Structured Query Language

SQL is everywhere

- just about every computer and every person on planet eventually touches something running SQL
- incredibly successful and solid technology
- runs universities (System Informatics iNtegrated University), banks, hospitals, governments, small businesses, large ones
- all Android Phones and iPhones have easy access to SQL database called SQLite
  - many applications on your phone use it directly

learning SQL ?

- weird obtuse kind of "non-language"
  - that most programmers can't stand
- based on solid mathematically built theory of operation (relational theory (extension of set theory))
- actually learn important theoretical concepts that apply to nearly every data storage system past and present

Non Procedural, Declarative Programming Language

- (procedural) programming languages - variables and data structures, using conditional logic (i.e., if-then-else) and looping constructs (i.e., do while ... end), ….
- procedural language defines both desired results and mechanism, or process, by which results are generated
- Nonprocedural languages define desired results, but process by which results are generated is left to an external agent
- manner in which statement is executed is left to component of your database engine known as *query optimizer*
- integrate SQL with programming language

SQL

- pronounce SQL "sequel“
- but you can also say "ESS-QUEUE-ELL" if you want
- stands for Structured Query Language
- language for interacting with data in (relational) database
- matches theory established many years ago defining properties of well structured data

SQL operations

- Create
  - putting data into tables - INSERT
- Read
  - query data out of tables - SELECT
- Update
  - change data already in table - UPDATE
- Delete
  - remove data from table – DELETE
- acronym "CRUD" is considered fundamental set of features every data storage system must have

SQL - language for doing CRUD operations

- SQL Data Definition Language - DDL
- to produce new tables or alter existing ones
- SQL only knows tables, and every operation produces tables
  - by modifying an existing one
  - or it returns new temporary table as your data set
- SQL Data Manipulation Language – DML
- SQL Security

SELECT

sqlzoo.net

- 3 laboratory works
- A Gentle Introduction to
- Structured Query Language

Why is it called SQLzoo?

- The animals of this zoo are SQL engines - one of each species
- They have been caged and tamed
- The public can poke, prod and gawp - the exhibits and the public are protected from each other

Who runs this site?

- Andrew Cumming is a lecturer at Napier University in Edinburgh, Scotland, UK
- is the zoo keeper, he feeds the animals and shovels away the waste

Can I shake his hand? buy him a drink?

- Next time you are visiting Edinburgh, UK you can shake his hand at the Tollcross State Circus which meets on Monday, 7pm to 9pm in Tollcross primary school

Can I buy him a drink?

- You can buy him a drink afterwards, in the Blue Blazer, Spittal Street; a pint of 80 Shilling please
- I (not Techical University of Cluj Napoca) will reimburse the expenses
  - bring a proof

Notes for teachers

- *This material was designed to be used in supervised tutorials at Napier University*
- Teachers from other institutions are welcome to use it in any way they see fit, however I have a few requests / suggestions:
- Reliability of the SQL Engines
  - Sometimes long running processes accumulate
- Feedback
  - If you are making use of this material please let me know what worked well and what didn‘t
  - Let me know if any of the questions are poorly phrased or confusing. I have tried to keep the language as simple as possible. Many students do not have English as their first language - I would like to hear more from them

Acknowledgements

- The CIA
  - Not just spying and wet work, they help you with your geography homework
- BBC News - Country Profiles
- Internet Movie Database
  - wonderful way to waste hours
- Scotland's Parliament
  - They've got my vote.
- TRAVELINE, Edinburgh City Council
- Amazon, New Scientist

SQL Basic Concepts and Principles

BASIC CONCEPTS AND PRINCIPLES

- in 1970, Dr. E. F. Codd of IBM's research laboratory published paper titled "A Relational Model of Data for Large Shared Data Banks" that proposed that data be represented as relations, sets of ***tables***
- redundant data used to link records in different tables

- each table in relational database includes information that uniquely identifies row in that table (known as ***primary key***), along with additional information needed to describe entity completely
- some tables also include information used to navigate to another table; this is where “redundant data” mentioned earlier comes in
- these columns are known as ***foreign keys***, and they serve purpose as lines that connect entities

Terminology

- Entity - something of interest to database user community; examples include customers, parts, geographic locations, etc.
- Column - individual piece of data stored in table
- Row - set of columns that together completely describe entity; also called record
- Table - set of rows, held on permanent storage (persistent)
- Result set - another name for nonpersistent table, generally result of SQL query
- Primary Key - one or more columns that can be used as unique identifier for each row in table
- Foreign Key - one or more columns that can be used together to identify single row in another table

Terminology

- Result set - another name for nonpersistent table, generally result of SQL query

- SQL goes with relational model - result of SQL query is table (also called, in this context, *result set*)
- table can be created in relational database simply by storing result set of query
- query can use both permanent tables and result sets from other queries as inputs

Example

EXAMPLE

Northwind sample database

- Northwind Traders, Access database, sample database
- contains sales data for fictitious company called Northwind Traders, which imports and exports specialty foods from around the world
- like real world application but names of companies, products, employees, and any other data used or mentioned do not in any way represent any real world individual, company, product, etc.
- can use Northwind to explore nearly every important aspect of database

- Northwind Sample Databases
- https://archive.codeplex.com/?p=northwinddatabase
- Northwind and pubs Sample Databases for SQL Server
- https://www.microsoft.com/en-us/download/details.aspx?id=23654
- https://github.com/fsprojects/SQLProvider/blob/master/docs/files/msaccess/Northwind.accdb

categories

- CategoryID, integer, Primary Key
- CategoryName, varchar(15)
- Description, text
- Picture, varbinary(MAX)

- CategoryID CategoryName Description
- 1 Beverages Soft drinks, coffees, teas, beers, and ales
- 2 Condiments Sweet and savory sauces, relishes, spreads, and seasonings
- 3 Confections Desserts, candies, and sweet breads
- 4 Dairy Products Milk and Cheeses
- 5 Grains/Cereals Breads, crackers, pasta, and cereal
- 6 Meat/Poultry Prepared meats
- 7 Produce Dried fruit and bean curd
- 8 Seafood Seaweed and fish

customers

- CustomerID, varchar(5), Primary Key
- CompanyName, varchar(40)
- ContactName, varchar(30)
- ContactTitle, varchar(30)
- Address, varchar(60)
- City, varchar(15)
- Region, varchar(15)
- PostalCode, varchar(10)
- Country, varchar(15)
- Phone, varchar(24)
- Fax, varchar(24)

- CustomerID CompanyName ContactName ContactTitle Address City
  Region PostalCode Country Phone Fax
- ALFKI Alfreds Futterkiste Maria Anders Sales Representative Obere Str. 57 Berlin NULL
  12209 Germany 030-0074321 030-0076545
- ANATR Ana Trujillo Emparedados y helados Ana Trujillo Owner Avda. de la Constitución 2222
  México D.F. NULL 05021 Mexico (5) 555-4729 (5) 555-3745
- ANTON Antonio Moreno Taquería Antonio Moreno Owner Mataderos 2312 México
  D.F. NULL 05023 Mexico (5) 555-3932 NULL
- AROUT Around the Horn Thomas Hardy Sales Representative 120 Hanover Sq.
  London NULL WA1 1DP UK (171) 555-7788 (171) 555-6750
- BERGS Berglunds snabbköp Christina Berglund Order Administrator Berguvsvägen 8
  Luleå NULL S-958 22 Sweden 0921-12 34 65 0921-12 34 67
- BLAUS Blauer See Delikatessen Hanna Moos Sales Representative Forsterstr. 57 Mannheim NULL
  68306 Germany 0621-08460 0621-08924
- BLONP Blondesddsl père et fils Frédérique Citeaux Marketing Manager 24, place Kléber
  Strasbourg NULL 67000 France 88.60.15.31 88.60.15.32
- BOLID Bólido Comidas preparadas Martín Sommer Owner C/ Araquil, 67
  Madrid NULL 28023 Spain (91) 555 22 82 (91) 555 91 99
- BONAP Bon app' Laurence Lebihan Owner 12, rue des Bouchers Marseille NULL
  13008 France 91.24.45.40 91.24.45.41
- BOTTM Bottom-Dollar Markets Elizabeth Lincoln Accounting Manager 23 Tsawassen Blvd.
  Tsawassen BC T2F 8M4 Canada (604) 555-4729 (604) 555-3745

employees

- EmployeeID, integer, Primary Key
- LastName, varchar(20); FirstName, varchar(10)
- Title, varchar(30); TitleOfCourtesy, varchar(25)
- BirthDate, date; HireDate, date
- Address, varchar(60)
- City, varchar(15)
- Region, varchar(15)
- PostalCode, varchar(10)
- Country, varchar(15)
- HomePhone, varchar(24); Extension, varchar(4)
- Photo, varchar(50)
- Notes, text
- ReportsTo, integer Foreign Key refer PK

