

DataBase Design

Proiectarea Bazelor de Date

HardCore DataBase

4th year Computer Science

DataBase people

- Delia MITREA
- Cristi MOCAN
- Me - Calin CENAN
- You

Course's GOAL

- follow 2nd year DataBase course
- course is not about database design only
- present advance topics in database – reports, procedural SQL extension, stored procedures, triggers, ..., concurrent access, transactions, ..., No SQL, Business Intelligence, DataBase Administration, ..., Persistence Object-Relational Mapping

DataBase Schedule

- 2 hour / week - laboratory Wednesday 08 – 16 room D3 Dorobantilor
- 2 hour / week - course Friday 8-10 room F Baritiu
- 2 ore / sapt. - curs joi ora 8-10 sala F Baritiu
- 2 ore / sapt. - laborator vineri 8-16 sala D3 Dorobantilor

Grade

- 50% - 2 hour Exam.
- 50% - Laboratory activity
 - Lab. Works & Programming Assignments

Course Textbooks

- Al. Lelutiu, Perenitatea conceptelor de baze de date
- R. Ramakrishnan, J. Gerhrke, Database Management Systems, McGraw Hill, 2002
- J. Ullman, H.G. Molina, J. Widom, Database Systems, Prentice Hall, 2001

Course Textbooks

- R. Dollinger, Baze de Date si Gestiunea Tranzactiilor, Ed. Albastra 1998
- R. Dollinger, Utilizarea sistemului SQL Server (SQL 7.0, SQL 2000) Ed. Albastra 2001

Course Textbooks

- Matthew Shepler, Writing Stored Procedures for Microsoft SQL Server, Sams Publishing 2000

Course Textbooks

- R. Dollinger, Utilizarea sistemului SQL Server (SQL 7.0, SQL 2000) Ed. Albastra 2001

DataBase course

- Romantic hero
- Barbarian
- Civilization
- Lost cause

The Charge of the Light Brigade

- Half a league, half a league,
- Half a league onward,
- All in the valley of Death
- Rode the six hundred.
- "Forward the Light Brigade!"
- Charge for the guns!" he said.
- Into the valley of Death
- Rode the six hundred.

The Charge of the Light Brigade

- Forward, the Light Brigade!"
- Was there a man dismay'd?
- Not tho' the soldier knew
- Some one had blunder'd.
- Theirs not to make reply,
- Theirs not to reason why,
- Theirs but to do and die.
- Into the valley of Death
- Rode the six hundred.

The Charge of the Light Brigade

- Cannon to right of them,
- Cannon to left of them,
- Cannon in front of them
- Volley'd and thunder'd;
- Storm'd at with shot and shell,
- Boldly they rode and well,
- Into the jaws of Death,
- Into the mouth of hell
- Rode the six hundred.

The Charge of the Light Brigade

- Flash'd all their sabres bare,
- Flash'd as they turn'd in air
- Sabring the gunners there,
- Charging an army, while
- All the world wonder'd.
- Plunged in the battery-smoke
- Right thro' the line they broke;
- Cossack and Russian
- Reel'd from the sabre-stroke
- Shatter'd and sunder'd.
- Then they rode back, but not,
- Not the six hundred.

The Charge of the Light Brigade

- Cannon to right of them,
- Cannon to left of them,
- Cannon behind them
- Volley'd and thunder'd;
- Storm'd at with shot and shell,
- While horse and hero fell,
- They that had fought so well
- Came thro' the jaws of Death,
- Back from the mouth of hell,
- All that was left of them,
- Left of six hundred.

The Charge of the Light Brigade

- When can their glory fade?
- O the wild charge they made !
- All the world wonder'd.
- Honor the charge they made !
- Honor the Light Brigade ,
- Noble six hundred !

- *The Charge of the Light Brigade*
- *by Alfred, Lord Tennyson*

DataBase people

- The Charge Of The Light Brigade 1968
- Me - Calin CENAN (10 min.)
- You (11 min.)

DataBase

C.J. Date, Ramakrishnan, ...

- ...
- ...
- Manele: Whisky și Red Bull
- ...
- ...
- Computer Science Licence
- ...

Computer Science

- DataBase centric
- Object Oriented

Vince Lombardi

- “*Winning is a habit*”
- “*Winning is not a sometime thing, it is an all the time thing. You don't do things right once in a while...you do them right all the time.*”

DataBase Design

Course 4th year

DataBase Design

“Data modeling is not optional”

no database or system was ever built
without at least an implicit model

- software programs are designed to implement a process model (or functional specification)
 - specifying business processes that the system is to perform
- same way, database is specified by **data model**, describing what sort of data will be held and how it will be organized

Design – Choice & Creativity

- in design, we do not expect to find a single correct answer, although we will certainly be able to identify many that are incorrect
- 2 data modelers given same set of requirements may produce quite different solutions

- First choice of what symbols or codes we use to represent real-world facts in database
- person's age could be represented by Birth Date, Age at Date of ..., or even by code corresponding to range

- Second choice there is usually more than one way to organize (classify) data
 - into tables and columns in relational model

- Third choice requirements from which we work in practice are usually incomplete, or at least loose enough to accommodate a variety of different solutions

- Fourth choice we have some options as to which part of system will handle each business requirement

- Finally, and perhaps most importantly
 - new information systems seldom deliver value simply by automating current way of doing things
- to exploit information technology fully, we generally need to change our business processes and data required to support them
- data modeler becomes a player in helping to design new way of doing business, rather than merely reflecting the old

- We want you to learn not only to produce sound, workable models (that will not fall down) but to be able to develop and compare different options
- *not throw away rule book, not suggest that anything goes*

Data Model Important

- Leverage
- Conciseness
- Data Quality

Leverage

- small change to data model may have major impact on system as whole
- programs are far more complex and take longer to specify and construct than database
 - their content and structure are heavily influenced by database design

Leverage

- well-designed data model can make programming simpler and cheaper
- poor data organization can be expensive to fix

Conciseness

- data model is powerful tool for expressing information systems requirements and capabilities
- implicitly defines whole set of screens, reports, and processes needed to capture, update, retrieve, and delete specified data

Data Quality

- database is usually valuable business asset built up over long period
- establishing common understanding of what is to be held in each table and column, and how it is to be interpreted
- problems with data quality can be traced to lack of consistency
 - in defining and interpreting data
 - implementing mechanisms to enforce definitions

What Makes a Good Data Model?

- Completeness
- Non-redundancy
- Enforcement of Business Rules
- Data Reusability
- Stability and Flexibility
- Elegance
- Communication
- Integration
- Conflicting Objectives
- Performance

Completeness

- Does the model support all the necessary data?

Non-redundancy

- Does the model specify a database in which the same fact could be recorded more than once?

Enforcement of Business Rules

- How accurately does the model reflect and enforce the rules that apply to the business' data?

Data Reusability

- Will data stored in database be re-useable for purposes beyond those anticipated in process model?
- once an organization has captured data to serve particular requirement, other potential uses and users almost invariably emerge

Stability and Flexibility

- How well will the model cope with possible changes to business requirements?
- can any new data required to support such changes be accommodated in existing tables?
 - or will we be forced to make major structural changes, with corresponding impact on the rest of the system?

Stability and Flexibility

- data model is **stable** in the face of a change to requirements if we do not need to modify it at all
 - we can sensibly talk of models being more or less stable, depending on level of change required
- data model is **flexible** if it can be readily extended to accommodate likely new requirements with only minimal impact on existing structure

Elegance

- Does the data model provide a reasonably neat and simple classification of the data?
- simple
- consistent
- easily described and summarized

Communication

- How effective is the model in supporting communication among various stakeholders in the design of a system?
- Do tables and columns represent business concepts that users and business specialists are familiar with and can easily verify?
- Will programmers interpret model correctly?

Integration

- How will the proposed database fit with organization's existing and future databases?
 - common for the same data to appear in more than one database and for problems to arise
- How easy is it to keep different versions in step, or to assemble a complete picture?

Conflicting Objectives

- above aims will conflict with one another
- elegant but radical solution may be difficult to communicate to conservative users
- can be attracted to elegant model that we exclude requirements that do not fit
- model that accurately enforces large number of business rules will be unstable if some rules change
- model that is easy to understand because (reflects perspectives of immediate system users) may not support reusability or integrate well with other

Performance

- system user will not be satisfied if our complete, non-redundant, flexible, and elegant database cannot meet throughput and response-time requirements
- differs from our other criteria because it depends heavily on software and hardware platforms on which database will run
 - exploiting their capabilities is technical task
 - quite different from more business-focused modeling

Performance

- performance requirements are usually “added to the mix” at a later stage than other criteria, and then only when necessary
- usual (and recommended) procedure is to develop data model without considering performance
- attempt to implement it with available hardware and software
- if it is not possible to achieve adequate performance in this way do we consider modifying the model itself

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 Technical 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=northwinddata
base](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://www.microsoft.com/en-us/download/details.aspx?id=23654)
- [https://github.com/fsprojects/SQLProvider/blob/
master/docs/files/msaccess/Northwind.accdb](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
spreads, and seasonings | Sweet and savory sauces, relishes, |
| 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 12209 Germany	Maria Anders 030-0074321030-0076545	Sales Representative	Obere Str. 57 Berlin	NULL
•	ANATR	Ana Trujillo Emparedados y helados México D.F. NULL	Ana Trujillo 05021 Mexico	Owner (5) 555-4729	Avda. de la Constitución 2222 (5) 555-3745	
•	ANTON	Antonio Moreno Taquería D.F. NULL	Antonio Moreno Mexico	Owner (5) 555-3932	Mataderos 2312	México
•	AROUT	Around the Horn London NULL	Thomas Hardy WA1 1DP UK	Sales Representative (171) 555-7788	120 Hanover Sq. (171) 555-6750	
•	BERGS	Berglunds snabbköp Luleå NULL	Christina Berglund S-958 22 Sweden	Order Administrator 0921-12 34 65	Berguvsvägen 8 0921-12 34 67	
•	BLAUS	Blauer See Delikatessen 68306 Germany	Hanna Moos 0621-08460	Sales Representative 0621-08924	Forsterstr. 57 Mannheim	NULL
•	BLONP	Blondesddsl père et fils Strasbourg NULL	Frédérique Citeaux 67000 France	Marketing Manager 88.60.15.31 88.60.15.32	24, place Kléber	
•	BOLID	Bólido Comidas preparadas Madrid NULL	Martín Sommer 28023 Spain	Owner (91) 555 22 82	C/ Araquil, 67 (91) 555 91 99	
•	BONAP	Bon app' 13008 France	Laurence Lebihan 91.24.45.40	Owner 91.24.45.41	12, rue des Bouchers	Marseille
•	BOTTM	Bottom-Dollar Markets Tsawassen BC	Elizabeth Lincoln T2F 8M4	Accounting Manager (604) 555-4729	23 Tsawassen Blvd. (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 Region	FirstName PostalCode	Title Country	TitleOfCourtesy HomePhone	BirthDate Extension	HireDate Notes	Address ReportsTo	City PhotoPath
•	1	Davolio 507 - 20th Ave. E.	Nancy	Sales Representative Ms.	USA (206) 555-9857	1948-12-08 00:00:00.000	5467	1992-05-01 00:00:00.000	
•	Apt. 2A	Seattle from Colorado State University in 1970. She also completed "The Art of the Cold Call." Nancy is a member of Toastmasters International.	WA	98122	USA (206) 555-9857		5467	Education includes a BA in psychology 2	
•	2	Fuller 908 W. Capital Way	Andrew Tacoma	Vice President, Sales Dr. WA 98401	USA USA (206) 555-9482	1952-02-19 00:00:00.000	3457	1992-08-14 00:00:00.000	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 722 Moss Bay Blvd.	Janet Kirkland	Sales Representative Ms. WA 98033	USA (206) 555-3412	1963-08-30 00:00:00.000	3355	1992-04-01 00:00:00.000	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 4110 Old Redmond Rd.	Margaret	Sales Representative Mrs. Redmond WA 98052	USA USA (206) 555-8122	1937-09-19 00:00:00.000	5176	1993-05-03 00:00:00.000	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 14 Garrett Hill	Steven	Sales Manager NULL	Mr. SW1 8JR UK (71) 555-4848	1955-03-04 00:00:00.000	3453	1993-10-17 00:00:00.000	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 Coventry House	Michael	Sales Representative Mr.		1963-07-02 00:00:00.000		1993-10-17 00:00:00.000	
•	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 Edgeham Hollow	Robert	Sales Representative Mr.		1960-05-29 00:00:00.000		1994-01-02 00:00:00.000	
•	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 00:00:00.0004726 - 11th Ave. N.E. Seattle	Laura	Inside Sales Coordinator WA 98105	Ms. USA (206) 555-1189	1958-01-09 00:00:00.000	2344	1994-03-05	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 7 Hounds tooth Rd.	Anne	Sales Representative Ms. NULL London	UK (71) 555-4444	1966-01-27 00:00:00.000	452	1994-11-15 00:00:00.000	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)

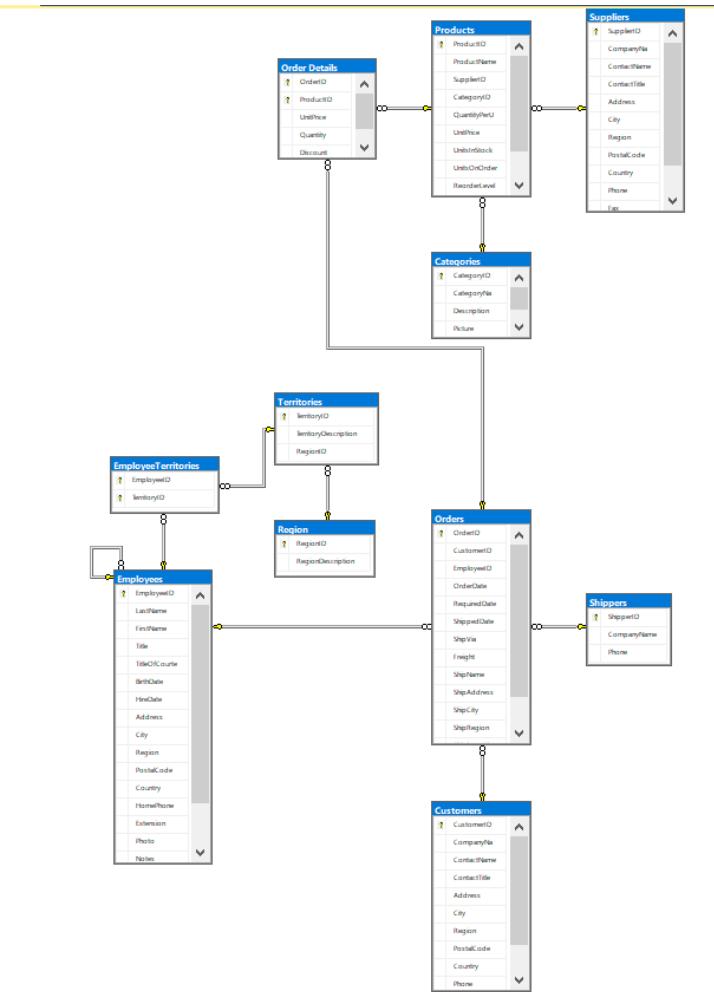
orders

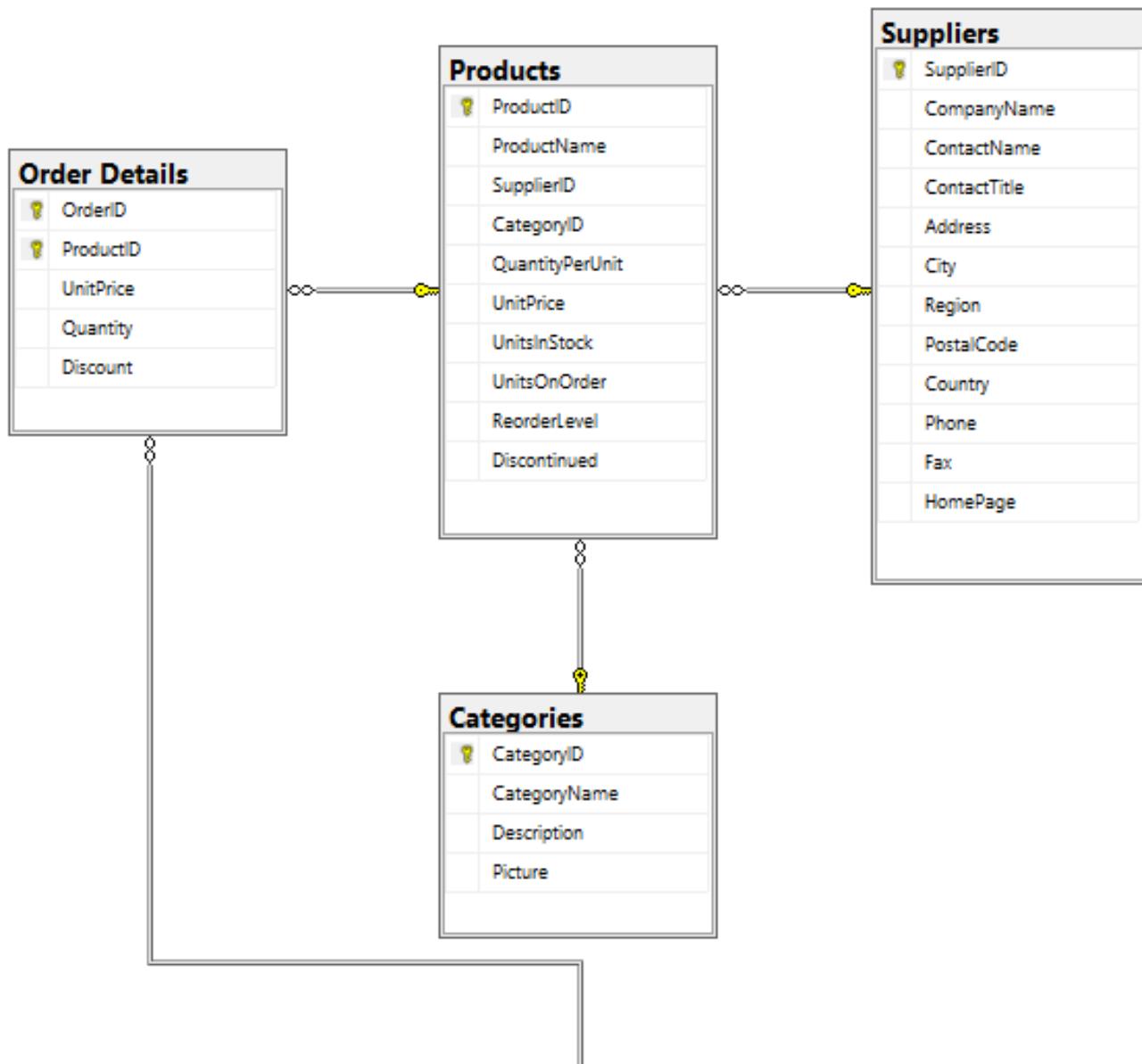
- OrderID, integer, Primary Key
- CustomerID, varchar(5); EmployeeID, integer - Foreign Keys
- OrderDate, date
- RequiredDate, date; ShippedDate, date
- ShipVia, integer Foreign Key
- Freight, double
- ShipName, varchar(40)
- ShipAddress, varchar(60)
- ShipCity, varchar(15)
- ShipRegion, varchar(15)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

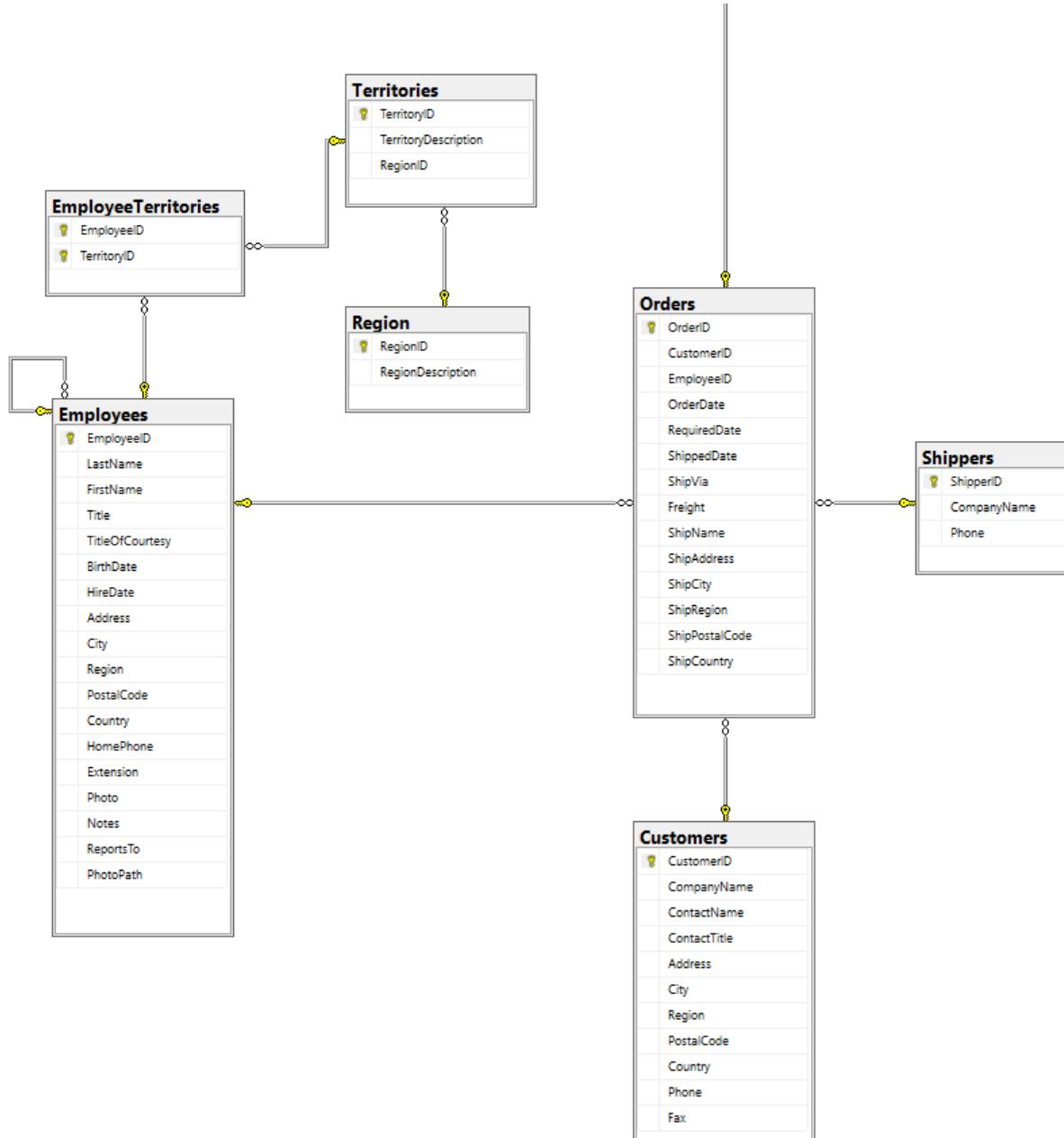
order_details

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

DataBase Diagram







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 its 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'étables
 - Pavlova, Ltd.
 - New Orleans Cajun Delights
 - New Orleans Cajun Delights
 - Plutzer Lebensmittelgroßmärkte AG
- CompanyName
 - Aux joyeux ecclésiastiques
 - Bigfoot Breweries
 - Exotic Liquids
 - Forêts d'éables
 - 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. $5 < \text{null}$ or $\text{null} < > \text{null}$ or $\text{null} = \text{null}$
- 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 IN 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	ReorderLevel	
		UnitPrice	UnitsInStock	UnitsOnOrder		
•	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	53	0	0		
•	5	Chef Anton's Gumbo Mix	2	36 boxes	21.35	0 0
	0	0				
•	6	Grandma's Boysenberry Spread		2	12 - 8 oz jars	25.00
	120	120	0	25		
•	7	Uncle Bob's Organic Dried Pears		7	12 - 1 lb pkgs.	30.00
	15	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, sReportsTo, 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'éables
 - 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 \emptyset 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.*

Structured Query Language

SQL Data Definition Language

Table Creation - Data Types

- Character Data - if you want to store strings, for example up to 50 characters in length, you could use:
 - `char(50) /* fixed-length */`
 - `varchar(50) /* variable-length */`
- DataBase dependent:
- in MySQL `char` columns is currently 255 chars, whereas `varchar` columns can be up to 65,535 chars
- in SQL Server there are also `nchar()`, `nvarchar()`, store Unicode characters (essential if you require use of extended character sets); 2 vs 4 bytes each char

Table Creation - Data Types

- Character Data - if you want to store longer strings (such as emails, documents, etc.), then you will want to use in MySQL one of text types (mediumtext and longtext), or in SQL Server varchar(MAX) or nvarchar(MAX)
- use *char* when all strings to be stored in column are of similar length, such as state abbreviations, phone numbers and *varchar* when strings to be stored in column are of varying lengths
- both *char* and *varchar* are used in similar fashion in all major database servers (engines)

Table Creation - Data Types

- Character sets: databases can store data using various character sets, both single/multibyte, some of supported character sets in MySQL
 - Charset Description Default collation MaxLen (in bytes)
 - big5 Big5 Traditional Chinese big5_chinese_ci 2
 - cp850 DOS West European cp850_general_ci 1
 - hp8 HP West European hp8_english_ci 1
 - koi8r KOI8-R Relcom Russian koi8r_general_ci 1
 - latin2 ISO 8859-2 Central European latin2_general_ci 1
 - ascii US ASCII ascii_general_ci 1
 - ujis EUC-JP Japanese ujis_japanese_ci 3
 - hebrew ISO 8859-8 Hebrew hebrew_general_ci 1
 - gb2312 GB2312 Simplified Chinese gb2312_chinese_ci 2
 - greek ISO 8859-7 Greek greek_general_ci 1
 - gbk GBK Simplified Chinese gbk_chinese_ci 2
 - armSCII8 ARMSCII-8 Armenian armSCII8_general_ci 1
 - **utf8 UTF-8 Unicode utf8_general_ci 3**

Table Creation - Data Types

- Character Data - to work with string ranges, you need to know order of characters within your character set (order in which characters within character set are sorted is called a ***collation***).

Table Creation - Data Types

- Character Data
- if you create column for free-form data entry, such as notes column to hold data about customer interactions with your company's customer service department, then *varchar* will probably be adequate
- if you are storing documents, however, you should choose *longtext* type

Table Creation - Data Types

- Numeric Data - there are several different numeric data types that reflect various ways in which numbers are used
- column indicating whether customer order has been shipped - referred to as Boolean, would contain 0 to indicate false and 1 to indicate true
- system-generated primary key for transaction table - generally start at 1 and increase in increments of 1 up to potentially very large number

Table Creation - Data Types

- Numeric Data
- item number for customer's electronic shopping basket - values for this would be positive whole numbers between 1 and, at most, 200
- price for item from customer's electronic shopping basket - values for this would have 2 decimal positions (money)
- positional data for circuit board drill machine - high-precision scientific or manufacturing data often requires accuracy to 8 decimal points

Table Creation - Data Types

- Numeric Data
- *int* in MySQL, *integer* in SQL Server represent numbers between $-2,147,483,648$ to $2,147,483,647$ or from 0 to $4,294,967,295$ for unsigned variant
- *Float(p,s)* $-3.402823466E+38$ to $-1.175494351E-38$ and $1.175494351E-38$ to $3.402823466E+38$
- *Double(p,s)* $-1.7976931348623157E+308$ to $-2.2250738585072014E-308$ and $2.2250738585072014E-308$ to $1.7976931348623157E+308$

- using floating-point type, you can specify *precision* (total number of allowable digits both to left and to right of decimal point) and *scale* (number of allowable digits to right of decimal point), but they are not required
- data stored in column will be rounded if number of digits exceeds scale and/or precision of column
- for example, column defined float(4,2) will store total of four digits, two to left of decimal and two to right of decimal; number 17.8675 would be rounded to 17.87, and attempting to store number 178.375 would generate error

Table Creation - Data Types

- Temporal Data
- for character strings and numerical data there are pretty much the same way of storing values and same function in all (relational) databases: MySQL, SQL Server, Access, Oracle, DB2, SQLite, PostgreSQL, ...
- different ways of storing temporal data in databases

MySQL temporal types

- *Type* Default format Allowable values
- *Date* YYYY-MM-DD 1000-01-01 to 9999-12-31
- *Time* HHH:MI:SS -838:59:59 to 838:59:59
- *Datetime* YYYY-MM-DD HH:MI:SS 1000-01-01 00:00:00 to 9999-12-31 23:59:59
- *Timestamp* YYYY-MM-DD HH:MI:SS 1970-01-01 00:00:00 to 2037-12-31 23:59:59
- *Year* YYYY 1901 to 2155

SQL SERVER temporal types

- Data type Format Range Accuracy Storage size (bytes)
- *Time* hh:mm:ss[.nnnnnnn] 00:00:00.0000000 through 23:59:59.9999999 100 nanoseconds 3 to 5
- *Date* YYYY-MM-DD 0001-01-01 through 9999-12-31 1 day 3
- *Smalldatetime* YYYY-MM-DD hh:mm:ss 1900-01-01 through 2079-06-06 1 minute 4
- *Datetime* YYYY-MM-DD hh:mm:ss[.nnn] 1753-01-01 through 9999-12-31 0.00333 second 8
- *Datetime2* YYYY-MM-DD hh:mm:ss[.nnnnnnn] 0001-01-01 00:00:00.0000000 through 9999-12-31 23:59:59.9999999 100 nanoseconds 6 to 8
- *Datetimeoffset* YYYY-MM-DD hh:mm:ss[.nnnnnnn] [+|-]hh:mm 0001-01-01 00:00:00.0000000 through 9999-12-31 23:59:59.9999999 (in UTC) 100 nanoseconds 8 to 10 (TimeZone Offset)

Table Creation

- CREATE TABLE person
 - (person_id SMALLINT UNSIGNED,
 - firstname VARCHAR(20),
 - lastname VARCHAR(20),
 - gender CHAR(1),
 - birth_date DATE,
 - street VARCHAR(30),
 - city VARCHAR(20),
 - state VARCHAR(20),
 - country VARCHAR(20),
 - postal_code VARCHAR(20),
 - CONSTRAINT pk_person PRIMARY KEY (person_id));
- *Query OK, 0 rows affected*

- everything should be fairly self-explanatory ...
- except for last item - when define table, need to tell database server what column or columns will serve as Primary Key - do this by creating *constraint* on table (can add several types of constraints to table definition)
- CONSTRAINT pk_person PRIMARY KEY (person_id) - created on person_id column and given name pk_person and is of type PK

- another type of constraint called *check constraint* constrains allowable values for particular column
- MySQL
 - gender CHAR(1) CHECK (gender IN ('M','F')),
- SQL Server
 - CONSTRAINT [CK_Products_UnitPrice] CHECK (([UnitPrice] >= 0))

Table Creation

- if you forget to create constraints when you first create table, you can add it later via
- `ALTER TABLE table_name`
- tables which will not be used could be eliminated from database schema by issuing
- `DROP TABLE table_name`

Establishing Relationships between Tables

- relationships are established by columns with values that are shared/match between two tables
- in a one-to-many relationship, one row (from one column) in first table matches same value in multiple rows in one column of second table
- database that is properly linked together using Foreign Keys is said to have *referential integrity*

Establishing Relationships between Tables

- `ALTER TABLE Products WITH NOCHECK ADD CONSTRAINT FK_Products_Categories FOREIGN KEY(CategoryID) REFERENCES Categories (CategoryID)`
- adding Foreign Key constraints to table on many side of one-to-many relationship creates links
- Foreign Key constraints need to be added to only one side of relationship, in a one-to-many relationship, they are added to many side

Database Schema creation

- usually these operations are done in front-end tools like MySQL Workbench or SQL Server Management Studio
- which translate DataBase Addministrator intent into SQL language (proper CREATE, ALTER, DROP tables)
- everything which is done in database server / engine it is actually SQL language

SQL Data Manipulation Language

CRUD CREATE (INSERT) READ (SELECT) UPDATE DELETE

Codd's rules (for relational DataBase)

- **7. The system must support set-at-a-time insert, update, and delete operations.**
- means that **system must be able to perform insertions, updates, and deletions of multiple rows in single operation**

- simplest way to populate character column is to enclose string in quotes
- numeric data is quite straightforward
- working with temporal data is most involved when it comes to data generation and manipulation; some of complexity is caused by many ways in which single date and time can be described
 - Wednesday, September 15, 20010
 - 9/15/2010 2:14:56 P.M. EST 9/15/2010 19:14:56 GMT
 - 2612008 (Julian format) Star date [-4] 85712.03 14:14:56 (*Star Trek* format)
 - and you have to deal also with Time Zones

- `INSERT INTO string_tbl (char_fld, varchar_fld, text_fld) VALUES ('This is char data', 'This is varchar data', 'This is text data');`
- *Query OK, 1 row affected*
- `UPDATE string_tbl SET varchar_fld = 'This is a piece of extremely long, too long varchar data';`
- *ERROR : Data too long for column 'varchar_fld' at row 1*
- `DELETE FROM string_tbl;`
- *Query OK, 1 row affected (0.00 sec)*

including ' single quotes

- won't be able to insert following string because server will think that apostrophe in word *doesn't* marks end of string
- UPDATE string_tbl SET text_fld = 'This string doesn't work';
- need to add *escape* to string so that server treats apostrophe like any other character in string.
- all servers allow you to escape single quote by adding another single quote directly before:
- UPDATE string_tbl SET text_fld =
- 'This string didn't work, but it does now';
- *Query OK, 1 row affected (0.01 sec)*
- *Rows matched: 1 Changed: 1 Warnings: 0*

- UPDATE Orders SET OrderDate = '2016-07-04'
- WHERE OrderID = '10248'
- INSERT INTO Customers (
- CustomerID, CompanyName, ContactName,
ContactTitle, Address, City, Region, PostalCode,
Country, Phone) VALUES
- ('UTC-N', 'Technical University of Cluj-Napoca',
'Calin CENAN', 'Lecturer', 'Baritiu St. 26-28', 'Cluj-
Napoca', 'Cluj', '400124', 'Romania', '0264 401
200');

Adding incomplete records

- sometimes you might want to add a record to table before you have data for all record's columns – possible as long as you have data for primary key and all columns that have NOT NULL constraint
- `INSERT INTO Customers VALUES`
- `('UTC-N', 'Technical University of Cluj-Napoca', 'Calin CENAN', 'Lecturer', 'Baritiu St. 26-28', 'Cluj-Napoca', 'Cluj', '400124', 'Romania', '0264 401 200', NULL);` value of last column, Fax, is NULL

- `INSERT INTO Products (ProductName, SupplierID, CategoryID, QuantityPerUnit)`
- `VALUES ('Tirigomodina', 1, 1, "")`
- `ProductID` is automatically inserted by DB engine as next value in sequence
- `UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued` have default value of 0

- `SELECT * INTO Shipped FROM Orders`
- `WHERE 0=1;`
- copies data from one table into new table
(without any constraints or keys from original table)
- copy all columns into new table and no rows
(condition it is always false)
- create new, empty table using schema of another
 - just add WHERE clause that causes query to return no data

- `INSERT INTO Shipped (CustomerID, EmployeeID, OrderDate,[RequiredDate, ShippedDate, ShipVia, Freight, ShipName, ShipAddress, ShipCity, ShipRegion, ShipPostalCode, ShipCountry)`
- `SELECT CustomerID, EmployeeID, OrderDate,[RequiredDate, ShippedDate, ShipVia, Freight, ShipName, ShipAddress, ShipCity, ShipRegion, ShipPostalCode, ShipCountry FROM Orders WHERE Orders.ShippedDate IS NOT NULL;`

- keying in a succession of SQL INSERT statements is the slowest and most tedious way to enter data into database
- although clerks at airline ticket counters, fast food restaurants, and supermarkets and users at e-commerce Web sites don't need to know anything about SQL, somebody does
- in order to make users' life easier, someone has to write programs that process data coming in from keyboards, data entry pads, and bar code scanners, and sends it to database
- those programs are typically written in general-purpose language such as C, Java, Visual Basic or php and incorporate SQL statements that are then used in actual 'conversation' with database

Updating data in table

- UPDATE *table_name* SET *column_1* = *expression_1*,
column_2 = *expression_2*, ..., *column_n* = *expression_n*
- [WHERE predicates];
- UPDATE Orders SET
- OrderDate = DATEFROMPARTS(2017,
MONTH(OrderDate), DAY(OrderDate)) ,
- RequiredDate = DATEFROMPARTS(2017,
MONTH(RequiredDate), DAY(RequiredDate)) ,
- ShippedDate = DATEFROMPARTS(2017,
MONTH(ShippedDate), DAY(ShippedDate))
- *(830 rows affected)*

- SET clause specifies which columns will get new values and what those new values will be
- optional WHERE clause specifies which rows update applies to
 - if there is no WHERE clause, update is applied to all rows in table
- UPDATE Products SET ProductName = 'Ce Ai?!"
- WHERE ProductName = 'Chai'
- WHERE ProductID = 1

Deleting data from table

- `DELETE FROM table_name`
- `[WHERE predicates] ;`
- optional WHERE clause specifies which rows delete applies to
 - if there is no WHERE clause, delete is applied to all rows in table
- `DELETE FROM Orders`
- `WHERE OrderID IN`
- `(SELECT OrderID FROM Shipped)`

- DELETE FROM Orders
- WHERE OrderDate < ‘2017-01-01’
- all of 2016’s Orders records (and previously) are deleted

SQL Data Control Language

SECURITY

- GRANT and REVOKE control who may access various parts of database
- before you can grant database access to someone, you must have some way of identifying that person - some parts of user identification, such as issuing passwords and taking other security measures, are implementation-specific
- SQL has a standard way of identifying and categorizing users, however, so granting and revoking privileges can be handled

- SQL doesn't specify how user identifier is assigned; in many cases, operating system's login ID serves purpose
- role name it is other form of authorization identifier that enable access to a database
- every SQL session is started by user and have a *SQL-session user identifier*
- privileges associated with SQL-session user identifier determine what privileges that user has and what actions she may perform

- in small company, identifying users individually doesn't present any problem
- in larger organization roles come in - although company may have large number of employees, these employees do a limited number of jobs; all sales clerks require same privileges, all warehouse workers require different privileges - they play different roles in company
- job of maintaining authorizations for everyone is made much simpler
- new user is added to SALES_CLERK role name and immediately gains privileges assigned to that role; sales clerk leaving company is deleted from role; employee changing from one job category to another is deleted from one role name and added to another
- just as session initiated by a user is associated with SQL-session user identifier, it is also associated with SQL-session role name

Roles

- **create role with :**
- **CREATE ROLE <role name>**
- [WITH ADMIN {CURRENT_USER | CURRENT_ROLE}];
- when create role, role is automatically granted to you; also granted right to pass role-creation privilege on to others
- **for destroying role:**
- **DROP ROLE <role name> ;**

Users classes of users

1. **DataBase Administrator (DBA):** responsibility of DBA is to maintain database; have full rights to all objects in database; can also decide what privileges other users may have
2. **Database object owners:** users who create database or database objects such as tables and views are automatically owners of those objects; possesses all privileges related to object; database object owner's privileges are equal to those of DBA, but only with respect to object in question
3. **Grantees:** are users who have been granted selected privileges by DBA or database object owner; may or may not be given right to grant her privileges to others
4. **public:** all users are considered to be part of public, regardless of whether they have been specifically granted any privileges; privileges that are granted to PUBLIC may be exercised by any user

Granting Privileges

- GRANT <privilege list>
- ON <privilege object>
- TO <user list> [WITH GRANT OPTION]
- [GRANTED BY {CURRENT_USER |
CURRENT_ROLE}] ;
- <privilege list> ::= privilege [, privilege]...

