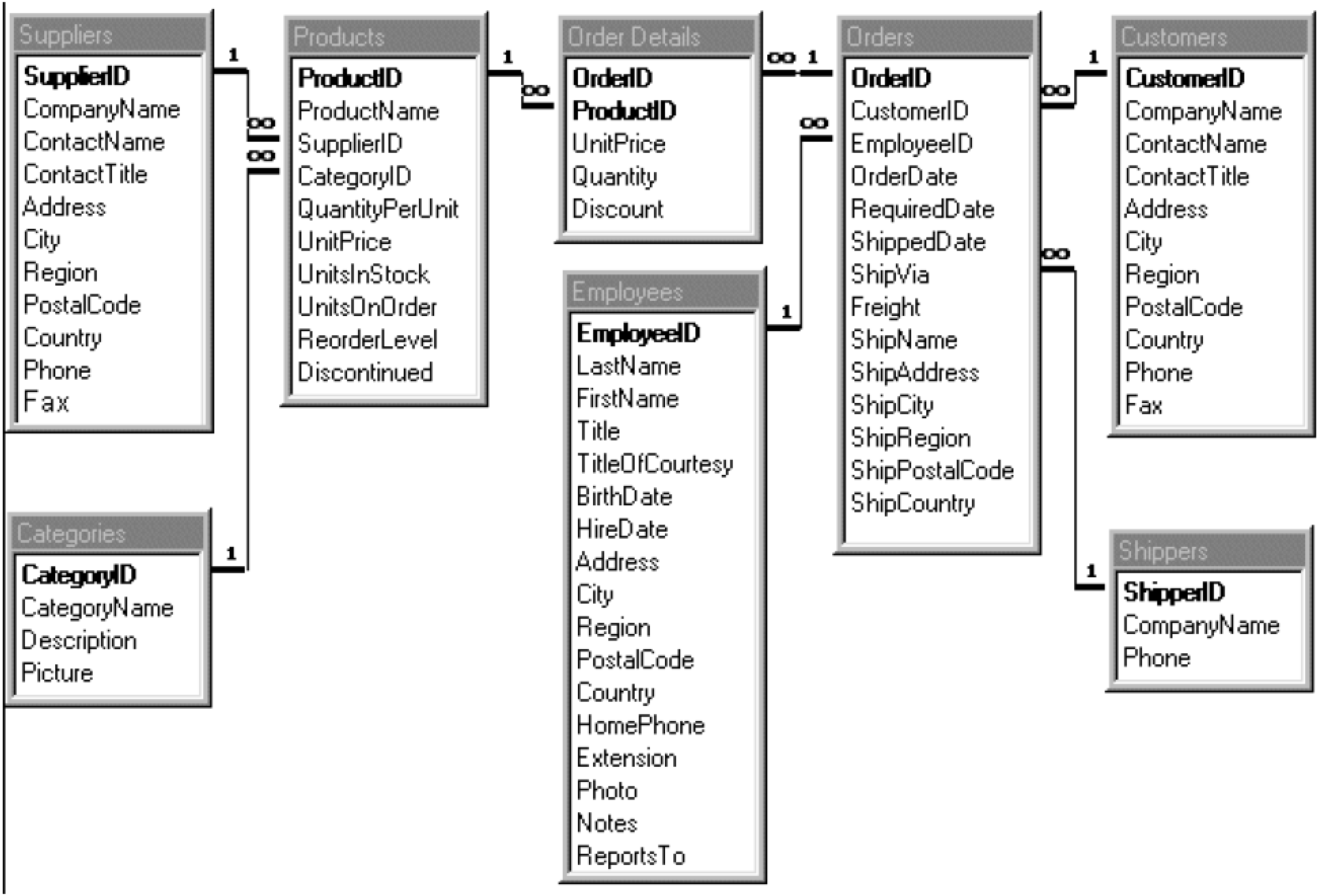
```
USE <database>;
```

- Tells the Query Engine which database you want to use for your query

SQL Basics

SELECT statement

```
SELECT <column1>, <column2>, <column3>,
<literal>, <math expression>
   FROM  <table A> ;
```