- EmployeeID LastName FirstName Title TitleOfCourtesy BirthDate HireDate Address City
  Region PostalCode Country HomePhone Extension Notes ReportsTo PhotoPath
- 1 Davolio Nancy Sales Representative Ms. 1948-12-08 00:00:00.000 1992-05-01 00:00:00.000
  507 - 20th Ave. E.
- Apt. 2A Seattle WA 98122 USA (206) 555-9857 5467 Education includes a BA in psychology
  from Colorado State University in 1970. She also completed "The Art of the Cold Call." Nancy is a member of Toastmasters International. 2
  http://accweb/emmployees/davolio.bmp
- 2 Fuller Andrew Vice President, Sales Dr. 1952-02-19 00:00:00.000 1992-08-14 00:00:00.000
  908 W. Capital Way Tacoma WA 98401 USA (206) 555-9482 3457 Andrew received
  his BTS commercial in 1974 and a Ph.D. in international marketing from the University of Dallas in 1981. He is fluent in French and Italian and reads German. He joined the company as a
  sales representative, was promoted to sales manager in January 1992 and to vice president of sales in March 1993. Andrew is a member of the Sales Management Roundtable, the
  Seattle Chamber of Commerce, and the Pacific Rim Importers Association. NULL http://accweb/emmployees/fuller.bmp
- 3 Leverling Janet Sales Representative Ms. 1963-08-30 00:00:00.000 1992-04-01 00:00:00.000
  722 Moss Bay Blvd. Kirkland WA 98033 USA (206) 555-3412 3355 Janet has a BS
  degree in chemistry from Boston College (1984). She has also completed a certificate program in food retailing management. Janet was hired as a sales associate in 1991 and promoted
  to sales representative in February 1992. 2 http://accweb/emmployees/leverling.bmp
- 4 Peacock Margaret Sales Representative Mrs. 1937-09-19 00:00:00.000 1993-05-03 00:00:00.000
  4110 Old Redmond Rd. Redmond WA 98052 USA (206) 555-8122 5176
  Margaret holds a BA in English literature from Concordia College (1958) and an MA from the American Institute of Culinary Arts (1966). She was assigned to the London
  office temporarily from July through November 1992. 2 http://accweb/emmployees/peacock.bmp
- 5 Buchanan Steven Sales Manager Mr. 1955-03-04 00:00:00.000 1993-10-17 00:00:00.000
  14 Garrett Hill London NULL SW1 8JR UK (71) 555-4848 3453 Steven Buchanan
  graduated from St. Andrews University, Scotland, with a BSC degree in 1976. Upon joining the company as a sales representative in 1992, he spent 6 months in an orientation program at
  the Seattle office and then returned to his permanent post in London. He was promoted to sales manager in March 1993. Mr. Buchanan has completed the courses "Successful
  Telemarketing" and "International Sales Management." He is fluent in French. 2 http://accweb/emmployees/buchanan.bmp
- 6 Suyama Michael Sales Representative Mr. 1963-07-02 00:00:00.000 1993-10-17 00:00:00.000
  Coventry House
- Miner Rd. London NULL EC2 7JR UK (71) 555-7773 428 Michael is a graduate of Sussex
  University (MA, economics, 1983) and the University of California at Los Angeles (MBA, marketing, 1986). He has also taken the courses "Multi-Cultural Selling" and "Time Management
  for the Sales Professional." He is fluent in Japanese and can read and write French, Portuguese, and Spanish. 5
  http://accweb/emmployees/davolio.bmp
- 7 King Robert Sales Representative Mr. 1960-05-29 00:00:00.000 1994-01-02 00:00:00.000
  Edgeham Hollow
- Winchester Way London NULL RG1 9SP UK (71) 555-5598 465 Robert King
  served in the Peace Corps and traveled extensively before completing his degree in English at the University of Michigan in 1992, the year he joined the company. After completing a
  course entitled "Selling in Europe," he was transferred to the London office in March 1993. 5 http://accweb/emmployees/davolio.bmp
- 8 Callahan Laura Inside Sales Coordinator Ms. 1958-01-09 00:00:00.000 1994-03-05
  00:00:00.000 4726 - 11th Ave. N.E. Seattle WA 98105 USA (206) 555-1189 2344 Laura received a
  BA in psychology from the University of Washington. She has also completed a course in business French. She reads and writes French. 2
  http://accweb/emmployees/davolio.bmp
- 9 Dodsworth Anne Sales Representative Ms. 1966-01-27 00:00:00.000 1994-11-15 00:00:00.000
  7 Houndstooth Rd. London NULL WG2 7LT UK (71) 555-4444 452 Anne has a BA
  degree in English from St. Lawrence College. She is fluent in French and German. 5 http://accweb/emmployees/davolio.bmp

- EmployeeID LastName FirstName ReportTo
- 1 Davolio Nancy 2
- 2 Fuller Andrew NULL

- values and reference between Foreign Key ReportTo and Primary Key EmployeeID means that Nancy Davolio report to Andrew Fuller and that last one report to none

products

- ProductID, integer, Primary Key
- ProductName, varchar(40)
- SupplierID, integer; CategoryID integer - Foreign Keys
- QuantityPerUnit, varchar(20)
- UnitPrice, double
- UnitsInStock, integer; UnitsOnOrder integer
- ReorderLevel, integer
- Discontinued, enum('y','n')

shippers

- ShipperID, integer, Primary Key
- CompanyName, varchar(40)
- Phone, varchar(24)

suppliers

- SupplierID, integer, Primary key
- CompanyName, varchar(40)
- ContactName, varchar(30)
- ContactTitle, varchar(30)
- Address, varchar(60)
- City, varchar(15)
- Region, varchar(15)
- PostalCode, varchar(10)
- Country, varchar(15)
- Phone, varchar(24); Fax, varchar(24)
- HomePage, varchar(255)