- **<privilege>** ::=
- SELECT [(<column name> [, <column name>]...)]
- | SELECT (<method designator> [, <method designator>...])
- | DELETE
- | INSERT [(<column name> [, <column name>]...)]
- | UPDATE [(<column name> [, <column name>]...)]
- | REFERENCES [(<column name> [, <column name>]...)]
- | USAGE
- | TRIGGER
- | UNDER
- | EXECUTE

- <privilege object> ::=
- [TABLE] <table name>
- | <view name>
- | DOMAIN <domain name>
- | CHARACTER SET <character set name>
- | COLLATION <collation name>
- | TRANSLATION <translation name>
- | TYPE <user-defined type name>
- | <specific routine designator>
- <user list> ::=
- authorizationID [, authorizationID]...
- | PUBLIC

- a lot of syntax ...
- not all privileges apply to all privilege objects
- SELECT, DELETE, INSERT, UPDATE, and REFERENCES privileges apply to TABLE
- SELECT privilege also applies to VIEWS
- USAGE privilege applies to DOMAIN, CHARACTER SET, COLLATION, and TRANSLATION
- TRIGGER privilege applies to TRIGGER
- UNDER privilege applies to user-defined types
- EXECUTE privilege applies to specific routines

- GRANT SELECT
- ON CUSTOMER
- TO SALES_MANAGER ;
- enables sales manager to query CUSTOMER table
- GRANT UPDATE
- ON CUSTOMER
- TO SALES_MANAGER ;
- enables the sales manager to change data

- GRANT ALL PRIVILEGES
- ON sinu
- TO rector;
- doing it all; for highly trusted person who has just been given major responsibility, rather than issuing a series

- GRANT UPDATE
- ON CUSTOMER
- TO SALES_MANAGER WITH GRANT OPTION ;
- passing on the power; similar to GRANT UPDATE example but also gives the right to grant update privilege to anyone she/he wants

Revoking Privileges

- REVOKE [GRANT OPTION FOR] <privilege list>
- ON <privilege object>
- FROM <user list>
- [GRANTED BY {CURRENT_USER |
CURRENT_ROLE}]
- {RESTRICT | CASCADE} ;

- major difference from GRANT syntax is addition of
- RESTRICT and CASCADE keywords and it isn't optional - one of two keywords is required
- SQL Server CASCADE keyword is optional, and RESTRICT sense is default assumed if CASCADE is not present
- if includes RESTRICT keyword, checks to see whether privilege being revoked was passed on to one or more other users; if it was, privilege isn't revoked, and you receive error message
- if includes CASCADE keyword, revokes privilege, as well as any dependent instances of this privilege that were granted by instance you're revoking

- with optional GRANT OPTION FOR clause, you can revoke user's ability to grant privilege without revoking his ability to use privilege

Granting Roles

- GRANT <role name> [{ , <role name>}...]
- TO <user list>
- [WITH ADMIN OPTION]
- [GRANTED BY {CURRENT_USER | CURRENT_ROLE}];
- can grant any number of roles to names in list of users
- if you want to grant role and extend to grantee right to grant same role to others, you do so with the WITH ADMIN OPTION clause

Revoking Roles

- REVOKE [ADMIN OPTION FOR] <role name> [{ , <role name>}...]
- FROM <user list>
- [GRANTED BY {CURRENT_USER | CURRENT_ROLE}]
- {RESTRICT | CASCADE}
- revoke one or more roles from users in user list
- can revoke admin option from role without revoking role itself

- 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.*

Structured Query Language

Aggregates

Grouping and Aggregates

- Aggregates are operator which has set operand
- Group By it is just a clause which forms sets which are passed on to aggregates operator / functions

Grouping and Aggregates

- sometimes you will want to find trends in your data that will require database server to cook data a bit before you can generate results
- `SELECT EmployeeID FROM Orders ORDER BY EmployeeID`
- With only 24 rows in the account table, it is relatively easy to see that four different
- with many rows difficult to see the number of orders made by employees

Grouping and Aggregates

- can ask database server to group data for you by using group by clause
- SELECT EmployeeID FROM Orders
- GROUP BY EmployeeID
- EmployeeID
- 1
- 2
- 3
- ...
- 8
- 9

Grouping and Aggregates

- to see how many Orders each Employee made you can use *aggregate function* in select clause to count number of rows in each group
- `SELECT EmployeeID, COUNT(EmployeeID) AS HowMany FROM Orders GROUP BY EmployeeID`
- EmployeeID HowMany
- 1 123
- 2 96
- 3 127
- ...
- 8 104
- 9 43

count()

- aggregate function counts number of rows in each group, and asterisk tells server to count everything in group
- SELECT EmployeeID, COUNT(*) AS HowMany
- FROM Orders
- GROUP BY EmployeeID

Having clause

- since group by clause runs *after* where clause has been evaluated, you cannot add filter conditions to your where clause
- SELECT EmployeeID, COUNT(*) AS HowMany FROM Orders
- GROUP BY EmployeeID **HAVING** COUNT(*) > 100
- EmployeeID HowMany
- 1 123
- 3 127
- 4 156
- 8 104

Having clause

- groups containing fewer than hundred orders have been filtered out via having clause, result set now contains only those employees who have made more then hundred orders

**SELECT * FROM
TestAggregation**

The screenshot shows a software interface with two tabs: "Results" and "Messages". The "Results" tab is active, displaying a table with three columns: "ID" and "Value". There are 9 rows of data. The first row has the ID 1 and Value 50.30. The second row has the ID 1 and Value 123.30. The third row has the ID 1 and Value 132.90. The fourth row has the ID 2 and Value 50.30. The fifth row has the ID 2 and Value 123.30. The sixth row has the ID 2 and Value 132.90. The seventh row has the ID 2 and Value 88.90. The eighth row has the ID 3 and Value 50.30. The ninth row has the ID 3 and Value 123.30.

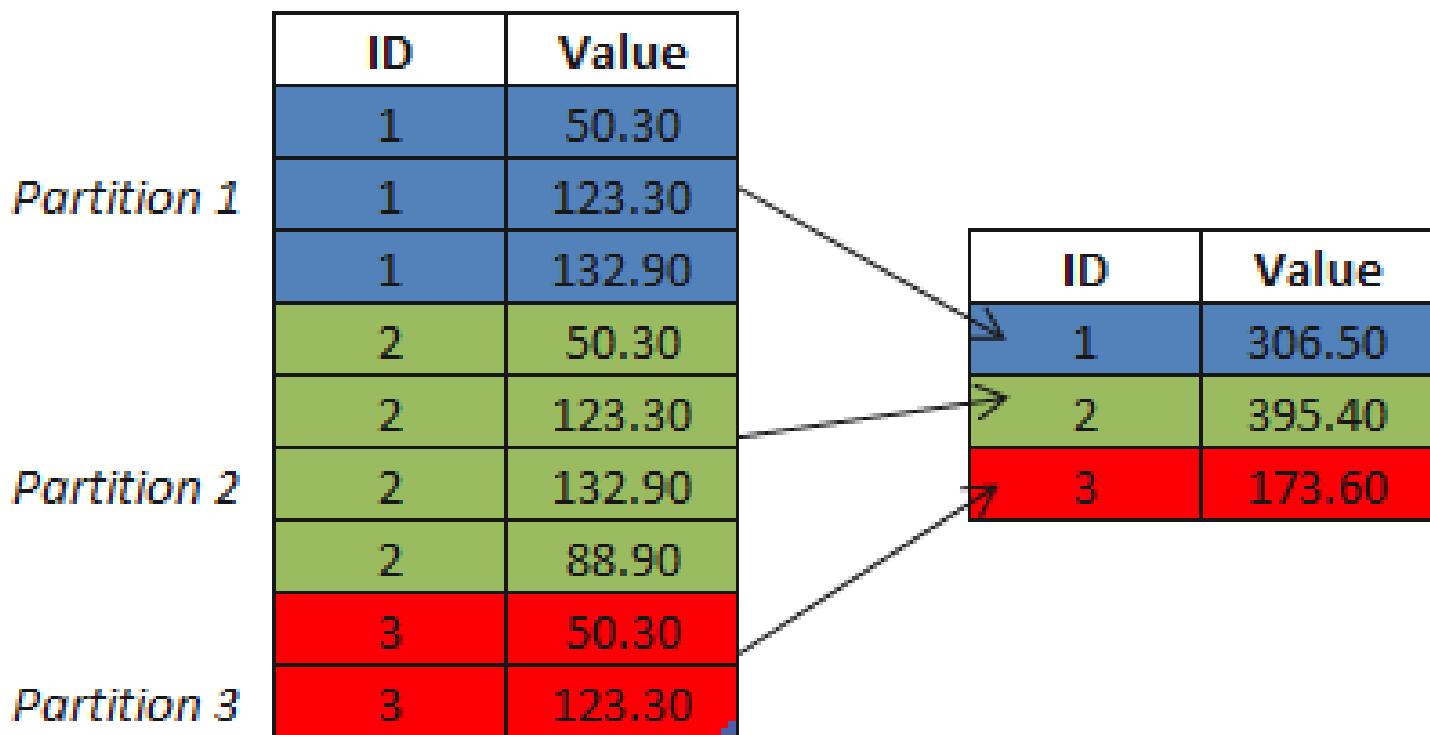
	ID	Value
1	1	50.30
2	1	123.30
3	1	132.90
4	2	50.30
5	2	123.30
6	2	132.90
7	2	88.90
8	3	50.30
9	3	123.30

**SELECT ID, SUM(Value)
FROM TestAggregation
GROUP BY ID;**

The screenshot shows a software interface with two tabs: "Results" and "Messages". The "Results" tab is active, displaying a table with three columns: "ID" and "(No column name)". There are 3 rows of data. The first row has the ID 1 and the value 306.50. The second row has the ID 2 and the value 395.40. The third row has the ID 3 and the value 173.60.

	ID	(No column name)
1	1	306.50
2	2	395.40
3	3	173.60

Partitions



... OVER(PARTITION BY ID) ...

ID	Value	Sum	Avg	Quantity		ID	Value	Sum	Avg	Quantity
1	50.30	306.50	102.166.666	3		1	50.30	306.50	102.166.666	3
1	123.30	306.50	102.166.666	3		1	123.30	306.50	102.166.666	3
1	132.90	306.50	102.166.666	3		1	132.90	306.50	102.166.666	3
2	50.30	395.40	98.850.000	4		2	50.30	395.40	98.850.000	4
2	123.30	395.40	98.850.000	4		2	123.30	395.40	98.850.000	4
2	132.90	395.40	98.850.000	4		2	132.90	395.40	98.850.000	4
2	88.90	395.40	98.850.000	4		2	88.90	395.40	98.850.000	4
3	50.30	173.60	86.800.000	2		3	50.30	173.60	86.800.000	2
3	123.30	173.60	86.800.000	2		3	123.30	173.60	86.800.000	2

Aggregate Functions

- perform specific operation over all rows in group
- belong to type of function known as ‘set function’, means function that applies to set of rows
- common aggregate functions implemented by all major servers include:

Aggregate Functions

- `Max()` - returns maximum value within set
- `Min()` - returns minimum value within set
- `Avg()` - returns average value across set
- `Sum()` - returns sum of values across set
- `Count()` - returns number of values in set

- SELECT
- MAX(UnitPrice) max_price,
- MIN(UnitPrice) min_price,
- AVG(UnitPrice) avg_price,
- SUM(UnitPrice) total_price,
- COUNT(UnitPrice) num_products
- FROM Products JOIN Categories ON
Categories.CategoryID = Products.CategoryID

max_price min_price avg_price total_price
num_products

- 263.50 0.00 28.4962 2222.71 78

- results from this query tell you that, across all products there is maximum value of UnitPrice \$263.50, a minimum values of \$0.00, an average value of UnitPrice of \$28.49, and total values of \$2,222.71 across all 78 products
- every value returned by query is generated by aggregate function, and since there is no group by clause, there is a single, *implicit* group (all rows returned by query)

- in most cases, however, you will want to retrieve additional columns along with columns generated by aggregate functions
- what if, for example, you wanted to extend previous query to execute the same aggregate functions for *each* product category type, instead of just for checking all products
- you would want to retrieve product's category name column along with aggregate functions

- SELECT CategoryName,
- MAX(UnitPrice) max_price,
- MIN(UnitPrice) min_price,
- AVG(UnitPrice) avg_price,
- SUM(UnitPrice) total_price,
- COUNT(UnitPrice) num_products
- FROM Products JOIN Categories ON
Categories.CategoryID = Products.CategoryID
- GROUP BY CategoryName

CategoryName	max_price	min_price	avg_price	total_price	num_products
Beverages	263.50	0.00	35.05	455.75	13
Condiments	43.90	10.00	23.06	276.75	12
Confections	81.00	9.20	25.16	327.08	13
Dairy Products	55.00	2.50	28.73	287.30	10
Grains/Cereals	38.00	7.00	20.25	141.75	7
Meat/Poultry	123.79	7.45	54.00	324.04	6
Produce	53.00	10.00	32.37	161.85	5
Seafood	62.50	6.00	20.68	248.19	12

- with the inclusion of group by clause, server knows to group together rows having same value in CategoryName column first and then to apply aggregate functions to each groups

Counting Distinct Values

- when using `count()` function to determine number of members in each group, have choice of counting *all* members or counting only *distinct* values for column across all members of group
- `sum()`, `max()`, `min()`, and `avg()` functions all ignore any null value

- `SELECT COUNT(*) FROM Customers`
- *92 rows in table Customers*
- `SELECT COUNT(Region) FROM Customers`
- *32 rows in which Region column it is not NULL*
- `SELECT COUNT(DISTINCT Region) FROM Customers`
- *19 such distinct values representing Region*

Grouping

- single-column
- multicolumn
- via expressions

- SELECT CategoryName, Products.ProductName,
- MAX(UnitPrice) max_price,
- MIN(UnitPrice) min_price,
- AVG(UnitPrice) avg_price,
- SUM(UnitPrice) total_price,
- COUNT(UnitPrice) num_products
- FROM Products JOIN Categories ON
Categories.CategoryID = Products.CategoryID
- GROUP BY CategoryName

Error

- Column 'Products.ProductName' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause
- on grouping and using aggregates you cannot use in SELECT clause expression which are not either aggregate or are used in GROUP BY
- previous error – server don't know what productName to present

- SELECT CategoryName
- MAX(UnitPrice) max_price,
- MIN(UnitPrice) min_price,
- AVG(UnitPrice) avg_price,
- SUM(UnitPrice) total_price,
- COUNT(UnitPrice) num_products
- FROM Products JOIN Categories ON
Categories.CategoryID = Products.CategoryID
- GROUP BY CategoryID

- SELECT CategoryName,
- MAX(UnitPrice) max_price,
- MIN(UnitPrice) min_price,
- AVG(UnitPrice) avg_price,
- SUM(UnitPrice) total_price,
- COUNT(UnitPrice) num_products
- FROM Products JOIN Categories ON
Categories.CategoryID = Products.CategoryID
- GROUP BY Products.CategoryID

SubQueries

subquery

- query contained within another SQL statement (which we refer to as *containing statement*)
- always enclosed within parentheses
- usually executed prior to containing statement
- returns result set that may consist of:
 - single row with single column
 - multiple rows with single column
 - multiple rows and columns

- when containing statement has finished executing, data returned by any subqueries is discarded, making subquery act like temporary table with *statement scope*
- one of most powerful tools in SQL

- if you use subquery in equality condition, but subquery returns more than one row, you will receive error
- `SELECT * FROM Products WHERE UnitPrice = (`
- `SELECT MAX(UnitPrice) FROM Products)`
- *better* - single thing cannot be equated to set
- `SELECT * FROM Products WHERE UnitPrice IN (`
- `SELECT MAX(UnitPrice) FROM Products)`

- along with differences regarding type of result set subquery returns (single/multiple rows and columns)
- subqueries are completely selfcontained (*called noncorrelated subqueries*), while others reference columns from containing statement (*called correlated subqueries*)

Correlated Subqueries

- is *dependent* on its containing statement from which it references one or more columns
- executed once for each candidate row (rows that might be included in final results)
- noncorrelated subquery is executed once prior to execution of containing statement

- SELECT DISTINCT Customers.CompanyName FROM Orders JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
- JOIN Products ON Products.ProductID = OrderDetails.ProductID
- JOIN Categories ON Categories.CategoryID = Products.CategoryID
- JOIN Customers ON Orders.CustomerID = Customers.CustomerID
- WHERE Categories.CategoryName = 'Beverages'
- ORDER BY CompanyName

- `SELECT DISTINCT Customers.CompanyName FROM Customers`
- `WHERE Customers.CustomerID IN (`
- *`SELECT DISTINCT Orders.CustomerID FROM Orders JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID`*
- *`JOIN Products ON Products.ProductID = OrderDetails.ProductID`*
- *`JOIN Categories ON Categories.CategoryID = Products.CategoryID`*
- *`WHERE Categories.CategoryName = 'Beverages')`*
- `ORDER BY CompanyName`

- Customers who Ordered Beverages
- Customers who do not Ordered Beverages

- SELECT DISTINCT Customers.CompanyName FROM Orders JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
- JOIN Products ON Products.ProductID = OrderDetails.ProductID
- JOIN Categories ON Categories.CategoryID = Products.CategoryID
- JOIN Customers ON Orders.CustomerID = Customers.CustomerID
- WHERE Categories.CategoryName != 'Beverages'
- ORDER BY CompanyName

- SELECT DISTINCT Customers.CompanyName FROM Customers
- WHERE Customers.CustomerID NOT IN (
- SELECT DISTINCT Orders.CustomerID FROM Orders JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
- JOIN Products ON Products.ProductID = OrderDetails.ProductID
- JOIN Categories ON Categories.CategoryID = Products.CategoryID
- WHERE Categories.CategoryName = 'Beverages')
- ORDER BY CompanyName

all operator

- in (membership) operator is used to see whether an element, expression can be found within set
- all operator allows you to make comparisons between single value and every value in set
- to build such condition, you will need to use one of comparison operators (=, <>, <, >, etc.) in conjunction with all operator

any operator

- any operator allows value to be compared to members of set of values
- unlike all, however, condition using any operator evaluates to true as soon as single comparison is favorable
- all operator evaluates to true only if comparisons against *all* members of set are favorable

Set-Comparison Operators

- SQL also supports op ANY and op ALL, where op is one of the arithmetic comparison operators {<,<=,=,<>,>=,>}
- SOME is also available, but it is just a synonym for ANY

Definition of Some (ANY) Clause

- $F <\text{comp}> \text{some } r \Leftrightarrow \exists t \in r \text{ s.t. } (F <\text{comp}> t)$

Where comp can be: $<$, \leq , $>$, $=$, \neq

(5 < some	0
	5
	6

) = true

(read: 5 < some tuple in the relation)

(5 < some	0
	5

) = false

(5 = some	0
	5

) = true

(5 ≠ some	0
	5

) = true (since $0 \neq 5$)

$(= \text{some}) \equiv \text{in}$

However, $(\neq \text{some}) \equiv \text{not in}$

Definition of All (ALL) Clause

- $F <\text{comp}> \text{all } r \Leftrightarrow \forall t \in r (F <\text{comp}> t)$

$(5 < \text{all} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline\end{array}) = \text{false}$

$(5 < \text{all} \begin{array}{|c|}\hline 6 \\ \hline 10 \\ \hline\end{array}) = \text{true}$

$(5 = \text{all} \begin{array}{|c|}\hline 4 \\ \hline 5 \\ \hline\end{array}) = \text{false}$

$(5 \neq \text{all} \begin{array}{|c|}\hline 4 \\ \hline 6 \\ \hline\end{array}) = \text{true} (\text{since } 5 \neq 4 \text{ and } 5 \neq 6)$

$(\neq \text{all}) \equiv \text{not in}$

However, $(= \text{all}) \equiv \text{in}$

/

Test for Empty Relations

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$
- except Operator

- Most expensive Beverage
- Most expensive Product from each Category

- SELECT * FROM Products
- WHERE UnitPrice = (
- SELECT MAX(UnitPrice) FROM Products JOIN Categories ON Products.CategoryID = Categories.CategoryID
- WHERE Categories.CategoryName = 'Beverages')

- `SELECT * FROM Products P1 JOIN Categories C1 ON P1.CategoryID = C1.CategoryID`
- `WHERE C1.CategoryName = 'Beverages' AND UnitPrice = (SELECT MAX(UnitPrice) FROM Products P2 JOIN Categories C2 ON P2.CategoryID = C2.CategoryID`
- `WHERE C2.CategoryName = 'Beverages')`

- `SELECT C1.CategoryName, P1.ProductName,
P1.UnitPrice FROM Products P1 JOIN Categories
C1 ON P1.CategoryID = C1.CategoryID`
- `WHERE UnitPrice >= ALL (`
- `SELECT UnitPrice FROM Products P2 JOIN
Categories C2 ON P2.CategoryID = C2.CategoryID`
- `WHERE C2.CategoryName = C1.CategoryName)`

CategoryName	ProductName	UnitPrice
--------------	-------------	-----------

- Seafood Carnarvon Tigers 62.50
- Confections Sir Rodney's Marmalade 81.00
- Meat/Poultry Thüringer Rostbratwurst 123.79
- Beverages Côte de Blaye 263.50
- Produce Manjimup Dried Apples 53.00
- Grains/Cereals Gnocchi di nonna Alice 38.00
- Dairy Products Raclette Courdavault 55.00
- Condiments Vegie-spread 43.90

Test for Empty Relations

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$
- 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 \emptyset empty set

- SELECT CompanyName, COUNT(*) FROM CustomCateg
- GROUP BY CompanyName
- ORDER BY 2 DESC

- SELECT DISTINCT CompanyName FROM CustomCateg C1
- WHERE NOT EXISTS (
- SELECT Categories.CategoryName FROM Categories
- EXCEPT
- (SELECT C2.CategoryName FROM CustomCateg C2
- WHERE C2.CompanyName = C1.CompanyName))

exists Operator

- most common operator used to build conditions that utilize correlated subqueries is the exists operator
- to identify that relationship exists without regard for the quantity
- subquery can return zero, one, or many rows, and condition simply checks whether the subquery returned any rows

Data Manipulation Using Correlated Subqueries

- UPDATE table t1
- SET t1.column =
- (SELECT expression
- FROM table t2
- WHERE t1.some_column =
t2.some_other_column);

Data Manipulation Using Correlated Subqueries

- SELECT Products.ProductID, Products.UnitPrice
- FROM Products
- ORDER BY ProductID
- SELECT OrderDetails.ProductID,
SUM(OrderDetails.Quantity*OrderDetails.UnitPrice) / SUM(OrderDetails.Quantity) AS AvgPrice
- FROM OrderDetails
- GROUP BY OrderDetails.ProductID
- ORDER BY ProductID

- | • ProductID | UnitPrice | • ProductID | AvgPrice |
|-------------|-----------|-------------|----------|
| • 1 | 18.00 | • 1 | 17.2434 |
| • 2 | 19.00 | • 2 | 17.5583 |
| • 3 | 10.00 | • 3 | 9.3902 |
| • 4 | 22.00 | • 4 | 20.8052 |
| • 5 | 21.35 | • 5 | 19.4669 |
| • 6 | 25.00 | • 6 | 24.4019 |
| • 7 | 30.00 | • 7 | 29.4416 |
| • 8 | 40.00 | • 8 | 36.9892 |
| • 9 | 97.00 | • 9 | 92.9157 |
| • 10 | 31.00 | • 10 | 29.8385 |
| • 11 | 21.00 | • 11 | 19.6912 |
| • 12 | 38.00 | • 12 | 37.4034 |

- UPDATE Products
- SET Products.UnitPrice =
- (SELECT POD.AvgPrice) FROM (
- SELECT OrderDetails.ProductID,
SUM(OrderDetails.Quantity*OrderDetails.UnitPrice) / SUM(OrderDetails.Quantity) AS AvgPrice
- FROM OrderDetails
- GROUP BY OrderDetails.ProductID) POD
- JOIN Products ON POD.ProductID =
Products.ProductID

Temp. table

- SELECT POD.ProductID, POD.AvgPrice FROM (
- SELECT OrderDetails.ProductID,
 SUM(OrderDetails.Quantity*OrderDetails.UnitPrice) / SUM(OrderDetails.Quantity) AS AvgPrice
- FROM OrderDetails
- GROUP BY OrderDetails.ProductID) POD
- JOIN Products ON POD.ProductID =
Products.ProductID

- Correlated subqueries are also common in delete statements
- run data maintenance script at end of period that removes unnecessary data.
- removes data from department table that has no child rows in employee table:
 - DELETE FROM department
 - WHERE NOT EXISTS (SELECT 1
 - FROM employee
 - WHERE employee.dept_id = department.dept_id);

Outer JOIN

- in all examples thus far that have included multiple tables, we haven't been concerned that join conditions might fail to find matches for all rows in tables
- for example, when joining Categories table to Products table, we did not mention possibility that
- value in CategoryID column of Products table might not match value in CategoryID column of Categories table
- if that were the case, then some of rows in one table (from left side) or other (from right side) would be left out of result set

- **SELECT * FROM Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID**
- **SELECT * FROM Categories LEFT OUTER JOIN Products ON Categories.CategoryID = Products.CategoryID**
- **SELECT * FROM Categories RIGHT OUTER JOIN Products ON Categories.CategoryID = Products.CategoryID**
- **SELECT * FROM Categories FULL OUTER JOIN Products ON Categories.CategoryID = Products.CategoryID**

- if you want your query to return *all* Products, *all* Categories need an *outer join* (FULL OUTER JOIN)
 - to include Categories without Products
 - to include Products without Categories
- columns are Null for all rows except for the matching values Categories.CategoryID = Products.CategoryID

Self Outer Joins

- `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`

Self Outer Joins

- `SELECT s.EmployeeID, s.FirstName, s.LastName,
s.Title, s.ReportsTo, m.EmployeeID, m.FirstName,
m.LastName, m.Title`
- `FROM Employees s LEFT OUTER JOIN
Employees m ON s.ReportsTo = m.EmployeeID`

- EmployeeID FirstName LastName Title ReportsTo
EmployeeID FirstName LastName Title
- 1 Nancy Davolio Sales Representative 2 2
Andrew Fuller Vice President, Sales
- 2 *Andrew Fuller* *Vice President, Sales* *NULL* *NULL* *NULL*
NULL *NULL*
- 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
- ...

Cross Joins

- **Cartesian product - result of joining multiple tables without specifying any join conditions**
- are not so common ... if, however, you *do* intend to generate Cartesian product of two tables specify *cross join*:
- `SELECT * FROM table t1 CROSS JOIN table t2;`

CASE

- in certain situations, you may want your SQL logic to branch in one direction or another depending on values of certain expressions.
- statements that can behave differently depending on data encountered during execution

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

- TitleOfCourtesy FirstName LastName
- Ms. Nancy Davolio
- Dr. Andrew Fuller
- Ms. Janet Leverling
- Mrs. Margaret Peacock
- Mr. Steven Buchanan
- Mr. Michael Suyama
- Mr. Robert King
- Ms. Laura Callahan
- Ms. Anne Dodsworth

- SELECT
- CASE
 - WHEN TitleOfCourtesy = 'Mr.'
 - THEN 'Mister'
 - WHEN TitleOfCourtesy = 'Mrs.'
 - THEN 'Missus'
 - WHEN TitleOfCourtesy = 'Ms.'
 - THEN 'Miss'
 - WHEN TitleOfCourtesy = 'Dr.'
 - THEN 'Doctor'
 - ELSE '' END Tit,
- FirstName, LastName FROM Employees

- CASE
 - WHEN TitleOfCourtesy = 'Mr.'
 - THEN 'Mister'
 - WHEN TitleOfCourtesy = 'Mrs.'
 - THEN 'Missus'
 - WHEN TitleOfCourtesy = 'Ms.'
 - THEN 'Miss'
 - WHEN TitleOfCourtesy = 'Dr.'
 - THEN 'Doctor'
 - ELSE '' END Tit,

- you could use conditional logic via *case expression*
- CASE
 - WHEN C_1 THEN E_1
 - WHEN C_2 THEN E_2
- ...
 - WHEN C_n THEN E_n
 - [ELSE ED]
- END

- symbols C_1, C_2, \dots, C_n represent conditions
- symbols E_1, E_2, \dots, E_n represent expressions to be returned by case expression
- if condition in when clause evaluates to true, then case expression returns corresponding expression
- ED symbol represents default expression, which case expression returns if *none* of conditions evaluate to true
 - else clause is optional

- although previous example returns string expressions, keep in mind that case expressions may return any type of expression, including subqueries – but with errors if subquery returned more than 1 value

- SELECT E.Title, E.FirstName, E.LastName,
- CASE
- WHEN E.ReportsTo IS NOT NULL THEN
- (SELECT CONCAT(i.FirstName, ' ', i.LastName)
FROM Employees i
- WHERE i.EmployeeID = E.ReportsTo)
- ELSE 'Unknown'
- END Boss
- FROM Employees E;

- Title FirstName LastName Boss
- Sales Representative Nancy Davolio Andrew Fuller
- Vice President, Sales Andrew Fuller Unknown
- Sales Representative Janet Leverling Andrew Fuller
- Sales Representative Margaret Peacock Andrew Fuller
- Sales Manager Steven Buchanan Andrew Fuller
- Sales Representative Michael Suyama Steven Buchanan
- Sales Representative Robert King Steven Buchanan
- Inside Sales CoordinatorLaura Callahan Andrew Fuller
- Sales Representative Anne Dodsworth Steven Buchanan

many other aspects related to SQL

- Transactions, Indexes, Constraints, Views
- Metadata
- Window function
- pre defined function (character strings, numbers, calendar data)
 - standard SQL or implemented in particular DB engine like MySQL or SQL Server

- 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.*

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 a SQL database called SQLite
 - many applications on your phone use it directly

- Who said ... *inches ... are everywhere* ... And in what context (movie, book, opera, ...)

learning SQL ?

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

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 database
- matches theory established many years ago defining properties of well structured data
- not exactly the same (which some detractors lament) but it's close enough to be useful

SQL operations

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

SQL - language for doing CRUD operations

- 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 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 its 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

points left up to vendors to implement

- semantic and syntactic differences
- opening database for processing
 - interfaces of ODBC, CLI, OLEDB, and others are not part of any standard
- dynamic and embedded SQL implementations might differ
- collating order
 - depends on whether ASCII or EBCDIC characters are used
 - UNICODE standard alleviates this problem
- different data types extensions
- differences in database catalog tables

CREATE statement

- used to create all objects that comprise database: tables, indices, constraints, ...

CREATE TABLE

- name of the column and its data type
- CREATE TABLE status (
 - status_id_n INT,
 - status_code_s CHAR(2),
 - status_desc_s VARCHAR(30))

DROP TABLE

- to dispose of table, or many other objects in the RDBMS
- DROP TABLE status

Getting the data in and out

- **INSERT**
 - adds new data to table
- **UPDATE**
 - updates data — i.e., changes existing values
- **DELETE**
 - removes data from table
- **SELECT**
 - retrieves data from database table

- INSERT INTO status (STATUS_ID_N,
STATUS_CODE_S, STATUS_DESC_S)
 - VALUES (8,'70','INVOICED')
- UPDATE status SET
 - status_desc_s = 'APPROVED',
 - status_code_s = '90'
 - WHERE status_id_n = 8
- DELETE status
 - WHERE status_id_n = 8

Data security

- comes down to granting access on object level: ability to connect and view particular table, set of tables, execute command
- Privileges
- assuming that there is a user JOHN_DOE defined in database
- to grant a permission :
 - GRANT SELECT ON v_custome_totals TO john_doe
 - GRANT SELECT, UPDATE ON v_custome_totals TO john_doe
- to revoke this privilege:
 - REVOKE SELECT ON v_custome_totals FROM john_doe
 - REVOKE ALL ON v_custome_totals FROM john_doe

SQL SELECT

sqlzoo.net

- A Gentle Introduction to
- Structured Query Language
- 3 - 4 laboratory works

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 Technical 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

Sample queries using following tables schema

- Sailors
 - (sid: integer, sname: string, rating: integer, age: real)
- Boats
 - (bid: integer, bname: string, color: string)
- Reserves
 - (sid: integer, bid: integer, day: date)

Sailors (sid, sname, rating, age)

- 22 Dustin 7 45.0
- 29 Brutus 1 33.0
- 31 Lubber 8 55.5
- 32 Andy 8 25.5
- 58 Rusty 10 35.0
- 64 Horatio 7 35.0
- 71 Zorba 10 16.0
- 74 Horatio 9 35.0
- 85 Art 3 25.5
- 95 Bob 3 63.5

Boats(bid, bname, color)

- 101 Interlake blue
- 102 Interlake red
- 103 Clipper green
- 104 Marine red

Reserves (sid, bid, day)

- 22 101 10/10/98
- 22 102 10/10/98
- 22 103 10/8/98
- 22 104 10/7/98
- 31 102 11/10/98
- 31 103 11/6/98
- 31 104 11/12/98
- 64 101 9/5/98
- 64 102 9/8/98
- 74 103 9/8/98

SQL

- basic form of an SQL query is as follows:
- SELECT [DISTINCT] *select-list*
- FROM *from-list*
- WHERE *qualification*

SELECT

- **from-list**
 - list of table names
 - table name can be followed by *range variable, an alias,* particularly useful when same table name appears more than once in from-list (many copies)
- **select-list**
 - is a list of (expressions involving) column names of tables named in from-list
 - column names can be prefixed by a range variable

SELECT

- **qualification**
 - WHERE clause is a boolean combination (i.e., an expression using logical connectives AND, OR, and NOT)
 - of conditions of form *left express1 op right express2*,
 - where op is one of the comparison operators
 $\{<, \leq, =, >, \geq, >\}$
 - *expression* is a *column* name, a *constant*, or an (arithmetic or string or calendar date) expression

Examples

- *Find the names and ages of all sailors.*
 - SELECT DISTINCT S.sname, S.age
 - FROM Sailors S
-
- *Find all sailors with a rating above 7.*
 - SELECT S.sid, S.sname, S.rating, S.age
 - FROM Sailors AS S
 - WHERE S.rating > 7

SELECT

- DISTINCT keyword
 - is optional
 - indicates that table computed as answer to this query should not contain *duplicates*, that is, two copies of the same row
 - default is that duplicates are not eliminated

Examples

- *Find the names of sailors who have reserved boat number 103.*
- SELECT S.sname
- FROM Sailors S, Reserves R
- WHERE S.sid = R.sid AND R.bid=103

Examples

- *Compute increments for the ratings of persons who have sailed two different boats on the same day*
- SELECT S.sname, S.rating+1 AS newRrating
- FROM Sailors S, Reserves R1, Reserves R2
- WHERE S.sid = R1.sid AND S.sid = R2.sid
AND R1.day = R2.day AND R1.bid <> R2.bid

String comparisons

- can use comparison operators (=,<,>, etc.) with ordering of strings determined alphabetically as usual
- pattern matching through LIKE operator and use of wild-card symbols
 - % stands for zero or more arbitrary characters
 - _ stands for exactly one, arbitrary, character

Collation

- if we need to sort strings by an order other than alphabetical (e.g., sort strings denoting month names in the calendar order January, February, March, etc.), SQL-92 supports a general concept of a *collation*, or sort order, for a character set
- allows user to specify which characters are ‘less than’ which others, and provides great flexibility in string manipulation
- different languages – different orders

SQL

- form of an SQL query is as follows:
- `SELECT [DISTINCT] select-list`
- `FROM from-list`
- `[WHERE qualification]`
- `[ORDER BY order_expression [ASC/DESC]]`

Examples

- *Find all information, about all sailors, show result in such way that higher ratings sailor are shown first*
- SELECT * FROM Sailors
- ORDER BY rating DESC

More Examples

- *Find the names of sailors who have reserved a red or a green boat.*
- SELECT S.sname
- FROM Sailors S, Reserves R, Boats B
- WHERE S.sid = R.sid AND R.bid = B.bid
- AND (B.color = 'red' OR B.color = 'green')
- *Find the names of sailors who have reserved both a red and a green boat.*

Examples

- SELECT S.sname
- FROM Sailors S, Reserves R1, Boats B1, Reserves R2, Boats B2
- WHERE S.sid = R1.sid AND R1.bid = B1.bid
- AND S.sid = R2.sid AND R2.bid = B2.bid
- AND B1.color='red' **AND** B2.color = 'green'

Examples

- SELECT S.sname
- FROM Sailors S, Reserves R, Boats B
- WHERE S.sid = R.sid AND
 R.bid = B.bid AND B.color = `red'
- **UNION**
- SELECT S2.sname
- FROM Sailors S2, Boats B2, Reserves R2
- WHERE S2.sid = R2.sid AND
 R2.bid = B2.bid AND B2.color = `green'

Examples

- SELECT S.sname
- FROM Sailors S, Reserves R, Boats B
- WHERE S.sid = R.sid AND
 R.bid = B.bid AND B.color = `red'
- **INTERSECT**
- SELECT S2.sname
- FROM Sailors S2, Boats B2, Reserves R2
- WHERE S2.sid = R2.sid AND
 R2.bid = B2.bid AND B2.color = `green'

Examples

- SELECT S.sname
- FROM Sailors S, Reserves R, Boats B
- WHERE S.sid = R.sid AND
 R.bid = B.bid AND B.color = `red'
- **EXCEPT**
- SELECT S2.sname
- FROM Sailors S2, Boats B2, Reserves R2
- WHERE S2.sid = R2.sid AND
 R2.bid = B2.bid AND B2.color = `green'

Examples

- *Find the names of all sailors who have reserved red boats but not green boats.*

Nested Queries

- has another query embedded within it
- embedded query is called a *subquery*

Examples

- *Find the names of sailors who have reserved boat 103.*
- SELECT S.sname
- FROM Sailors S
- WHERE S.sid IN
 - (SELECT R.sid
 - FROM Reserves R
 - WHERE R.bid = 103)

Nested Queries

- IN (\in) member of operator allows us to test whether a value is in a given set of elements
 - an SQL query can be used to generate the set to be tested
- very easy to modify this query to find all sailors who have *not* reserved boat 103, replacing IN by NOT IN

Correlated Nested Queries

- inner subquery could depend on the row that is currently being examined in the outer query (in terms of our conceptual evaluation strategy)
- EXISTS operator is another set comparison operator, such as IN
- allows us to test whether a set is nonempty

Examples

- *Find the names of sailors who have reserved boat number 103.*
- SELECT S.sname
- FROM Sailors S
- WHERE EXISTS (SELECT *
 - FROM Reserves R
 - WHERE R.bid = 103
 - AND R.sid = **S.sid**)

- by using NOT EXISTS instead of EXISTS, we can compute the names of sailors who have not reserved a red boat

Set-Comparison Operators

- SQL also supports op ANY and op ALL, where op is one of the arithmetic comparison operators {<,<=,=,<>,>=,>}
- SOME is also available, but it is just a synonym for ANY

Definition of Some (ANY) Clause

- $F <\text{comp}> \text{some } r \Leftrightarrow \exists t \in r \text{ s.t. } (F <\text{comp}> t)$

Where comp can be: $<$, \leq , $>$, $=$, \neq

(5 < some	0
	5
	6

) = true

(read: 5 < some tuple in the relation)

(5 < some	0
	5

) = false

(5 = some	0
	5

) = true

(5 ≠ some	0
	5

) = true (since $0 \neq 5$)

$(= \text{some}) \equiv \text{in}$

However, $(\neq \text{some}) \equiv \text{not in}$

Definition of All (ALL) Clause

- $F <\text{comp}> \text{all } r \Leftrightarrow \forall t \in r (F <\text{comp}> t)$

$(5 < \text{all} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline\end{array}) = \text{false}$

$(5 < \text{all} \begin{array}{|c|}\hline 6 \\ \hline 10 \\ \hline\end{array}) = \text{true}$

$(5 = \text{all} \begin{array}{|c|}\hline 4 \\ \hline 5 \\ \hline\end{array}) = \text{false}$

$(5 \neq \text{all} \begin{array}{|c|}\hline 4 \\ \hline 6 \\ \hline\end{array}) = \text{true} (\text{since } 5 \neq 4 \text{ and } 5 \neq 6)$

$(\neq \text{all}) \equiv \text{not in}$

However, $(= \text{all}) \equiv \text{in}$

/

Test for Empty Relations

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$

Examples

- *Find sailors whose rating is better than some sailor called Horatio.*
- SELECT S.sname
- FROM Sailors S
- WHERE S.rating > ANY (
 - SELECT S2.rating
 - FROM Sailors S2
 - WHERE S2.sname = 'Horatio')

- if there are several sailors called Horatio
 - query finds all sailors whose rating is better than that of *some* sailor called Horatio
- if there were *no* sailor called Horatio
 - comparison $S.rating > \text{ANY}$ is defined to return false, and above query returns an empty answer set

- To understand comparisons involving ANY, it is useful to think of the comparison being carried out repeatedly. In the example above, *S.rating* is successively compared with each rating value that is an answer to the nested query. Intuitively, the subquery must return a row that makes the comparison true, in order for *S.rating* > ANY ... to return true.