Suppliers

SupplierID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax

Categories

CategoryID
CategoryName
Description
Picture

Products

ProductID
ProductName
SupplierID
CategoryID
QuantityPerUnit
UnitPrice
UnitsInStock
UnitsOnOrder
ReorderLevel
Discontinued

Order Details

OrderID
ProductID
UnitPrice
Quantity
Discount

Employees

EmployeeID
LastName
FirstName
Title
TitleOfCourtesy
BirthDate
HireDate
Address
City
Region
PostalCode
Country
HomePhone
Extension
Photo
Notes
ReportsTo

Orders

OrderID
CustomerID
EmployeeID
OrderDate
RequiredDate
ShippedDate
ShipVia
Freight
ShipName
ShipAddress
ShipCity
ShipRegion
ShipPostalCode
ShipCountry

Customers

CustomerID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax

Shippers

ShipperID
CompanyName
Phone

SQL Basics

Examples

```
select *
     from  nwEmployees;

use NorthWinds;

select EmployeeID, LastName, FirstName
     from nwEmployees;
```

SQL Basics

SELECT statement

- Literals may be either `'Character'` (in quotes) or Numeric
- Math expressions

  Only use with columns defined as numeric data types

+	Add
-	Subtract
*	Multiply
/	Divide
**	Exponent

SQL Basics

SELECT statement

- Rename a column in the answer set with “AS”

```
    Select employeeID as 'EMP'
```

- Concatenate character columns with (MySQL only)

```
    Concat(<column1>,column2>)
```

- Comment out a line or part of a line of code by prefixing it with “- -” or embedding a “#”
  - In Yog, highlight code then `<ctrl>+<shift>+c` to comment `<ctrl>+<shift>+r` to remove comment
- Limit the size of the answer set with “limit” (MySQL)

```
Select LastName, FirstName
    from nwEmployees limit 100;
```

SQL Basics

Examples

```
select 'Roster', LastName, FirstName
     from nwEmployees;
select 'Roster' as 'Type', LastName, FirstName
     from nwEmployees;
select 22, LastName, FirstName
     from nwEmployees;
select 2 * 2, LastName, FirstName
     from nwEmployees;
select concat(FirstName,' ',LastName)
     from nwEmployees;
```

SQL Basics

SELECT statement

```
SELECT <column1>, <column2>, <column3>,
<literal>, <math expression> AS <label>
  FROM  <table A>
  WHERE  <condition> ;
```

SQL Basics

- The WHERE clause results in a subset of ROWs to appear in the answer set
- The condition in the WHERE clause takes this format:

  < operand > < operator > < operand >
- Operands may be columns or literals or expressions
- Operator may be

=	Equals	Like
<>	Not equals	Between
>	Greater than	In
<	Less than	

SQL Basics

- Operator may be: In or Like

  In (literal, literal, literal)

  Like `'string'` with % or _ as a wildcard
- Multiple conditions may be joined with Boolean operators

  `AND, OR`
- Conditions may be negated with Boolean operator

  `NOT`
- Answer Set rows may be sorted with “Order By”
- Order By defaults to Ascending, can specify DESC

SQL Basics

Distinct:

- The answer set may contain duplicate rows
- The “`distinct`” keyword before a column removes duplicates
- Example:
  - 87 Customers, each one has a country
  - How many distinct countries are they from?

SQL Basics

Examples

```
select Customerid, ContactName, Region, Country
      from nwCustomers;
select Customerid, ContactName, Region, Country
      from nwCustomers
      where Country = 'Brazil';
select Customerid, ContactName, Region, Country
      from nwCustomers
      where Country <> 'Brazil';
select ProductID, ProductName, UnitPrice
      from nwProducts
      where UnitPrice > 60;
```

SQL Basics

Examples

```
select ProductID, ProductName, UnitPrice
      from nwProducts
      where UnitPrice between 20 and 30;
select ProductID, ProductName, categoryid, UnitPrice
      from nwProducts
      where UnitPrice between 20 and 30
      and categoryid in (2, 4, 6);
select ProductID, ProductName, QuantityPerUnit
      from nwProducts
      where QuantityPerUnit like '%jars%';
```

