Structured Query Language

- Structured Query Language (SQL) was developed by the IBM Corporation in the late 1970's.

- SQL was endorsed as a United States national standard by the American National Standards Institute (ANSI) in 1992 [SQL-92].

- Each DBMS manufacturer follows the ANSI standard, but also adds extended features unique to their SQL

SQL DDL and DML

- **SQL statements can be divided into two categories:**
  - **Data definition language (DDL)** statements
    - Used for creating tables, relationships and other structures.
  - **Data manipulation language (DML)** statements.
    - Used for queries and data modification.

SQL Basics

"Structured Query Language"

- It is NOT a programming / PROCEDURAL language
- It does not process one record at a time, rather, it is a SET processing language
- All inputs to SQL are tables
- Output from a query is a table referred to as the "**Answer Set**"
- Some queries may produce "interim" temporary answer sets, temp tables
- 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

SQL Order of Execution

ORDER	CLAUSE	FUNCTION
1	from	Choose and join tables to get base data.
2	where	Filters the base data.
3	group by	Aggregates the base data.
4	having	Filters the aggregated data.
5	select	Returns the final data.
6	order by	Sorts the final data.
7	limit	Limits the returned data to a row count.

USE statement

```
USE <database>;
```

- Tells the Query Engine which database you want to use for your query
- Let's work with **Northwinds** database.