Examples

- *Find the sailors with the highest rating.*
- SELECT S.sid
- FROM Sailors S
- WHERE S.rating >= ALL
 - (SELECT S2.rating
 - FROM Sailors S2)

- If there were no sailor called Horatio, comparison $S.rating > \text{ALL}$ is defined to return true
- again useful to think of comparison being carried out repeatedly
- intuitively, comparison must be true for every returned row in order for $S.rating > \text{ALL} \dots$ to return true

- IN and NOT IN are equivalent to
- = ANY and <> ALL, respectively
- *Find the names of sailors who have reserved all boats.*

Examples

- SELECT S.sname
- FROM Sailors S
- WHERE NOT EXISTS
 - ((SELECT B.bid
 - FROM Boats B)
 - EXCEPT
 - (SELECT R.bid
 - FROM Reserves R
 - WHERE R.sid = S.sid))

Examples

- SELECT S.sname
- FROM Sailors S
- WHERE NOT EXISTS
 - (SELECT B.bid
 - FROM Boats B
 - WHERE NOT EXISTS
 - (SELECT R.bid
 - FROM Reserves R
 - WHERE R.bid = B.bid
 - AND R.sid = S.sid)

SQL

- general form of an SQL query :
- SELECT [DISTINCT] *select-list*
- FROM *from-list*
- [WHERE *qualification*]
- [GROUP BY *group_by_expression*]
[HAVING *search_condition*]
- [ORDER BY *order_expression* [ASC/DESC]]

Aggregate Operators

- perform some computation or summarization on a *set of values*
- SQL allows use of arithmetic expressions
- powerful class of constructs for computing *aggregate values* such as MIN and SUM
- SQL supports aggregate operations, which can be applied on any column, say A, of a relation:

Aggregate Operators

1. COUNT ([DISTINCT] A)
 - number of (unique) values in the A column
2. SUM ([DISTINCT] A)
 - sum of all (unique) values in the A column
3. AVG ([DISTINCT] A)
 - average of all (unique) values in the A column
4. MAX (A)
 - maximum value in the A column
5. MIN (A)
 - minimum value in the A column

Examples

- *Find the average age of all sailors.*
 - SELECT AVG (S.age)
 - FROM Sailors S
-
- *Find the average age of sailors with a rating of 10.*
 - SELECT AVG (S.age)
 - FROM Sailors S
 - WHERE S.rating = 10

Examples

- *Find the name and age of the oldest sailor.*
- SELECT S.sname, MAX (S.age)
- FROM Sailors S

Examples

- query is illegal in SQL, if the SELECT clause uses an aggregate operation, then it must use *only* aggregate operations unless the query contains a GROUP BY clause!
- SELECT S.sname, S.age
- FROM Sailors S
- WHERE S.age =
 - (SELECT MAX (S2.age)
 - FROM Sailors S2)

- because of the use of the aggregate operation, the subquery is guaranteed to return a single tuple with a single field
- SQL converts such a relation to a field value for the sake of the comparison

Examples

- Aggregate operations offer an alternative to the ANY and ALL constructs
- *Find the names of sailors who are older than the oldest sailor with a rating of 10*

Examples

- SELECT S.sname
- FROM Sailors S
- WHERE S.age > ALL
 - (SELECT S2.age
 - FROM Sailors S2
 - WHERE S2.rating = 10)

Examples

- SELECT S.sname
- FROM Sailors S
- WHERE S.age >
 - (SELECT MAX (S2.age)
 - FROM Sailors S2
 - WHERE S2.rating = 10)

Examples

- *Find the age of the youngest sailor for each rating level.*
- SELECT MIN (S.age)
- FROM Sailors S
- WHERE S.rating = i

GROUP BY and HAVING

- SELECT S.rating, MIN (S.age)
- FROM Sailors S
- GROUP BY S.rating

GROUP BY and HAVING

- GROUP BY grouping-list
 - obtain set of values
 - apply aggregate operators on these sets
- HAVING group-qualification
 - kind of WHERE for the result of GROUP BY

select-list

- consists of a list of column names (1) and a list of terms having the form $aggop$ (*column-name*) AS *new-name*
 - optional AS *new-name* term gives this column name in table that is result of query
 - any of aggregation operators can be used for *aggop*.
- Every column that appears in (1) must also appear in grouping-list
 - Reason is that each row in result corresponds to one *group*, which is collection of rows that agree on values of columns in grouping-list
 - if column appears in list (1), but not in grouping-list, it is not clear what value should be assigned to it in an answer row

group-qualification

- expressions appearing in HAVING clause must have a *single* value per group
- HAVING clause determines whether an answer row is to be generated for a given group
- column appearing in *group-qualification* must appear as the argument to an aggregation operator, or it must also appear in *grouping-list*
- if GROUP BY clause is omitted, entire table is regarded as a single group

Examples

- *Find the age of the youngest sailor who is eligible to vote (i.e., is at least 18 years old) for each rating level with at least two such sailors.*

Examples

- SELECT S.rating, MIN (S.age) AS MAge
- FROM Sailors S
- WHERE S.age >= 18
- GROUP BY S.rating
- HAVING COUNT (*) > 1
- *For each red boat, find the number of reservations for this boat.*

Examples

- SELECT B.bid, COUNT (*) AS sailorcount
- FROM Boats B, Reserves R
- WHERE R.bid = B.bid AND B.color = 'red'
- GROUP BY B.bid
- SELECT B.bid, COUNT (*) AS sailorcount
- FROM Boats B, Reserves R
- WHERE R.bid = B.bid
- GROUP BY B.bid
- HAVING B.color = 'red'

- Only columns that appear in the GROUP BY clause can appear in the HAVING clause, unless they appear as arguments to an aggregate operator in the HAVING clause

Null Values

- when a sailor joins a yacht club, he may not yet have a rating assigned
 - special value that denotes *unknown*
- suppose that Sailor table definition was modified to also include a *maiden-name* column, only married women who take their husband's last name have a maiden name, for single women and for men, *maiden-name* column is *inapplicable*

Null Values

- SQL provides a special column value called *null* to use in such situations
- use *null* when column value is either *unknown* or *inapplicable*

IS NULL

- special comparison operator to test whether a column value is *null*
- can also say IS NOT NULL

Examples

- *Find all reservations date,*
- *sailor's names and boat's names*

Null Values and Three Valued Logic

- Any comparison with *null* returns *unknown*
 - E.g. $5 < \text{null}$ or $\text{null} < > \text{null}$ or $\text{null} = \text{null}$
- 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.

Examples

- `SELECT Reserves.day, Sailors.sname,
Boats.bname`
- `FROM Sailors S, Boats B, Reserves R`
- `WHERE S.sid = R.sid AND R.bid=B.bid`

Examples

- SELECT Reserves.day, Sailors.sname, Boats.bname FROM
- Sailors S JOIN Reserves R
 - ON S.sid = R.sid
- JOIN Boats B
 - ON R.bid = B.bid

JOIN on two tables

- JOIN on Sailors and Reserves
- default to a inner join
 - only Sailor rows with matching Reserves row appear in result
- there are Sailors without any reservations ...

Outer JOIN

- left outer join
 - Sailor rows without a matching Reserves row appear in result
 - result columns inherited from Reserves assigned *null* values
- right outer join
 - Reserves rows without a matching Sailors row appear in the result
 - result columns inherited from Sailor assigned *null* values

Outer JOIN

- full outer join
 - both Sailors and Reserves rows without a match appear in result
- rows with match always appear in result for all these variants

Self-Join

- for example Sailors can have (Padawan) apprentice, and apprentice can have (Jedi) master
- Sailors
 - (sid: integer, sname: string, rating: integer, age: real, mid: integer)
 - master id (id of the master)

Examples

- *Find all information, about all sailors, show also master name*
- `SELECT S.* , M.sname AS master FROM Sailors S JOIN Sailors M ON S.mid = M.sid`
- `SELECT S.* , M.sname AS master FROM Sailors S, Sailors M WHERE S.mid = M.sid`
- have to use aliases to name two copies of the same tables sailors in order to JOIN

Cheat

TEMPORARY TABLES

Improving Efficiency of Multi-table Queries

- each RDBMS has its own algorithms to translate SQL query into set of binary instructions that could be further interpreted by engine to create execution plan for query
- plan indicates order in which tables are joined, WHERE clause conditions that are applied, whether indexes are used, ...
- for example, assume a goal to join tables CUSTOMER and ORDER_HEADER and then get all orders for customer FAIR PARK GARDENS only
- obvious operation will take less time if we limit row set from CUSTOMER table first (WHERE customer.cust_name_s = 'FAIR PARK GARDENS') and then perform join of resulting set with ORDER_HEADER

Query Optimizer

- all modern RDBMS have special mechanisms called optimizers that create execution plans for queries based on certain information accumulated in information schema
- in theory, makes complex query writing rules (that are different for all vendors and are beyond scope of this) obsolete
- SQL developers no longer have to specify tables and columns in certain order in FROM and WHERE clauses of the SELECT statement to achieve acceptable query performance - optimizer will do the job
- sometimes, usually for complex queries that join large number of tables, optimizers simply cannot find "optimal" way to parse query
- in situations like that, you can give query hint, using predefined vendor-specific set of keywords that overrides optimizer's algorithm and perform actions in the order specified by programmer
- hints can tell RDBMS which tables to join first, whether or not to use index or to perform full table scan, ...

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 IN 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

Thank you for your kindly attention!

Advanced Structured Query Language

HardCore SQL

select * from emp;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-1980	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-1981	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-1981	1250	500	30
7566	JONES	MANAGER	7839	02-APR-1981	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-1981	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-1981	2850		30
7782	CLARK	MANAGER	7839	09-JUN-1981	2450		10
7788	SCOTT	ANALYST	7566	09-DEC-1982	3000		20
7839	KING	PRESIDENT		17-NOV-1981	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-1981	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-1983	1100		20
7900	JAMES	CLERK	7698	03-DEC-1981	950		30
7902	FORD	ANALYST	7566	03-DEC-1981	3000		20
7934	MILLER	CLERK	7782	23-JAN-1982	1300		10

```
select * from dept;
```

- DEPTNO DNAME LOC
- 10 ACCOUNTING NEW YORK
- 20 RESEARCH DALLAS
- 30 SALES CHICAGO
- 40 OPERATIONS BOSTON

```
select * from emp_bonus
```

- EMPNO RECEIVED TYPE
- 7934 17-MAR-2005 1
- 7934 15-FEB-2005 2
- 7839 15-FEB-2005 3
- 7782 15-FEB-2005 1

select id from t10;

- ID
- 1
- 2
- 3
- ...
- 10

Limiting the Number of Rows Returned

- MySQL and PostgreSQL
- `select * from emp limit 5`
- SQL Server
- `select top 5 * from emp`

Returning n Random Records from Table

- take any built-in function supported by your DBMS for returning random values
- use function in ORDER BY clause to sort rows randomly
- use previous recipe's technique to limit number of randomly sorted rows to return
- select top 5 ename, job
 - from emp order by rand()
- select top 5 ename, job
 - from emp order by newid()

Transforming Nulls into Real Values

- select coalesce(comm,0) 2 from emp
- select case
 - when comm is null then 0
 - else comm end
 - from emp

Dealing with Nulls when Sorting

- /* NON-NULL COMM SORTED ASCENDING, ALL NULLS LAST */
- select ename, sal, comm from (
- select ename, sal, comm,
- case when comm is null then 0 else 1 end as is_null
- from emp)
- order by is_null desc, comm

Sorting on Data Dependent Key

- if JOB is "SALESMAN" sort on COMM;
- otherwise, sort by SAL
- select ename, sal, job, comm
- from emp
- order by case
- when job = 'SALESMAN' then comm
- else sal end

Using NULLs in Operations & Comparisons

- NULL is never equal to or not equal to any value, not even itself
- if you want to evaluate values returned by nullable column like you would evaluate real values
- find all employees in EMP whose commission (COMM) is less than the commission of employee "WARD"; employees with NULL commission should be included as well

Using NULLs in Operations & Comparisons

- use function such as COALESCE to transform NULL value into real value that can be used in standard evaluation
- select ename, comm from emp
- where coalesce(comm,0) < (select comm from emp where ename = 'WARD')

Window function

OVER(PARTITION BY ..)

- select e.empno, e.ename, e.sal, e.deptno,
e.sal*case
 - when eb.type = 1 then .1
 - when eb.type = 2 then .2
 - else .3 end as bonus
- from emp e, emp_bonus eb where e.empno =
eb.empno and e.deptno = 10

- EMPNO ENAME SAL DEPTNO BONUS
- -----
- 7934 MILLER 1300 10 130
- 7934 MILLER 1300 10 260
- 7839 KING 5000 10 1500
- 7782 CLARK 2450 10 245

- select deptno, sum(sal) as total_sal, sum(bonus) as total_bonus
- from (...) tempx
- group by deptno
- DEPTNO TOTAL_SAL TOTAL_BONUS
- -----
- 10 10050 2135
- TOTAL_BONUS is correct, TOTAL_SAL is incorrect: sum of all salaries in department 10 is 8750
- reason is duplicate rows in SAL column created by JOIN

Perform SUM of only DISTINCT salaries

- select deptno,
- sum(distinct sal) as total_sal,
- sum(bonus) as total_bonus
- from (...) tempx
- group by deptno

using window function SUM OVER

- select distinct deptno, total_sal, total_bonus from (
- select e.empno, e.ename,
- sum(distinct e.sal) over (partition by e.deptno) as total_sal,
- e.deptno,
- sum(e.sal*case
 - when eb.type = 1 then .1
 - when eb.type = 2 then .2
 - else .3 end) over (partition by deptno) as total_bonus
- from emp e, emp_bonus eb where e.empno = eb.empno and e.deptno = 10
-) tempx

Window function

- windowing function, SUM OVER, is called twice
- first to compute sum of distinct salaries for defined partition or group
- in this case, partition is DEPTNO 10 and sum of distinct salaries for DEPTNO 10 is 8750
- next computes sum of bonuses for same defined partition

Window function

- belong to type of function known as ‘set function’, means function that applies to set of rows
- word ‘window’ is used to refer to set of rows that function works on
- function whose result for given row is derived from window frame of that row PARTITION BY clause is used to define window frame for row
- unlike GROUP BY clause window functions have no grouping effect

**SELECT * FROM
TestAggregation**

The screenshot shows a results grid with two tabs: 'Results' and 'Messages'. The 'Results' tab contains a table with columns 'ID' and 'Value'. The data consists of nine rows, each with an ID value (1, 2, or 3) and a corresponding Value (50.30, 123.30, 132.90). The first row (ID 1, Value 50.30) has a dashed border around its entire cell.

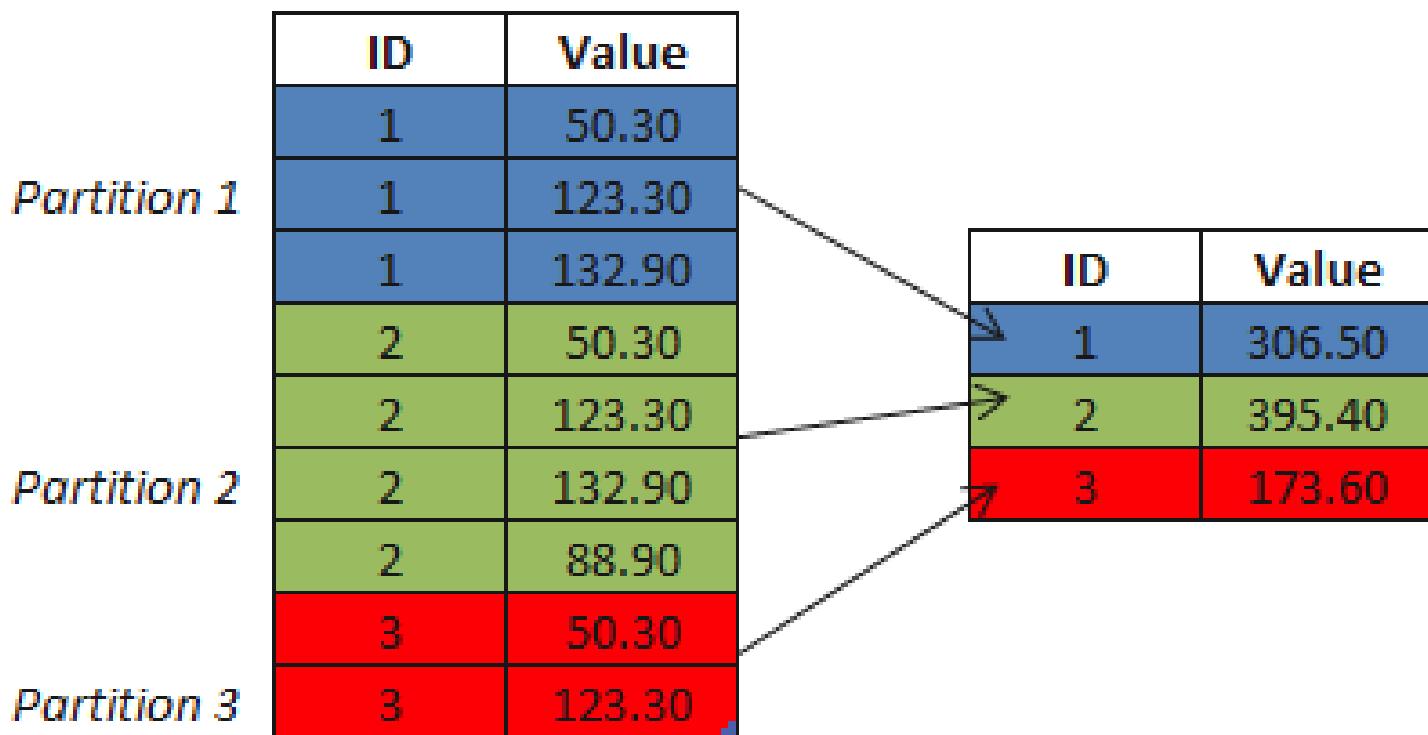
	ID	Value
1	1	50.30
2	1	123.30
3	1	132.90
4	2	50.30
5	2	123.30
6	2	132.90
7	2	88.90
8	3	50.30
9	3	123.30

**SELECT ID, SUM(Value)
FROM TestAggregation
GROUP BY ID;**

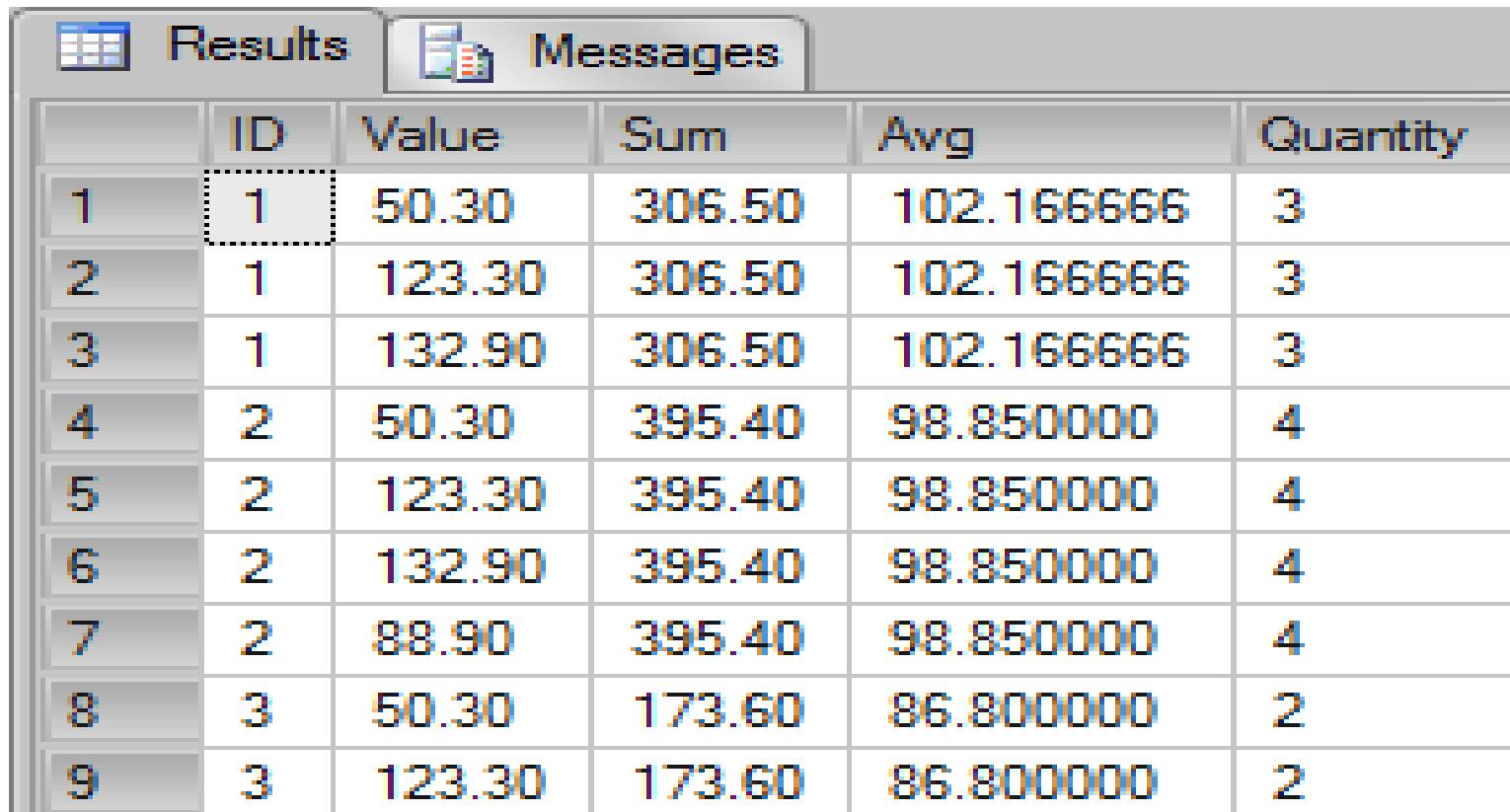
The screenshot shows a results grid with two tabs: 'Results' and 'Messages'. The 'Results' tab contains a table with columns 'ID' and '(No column name)'. The data consists of three rows, each with an ID value (1, 2, or 3) and a corresponding sum value (306.50, 395.40, 173.60). The second row (ID 2, sum 395.40) has a dashed border around its entire cell.

	ID	(No column name)
1	1	306.50
2	2	395.40
3	3	173.60

Partitions



```
SELECT ID, Value, SUM(Value) AS "Sum",
AVG(Value) AS "Avg", COUNT(Value) AS
"Quantity" FROM TestAggregation
GROUP BY ID;
```



	ID	Value	Sum	Avg	Quantity
1	1	50.30	306.50	102.166666	3
2	1	123.30	306.50	102.166666	3
3	1	132.90	306.50	102.166666	3
4	2	50.30	395.40	98.850000	4
5	2	123.30	395.40	98.850000	4
6	2	132.90	395.40	98.850000	4
7	2	88.90	395.40	98.850000	4
8	3	50.30	173.60	86.800000	2
9	3	123.30	173.60	86.800000	2

- `SELECT TestAggregation.ID, TestAggregation.Value, TabSum."Sum", TabAvg."Avg", TabCount."Quantity" FROM TestAggregation`
- `INNER JOIN (SELECT ID, SUM(Value) AS "Sum" FROM TestAggregation`
 - `GROUP BY ID) AS TabSum ON TabSum.ID = TestAggregation.ID`
- `INNER JOIN (SELECT ID, AVG(Value) AS "Avg" FROM TestAggregation`
 - `GROUP BY ID) AS TabAvg ON TabAvg.ID = TestAggregation.ID`
- `INNER JOIN (SELECT ID, COUNT(Value) AS "Quantity" FROM TestAggregation`
 - `GROUP BY ID) AS TabCount ON TabCount.ID = TestAggregation.ID`

- SELECT TestAggregation.ID,
TestAggregation.Value, AggregatedValues.[Sum],
AggregatedValues.[Avg],
AggregatedValues.Quantity
- FROM TestAggregation INNER JOIN (
• SELECT ID, SUM(Value) AS "Sum", AVG(Value) AS
"Avg", COUNT(Value) AS "Quantity" FROM
TestAggregation GROUP BY ID) AggregatedValues
ON AggregatedValues.ID=TestAggregation.ID

- SELECT ID, Value,
- SUM(Value) OVER() AS "Sum"
- AVG(Value) OVER() AS "Avg"
- COUNT(Value) OVER() AS "Quantity,,
- FROM TestAggregation

Results Messages

	ID	Value	Sum	Avg	Quantity
1	1	50.30	875.50	97.277777	9
2	1	123.30	875.50	97.277777	9
3	1	132.90	875.50	97.277777	9
4	2	50.30	875.50	97.277777	9
5	2	123.30	875.50	97.277777	9
6	2	132.90	875.50	97.277777	9
7	2	88.90	875.50	97.277777	9
8	3	50.30	875.50	97.277777	9
9	3	123.30	875.50	97.277777	9

- SELECT ID, Value,
- SUM(Value) OVER(PARTITION BY ID) AS "Sum"
- AVG(Value) OVER(PARTITION BY ID) AS "Avg"
- COUNT(Value) OVER(PARTITION BY ID) AS "Quantity"
- FROM TestAggregation

Results Messages

	ID	Value	Sum	Avg	Quantity
1	1	50.30	306.50	102.166666	3
2	1	123.30	306.50	102.166666	3
3	1	132.90	306.50	102.166666	3
4	2	50.30	395.40	98.850000	4
5	2	123.30	395.40	98.850000	4
6	2	132.90	395.40	98.850000	4
7	2	88.90	395.40	98.850000	4
8	3	50.30	173.60	86.800000	2
9	3	123.30	173.60	86.800000	2

... OVER(PARTITION BY ID) ...

ID	Value	Sum	Avg	Quantity		ID	Value	Sum	Avg	Quantity
1	50.30	306.50	102.166.666	3		1	50.30	306.50	102.166.666	3
1	123.30	306.50	102.166.666	3		1	123.30	306.50	102.166.666	3
1	132.90	306.50	102.166.666	3		1	132.90	306.50	102.166.666	3
2	50.30	395.40	98.850.000	4		2	50.30	395.40	98.850.000	4
2	123.30	395.40	98.850.000	4		2	123.30	395.40	98.850.000	4
2	132.90	395.40	98.850.000	4		2	132.90	395.40	98.850.000	4
2	88.90	395.40	98.850.000	4		2	88.90	395.40	98.850.000	4
3	50.30	173.60	86.800.000	2		3	50.30	173.60	86.800.000	2
3	123.30	173.60	86.800.000	2		3	123.30	173.60	86.800.000	2

```
SELECT id, SUM(value) FROM  
TestAggregation GROUP BY id
```

- id (No column name)
- 1 306.5
- 2 395.4
- 3 173.6

```
SELECT id, SUM(value) OVER (PARTITION  
    BY ID) FROM TestAggregation
```

- id (No column name)
- 1 306.5
- 1 306.5
- 1 306.5
- 2 395.4
- 2 395.4
- 2 395.4
- 2 395.4
- 3 173.6
- 3 173.6

```
SELECT DISTINCT id, SUM(value)
OVER(PARTITION BY ID) FROM TestAggregation
```

- id (No column name)
- 1 306.5
- 2 395.4
- 3 173.6

Windows Functions

- Aggregate functions
- Ranking functions
- Analytic functions
- NEXT VALUE FOR function

Windows Functions

Aggregate Functions

- AVG
- SUM
- COUNT
 - COUNT_BIG (returns BigInt)
- MIN
- MAX

Windows Functions

Aggregate Functions

- **CHECKSUM_AGG** - checksum of values in group
- **STDEV, STDEVP** - statistical standard deviation of all values in group
- **VAR, VARP** - statistical variance of all values in group

Windows Functions

Aggregate Functions

- GROUPING - indicates whether specified column expression in GROUP BY list is aggregated or not
- returns 1 for aggregated or 0 for not aggregated
- can be used only in SELECT <select> list, HAVING, and ORDER BY clauses when GROUP BY is specified
- GROUPING_ID - computes level of grouping

- SELECT deptno, ename, sal
- ,ROW_NUMBER() OVER (ORDER BY deptno, ename) AS "Row Number"
- ,RANK() OVER (ORDER BY sal) AS Rank
- ,DENSE_RANK() OVER (ORDER BY sal) AS "Dense Rank"
- ,NTILE(4) OVER (ORDER BY deptno, ename) AS Quartile
- FROM EMP order by deptno, ename

FROM EMP order by deptno, ename

	deptno	ename	sal	Row Number	Rank	Dense Rank	Quartile
•	10	CLARK	2450.00	1	9	8	1
•	10	KING	5000.00	2	14	12	1
•	10	MILLER	1300.00	3	6	5	1
•	20	ADAMS	1100.00	4	3	3	1
•	20	FORD	3000.00	5	12	11	2
•	20	JONES	2975.00	6	11	10	2
•	20	SCOTT	3000.00	7	12	11	2
•	20	SMITH	800.00	8	1	1	2
•	30	ALLEN	1600.00	9	8	7	3
•	30	BLAKE	2850.00	10	10	9	3
•	30	JAMES	950.00	11	2	2	3
•	30	MARTIN	1250.00	12	4	4	4
•	30	TURNER	1500.00	13	7	6	4
•	30	WARD	1250.00	14	4	4	4

FROM EMP order by sal

	deptno	ename	sal	Row Number	Rank	Dense Rank	Quartile
•	20	SMITH	800.00	8	1	1	2
•	30	JAMES	950.00	11	2	2	3
•	20	ADAMS	1100.00	4	3	3	1
•	30	MARTIN	1250.00	12	4	4	4
•	30	WARD	1250.00	14	5	4	4
•	10	MILLER	1300.00	3	6	5	1
•	30	TURNER	1500.00	13	7	6	4
•	30	ALLEN	1600.00	9	8	7	3
•	10	CLARK	2450.00	1	9	8	1
•	30	BLAKE	2850.00	10	10	9	3
•	20	JONES	2975.00	6	11	10	2
•	20	SCOTT	3000.00	7	12	11	2
•	20	FORD	3000.00	5	12	11	2
•	10	KING	5000.00	2	14	12	1

Row_Number()

- generate sequence of numbers based in set in specific order
- returns sequence number of each row inside set in order that you specify

NTILE()

- used for calculating summary statistics; distributes rows within an ordered partition into specified number of “buckets” or groups
- groups are numbered, starting at one
- for each row, NTILE returns number of group to which row belongs

Rank() & Dense_Rank()

- return position in ranking for each row inside partition; calculated by 1 plus number of previous rows
- RANK returns result with gap after tie, whereas DENSE_RANK doesn't

Insert, Update, Delete

INSERT

- insert one row at a time
- insert into dept (deptno,dname,loc)
 - values (50,'PROGRAMMING','BALTIMORE')
- insert multiple rows at a time
- insert into dept (deptno,dname,loc)
 - values (1,'A','B'), (2,'B','C')
- insert row of default values without having to specify those values

Copy rows from one table into another

- follow INSERT statement with query that returns desired rows
- insert into dept_east (deptno,dname,loc)
- select deptno,dname,loc
- from dept
- where loc in ('NEW YORK','BOSTON')

Block Insert to certain columns

- create view with only those columns you wish to expose.
- then force all inserts to go through that view
- create view new_emps as
- select empno, ename, job from emp
- Grant access to view to those users and programs allowed to populate only the three fields in view
- do not Grant those users insert access to EMP table

- `insert into new_emps (empno ename, job)`
- `values (1, 'Jonathan', 'Editor')`
- will be translated behind the scenes into:
- `insert into emp (empno ename, job)`
- `values (1, 'Jonathan', 'Editor')`

UPDATE

- bump all SAL values by 10%.
- update emp
- set sal = sal*1.10
- where deptno = 20

Update when corresponding rows exist

- update emp
- set sal=sal*1.20
- where empno in (select empno from emp_bonus)
- alternatively, can use EXISTS instead of IN
- update emp
- set sal = sal*1.20
- where exists (select null
 - from emp_bonus
 - where emp.empno = emp_bonus.empno)

Update with values from another table

- update salaries and commission of employees in table EMP using values table NEW_SAL if there is match between EMP.DEPTNO and NEW_SAL.DEPTNO, update EMP.SAL to NEW_SAL.SAL, and EMP.COMM to 50% of NEW_SAL.SAL
- MS SQL Server you can join directly in UPDATE
- update e
 - set e.sal = ns.sal, e.comm = ns.sal/2
- from emp e, new_sal ns where ns.deptno = e.deptno

Update with values from another table

- update emp e
- set (e.sal,e.comm) = (select ns.sal, ns.sal/2
 - from new_sal ns where ns.deptno=e.deptno)
- where exists (select null from new_sal ns where ns.deptno = e.deptno)

Delete records from table

- Delete all records from table
- delete from emp
- Delete specific records from table
- delete from emp
- where deptno = 10

Metadata Queries

```
OBJECT_ID('[ database_name . [ schema_name ]  
.] object_name' [ , 'object_type' ]);
```

- getting the Identifier of an Object

```
DB_ID ( [ 'database_name' ] ) RETURNS  
int;
```

- Identifier of a Database
- function takes an argument that is optional - If you call without an argument, it returns name of current database

```
DB_NAME ( [ database_id ] ) RETURNS  
        nvarchar(128);
```

- Getting the Current Database
- function takes an optional argument - If you call this function without an argument, it finds name of database that is currently selected

```
USER_NAME( [ server_user_id ] ) RETURNS  
nvarchar(128);
```

- get the actual username of the user connected

```
HOST_NAME() RETURNS  
    nvarchar(128);
```

- get the name of computer that is currently being used

List Tables in Schema

- select table_name
- from information_schema.tables
- where table_schema = 'DBDCourse'

List Table's Columns (Indexed)

- select column_name, data_type, ordinal_position
- from information_schema.columns
- where table_schema = 'DBDCourse'
- and table_name = 'EMP'

- select a.tablename, a.indexname, b.column_name
- from pg_catalog.pg_indexes a,
information_schema.columns b
- where table_schema = 'DBDCourse'
- and a.tablename = b.table_name

List Constraints on Table

- select a.table_name, a.constraint_name,
b.column_name, a.constraint_type from
information_schema.table_constraints a,
information_schema.key_column_usage b
- where a.table_name = 'EMP' and a.table_schema
= 'DBDCourse'
- and a.table_name = b.table_name
- and a.table_schema = b.table_schema
- and a.constraint_name = b.constraint_name

Working with Strings

```
PARSE ( string_value AS data_type [  
    USING culture ] )
```

- Parsing consists of scanning of expression or word to match pattern
- to check every symbol in combination to find out if it is number or something else
- takes one argument, passed as string and accompanied by its data type preceded by AS keyword
- used to "scan" an expression that is passed as *string_value* argument; expression must follow rules of *data_type* argument

PARSE (string_value AS data_type [USING culture])

- for example to find out if some value is an integer; if you want to know whether some value is date,(value must follow rules of date value as specified in Date tab in Customize Regional Options accessible from Regional and Language Options)
- If argument may include international characters or formats (Unicode), you should indicate language, called *culture*, that argument follows
- If PARSE() is not able to determine type or if value of argument doesn't follow rule of *data_type*, function produces (throws) an error

`TRY_PARSE (string_value AS
data_type [USING culture])`

- as an alternative provides TRY_PARSE() function
- If parsing operation fails, TRY_PARSE produces NULL (if parsing operation fails)
- in most cases, you should prefer TRY_PARSE() instead of PARSE()

CAST (Expression AS DataType [(length)])

- Casting a Value: in most cases, value user submits is primarily considered string; If value user provides must be treated as something other than string, before using such a value, you should first convert it to appropriate type
- If CAST() function is not able to cast expression, it produces (throws) an error; as an alternative provides a function named
- TRY_CAST (expression AS data_type [(length)])

`CONVERT(DataType [(length)] , Expression [, style])`

- can be used to convert value from its original type into non-similar type
- first argument must be known data type
- Expression is value that needs to be converted
- If conversion is performed on date or time value, the style argument is number that indicates how that conversion must proceed
- `TRY_CONVERT (data_type [(length)], expression [, style])`

`int LEN(String)`

- Length of a String
- number of characters or symbols it contains

```
CONCAT(string string_value1,  
       string str_val2 [, string str_val_N])  
RETURNS string;
```

- Concatenating Strings
- function takes an unlimited number of strings as arguments; function returns string
- in reality, each of arguments can be char or one of its variants: nchar, char(n), varchar, nvarchar, or nvarchar(n), or nvarchar(max)

int ASCII(String)

- Converting From Integer to ASCII
- function takes as argument string and returns ASCII code of first left character of string

CHAR(int value) RETURNS char;

- Converting From ASCII to Integer
- function takes as argument numeric value as an integer; upon conversion, function returns ASCII equivalent of that number

```
LOWER(String) RETURNS varchar; /  
UPPER(String) RETURNS varchar;
```

- Converting to Lowercase / Uppercase
- function takes as argument a string which would be converted to lowercase / uppercase; after conversion function returns new string

`LEFT(String, NumberOfCharacters)` RETURNS varchar /
`RIGHT(String, NumberOfCharacters)` RETURNS
varchar

- Sub-Strings Characters of a String
- function takes two arguments: first argument specifies original string, second argument specifies number of characters from most left / right that will constitute sub-string - after operation, function returns new string

```
REPLACE(String, FindString, ReplaceWith)
        RETURNS varchar;
```

- Replacing Occurrences in a String
- function takes three arguments: first is string that will be used as reference; second argument is character or sub-string to look for
- If *FindString* sub-string is found in *String*, then it is replaced with value of last argument, *ReplaceWith*

Embed Quotes Within String Literals

- select ‘Q’’s powers’ from t1

Remove Unwanted Characters from String

- for example remove vowels and all zeros - use functions TRANSLATE and REPLACE
- select ename, replace (replace (replace (replace (replace (ename,'A',''),'E',''),'I',''),'O',''),'U','') as stripped1, sal, replace(sal,0,"") stripped2 from emp

Extracting Initials from Name

- George Strawberry – G.S.
- select replace(replace(
- translate(replace(' George Strawberry ','.', ','),
- 'abcdefghijklmnopqrstuvwxyz',
- rpad('#',26,'#')), '#',''),',','.') ||'.'
- from t1

Extracting n^{th} Delimited Substring

Parsing an IP Address

- select
- split_part(y.ip,'.',1) as a, split_part(y.ip,'.',2) as b,
- split_part(y.ip,'.',3) as c, split_part(y.ip,'.',4) as d
- from (select cast('193.226.17.55' as text) as ip
from t1) as y

Working with Numbers

Mathematical Functions and Operators

- + addition - subtraction
- * multiplication / division (integer division truncates results)
- % modulo (remainder)
- ^ exponentiation
- |/ square root ||/ cube root
- ! factorial !! factorial (prefix operator)
- @ absolute value
- & bitwise AND | bitwise OR
- # bitwise XOR ~ bitwise NOT
- << bitwise shift left >> bitwise shift right

Arithmetic Functions

- **SIGN(Expression)**
- **ABS(Expression)** RETURNS Data Type of Argument
- **CEILING(Expression)** **FLOOR(Expression)**
- **EXP(Expression)** **LOG(Expression)**
- **POWER(x, y)**
- **SQRT(Expression)**

Arithmetic Functions

- PI()
- RADIANS(Expression)
- DEGREES(Expression)
- COS(Expression) SIN(Expression)
- TAN(Expression)

Trigonometric Functions

- $\text{acos}(x)$ inverse cosine $\text{asin}(x)$ inverse sine
- $\text{atan}(x)$ inverse tangent $\text{atan2}(x, y)$
inverse tangent of x/y
- $\text{cos}(x)$ cosine $\text{cot}(x)$
cotangent
- $\text{sin}(x)$ sine $\text{tan}(x)$ tangent

Mathematical Functions

- $\text{abs}(x)$ - (same type as x): absolute value
- $\text{cbrt}(dp)$ – dp : cube root
- $\text{ceil}(dp \text{ or numeric})$ - (same as input): smallest integer not less than argument
- $\text{floor}(dp \text{ or numeric})$ - (same as input): largest integer not greater than argument
- $\text{degrees}(dp)$ – dp : radians to degrees
- $\text{radians}(dp)$ – dp : degrees to radians
- $\text{exp}(dp \text{ or numeric})$ - (same as input): exponential

Mathematical Functions

- `ln(dp or numeric)` - (same as input): natural logarithm
- `log(dp or numeric)` - (same as input): base 10 logarithm
- `log(b numeric, x numeric)` - numeric: logarithm to base b
- `mod(y, x)` - (same as argument types): remainder of y/x
- `pi()` – dp: "π" constant $\pi() = 3.14159265358979$
- `pow(a dp, b dp)` – dp: a raised to power of b
- `pow(a numeric, b numeric)` – numeric: a raised to power of b
- `random()` -dp: random value between 0.0 and 1.0

Mathematical Functions

- `round(dp or numeric)` - (same as input): round to nearest integer
- `round(v numeric, s integer)` – numeric: round to s decimal places
- `setseed(dp)` - int32: set seed for subsequent `random()` calls
- `sign(dp or numeric)` - (same as input): sign of argument
- `sqrt(dp or numeric)` - (same as input): square root
- `trunc(dp or numeric)` - (same as input): truncate toward zero
- `trunc(v numeric, s integer)` – numeric: truncate to s decimal places

Generate Running Total

- select e.ename, e.sal,
- (select sum(d.sal) from emp d
 - where d.empno <= e.empno) as running_total
- from emp e

ENAME SAL RUNNING_TOTAL

- SMITH 800 800
- ALLEN 1600 2400
- WARD 1250 3650
- JONES 2975 6625
- MARTIN 1250 7875
- BLAKE 2850 10725
- CLARK 2450 13175
- SCOTT 3000 16175
- KING 5000 21175
- TURNER 1500 22675
- ADAMS 1100 23775
- JAMES 950 24725
- FORD 3000 27725
- MILLER 1300 29025

Calculate Mode

element that appears most frequently

- select sal from emp
- group by sal
- having count(*) >= all (select count(*)
 - from emp
 - group by sal)

Compute Averages Without High and Low Values

- select avg(sal) from emp where sal not in (
 - (select min(sal) from emp),
 - (select max(sal) from emp))

Date Arithmetic

DATE

- data type counts dates starting from January 1st 0001 up to December 31st 9999
- various rules you must follow to represent a date - can be checked in Date tab of Customize Regional Options accessible from Regional and Language Options of Control Panel

DATE

- initialize DATE variable, use one of following formulas
- YYYYMMDD YYYY-MM-DD
- MM-DD-YY MM-DD-YYYY
- MM/DD/YY MM/DD/YYYY
- DD-MMM-YY DD-MMMM-YY
- DD-MMM-YYYY DD-MMMM-YYYY

DATE

- `DECLARE @OneDay DATE;`
- `SET @OneDay = N'19640119';`
- `SELECT @OneDay AS [Birth Day];`