SQL Basics

Examples: Using distinct

```
Select CompanyName, ContactName, Country
      from nwCustomers;
Select Country
      from nwCustomers;
Select Distinct Country
      from nwCustomers;
```

Handling Dates in MySQL

- MySQL supports DATE, DATETIME, and TIMESTAMP data types
- Columns with a data type of “TIMESTAMP” are stored as a 4-byte binary integer representing the number of seconds since 1970-01-01 00-00-00 UTC. TIMESTAMP has a range of '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.
- If no value is provided for the TIME portion of a DATETIME column, it defaults to 00:00.00.0000
- To make it easier for humans to deal with date/time, MySQL allows us to reference dates/times in this format:

  YYYY-MM-DD and HH:MM.SS.nnn
- If you pass the date to MySQL as text in YYYY-MM-DD format, it will automatically convert it to the proper binary number
- If you pass the time to MySQL as text in HH:MM.SS.nnn format, it will automatically convert it to the proper binary number

Handling Dates in MySQL

YYYY-MM-DD and hh:mm.ss.nnn

- YYYY is four digits from 1000 through 9999 that represent a year.
- MM is two digits, ranging from 01 to 12, that represent a month in the specified year.
- DD is two digits, ranging from 01 to 31 depending on the month, that represent a day of the specified month.
- hh is two digits, ranging from 00 to 23, that represent the hour.
- mm is two digits, ranging from 00 to 59, that represent the minute.
- ss is two digits, ranging from 00 to 59, that represent the second.
- nnn is zero to three digits, ranging from 0 to 999, that represent the fractional seconds.

Examples: Using DATES

```
Select Now();
Select Curdate();
Select Curtime();
Select Lastname, Firstname, Extract(Year From HireDate) AS HireYear
       from NWEmployees;
SELECT EmployeeID, Lastname, Firstname,
       ROUND(DATEDIFF(HireDate, BirthDate)/365,0)   AS HIRE_AGE
    FROM NWEmployees;
```

Examples: Using DATES

```
SELECT DATE_FORMAT(HireDate,'%b %d %Y %h:%i %p')FROM NWEMPLOYEES;
```

Format	Description
%a	Abbreviated weekday name (Sun-Sat)
%b	Abbreviated month name (Jan-Dec)
%c	Month, numeric (0-12)
%D	Day of month with English suffix (0th, 1st, 2nd, 3rd, �)
%d	Day of month, numeric (00-31)
%e	Day of month, numeric (0-31)
%f	Microseconds (000000-999999)
%H	Hour (00-23)
%h	Hour (01-12)
%I	Hour (01-12)
%i	Minutes, numeric (00-59)
%j	Day of year (001-366)
%k	Hour (0-23)
%l	Hour (1-12)

http://www.w3schools.com/sql/func_date_format.asp

SQL Basics

SELECT statement

```
SELECT <column1>, <column2>, <column3>, <literal>,
<math expression>
   FROM  <tableA>
   WHERE  <condition>
ORDER BY  <column1>, <column2>  [DESC] ;
```

SQL Basics

SQL provides the following GROUP FUNCTIONS

SUM – Provides the sum of the values in a column across many rows

AVG - Provides the average of the values in a column across many rows

COUNT – Provides a count of how many rows have a value in a column, counted across many rows

MIN - Provides the lowest value in a column across many rows

MAX - Provides the highest value in a column across many rows

SQL Basics

GROUP FUNCTIONS

- SUM, AVG must only be used with NUMERIC columns
- MIN, MAX can be used with any data type
- COUNT can be used with any column, or with a (*) to simply count rows
- Group functions require SQL to create an interim answer set, and then process the group function against the interim answer set, delivering a final answer set that contains only the final total for the function. Always returns an integer value.
- When you combine a GROUP FUNCTION with a WHERE clause, keep in mind that the WHERE clause simply reduces the number of rows in the INTERIM answer set before the GROUP function does its calculation.