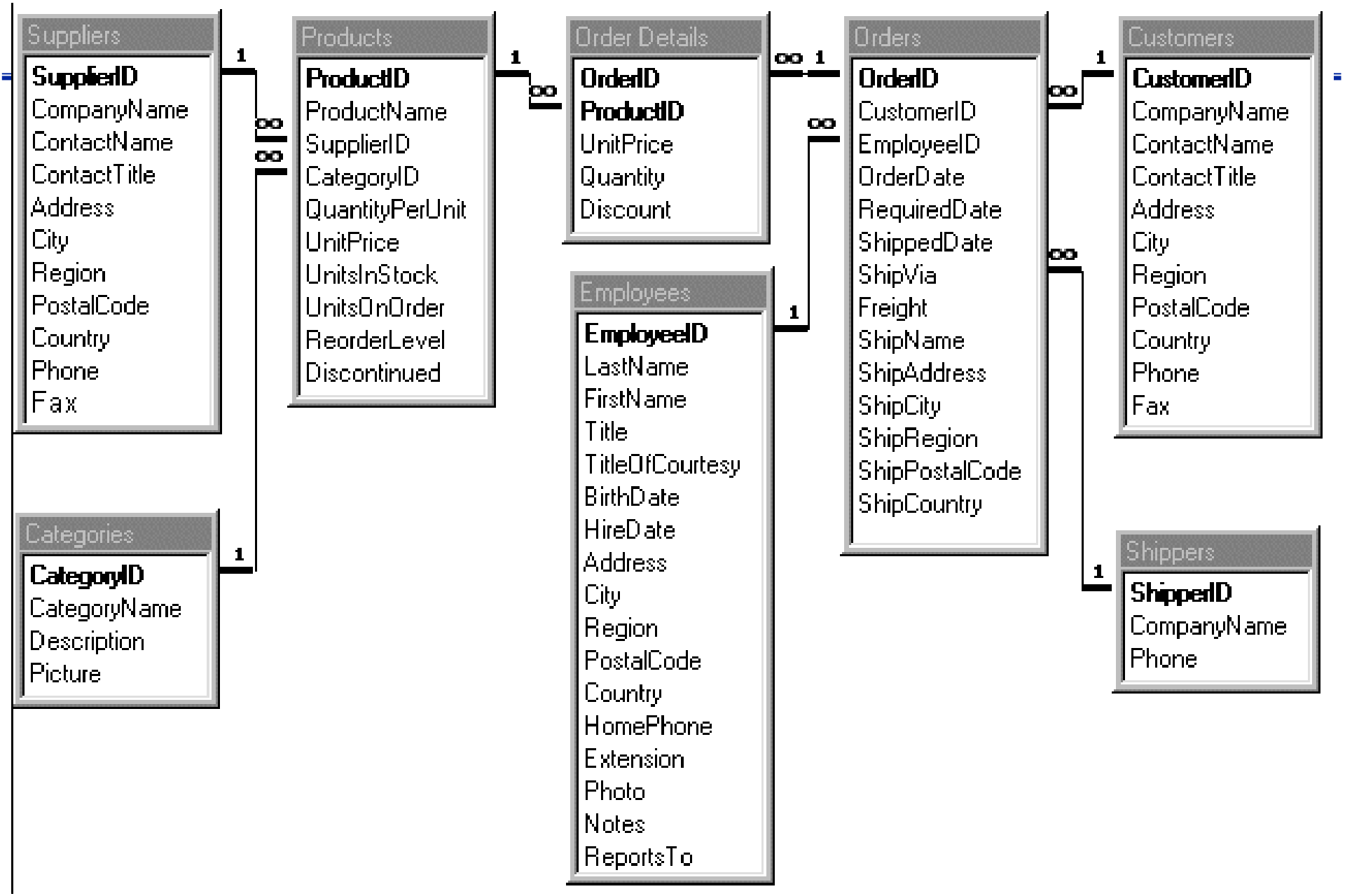
The Northwinds Database

- Sample relational database from Microsoft
- They purchase food items (“products”) from suppliers and sell them to customers
- 8 tables
- Primary Keys
- Foreign Keys

- Download the "**nwNorthWinds.sql**" script file (attached below), and execute it. It will create your database and tables, and then load the tables with data. The script should run fine as-is without any modification. The **nwNorthWinds.sql** contains 9 scripts for creating each of 8 tables, and one to verify that everything worked OK.
- After creating all 8 tables, run the "Verify" script. You should see the following tables and row counts for each table.

nwNorthWinds.sql

table_name	table_rows
nwcategories	8
nwcustomers	87
nwemployees	9
nworderdetails	2155
nworders	830
nwproducts	77
nwshippers	3
nwsuppliers	29


Suppliers
SupplierID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax
Products
ProductID
ProductName
SupplierID
CategoryID
QuantityPerUnit
UnitPrice
UnitsInStock
UnitsOnOrder
ReorderLevel
Discontinued
Order Details
OrderID
ProductID
UnitPrice
Quantity
Discount
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
Employees
EmployeeID
LastName
FirstName
Title
TitleOfCourtesy
BirthDate
HireDate
Address
City
Region
PostalCode
Country
HomePhone
Extension
Photo
Notes
ReportsTo
Categories
CategoryID
CategoryName
Description
Picture
Shippers
ShipperID
CompanyName
Phone
1
∞

SQL Basics

- USE northwinds;

- SHOW TABLES;
- DESC nwemployees;

- ## information_schema ##
- SELECT *
- FROM information_schema.tables
- WHERE TABLE_SCHEMA = 'northwinds';

- -- Verify row counts
- SELECT table_name, table_rows
- FROM information_schema.tables
- WHERE TABLE_NAME LIKE 'nw%';

SELECT statement

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

Examples

```
select *
      from  nwEmployees;

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

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

SELECT statement

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

- Concatenate character columns with **`CONCAT(column1, 'literal', column2)`** (multiple input columns combined into a single output column)

- Comment out a line of code by prefixing it with “ - - ” or #
- Commenting out syntax varies depending on DMBS tools.

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;
```

SELECT statement with WHERE clause

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

- The WHERE clause selects 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	

- Operator may be In or Like
  - In (literal, literal, literal)
  - Like `'string'` with % or _ as a wildcard
    - % = zero or more of any character
    - _ = exactly one of any character
- Multiple conditions may be joined with Boolean operators
  - AND, OR
- Conditions may be negated with Boolean operator
  - NOT

- **`Between`** is INCLUSIVE

- Answer Set rows may be sorted with “Order By”
  - Options are ASC or DESC
  - Defaults to ASC

- **`Boolean`** expressions are not English !
  - **`NOT`** negates the whole condition

```
SELECT Customerid, ContactName, Region, Country
    FROM nwCustomers
    where Country = 'Brazil‘
SELECT Customerid, ContactName, Region, Country
    FROM nwCustomers
    where Country NOT = 'Brazil'
SELECT Customerid, ContactName, Region, Country
    FROM nwCustomers
    where NOT Country = 'Brazil'
```

- When combining **WHERE** conditions using Boolean operators, please make a habit of using parentheses

```
SELECT Productname, SupplierID, CategoryID,
    UnitPrice, UnitsInStock
   FROM nwProducts
   WHERE SupplierID = 1 AND CategoryID = 2 OR
    CategoryID = 3 AND UnitPrice > 20 OR
    UnitsInStock < 12;

SELECT Productname, SupplierID, CategoryID,
    UnitPrice, UnitsInStock
   FROM nwProducts
   WHERE SupplierID = 1 AND (CategoryID = 2 OR
    CategoryID = 3 AND UnitPrice > 20) OR
    UnitsInStock < 12;
```

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;
```

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%'
```

Distinct:

- SQL cannot easily determine whether or not a row is a duplicate of another row
- The answer set may contain duplicate rows
- The “distinct” keyword before a column removes duplicates from the answer set
  - 87 Customers, each one has a country
  - How many distinct countries are they from?