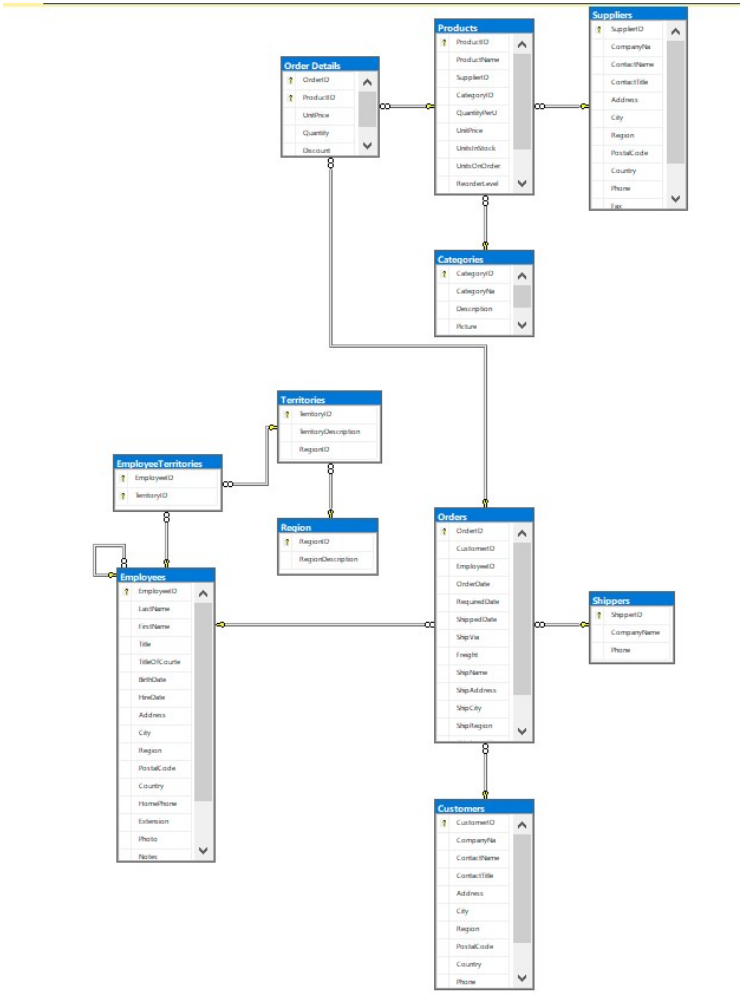
orders

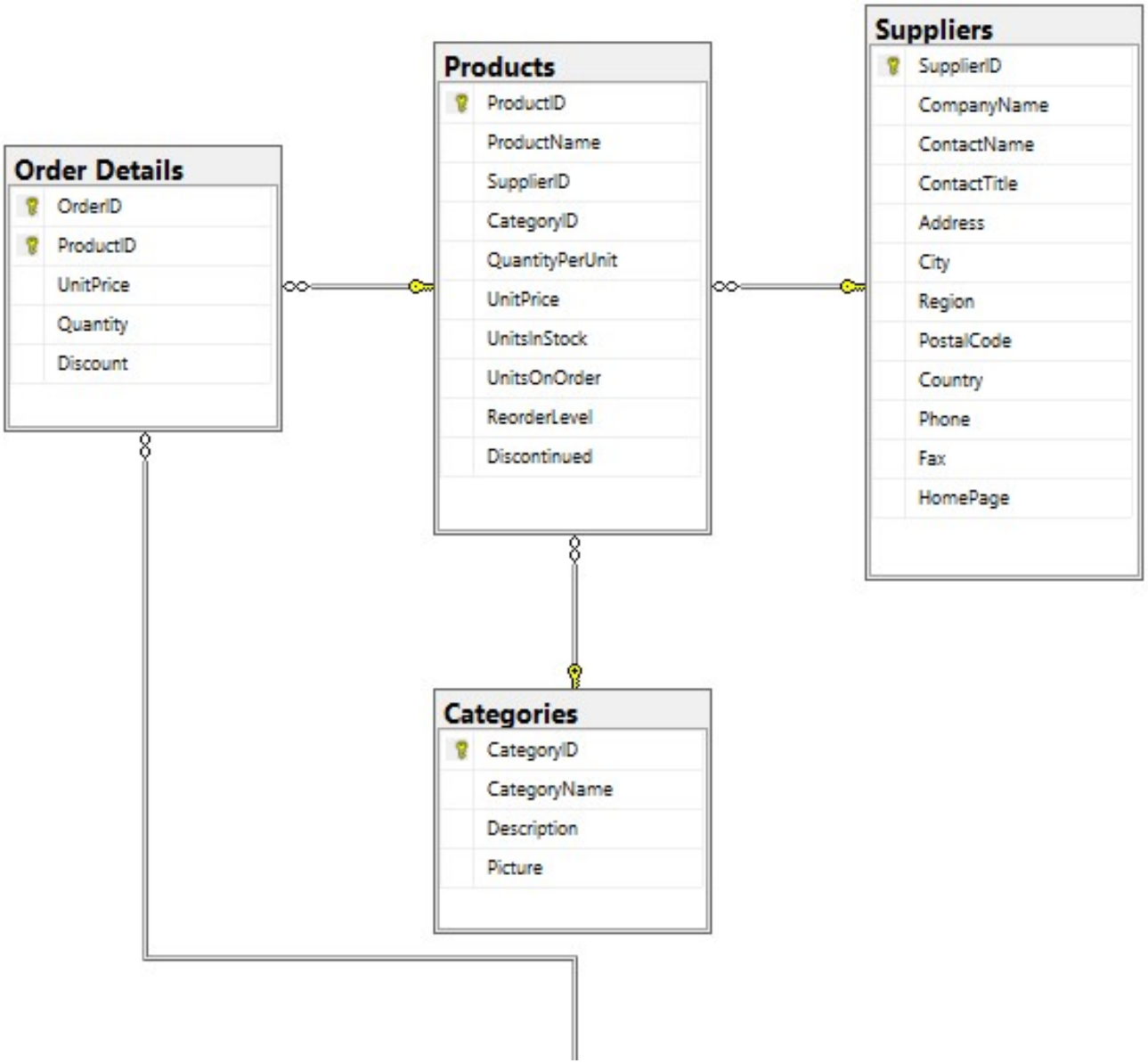
- OrderID, integer, Primary Key
- CustomerID, varchar(5); EmployeeID, integer - Foreign Keys
- OrderDate, date
- RequiredDate, date; ShippedDate, date
- ShipVia, integer Foreign Key
- Freight, double
- ShipName, varchar(40)
- ShipAddress, varchar(60)
- ShipCity, varchar(15)
- ShipRegion, varchar(15)
- ShipPostalCode, varchar(10)
- ShipCountry, varchar(15)

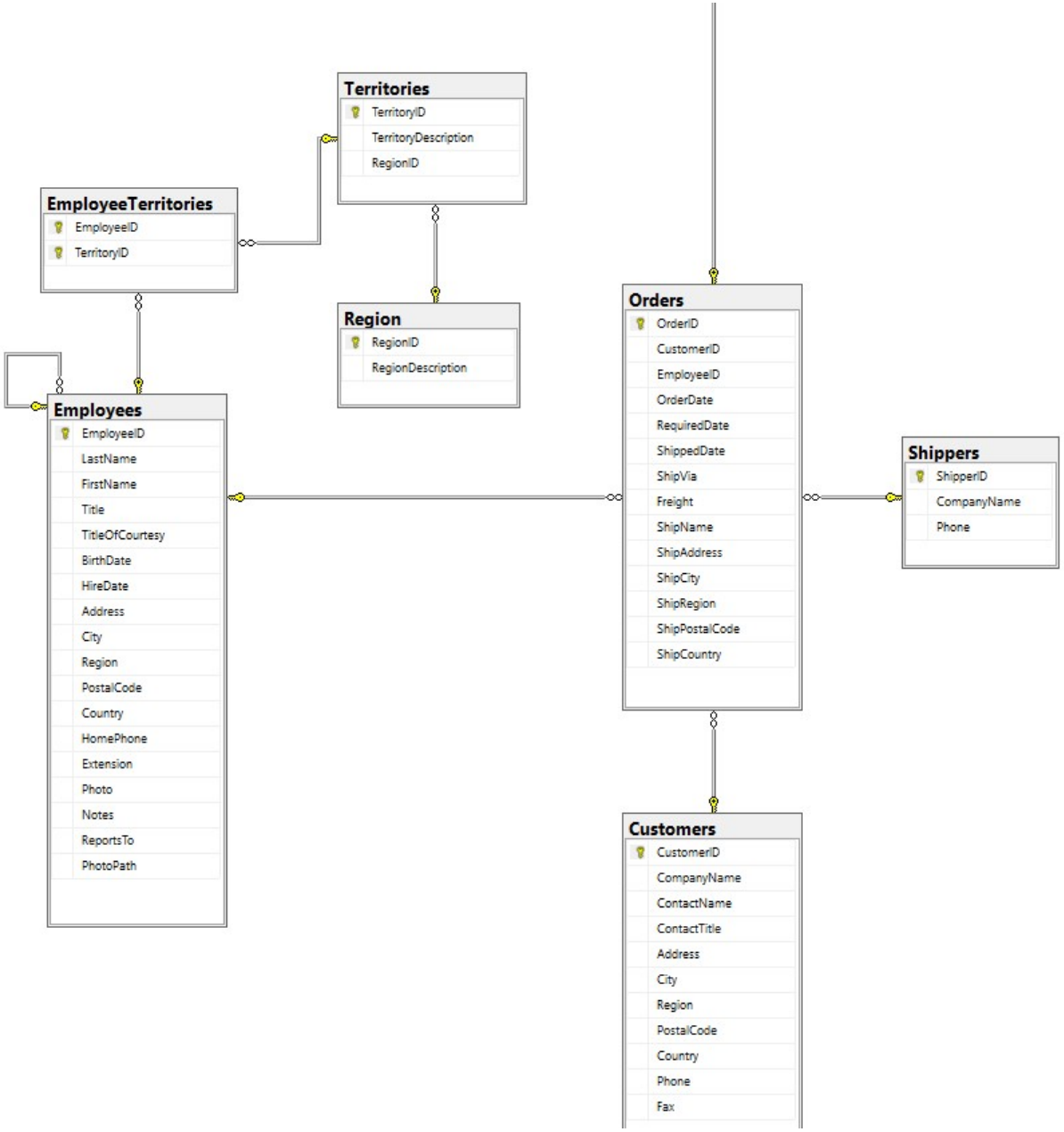
order_details

- ID, integer, OrderID, integer (Foreign Key refer order table) – Primary Key
- ProductID, integer – Foreign Key
- UnitPrice, double
- Quantity, integer
- Discount, float

DataBase Diagram




Order Details
OrderID
ProductID
UnitPrice
Quantity
Discount
Products
ProductID
ProductName
SupplierID
CategoryID
QuantityPerUnit
UnitPrice
UnitsInStock
UnitsOnOrder
ReorderLevel
Discontinued
Suppliers
SupplierID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax
HomePage
Categories
CategoryID
CategoryName
Description
Picture