SQL Basics

Examples –

```
select COUNT(*) as 'Total'
      from nwEmployees;
select COUNT(Distinct Country) as 'Countries'
      from nwCustomers;
select SUM(UnitPrice) as 'Total Price'
      from nwProducts;
select MAX(UnitPrice) as 'High Price'
      from nwProducts;
select MIN(UnitPrice) as 'Low Price'
      from nwProducts;
      where UnitPrice > 0
select AVG(UnitsInStock) as 'Average Inventory'
      from nwProducts;
```

GROUP BY

- Group functions process against an interim answer set to return a value across many rows.
- Using a GROUP BY clause enables SQL to provide subtotals. The GROUP BY tells SQL to perform the group function against a subset of rows in the interim answer set and provide a total for each subset of rows.

VERY IMPORTANT RULE

When using a GROUP BY every column in the SELECT statement must either be a GROUP FUNCTION or a COLUMN that you are grouping by.

Examples –

```
select Country, COUNT(*) as 'Total'
      from nwCustomers
      GROUP BY Country
select Country, COUNT(Country) as 'Total'
      from nwCustomers
      GROUP BY Country
```

Why are these the same?

```
select CustomerID, Country, COUNT(Country) as 'Total'
      from nwCustomers
      GROUP BY Country
```

Why is this incorrect?

SQL Basics

More Examples –

From which supplier does Northwinds carry the most inventory?

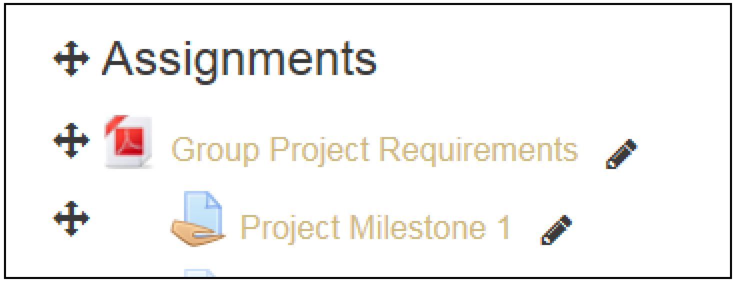
```
select SupplierID, SUM(UnitsInStock)as 'Inventory'
   from nwProducts
   group by SupplierID
```

In which month in 2013 did Northwinds ship the most orders?

```
 SELECT EXTRACT(MONTH FROM ShippedDate) AS 'Month',
   COUNT(OrderID) AS 'Orders'
   FROM nwOrders
   WHERE EXTRACT(YEAR FROM ShippedDate) = '2013'
   GROUP BY EXTRACT(MONTH FROM ShippedDate)
   ORDER BY 2 DESC
   Limit 1
```

Milestone 1:

- Link to submit is in Week Four (when the assignment was announced.)
- Looks like this:



Assignments

Group Project Requirements

Project Milestone 1

- Due Date: Friday, October 6, 5:00 p.m. (One submission per team)
- Link to your github repo: email to your TA. (One submission per team)
- Yes, I changed the requirement last week and added a summary of your application architecture

HAVING

- Is simply like a WHERE clause against the answer set when you use a GROUP BY

SQL Basics

Examples –

```
select Country, COUNT(*) as 'Total'
      from nwCustomers
      GROUP BY Country
```

In which countries does Northwinds have more than five customers?

```
select Country, COUNT(*) as 'Total'
      from nwCustomers
      GROUP BY Country
      HAVING  COUNT(*) > 5
      order by 2 desc
```

SubQuery

Simply: a query within a query. The answer set to an “inner” query is used as a predicate in a where clause in the “outer” query.

- The subquery must return only one column.
- If the outer query WHERE clause contains an “equals” condition, the subquery must return ONE row.
- If the outer query WHERE clause contains an “in” condition, the subquery may return multiple rows, presented as a list of values.
- The Subquery is embedded within parentheses
- Outer and inner queries can hit two different tables

SQL Basics

Examples –

```
select ProductID, ProductName, UnitPrice
   from nwProducts
   where UnitPrice = (

      select MAX(UnitPrice)
            from nwProducts )
```

Note that with the “equals” condition, the inner query returns only one value (one row, one column)

Examples –

```
select CustomerID, OrderID
   from nwOrders
   where OrderID in (

      select OrderID
            from nwOrderDetails
            where Quantity > 100 )

   Order by CustomerID
```

- Note that with the “in” condition, the inner query returns many values (many rows, one column) as a list
- Uses Two different tables

Getting Data from Multiple Tables – The JOIN

- In order to run a query that retrieves data from multiple tables, those tables must be **JOIN**ed.
- Joining two tables requires that the two tables have a common key (typically a foreign key relationship) that appears in both tables.
- The common key columns need NOT have the same name, but must be of the same data type and length.
- A JOIN is one of the most **resource intensive** activities one can do in a relational database.