DATE DATEFROMPARTS(int year, int month, int day)

- Creating a Date
- function allows to create date if you have year, month, and day

GETDATE(); SYSDATETIME();

- Current System Date
- get current date (and time) of the computer
- get date with more precision, call
SYSDATETIME function

FORMATTING & CONTROLLING DISPLAY OF DATES

```
FORMAT(value, nvarchar format [, culture ]  
) RETURNS nvarchar
```

- Displaying the Numeric Day
- days of months are numbered from 1 to 31, depending on month
- display day of month is to use d (lowercase) in combination that includes month and year
- d produces day in 1 digit if number is between 1 and 9 (no leading 0) or in 2 digits
- dd produces day in 2 digits if number is between 1 and 9, it displays with leading 0

Displaying the Numeric Day example

- `DECLARE @DateValue DATE, @StrValue nvarchar(50);`
- `SET @DateValue = N'20120603';`
- `SET @StrValue = FORMAT(@DateValue, N'dd');`

Displaying Weekday Names

- names of week use two formats: 3 letters or full name
- to display name with 3 letters, use "ddd" in format argument (names will be Sun, Mon, Tue, Wed, Thu, Fri, or Sat)
- to display complete name of weekday, pass "dddd"?
- `SET @DateValue = N'20150406'; SET @StrValue = FORMAT(@DateValue, N'ddd'); --- / "dddd"?`
- Mon / Monday

Displaying Numeric Months

- to display number of month, pass format argument as
- MM (uppercase) - month is provided as an integer with 2 digits; If number is between 1 and 9, it displays with leading 0
- M (uppercase) - would produce number of month without leading 0
- display month by its name using short / long name - pass format as MMM (uppercase) / MMM (uppercase)

Displaying Numeric Months

- `SET @DateValue = N'20150406';`
- `SET @StrValue = FORMAT(@DateValue, N'ddd, dd-MMM');`
- `Mon, 06-Apr`
- `SET @StrValue = FORMAT(@DateValue, N'dddd, dd-MMMM');`
- `Monday, 06-April`

Displaying the Year of a Date

- If you want to display year in two digits, pass yy (lowercases) as format
- year value represented with two digits is hardly explicit, unless you have good reason for using it - alternative is to use all four digits to display year - format is created with the yyy (lowercase) or yyyy (lowercase)

OPERATIONS ON DATES

DATEADD(TypeOfValue, ValueToAdd, DateOrTimeReferenced)

- first argument specifies type of value that will be added
- year, yy, yyyy - number of years will be added to date value
- quarter, q, qq - number of quarters of a year will be added to date value
- month, m, mm - number of months will be added to date value
- dayofyear, y, dy - number of days of year will be added to date value

DATEADD(TypeOfValue, ValueToAdd, DateOrTimeReferenced)

- day, d, dd - number of days will be added to date value
- week, wk, ww - number of weeks will be added to date value
- SET @Original = N'20150406';
- SET @Result = DATEADD(wk, 2, @Original);

DATEDIFF(TypeOfValue, StartDate, EndDate)

- Finding the Difference of Two Date Values
- first argument specifies type of value function must produce (uses same value as those of DATEADD() function)
- second argument is starting date
- third argument is end date

- `DECLARE @DateHired As date, @CurrentDate As date;`
- `SET @DateHired = N'2010/10/01';`
- `SET @CurrentDate = GETDATE();`
- `SELECT DATEDIFF(dd, @DateHired, @CurrentDate) AS [Student Days];`

DATEPART(int DatePart, date Value)

- is part of date (a date or time value) for which an integer will be returned
- year - yy, yyyy
- quarter - qq, q
- month - mm, m
- dayofyear - dy, y
- day - dd , d
- week - wk , ww
- weekday - dw

DATEPART(int DatePart, date Value)

- hour - hh
- minute - mi, n
- second - ss , s
- millisecond - ms
- microsecond - mcs
- nanosecond - ns

```
int DAY(date Value);
```

- DATEPART(d, @DateValue)

`int MONTH(date Value);`

- `DATEPART(m, @DateValue)`

```
int YEAR(date Value);
```

- DATEPART(y, @DateValue)

- DATE – to the left of decimal point
- TIME – to the right of decimal point

`TIMEFROMPARTS(int hour, int minute, int seconds, int fractions, int precision) RETURNS time;`

- Creating a Time From Parts

DATEADD(TypeOfValue, ValueToAdd,
DateOrTimeReferenced) DATEDIFF(TypeOfValue,
StartDate, EndDate)

- hour - hh
- minute – n, mi
- second – s, ss
- millisecond – ms
- microsecond – mcs
- nanosecond - ns

recursive WITH RECURSIVE QUERIES USING COMMON TABLE EXPRESSIONS

Create Delimited List from Table Rows

- 10 CLARK
- 10 KING
- 10 MILLER
- 20 SMITH
- 20 ADAMS
- 20 FORD
- 20 SCOTT
- 20 JONES
- 30 ALLEN
- 30 BLAKE
- 30 MARTIN
- 30 JAMES
- 30 TURNER
- 30 WARD
- 10 CLARK, KING, MILLER
- 20 SMITH, JONES, SCOTT, ADAMS, FORD
- 30 ALLEN, WARD, MARTIN, BLAKE, TURNER, JAMES

- with x (deptno, cnt, list, empno, len)
- as (
- select deptno, count(*) over(partition by deptno),
- cast(ename as varchar(100)),
- empno,
- 1
- from emp
- union all
- select x.deptno, x.cnt,
- cast(x.list + ', ' + e.ename as varchar(100)),
- e.empno, x.len+1
- from emp e, x
- where e.deptno = x.deptno
- and e.empno > x.empno
-)
- select deptno, list
- from x
- where len = cnt
- order by 1

- with x (deptno, cnt, list, empno, len)
- as (select deptno, count(*) over (partition by deptno), cast(ename as varchar(100)), empno, 1
- from emp
- union all select x.deptno, x.cnt, cast(x.list + ',' + e.ename as varchar(100)), e.empno, x.len+1
- from emp e, x where e.deptno = x.deptno and e.empno > x.empno)
- select deptno, list from x where len = cnt
- order by deptno

- first query in WITH clause (upper portion of UNION ALL) returns following information about each employee: department, number of employees in that department, name, ID and constant 1 (which at this point doesn't do anything)
- recursion takes place in second query (lower half of the UNION ALL) to build list

- examine third SELECT-list item from second query in union
- $\text{cast}(x.list + ', ' + e.ename as varchar(100)),$
- and WHERE clause from that same query
- $\text{where e.deptno} = x.deptno \text{ and } e.empno > x.empno$

- solution works by first ensuring employees are in same department
- then, for every employee returned by upper portion of UNION ALL, append name of employees who have greater EMPNO
 - ensure that no employee will have his own name appended

- $x.\text{len}+1$
- increments LEN (which starts at 1) every time an employee has been evaluated; when incremented value equals number of employees in department
- $\text{where } \text{len} = \text{cnt}$
- you know you have evaluated all employees and have completed building list
- crucial to query, signals when list is complete, stops recursion from running longer than necessary

Common Table Expression (CTE)

- provides significant advantage of being able to reference itself, thereby creating recursive common table expressions
- initial CTE is repeatedly executed to return subsets of data until complete result set is obtained

Hierarchical Data

- common use of recursive queries; for example
- displaying employees in an organizational chart
- data in Bill of Materials scenario in which parent product has one or more components and those components may, in turn, have subcomponents or may be components of other parents

Structure of Recursive CTE

- similar to recursive routines in other programming languages; although recursive routine in other languages returns scalar value
 - recursive CTE can return multiple rows
1. Invocation of routine
 2. Recursive invocation of routine
 3. Termination check

Invocation of routine

- first invocation of recursive CTE consists of one or more CTE query definitions joined by UNION ALL, UNION, EXCEPT, or INTERSECT operators
- query definitions form base result set of CTE structure, they are *referred to as anchor members*, unless they reference CTE itself
- anchor-member query definitions must be positioned before first recursive member definition, and UNION ALL operator must be used to join last anchor member with first recursive member

Recursive invocation of routine

- includes one or more CTE query definitions joined by UNION ALL operators that reference CTE itself
- these query definitions are *referred to as recursive members*

Termination check

- is implicit - recursion stops when no rows are returned from previous invocation

Structure of Recursive CTE

- WITH cte_name (column_name [,...n])
- AS
- (
- CTE query definition - anchor member is defined
- UNION ALL
- CTE query definition - recursive member is defined referencing cte_name
-) -- Statement using CTE
- SELECT * FROM cte_name

1. split CTE expression into anchor and recursive members
2. run anchor member(s) creating first invocation or base result set (T_0).
3. run recursive member(s) with T_i as input and T_{i+1} as output
4. repeat step 3 until empty set is returned
5. return result set - UNION ALL of T_0 to T_n

Anchor

deptno	cnt	list	empno	len
10 3	CLARK	7782	1	
10 3	KING	7839	1	
10 3	MILLER	7934	1	
20 5	FORD	7902	1	
20 5	ADAMS	7876	1	
20 5	SCOTT	7788	1	
20 5	SMITH	7369	1	
20 5	JONES	7566	1	
30 6	MARTIN	7654	1	
30 6	BLAKE	7698	1	
30 6	ALLEN	7499	1	
30 6	WARD	7521	1	
30 6	TURNER	7844	1	
30 6	JAMES	7900	1	

1. Recursive

- Anchor UNION ALL

deptno	cnt	list	empno	len
10	3	CLARK, KING	7839	2
10	3	CLARK, MILLER	7934	2
10	3	KING, MILLER	7934	2
20	5	ADAMS, FORD	7902	2
20	5	SCOTT, ADAMS	7876	2
20	5	SCOTT, FORD	7902	2
20	5	SMITH, JONES	7566	2
20	5	SMITH, SCOTT	7788	2
20	5	SMITH, ADAMS	7876	2
20	5	SMITH, FORD	7902	2
20	5	JONES, SCOTT	7788	2

1. Recursive

•	20	5	JONES, ADAMS	7876	2
•	20	5	JONES, FORD	7902	2
•	30	6	MARTIN, BLAKE	7698	2
•	30	6	MARTIN, TURNER	7844	2
•	30	6	MARTIN, JAMES	7900	2
•	30	6	BLAKE, TURNER	7844	2
•	30	6	BLAKE, JAMES	7900	2
•	30	6	ALLEN, WARD	7521	2
•	30	6	ALLEN, MARTIN	7654	2
•	30	6	ALLEN, BLAKE	7698	2
•	30	6	ALLEN, TURNER	7844	2
•	30	6	ALLEN, JAMES	7900	2
•	30	6	WARD, MARTIN	7654	2
•	30	6	WARD, BLAKE	7698	2
•	30	6	WARD, TURNER	7844	2
•	30	6	WARD, JAMES	7900	2
•	30	6	TURNER, JAMES	7900	2

2. Recursive

- Anchor UNION ALL 1. Recursive UNION ALL

	deptno	cnt	list	empno	cnt
•	10	3	CLARK, KING, MILLER	7934	3
•	20	5	SCOTT, ADAMS, FORD	7902	3
•	20	5	SMITH, JONES, SCOTT	7788	3
•	20	5	SMITH, JONES, ADAMS	7876	3
•	20	5	SMITH, SCOTT, ADAMS	7876	3
•	20	5	SMITH, JONES, FORD	7902	3
•	20	5	SMITH, SCOTT, FORD	7902	3
•	20	5	SMITH, ADAMS, FORD	7902	3
•	20	5	JONES, SCOTT, ADAMS	7876	3
•	20	5	JONES, SCOTT, FORD	7902	3
•	20	5	JONES, ADAMS, FORD	7902	3
•	30	6	MARTIN, BLAKE, JAMES	7900	3
•	30	6	MARTIN, TURNER, JAMES	7900	3
•	30	6	MARTIN, BLAKE, TURNER	7844	3

2. Recursive

- 30 6 BLAKE, TURNER, JAMES 7900 3
- 30 6 ALLEN, WARD, MARTIN 7654 3
- 30 6 ALLEN, WARD, BLAKE 7698 3
- 30 6 ALLEN, MARTIN, BLAKE 7698 3
- 30 6 ALLEN, MARTIN, JAMES 7900 3
- 30 6 ALLEN, BLAKE, JAMES 7900 3
- 30 6 ALLEN, TURNER, JAMES 7900 3
- 30 6 ALLEN, WARD, TURNER 7844 3
- 30 6 ALLEN, MARTIN, TURNER 7844 3
- 30 6 ALLEN, BLAKE, TURNER 7844 3
- 30 6 ALLEN, WARD, JAMES 7900 3
- 30 6 WARD, MARTIN, BLAKE 7698 3
- 30 6 WARD, MARTIN, JAMES 7900 3
- 30 6 WARD, BLAKE, JAMES 7900 3
- 30 6 WARD, TURNER, JAMES 7900 3
- 30 6 WARD, MARTIN, TURNER 7844 3
- 30 6 WARD, BLAKE, TURNER 7844 3

5. Recursive

		deptno	cnt	list
•	10	3	CLARK	7782
•	10	3	KING	7839
•	10	3	MILLER	7934
•	20	5	SMITH	7369
•	20	5	JONES	7566
•	20	5	SCOTT	7788
•	20	5	ADAMS	7876
•	20	5	FORD	7902
•	30	6	ALLEN	7499
•	30	6	WARD	7521
•	30	6	MARTIN	7654
•	30	6	BLAKE	7698
•	30	6	TURNER	7844
•	30	6	JAMES	7900
•	10	3	CLARK, KING	7839
•	10	3	CLARK, MILLER	7934
•	10	3	KING, MILLER	7934
•	20	5	SMITH, JONES	7566
•	20	5	SMITH, SCOTT	7788
•	20	5	JONES, SCOTT	7788
•	20	5	JONES, ADAMS	7876
•	20	5	SMITH, ADAMS	7876
•	20	5	SCOTT, ADAMS	7876
•	20	5	SCOTT, FORD	7902
•	20	5	ADAMS, FORD	7902
•	20	5	SMITH, FORD	7902
•	20	5	JONES, FORD	7902
•	30	6	ALLEN, WARD	7521
•	30	6	ALLEN, MARTIN	7654
•	30	6	WARD, MARTIN	7654
•	30	6	WARD, BLAKE	7698
•	30	6	ALLEN, BLAKE	7698
•	30	6	MARTIN, BLAKE	7698
•	30	6	MARTIN, TURNER	7844
•	30	6	BLAKE, TURNER	7844
•	30	6	ALLEN, TURNER	7844
•	30	6	WARD, TURNER	7844
•	30	6	WARD, JAMES	7900
•	30	6	TURNER, JAMES	7900
•	30	6	ALLEN, JAMES	7900
•	30	6	BLAKE, JAMES	7900
•	30	6	MARTIN, JAMES	7900
•	10	3	CLARK, KING, MILLER	7934
•	20	5	SMITH, JONES, SCOTT	7788
•	20	5	SMITH, JONES, ADAMS	7876
•	20	5	SMITH, SCOTT, ADAMS	7876
•	20	5	JONES, SCOTT, ADAMS	7876

5. Recursive

- Anchor UNION ALL 1. Recursive UNION ALL 2. Recursive UNION ALL ... 4. Recursive UNION ALL

deptnocnt	list	empno	len
10	CLARK	7782	1
10	KING	7839	1
10	MILLER	7934	1
20	SMITH	7369	1
20	JONES	7566	1
20	SCOTT	7788	1
20	ADAMS	7876	1
20	FORD	7902	1
30	ALLEN	7499	1
30	WARD	7521	1
30	MARTIN	7654	1
30	BLAKE	7698	1
30	TURNER	7844	1
30	JAMES	7900	1

5. Recursive

•	10	3	CLARK, KING	7839	2
•	10	3	CLARK, MILLER	7934	2
•	10	3	KING, MILLER	7934	2
•	20	5	SMITH, JONES	7566	2
•	20	5	SMITH, SCOTT	7788	2
•	20	5	JONES, SCOTT	7788	2
•	20	5	JONES, ADAMS	7876	2
•	20	5	SMITH, ADAMS	7876	2
•	20	5	SCOTT, ADAMS	7876	2
•	20	5	SCOTT, FORD	7902	2
•	20	5	ADAMS, FORD	7902	2
•	20	5	SMITH, FORD	7902	2
•	20	5	JONES, FORD	7902	2
•	30	6	ALLEN, WARD	7521	2
•	30	6	ALLEN, MARTIN	7654	2
•	30	6	WARD, MARTIN	7654	2
•	30	6	WARD, BLAKE	7698	2
•	30	6	ALLEN, BLAKE	7698	2
•	30	6	MARTIN, BLAKE	7698	2
•	30	6	MARTIN, TURNER	7844	2
•	30	6	BLAKE, TURNER	7844	2
•	30	6	ALLEN, TURNER	7844	2
•	30	6	WARD, TURNER	7844	2
•	30	6	WARD, JAMES	7900	2
•	30	6	TURNER, JAMES	7900	2
•	30	6	ALLEN, JAMES	7900	2
•	30	6	BLAKE, JAMES	7900	2
•	30	6	MARTIN, JAMES	7900	2

5. Recursive

•	10	3	CLARK, KING, MILLER	7934	3
•	20	5	SMITH, JONES, SCOTT	7788	3
•	20	5	SMITH, JONES, ADAMS	7876	3
•	20	5	SMITH, SCOTT, ADAMS	7876	3
•	20	5	JONES, SCOTT, ADAMS	7876	3
•	20	5	JONES, SCOTT, FORD	7902	3
•	20	5	JONES, ADAMS, FORD	7902	3
•	20	5	SMITH, SCOTT, FORD	7902	3
•	20	5	SCOTT, ADAMS, FORD	7902	3
•	20	5	SMITH, JONES, FORD	7902	3
•	20	5	SMITH, ADAMS, FORD	7902	3
•	30	6	ALLEN, WARD, MARTIN	7654	3
•	30	6	ALLEN, WARD, BLAKE	7698	3
•	30	6	ALLEN, MARTIN, BLAKE	7698	3
•	30	6	WARD, MARTIN, BLAKE	7698	3
•	30	6	WARD, MARTIN, TURNER	7844	3
•	30	6	WARD, BLAKE, TURNER	7844	3
•	30	6	ALLEN, BLAKE, TURNER	7844	3
•	30	6	ALLEN, MARTIN, TURNER	7844	3
•	30	6	ALLEN, WARD, TURNER	7844	3
•	30	6	MARTIN, BLAKE, TURNER	7844	3
•	30	6	MARTIN, BLAKE, JAMES	7900	3
•	30	6	MARTIN, TURNER, JAMES	7900	3
•	30	6	BLAKE, TURNER, JAMES	7900	3
•	30	6	ALLEN, WARD, JAMES	7900	3
•	30	6	ALLEN, MARTIN, JAMES	7900	3
•	30	6	ALLEN, BLAKE, JAMES	7900	3
•	30	6	ALLEN, TURNER, JAMES	7900	3
•	30	6	WARD, BLAKE, JAMES	7900	3
•	30	6	WARD, MARTIN, JAMES	7900	3
•	30	6	WARD, TURNER, JAMES	7900	3

5. Recursive

•	20	5	SMITH, JONES, SCOTT, ADAMS	7876	4
•	20	5	SMITH, JONES, SCOTT, FORD	7902	4
•	20	5	SMITH, SCOTT, ADAMS, FORD	7902	4
•	20	5	SMITH, JONES, ADAMS, FORD	7902	4
•	20	5	JONES, SCOTT, ADAMS, FORD	7902	4
•	30	6	ALLEN, WARD, MARTIN, BLAKE	7698	4
•	30	6	ALLEN, WARD, MARTIN, TURNER	7844	4
•	30	6	ALLEN, MARTIN, BLAKE, TURNER	7844	4
•	30	6	ALLEN, WARD, BLAKE, TURNER	7844	4
•	30	6	WARD, MARTIN, BLAKE, TURNER	7844	4
•	30	6	WARD, MARTIN, BLAKE, JAMES	7900	4
•	30	6	ALLEN, BLAKE, TURNER, JAMES	7900	4
•	30	6	WARD, MARTIN, TURNER, JAMES	7900	4
•	30	6	WARD, BLAKE, TURNER, JAMES	7900	4
•	30	6	ALLEN, WARD, BLAKE, JAMES	7900	4
•	30	6	ALLEN, MARTIN, BLAKE, JAMES	7900	4
•	30	6	ALLEN, WARD, MARTIN, JAMES	7900	4
•	30	6	ALLEN, MARTIN, TURNER, JAMES	7900	4
•	30	6	ALLEN, WARD, TURNER, JAMES	7900	4
•	30	6	MARTIN, BLAKE, TURNER, JAMES	7900	4

5. Recursive

- 20 5 SMITH, JONES, SCOTT, ADAMS, FORD 7902 5
- 30 6 ALLEN, WARD, MARTIN, BLAKE, TURNER 7844 5
- 30 6 ALLEN, WARD, MARTIN, BLAKE, JAMES 7900 5
- 30 6 ALLEN, WARD, MARTIN, TURNER, JAMES 7900 5
- 30 6 ALLEN, MARTIN, BLAKE, TURNER, JAMES 7900 5
- 30 6 ALLEN, WARD, BLAKE, TURNER, JAMES 7900 5
- 30 6 WARD, MARTIN, BLAKE, TURNER, JAMES 7900 5
- 30 6 ALLEN, WAD, MARTIN, BLAKE, TURNER, JAMES 7900 6

- select deptno, list from x where len = cnt
- order by deptno
- 10 CLARK, KING, MILLER
- 20 SMITH, ADAMS, FORD, SCOTT, JONES
- 30 ALLEN, BLAKE, MARTIN, JAMES, TURNER, WARD

Working with Ranges

Ranges

- are common in everyday life
- for example, projects that we work on range over consecutive periods of time
- in SQL, it's often necessary to search for ranges, or to generate ranges, or to otherwise manipulate range-based data

1992: Institutul Politehnic Universitatea Tehnica din Cluj-Napoca

- 1992-2030 Automatică și Calculatoare
- 1992-2030 Electronică, Telecomunicații și Tehnologia Informației
- 1992-2030 Inginerie Electrică
- 1992-2030 Construcții și Instalații
- *2007 Erase ...*
- 1992-2007 Construcții și Instalații
- 2007-2030 Construcții
- 2007-2030 Construcții și Instalații

1992: Institutul Politehnic Universitatea Tehnica din Cluj-Napoca

- 1992-2030 Construcții de Mașini
- 1992-2030 Mecanică
- 1992-2011 Știința și Ingineria Materialelor
- 2011-2030 Ingineria Materialelor și a Mediului
- 1998-2030 Arhitectură și Urbanism
- 2011-2030 Inginerie CUN Baia-Mare
- 2011-2014 Resurse Minerale și Mediu CUN Baia-Mare
- 2011-2030 Litere CUN Baia-Mare
- 2011-2030 Științe CUN Baia-Mare

Temporal DataBase

- **don't delete anything from database**
- **change range of time at which information was valid**

Locate Range of Consecutive Values

- PROJ_ID PROJ_START PROJ_END
- 1 01-JAN-2005 02-JAN-2005
- 2 02-JAN-2005 03-JAN-2005
- 3 03-JAN-2005 04-JAN-2005
- 4 04-JAN-2005 05-JAN-2005
- 5 06-JAN-2005 07-JAN-2005
- ...
- 12 27-JAN-2005 28-JAN-2005
- 13 28-JAN-2005 29-JAN-2005
- 14 29-JAN-2005 30-JAN-2005

Locate Range of Consecutive Values

- excluding first row, each row's PROJ_START should equal PROJ_END of row before it
 - "before" is defined as PROJ_ID-1 for current row
- want to find range of dates for consecutive projects, return all rows where current PROJ_END equals next row's PROJ_START

- SELECT v1.proj_id, v1.proj_start, v1.proj_end
 - FROM V v1, V v2
 - WHERE v1.proj_end = v2.proj_start
-
- ALTER TABLE V
 - CONSTRAINT CHK_StartEnd
 - CHECK (proj_start < proj_end);

Locate Beginning and End of Range of Consecutive Values

- PROJ_ID PROJ_START PROJ_END
- 1 01-JAN-2005 02-JAN-2005
- 2 02-JAN-2005 03-JAN-2005
- 3 03-JAN-2005 04-JAN-2005
- 4 04-JAN-2005 05-JAN-2005
- 5 06-JAN-2005 07-JAN-2005
- 6 16-JAN-2005 17-JAN-2005
- 7 17-JAN-2005 18-JAN-2005
- 8 18-JAN-2005 19-JAN-2005
- 9 19-JAN-2005 20-JAN-2005
- 10 21-JAN-2005 22-JAN-2005
- 11 26-JAN-2005 27-JAN-2005
- 12 27-JAN-2005 28-JAN-2005
- 13 28-JAN-2005 29-JAN-2005
- 14 29-JAN-2005 30-JAN-2005

Locate Beginning and End of Range of Consecutive Values

- PROJ_ID PROJ_START PROJ_END
- 1 01-JAN-2005 02-JAN-2005
- 2 02-JAN-2005 03-JAN-2005
- 3 03-JAN-2005 04-JAN-2005
- 4 04-JAN-2005 05-JAN-2005
- 5 06-JAN-2005 07-JAN-2005
- 6 16-JAN-2005 17-JAN-2005
- 7 17-JAN-2005 18-JAN-2005
- 8 18-JAN-2005 19-JAN-2005
- 9 19-JAN-2005 20-JAN-2005
- 10 21-JAN-2005 22-JAN-2005
- 11 26-JAN-2005 27-JAN-2005
- 12 27-JAN-2005 28-JAN-2005
- 13 28-JAN-2005 29-JAN-2005
- 14 29-JAN-2005 30-JAN-2005

Locate Beginning and End of Range of Consecutive Values

- GROUP_PROJ_ID PROJ_START PROJ_END
- 1 01-JAN-2005 05-JAN-2005
- 2 06-JAN-2005 07-JAN-2005
- 3 16-JAN-2005 20-JAN-2005
- 4 21-JAN-2005 22-JAN-2005
- 5 26-JAN-2005 30-JAN-2005

- select V.* ,
- case
- when (select b.proj_id from V b where V.proj_start = b.proj_end) is not null
- then 0 else 1 end as flag
- from V

V2

- PROJ_ID PROJ_START PROJ_END FLAG
- 1 01-JAN-2005 02-JAN-2005 1
- 2 02-JAN-2005 03-JAN-2005 0
- 3 03-JAN-2005 04-JAN-2005 0
- 4 04-JAN-2005 05-JAN-2005 0
- 5 06-JAN-2005 07-JAN-2005 1
- 6 16-JAN-2005 17-JAN-2005 1
- 7 17-JAN-2005 18-JAN-2005 0
- 8 18-JAN-2005 19-JAN-2005 0
- 9 19-JAN-2005 20-JAN-2005 0
- 10 21-JAN-2005 22-JAN-2005 1
- 11 26-JAN-2005 27-JAN-2005 1
- 12 27-JAN-2005 28-JAN-2005 0
- 13 28-JAN-2005 29-JAN-2005 0
- 14 29-JAN-2005 30-JAN-2005 0

- view V2 uses scalar subquery in CASE expression to determine whether or not particular row is part of set of consecutive values; aliased FLAG, returns 0 if current row is part of consecutive set or 1 if it is not
 - membership in consecutive set is determined by whether or not there is record with PROJ_END value that matches current row's PROJ_START value

- select proj_grp,
- min(proj_start) as proj_start,
- max(proj_end) as proj_end
- from (select a.proj_id, a.proj_start,
a.proj_end, (select sum(b.flag) from V2 b
where b.proj_id <= V2.proj_id) as proj_grp
- from V2) x
- group by proj_grp

- `select a.proj_id, a.proj_start, a.proj_end,`
- `(select sum(b.flag) from v2 b where b.proj_id <= V2.proj_id) as proj_grp from V2`
- returns all rows from view V2 along with running total on FLAG; this running total is what creates our groups

- PROJ_ID PROJ_START PROJ_END FLAG
- 1 01-JAN-2005 02-JAN-2005 1
- 2 02-JAN-2005 03-JAN-2005 1
- 3 03-JAN-2005 04-JAN-2005 1
- 4 04-JAN-2005 05-JAN-2005 1
- 5 06-JAN-2005 07-JAN-2005 2
- 6 16-JAN-2005 17-JAN-2005 3
- 7 17-JAN-2005 18-JAN-2005 3
- 8 18-JAN-2005 19-JAN-2005 3
- 9 19-JAN-2005 20-JAN-2005 3
- 10 21-JAN-2005 22-JAN-2005 4
- 11 26-JAN-2005 27-JAN-2005 5
- 12 27-JAN-2005 28-JAN-2005 5
- 13 28-JAN-2005 29-JAN-2005 5
- 14 29-JAN-2005 30-JAN-2005 5

Find Differences Between Rows in Same Group or Partition

- return DEPTNO, ENAME, and SAL of each employee along with difference in SAL between employees in same department - difference should be between each current employee and employee hired immediately afterwards; for each employee hired last in his department, return "N/A" for difference
- correlation between seniority and salary on "per department" basis

- SELECT deptno, ename, hiredate, sal,
- coalesce(cast (sal - next_sal as char(10)),
'N/A') as diff
- FROM (...)

- select e.deptno, e.ename, e.hiredate, e.sal,
(select min(hiredate) from emp d where
e.deptno = d.deptno and d.hiredate >
e.hiredate) as next_hire from emp e order by
deptno, hiredate

- DEPTNO ENAME HIREDATE SAL NEXT_HIRE
- 10 CLARK 09-JUN-1981 2450 17-NOV-1981
- 10 KING 17-NOV-1981 5000 23-JAN-1982
- 10 MILLER 23-JAN-1982 1300
- 20 SMITH 17-DEC-1980 800 02-APR-1981
- 20 JONES 02-APR-1981 2975 03-DEC-1981
- 20 FORD 03-DEC-1981 3000 09-DEC-1982
- 20 SCOTT 09-DEC-1982 3000 12-JAN-1983
- 20 ADAMS 12-JAN-1983 1100
- 30 ALLEN 20-FEB-1981 1600 22-FEB-1981
- 30 WARD 22-FEB-1981 1250 01-MAY-1981
- 30 BLAKE 01-MAY-1981 2850 08-SEP-1981
- 30 TURNER 08-SEP-1981 1500 28-SEP-1981
- 30 MARTIN 28-SEP-1981 1250 03-DEC-1981
- 30 JAMES 03-DEC-1981 950

- select e.deptno, e.ename, e.hiredate, e.sal,
- (select min(sal) from emp d where d.deptno = e.deptno and d.hiredate =
- (select min(hiredate) from emp c
- where e.deptno = c.deptno and c.hiredate > e.hiredate)) as next_sal
- from emp e

Generate Consecutive Numeric Values

- with x (id) as (
- select 1 from t1
- union all
- select id+1 from x where id+1 <= 10)
- select * from x

Advanced Searching

Paginate Through Result Set

- there is no concept of first, last, or next in SQL, impose order on rows you are working with
- use window function ROW_NUMBER OVER to impose order, and specify window of records that you want returned in your WHERE clause
- for example, to return rows 1 through 5

- select ename, sal from (select row_number() over (order by sal) as rn, ename, sal from emp) x
- where rn between 1 and 5
- where rn between 6 and 10
- window function ROW_NUMBER OVER will assign unique number to each data item in increasing order starting from 1

Skip Rows from Table

- select ename, sal from (
- select row_number() over (order by ename)
rn, ename, sal from emp) x
- where mod(rn,2) = 1

Find Records with High & Low Values

- select ename, sal from (
- select ename, sal,
- min(sal)over() min_sal,
- max(sal)over() max_sal
- from emp) x
- where sal in (min_sal, max_sal)

- select e.ename, e.sal,
- (select min(sal) from emp d where d.sal > e.sal) as forward,
- (select max(sal) from emp d where d.sal < e.sal) as rewind
- from emp e order by sal

- SMITH 800 950
- JAMES 950 1100 800
- ADAMS 1100 1250 950
- WARD 1250 1250 1100
- MARTIN 1250 1300 1250
- MILLER 1300 1500 1250
- TURNER 1500 1600 1300
- ALLEN 1600 2450 1500
- CLARK 2450 2850 1600
- BLAKE 2850 2975 2450
- JONES 2975 3000 2850
- SCOTT 3000 3000 2975
- FORD 3000 5000 3000
- KING 5000 800

Rank Results

- rank salaries in table EMP while allowing for ties
- select dense_rank() over(order by sal) rnk, ename, sal from emp

- 1 SMITH 800
- 2 JAMES 950
- 3 ADAMS 1100
- 4 WARD 1250
- 4 MARTIN 1250
- 5 MILLER 1300
- 6 TURNER 1500
- 7 ALLEN 1600
- 8 CLARK 2450
- 9 BLAKE 2850
- 10 JONES 2975
- 11 SCOTT 3000
- 11 FORD 3000
- 11 KING 5000

Reporting and Warehousing

Pivot Result Set

- DEPTNO CNT
- 10 3
- 20 5
- 30 6
- Dept_No_10 Dept_No_20 Dept_No_30
- 3 5 6

- select deptno,
- case when deptno=10 then 1 else 0 end as dept_no_10,
- case when deptno=20 then 1 else 0 end as dept_no_20,
- case when deptno=30 then 1 else 0 end as dept_no_30
- from emp order by deptno

- select
- sum(case when deptno=10 then 1 else 0 end)
as dept_no_10,
- sum(case when deptno=20 then 1 else 0 end)
as dept_no_20,
- sum(case when deptno=30 then 1 else 0 end)
as dept_no_30
- from emp

- select
- max(case when deptno=10 then empcount
else null end) as dept_no_10
- max(case when deptno=20 then empcount
else null end) as dept_no_20,
- max(case when deptno=10 then empcount
else null end) as dept_no_30
- from (select deptno, count(*) as empcount
from emp group by deptno) x

- 10 3 NULL NULL
- 20 NULL 5 NULL
- 30 NULL NULL 6

Pivot Result Set in Multiple Rows

- *CLERKS ANALYSTS MGRS PREZ SALES*
- MILLER FORD CLARK KING TURNER
- JAMES SCOTT BLAKE MARTIN
- ADAMS JONES WARD
- SMITH ALLEN

- select
- max(case when job='CLERK' then ename else null end) as clerks,
- max(case when job='ANALYST' then ename else null end) as analysts,
- max(case when job='MANAGER' then ename else null end) as mgrs,
- max(case when job='PRESIDENT' then ename else null end) as prez,
- max(case when job='SALESMAN' then ename else null end) as sales
- from (select job, ename,
row_number()over(partition by job order by ename)
rn from emp) group by rn

- ANALYST FORD 1
- ANALYST SCOTT 2
- CLERK ADAMS 1
- CLERK JAMES 2
- CLERK MILLER 3
- CLERK SMITH 4
- MANAGER BLAKE 1
- MANAGER CLARK 2
- MANAGER JONES 3
- PRESIDENT KING 1
- SALESMAN ALLEN 1
- SALESMAN MARTIN 2
- SALESMAN TURNER 3
- SALESMAN WARD 4

- | | RN | CLERKS | ANALYSTS | MGRS | PREZ | SALES |
|---|----|--------|----------|-------|------|--------|
| • | 1 | FORD | | | | |
| • | 2 | SCOTT | | | | |
| • | 1 | | ADAMS | | | |
| • | 2 | | JAMES | | | |
| • | 3 | | MILLER | | | |
| • | 4 | | SMITH | | | |
| • | 1 | | | BLAKE | | |
| • | 2 | | | CLARK | | |
| • | 3 | | | JONES | | |
| • | 1 | | | | KING | |
| • | 1 | | | | | ALLEN |
| • | 2 | | | | | MARTIN |
| • | 3 | | | | | TURNER |
| • | 4 | | | | | WARD |

Calculate Simple Subtotals

- simple subtotal defined as result set that contains values from aggregation of one column along with grand total value for table
- example would be result set that sums salaries in table EMP by JOB, and that also includes sum of all salaries
- | JOB | SAL |
|-----------|-------|
| ANALYST | 6000 |
| CLERK | 4150 |
| MANAGER | 8275 |
| PRESIDENT | 5000 |
| SALESMAN | 5600 |
| TOTAL | 29025 |

with rollup

- select coalesce(job,'TOTAL') job,
- sum(sal) sal
- from emp
- group by job
- with rollup

SQL SERVER PIVOT OPERATOR

Create Cross-Tab Reports

- select [10] as dept_10, [20] as dept_20,
- [30] as dept_30, [40] as dept_40
- from (select deptno, empno from emp) driver
- pivot (count(driver.empno) for driver.deptno
in ([10], [20], [30], [40])) as empPivot

- If there are any DEPTNOs with a value of 10, perform aggregate operation defined (COUNT(DRIVER.EMPNO)) for those rows
- Repeat for DEPTNOs 20, 30, and 40.
- items listed in brackets serve not only to define values for which aggregation is performed; items also become column names in result set (without square brackets)
- in SELECT clause of solution, items in FOR list are referenced and aliased
- inline view DRIVER is just that, an inline view, you may put more complex SQL in there

syntax for PIVOT

- SELECT <non-pivoted column>,
 [first pivoted column] AS <column name>,
 [second pivoted column] AS <column name>,
• ...
• [last pivoted column] AS <column name>
- FROM
 (<SELECT query that produces the data>)
- AS <alias for the source query>
- PIVOT
• (
- <aggregation function>(<column being aggregated>)
- FOR
• [<column that contains the values that will become column headers>]
- IN ([first pivoted column], [second pivoted column],
• ... [last pivoted column])
-) AS <alias for the pivot table>
- <optional ORDER BY clause>;

syntax for PIVOT

- SELECT <non-pivoted column>,
- [first pivoted column] AS <column name>,
- [second pivoted column] AS <column name>,
- ...
- [last pivoted column] AS <column name>
- FROM

syntax for PIVOT

- SELECT
- ...
- FROM
- (<SELECT query that produces the data>)
- AS <alias for the source query>
- PIVOT
- (
- ...)
-) AS <alias for the pivot table>
- <optional ORDER BY clause>;

syntax for PIVOT

- SELECT
- ...
- FROM
 - (<SELECT query that produces the data>)
 - AS <alias for the source query>
- **PIVOT**
- (
- **<aggregation function>(<column being aggregated>)**
- **FOR**
- [<column that contains the values that will become column headers>]
- IN ([first pivoted column], [second pivoted column],
 - ... [last pivoted column])
-) AS <alias for the pivot table>
- <optional ORDER BY clause>;

syntax for PIVOT

- PIVOT (
- <aggregation function>(<column being aggregated>)
- FOR
- [<column that contains the values that will become column headers>]
- IN ([first pivoted column], [second pivoted column],
• ... [last pivoted column])
-) AS <alias for the pivot table>
- <optional ORDER BY clause>;

```
select deptno, avg(sal)  
from emp group by deptno
```

- 10 2916.666666
- 20 2175.000000
- 30 1566.666666

```
select job, avg(sal)  
from emp group by job
```

- ANALYST 3000.000000
- CLERK 1037.500000
- MANAGER 2758.333333
- PRESIDENT 5000.000000
- SALESMAN 1400.000000

- SELECT 'Average Sal' AS Average_Sal_by_Dept_No,
- [10], [20], [30], [40]
- FROM
- (SELECT DeptNo, Sal
- FROM Emp) AS SourceTable
- PIVOT
- (
- AVG(Sal)
- FOR DeptNo IN ([10], [20], [30], [40])
-) AS PivotTable;

- Average_Sal_by_Dept_No 10 20 30 40
- Average Sal 2916.66 2175.00 1566.66
NULL

- SELECT *
- FROM
- (SELECT Job, Sal
- FROM Emp) AS SourceTable
- PIVOT
- (
- AVG(Sal)
- FOR Job IN ([ANALYST], [CLERK], [MANAGER],
[PRESIDENT], [SALESMAN])
-) AS PivotTable;

- ANALYST CLERKMANAGER PRESIDENT SALESMAN
- 3000.00 1037.50 2758.33 5000.00 1400.00

Hierarchical Queries

Express Parent-Child Relationship

- SELECT e.ename, m.ename as manager
- FROM emp e LEFT OUTER JOIN emp m
- ON e.mgr = m.empno

Child-Parent-Grandparent

- SELECT
- c.ename as Child,
- p.ename as Parent,
- g.ename as Grandparent
- FROM emp c
- LEFT OUTER JOIN emp p ON c.mgr = p.empno
- LEFT OUTER JOIN emp g ON p.mgr = g.empno

- Child Parent Grandparent
- SMITH FORD JONES
- ALLEN BLAKE KING
- WARD BLAKE KING
- JONES KING NULL
- MARTIN BLAKE KING
- BLAKE KING NULL
- CLARK KING NULL
- SCOTT JONES KING
- KING NULL NULL
- TURNER BLAKE KING
- ADAMS SCOTT JONES
- JAMES BLAKE KING
- FORD JONES KING
- MILLER CLARK KING

- with x (tree,mgr,depth)
- as (
- select cast(ename as varchar(100)), mgr, 0
- from emp
- union all
- select cast(x.tree+'-->' +e.ename as varchar(100)),
e.mgr, x.depth+1
- from emp e, x where x.mgr = e.empno)
- select tree leaf____branch____root from x
- where depth = 2

tree	mgr	depth
MILLER-->CLARK-->KING	NULL	2
FORD-->JONES-->KING	NULL	2
JAMES-->BLAKE-->KING	NULL	2
ADAMS-->SCOTT-->JONES	7839	2
TURNER-->BLAKE-->KING	NULL	2
SCOTT-->JONES-->KING	NULL	2
MARTIN-->BLAKE-->KING	NULL	2
WARD-->BLAKE-->KING	NULL	2
ALLEN-->BLAKE-->KING	NULL	2
SMITH-->FORD-->JONES	7839	2

where depth = 3

Create Hierarchical View of Table

- with x (ename,tree,mgr,depth)
- as (
- select ename, cast(ename as varchar(100)), mgr, 0
- from emp
- union all
- select x.ename, cast(x.tree+'-->' +e.ename as varchar(100)), e.mgr, x.depth+1
- from emp e, x where x.mgr = e.empno)
-
- select ename, tree,depth from x where mgr is null

• ename	tree	depth
• KING	KING	0
• MILLER	MILLER-->CLARK-->KING	2
• FORD	FORD-->JONES-->KING	2
• JAMES	JAMES-->BLAKE-->KING	2
• ADAMS	ADAMS-->SCOTT-->JONES-->KING	3
• TURNER	TURNER-->BLAKE-->KING	2
• SCOTT	SCOTT-->JONES-->KING	2
• CLARK	CLARK-->KING	1
• BLAKE	BLAKE-->KING	1
• MARTIN	MARTIN-->BLAKE-->KING	2
• JONES	JONES-->KING	1
• WARD	WARD-->BLAKE-->KING	2
• ALLEN	ALLEN-->BLAKE-->KING	2
• SMITH	SMITH-->FORD-->JONES-->KING	3

- It's good to be the King!