Territories
TerritoryID
TerritoryDescription
RegionID
EmployeeTerritories
EmployeeID
TerritoryID
Region
RegionID
RegionDescription
Orders
OrderID
CustomerID
EmployeeID
OrderDate
RequiredDate
ShippedDate
ShipVia
Freight
ShipName
ShipAddress
ShipCity
ShipRegion
ShipPostalCode
ShipCountry
Shippers
ShipperID
CompanyName
Phone
Employees
EmployeeID
LastName
FirstName
Title
TitleOfCourtesy
BirthDate
HireDate
Address
City
Region
PostalCode
Country
HomePhone
Extension
Photo
Notes
ReportsTo
PhotoPath
Customers
CustomerID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax

SQL Basic Concepts and Principles

Tables

- basic building unit of relational database
- fairly intuitive way of organizing data
  - has been around for centuries
- consists of rows and columns
  - called records and fields in database jargon
- each table has unique name in database
  - unique fully qualified name includes schema or database name as prefix

- the dot (.) notation in fully qualified name is commonly used in programming world to describe hierarchy of objects and their properties
- for example, table field in MS SQL Server database could be referred ACME.DBO.CUSTOMER.CUST_ID_N
  - where ACME is database name
  - DBO is table owner (Microsoft standard)
  - CUSTOMER is name of table
  - CUST_ID_N is column name

column, domain

- each column has unique name within table, and any table must have at least one column
- records in table are not stored or retrieved in any particular order
- record is composed of number of cells, where each cell has unique name and might contain some data
- data within column must be of same type
  - for example, field AMOUNT contains only numbers
  - field DESCRIPTION, only words
- set of data within one field is said to be column's *domain*

Primary key

- primary role is to uniquely identify each record in table
  - based on idea of field (or fields) that contains set unique values
- *in the days of legacy databases, the records were always stored in some predefined order; if such an order had to be broken (because somebody had inserted records in a wrong order or business rule was changed), then the whole table (and, most likely, the whole database) had to be rebuilt*
- *RDBMS abolishes fixed order for records, but it still needs some mechanism of identifying the records uniquely, and primary key serves exactly this purpose*
- by it is very nature, PK cannot be empty
  - means that in table with defined primary key, PK fields must contain data for each record

Primary key

- is a requirement to have primary key on each and every table
- many RDBMS implementations would warn you if you create table without defining PK
- purists go even further, specifying that PK should be *meaningless* in sense that they would use some generated unique value (like EMPLOYEE_ID) instead of, say, Social Security numbers (despite that these are unique as well)
- primary key could consist of one or more columns, i.e., though some fields may contain duplicate values, their combination (set) is unique through the entire table
- key that consists of several columns is called *composite key*

Relationships, Foreign key

- RDBMS is built upon parent/child relationship based solely on values in table columns
  - relationships meaningful in logical terms, not in low-level computer specific pointers
- take the example of our fictitious order entry database
- ORDER_HEADER table is related to CUSTOMER table since both of these tables have a *common set of values*
  - ORDHDR_CUSTID_FN (customer ID) in ORDER_HEADER (and its values) corresponds to
  - CUST_ID_N in CUSTOMER

Relationships, Foreign key

- CUST_ID_N is said to be *primary key* for CUSTOMER table
- and *foreign key* for ORDER_HEADER table
- ORDER_HEADER has its own primary key — ORDHDR_ID_N which uniquely identifies orders
- in addition it will have foreign key ORDHDR_CUSTID_FN field
- values in that field correspond to values in CUST_ID_N primary key field for CUSTOMER table
- unlike primary key, foreign key is not required to be unique
  - one customer could place several orders

Relationships, Foreign key

- by looking into ORDER_HEADER table you can find which customers placed particular orders
- table ORDER_HEADER became related to table CUSTOMER
- became easy to find customer based on orders, or find orders for customer
- no longer need to know database layout, order of records in table, or master some low-level pointers or proprietary programming language to query data
- possible to run ad-hoc queries formulated in standard English-like language — Structured Query Language

SELECT

SELECT

- SELECT Employees.FirstName, Employees.LastName, Employees.BirthDate
- FROM Employees
- WHERE TitleOfCourtesy = 'Mr.’

FirstName	LastName	BirthDate
Steven	Buchanan	1955-03-04
Michael	Suyama	1963-07-02
Robert	King	1960-05-29

Query Clauses

- Several components or *clauses* make up SELECT
- 1. SELECT - determines which columns to include in query's result set
- 2. FROM - identifies tables from which to draw data and how tables should be joined
- 3. WHERE - filters out unwanted data
- GROUP BY, HAVING, ORDER BY

Query Clauses

- SELECT, FROM, WHERE
- 4. GROUP BY - used to group rows together by common column values
- 5. HAVING –filters out unwanted groups
- 6. ORDER BY - sorts rows of final result set by one or more columns

SELECT * FROM Categories

- *Show me all the columns (*) and all the rows (No WHERE) in the* Categories *table*

- choose to include only subset of columns in the Categories table as well:
- SELECT CategoryName, Description
- FROM Categories

- SELECT *clause determines which of all possible columns should be included in query's result set*

Include in SELECT clause

- columns from table or tables named in FROM clause
- Expressions
  - such as transaction.amount * −1
- Function calls
  - built-in function calls
    - such as ROUND(transaction.amount, 2)
  - User-Defined Function calls

- SELECT ProductName, UnitPrice,
- UnitsInStock + UnitsOnOrder,
- ROUND(UnitPrice * (UnitsInStock + UnitsOnOrder), 2)
- FROM Products

Column Aliases

- SELECT ProductName,
- UnitPrice,
- UnitsInStock + UnitsOnOrder AS TotalUnitsInStocks,
- ROUND(UnitPrice * (UnitsInStock + UnitsOnOrder),2) TotalPriceInStocks
- FROM Products

Removing Duplicates

SELECT CustomerID

FROM Orders

- 830 rows

- since some customers have more than one order, you will see same customer ID once for each order owned by that customer
- ALL keyword is default and never needs to be explicitly named

SELECT **DISTINCT** CustomerID

FROM Orders

- 89 rows

from Clause

- list of one or more tables
- from *clause defines tables used by query, along with means of linking tables together*

Tables

- 1. Permanent tables (created using CREATE TABLE statement)
- 2. Temporary tables (rows returned by subquery)
- 3 Virtual tables (created using CREATE VIEW statement)

Subquery-generated tables

- is query contained within another query
- surrounded by parentheses and can be found in various parts of select statement
- within from clause subquery serves role of generating temporary table that is visible from all other query clauses and can interact with other tables named in from clause

Ms. Employees from WA

- SELECT e.FirstName, e.LastName
- FROM (SELECT Employees.FirstName, Employees.LastName, Employees.TitleOfCourtesy, Employees.Region FROM Employees WHERE region = ‘WA’) e
- WHERE e.TitleOfCourtesy = 'Ms.'

Employees from WA

- SELECT
- Employees.FirstName, Employees.LastName, Employees.TitleOfCourtesy, Employees.Region
- FROM Employees
- WHERE region = 'WA')

Views

- query that is stored in data dictionary - looks and acts like table, but there is no data associated (stored) with view (*virtual* table)

- CREATE VIEW WAEmployees AS
- SELECT Employees.FirstName, Employees.LastName, Employees.TitleOfCourtesy, Employees.Region
- FROM Employees
- WHERE region = 'WA'
- SELECT FirstName, LastName
- FROM WAEmployees

Table Links

- second deviation from simple from clause definition is that if more than one table appears in from clause, conditions used to *link* tables must be included as well
- method of joining multiple tables
- SELECT Products.ProductName, Categories.CategoryName
- FROM Products INNER JOIN Categories ON Categories.CategoryID = Products.CategoryID

Table Aliases

- when multiple tables are joined in single query, you need way to identify which table you are referring to when you reference columns
- 1. use entire table name, such as Products.ProductName
- 2. assign each table *alias* and use alias throughout query

- SELECT Products.ProductName, Categories.CategoryName FROM Products INNER JOIN Categories ON Categories.CategoryID = Products.CategoryID

- SELECT ProductName, CategoryName
- FROM Products INNER JOIN Categories ON Categories.CategoryID = Products.CategoryID

- SELECT
- s.FirstName, s.LastName, s.Title,
- m.BossFirstName, m.BossLastName, m.BossTitle
- FROM Employees s INNER JOIN (SELECT EmployeeID, FirstName AS BossFirstName , LastName AS BossLastName, Title AS BossTitle FROM Employees) m
- ON s.ReportsTo = m.EmployeeID

Alias

- actually JOIN two copies of Employees, alias s and m, ON s.ReportsTo = m.EmployeeID
- SELECT s.FirstName, s.LastName, s.Title and
- m.FirstName, m.LastName, m.Title
- cause there will be conflicts add aliases
- m.BossFirstName, m.BossLastName, m.BossTitle

where Clause

- most of the time you will not wish to retrieve *every* row from table but will want way to filter out those rows that are not of interest
- where *clause mechanism for filtering out unwanted rows from your result set*
- contains single *filter condition*, but you can include many conditions as required, separated using operators such *and*, *or*, *not*

Ms. Employees from WA

- SELECT FirstName, LastName
- FROM Employees
- WHERE
- Region = 'WA' AND TitleOfCourtesy = 'Ms.'