Basic Example

- Let’s join nwOrders to nwEmployees
- nwOrders has 830 rows, each with an EmployeeID
- nwEmployees has 9 rows, each with an EmployeeID
- They have a common key: EmployeeID (primary key in nwEmployees; foreign key in nwOrders)
- We want SQL to join the rows in nwEmployees and nwOrders where the EmployeeID matches

Advanced SQL

Provide a listing showing Northwinds employees, sorted by LastName, and a count of each employee’s orders

```
Select LastName, Firstname, count(OrderID)as 'Orders'
      from nwEmployees, nwOrders
      where nwEmployees.EmployeeID =
            nwOrders.EmployeeID
      GROUP BY LastName, FirstName
      Order By 1
```

Qualifying the Column Names

- Since the column “EmployeeID” exists in BOTH tables in this query, when referring to EmployeeID, we need to tell SQL which one.
- Therefore, we suffix the table name in front of the column name separated by a “ . ”
- Failure to fully qualify the column name will result in an “ambiguous column” error

Advanced SQL

Alternative

To save some typing, we can define an “alias” for each table. We can temporarily – only for the duration of this query -- rename the nwEmployees table “E”, and rename the nwOrders table “O”.

```
Select LastName, Firstname, count(OrderID) as 'Orders'
      from nwEmployees E, nwOrders O
      where E.EmployeeID = O.EmployeeID
      GROUP BY LastName, FirstName
      Order By 1
```

Advanced SQL

Alternative

- SQL allows another syntax option for doing the JOIN.
- These queries are equivalent:

```
Select LastName, Firstname, count(OrderID) as 'Orders'
      from nwEmployees E, nwOrders O
      where E.EmployeeID = O.EmployeeID
      GROUP BY LastName, FirstName
      Order By 1
Select LastName, Firstname, count(OrderID) as 'Orders'
      from nwEmployees E JOIN nwOrders O
      on E.EmployeeID = O.EmployeeID
      GROUP BY LastName, FirstName
      Order By 1
```

Beware the Cartesian Product

- Named after René DesCartes, philosopher & mathematician
- Famous for “Cogito ergo sum”
- Product = one table multiplied by another table
- The JOIN often creates a product, then selects rows from the product where the keys match
- For example, let’s join nwOrders to nwEmployees
- nwOrders has 14 columns, 830 rows
- nwEmployees has 17 columns, 9 rows
- The Cartesian product has 31 (14+17) columns, and 7470 (830 * 9) rows – most of which are meaningless

Cartesian Product

- SQL must go through the Cartesian Product (which is an INTERIM answer set) row-by-row, and select only those rows where the EmployeID from nwEmployees is equal to the EmployeeID from nwOrders
- Therefore, we must include the WHERE clause that describes this condition
- Failure to fully qualify a JOIN operation with a join condition that matches all necessary keys will cause your answer set to include part or all of the Cartesian Product (which is mostly meaningless)
- The JOIN requires SQL to do a lot of work which consumes a lot of disk I/O and memory (= expensive)

Cartesian Product

Create a table JoinCustomer:

```
DROP TABLE IF EXISTS JoinCustomer;
CREATE TABLE JoinCustomer  (
       CustomerID  CHAR(5),
       CompanyName VARCHAR(430),
       ContactName VARCHAR(30),
       Country VARCHAR(15)
       );
```

Load it with data from nwCustomers where Country = ‘USA’

```
INSERT INTO JoinCustomer
  (CustomerID, CompanyName, Contactname, Country)
SELECT CustomerID, CompanyName, ContactName, Country
   FROM nwCustomers
     where Country = 'USA';
```

Loads 13 rows of data.