Examples

Using distinct

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

Dates inside the database

- When you create a column and assign it a datatype
  - DATE,
  - DATETIME, or
  - TIMESTAMP

- DATE:
  - 'YYYY-MM-DD' format.
  - Supported range is '1000-01-01' to '9999-12-31’

- DATETIME:
  - 'YYYY-MM-DD HH:MM:SS.999999' format.
  - Supported range is

    '1000-01-01 00:00:00' to '9999-12-31 23:59:59’
- TIMESTAMP:
  - Supported range is

    '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.

Either may include fractional seconds as well (microseconds)

- MySQL converts dates in TIMESTAMP from current time zone to UTC upon storage. Converts in reverse upon retrieval.
- time_zone is a system variable
- strict mode forces valid dates and generates an error if invalid
- “relaxed” mode issues warnings on invalid dates

- 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.mmmmmm

- 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

Data Type	Storage Required Before MySQL 5.6.4	Storage Required as of MySQL 5.6.4
YEAR	1 byte	1 byte
DATE	3 bytes	3 bytes
TIME	3 bytes	3 bytes + fractional seconds storage
DATETIME	8 bytes	5 bytes + fractional seconds storage
TIMESTAMP	4 bytes	4 bytes + fractional seconds storage

As of MySQL 5.6.4, storage for YEAR and DATE remains unchanged. However, TIME, DATETIME, and TIMESTAMP are represented differently. DATETIME is packed more efficiently, requiring 5 rather than 8 bytes for the nonfractional part, and all three parts have a fractional part that requires from 0 to 3 bytes, depending on the fractional seconds precision of stored values.

Fractional Seconds Precision	Storage Required
0	0 bytes
1, 2	1 byte
3, 4	2 bytes
5, 6	3 bytes

Examples: Using DATES

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

SELECT Lastname, Firstname, Date_Format(BirthDate, '%M')
    FROM NWEmployees;

SELECT Lastname, Firstname, Date_Format(BirthDate, '%m/%d/%Y')
        AS BirthDateFROM NWEmployees;

SELECT LastName, FirstName, hiredate, CURDATE() AS CurYear,
    YEAR(CURDATE()) - YEAR(hiredate) AS Year_worked FROM
    nwEmployees;
```

DATE FUNCTIONS

Name	Description
ADDDATE()	Add time values (intervals) to a date value
ADDTIME()	Add time
CONVERT_TZ()	Convert from one time zone to another
CURDATE()	Return the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Synonyms for NOW()
CURTIME()	Return the current time
DATE()	Extract the date part of a date or datetime expression
DATE_ADD()	Add time values (intervals) to a date value
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract a time value (interval) from a date
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (0-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
EXTRACT()	Extract part of a date
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format Unix timestamp as a date
GET_FORMAT()	Return a date format string
HOUR()	Extract the hour
LAST_DAY	Return the last day of the month for the argument
LOCALTIME(), LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
MAKEDATE()	Create a date from the year and day of year
MAKETIME()	Create time from hour, minute, second
MICROSECOND()	Return the microseconds from argument
MINUTE()	Return the minute from the argument
MONTH()	Return the month from the date passed

Date_Format(Date,Format)

Specifier	Description
%D	Day of the month with English suffix (0th, 1st, 2nd, 3rd, ...)
%d	Day of the month, numeric (00..31)
%e	Day of the 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)
%M	Month name (January..December)
%m	Month, numeric (00..12)
%p	AM or PM
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S	Seconds (00..59)
%s	Seconds (00..59)
%T	Time, 24-hour (hh:mm:ss)
%U	Week (00..53), where Sunday is the first day of the week; WEEK() mode 0
%u	Week (00..53), where Monday is the first day of the week; WEEK() mode 1

MySQL Date Functions

https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html

SQL Tutorial

- This website (W3Schools.com) is the BEST online tutorial for students learning the SQL language.

- Click https://www.w3schools.com/sql/ link to open resource.

For Toad: https://www.toadworld.com/