Suppliers Products Categories

- SELECT
- Suppliers.CompanyName, Products.ProductName, Categories.CategoryName
- FROM Products
- JOIN Suppliers ON Products.SupplierID=Suppliers.SupplierID
- JOIN Categories ON Products.CategoryID=Categories.CategoryID

Suppliers of Beverages

- SELECT
- Suppliers.CompanyName, Products.ProductName, Categories.CategoryName
- FROM Products
- JOIN Suppliers ON Products.SupplierID=Suppliers.SupplierID
- JOIN Categories ON Products.CategoryID=Categories.CategoryID
- WHERE Categories.CategoryName = 'Beverages'

Suppliers of Condiments

- SELECT
- Suppliers.CompanyName, Products.ProductName, Categories.CategoryName
- FROM Products
- JOIN Suppliers ON Products.SupplierID=Suppliers.SupplierID
- JOIN Categories ON Products.CategoryID=Categories.CategoryID
- WHERE Categories.CategoryName = 'Condiments'

Suppliers of Beverages OR Condiments

- SELECT Suppliers.CompanyName
- FROM Products
- JOIN Suppliers ON Products.SupplierID=Suppliers.SupplierID
- JOIN Categories ON Products.CategoryID=Categories.CategoryID

- WHERE Categories.CategoryName = 'Beverages' OR Categories.CategoryName = 'Condiments'

Gotcha ?!

- In programming, a gotcha is a feature of a system, a program or a programming language that works in the way it is documented but is counter-intuitive and almost invites mistakes because it is both enticingly easy to invoke and completely unexpected and/or unreasonable in its outcome.

Suppliers of Beverages OR Condiments

- SELECT DISTINCT Suppliers.CompanyName
- FROM Products
- JOIN Suppliers ON Products.SupplierID=Suppliers.SupplierID
- JOIN Categories ON Products.CategoryID=Categories.CategoryID

- WHERE Categories.CategoryName = 'Beverages' OR Categories.CategoryName = 'Condiments'

- CompanyName
- Exotic Liquids
- Exotic Liquids
- Refrescos Americanas LTDA
- Bigfoot Breweries
- Bigfoot Breweries
- Aux joyeux ecclésiastiques
- Aux joyeux ecclésiastiques
- Leka Trading
- Bigfoot Breweries
- Pavlova, Ltd.
- Plutzer Lebensmittelgroßmärkte AG
- Karkki Oy
- Exotic Liquids
- New Orleans Cajun Delights
- New Orleans Cajun Delights
- Grandma Kelly's Homestead
- Grandma Kelly's Homestead
- Mayumi's
- Leka Trading
- Forêts d'érables
- Pavlova, Ltd.
- New Orleans Cajun Delights
- New Orleans Cajun Delights
- Plutzer Lebensmittelgroßmärkte AG

- CompanyName
- Aux joyeux ecclésiastiques
- Bigfoot Breweries
- Exotic Liquids
- Forêts d'érables
- Grandma Kelly's Homestead
- Karkki Oy
- Leka Trading
- Mayumi's
- New Orleans Cajun Delights
- Pavlova, Ltd.
- Plutzer Lebensmittelgroßmärkte AG
- Refrescos Americanas LTDA

Suppliers of Beverages AND Condiments

- SELECT DISTINCT Suppliers.CompanyName
- FROM Products
- JOIN Suppliers ON Products.SupplierID=Suppliers.SupplierID
- JOIN Categories ON Products.CategoryID=Categories.CategoryID

- WHERE Categories.CategoryName = 'Beverages' AND Categories.CategoryName = 'Condiments'

- Exotic Liquids
- Leka Trading
- Pavlova, Ltd.
- Plutzer Lebensmittelgroßmärkte AG

Gotcha ?!

- In programming, a gotcha is a feature of a system, a program or a programming language that works in the way it is documented but is counter-intuitive and almost invites mistakes because it is both enticingly easy to invoke and completely unexpected and/or unreasonable in its outcome.

Suppliers of Beverages AND Condiments

- SELECT DISTINCT Suppliers.CompanyName
- FROM Suppliers
- JOIN Products p1 ON p1.SupplierID=Suppliers.SupplierID
- JOIN Categories c1 ON p1.CategoryID=c1.CategoryID
- JOIN Products p2 ON p2.SupplierID=Suppliers.SupplierID
- JOIN Categories c2 ON p2.CategoryID=c2.CategoryID
- WHERE
- c1.CategoryName = 'Beverages' AND c2.CategoryName = 'Condiments'

Suppliers of Beverages AND Condiments

- SELECT DISTINCT Suppliers.CompanyName
- FROM Products
- JOIN Suppliers ON Products.SupplierID=Suppliers.SupplierID
- JOIN Categories ON Products.CategoryID=Categories.CategoryID
- WHERE Categories.CategoryName = 'Beverages’
- INTERSECT
- SELECT DISTINCT Suppliers.CompanyName
- FROM Products
- JOIN Suppliers ON Products.SupplierID=Suppliers.SupplierID
- JOIN Categories ON Products.CategoryID=Categories.CategoryID
- WHERE Categories.CategoryName = 'Condiments'

Suppliers of Beverages OR Condiments

- SELECT DISTINCT Suppliers.CompanyName
- FROM Products
- JOIN Suppliers ON Products.SupplierID=Suppliers.SupplierID
- JOIN Categories ON Products.CategoryID=Categories.CategoryID
- WHERE Categories.CategoryName = 'Beverages'
- UNION
- SELECT DISTINCT Suppliers.CompanyName
- FROM Products
- JOIN Suppliers ON Products.SupplierID=Suppliers.SupplierID
- JOIN Categories ON Products.CategoryID=Categories.CategoryID
- WHERE Categories.CategoryName = 'Condiments'

use parentheses to group conditions together in WHERE clause

- use parentheses to group conditions together in WHERE clause
- Sales Representative hired after 1994
- and
- Sales Manager hired before 1994

- SELECT
- FirstName, LastName, Title, HireDate FROM Employees
- WHERE
- (Title = 'Sales Representative' AND HireDate > '1994-01-01’)
- OR
- (Title = 'Sales Manager' AND HireDate < '1994-01-01')

group by and **having** Clauses

- all queries thus far have retrieved raw data without any manipulation
- sometimes you will want to find trends in your data that will require database server to prepare data before you retrieve your result set
- such mechanism is group by clause, which is used to group data by column values
- for example, rather than looking at list of employees and titles you might want to look at list of titles along with number of employees assigned to each
- when using group by clause, you may also use having clause, which allows you to filter group data in same way the where clause lets you filter raw data

- SELECT Title, COUNT(EmployeeID)
- FROM Employees GROUP BY Title

- Title (No column name)
- Inside Sales Coordinator 1
- Sales Manager 1
- Sales Representative 6
- Vice President, Sales 1

- SELECT Title, COUNT(EmployeeID)
- FROM Employees
- GROUP BY Title
- HAVING COUNT(EmployeeID) > 1

- Title (No column name)
- Sales Representative 6

order by Clause

- rows in result set returned from query are not in any particular order
- if you want your result set in particular order, you will need to instruct database server to sort results using order by clause:
- order by *clause is mechanism for sorting your result set using either raw column data or expressions based on column data*

- SELECT Products.ProductName, Categories.CategoryName, Suppliers.CompanyName
- FROM Products
- JOIN Suppliers ON Products.SupplierID=Suppliers.SupplierID
- JOIN Categories ON Products.CategoryID=Categories.CategoryID
- ORDER BY Products.ProductName

- SELECT Products.ProductName, Categories.CategoryName, Suppliers.CompanyName
- FROM Products
- JOIN Suppliers ON Products.SupplierID=Suppliers.SupplierID
- JOIN Categories ON Products.CategoryID=Categories.CategoryID
- ORDER BY Categories.CategoryName, Products.ProductName

Sorting, Group By via Expressions

- SELECT RIGHT (Products.ProductName, 1), COUNT(*)
- FROM Products
- GROUP BY RIGHT (Products.ProductName, 1)
- ORDER BY COUNT(*) DESC

(No column name)	(No column name)
e	17
s	9
t	9
a	7
d	6
u	6
r	5
i	4
n	3
g	3
x	3
o	2
l	2
p	1

Sorting via Numeric Placeholders

- can reference columns, in sorting, by their *position* in select clause rather than by name
- SELECT RIGHT (Products.ProductName, 1), COUNT(*)
- FROM Products
- GROUP BY RIGHT (Products.ProductName, 1)
- ORDER BY 2 DESC

FILTER

- where clause may contain one or more *conditions*, separated by
- binary operators *and* and or *or*
- unary operator *not*
- if where clause includes three or more conditions using both *and* and *or* or *not* operators, should use parentheses to make your intent clear

- typically difficult for person to evaluate where clause that includes not operator, which is why you won’t encounter it very often
- WHERE NOT (TitleOfCourtesy = 'Mr.’)
- WHERE TitleOfCourtesy != 'Mr.’