Cartesian Product

Create a table JoinOrder:

```
DROP TABLE IF EXISTS JoinOrder;
CREATE TABLE JoinOrder  (
       OrderID  INT(11),
       CustomerID  CHAR(5),
       OrderDate  DATE,
       ShipCountry  VARCHAR(15)
       );
```

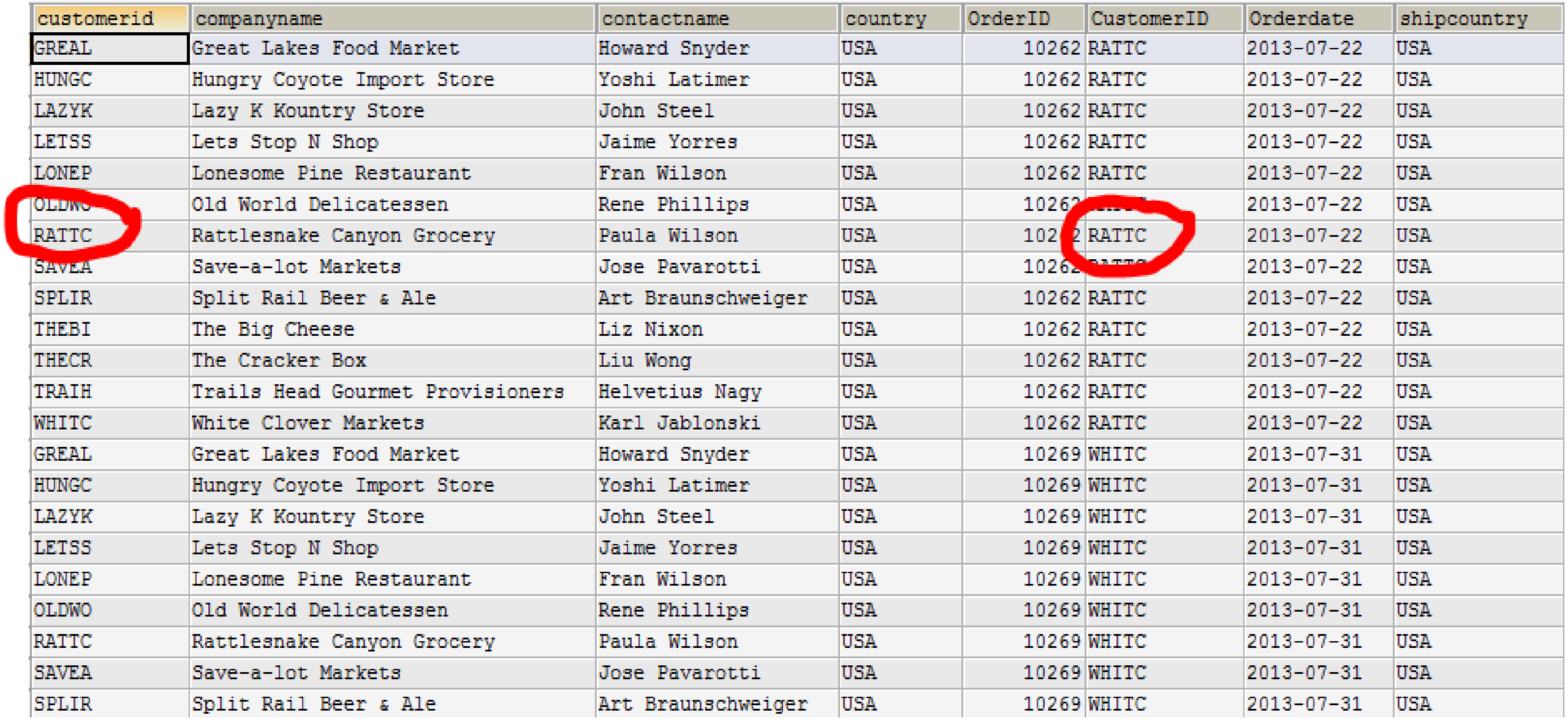
Load it with data from nwOrders where ShipCountry = ‘USA’

```
INSERT INTO JoinOrder
   (OrderID, CustomerID, Orderdate, ShipCountry)
SELECT OrderID, CustomerID, Orderdate, ShipCountry
   FROM nwOrders WHERE ShipCountry = 'USA';
```

Loads 122 rows of data.