**Thank you for your kindly
attention!**

Business Intelligence

Reporting Services

- SQL Server Reporting Services is one component of Microsoft's Business Intelligence platform

Business Intelligence (BI)

- refers to computer-based techniques used in spotting, digging, analyzing business data
- provide historical, current, and predictive views of business operations, common functions
 - reporting
 - online analytical processing, data mining
 - business performance management, benchmarking, predictive analytics

Business Intelligence

- aims to support better business decision-making
- can be called a decision support system (DSS)
- often BI applications use data gathered from a data warehouse or a data mart
 - however, not all data warehouses are used for business intelligence, nor do all business intelligence applications require a data warehouse

Life is filled with decisions

- Should I read these course notes slides or should I skip these
- first bit of business intelligence: Read this!

Data warehouse

- repository of an organization's electronically stored data
- designed to facilitate reporting and analysis
- definition focuses on data storage
- means to retrieve and analyze data, to extract, transform and load data, and to manage the data dictionary are also considered essential components of a data warehousing system

Extract, Transform, Load

- process in database usage and especially in data warehousing that involves
- Extracting data from outside sources
- Transforming it to fit operational needs
- Loading it into the end target (database or data warehouse)

problem
example

Report Classic Models

- company which buys collectable model cars, trains, trucks, buses, trains and ships directly from manufacturers and sells them to distributors across the globe

Offices

- Classic Models Inc. has 7 offices worldwide (San Francisco, Boston, NYC, Paris, Tokyo, Sydney, London)
- based on geography each office is assigned to a sales territory
 - APAC, NA, EMEA, JAPAN

Employees

- company has 23 employees
 - 6 Execs and 17 Sales Reps
- assigned to one of company's 7offices
- Sales Reps also assigned to a number of customers (distributors) in their respective geographies that they sell to
- Sales Rep reports to Sales Manager for his/her territory ... exceptions
- Execs don't work directly with customers

Customers

- Classic Models Inc. has 122 customers across the world

Products

- Classic Models Inc. sells 110 unique models which they purchase from 13 vendors
- for each product price at which product was purchased from vendor as well as Manufacturer Suggested Retail Price are provided
 - MSRP price is on average 45% (30% to 60%) above buyPrice

Product Lines

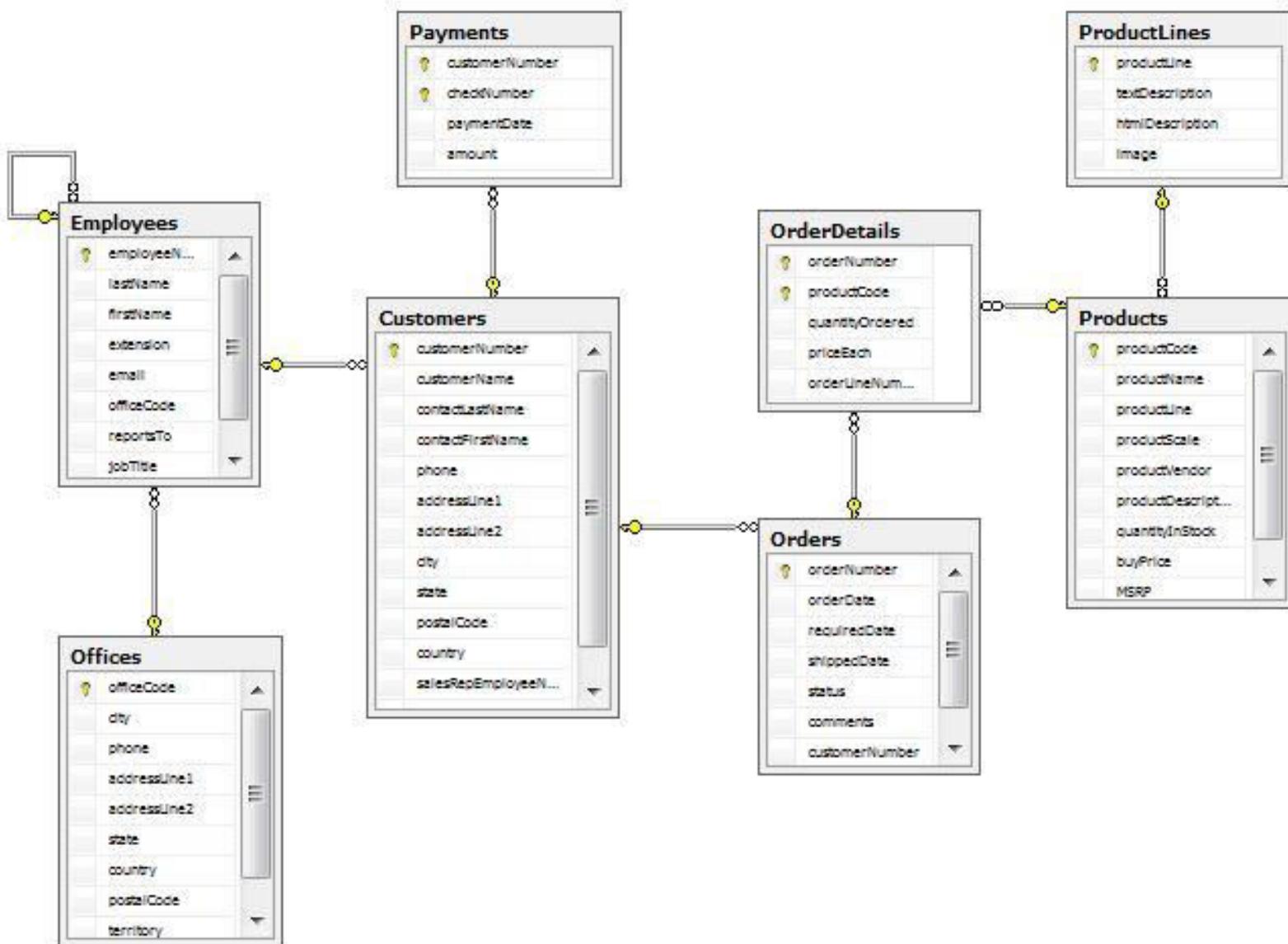
- models are classified as 7 distinct product lines
 - Classic Cars, Vintage Cars, Motorcycles, Trucks and Buses, Planes, Ships, Trains

Orders

- customers place their orders and expect to receive them approximately within 6 to 10 days
- once an order is placed it's assembled and shipped within 1 to 6 days (7-8 for Japan)
- can be in one of these states:
 - In Process, Shipped, Cancelled, Disputed, Resolved, On Hold

Order Details

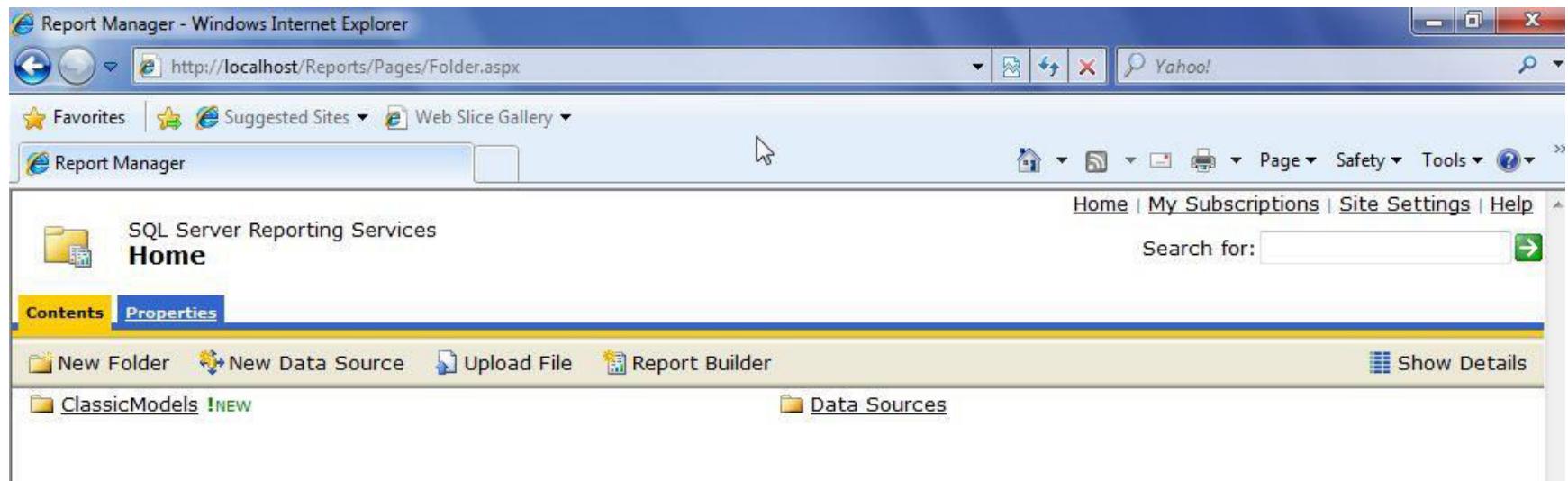
- each order contains an average of 9 unique products (order line items) with an average quantity of 35 per product
- reflects negotiated price per product
 - based on corresponding product's MSRP as well as quantity per product and maybe customer or other elements



Big Boss

- better management = business intelligence
- predict future based on past history
 - already stored in database
- what was the value added
 - by different dimensions

Report Manager



Dimensions

The screenshot shows a Microsoft Internet Explorer window titled "Report Manager - Windows Internet Explorer". The address bar displays the URL <http://localhost/Reports/Pages/Folder.aspx?ItemPath=%2fClassicModels&ViewMode=List>. The page content is the "Report Manager" interface for the "ClassicModels" folder under "SQL Server Reporting Services". The left sidebar shows a tree view with nodes: Home > **ClassicModels**, which contains sub-nodes: SalesEmployee !NEW, SalesGeographically !NEW, SalesProduct !NEW, SalesTime !NEW, and SalesVendor !NEW. The top navigation bar includes links for Home, My Subscriptions, Site Settings, Help, and a search bar. The bottom toolbar includes buttons for New Folder, New Data Source, Upload File, Report Builder, and Show Details.

Employee

The screenshot shows a Microsoft Internet Explorer window displaying a report from SQL Server Reporting Services. The title bar reads "Report Manager - Windows Internet Explorer" and the address bar shows the URL "http://localhost/Reports/Pages/Report.aspx?ItemPath=%2fClassicModels%2fSalesEmployee". The report itself is titled "Sales Employee" and displays a detailed table of sales data. The table has columns for Territory, Office, Employee, Customer, Order Date, Buy Price, Price / Item, and Quantity. The data is grouped by Territory (APAC and EMEA) and Office (4 Paris). The report includes several rows of data for each office, such as Gerard Hernandez, Loui Bondur, Martin Gerard, CAF Imports, Corrida Auto Replicas, Ltd., and Enaco Distributors.

Territory	Office	Employee	Customer	Order Date	Buy Price	Price / Item	Quantity
APAC					19,857.3	32,865.27	12,878
EMEA					77,084.62	128,224.01	49,578
	4 Paris	Gerard Hernandez			51,952.28	86,624.44	33,881
		Loui Bondur			20,881.46	34,668.3	14,231
		Martin Gerard			9,543.25	16,225.21	6,180
			CAF Imports		6,291.76	10,582.34	4,180
			Corrida Auto Replicas, Ltd.		745.7	1,249.62	468
				28 May 2003	1,738.3	3,064.57	1,161
				26 Jan 2004	961.59	1,623.71	611
				01 Nov 2004	315.97	623.43	241
					460.74	817.43	301
			Enaco Distributors		1,141.93	1,776.13	882
			Iberia Gift Imports, Corp.		744.26	1,298.02	589
			Vida Sport, Ltd		1,921.57	3,194	1,078
		Pamela Castillo			15,235.81	25,148.59	9,290

Employee - ?

- Employee
 - Customer
 - Date
- Employee
 - Date
 - Customer

Geographically

The screenshot shows a Microsoft Excel spreadsheet with data from the Northwind database. The data is organized into several columns: Territory, Country, Customer, Order Date, Buy Price, Price / Item, Quantity, Discount, and Value. The rows are grouped by Territory (APAC, EMEA, Japan, NA) and Country (Australia, New Zealand, Singapore). The data includes various customers like Anna's Decorations, Ltd, Australian Collectables, Ltd, etc., with their respective order dates, prices, and quantities.

Territory	Country	Customer	Order Date	Buy Price	Price / Item	Quantity	Discount	Value
APAC				19,857.3	32,865.27	12,878	3,843.87	
	Australia			10,063.73	16,687.78	6,246	1,897.24	
		Anna's Decorations, Ltd		2,590.99	4,314.94	1,469	524.39	
			11 Sep 2003	828.76	1,374.9	430	164.05	
			04 Nov 2003	651.4	1,130.7	444	167.63	
			20 Jan 2005	558.84	898.11	256	91.84	
			09 Mar 2005	551.99	911.23	339	100.87	
		Australian Collectables, Ltd		1,069.46	1,816.84	705	221.35	
				3,149.21	5,159.19	1,926	564.63	
				900.36	1,528.57	545	152.3	
				2,353.71	3,868.24	1,601	434.57	
	New Zealand			7,845.78	13,003.07	5,396	1,605.81	
	Singapore			1,947.79	3,174.42	1,236	340.82	
EMEA				77,084.62	128,224.01	49,578	14,497.67	
Japan				7,651.64	12,635.69	4,923	1,405.37	
NA				58,916.55	98,220.45	38,137	10,769.56	

Products

Sales Product

Product Line	Product Name	Date	Buy Price	Price / Item	Quantity	Price	Value Added
Classic Cars			65,924.62	109,084.52	35,582	3,853,922.49	1,526,212.2
Motorcycles			18,254.99	31,348.93	12,778	1,121,426.12	469,255.3
Planes			16,675.4	26,989.94	11,872	954,637.54	365,960.71
	1900s Vintage Bi-Plane		959	1,726.39	940	58,434.07	26,239.07
	1900s Vintage Tri-Plane		1,014.44	1,896.02	1,009	68,276.35	31,720.28
		17 Feb 2003	36.23	71	26	1,846	904.02
		29 Apr 2003	36.23	71.73	29	2,080.17	1,029.5
		27 Jun 2003	36.23	61.58	46	2,832.68	1,166.1
		25 Aug 2003	36.23	71.73	33	2,367.09	1,171.5
		09 Oct 2003	36.23	66.65	33	2,199.45	1,003.86
		28 Oct 2003	36.23	68.1	48	3,268.8	1,529.76
		11 Nov 2003	36.23	66.65	27	1,799.55	821.34
		15 Nov 2003	36.23	64.48	33	2,127.84	932.25
		01 Dec 2003	36.23	70.28	39	2,740.92	1,327.95
		12 Jan 2004	36.23	68.1	40	2,724	1,274.8

Geography - Products

- report to illustrate the geography of sold products

Time

Sales by Time

Year	Quarter	Month	Order Date	Buy Price	Price / Item	Quantity	Price	Value Added
2003				57,567.79	95,687.83	36,439	3,317,348.39	1,320,622.94
2004				77,427.76	129,122.11	49,487	4,515,905.51	1,809,381.14
	Q1			13,341.98	22,274.89	8,694	799,579.31	319,596.74
		Q1		4,877.01	8,249.85	3,245	292,385.21	119,453.03
			02 Jan 2004	872.33	1,438.31	530	49,614.72	19,397.02
			09 Jan 2004	383.66	646.57	269	21,053.69	8,598.51
				85.68	129.2	39	5,038.8	1,697.28
				51.61	82.58	28	2,312.24	867.16
				64.58	97.4	20	1,948	656.4
				34.25	66.45	43	2,857.35	1,384.6
				26.3	56.55	36	2,035.8	1,089
				48.64	79.67	22	1,752.74	682.66
				39.83	90.52	33	2,987.16	1,672.77
				32.77	44.2	48	2,121.6	548.64
			12 Jan 2004	877.09	1,443.06	577	47,177.59	18,374.31
			15 Jan 2004	761.07	1,399.57	530	49,165.16	22,311.18

Time

- will always be a dimension in any business

Vendor

Sales Vendor

Vendor	Product Name	Buy Price	Price / Item	Quantity	Price	Value Added
Autoart Studio Design		11,169.21	20,988.33	7,702	736,928.03	344,926.37
Carousel DieCast Legends		11,666.07	18,734.9	8,735	667,190	251,702.34
Classic Metal Creations		15,191.08	25,957.87	9,678	934,554.42	384,438.06
	1928 British Royal Navy Airplane	1,868.72	2,746.46	972	94,885.37	30,014.09
	1938 Cadillac V-16 Presidential Limousine	577.08	1,127.18	955	38,449.09	18,766.54
	1949 Jaguar XK 120	1,181.25	2,018.21	949	76,670.02	31,829.77
	1952 Alpine Renault 1300	2,760.24	5,524.66	961	190,017.96	95,282.58
	1954 Greyhound Scenicruiser	727.44	1,364.12	955	46,519.05	21,708.15
	1956 Porsche 356A Coupe	2,654.1	3,463	1,052	134,240.71	30,829.11
	1957 Corvette Convertible	1,888.11	3,490.85	1,013	130,749.31	59,910.22
	1961 Chevrolet Impala	872.91	1,978.17	941	69,120.97	38,698.44
	1962 City of Detroit Streetcar	1,012.23	1,452.78	966	52,123.81	15,908.47
	1965 Aston Martin DB5	1,649	2,792.44	914	101,778.13	41,490.69
Exoto Designs		14,250.53	22,167.6	8,604	793,392.31	282,537.09
Gearbox Collectibles		14,165.46	23,984.34	8,352	828,013.76	338,223.61

Vendor

- based on this example and just this view it seems that ?

Vendor

- AutoArt Studio Design is the best vendor
 - cheap buy price, large value added, highest profitability
- Carousel Diecast Legends is the worst vendor
 - large buy price, small value added, lowest profitability

Dashboard Report

- report to senior management that provides at-a-glance perspective on current status of company in context of predetermined metrics
- depending on organization, those metrics may include cost, profitability, value added, time, requirements, risk, customer satisfaction, or other measures critical to management team
- provides management with quick understanding of current business posture, without detailed explanation of causes or solutions

Business Intelligence

Data Warehouse
Cub

Business intelligence

- Business intelligence is the delivery of accurate, useful information to the appropriate decision makers within the necessary timeframe to support effective decision making.

Transactional data

- information stored to track interactions, or business transactions, carried out by organization
- business transactions of an organization need to be tracked for that organization to operate
- organization needs to keep track of what it has done and what it needs to do

OnLine Transaction Processing

- when these transactions are stored on and managed by computers, we refer to OLTP
- OLTP systems record business interactions as they happen
- support day-to-day operation of organization
- optimized for efficiently processing and storing transactions - breaking data up into small chunks using rules of database normalization

Business intelligence measures

- are not designed to reflect events of one transaction, but to reflect net result of number of transactions over selected period of time
- are often aggregates of hundreds, thousands, or even millions of individual transactions
- designing system to provide these aggregates efficiently requires entirely different optimizations

- need to do is take information stored in OLTP systems and move it into different data store
- intermediate data store can then serve as source for our measure calculations
- when data is stored in this manner, it is referred to as *data mart*

Data Mart

- body of historical data in an electronic repository that does not participate in daily operations of organization
- used to create business intelligence
- usually applies to specific area of organization

Data Warehouse

- tend to be large, one-stop-shopping repositories where all historical data for organization would be stored
- data marts are smaller undertakings that focus on particular aspect of organization
- Business Intelligence
 - Data Warehouse, Data Mart, Cube

Extract, Transform, and Load (ETL) process

- extracts data from one or more OLTP systems, performs any required data consolidation and cleansing (removes inconsistencies and errors from transactional data so it has consistency necessary for use in data mart) to transform data into consistent format, and loads the cleansed data by inserting it into data mart

OnLine Analytical Processing

- type of system that would be tuned to needs of data analysts, online analytical processing, or OLAP, system
- enable users to quickly and easily retrieve information from data, usually data mart, for analysis
- OLAP systems present data using measures, dimensions, hierarchies, and cubes

Data Warehouse Design Cube

Data Mart Structure

- The data we use for business intelligence can be divided into four categories
 - measures
 - dimensions
 - attributes
 - hierarchies

Measure

- forms basis of everything we do with business intelligence - building blocks for effective decision making
- numeric quantity expressing some aspect of organization's performance
- information represented by this quantity is used to support or evaluate decision making and performance of organization
- can also be called a fact; tables that hold measure information known as *fact tables*

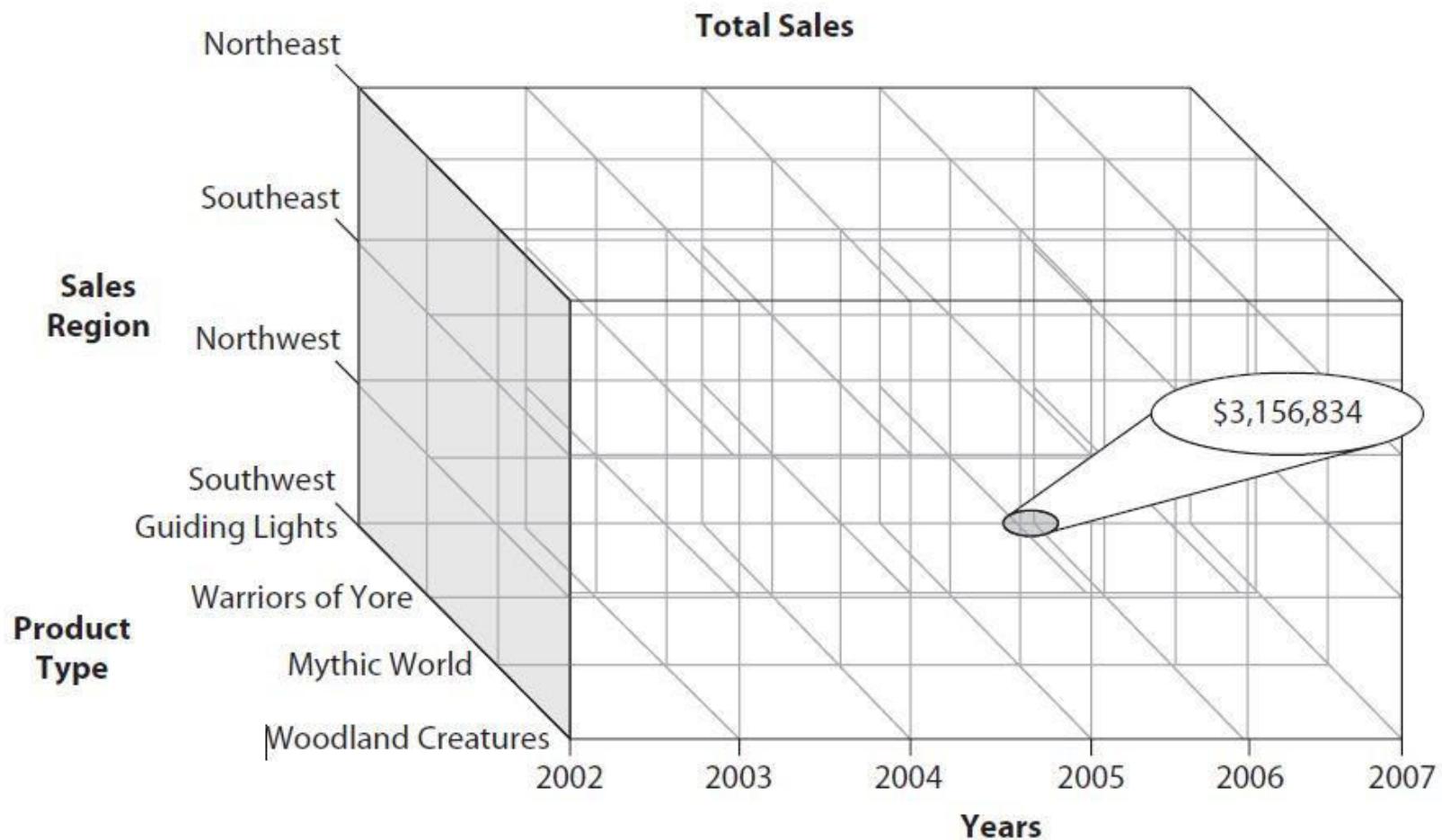
Dimension

- sales is an example of measure that is often required for effective decision making
- however, it is not often that decision makers want to see one single aggregate representing total sales for all products for all salespersons for entire lifetime of organization
- more likely to want to slice and dice this total into smaller parts

Dimension

- categorization used to spread out an aggregate measure to reveal its constituent parts
- used to facilitate this slicing and dicing

Cube



Cube

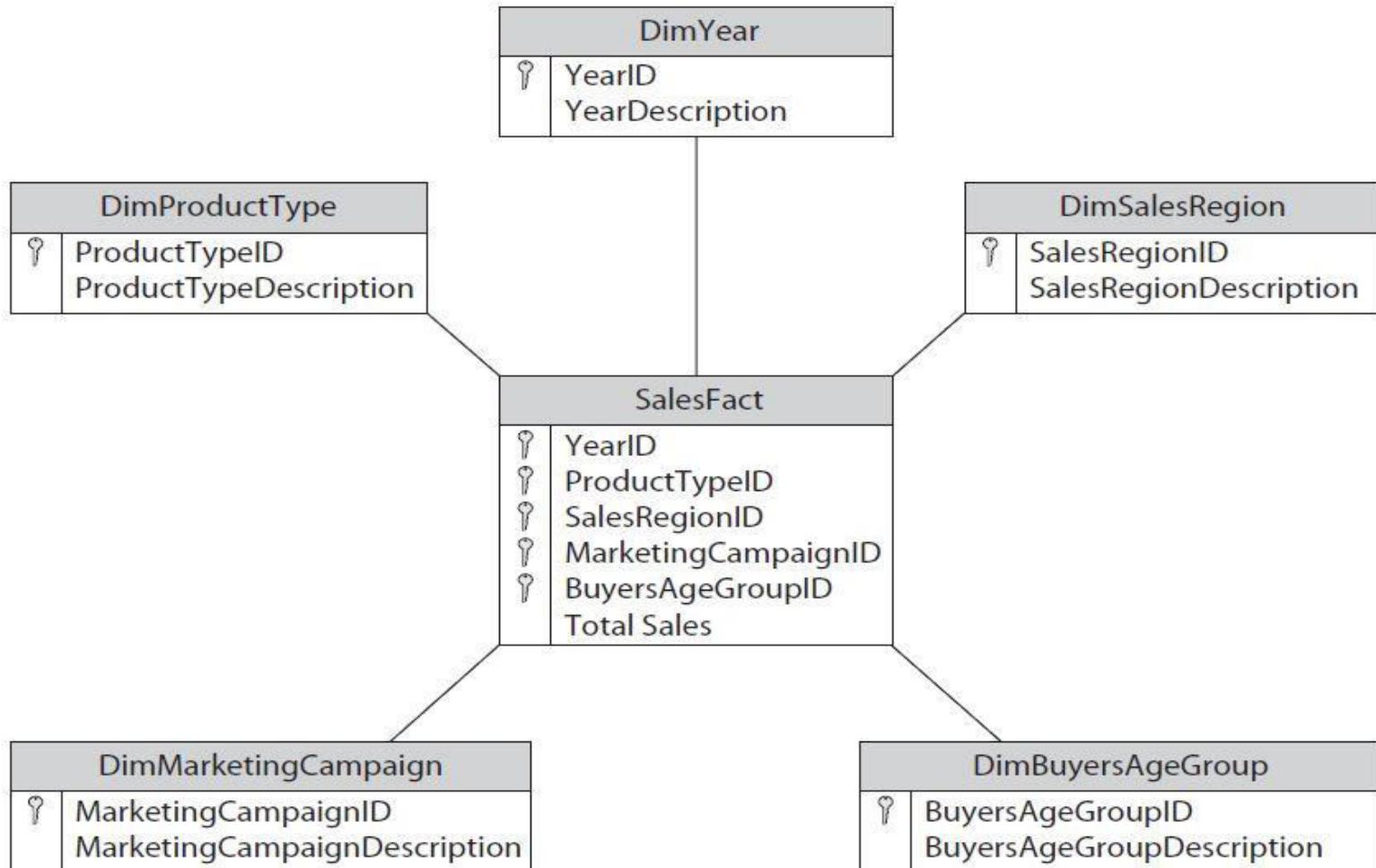
- can continue to spread out measure using additional dimensions, such as marketing campaign dimension and buyer's age
- becomes difficult to represent in illustration
- refer to these structures with four, five, or even more dimensions as cubes
- typical dimension: time, space, products

- measures and dimensions are stored in data mart in one of two layouts, or schemas
- star schema
- snowflake schema

Star Schema

- measures are stored in fact table
- dimensions are stored in dimension tables
- center of star is formed by fact table
- fact table has column for measure and column for each dimension containing foreign key for member of that dimension
- primary key for this table is created by concatenating all of foreign key fields
- fact table may contain multiple measures

Star Schema



Attributes

- additional information about dimension members
- information pertaining to dimension member that is not the unique identifier or description of member
- information about dimension that users likely want as part of their business intelligence output
- also used to store information that may be used to limit or filter records
- stored as additional columns in dimension tables

Hierarchies

- in many cases, dimension is part of larger structure with many levels
- structure known as hierarchy
- structure made up of two or more levels of related dimensions
- dimension at upper level of hierarchy completely contains one or more dimensions from next lower level

Hierarchies

- in star schema information about hierarchies is stored right in dimension tables
- primary key in each of dimension tables is at the lowest level of hierarchy
- fact table must be changed so its foreign keys point to lowest levels in hierarchies as well

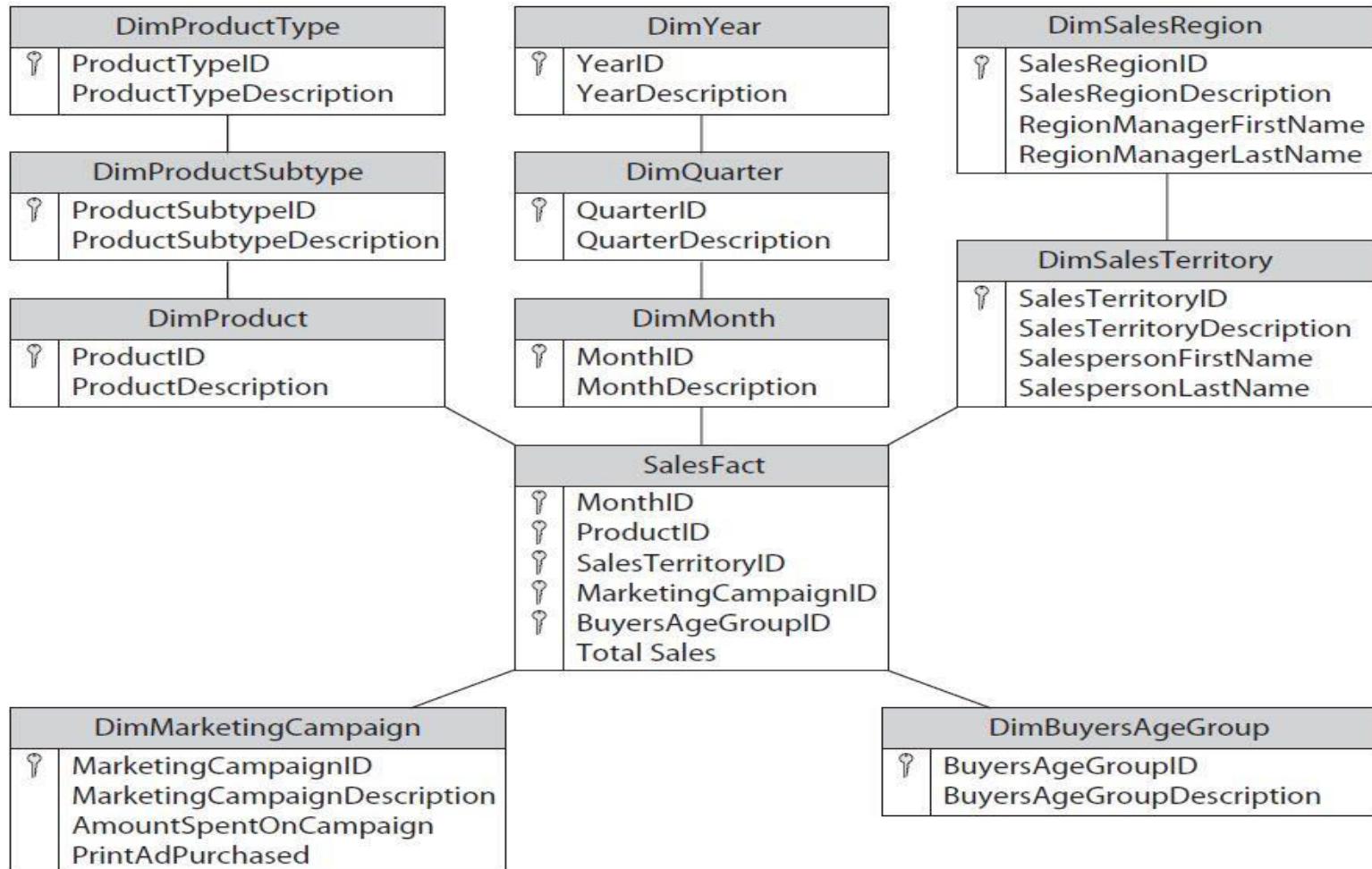
Hierarchies

- fact table will have one row (and one set of measures) for each unique combination of members at lowest level of all hierarchies
- measures for hierarchy levels above lowest level are not stored in data mart
- these measures have to be calculated by taking an aggregate of measures stored at lowest level
- for example, if user wants to see total sales for sales region, calculated by aggregating total sales for all of territories within region

Snowflake Schema

- represents hierarchies in a manner that is more familiar to those working with relational databases
- each level of hierarchy is stored in separate dimension table

Snowflake Schema



Snowflakes, Stars Analysis Services

- snowflake schema has all advantages of good relational design
 - does not result in duplicate data
- disadvantage of snowflake design is that it requires a number of table joins when aggregating measures at upper levels of hierarchy

Snowflakes, Stars Analysis Services

- in both snowflake and star schemas, we have to calculate aggregates on the fly when user wants to see data at any level above lowest level in each dimension
- in schema with large dimensions or with dimensions that have large number of members, can take significant amount of time

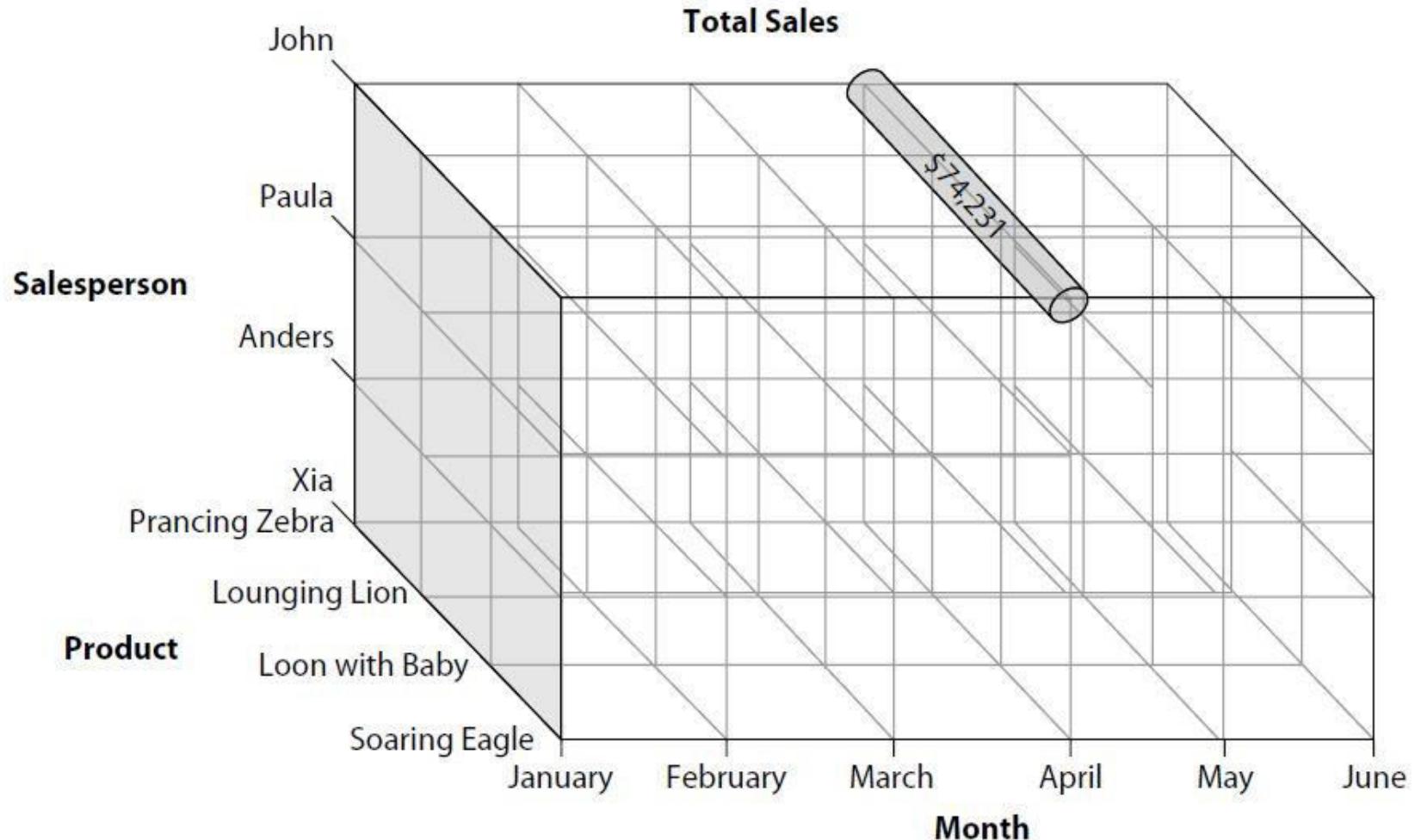
Cube

- structure that contains value for one or more measures for each unique combination of members of all its dimensions
 - detail, or leaf-level, values
- contains aggregated values formed by dimension hierarchies or when one or more of dimensions are left out

Aggregate

- value formed by combining values from given dimension or set of dimensions to create a single value
- often done by adding values together using sum aggregate, but other aggregation calculations can also be used

Sales cube with aggregation for John's total sales for April



Preprocessed Aggregates

- cubes require quite a few aggregate calculations as user navigates through
 - can slow down analysis
- to combat this, some or all of possible data aggregates in cubes are calculated ahead of time and stored within cube itself
- these stored values are known as *preprocessed aggregates*