Condition

- made up of one or more *expressions* coupled with one or more *operators*
- expression can be any of the following:
  - number
  - string literal
  - column in table or view
  - built-in function
  - list of expressions, such as (‘Mrs.', ‘Ms.', ‘Mr.’)
  - subquery
- operators used within conditions include:
  - comparison operators
  - arithmetic operators

Condition

- large percentage of filter conditions will be ***equality*** *condition* of the form
  - '*column = expression*'
  - '*expression = expression*'
- another fairly common type of condition is ***inequality*** *condition*, which asserts that two expressions are *not* equal
  - '*expression != expression*'
  - '*expression <> expression*'

Condition

- you can build conditions that check whether expression falls within certain ***range*** (common when working with numeric or temporal data)
- ranges of dates and numbers are easy to understand, you can also build conditions that search for ranges of strings
- SELECT FirstName, LastName FROM Employees
- WHERE HireDate > '1993-01-01' AND
  HireDate < '1994-01-01'

Condition

- when you have *both* an upper and lower limit for your range, you may choose to use the ***between*** operator rather than using two separate conditions
- SELECT FirstName, LastName FROM Employees
- WHERE HireDate BETWEEN ('1993-01-01', '1994-01-01')

Condition

- in some cases, you will not be restricting an expression to single value or range of values, but rather to finite set of values
- ***in*** operator checks ***membership***
- SELECT FirstName, LastName FROM Employees
- WHERE TitleOfCourtesy = 'Mrs.' OR
  TitleOfCourtesy = 'Ms.'
- WHERE TitleOfCourtesy IN ('Mrs.', 'Ms.')

Condition

- with *in* operator, you can write single condition no matter how many expressions are in set
- along with writing your own set of expressions you can use subquery to generate set for you

- sometimes you want to see whether particular expression exists within set of expressions, and sometimes you want to see whether expression does *not* exist
- can use *not in* operator

Condition

- SELECT ProductName, QuantityPerUnit, UnitsInStock, ReorderLevel
- FROM Products
- WHERE SupplierID IN (7,24)

- SELECT ProductName, QuantityPerUnit, UnitsInStock, ReorderLevel
- FROM Products
- WHERE SupplierID IN (
- SELECT SupplierID FROM Suppliers WHERE Country = 'Australia')

Condition

- when searching for partial string matches, you might be interested in:
  - strings beginning/ending with certain character
  - strings beginning/ending with substring
  - strings containing certain character anywhere within string
  - strings containing substring anywhere within string
  - strings with specific format, regardless of individual characters
- can build search expressions to identify these and many others partial string ***matches*** by using ***wildcard*** characters

Condition

- wildcard character _ matches exactly one character
- underscore character takes place of single character
- wildcard character % matches any number of characters (including 0)
- percent sign can take place of variable number of characters
- when building conditions that utilize search expressions, you use ***like*** operator

Condition

- SELECT FirstName, LastName, Title FROM Employees WHERE EmployeeID IN (
- SELECT DISTINCT ReportsTo FROM Employees)

- SELECT FirstName, LastName, Title FROM Employees WHERE
- Title LIKE '%president%' OR
- Title LIKE '%manager%'

Null

- Null is absence of value, various flavors of null:
- *Not applicable*
  - such as employee ID column for transaction that took place at ATM machine
- *Value not yet known*
  - such as federal ID is not known at the time customer row is created
- *Value undefined*
  - such as account is created for product that has not yet been added to the database

- expression can *be* null, but it can never *equal* null
- two nulls are never equal to each other
- test whether expression is null, you need to use the ***is null*** operator
- SELECT FirstName, LastName, Title
- FROM Employees
- WHERE ReportsTo IS NULL
- WHERE ReportsTo IS NOT NULL

- SELECT FirstName, LastName, Title FROM Employees
- WHERE ReportsTo !=2

- account for possibility that some rows might contain null in ReportsTo column

- SELECT FirstName, LastName, Title FROM Employees
- WHERE ReportsTo !=2 OR ReportsTo IS NULL

Null Values and Three Valued Logic

- Any comparison with *null* returns *unknown*
  - *E.g. 5 < null or null <> null or null = null*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*, (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown) = unknown*
  - AND: *(true* **and** *unknown) = unknown, (false* **and** *unknown) = false,*
    (unknown **and** *unknown) = unknown*
  - NOT*: (***not** *unknown) = unknown*
  - "*P* **is unknown"** evaluates to true if predicate *P* evaluates to *unknown*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*

Null Values and Aggregates

- Total all amounts

  select sum (*amount)*
  from *table*

  – Above statement ignores null amounts
  – result is null if there is no non-null amount, that is the

- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes.

SQL Functions

Standard SQL Functions

- BIT_LENGTH (expression)
  - returns the length of the expression, usually string, in bits.
- CAST (value AS data type)
  - converts supplied value from one data type into another *compatible* data type
- CHAR_LENGTH (expression)
  - returns length of expression, usually string, in characters
- CONVERT (expression USING conversion)
  - returns string converted according to rules specified in conversion parameter
- CURRENT_DATE
  - returns current date of system

Standard SQL Functions

- CURRENT_TIME (precision)
  - returns current time of system, of specified precision
- CURRENT_TIMESTAMP (precision)
  - returns current time and current date of system, of specified precision
- EXTRACT (part FROM expression)
  - extracts specified named part of expression
- LOWER (expression)
  - converts character string from uppercase (or mixed case) into lowercase letters
- OCTET_LENGTH (*expression*)
  - returns length of expression in *bytes* (byte containing 8 bits)

Standard SQL Functions

- POSITION (*char expression I*N *source*)
  - returns position of the char expression in source.
- SUBSTRING (*string expression, start, length*)
  - returns string part of *string expression*, from *start* position up to specified *length*
- TRANSLATE (*string expression* USING *translation rule*)
  - returns string translated into another string according to specified rules
- TRIM(LEADING | TRAILING | BOTH *char expression* FROM *string expression*)
  - returns string from string expression where *leading*, *trailing,* or *both* char expression characters are removed
- UPPER (*expression*)
  - converts character string from lowercase (or mixed case) into uppercase letters

Numeric functions

- ABS (n)
  - returns absolute value of a number *n*
- CEILING (n)
  - returns smallest integer that is greater than or equal to *n*.
- EXP (n)
  - returns exponential value of *n*
- FLOOR (n)
  - returns largest integer less than or equal to *n*
- MOD (n,m) or %
  - returns remainder of *n* divided by *m*

Numeric functions

- POWER.(m,n)
  - returns value of *m* raised into n^{th} power
- RAND (n)
  - returns a random number between 0 and 1
- ROUND (n,m,[0])
  - returns number *n* rounded to *m* decimal places; last argument - 0 - is a default
  - similar to TRUNCate
- SIGN(n)
  - returns -1, if *n* is negative number, 1 if it is positive number, and 0 if number 0

String functions

- ASCII (string)
  - returns ASCII code of the first character of string
- CHAR (number) NCHAR (number)
  - returns character for ASCII code
- CONCAT (string1, string2) or '+‘
  - returns result of concatenation of two strings
- CHARINDEX (string1,string2, n)
  - returns position of occurrence of substring within string
- LEFT (string, n)
  - returns *n* number of characters starting from left
- LENGTH (string) or LEN (string)
  - returns number of characters in string

String functions

- DATALENGTH (expression)
  - returns number of bytes in expression, which could be any data type
- LTRIM (string)
  - returns string with leading blank characters removed
- REPLACE (string1, string2, string3)
  - replaces all occurrences of *string1* within *string2* with *string3*