Cartesian Product from an unqualified join:

```
SELECT c.customerid, companyname, contactname, country,
       OrderID, o.CustomerID,Orderdate, shipcountry
   FROM joincustomer c, joinorder o
```



customerid	companyname	contactname	country	OrderID	CustomerID	Orderdate	shipcountry
GREAL	Great Lakes Food Market	Howard Snyder	USA	10262	RATTC	2013-07-22	USA
HUNGC	Hungry Coyote Import Store	Yoshi Latimer	USA	10262	RATTC	2013-07-22	USA
LAZYK	Lazy K Kountry Store	John Steel	USA	10262	RATTC	2013-07-22	USA
LETSS	Lets Stop N Shop	Jaime Yorres	USA	10262	RATTC	2013-07-22	USA
LONEP	Lonesome Pine Restaurant	Fran Wilson	USA	10262	RATTC	2013-07-22	USA
OLDWO	Old World Delicatessen	Rene Phillips	USA	10262	RATTC	2013-07-22	USA
RATTC	Rattlesnake Canyon Grocery	Paula Wilson	USA	10262	RATTC	2013-07-22	USA
SAVEA	Save-a-lot Markets	Jose Pavarotti	USA	10262	RATTC	2013-07-22	USA
SPLIR	Split Rail Beer & Ale	Art Braunschweiger	USA	10262	RATTC	2013-07-22	USA
THEBI	The Big Cheese	Liz Nixon	USA	10262	RATTC	2013-07-22	USA
THECR	The Cracker Box	Liu Wong	USA	10262	RATTC	2013-07-22	USA
TRAIH	Trails Head Gourmet Provisioners	Helvetius Nagy	USA	10262	RATTC	2013-07-22	USA
WHITC	White Clover Markets	Karl Jablonski	USA	10262	RATTC	2013-07-22	USA
GREAL	Great Lakes Food Market	Howard Snyder	USA	10269	WHITC	2013-07-31	USA
HUNGC	Hungry Coyote Import Store	Yoshi Latimer	USA	10269	WHITC	2013-07-31	USA
LAZYK	Lazy K Kountry Store	John Steel	USA	10269	WHITC	2013-07-31	USA
LETSS	Lets Stop N Shop	Jaime Yorres	USA	10269	WHITC	2013-07-31	USA
LONEP	Lonesome Pine Restaurant	Fran Wilson	USA	10269	WHITC	2013-07-31	USA
OLDWO	Old World Delicatessen	Rene Phillips	USA	10269	WHITC	2013-07-31	USA
RATTC	Rattlesnake Canyon Grocery	Paula Wilson	USA	10269	WHITC	2013-07-31	USA
SAVEA	Save-a-lot Markets	Jose Pavarotti	USA	10269	WHITC	2013-07-31	USA
SPLIR	Split Rail Beer & Ale	Art Braunschweiger	USA	10269	WHITC	2013-07-31	USA

Cartesian Product

- The Cartesian Product contains 1586 rows. (122 * 13)
- Every row of JoinCustomer was multiplied by every row of JoinOrder.
- Only the rows where C.CustomerID = O.CustomerID are meaningful.

Advanced SQL

Joining three or more tables

- Every PAIR of tables being joined must have a common key
- Every PAIR of common keys must have a condition stated in a WHERE clause or in the “ON” clause of the JOIN
- Otherwise, your JOIN is not fully qualified and will result in a Cartesian Product (meaningless output)

Examples – Joining three tables

Create a report showing each employee and the total value of their orders sorted from highest value to lowest. (Order Value = UnitPrice * Quantity for each item on the order.)

```
Select LastName, Firstname,
   sum(UnitPrice * Quantity) as 'OrderValue'
   from nwEmployees E
   JOIN nwOrders O on E.EmployeeID = O.EmployeeID
   JOIN nwOrderDetails D on O.OrderID = D.OrderID
   GROUP BY LastName, FirstName
   Order By 3 desc
```