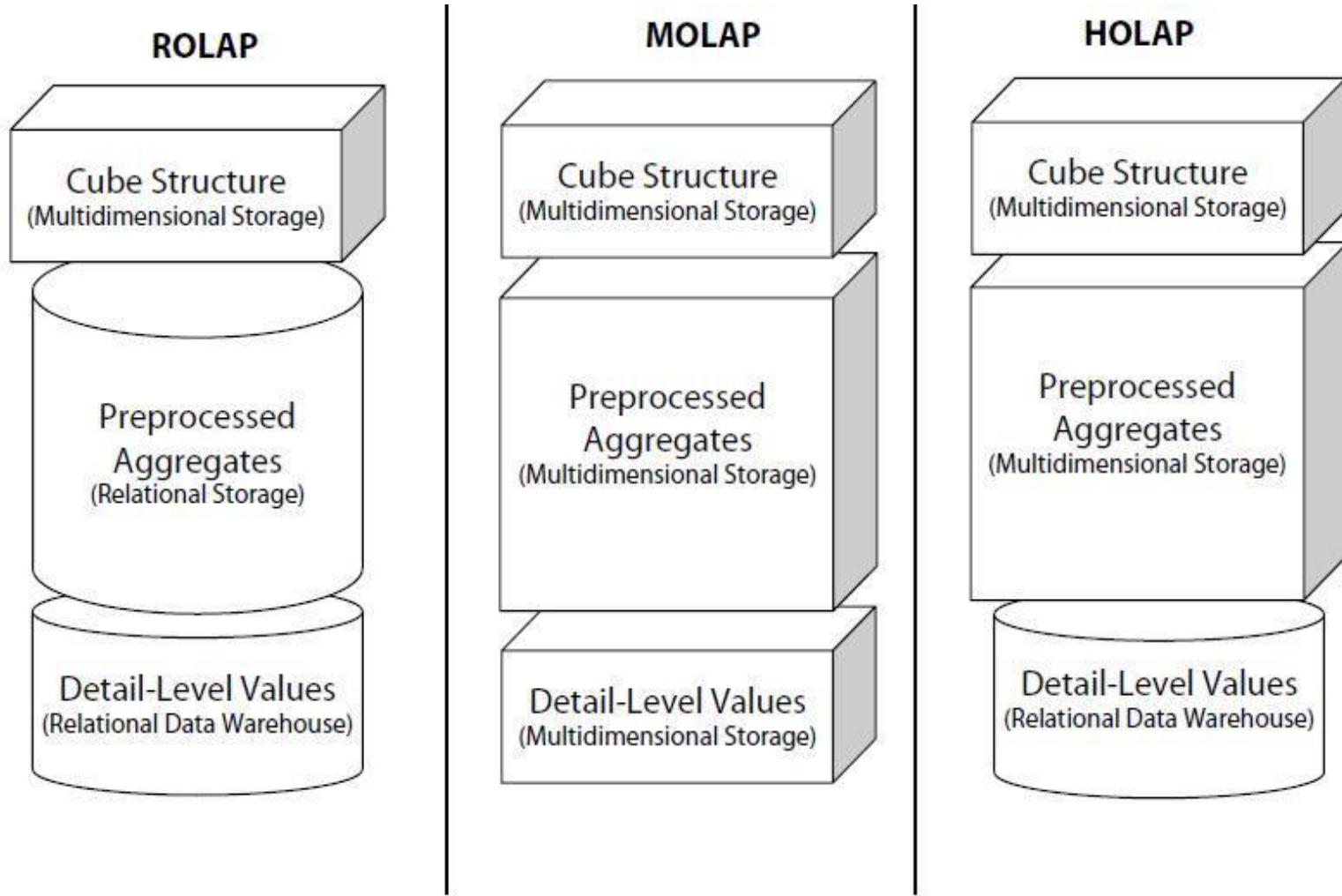
Multi-Dimensional Database

- structured around measures, dimensions, hierarchies, and cubes
 - rather than tables, rows, columns, relations
- natural way to store information used for *business intelligence*
- provides structure for storing pre-processed aggregates

Architecture

- key part of OLAP system is cube and preprocessed aggregates it contains
- OLAP systems typically use one of three different architectures for storing cube data

Architecture



Relational OLAP (ROLAP)

- stores cube structure in multidimensional database
- leaf-level measures are left in relational data mart that serves as source of cube
- preprocessed aggregates are also stored in relational database table
- when decision maker requests value of measure for set of dimension, ROLAP system first checks to determine whether dimension specify aggregate or leaf-level value
 - if aggregate is specified, value is selected from relational table
 - if leaf-level value is specified, value is selected from data mart

Relational OLAP (ROLAP)

- can store larger amounts of data than other OLAP architectures
- leaf-level values returned are always as up-to-date as data mart itself
 - does not add latency to leaf-level data
- disadvantage of ROLAP system is that retrieval of aggregate and leaf-level values is slower than other OLAP architectures

Multidimensional OLAP (MOLAP)

- also stores cube structure in multidimensional database
- both preprocessed aggregate values and copy of leaf-level values are placed in multidimensional database as well
- all data requests answered from multidimensional database, high responsive
- additional time required when loading MOLAP system - all leaf level data copied

Hybrid OLAP (HOLAP)

- Hybrid OLAP combines ROLAP and MOLAP storage

Unified Dimensional Model (UDM)

- Microsoft introduced new technology called *Unified Dimensional Model (UDM)*
- designed to provide all benefits of an OLAP system with multidimensional storage and preprocessed aggregates
- avoids number of drawbacks of more traditional OLAP systems

Unified Dimensional Model

- structure that sits over the top of data mart and looks exactly like an OLAP system to the end user - it does not require a data mart
- can build UDM over one or more OLTP systems
- can even mix data mart and OLTP system data in same UDM
- can even include data from databases from other vendors and Extensible Markup Language (XML)-formatted data

Unified Dimensional Model

- can define measures, dimensions, hierarchies, and cubes, either from star and snowflake schemas, or directly from relational database tables
- latter capability enables us to provide business intelligence without first having to build data mart

Data Sources

- UDM begins with one or more *data sources*
- stores information necessary to connect to database that provides data to UDM
- includes server name, database name, logon credentials, ...
- OLE DB providers are available for accessing different databases

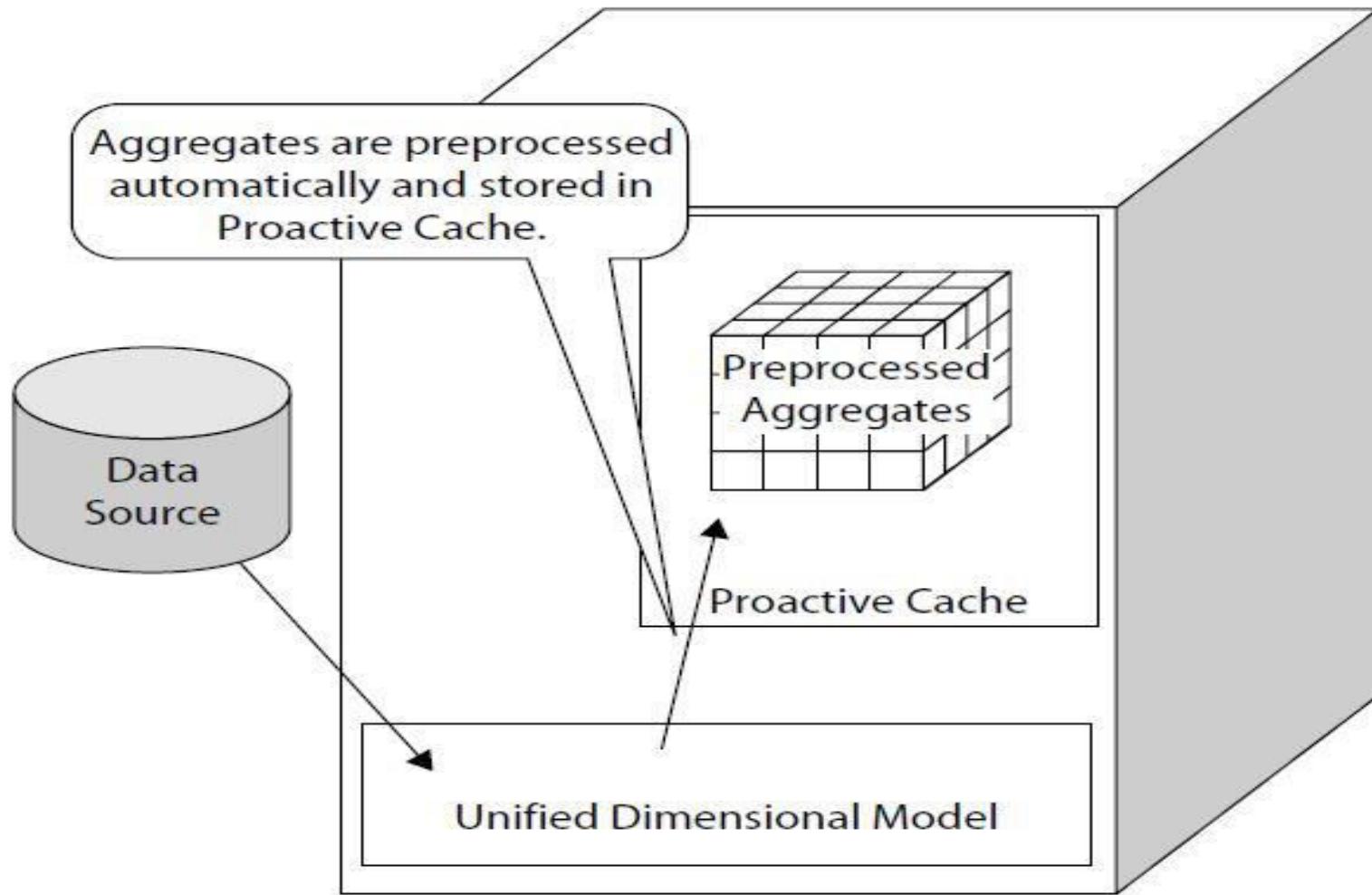
Data Views

- used to determine which tables and fields are utilized
- combine tables and fields from number of different data sources
- this multiple data source capability of data views is what puts “*Unified*” in *Unified Dimensional Model*

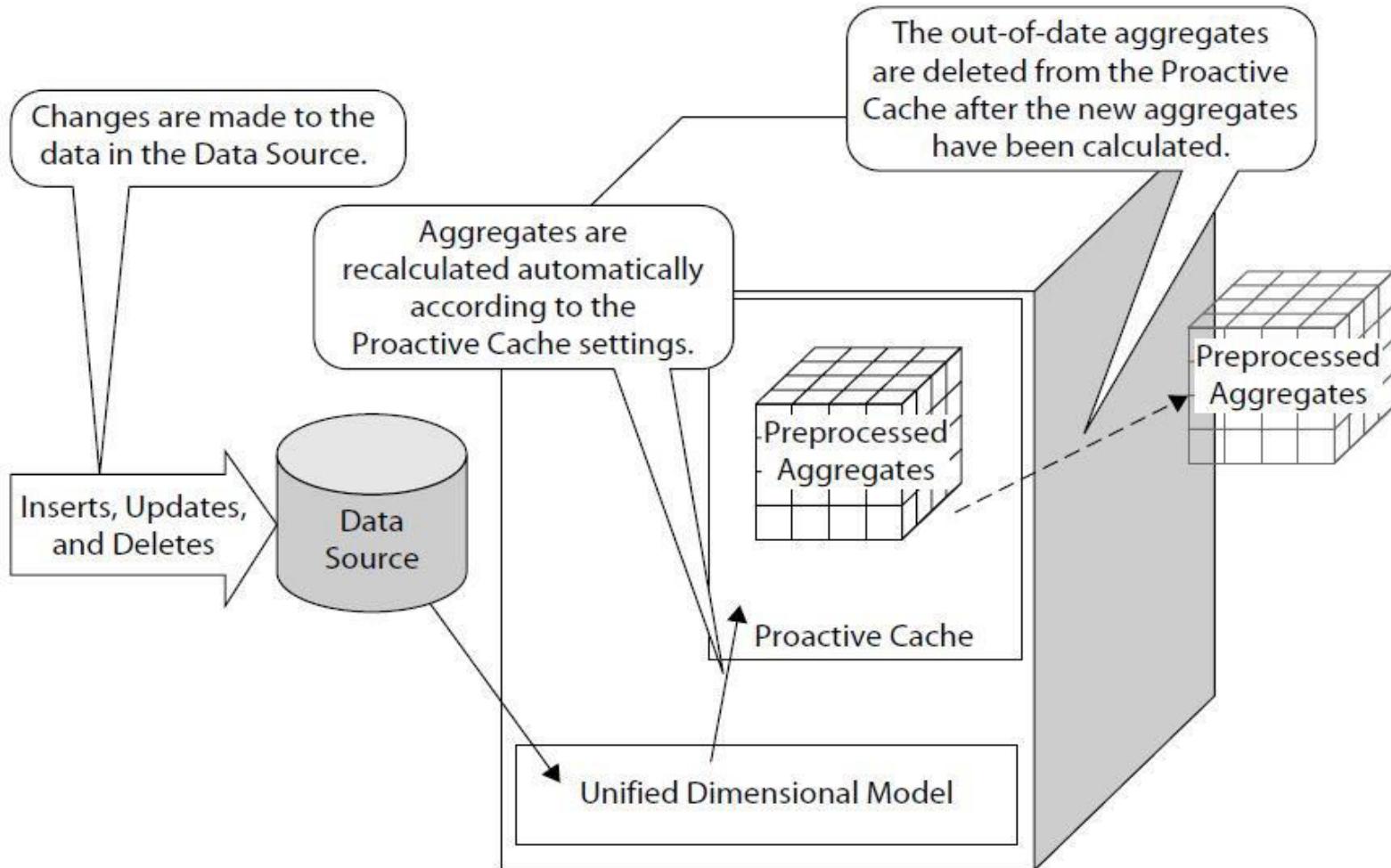
Proactive Caching

- to obtain same performance benefits of traditional OLAP systems, UDM uses *preprocessed aggregates*
- to facilitate high availability, these preprocessed aggregates are stored in *proactive cache*

Proactive Caching



Proactive cache deleted re-created in response to changes in data



XML Definitions

- definitions of all objects in UDM are stored as XML files
- act as source code for UDM

UDM Advantages

- OLAP built on transactional data
- extremely low latency
- ease of creation and maintenance

Design

- what are decision makers needs (analysis)
- what facts, figures, statistics, ... you need for effective decision making (measures, facts)
- how should this information be sliced and diced for analysis (dimensions)
- what additional information can aid in finding exactly what is needed (attributes)

Extract Transform Load (ETL)

Integration Services

Extract Transform Load (ETL)

- decision makers - data marts – must contain useful data - *Integration Services*
- enables us to change location of data by copying it from (OLTP) databases
- as it is being copied, we can also transform data from format required by OLTP systems to format required by OLAP systems

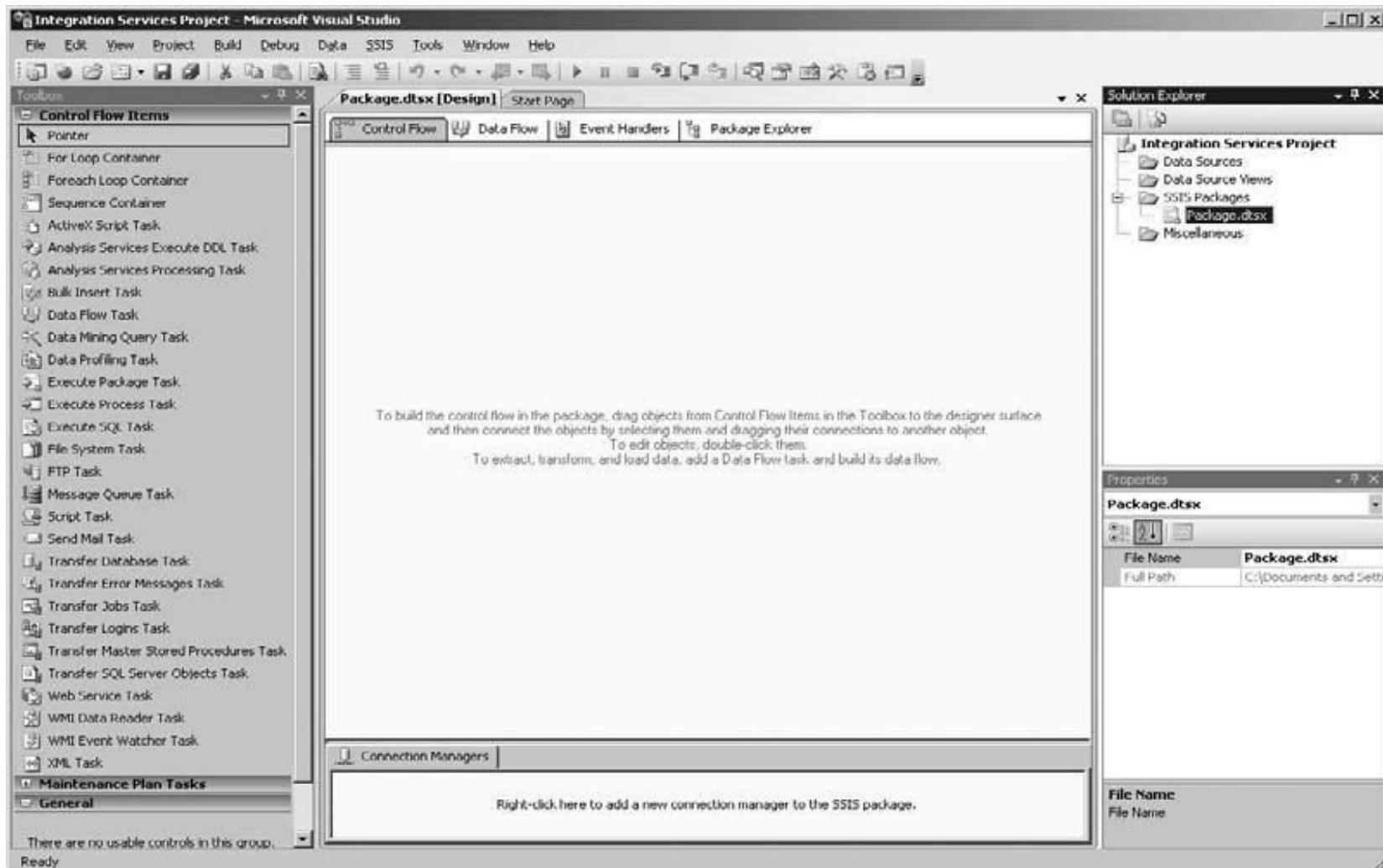
Integration Services

- create structures called packages, which are used to move data between systems
- packages contain data sources and data destinations
- Microsoft's approach to data transfer
- easy-to-use, flexible, capable, highly scalable Extract, Transform, Load ETL tool
- uses a drag-and-drop development style

Integration Services Package

- operation is defined by control flow
 - sequence of tasks that will be performed
- Integration Services packages are created using Integration Services project in Business Intelligence Development Studio

Integration Services Project

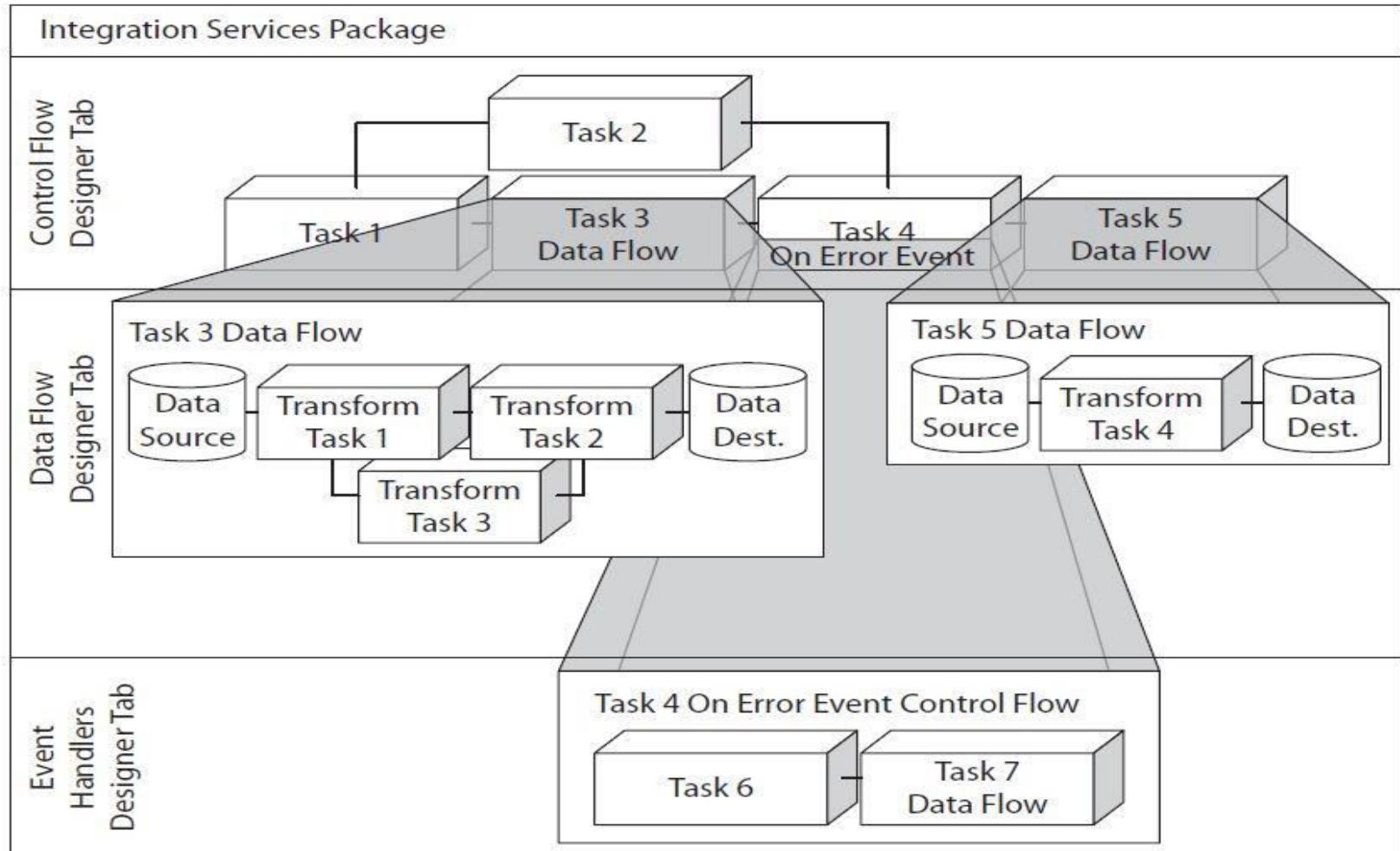


Integration Services Project design layout

- Designer Tab
 - Control Flow
 - Data flow
 - Event Handlers
 - Package Explorer
- Connections tray (special area at bottom of Designer window)
 - for defining Connection Managers

- event handler defined as control flow
- (however, event handler control flows are created on Event Handlers Designer tab rather than on Control Flow Designer tab)
- control flow
 - tasks
 - data source, transformation, data destination

Integration Services package structure



Control Flow Items

- For Loop Container, Foreach Loop Container
- Bulk Insert Task
- **Data Flow Task**
- Execute Package Task, Message Queue Task
- Send Mail Task
- Maintenance Plan Tasks, Back Up Database Task
- Execute SQL Server Agent Job Task
- Execute T-SQL Statement Task

Data Flow Task

- Toolbox is divided into three areas
 - data flow sources
 - data flow transformations
 - data flow destinations

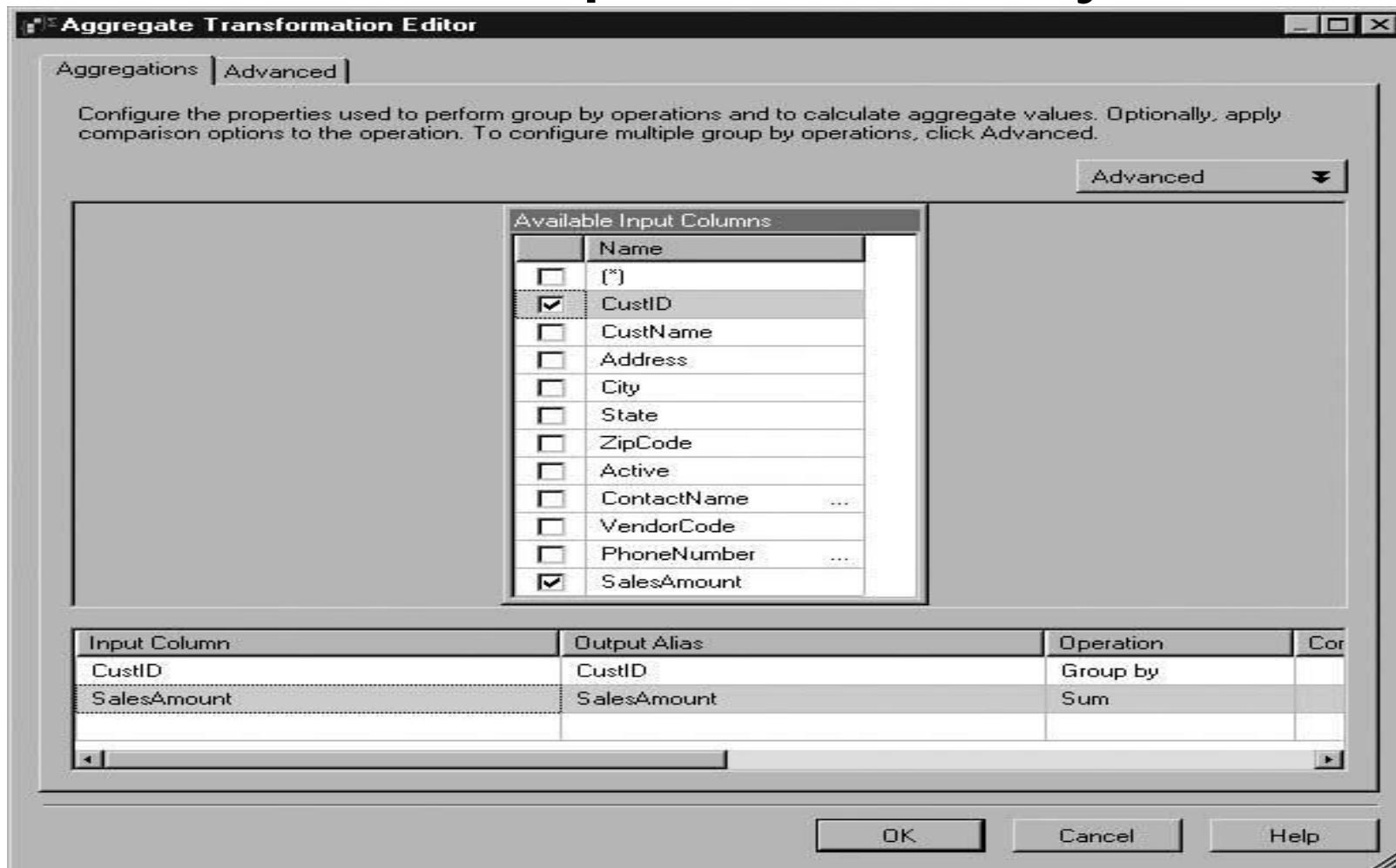
Data Flow Sources, Destinations

- ADO.NET Source
- Excel Source
- Flat File Source
- **OLE DB Source**
- Raw File Source
- XML Source

Data Flow Transformations - Aggregate

- select columns to participate in aggregation from Available Input Columns
- select Operation that is applied to each of selected columns
- column can either be used as group by or it can be aggregated
- Average
- Count; Count distinct
- Maximum; Minimum
- Sum

sum of SalesAmount for each customer represented by CustID



Data Flow Transformations - Audit

- lets us add columns that contain information about package execution to data flow
- audit columns can be stored in data destination and used to determine when package was run, where it was run, ...

Data Flow Transformations - Character Map

- enables us to modify contents of character-based columns
- modified column can be placed in place of original column, or it can be added as new column

Data Flow Transformations - Conditional Split

- enables us to split data flow into multiple outputs

Data Flow Transformations - Copy Column

- lets us create new columns in data flow that are copies of existing columns

Data Flow Transformations - Data Conversion

- enables us to convert columns from one data type to another
- can either replace existing column or be added as new column

Data Flow Transformations - Derived Column

- enables us to create value derived from an expression
- expression can use values from variables and content of columns in data flow

Data Flow Transformations - Export Column

- lets us take content of text or image column and write it out to file
- pathname and filename are specified in different column in data flow
- content of text or image column can be written to different file for each row

Data Flow Transformations - Import Column

- lets us take content of set of files and insert it into text or image column in data flow
- pathname and filename are specified in different column in data flow
- content of different text or image file can be written to each row in data flow

Data Flow Transformations - Fuzzy Grouping

- enables us to find groups of rows in data flow based on non-exact matches
- often used to find possible duplicate rows based on names, addresses, ...
- for example, Ms. Kathy Jones, Ms. Kathryn Jones, and Ms. Cathy Jones may be three entries for same person
- transformation selects one of rows in group as best candidate for other rows to be combined into
- once groups and their model rows have been identified, we can use another transformation to combine any unique information from non-model rows into model row and delete non-model rows

Data Flow Transformations - Lookup

- works similarly to Fuzzy Lookup transformation
- difference is the Lookup transformation requires exact matches, rather than using similarity scores

Data Flow Transformations - Merge

- merges two data flows together
- to work properly, both input data flows must be sorted using same sort order

Data Flow Transformations – Merge Join

- enables us to merge two data flows together by executing an inner join, left outer join, or full outer join
- as with Merge transformation, both of input data flows must be sorted

Data Flow Transformations – Multicast

- lets us take single data flow and use it as input to several data flow transformations or data flow destinations

Data Flow Transformations – OLE DB Command

- enables us to execute SQL statement for each row in data flow
- question marks can be used to create parameterized query

Data Flow Transformations – Pivot

- enables us to take normalized data and change it into less normalized structure
- taking content of one or more columns in input data flow and using it as column names in output data flow
- data in these newly created columns is calculated by taking an aggregate of contents of another column from input data flow
 - number of rows from input may define single row in output

Data Flow Transformations – Unpivot

- enables us to take a denormalized data flow and turn it into normalized data

Data Flow Transformations – Row Count

- lets us determine number of rows in data flow
- count is then stored in package variable, can then be used in expressions to modify control flow or data flow in package

Data Flow Transformations – Script

- lets us create .NET code for execution as part of our data flow
- it can be used as data source, data destination, or data transformation

Data Flow Transformations – Slowly Changing Dimension

- enables us to use data flow to update information in slowly changing dimension of data mart

Data Flow Transformations – Sort

- enables us to sort rows in data flow

Data Flow Transformations – Term Extraction

- lets us extract list of words and phrases from column containing free form text
- identifies recurring nouns phrases in text, along with score showing frequency of occurrence for each word or phrase
- information can then be used to help discover content of unstructured, textual data

Data Flow Transformations – Term Lookup

- functions almost identically to Term Extraction transformation
- difference is Term Lookup starts with table of terms to look for in free form text,
 - whereas Term Extraction creates its own list on-the-fly

Data Flow Transformations – Union

- let us merge several data flows into single data flow
- any number of data flows can be unioned together
- only limitation is they all must be able to contribute fields to all of columns defined in output

Procedural Extensions Structured Query Language

Course Notes SQL Procedural Extension

Sample DataBases

ADVENTURE WORKS

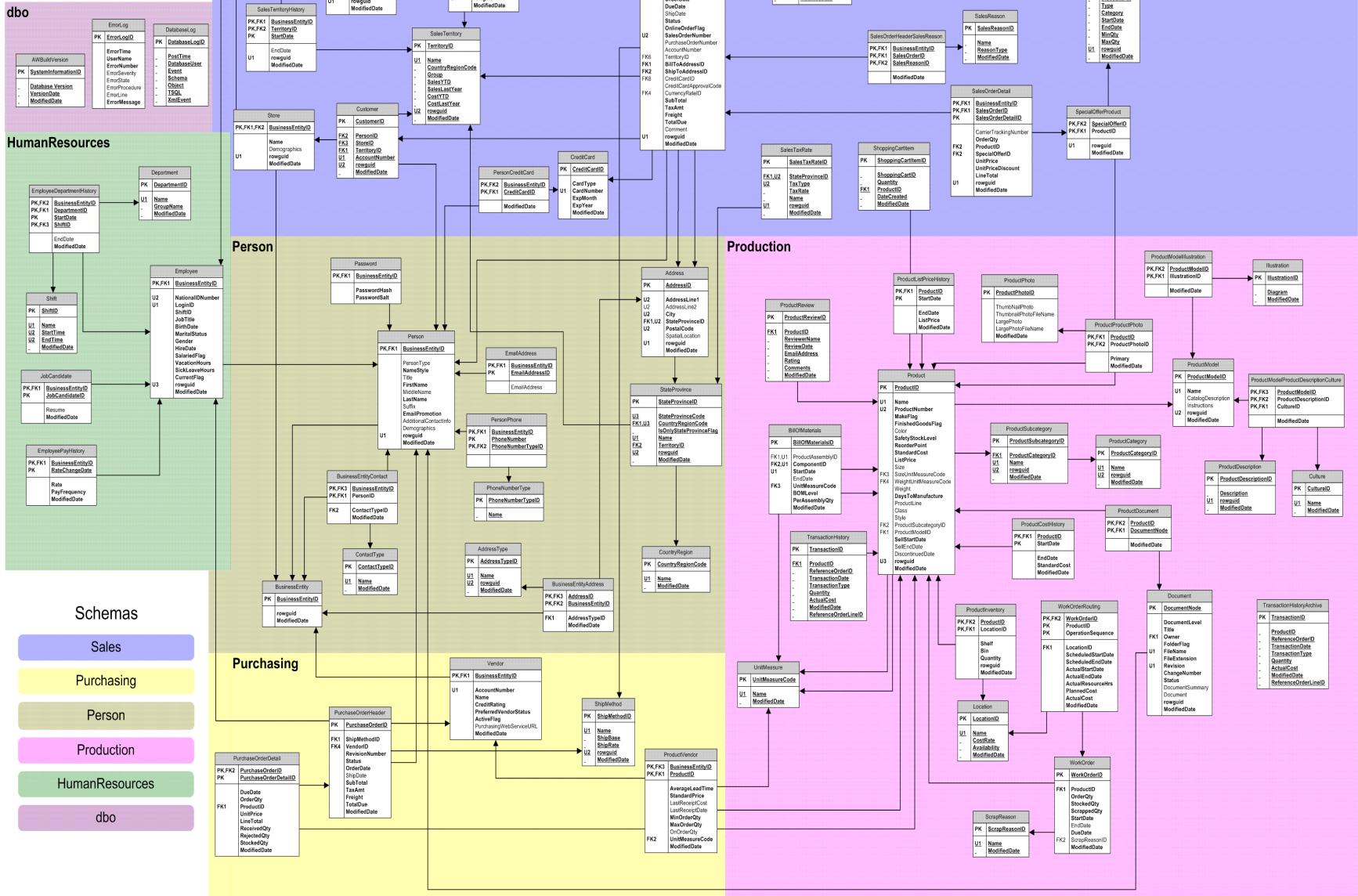
AdventureWorks OLTP

- database supports standard online transaction processing scenarios for fictitious bicycle manufacturer (Adventure Works Cycles)
- scenarios include Manufacturing, Sales, Purchasing, Product Management, Contact Management, and Human Resources

AdventureWorks 2008 OLTP Schema

Best Print Results if:
11X17 paper
Landscape
Fit to 1 sheet

Samples and Sample Databases at <http://CodePlex.com/SqlServerSamples>



Adventure Works Cycles

- large manufacturing company
- manufactures and sells metal and composite bicycles to North American, European and Asian commercial markets
- base operation is located in Bothell, Washington with 290 employees
- several regional sales teams are located throughout their market base

Sales and Marketing

- two types of customers:
- Individuals - consumers who buy products from Adventure Works Cycles online store
- Stores - retail or wholesale stores that buy products for resale from Adventure Works Cycles sales representatives
- Customer table contains one record for each customer
- column CustomerType indicates whether customer is an individual consumer (CustomerType= 'I') or store (CustomerType= 'S')
- data specific to these customer types is maintained in Individual and Store tables, respectively

Individual

- major tables: Person.Contact, Sales.Customer, Sales.Individual, Sales.SalesOrderHeader, Sales.SalesOrderDetail
- sales and demographic data have been trended for data mining scenarios
- demographic data (income, hobbies, number of cars, and so on) is stored as xml data in Demographics column of Individual table

Store

- major tables: Person.Contact, Sales.Customer, Sales.Store, Sales.StoreContact, Sales.SalesOrderHeader, Sales.SalesOrderDetail
- stores are categorized by size: large, medium, and small
- store contacts are employees of store who interact with sales representatives

Production

- bicycles that are manufactured
- bicycle components that are replacement parts, such as wheels, pedals, or brake assemblies
- bicycle apparel that is purchased from vendors for resale to customers
- bicycle accessories that are purchased from vendors for resale to customers

Production

- Production.Culture - languages used in localized product descriptions
- Production.Location - list of locations where products and parts are stored as inventory
- ProductModelProductDescriptionCulture - cross-reference between product models, product descriptions, and languages description has been localized to
- Production.ProductDescription - full description of products in various languages

Production

- Production.Product - about each product sold or used to manufacture bicycles and bicycle components
 - FinishedGoodsFlag column indicates whether product is sold
 - products that are not sold are components of product that is sold
- Production.BillOfMaterials - list of all components used to manufacture bicycles and bicycle subassemblies
 - ProductAssemblyID column represents parent, or primary, product, and ComponentID represents child, or individual, parts used to build parent assembly

Production

- Production.ProductCategory - most general classification of products
- Production.ProductModel - product models associated
 - CatalogDescription column contains additional product information by using xml data type
 - Instructions column contains product manufacturing instructions by using xml data type
- Production.ProductSubcategory - subcategories of product categories

Production

- Production.ProductInventory - inventory level of products by their location
- Production.ProductListPriceHistory - list price of products over time
- Production.ProductCostHistory - cost of products over time
- Production.ProductPhoto - images of products; stored by using varbinary(max) data type
- Production.ProductReview - customer reviews of products

Manufacture

- Manufacturing processes:
 - Bill of materials - lists products that are used or contained in another product
 - Work orders - manufacturing orders by work center
 - Locations - defines major manufacturing and inventory areas
 - Manufacturing and product assembly instructions by work center
- Product inventory - physical location of product in warehouse or manufacturing area, and quantity available
- Engineering documentation - technical specifications and maintenance documentation for bicycles or bicycle components

Manufacture

- Production.BillOfMaterials - list of all components used to manufacture bicycles and bicycle subassemblies
- there is an intrinsic recursive relationship in bill of material structure that indicates relationship between parent product and components that make up that product
 - ProductAssemblyID column represents parent, or primary, product
 - ComponentID represents child, or individual, parts used to build parent assembly
 - BOM_Level column indicates level of ComponentID relative to ProductAssemblyID

Manufacture

- Production.Document - engineering specifications and other technical documentation
- Production.Illustration - bicycle manufacturing illustrations
- Production.Location - list of inventory and manufacturing areas in which products and parts are stored as inventory or built
- Production.WorkOrder - defines products and quantity that must be manufactured to meet current and forecasted sales
- Production.WorkOrderRouting - details for each work order; includes sequence of work centers product travels through in manufacturing or assembly process

Manufacture

- Production.ScrapReason - list of common reasons why bicycles or bicycles parts are rejected during manufacturing process
- WorkOrderRouting tracks quantity scrapped and reason for scrapping by product
 - depending on severity of problem, product must be fixed or replaced before product can move to next work center

Purchase and Vendor

- purchasing department buys raw materials and parts used in manufacture of bicycles
- also purchases products for resale, such as bicycle apparel and bicycle add-ons
- information about these products and vendors from whom they are obtained is stored

Purchase and Vendor

- Person.Address - customers may have more than one address
- associative table VendorAddress maps vendors to their addresses
- also contains address information for employees and customers
- Person.Contact - names of vendor employees; vendor may have more than one contact
- associative table VendorContact maps contacts to vendors
- Production.ProductVendor - maps vendors to products they supply; product may be supplied by more than one vendor, and vendor may supply more than one product

Purchase and Vendor

- Purchasing.PurchaseOrderDetail - details of purchase order, such as products ordered, quantity, and unit price
- Purchasing.PurchaseOrderHeader - purchase order summary information, such as total due, order date, and order status
 - create a master-detail relationship
- Purchasing.ShipMethod - lookup table that is used to maintain standard ways of shipping products
- Purchasing.Vendor - details about vendors, such as the vendor name and account number
- Purchasing.VendorAddress - links customers to address information
- Purchasing.VendorContact - address information for all customers

Business Rules **ARCHITECTURE**

Business Logic

- *Business logic*, sometimes called *business rules*, is code in your application that forces users to follow processes that makes your business work well
- rules can be process flows or data formatting
- provide definition of how and what type of data is stored in tables of your database

Example of business logic

- formatting of phone number; all phone numbers stored in database look the same
- forces your users to create consistency
- if you don't have consistency, you could end up with phone numbers that look like:
 - 9995551212
 - 999-555-1212
 - (999)555-1212
 - 999 555 1212

Example of business logic

- inconsistency is not a big deal in most instances
- as long as users are collecting all data you need, this rule is just formatting
- business logic is more than formatting, however

True example of business logic

- company in business of supplying hard disk drives
- agreement with computer manufacturer at 15% discount
- person taking the order would look up discount on a sheet of paper and manually put discount in order
- manufacturer called to request more hard drives and instead of the 15% reduction, it received only a 1.5%
- mistake was enough to cause manufacturer to find another hard drive supplier
- if hard and fast business rules were defined and adhered to the company would not have lost one of its largest customers

True example of business logic

- one of largest computer manufacturers fined thousands of dollars for shipping computers to wrong address
- United States has export laws that define which countries can receive high-tech equipment
- company shipped systems to country that was not supposed to receive this type of equipment
- if correct business logic had been in place in manufacturer's order-taking system, it would have saved the company a great deal of money