String functions

- RTRIM (string)
  - returns string with trailing blank characters removed
- STR (expression)
  - converts argument expression into a character string
- SUBSTRING (string, n, m)
  - returns a part of a string starting from n^{th} character for the length of *m* characters

Date and time functions
MS SQL Server 2000

- DATEADD (month, number, date)
  - returns date plus date part (year, month, day)
- GETDATE
  - returns current date in session's time zone
- CONVERT or CAST
  - returns date from value according to specific format
- DAY (MONTH, YEAR)
  - returns DAY part (integer) of specified datetime expression

Date and time functions
MS SQL Server 2000

- DATEPART (date part, datetime)
  - returns requested date part (day, month, year)
- DATEDIFF
  - calculates difference between two dates
- DATEADD (day, n, m)
  - calculates what day would be next relative to some other supplied date
- DATEADD (datepart, n, m)
  - calculates what date would be next relative to some other supplied date

Time zone functions

- deal with Earth's different time zones
- functions always return time zone in which machine is located

Miscellaneous functions

- COALESCE (expression1, expression2, expression3 ...)
  - returns first argument on list that is not NULL
- CASE (expression)
  WHEN <compare value>
  THEN <substitute value>
  ELSE END
  - compares input expression to some predefined values, and outputs substitute value, either hard coded or calculated
- ISNULL (expression, value)
  - checks whether expression is NULL, and if it is returns specified value

Querying Multiple Tables

JOIN

JOIN

- queries against single table are certainly not rare, but you will find that most of your queries will require two, three, or even more tables

SELECT * FROM Categories

- CategoryID CategoryName Description
- 1 Beverages Soft drinks, coffees, teas, beers, and ales
- 2 Condiments Sweet and savory sauces, relishes, spreads, and seasonings
- 3 Confections Desserts, candies, and sweet breads
- 4 Dairy Products Milk and Cheeses
- 5 Grains/Cereals Breads, crackers, pasta, and cereal
- 6 Meat/Poultry Prepared meats
- 7 Produce Dried fruit and bean curd
- 8 Seafood Seaweed and fish

SELECT * FROM Products

- ProductID ProductName CategoryID QuantityPerUnit UnitPrice UnitsInStock UnitsOnOrder ReorderLevel
- 1 Chai 1 10 boxes x 20 bags 18.00 39 0 10
- 2 Chang 1 24 - 12 oz bottles 19.00 17 40 25
- 3 Aniseed Syrup 2 12 - 550 ml bottles 10.00 13 70 25
- 4 Chef Anton's Cajun Seasoning 2 48 - 6 oz jars 22.00 53 0 0
- 5 Chef Anton's Gumbo Mix 2 36 boxes 21.35 0 0 0
- 6 Grandma's Boysenberry Spread 2 12 - 8 oz jars 25.00 120 0 25
- 7 Uncle Bob's Organic Dried Pears 7 12 - 1 lb pkgs. 30.00 15 0 10
- 8 Northwoods Cranberry Sauce 2 12 - 12 oz jars 40.00 6 0 0
- …

- retrieve data from both tables - answer lies in Products.CategoryID which holds ID of category to which each products is assigned (in more formal terms, Products.CategoryID column is *Foreign Key* to Category table – values match Primary Key)

Cartesian Product

- put Product and Category tables into from clause of query and
- SELECT Products.ProductID,Products.ProductName, Products.CategoryID, Categories.CategoryID, Categories.CategoryName
- FROM Products, Categories
- because query didn't specify *how* tables should be joined, server generated *Cartesian product*, which is *every* permutation of two tables
- (8 Categories × 77 Products = 616 permutations)

- ProductID ProductName CategoryID CategoryID CategoryName
- 1 Chai 1 1 Beverages
- 2 Chang 1 1 Beverages
- 3 Aniseed Syrup 2 1 Beverages
- 4 Chef Anton's Cajun Seasoning 2 1 Beverages
- 5 Chef Anton's Gumbo Mix 2 1 Beverages
- 6 Grandma's Boysenberry Spread 2 1 Beverages
- 7 Uncle Bob's Organic Dried Pears 7 1 Beverages
- 8 Northwoods Cranberry Sauce 2 1 Beverages
- …

Cartesian Product

- this type of join is known as ***cross join***, and it is rarely used

Inner Joins

- to modify previous query so that 77 rows are included in result set (one for each product), you need to describe how two tables are related

Inner Joins

- SELECT Products.ProductID, Products.ProductName, Products.CategoryID, Categories.CategoryID, Categories.CategoryName
- FROM Products, Categories WHERE Products.CategoryID = Categories.CategoryID
- or
- SELECT Products.ProductID, Products.ProductName, Products.CategoryID, Categories.CategoryID, Categories.CategoryName
- FROM Products INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID

- ProductID ProductName CategoryID CategoryID CategoryName
- 1 Chai 1 1 Beverages
- 2 Chang 1 1 Beverages
- 3 Aniseed Syrup 2 2 Condiments
- 4 Chef Anton's Cajun Seasoning 2 2 Condiments
- 5 Chef Anton's Gumbo Mix 2 2 Condiments
- 6 Grandma's Boysenberry Spread 2 2 Condiments
- 7 Uncle Bob's Organic Dried Pears 7 7 Produce
- 8 Northwoods Cranberry Sauce 2 2 Condiments
- …

Inner Joins

- if value exists for CategoryID column in one table but *not* the other, then join fails for rows containing that value and those rows are excluded from result set
- this type of join is known ***inner join***, and it is most commonly used type of join
- if you want to include all rows from one table or the other regardless of whether match exists, you need to specify *outer join*
- if you do not specify type of join, server will do inner join by default

- SELECT Products.ProductID, Products.ProductName, Products.CategoryID, Categories.CategoryID, Categories.CategoryName
- FROM Products INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID
- WHERE Products.ProductName LIKE 'q%'
- Join conditions and filter conditions are separated into two different clauses (on subclause and where clause, respectively), making query easier to understand

Joining Three or More Tables

- is similar to joining two tables
- with two-table join, there are two tables and one join type in from clause, and a single on subclause to define how tables are joined
- with three-table join, there are three tables and two join types in from clause, and two on subclauses

- SELECT Products.ProductID, Products.ProductName, Products.CategoryID, Categories.CategoryID, Categories.CategoryName, Products.SupplierID, Suppliers.SupplierID, Suppliers.CompanyName
- FROM Products INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID
- INNER JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID
- WHERE Products.ProductName LIKE 'q%'

- order in which tables are named it is irrelevant
- Products Categories Suppliers
- Products Suppliers Categories
- Categories Products Suppliers
- Suppliers Products Categories
- will get exact same results

- order in which tables are named it is irrelevant
- Categories Suppliers Products
- Suppliers Categories Products
- will not work because we cannot JOIN Categories and Suppliers – there is no link

- order in which tables are named it is irrelevant
- using statistics gathered from your database objects, server must pick one of three tables as a starting point (chosen table is known as *driving table*), and then decide in which order to join remaining tables
- therefore, order in which tables appear in your from clause is not significant
- if, however, you believe that tables in your query should always be joined in particular order, you can place tables in the desired order and then
  - specify keyword STRAIGHT_JOIN in MySQL
  - request FORCE ORDER option in SQL Server

Using Subqueries As Tables in JOIN

- SELECT Products.ProductID, Products.ProductName, Products.CategoryID, Categories.CategoryID, Categories.CategoryName, Products.SupplierID, Suppliers.SupplierID, Suppliers.CompanyName
- FROM Products INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID
- INNER JOIN Suppliers ON Products.SupplierID=Suppliers.SupplierID
- WHERE Categories.CategoryName = 'Beverages' AND
- (Suppliers.CompanyName LIKE '%Liquid%' OR Suppliers.CompanyName LIKE '%Brew%')

- SELECT Products.ProductID, Products.ProductName, Products.CategoryID, c.CategoryID, c.CategoryName, Products.SupplierID, s.SupplierID, s.CompanyName
- FROM Products INNER JOIN
- (SELECT * FROM Categories WHERE Categories.CategoryName = 'Beverages') c
- ON Products.CategoryID = c.CategoryID
- INNER JOIN
- (SELECT * FROM Suppliers WHERE Suppliers.CompanyName LIKE '%Liquid%' OR Suppliers.CompanyName LIKE '%Brew%') s
- ON Products.SupplierID = s.SupplierID

- first subquery is given alias c, finds all Beverages
- second subquery is given alias s, finds suppliers of liquids and brews
- results are the same
- lack of where clause in main query; since all filter conditions are all inside subqueries