Examples – Joining three tables

Same Query, Different Syntax

```
Select LastName, Firstname,
   sum(UnitPrice * Quantity) as 'OrderValue'
   from nwEmployees E, nwOrders O, nwOrderDetails D
   where E.EmployeeID = O.EmployeeID
     and O.OrderID = D.OrderID
   GROUP BY LastName, FirstName
   Order By 3 desc
```

Advanced SQL

- **Join types**

Explicit inner join

```
SELECT * FROM employee
INNER JOIN department ON employee.DepartmentID =
   department.DepartmentID
```

Implicit inner join:

```
SELECT * FROM employee, department
   WHERE employee.DepartmentID =
   department.DepartmentID
```

Advanced SQL

- Join types (cont.)

Outer Join

```
SELECT * FROM employee
   LEFT OUTER JOIN department ON
   employee.DepartmentID =
   department.DepartmentID
```

Returns ALL rows from LEFT table and only matching rows from RIGHT table.

- **Analysis using Outer Joins**
  - Are there some rows in nworders that do not have a valid foreign key reference to nwemployees?
  - Are there some customers that have no orders?

```
SELECT COUNT(customerid)FROM nwcustomers
```

- there are 87 customers in nwcustomers

```
SELECT COUNT(distinct customerid)FROM nworders
```

- there are 89 distinct customers in nworders

- What’s the difference?

Advanced SQL

- We want to find the orders in nworders whose customerID is NOT in nwcustomers

Method One: use a subquery

```
   SELECT DISTINCT customerID
       FROM nworders
       WHERE customerID NOT IN (
              SELECT customerID FROM nwcustomers));
```

Method Two: use an outer join

```
SELECT DISTINCT O.customerID
       FROM nworders O LEFT OUTER JOIN nwcustomers C
       ON O.customerID = C.customerID
       WHERE C.customerID IS NULL
```

This shows us FOUR customers who have orders in nworders that have no matching row in nwcustomers

Advanced SQL

- We want to find the customers in nwcustomers who have no orders in nworders

Method One: use a subquery

```
SELECT customerID
    FROM nwcustomers
    WHERE customerID NOT IN (
          SELECT DISTINCT customerID FROM nworders);
```

Method Two: use an outer join

```
SELECT DISTINCT C.customerID
       FROM nworders O RIGHT OUTER JOIN nwcustomers C
       ON O.customerID = C.customerID
       WHERE O.customerID IS NULL
```

This shows us TWO customers who have no orders in nworders

Advanced SQL

DDL = Data Definition Language

- Some SQL commands are used to DEFINE or MODIFY the structures in the database.
  - `Create`
  - `Alter`
  - `Drop`

DML = Data Manipulation Language

- Some SQL commands are used to MODIFY the data in the database
  - `Update`
  - `Insert`
  - `Delete`

CREATE statement

```
   CREATE TABLE <table name>
   (column  DATATYPE(L),
    column  DATATYPE(L) NOT NULL,
    column  DATATYPE(L) CONSTRAINT <constr
      name> TYPE,
    column  DATATYPE(L))
    TABLESPACE <tablespace name>;
```

Advanced SQL

ALTER statement

```
ALTER TABLE <table name>
   ADD COLUMN column DATATYPE(L),
   CONSTRAINT <constr name> TYPE

ALTER TABLE <table name>
   DROP CONSTRAINT <constr name>
```

DROP statement

```
DROP TABLE <table name>
```

Advanced SQL

UPDATE statement

```
   UPDATE <table name>
   SET column = <value>
   WHERE <condition>
```

NOTE: if the WHERE is omitted, ALL rows are updated.

DELETE statement

```
   DELETE FROM <table name>
   WHERE <condition>
```

NOTE: if the WHERE is omitted, ALL rows are deleted.

INSERT statement

```
INSERT INTO <table name>
VALUES (value, value, value, value)
```

(must have a value or NULL for every column in the table)

```
INSERT INTO <table name> (column, column,
   column)
VALUES (value, value, value)
```

(if no column/value is specified, NULL or default will be assigned)

Getting more help with SQL

Best Online Tutorial: http://www.w3schools.com/sql/