Client / Server

- processing occurred on centralized mainframe, all data & business logic were stored in same place
 - easy make changes to application & business logic
- decentralized computing
 - more processing of business logic moved to smaller and less expensive PCs
 - more expensive central servers used to store and serve up data
- difficult to make changes to business & application logic - rolled out to several places

Internet and network computers

- computing is moving back centralized
- almost all processing (with exception of some simple data validation) is performed on central computers
- only functions performed by hardware sitting on user's desk are to display information to user and to gather input

Fat Client

- maintaining all business logic in client could be very difficult and time-consuming
- change in fat client results in having to recompile entire client; in order for all users to take advantage of new client, it must be installed / deployed on all computers

Thin Client

- developers began to place all business and application logic in stored procedures that reside in database layer
- enabled developers to make changes whenever they needed to and, as long as modifications to stored procedures did not return any unexpected data, client would continue to work as it had in past

Disadvantages

- lack of ability to grow to enterprise levels
- run into locking and blocking issues as well as general slow response times
- SQL Server is designed as database server, not an application server
- if you ever have to migrate application to another RDBMS, such as Oracle, SQL Server functions and datatypes do not necessarily have an exact correspondence

Multi-tier applications

- Web-enabled applications
- Middle-tier frameworks

Procedural Logic in Structured Query Language

Imperative vs. Declarative

Declarative

- Passenger ask the driver “get me to the airport!”
- Structured Query Language – SQL
- Prolog

Imperative

- Passenger, like GPS, gives the driver turn-by-turn directions to get there
- C, Java, php

T-SQL

- American National Standards Institute (ANSI) approved SQL Structured Query Language standard
- Transact - SQL – MS SQL Server dialect of SQL
- Procedural Language – SQL – Oracle
- ...
- standard SQL + procedural extensions

Stored Procedures

- server-side T-SQL code modules
- intermediate layer, custom server-side *application programming interface (API)* that sits between user applications and data stored in database

Functions

- *User-Defined Functions* (UDFs) can perform queries and calculations, and return either scalar values or tabular result sets
- restrictions: can't use certain nondeterministic system functions, can't perform DML or DDL statements, so they can't make modifications to database structure or content, can't perform dynamic SQL queries or change state of the database (cause side effects)

Triggers

- procedural code that is automatically executed in response to certain events on particular table or view in database
- stored procedure, function with implicit call (not explicit call)

past

- working with database; use SQL Server
- retrieve data from database
- adding, updating, and deleting data
- design database

future

- learn the skills for working with database features like views, scripts, stored procedures, functions, triggers, and transactions
- learn skills for working with database security, XML, BLOBs, and CLR interaction
- features that give database much of its power
- you'll have powerful set of SQL skills

Views

Views

- view is SELECT statement that's stored with database
- SELECT queries can be complicated, particularly if they use multiple joins, subqueries, or complex functions
- may want to save queries you use regularly - view is stored as part of the database
- it can be used by all users and application programs that have access to database
- to create view, you use CREATE VIEW statement
- think of view as virtual table that consists only of rows and columns specified in CREATE VIEW statement

Views

- tables that are listed in FROM clause are called base tables for view
- since view refers back to base tables, it doesn't store any data itself, and it always reflects most current data in base tables
- query on which it's based is optimized by SQL Server before it's saved in database
- to use view, refer to it from another SQL statement anywhere you would normally use table in any of data manipulation statements: SELECT, INSERT, UPDATE, and DELETE

Benefits of using Views

- Design independence
- Data security
- Flexibility
- Simplified queries
- Updatability

Benefits of using Views

- Design independence
 - data that's accessed through view is independent of underlying database structure; can change design of database and then modify view as necessary so that queries that use it don't need to be changed
- Data security
 - create views that provide access only to data that specific users are allowed to see

Benefits of using Views

- Flexibility
 - create custom views to accommodate different needs.
- Simplified queries
 - hide complexity of retrieval operations; then, data can be retrieved using simple SELECT statements
- Updatability
 - with certain restrictions, view can be used to update, insert, and delete data from base table

Benefits of using Views

- **data that you access through view isn't dependent on structure of database**
- suppose view refers to table that you've decided to divide into two tables; to accommodate this change, you simply modify view; you don't have to modify any statements that refer to view
- means that users who query database using view don't have to be aware of change in (physical) database structure, and application programs that use view don't have to be modified

Benefits of using Views

- **can use views to restrict access to database**
- include just columns and rows you want a user or application program to have access to in views
- then let user or program access data only through views; for example, restricts access to table that contains information on investors:
 - view provides access to name and address information that might be needed by support staff that maintains table
 - another view that includes investment information could be used by consultants who manage investments

Benefits of using Views

- Views are flexible
- used to provide just data that's needed for specific situations
- Views can hide complexity of SELECT statement
- makes it easier for end users and application programs to retrieve data they need
- Views can be used not only to retrieve data, but to modify data as well

Views can be nested

- because SELECT statement can refer to view, SELECT statement you code within definition of view can also refer to another view
- recommend you avoid using nested views
 - dependencies between tables and views can become confusing, which can make problems difficult to locate
- SQL Server views can be nested up to 32 levels deep

- SELECT statement for view can use any of features of normal SELECT statement with two exceptions
- it can't include an ORDER BY clause unless it also uses TOP clause or OFFSET and FETCH clauses
 - if you want to sort result set that's extracted from view, you have to include ORDER BY clause in SELECT statement that refers to view
- it can't include INTO keyword

- by default, columns in view are given same names as columns in base tables; if view contains calculated column you'll want to name that column just as you do in other SELECT statements
- you'll need to rename columns from different tables that have same name

CREATE VIEW statement provides optional clauses

- WITH ENCRYPTION clause prevents other users from examining SELECT statement on which view is based
- WITH SCHEMABINDING clause protects view by binding it to database structure, or schema
 - prevents underlying base tables from being deleted or modified in any way that affects view
 - typically use this option for production but not for testing
- WITH CHECK OPTION clause prevents row in view from being updated if that would cause row to be excluded from view

syntax of CREATE VIEW statement

- CREATE VIEW view_name [(column_name_1 [, column_name_2]...)]
- [WITH {ENCRYPTION | SCHEMABINDING | ENCRYPTION ,SCHEMABINDING}]
- AS
- select_statement [WITH CHECK OPTION]

Updatable View

- select list
 - can't include DISTINCT or TOP clause
 - can't include aggregate functions
 - can't include calculated columns
 - can't include GROUP BY or HAVING clause
- two SELECT statements can't be joined by union operation - view can't include UNION operator

Updatable View

- using INSERT, UPDATE, and DELETE statements to update data through view is inflexible and prone to errors
- you should avoid this technique whenever possible
- instead consider using INSTEAD OF triggers to update data through view

Updatable View

- can modify data of an underlying base table through view
 - any modifications must reference columns from only one base table
 - columns being modified in view must directly reference underlying data in table columns
 - columns cannot be derived in any other way

- CREATE VIEW InvoiceCredit AS
- SELECT InvoiceNumber, InvoiceDate, InvoiceTotal,
PaymentTotal, CreditTotal FROM Invoices
- WHERE InvoiceTotal - PaymentTotal - CreditTotal
 $> 0;$
- UPDATE InvoiceCredit
- SET CreditTotal = CreditTotal + 200
- WHERE InvoiceTotal - PaymentTotal - CreditTotal
 $\geq 200;$

- INSERT statement that fails due to columns with null values
- INSERT INTO InvoiceCredit (InvoiceNumber, InvoiceDate, InvoiceTotal)
- VALUES (1RA2 39 881, '2016-05-04', 417.34);
- cannot insert value NULL into column ... does not allow nulls

- can use View Designer to create or modify view
- tool is similar to Query Designer

How to code scripts

- most of scripts you've created so far have consisted of single SQL statement
- script can include any number of statements, and those statements can be divided into one or more batches
- to indicate end of a batch, you code GO command
- because new database must exist before you can add tables to it, CREATE DATABASE statement must be coded in separate batch that's executed before CREATE TABLE statements ... before INSERT

SQL script

- series of SQL statements that you can store in file
 - each script can contain one or more batches that are executed as a unit
- to signal end of batch, use GO command
 - GO command isn't required after last batch in script or for script that contains single
- if statement must be executed before statements that follow can succeed, include GO command after it
- statements within batch are executed in order that they appear; need to code statements that depend on other statements after statements they depend on

SQL script

- execute batch that contains CREATE DATABASE statement before you can execute other statements that refer to database
- statements CREATE VIEW, CREATE PROCEDURE, CREATE FUNCTION, CREATE TRIGGER, and CREATE SCHEMA can't be combined with other statements in batch
- include documentation

- `/*`
- Creates tables in a database ...
- Author: Me
- Created: 2017-03-21
- Modified: 2017-03-23
- `*/`

Statement Block

- BEGIN ... END defines statement block

PRINT

- use PRINT statement to return message to client
- PRINT statement is simple and useful server-side debugging tool; printing constants and variable values to standard output during script execution
- doesn't work in functions because of built-in restrictions on functions causing side effects

Control Flow of Execution

- IF...ELSE - controls flow of execution based on condition
- WHILE - repeats statements while specific condition is true
- BREAK exits innermost loop
- CONTINUE - returns to beginning of loop
- TRY ... CATCH - controls flow of execution when an error occurs
- GOTO - unconditionally changes flow of execution
- RETURN - exits unconditionally

Statements for script processing

- DECLARE - declares local variable
- SET - sets value of local variable or session variable
- EXEC - executes dynamic SQL statement or stored procedure

Scalar Variables

- scalar variables, which can contain a single value
- also known as local variables - scope is limited to single batch
- DECLARE statement to create variable and specify type of data it can contain
- SET statement to assign value to variable
- can also assign value to variable within select list of SELECT statement

Scalar Variables

- used to store data, defined with standard data type and contains single value
- initial value of variable is always null
- can also create table variables to store an entire result set
- name of a variable must always start with an at sign (@)
- scope of variable is batch in which it's defined
- can use variable in any expression, but you can't use it in place of object name or keyword

syntax

- `DECLARE @variable_name_1 data_type [,
@variable_name_2 data_type]...`
- `SET @variable_name = expression`
- `SELECT @variable_name_1 =
column_specification_1`
- `[, @variable_name_2 = column_specificat
ion_2] ...`

- `DECLARE @MaxInvoice money, @MinInvoice money;`
- `DECLARE @PercentDifference decimal(8,2);`
- `DECLARE @InvoiceCount int, @VendorIDVar int;`
- `SET @VendorIDVar = 95;`
- `SET @MaxInvoice = (SELECT MAX(InvoiceTotal) FROM Invoices WHERE VendorID = @VendorIDVar);`
- `SELECT @MinInvoice = MIN(InvoiceTotal),
@InvoiceCount = COUNT(*) FROM Invoices WHERE
VendorID = @VendorIDVar;`
- `SET @PercentDifference = (@MaxInvoice -
@MinInvoice) / @MinInvoice * 100;`

- PRINT 'Maximum invoice is \$' +
CONVERT(varchar,@MaxInvoice,1) + '.'
- PRINT 'Minimum invoice is \$' +
CONVERT(varchar,@MinInvoice,1) + '.'
- PRINT 'Maximum is ' + CONVERT(varchar,@PercentDifference)
+ '% more than minimum.';
- PRINT 'Number of invoices: ' +
CONVERT(varchar,@InvoiceCount) + '.'
 - Maximum invoice is \$46.21.
 - Minimum invoice is \$16.33.
 - Maximum is 182.97% more than minimum.
 - Number of invoices: 6.

Table Variables

- can store contents of entire table or data set
- specify table data type in DECLARE statement
- then define columns and constraints for table using same syntax that you use for CREATE TABLE statement
- used in place of table name in INSERT, UPDATE, DELETE and SELECT statements
- can't use instead of table name in INTO clause of SELECT INTO

Table Variables

- can store an entire result set rather than single value
- like scalar variable, table variable has local scope, so it's available only within batch where it's declared
- can use table variable like standard table

syntax

- `DECLARE @table_name TABLE`
- `(column_name_1 data_type
[column_attributes])`
- `[, column_name_2 data_type
[column_attributes]]...`
- `[, table_attributes])`

- DECLARE @BigVendors TABLE (
- VendorID int, VendorName varchar(50)) ;
- INSERT @BigVendors SELECT VendorID,
 VendorName FROM Vendors WHERE VendorID
 IN
- (SELECT VendorID FROM Invoices WHERE
 InvoiceTotal > 5000);
- SELECT * FROM @BigVendors;

Temporary Tables

- unlike table variable, temporary table exists for duration of database session in which it's created
 - can refer to table from more than one script
- stored in system database named ***tempdb***
- useful for testing queries or for storing data temporarily in complex script
- local temporary table is visible only within current session
- global temporary table is visible to all sessions
- to identify local temporary table prefix name with number sign (#)
- to identify global temporary table prefix name with two number signs (##)

Example Temporary Table

- CREATE TABLE #AllUserTables
- (TableID int IDENTITY, TableName
varchar(128));
- GO
- INSERT #AllUserTables (TableName)
- SELECT name FROM sys.tables

Perform Conditional Processing

- evaluates conditional expression after IF keyword
- If condition is true, statement or block of statements after IF keyword is executed
- otherwise statement or block of statements after ELSE keyword is executed
 - if this keyword is included

syntax of IF statement

- IF Boolean_expression
- {statement|BEGIN...END}
- [ELSE
- {statement|BEGIN...END}]

- `DECLARE @grade int;`
- `SET @grade = ?;`
- `IF @grade = 10`
 - `BEGIN`
 - `PRINT 'Genius (canta la mare ...).';`
 - `PRINT 'Sa curga vinul!';`
 - `END`
- `ELSE IF @grade <= 4`
 - `PRINT 'Fail!';`
- `ELSE IF (@grade > 4 AND @grade < 10)`
 - `PRINT 'True!';`
- `ELSE`
 - `PRINT 'UNKNOWN.';`

- if you need to execute two or more SQL statements within an IF or ELSE clause, enclose them within BEGIN...END block
- can nest IF...ELSE statements within other IF...ELSE statements

Perform Repetitive Processing

- use WHILE coding technique referred to as loop
- repeat statement or block of statements while condition is true; statement is executed as long as conditional expression in WHILE clause is true
- if you need to execute two or more SQL statements within WHILE loop, enclose statements within BEGIN and END keywords
- to exit from WHILE loop immediately without testing expression, use BREAK statement
- to return to beginning of WHILE loop without executing any additional statements in loop, use CONTINUE statement

syntax of WHILE

- WHILE expression
- {statement|BEGIN...END}
- [BREAK]
- [CONTINUE]

Cursor

- by default, SQL statements work with entire result set rather than individual rows
- may sometimes need to work with data in result set one row at a time
- use **cursor**

Syntax of Cursor

1. `DECLARE cursor_name CURSOR FOR select_statement;`
 - Open the cursor
2. `OPEN cursor_name;`
 - get column values from row and store them in series of variables
3. `FETCH NEXT FROM cursor_name INTO @variable1[, @variable2][, @variable3]...;`
 - Close and deallocate cursor
4. `CLOSE cursor_name;`
5. `DEALLOCATE cursor_name;`

- `DECLARE @InvoiceIDVar int, @InvoiceTotalVar money, @UpdateCount int;`
- `SET @UpdateCount = 0;`
- `DECLARE Invoices_Cursor CURSOR FOR`
- `SELECT InvoiceID, InvoiceTotal FROM Invoices WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0;`
- `OPEN Invoices_Cursor;`
- `FETCH NEXT FROM Invoices_Cursor INTO @InvoiceIDVar, @InvoiceTotalVar;`

- WHILE @@FETCH_STATUS <> -1 BEGIN
 - IF @InvoiceTotalVar > 1000 BEGIN
 - UPDATE Invoices SET CreditTotal = CreditTotal + (InvoiceTotal * .1)
 - WHERE InvoiceID = @InvoiceIDVar;
 - SET @UpdateCount = @UpdateCount +1;
 - END;
 - FETCH NEXT FROM Invoices_Cursor INTO @InvoiceIDVar,
@InvoiceTotalVar;
 - END;
- CLOSE Invoices_Cursor; DEALLOCATE Invoices_Cursor;

- @@FETCH_STATUS system function returns
- 0 if row was fetched successfully or
- -1 if row can't be fetched because end of result set has been reached

Handle Errors

- TRY...CATCH statement to handle errors
- works similarly to exception handling statements that are available from other languages
- code TRY block around any statements that might cause an error to be raised
- following TRY block, you must code a single CATCH block

Handle Errors

- TRY block begins with BEGIN TRY keywords and ends with END TRY keywords
- CATCH block begins with BEGIN CATCH keywords and ends with END CATCH keywords
- within CATCH block include statements that handle error that might be raised in TRY block
- although it's common to use CATCH block to display information to user, you can also perform other error handling tasks such as writing information about error to log table or rolling back a transaction
- TRY block must be followed immediately by single CATCH block

syntax of TRY...CATCH

- BEGIN TRY
 - {sql_statement|statement_block}
- END TRY BEGIN CATCH
- {sql_statement|statement_block}
 - functions you can use within CATCH block
- END CATCH

Functions you can use within CATCH block

- `ERROR_NUMBER()` - returns error number
- `ERROR_MESSAGE()` - returns error message
- `ERROR_SEVERITY()` - returns severity of error
- `ERROR_STATE()` - returns state of error

- TRY...CATCH must be coded within single batch, stored procedure, or trigger
- can nest TRY...CATCH statement
- some types of errors aren't handled, errors with low severity (10 or lower) are considered warnings and aren't handled
- errors with high severity often cause database connection to be closed, which prevents them from being handled

- BEGIN TRY
 - INSERT Invoices
 - VALUES (799, 'ZKK-799', '2015-03-27', 299.95, 0, 0, 1, '2015-05-05', NULL);
- PRINT 'SUCCESS: Record was inserted.';
- END TRY BEGIN CATCH
 - PRINT 'FAILURE: Record was not inserted.';
 - PRINT 'Error ' + CONVERT(varchar, ERROR_NUMBER(), 1) + ':' + ERROR_MESSAGE();
- END CATCH;

- INSERT statement that's coded within TRY block provides vendor ID that doesn't exist
- when attempts to execute this statement, foreign key constraint will be violated and error will be raised
- program execution will skip over PRINT statement that follows INSERT statement and jump into CATCH block
- causes the message to be displayed
- if INSERT statement had executed successfully, program execution would have continued by executing PRINT statement immediately following INSERT statement and skipping CATCH block; code would have displayed message indicating that INSERT statement executed successfully

Procedures ... Scripts

- procedural statements can help you manage database and automate tasks
- extend functionality by creating database objects that store program code within database
- three types of programs provide powerful and flexible way to control how a database is used: procedural programming options – scripts
 - 1. stored procedures**
 - 2. user-defined functions**
 - 3. triggers**

dynamic SQL

- {EXEC|EXECUTE} ('SQL_String')
- DECLARE @TableNameVar varchar(128);
- SET @TableNameVar = 'Invoices';
- EXEC ('SELECT * FROM ' + @TableNameVar +
' ;');
- contents of SQL string at execution
- SELECT * FROM Invoices;

dynamic SQL

- define SQL statement as script executes
- EXEC statement executes string that contains SQL statement

scripts

- of the four types of procedural programs, only scripts can contain two or more batches
- scripts can be executed by SQL Server tools such as ssms Sql Server Management Studio and SQLCMD utility
- stored in files outside of database
- tend to be used most often by SQL Server programmers and database administrators

stored procedures, user-defined functions, triggers

- executable database objects - stored within database
- to create these objects use DDL statements
- remain as part of database until they're explicitly dropped - differ by how they're executed
 - stored procedures and user-defined functions can be run from any database connection that can run SQL statement
 - triggers run automatically in response to execution of an action query on specific table
- both stored procedures and user-defined functions can use parameters, but triggers can't

stored procedures, user-defined functions, triggers

- stored procedures give SQL programmer control over who accesses database and how; since some application programmers don't have expertise to write certain types of complex SQL queries, stored procedures can simplify their use of database
- user-defined functions are most often used by SQL programmers within stored procedures and triggers that they write, although they can also be used by application programmers and end users
- triggers are special procedures that execute when action query, such as INSERT, UPDATE, or DELETE statement, is executed; like constraints, you can use triggers to prevent database errors, but triggers give you greater control and flexibility

stored procedures, user-defined functions, triggers

- stored procedures give SQL programmer control over who accesses database and how; since some application programmers don't have expertise to write certain types of complex SQL queries, stored procedures can simplify their use of database

stored procedures, user-defined functions, triggers

- user-defined functions are most often used by SQL programmers within stored procedures and triggers that they write, although they can also be used by application programmers and end users

stored procedures, user-defined functions, triggers

- triggers are special procedures that execute when action query, such as INSERT, UPDATE, or DELETE statement, is executed; like constraints, you can use triggers to prevent database errors, but triggers give you greater control and flexibility

User-Defined Functions

- like functions in other programming languages, T-SQL UDFs provide convenient way for developers to define routines that accept parameters, perform actions based on those parameters, and return data to the caller - come in three flavors:
 1. inline table-valued functions (TVFs)
 2. multi-statement TVFs
 3. scalar functions

- supports ability to create CLR integration UDFs, ...
- call, or invoke, scalar-valued function from within any expression
- invoke table-valued function anywhere you'd refer to table or view
- within specific function return value with data type that's compatible with data type specified by RETURNS clause
- function can include as many statements as are necessary to calculate return value

Scalar Functions

Scalar Function

- accepts zero or more parameters; returns single scalar value as result
- familiar with T-SQL built-in scalar functions
- **CREATE FUNCTION** statement allows you to create custom scalar functions that behave like built-in scalar functions

syntax for creating scalar-valued function

- CREATE FUNCTION [schema_name.]function_name
- ([@parameter_name data_type [= default]] [, ...])
- RETURNS data_type
- [WITH [ENCRYPTION] [, SCHEMABINDING] [, EXECUTE_AS_clause]]
- [AS]
- BEGIN
- [sql_statements]
- RETURN scalar_expression
- END

CREATE FUNCTION

- CREATE FUNCTION dbo.CalculateCircleArea
(@Radius float = 1.0) RETURNS float
- WITH RETURNS NULL ON NULL INPUT AS
- BEGIN
 - RETURN PI() * POWER(@Radius, 2);
- END;

- defines schema and name of function
(dbo.CalculateCircleArea)
- single required parameter, radius of circle
(@Radius)
- @Radius parameter is defined as float type
- parameter is assigned default value of 1.0
- RETURNS keyword specifies data type of result; will return float result

- additional options following WITH keyword
- example uses RETURNS NULL ON NULL INPUT function option for performance improvement - option that returns NULL if any of parameters passed in are NULL
 - won't execute body of function
- AS keyword indicates start of function body which must be enclosed in T-SQL BEGIN and END keywords
- RETURN statement must be last statement before END keyword in every scalar UDF

- `SELECT dbo.CalculateCircleArea(10);`
- `SELECT dbo.CalculateCircleArea(NULL);`
- `SELECT dbo.CalculateCircleArea(2.5);`
- **EXECUTE AS** manages caller security on UDF - can specify
 - CALLER indicates that UDF should run under security context of user calling function (default)
 - SELF indicates that UDF should run under security context of user who created (or altered) function
 - OWNER indicates that UDF should run under security context of owner
 - can specify that UDF should run under security context of specific user

- if you create UDF that accepts no parameters, you still need to include empty parentheses after function name, both when creating and when invoking
- when procedures are assigned default values, you can simply leave parameter off your parameter list completely when calling
- this isn't an option with UDFs - to use UDF default value, you must use DEFAULT keyword when calling

Options

- provide several creation-time options that allow you to improve performance and security
- ENCRYPTION used to store UDF in database in obfuscated format
- SCHEMABINDING indicates that UDF will be bound to database objects referenced in body of function - attempts to change or drop referenced tables and other database objects result in an error
- CALLED ON NULL INPUT option is opposite of RETURNS NULL ON NULL INPUT - is default option

Recursion

- CREATE FUNCTION dbo.CalculateFactorial (@n int = 1)
 - RETURNS decimal(38, 0)
- WITH RETURNS NULL ON NULL INPUT AS
- BEGIN
 - RETURN
 - (CASE
 - WHEN @n <= 0 THEN NULL
 - *WHEN @n > 1 THEN CAST(@n AS float) * dbo.CalculateFactorial (@n - 1)*
 - WHEN @n = 1 THEN 1
 - END);
- END;

- `SELECT dbo.CalculateFactorial(33);`
- causes SQL Server error message similar to:
- Msg 217, Level 16, State 1, Line 1
- Maximum stored procedure, function, trigger,
or view ***nesting level exceeded (limit 32).***

- CREATE FUNCTION dbo.CalculateFactorial (@n int = 1)
RETURNS float
- WITH RETURNS NULL ON NULL INPUT AS
- BEGIN
- DECLARE @result float;
- SET @result = NULL;
- IF @n > 0 BEGIN
 - ...
- END;
- RETURN @result;
- END;

- IF @n > 0 BEGIN
 - SET @result = 1.0;
 - WITH Numbers (num)
 - AS (
 - SELECT 1 UNION ALL SELECT num + 1 FROM Numbers WHERE num < @n)
 - SELECT @result = @result * num FROM Numbers;
- END;

- by default, SQL Server allows up 100 levels of recursion in CTE (can override this with MAXRECURSION option)

Multi-statement Table-Valued Functions

Multi-statement Table-Valued Functions

- similar in style to scalar UDFs, but instead of returning single scalar value, return result as table data type.
- return type following RETURNS keyword is actually table variable declaration, with its structure declared immediately following table variable name
- RETURNS NULL / CALLED ON NULL INPUT options aren't valid
- RETURN statement in body has no values following it
- inside body can use SQL DML statements INSERT, UPDATE, and DELETE to create and manipulate return results

- for example of multi-statement table value function let's create a product pull list for AdventureWorks
- matches sales orders stored in Sales.SalesOrderDetail table against product inventory in Production.ProductInventory table
- effectively creates a list for employees telling them exactly which inventory bin to go to when they need to fill an order

business rules to multi-statement table value function

- in some cases, number of ordered items may be more than are available in one bin; will instruct employee to grab product from multiple bins
- any partial fills from bin will be reported on list
- any substitution work (for example, substituting different-colored item of same model) will be handled by separate business process and won't be allowed on this list
- no zero fills (ordered items for which there is no matching product in inventory) will be reported back on the list

- let's say there are three customers: Calin, Liviu, and Klaus; each of these customers places an order for exactly five of item number 783, Black Mamba 200 42-inch mountain bike
- let's also say that AdventureWorks has six of this particular inventory item in bin 1, shelf A, location 7, and another three of this particular item in bin 2, shelf B, location 10
- business rules will create pull list like following:

- Calin's order: pull five of item 783 from bin 1, shelf A, location 7; mark order as complete fill
- Liviu's order: pull one of item 783 from bin 1, shelf A, location 7; mark order as partial fill
- Liviu's order: pull three of item 783 from bin 2, shelf B, location 10; mark order as partial fill
- in this example, there are only 9 of ordered items in inventory, but 15 total items have been ordered (3 customers multiplied by 5 items each)
- because of this, Klaus's order is zero-filled - no items are pulled from inventory to fill his order

Look Pa, No Cursors!

- developers may see this problem as opportunity to code cursor-based; from procedural point of view, it essentially calls for performing nested loops through AdventureWorks' customer orders and inventory to match them up
- task can be completed in set-based fashion using numbers table

- CREATE TABLE dbo.Numbers (Num int NOT NULL PRIMARY KEY); GO
- -- Fill the numbers table with numbers from 0 to 30,000 WITH NumCTE (Num)
- AS (
- SELECT 0 UNION ALL
- SELECT Num + 1 FROM NumCTE WHERE Num < 30000)
- INSERT INTO dbo.Numbers (Num) SELECT Num FROM NumCTE OPTION (MAXRECURSION 0);

- ***CREATE FUNCTION dbo.GetProductPullList()***
- ***RETURNS @result table (***
- SalesOrderID int NOT NULL, ProductID int NOT NULL,
- LocationID smallint NOT NULL, Shelf nvarchar(10) NOT NULL,
- Bin tinyint NOT NULL, QuantityInBin smallint NOT NULL,
- QuantityOnOrder smallint NOT NULL, QuantityToPull smallint NOT NULL, PartialFillFlag nchar(1) NOT NULL,
- PRIMARY KEY (SalesOrderID, ProductID, LocationID, Shelf, Bin))
- AS
- BEGIN
- ***INSERT INTO @result (***

- ***INSERT INTO @result (***
- SalesOrderID, ProductID, LocationID, Shelf, Bin,
- QuantityInBin, QuantityOnOrder, QuantityToPull, PartialFillFlag)
- ***SELECT***
- Order_Details.SalesOrderID, Order_Details.ProductID,
- Inventory_Details.LocationID, Inventory_Details.Shelf,
- Inventory_Details.Bin, Inventory_Details.Quantity,
- Order_Details.OrderQty, COUNT(*) AS PullQty,
- CASE WHEN COUNT(*) < Order_Details.OrderQty THEN N'Y'
- ELSE N'N' END AS PartialFillFlag
- FROM
- (

- FROM (
- SELECT ROW_NUMBER() OVER (
- PARTITION BY p.ProductID ORDER BY p.ProductID, p.LocationID, p.Shelf, p.Bin) AS Num, p.ProductID, p.LocationID, p.Shelf, p.Bin, p.Quantity
- FROM Production.ProductInventory p INNER JOIN dbo.Numbers n ON n.Num BETWEEN 1 AND Quantity) Inventory_Details INNER JOIN (
- SELECT ROW_NUMBER() OVER (
- PARTITION BY o.ProductID ORDER BY o.ProductID, o.SalesOrderID) AS Num, o.ProductID, o.SalesOrderID, o.OrderQty
- FROM Sales.SalesOrderDetail o INNER JOIN dbo.Numbers n ON n.Num BETWEEN 1 AND o.OrderQty) Order_Details
- ON Inventory_Details.ProductID = Order_Details.ProductID AND Inventory_Details.Num = Order_Details.Num GROUP BY Order_Details.SalesOrderID, Order_Details.ProductID, Inventory_Details.LocationID, Inventory_Details.Shelf, Inventory_Details.Bin, Inventory_Details.Quantity, Order_Details.OrderQty;
- ***RETURN;***

- when function ends, @result table variable is returned to caller

- source for SELECT query is composed of two subqueries joined together
- first subquery, aliased as InventoryDetails, returns single row for every item in inventory with information identifying precise location where inventory item can be found:
- `SELECT ROW_NUMBER() OVER (`
- `PARTITION BY p.ProductID ORDER BY p.ProductID,`
`p.LocationID, p.Shelf, p.Bin) AS Num, p.ProductID, p.`
`LocationID, p.Shelf, p.Bin, p.Quantity`
- `FROM Production.ProductInventory p`
- `INNER JOIN dbo.Numbers n`
- `ON n.Num BETWEEN 1 AND Quantity) Inventory_Details`

- InventoryDetails subquery is inner-joined to second subquery, identified as Order_Details:
- SELECT ROW_NUMBER() OVER (
- PARTITION BY o.ProductID ORDER BY o.ProductID, o.SalesOrderID) AS Num, o.ProductID, o.SalesOrderID, o.OrderQty FROM Sales.SalesOrderDetail o
- INNER JOIN dbo.Numbers n
- ON n.Num BETWEEN 1 AND o.OrderQty
-) Order_Details
- subquery breaks up quantities of items in all order details into individual rows

- rows of both subqueries are joined based on their ProductID numbers and unique row numbers assigned to each row of each subquery
- effectively assigns one item from inventory to fill exactly one item in each order

- SELECT statement also requires GROUP BY to aggregate total number of items to be pulled from each bin to fill each order detail, as opposed to returning raw inventory-to-order detail items on one-to-one basis:
- GROUP BY Order_Details.SalesOrderID,
Order_Details.ProductID,
Inventory_Details.LocationID,
Inventory_Details.Shelf, Inventory_Details.Bin,
Inventory_Details.Quantity, Order_Details.OrderQty;

- SELECT
- p.Name AS ProductName, p.ProductNumber,
p.Color, ppl.SalesOrderID, ppl.ProductID,
ppl.LocationID, ppl.Shelf, ppl.Bin,
ppl.QuantityInBin, ppl.QuantityOnOrder,
ppl.QuantityToPull, ppl.PartialFillFlag
- FROM Production.Product p INNER JOIN
dbo.GetProductPullList() ppl ON p.ProductID =
ppl.ProductID;

Inline Table-Valued Functions

Inline Table-Valued Functions

- similar to multi-statement table valued functions return a tabular rowset result.
- consists of only a single ***SELECT*** query - inline table value function is literally expanded by query optimizer as part of SELECT statement that contains it, much like view
- sometimes referred to ***parameterized views***

Inline Table-Valued Functions

- performs function commonly implemented by developers which determines that stored procedure requires that large or variable number of parameters be passed in to accomplish particular goal; ideal situation would be to pass array as parameter
- T-SQL doesn't provide array data type: can split list of values into table to simulate an array: gives you **flexibility of *array*** that you can use in SQL joins
- stored procedures allows table-valued parameters

Inline Table-Valued Functions

- declaration must simply state that result is table via RETURNS clause
- body of inline table-valued function consists of SQL query after RETURN statement
- because inline table-valued function returns result of single SELECT query, you don't need to bother with declaring table variable or defining return-table structure - structure of result is implied by SELECT query that makes up body

- for example consider the problem to split comma-delimited list of strings into table to simulate an array
- gives you flexibility of an array that could be passed to functions or stored procedures

- CREATE FUNCTION dbo.GetCommaSplit (@String nvarchar(max))
RETURNS table AS
- RETURN (
- WITH Splitter (Num, String) AS (
- SELECT Num, SUBSTRING(@String, Num,
• CASE CHARINDEX(N ',', @String, Num)
• WHEN 0 THEN LEN(@String) - Num + 1 ELSE CHARINDEX(N ',',
• @String, Num) - Num END) AS String FROM dbo.Numbers WHERE
• Num <= LEN(@String)
- AND (SUBSTRING(@String, Num - 1, 1) = N',' OR Num = 0))
- SELECT ROW_NUMBER() OVER (ORDER BY Num) AS Num,
• RTRIM(LTRIM(String)) AS Element FROM Splitter WHERE String <>
•);

- function accepts comma-delimited varchar(max) string and returns a table with two columns, Num and Element
- Num column contains unique number for each element of array, counting from 1 to number of elements in comma- delimited string
- Element column contains substrings extracted from comma-delimited list
- looping and procedural constructs aren't allowed in inline table-valued function

- CASE expressions required to handle two special cases:
- first item in list, it isn't preceded by comma
- last item in list, it isn't followed by comma

- SELECT T1.String + T2.String
- FROM GetCommaSplit("1st parameter is , 2nd parameter is , and 3rd parameter is ") as T1
- JOIN SELECT GetCommaSplit("1, to, C, D) as T2
- ON T1.Num = T2.Num
- ?

- SELECT T1.String + T2.String
 - FROM GetCommaSplit("1st parameter is , 2nd parameter is , and 3rd parameter is ") as T1
 - JOIN SELECT GetCommaSplit("1, to, C, D) as T2
 - ON T1.Num = T2.Num
-
- 1st parameter is 1
 - 2nd parameter is to
 - and 3rd parameter is C

Restrictions on User-Defined Functions

- **Nondeterministic Functions**
- **Change State of Database**
- **Side Effects**

prohibits use of nondeterministic functions

- *deterministic* function is one that returns same value every time when passed given set of parameters
- *nondeterministic* function can return different results with same set of parameters passed to it
- ABS, mathematical absolute value function is an example of deterministic function
- functions such as RAND and NEWID are nondeterministic because they return different result every time they're called

restrictions on the use of nondeterministic

- @@CONNECTIONS @@PACK_RECEIVED
- @@TOTAL_WRITE @@CPU_BUSY
- @@PACK_SENT CURRENT_TIMESTAMP
- @@DBTS@ @PACKET_ERRORS
- GET_TRANSMISSION_STATUS @@IDLE
- @@TIMETICKS GETDATE @@IO_BUSY
- @@TOTAL_ERRORS GETUTCDATE
- @@MAX_CONNECTIONS @@TOTAL_READ

aren't allowed to change
state of database

aren't allowed to cause
other side effects

- can't execute PRINT statements from within UDF
- can't execute INSERT, UPDATE or DELETE statements against database tables
- . Some other restrictions include the following:

- can't create temporary tables within UDF
 - can, however, create and modify table variables in body of UDF
- can't execute CREATE, ALTER, or DROP on database tables from within UDF
- dynamic SQL isn't allowed within UDF, although XPs and SQLCLR functions can be called

Procedural Extensions Structured Query Language

Stored Procedures

Stored Procedure

- Stored Procedures provide means for creating server-side subroutines
- CREATE PROCEDURE statement creates stored procedure
- first time procedure is executed, each SQL statement it contains is compiled and executed to create execution plan; then procedure is stored in compiled form within database
- to execute, or call, stored procedure, you use EXEC/EXECUTE statement

Stored Procedure

- for each subsequent execution, SQL statements are executed without compilation, they're precompiled; makes execution of stored procedure faster than execution of equivalent SQL script
- to delete Stored Procedure from database, use DROP PROCEDURE statement
- to modify definition of Stored Procedure use ALTER PROCEDURE statement to specify new definition
- can call Stored Procedure from within another stored procedure; can even call from within itself - recursively

Advantages

- application programmers and end users don't need to know structure of database or how to code complex SQL
- can restrict and control access to database
 - can prevent both accidental errors
 - and malicious damage

Advantages

- offer code modularization and security
- code modules helps reduce redundant code, eliminating potential maintenance nightmares caused by duplicate code stored in multiple locations
- can deny users ability to perform direct queries against tables, but still allow them to use Stored Procedures to retrieve relevant data
- centralized administration of portions of your database application code

Advantages

- prevent SQL injection by using procedures with parameters to filter and validate all inputs
- can effectively build Application Programming Interface (API) for your database; adherence to such an API can help ensure consistent access across applications and make development easier for front-end and client-side developers
- compiled into machine language, there will be instances that placing business logic directly in database layer will perform better than other architectures

Disadvantages

- tightly couple your code to DataBase Management System
 - difficult to port to another Relational DBMS (such as PostgreSQL, Oracle, DB2, or MySQL)
 - loosely coupled application is much easier to port

portability / performance

- to get true portability you have to take great care to code everything in plain vanilla SQL, meaning lot of platform-specific performance-enhancing functionality offered by SQL Server is off-limits
- balance between portability and performance needs to be determined by your business requirements and corporate IT policies on a per-project basis

Stored Procedure name

- Microsoft recommends against naming Stored Procedures with prefix sp_ - used by SQL Server to name system Stored Procedures
- name can specify schema; if don't specify schema name SQL Server creates it default schema for your login

Parameters

- specified in comma-separated list following procedure name in CREATE PROCEDURE statement
- when call Stored Procedure can specify parameters in any order; and you can omit them altogether if you assigned default value
- can also specify OUTPUT parameters, which return values from procedure
- this makes Stored Procedure parameters far more flexible than those of User-Defined Functions

Parameters

- each parameter is declared as specific type and can also be declared as OUTPUT or with VARYING keyword (for cursor parameters only)
- when calling can specify parameters by position or by name
- if you specify unnamed parameter list, values are assigned based on position
- if you specify named parameters in format @parameter = value, they can be in any order
- if your parameter specifies default value in declaration, you don't have to pass in value for that parameter

communicate with caller

- Stored Procedure RETURN statement can return *int* value to caller; unlike functions, procedures don't require RETURN statement; .if RETURN statement is left out 0 is returned by default if no errors were raised during execution
- Stored Procedures don't have restrictions on database side effects and determinism
 - can read, write, delete, and update permanent tables

communicate with caller

- when temporary table is created in procedure that temporary table is available to any procedure called by that Stored Procedure
- scope of local temporary table is current session; prefixed with #
- scope of global temporary table is all sessions; prefixed with ##
- global temporary tables are accessible by all users and all connections after they're created; provides useful method of passing an entire table of temporary results from one procedure to another for further processing

communicate with caller

- output parameters provide primary method of retrieving scalar results from Stored Procedure
- output parameters are specified with OUTPUT keyword
- return table-type results from Stored Procedure; can return result sets like virtual tables that can be accessed by caller
- can return multiple result sets with single call

Return

- used only to return *int* status or error code
- normal practice is to return
- value of 0 to indicate success
- and nonzero value or an error code to indicate an error or failure

Best practices

- use schema names when creating or referencing Stored Procedures and database objects; helps find objects more quickly and thus reduces compile lock, which results in less processing time.
- don't use SP_ and sys prefixes to name user-created database objects; reserved and have different behaviors
- avoid using scalar functions in SELECT statements that return many rows of data; applied to every row, behavior is like row-based processing and degrades performance

Best practices

- use parameters when calling Stored Procedures
- explicitly create parameters with type, size, and precision to avoid type conversions
- use TRY...CATCH feature for error handling; place it outside loop for better performance
 - creates less performance overhead
 - makes error reporting more accurate with significantly less programming

Best practices

- avoid using `SELECT *`, and select only columns you need
- use `UNION ALL` operator instead of `UNION` or `OR` operator, unless there is specific need for distinct values
 - `UNION` filters and removes duplicate records, whereas `UNION ALL` operator requires less processing overhead because duplicates aren't filtered out of result set

keep transactions as short as possible

- use explicit transactions by using BEGIN/END TRANSACTION, and keep transactions as short as possible
 - longer the transaction, more chances you have for locking or blocking, and in some cases deadlocking

example Stored Procedure performs running total

- WITH RunningTotalCTE AS (
- SELECT soh.SalesOrderNumber, soh.OrderDate, soh.TotalDue, (
- SELECT SUM(soh1.TotalDue) FROM Sales.SalesOrderHeader sohl
- WHERE sohl.SalesOrderNumber <= soh.SalesOrderNumber) AS
- RunningTotal,
- SUM(soh.TotalDue) OVER () AS GrandTotal FROM
- Sales.SalesOrderHeader soh WHERE DATEPART(year,
- soh.OrderDate) = @Year
- GROUP BY soh.SalesOrderNumber, soh.OrderDate, soh.TotalDue)
- SELECT rt.SalesOrderNumber, rt.OrderDate, rt.TotalDue,
- rt.RunningTotal, (rt.RunningTotal / rt.GrandTotal) * 100 AS
- PercentTotal
- FROM RunningTotalCTE rt ORDER BY rt.SalesOrderNumber;

Running Sums

- commonly used business reporting tool
- calculates totals as of certain points in time
- in example running sum is calculated per order, for each day over course of given year
- gives you total sales amount as of date when each order is placed
- when first order is placed, running sum is equal to amount of that order
- when second order is placed, running sum is equal to amount of first order plus amount of second order... and so on

- next example is a “recommended products list” following business rules:

- list should include additional items on orders that contain product selected by customer
- if product selected by customer is product ID 773 (Silver Mountain-100 44-inch bike), then items previously bought by other customers in conjunction with this bike like product ID 712 (AWC logo cap) should be recommended
- products that are in same category as product customer selected should not be recommended

- recommended product list should never contain more than ten items
- recommended products should be listed in descending order of total quantity that has been ordered
- bestselling items will be listed in recommendations list first

CREATE PROCEDURE Production.