Performance

- JOIN 77 rows (Products) with 8 rows (Categories) with 29 rows (Suppliers)
- filter 17.864 rows, result 5 rows
- JOIN 77 rows (Products) with 1 rows (Categories) with 2 rows (Suppliers)
- result 5 rows

Using Same Table Twice: Self-Joins

- for this to work, you will need to give each instance of table different alias so that server knows which one you are referring to in various clauses
- SELECT s.EmployeeID, s.FirstName, s.LastName, s.Title, s.ReportsTo, m.EmployeeID, m.FirstName, m.LastName, m.Title
- FROM Employees s INNER JOIN Employees m ON s.ReportsTo = m.EmployeeID

Using Same Table Twice: Self-Joins

- not only can you include same table more than once in same query, but you can actually join table to itself
- Employees table, for example, includes *selfreferencing Foreign Key*, which means that it includes column ReportsTo that points to *Primary Key* within same table
  - column points to employee's manager

- EmployeeID FirstName LastName Title ReportsTo EmployeeID FirstName LastName Title
- 1 Nancy Davolio Sales Representative 2 2 Andrew Fuller Vice President, Sales
- 3 Janet Leverling Sales Representative 2 2 Andrew Fuller Vice President, Sales
- 4 Margaret Peacock Sales Representative 2 2 Andrew Fuller Vice President, Sales
- 5 Steven Buchanan Sales Manager 2 2 Andrew Fuller Vice President, Sales
- 6 Michael Suyama Sales Representative 5 5 Steven Buchanan Sales Manager
- 7 Robert King Sales Representative 5 5 Steven Buchanan Sales Manager
- 8 Laura Callahan Inside Sales Coordinator 2 2 Andrew Fuller Vice President, Sales
- 9 Anne Dodsworth Sales Representative 5 5 Steven Buchanan Sales Manager

Equi-Joins vs Non-Equi-Joins

- all of queries shown thus far have employed ***equi-joins***, meaning that values from two tables must match for join to succeed
- ***equi-join*** always employs an equals sign ON Products.CategoryID =Categories.CategoryID

- for example, let's say that managers has decided to have tennis tournament for all their people and you have been asked to create list of all pairings

- SELECT m.FirstName, m.LastName,
- s1.FirstName + ' ' + s1.LastName,
- ' vs. ',
- s2.FirstName + ' ' + s2.LastName
- FROM Employees m
- INNER JOIN Employees s1 ON s1.ReportsTo = m.EmployeeID
- INNER JOIN Employees s2 ON s2.ReportsTo = m.EmployeeID
- WHERE s2.LastName <> s1.LastName

Non-Equi-Joins

- SELECT s1.FirstName + ' ' + s1.LastName,
- ' vs. ' , s2.FirstName + ' ' + s2.LastName
- FROM Employees s1 INNER JOIN Employees s2 ON s1.EmployeeID != s2.EmployeeID

- Join Employees table to itself and return all rows where EmployeeID don’t match (since person can’t play tennis against himself)

- problem is that for each pairing (e.g., Andrew Fuller vs. Nancy Davolio), there is also a reverse pairing (e.g., Nancy Davolio vs. Andrew Fuller)
- way to achieve desired results is to use join condition
- SELECT s1.FirstName + ' ' + s1.LastName,
- ' vs. ', s2.FirstName + ' ' + s2.LastName
- FROM Employees s1 INNER JOIN Employees s2 ON s1.EmployeeID < s2.EmployeeID
- 36 rows = 9 * 8 / 2

SETS

Set Operators

- SQL language includes three set operators
- each set operator has two flavors, one that includes duplicates and another that removes duplicates (but not necessarily *all* of duplicates)
- data sets should be union compatible
- union
- intersection
- except

union Operator

- allow you to combine multiple data sets
- union sorts combined set and removes duplicates,
- union all does not (number of rows in final data set will always equal sum of number of rows in sets)

- SELECT DISTINCT Suppliers.CompanyName
- FROM Categories JOIN Products ON Categories.CategoryID = Products.CategoryID
- JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID
- WHERE CategoryName = 'Beverages'

- CompanyName

1. Aux joyeux ecclésiastiques
2. Bigfoot Breweries
3. Exotic Liquids
4. Karkki Oy
5. Leka Trading
6. Pavlova, Ltd.
7. Plutzer Lebensmittelgroßmärkte AG
8. Refrescos Americanas LTDA

- SELECT DISTINCT Suppliers.CompanyName
- FROM Categories JOIN Products ON Categories.CategoryID = Products.CategoryID
- JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID
- WHERE CategoryName = 'Condiments'

- CompanyName

1. Exotic Liquids
2. Forêts d'érables
3. Grandma Kelly's Homestead
4. Leka Trading
5. Mayumi's
6. New Orleans Cajun Delights
7. Pavlova, Ltd.
8. Plutzer Lebensmittelgroßmärkte AG

- SELECT DISTINCT Suppliers.CompanyName
- FROM Categories JOIN Products ON Categories.CategoryID = Products.CategoryID
- JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID
- WHERE CategoryName = 'Beverages'
- UNION
- SELECT DISTINCT Suppliers.CompanyName
- FROM Categories JOIN Products ON Categories.CategoryID = Products.CategoryID
- JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID
- WHERE CategoryName = 'Condiments'

- CompanyName

1. Aux joyeux ecclésiastiques
2. Bigfoot Breweries
3. Exotic Liquids
4. Forêts d'érables
5. Grandma Kelly's Homestead
6. Karkki Oy
7. Leka Trading
8. Mayumi's
9. New Orleans Cajun Delights
10. Pavlova, Ltd.
11. Plutzer Lebensmittelgroßmärkte AG
12. Refrescos Americanas LTDA

intersect Operator

- SELECT DISTINCT Suppliers.CompanyName
- FROM Categories JOIN Products ON Categories.CategoryID = Products.CategoryID
- JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID
- WHERE CategoryName = 'Beverages'
- INTERSECT
- SELECT DISTINCT Suppliers.CompanyName
- FROM Categories JOIN Products ON Categories.CategoryID = Products.CategoryID
- JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID
- WHERE CategoryName = 'Condiments'

- CompanyName

1. Exotic Liquids
2. Leka Trading
3. Pavlova, Ltd.
4. Plutzer Lebensmittelgroßmärkte AG

except Operator

- Find Customers who Ordered Products from all Categories

CREATE VIEW CustomCateg AS

- SELECT Customers.CompanyName, Categories.CategoryName
- FROM Customers JOIN Orders ON Orders.CustomerID = Customers.CustomerID
- JOIN OrderDetails ON OrderDetails.OrderID = Orders.OrderID
- JOIN Products ON Products.ProductID = OrderDetails.ProductID
- JOIN Categories ON Categories.CategoryID = Products.CategoryID

SELECT * FROM CustomCateg

	CompanyName	CategoryName
1.	Alfreds Futterkiste	Produce
2.	Alfreds Futterkiste	Beverages
3.	Alfreds Futterkiste	Seafood
4.	Alfreds Futterkiste	Condiments
5.	Alfreds Futterkiste	Dairy Products
6.	Ana Trujillo Emparedados y helados	Dairy Products
7.	Ana Trujillo Emparedados y helados	Beverages
8.	Ana Trujillo Emparedados y helados	Produce
9.	Ana Trujillo Emparedados y helados	Grains/Cereals
10.	Ana Trujillo Emparedados y helados	Seafood
11.	…	

- SELECT CategoryName FROM CustomCateg WHERE CompanyName = 'Alfreds Futterkiste'
- CategoryName

1. Beverages
2. Condiments
3. Dairy Products
4. Produce
5. Seafood

- SELECT CategoryName FROM CustomCateg WHERE CompanyName = 'QUICK-Stop'
- CategoryName

1. Beverages
2. Condiments
3. Confections
4. Dairy Products
5. Grains/Cereals
6. Meat/Poultry
7. Produce
8. Seafood

- SELECT Categories.CategoryName FROM Categories EXCEPT
- SELECT CategoryName FROM CustomCateg WHERE CompanyName = 'Alfreds Futterkiste'
- CategoryName

1. Confections
2. Grains/Cereals
3. Meat/Poultry

- SELECT Categories.CategoryName FROM Categories EXCEPT
- SELECT CategoryName FROM CustomCateg WHERE CompanyName = 'QUICK-Stop'
- CategoryName

- result it is the Ø empty set

- as always it is not possible to close whiteout **thank you for your kindly attention!**
- *If you get half as much pleasure - the guilty variety, to be sure - from reading this slides as I get from writing it, we're all doing pretty well.*