GetProductRecommendations (@ProductID int)

- WITH RecommendedProducts (ProductID, ProductSubCategoryID, TotalQtyOrdered, TotalDollarsOrdered)
 - AS (SELECT od2.ProductID, pi.ProductSubCategoryID,
 - SUM(od2.OrderQty) AS TotalQtyOrdered,
 - SUM(od2.UnitPrice * od2.OrderQty) AS TotalDollarsOrdered
 - FROM Sales.SalesOrderDetail odi INNER JOIN Sales.SalesOrderDetail od2 ON odi.SalesOrderID = od2.SalesOrderID INNER JOIN Production.Product pi ON od2.ProductID = pi.ProductID
 - WHERE odi.ProductID = @ProductID AND od2.ProductID <> @ProductID
 - GROUP BY od2.ProductID, pi.ProductSubcategoryID)
 - SELECT TOP(10) ROW_NUMBER() OVER (ORDER BY rp.TotalQtyOrdered DESC) AS Rank,
 - rp.TotalQtyOrdered, rp.ProductID, rp.TotalDollarsOrdered, p.[Name]
 - FROM RecommendedProducts rp INNER JOIN Production.Product p ON rp.ProductID = p.ProductID WHERE rp.ProductSubcategoryID <> (SELECT ProductSubcategoryID FROM Production.Product WHERE ProductID = @ProductID) ORDER BY TotalQtyOrdered DESC;

- in body of CTE, Sales.SalesOrderDetail table is joined to itself based on SalesOrderID; join to Production.Product is also included to get each product's SubcategoryID
- point of self-join is to grab total quantity ordered (OrderQty) and total dollars ordered (UnitPrice * OrderQty) for each product
- query is designed to include only orders that contain product @ProductID passed in WHERE clause, and it also eliminates results for @ProductID itself from final results
- all results are grouped by ProductID and ProductSubcategoryID

- final part of CTE excludes products that are in same category as item passed in by @ProductID; then limits results to top ten and numbers results from highest to lowest by TotalQtyOrdered
- also joins on Production. Product table to get each product's name
- EXECUTE
Production.GetProductRecommendations ...

- implementing business logic in Stored Procedure provides layer of abstraction that makes it easier to use from front-end applications - don't need to worry about details - need to pass ProductID parameter to procedure which will return relevant information in well-defined result set

- promotes code reuse
- if need to change business logic, it can be done one time, in one place
- management decides to make suggestions based total dollars' worth of product ordered instead of total quantity ordered
- change ORDER BY clause from
 - ORDER BY TotalOtyOrdered DESC;
- to
 - ORDER BY TotalDollarsOrdered DESC;

Recursion in Stored Procedures

- Stored Procedures can call themselves recursively
- there is SQL Server-imposed limit of 32 levels of recursion
- to demonstrate recursion, let's solve puzzle Towers of Hanoi: consists of three pegs and specified number of discs of varying sizes that slide onto pegs

CREATE PROCEDURE dbo.ShowTowers

- WITH FiveNumbers(Num) -- Recursive CTE generates table with numbers 1...5
- AS (SELECT 1 UNION ALL SELECT Num + 1 FROM FiveNumbers WHERE Num < 5),
- GetTowerA (Disc) -- The discs for Tower A
- AS (SELECT COALESCE(a.Disc, -1) AS Disc
- FROM FiveNumbers f LEFT JOIN #TowerA a ON f.Num = a.Disc),
- GetTowerB (Disc) -- The discs for Tower B
- AS (SELECT COALESCE(b.Disc, -1) AS Disc
- FROM FiveNumbers f LEFT JOIN #TowerB b ON f.Num = b.Disc),
- GetTowerC (Disc) -- The discs for Tower C
- AS (SELECT COALESCE(c.Disc, -1) AS Disc
- FROM FiveNumbers f LEFT JOIN #TowerC c ON f.Num = c.Disc)

CREATE PROCEDURE dbo.ShowTowers

- SELECT CASE a.Disc
- WHEN 5 THEN '=====5===== '
- WHEN 4 THEN ' =====4==== '
- WHEN 3 THEN ' ===3== '
- WHEN 2 THEN ' ==2== '
- WHEN 1 THEN ' =1= '
- ELSE ' | '
- END AS Tower_A,

CREATE PROCEDURE dbo.ShowTowers

- SELECT CASE b.Disc
- WHEN 5 THEN '=====5===== '
- WHEN 4 THEN ' =====4===== '
- WHEN 3 THEN ' ===3== '
- WHEN 2 THEN ' ==2== '
- WHEN 1 THEN ' =1= '
- ELSE ' | '
- END AS Tower_b,

CREATE PROCEDURE dbo.ShowTowers

- SELECT CASE c.Disc
- WHEN 5 THEN '=====5===== '
- WHEN 4 THEN ' =====4==== '
- WHEN 3 THEN ' ===3== '
- WHEN 2 THEN ' ==2== '
- WHEN 1 THEN ' =1= '
- ELSE ' | '
- END AS Tower_C,

CREATE PROCEDURE dbo.ShowTowers

- FROM (
- SELECT ROW_NUMBER() OVER(ORDER BY Disc) AS Num, COALESCE(Disc, -1) AS Disc FROM GetTowerA)
- a FULL OUTER JOIN (
- SELECT ROW_NUMBER() OVER(ORDER BY Disc) AS Num, COALESCE(Disc, -1) AS Disc FROM GetTowerB)
- b ON a.Num = b.Num FULL OUTER JOIN (
- SELECT ROW_NUMBER() OVER(ORDER BY Disc) AS Num, COALESCE(Disc, -1) AS Disc FROM GetTowerC)
- c ON b.Num = c.Num
- ORDER BY a.Num;

CREATE PROCEDURE dbo.MoveOneDisc (@Source nchar(1), @Dest nchar(1))

- -- @SmallestDisc is the smallest disc on the source tower
- DECLARE @SmallestDisc int = 0;
- -- IF... ELSE conditional statement gets smallest disc from correct source
- IF @Source = N'A'
- BEGIN
- -- This gets the smallest disc from Tower A
- SELECT @SmallestDisc = MIN(Disc) FROM #TowerA;
- -- Then delete it from Tower A
- DELETE FROM #TowerA WHERE Disc = @SmallestDisc;
- END
- ELSE IF @Source = N'B'

```
CREATE PROCEDURE dbo.MoveOneDisc  
(@Source nchar(1), @Dest nchar(1))
```

- ELSE IF @Source = N'B'
- BEGIN
- -- This gets the smallest disc from Tower B
- SELECT @SmallestDisc = MIN(Disc) FROM #TowerB;
- -- Then delete it from Tower B
- DELETE FROM #TowerB WHERE Disc = @SmallestDisc;
- END
- ELSE IF @Source = N'C'

```
CREATE PROCEDURE dbo.MoveOneDisc  
(@Source nchar(1), @Dest nchar(1))
```

- ELSE IF @Source = N'C'
- BEGIN
- -- This gets the smallest disc from Tower C
- SELECT @SmallestDisc = MIN(Disc) FROM #TowerC;
- -- Then delete it from Tower C
- DELETE FROM #TowerC WHERE Disc = @SmallestDisc;
- END
- -- Show the disc move performed

CREATE PROCEDURE dbo.MoveOneDisc (@Source nchar(1), @Dest nchar(1))

- SELECT N'Moving Disc (' + CAST(COALESCE(@SmallestDisc, 0) AS nchar(1)) + N') FROM Tower ' + @Source + N' to Tower ' + @Dest + ':' AS Description;
- -- Perform the move - INSERT disc from source into destination
- IF @Dest = N'A'
 - INSERT INTO #TowerA (Disc) VALUES (@SmallestDisc);
- ELSE IF @Dest = N'B'
 - INSERT INTO #TowerB (Disc) VALUES (@SmallestDisc);
- ELSE IF @Dest = N'C'
 - INSERT INTO #TowerC (Disc) VALUES (@SmallestDisc);
- -- Show the towers
- EXECUTE dbo.ShowTowers; END;

CREATE PROCEDURE dbo.MoveDiscs

- Stored Procedure moves multiple discs recursive
- CREATE PROCEDURE dbo.MoveDiscs
- (@DiscNum int,
 - @MoveNum int OUTPUT,
 - @Source nchar(1) = N'A',
 - @Dest nchar(1) = N'C',
 - @Aux nchar(1) = N'B'
-) AS
- BEGIN

CREATE PROCEDURE dbo.MoveDiscs

- -- If number of discs to move is 0, solution has been found
- IF @DiscNum = 0 PRINT N'Done';
- ELSE
 - BEGIN
 - -- If number of discs to move is 1, go ahead
 - IF @DiscNum = 1
 - BEGIN
 - -- Increase move counter by 1
 - SELECT @MoveNum = @MoveNum + 1;
 - -- finally move one disc from source to destination
 - EXEC dbo.MoveOneDisc @Source, @Dest;
 - END ELSE BEGIN

CREATE PROCEDURE dbo.MoveDiscs

- -- Determine number of discs to move from source to
- DECLARE @n int = @DiscNum - 1;
- -- Move (@DiscNum - 1) discs from source to auxiliary tower
- EXEC dbo.MoveDiscs @n, @MoveNum OUTPUT, @Source, @Aux, @Dest;
- -- Move 1 disc from source to final destination tower
- EXEC dbo.MoveDiscs 1, @MoveNum OUTPUT, @Source, @Dest, @Aux;
- -- Move (@DiscNum - 1) discs from auxiliary to final destination
- EXEC dbo.MoveDiscs @n, @MoveNum OUTPUT, @Aux, @Dest, @Source; END; END;

CREATE PROCEDURE dbo.SolveTowers

- Stored Procedure creates three towers and populates Tower A with 5 discs
- CREATE PROCEDURE dbo.SolveTowers
- AS BEGIN
- CREATE TABLE #TowerA (Disc int PRIMARY KEY NOT NULL);
- CREATE TABLE #TowerB (Disc int PRIMARY KEY NOT NULL);
- CREATE TABLE #TowerC (Disc int PRIMARY KEY NOT NULL);
- -- Populate Tower A with all five discs

CREATE PROCEDURE dbo.SolveTowers

- -- Solve the puzzle specify parameters with defaults
- EXECUTE dbo.MoveDiscs 5, @MoveNum OUTPUT;
- -- How many moves did it take?
- PRINT N'Solved in ' + CAST (@MoveNum AS nvarchar(10)) + N' moves.';
- -- Drop temp tables to clean up
- DROP TABLE #TowerC;
- DROP TABLE #TowerB;
- DROP TABLE #TowerA; END;

CREATE PROCEDURE dbo.SolveTowers

- -- Populate Tower A with all five discs
- INSERT INTO #TowerA (Disc) VALUES (1), (2), (3), (4), (5);
- -- Initialize move number to 0
- DECLARE @MoveNum int = 0;
- -- Show initial state of towers
- EXECUTE dbo.ShowTowers;
- -- Solve the puzzle specify parameters with defaults

Solve puzzle

```
EXECUTE dbo.SolveTowers
```

- results are best viewed in Results to Text mode
- put SSMS in Results to Text mode by pressing Ctrl+T while in Query Editor window
- to switch to Results to Grid mode, press Ctrl+D

EXECUTE dbo.SolveTowers

discs are moved from tower to tower

```
Results
Description
-----
Moving Disc (1) from Tower C to Tower B:

Tower_A          Tower_B          Tower_C
-----
|                |                |
|                |                |
---3---          |                |
-----4-----      --1--            |
=====5=====     ==2==            |

Description
-----
Moving Disc (3) from Tower A to Tower C:

Tower_A          Tower_B          Tower_C
-----
|                |                |
|                |                |
|                |                |
---4---          |                |
=====5=====     ==2==            ===3====
```

- Stored Procedures can call themselves recursively: MoveDiscs procedure, which calls itself until puzzle is solved
- when default values are assigned to parameters in procedure declaration, you don't have to specify values for them when you call procedure: SolveTowers procedure, which calls MoveDiscs

- scope of temporary tables created in procedure includes procedure in which they're created, as well as any procedure it calls and any procedure they in turn call: SolveTowers creates three temporary tables - procedures called by SolveTowers and those called by those procedures (and so on) can also access these same temporary tables
- MoveDiscs Stored Procedure demonstrates output parameters: uses output parameter to update count of total number of moves performed

Table-Valued Parameters

- can pass table-valued parameters to Stored Procedures and User-Defined Functions

methods of passing multiple data to functions procedures

- passing lots and lots of parameters - can accept up to 2,100 parameters
 - results in complex code that can be extremely difficult to manage
- converting multiple rows to an intermediate format like comma-delimited or XML
 - have to parse out parameter into temporary table to extract rows from intermediate format - conversions can be costly, especially when large amounts of data are involved

methods of passing multiple data to functions procedures

- place rows in permanent or temporary table and call procedure
 - eliminates conversions
 - managing multiple sets of input rows from multiple simultaneous users can introduce a lot of overhead
- calling procedures multiple times with single row of data each time
 - manipulating potentially large number of rows of data, one row at a time, can result in big performance penalty

CREATE PROCEDURE

HumanResources.GetEmployees

- (@LastNameTable
HumanResources.LastNameTableType READONLY) AS
BEGIN
- SELECT p.LastName, p.FirstName, p.MiddleName,
e.NationalIDNumber, e.Gender, e.HireDate
- FROM HumanResources.Employee e
- INNER JOIN Person.Person p ON e.BusinessEntityID =
p.BusinessEntityID
- INNER JOIN @LastNameTable Int ON p.LastName =
Int.LastName
- ORDER BY p.LastName, p.FirstName, p.MiddleName;
- END;

- to create table-valued parameter, must first create table type that defines structure
- CREATE TYPE HumanResources.LastNameTableType
- AS TABLE (LastName nvarchar(50) NOT NULL PRIMARY KEY);
- table-valued parameter is declared READONLY, which is mandatory - can query and join to rows in table-valued parameter just like table variable, you can't manipulate rows with INSERT, UPDATE, DELETE statements

Calling Procedure with table-valued parameter

- `DECLARE @LastNameList
HumanResources.LastNameTableType;`
- `INSERT INTO @LastNameList (LastName)`
- `VALUES`
- `(N'Walters'), (N'Anderson'), (N'Chen'), (N'Rettig'),`
- `(N'Lugo'), (N'Zwilling'), (N'Johnson');`
- `EXECUTE HumanResources.GetEmployees
@LastNameList;`

- to call procedure with table-valued parameter
- declare variable of same type as table-valued parameter
- populate variable with rows of data and pass variable as parameter to procedure
- . Listing 5-10 demonstrates how to call the HumanResources . GetEmployees SP with a table-valued parameter. The results are shown in Figure 5-7.

Results Messages

	LastName	FirstNa...	MiddleNa...	NationalIDNum...	Gen...	HireDate
1	Anderson	Nancy	A	693325305	F	1999-02-03 00:00:00.000
2	Chen	Heo	O	416679555	M	1999-03-10 00:00:00.000
3	Chen	John	Y	305522471	M	1999-03-13 00:00:00.000
4	Johnson	Barry	K	912265825	M	1998-02-07 00:00:00.000
5	Johnson	David	N	498138869	M	1999-01-03 00:00:00.000
6	Johnson	Willis	T	332040978	M	1999-01-14 00:00:00.000
7	Lugo	Jose	R	788456780	M	1999-03-14 00:00:00.000
8	Rettig	Bjorn	M	420023788	M	1999-02-08 00:00:00.000
9	Walters	Rob	NULL	112457891	M	1998-01-05 00:00:00.000
10	Zwilling	Michael	J	582347317	M	2000-03-26 00:00:00.000

restrictions table-valued parameters

- read-only
- can't use table-valued parameter as target of an INSERT EXEC or SELECT INTO assignment statement
- table-valued parameters are scoped just like other parameters and local variables declared in procedure or function - they aren't visible outside of procedure in which they're declared; doesn't maintain column-level statistics for table-valued parameters, which can affect performance

Temporary Stored Procedures

- created just like any other Stored Procedures, only difference is that name begin with number sign (#) for local temporary procedure and two number signs (##) for global temporary procedure
- third possibility is to create temporary SP in tempdb database - scope of anything created in tempdb database is until instance is restarted, because tempdb is re-created each time an instance is restarted
- local temporary procedure is visible only to current session and is dropped when current session ends
- global temporary procedure is visible to all connections and is automatically dropped when last session using it ends

Temporary Stored Procedures

- normally you won't use temporary SPs; usually used for specialized solutions, like database drivers
- Open Database Connectivity (ODBC) drivers, for instance, use temporary stored procedure to implement SQL Server connectivity functions
- temporary stored procedures are useful when you want advantages of using procedures, such as execution plan reuse and improved error handling, with advantages of ad hoc code
- temporary stored procedures bring some other effects, as well; they're often not destroyed until connection is closed or explicitly dropped - may cause procedures to fill up tempdb over time and cause queries to fail
- creating temporary stored procedures in transaction may also cause blocking problems, because procedures creation causes data-page locking in several system tables for transaction duration

Recompilation and Caching

- features that work behind scenes to optimize stored procedure performance
- first time you execute procedure SQL Server compiles it into query plan, which it then caches
- compilation process invokes certain amount of overhead, which can be substantial for procedures that are complex or that are run very often
- SQL Server uses complex caching mechanism to store and reuse query plans on subsequent calls to same stored procedure, in an effort to minimize impact of compilation overhead

Stored Procedure Statistics

- SQL Server 2014 provides dynamic management functions to expose stored procedure query-plan usage and caching information that can be useful for performance tuning and general troubleshooting

Parameter Sniffing

- to further optimize stored procedure calls
- during compilation or recompilation of procedure SQL Server captures parameters used and passes values along to optimizer
- optimizer then generates caches query plan optimized for those parameters - can actually cause problems in some cases (for example, when stored procedure can return wildly varying numbers of rows based on parameters passed in)
- in cases where you expect widely varying numbers of rows to be returned by procedure can override parameter sniffing on per-procedure basis

Recompilation

- recompilation of stored procedures is performed on individual statements rather than entire procedure to avoid unnecessary recompiles and consuming CPU resources
- if object is modified between executions, each statement in procedure that references this object is recompiled
- if sufficient data has changed in table that is being referenced by procedure since original query plan was generated, procedure recompiles plan
- use of temporary table may cause procedure to be recompiled every time it is executed
- if procedure was created with recompile option, this may cause to be recompiled every time procedure is executed

Triggers

triggers

- means of executing code in response to database object, database, and server events
- Data Manipulation Language (DML) triggers, which fire in response to INSERT, UPDATE, and DELETE events against tables;
- Data Definition Language (DDL) triggers, which fire in response to CREATE, ALTER, and DROP statements;
 - can also fire in response to some system Stored Procedures that perform DDL-like operations
- logon triggers, which fire in response to LOGON events

triggers

- form of specialized procedure closely tied to your data and database objects
- in past, DML triggers were used to enforce various aspects of business logic, such as foreign key and other constraints on data, and other more complex business logic
- Declarative Referential Integrity (DRI) and robust check constraints have supplanted DML triggers in many areas, but they're still useful in their own right

DML Triggers

- code that is executed (fired) in response to INSERT, UPDATE, or DELETE on table
- can create multiple triggers on same objects; will fire in no specific order (can specify by using system stored procedure)
- created via CREATE TRIGGER statement, which allows you to specify following details:
 - name, table, events
 - AFTER/FOR and INSTEAD OF indicators
 - options like ENCRYPTION and EXECUTE AS clauses

CREATE TRIGGER

- name of the trigger; schema must be same as schema for table on which trigger executes
- table or view on which trigger executes
- triggering events: can be any combination of INSERT, UPDATE, and DELETE
- AFTER/FOR and INSTEAD OF indicators, determine whether trigger is fired after statement completes or trigger overrides statement
- additional options like ENCRYPTION and EXECUTE AS clauses, which allow you to obfuscate trigger source code and specify context under which trigger executes

- in addition to CREATE TRIGGER provides ALTER TRIGGER statement to modify, DROP TRIGGER statement to remove and DISABLE TRIGGER and ENABLE TRIGGER statements
- disabling triggers can greatly improve performance when apply batch of modifications
 - just make sure rules enforced by triggers are checked another way
 - don't forget to re-enable trigger at end

CREATE TRIGGER

- CREATE TRIGGER HumanResources.EmployeeUpdateTrigger
- ON HumanResources.Employee AFTER UPDATE
- NOT FOR REPLICATION
- AS BEGIN
- IF @@ROWCOUNT = 0 RETURN
- UPDATE HumanResources.Employee SET ModifiedDate = GETDATE() WHERE EXISTS (
- SELECT 1 FROM inserted i WHERE i.BusinessEntityID = HumanResources.Employee.BusinessEntityID);
- END;

CREATE TRIGGER

- CREATE TRIGGER
HumanResources.EmployeeUpdateTrigger
- ON HumanResources.Employee
- AFTER UPDATE
- NOT FOR REPLICATION
- keywords prevent replication events from firing trigger

- checking @@ROWCOUNT at start of trigger helps ensure that your triggers are efficient - must be done at very first line - any previous action in trigger, even SET commands, could change value
- trigger turns off rows affected messages via
- SET NOCOUNT ON statement
- -- Turn off "rows affected" messages

- IF statement contains UPDATE statement that sets ModifiedDate column to current date and time when rows in table are updated
- important concept of trigger programming is to be sure you account for multiple row updates
- it's not safe to assume that DML statement will update only single row of your table; triggers are set-oriented
- use virtual table

inserted and deleted Virtual Tables

- when trigger fires SQL Server populates inserted and deleted virtual tables and makes them available within body of trigger
- virtual tables have same structure as affected table and contain data from all affected rows
- virtual tables are read-only and can't be modified directly

inserted and deleted Virtual Tables

- inserted table contains all rows inserted into destination table by INSERT
- deleted table contains all rows deleted from destination table by DELETE
- for UPDATE statements, rows are treated as DELETE followed by INSERT
 - pre-UPDATE-affected rows are stored in deleted
 - post-UPDATE-affected rows are stored in inserted

- imagine very simple way to f***k database

- UPDATE HumanResources.Employee SET MaritalStatus = 'M' WHERE FirstName IN ('Ion', 'Nina');
- if RECURSIVE_TRIGGERS database option is turned on in AdventureWorks database, HumanResources . EmployeeUpdateTrigger will error out with message that “nesting limit has been exceeded”
- caused by trigger recursively firing itself after UPDATE statement
- ALTER DATABASE AdventureWorks SET RECURSIVE_TRIGGERS OFF
- ALTER DATABASE AdventureWorks SET RECURSIVE_TRIGGERS ON

Nested and Recursive Triggers

- SQL Server supports triggers firing other triggers (nested triggers) - nested trigger is trigger that is fired by action of another trigger, on same or different table
- triggers can be nested up to 32 levels deep
- additional levels of nesting affect performance
- turned on by default; but can turn them off with `EXEC sp_configure 'nested triggers', 0;`

Auditing with DML Triggers

- common use for DML triggers is auditing DML actions against tables
- maintain record of changes to data
- better use feature known as Change Data Capture (CDC), which provides built-in auditing functionality

UPDATE() and COLUMNS_UPDATED() System Functions

- triggers can take advantage of two system functions, UPDATE() and COLUMNS_UPDATED(), to tell which columns are affected
- UPDATE() takes name of column as parameter and returns true or false
- COLUMNS_UPDATED() returns bit pattern indicating which columns are affected

Triggers on Views

- create INSTEAD OF triggers on views - can be useful for updating views that are otherwise non-updatable (with multiple base tables ...)
- AdventureWorks database comes with view named Sales.vSalesPerson, which is formed by joining 11 separate tables together
- trigger allows you to update specific columns of two of base tables used in view by executing UPDATE statements directly

CREATE TRIGGER

Sales.vIndividualCustomerUpdate

- ON Sales.vIndividualCustomer INSTEAD OF UPDATE NOT FOR REPLICATION AS BEGIN
- IF UPDATE(FirstName) OR UPDATE(MiddleName) OR UPDATE(LastName)
- BEGIN
 - UPDATE Person.Person SET FirstName = i.FirstName, MiddleName = i.MiddleName, LastName = i.LastName
 - FROM inserted i WHERE i.BusinessEntityID = Person.Person.BusinessEntityID;
 - SET @UpdateSuccessful = 1;
- END;

- IF UPDATE(EmailAddress)
- BEGIN
 - UPDATE Person.EmailAddress SET EmailAddress =
i.EmailAddress FROM inserted i WHERE i.BusinessEntityID
= Person.EmailAddress.BusinessEntityID;
 - SET @UpdateSuccessful = 1;
- END;
- IF @UpdateSuccessful = 0
 - RAISERROR('Must specify updatable columns.', 10, 127);
- END;

- trigger checks to see whether columns were affected by UPDATE statement
- if proper columns were affected by UPDATE statement, trigger performs updates on appropriate base tables for view
- columns that are updatable by trigger are:
- FirstName, MiddleName, and LastName columns from Person.Person table
- EmailAddress column from Person.EmailAddress table

DDL Triggers

- fire when DDL events occur in database or on the server
- CREATE TRIGGER statement for DDL triggers is only slightly different from DML trigger syntax, with major difference being that you must specify scope for trigger: either ALL SERVER or DATABASE
- DATABASE scope causes trigger to fire if event occurs in database
- ALL SERVER scope causes trigger to fire if event occurs anywhere on server

DDL Triggers

- are largely of form CREATE, ALTER, DROP, GRANT, DENY, or REVOKE
- useful when you want to prevent changes to your database, perform actions in response to change in database, or audit changes
- which DDL statements can fire DDL trigger depends on scope of trigger

Logon Triggers

- fire in response to SQL Server LOGON event, after authentication succeeds, but before user session is established
- can perform tasks ranging from simple LOGON event auditing to more advanced tasks like restricting number of simultaneous sessions or denying users ability to create sessions at certain times

- **Thank you very much for kindly attention!**

- three assignments for two weeks
- March 20th
- March 27th
- April 3rd
-

1 - one

- in your preferred DataBase Management System (Oracle, SQL Server, PostgreSQL, MySQL, ...)
- in your preferred Database domain
- implement 2-3 stored procedure
- implement 2-3 scalar function
- implement 2-3 Table function stored procedure
- implement 2-3 triggers
- show me the code and how you use the code

2 - two Hotel example problem

- price depend on room type and season
 - season – begin date, end date
- tourist stay
 - check in date
 - check out date
 - Room
- *How much do they have to pay? (if stay spans multiple intervals)*

3- three Genealogy tree

- Implement a genealogy tree
- [Mathematics Genealogy](#)
- implement code to calculate the number of descendants
- descendants of Euler = 10
- or
- [House of Windsor](#)
- implement code to calculate the number of descendants
- descendants of Queen Elisabeth = 17?

- or your own genealogy
- or Genealogia regelui Mihai I Romania

- or if you are not a coward ...
- genealogy of *manele* (Guta it is interesting case ...)
- *nonebrity* ... how many love partners Bianca Dragusanu had? How many love interests Catalin Botezatu had? Who are common friends ...?

DataBase Administrator

SQL Server 2008

DBA RESPONSIBILITIES

DataBase Design course notes 06
SQL Server Administration

Security

- Securing organization's systems and data
- Choosing authentication mode
- TCP port security

Availability

- Ensuring database is available when required
- failover clustering
- database mirroring,
- design redundancy into server components

Reliability

- Proactive maintenance and design practices

Recoverability

- Plan of attack for restoring database as quickly as possible

PLANNING & INSTALLATION

DataBase Design course notes 06
SQL Server Administration

Planning and Installation

- **Storage system sizing**
- Physical server design
- Installing and upgrading SQL Server 2008
- Failover clustering

Characterizing I/O workload

- OLTP vs. OLAP/ Decision Support System
- OnLine Transaction Processing
 - random I/O dominant; read/write intensive
 - frequent & small transactions
- OnLine Analytical Processing
 - sequential I/O dominant; read intensive
 - fewer but larger transactions

Volume of workload

- typically measured by the number of disk reads and writes per second

Calculating number of disks required

- Required # Disks = (Reads/sec + (Writes/sec * RAID adjuster)) / Disk IOPS
- I/O per second capacity (IOPS) of individual disks involved
 - 125 IOPS figure is a reasonable average for the purposes of estimation

Storage formats

- ATA
 - IDE, parallel interface
 - integrates disk controller on disk itself and uses ribbon-style cables to connect to host
- S-ATA
 - serial ATA
 - faster data transfer, thinner cables
 - Native Command Queuing (NCQ) queued disk requests are reordered to maximize throughput
 - much higher capacity per disk, with increased latency of disk requests

Storage formats

- SCSI
 - higher performance for higher cost
 - commonly found in server-based RAID implementations
 - SCSI controller card, up to 15 disks can be connected for each channel on controller
 - database application can use SCSI drives for storing database and SATA drives for storing online disk backups

Storage formats

- SAS Serial Attached SCSI
- Fibre Channel
 - high-speed, serial duplex communications between storage systems and server hosts
- Solid-state disks
 - with no moving parts are more robust
 - promise near-zero seek time, high performance, and low power consumption

Bus bandwidth

- Typical SCSI bus used today is *Ultra320*, with maximum throughput of 320MB/second per channel

Disk Capacity

- guiding principle in designing high-performance storage solutions for database is to stripe data across a large number of dedicated disks and multiple controllers
- resultant performance is much greater than what you'd achieve with fewer, higher-capacity disks

Selecting appropriate RAID level

- good server design is one that has no, or very few, single points of failure
 - among most common server components that fail are disks
- contributing factors include heat and vibration
- data center takes measures to reduce failure rates, locating servers in temperature-controlled positions with low vibration levels
- disk redundancy with ***Redundant Array of Independent/Inexpensive Disks***

RAID 0

- provides no redundancy at all
- involves striping data across all disks
- improves performance, but if any of the disks in the array fail, then the whole array fails
- actually increases chance of failure

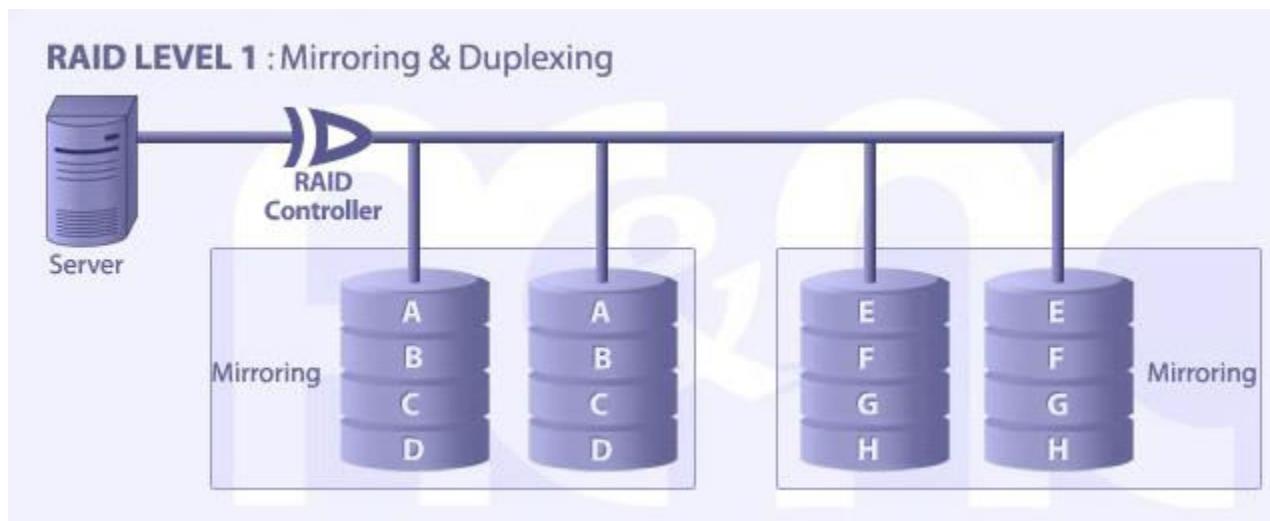
RAID 0



RAID 1

- essentially disk mirroring
- each disk in RAID 1 array has mirror partner
- if one of disks in pair fails, then other disk is still available and operations continue without loss
- useful for backups and transaction logs
- good read performance
- write performance suffers little or no overhead
- downside is lower disk utilization, 50 percent utilization level

RAID 1



RAID 5

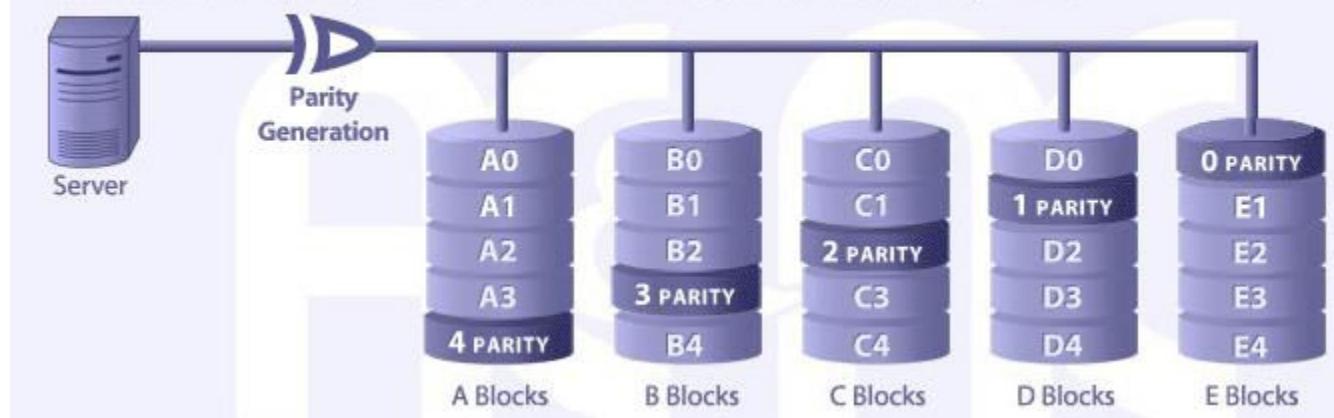
- requires at least three disks
- using parity to provide redundancy
- when disk failure occurs data stored on that disk is dynamically recovered using parity information on remaining disks

RAID 5

- disk utilization calculated as # of drives-1 / # of drives
- for 3 disk volumes, utilization is 66%, for 5 disk, 80%
- lower overall storage cost
- each write to array involves multiple disk operations for parity calculation and storage; write performance is lower
- in the event of disk failure, read performance is also degraded significantly
- suitable for predominantly read-only profiles or with budgetary constraints that can handle write overhead

RAID 5

RAID LEVEL 5 : Independent Data Disks with Distributed Parity Blocks

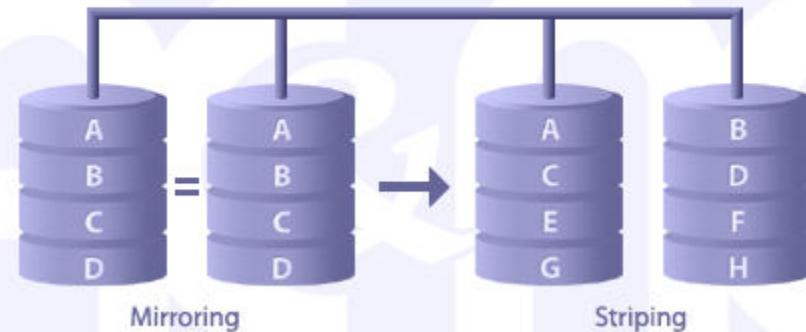


RAID 10

- combines best features of RAID 1 and 0, without any of the downsides of RAID 5
- also known as RAID 1+0
- highest performance RAID option
- downside is the cost
- requiring at least four disks
- benefit from lots of disks to stripe across, each of which requires mirror partner

RAID 10

RAID LEVEL 10 : Very High Reliability Combined with High Performance



RAID can be implemented

- software level, via Windows operating system
 - shouldn't be used in server implementations
 - consumes operating system resources and doesn't offer same feature set as hardware implementations
- hardware level, using dedicated RAID controller
 - requires no operating system resources
 - provides additional benefits, battery- backed, disk cache, support for swapping out failed disks without bringing down system

- process of selecting RAID levels and calculating required number of disks is significantly different in Storage Area Networks compared to traditional direct-attached storage solution
- configuring and monitoring virtualized SAN storage is special skill
- DBAs should insist on SAN vendor involvement in setup and configuration of storage for database

Planning and Installation

- Storage system sizing
- **Physical server design**
- Installing and upgrading SQL Server 2008
- Failover clustering

- SQL Server volumes should be formatted with a 64K allocation unit size using the NTFS file system
- after the underlying partition has been track-aligned

- when choosing server, pay attention to server's I/O capacity, measured by amount of supported PCI slots and bus type
- servers use PCI Express bus, which is capable of transmitting up to 50MB/second per *lane*
- good server selection is one that supports multiple PCI-E slots
- for example seven PCI-E slots comprised of 3x8 slots and 4x4 slots for a total of 40 lanes, could support up to seven controller cards driving a very high number of disks

use multiple controller cards

- important to ensure there are no bottlenecks while writing to transaction log
- store transaction log on dedicated, RAID-protected disks, optionally connected to a dedicated disk controller channel or separate controller card
- using multiple controller cards helps to increase disk performance and therefore reduce impact of most common hardware bottleneck

Cache

- disk controller cache improves performance for both reads and writes
- when data is read from disk, if requested data is stored in controller cache, then physical reads of disk aren't required
- when data is written to disk, it can be written to cache and applied to disk at a later point, thus increasing performance

SQLIO

- tool used to measure I/O performance capacity of storage system
- run from command line, takes parameters that are used to generate I/O of a particular type
- at completion of test, returns various capacity statistics
- use prior to installation of SQL Server to measure effectiveness of various storage configurations
- identify optimal storage configuration
- often exposes various hardware and driver/firmware-related issues

SQLIOSIM

- storage verification tool
- issues disk reads and writes using same I/O patterns as database
- uses checksums to verify integrity of written data pages

- SET STATISTICS IO ON
- SELECT COUNT(*) FROM employees
- SET STATISTICS IO OFF
- Results:
- Table 'Employees'. Scan count 1, logical reads 53, physical reads 0, read-ahead reads 0.

- scan count tells us number of scans performed
- logical reads show number of pages read from cache
- physical reads show number of pages read from disk
- read-ahead reads indicate number of pages placed in cache in anticipation of future reads

CPU architecture

- servers suitable for SQL Server deployments are typically ones with support for 2 or 4 dual- or quad-core
- delivers between 4 and 16 CPU cores in a relatively cheap two- or four-way server.
- such CPU power is usually enough for most database server requirements

64-bit platform

- Itanium CPUs are best used in delivering supercomputer-like performance in systems
- vast bulk of 64-bit deployments in use today are based on Xeon or Opteron x64 CPUs
- with the exception of all but the largest systems, the x64 platform represents the best choice from both a cost and a performance perspective

Memory configuration

- insufficient RAM is a common problem in SQL Server systems experiencing performance problems
- RAM is both reasonably inexpensive and relatively easy to upgrade
- consider amount, type, and capacity of RAM
 - server will be used for future system consolidation, or exact RAM requirements can't be accurately predicted, then apart from loading system up with maximum possible memory, it's important to allow for future memory upgrades

Hot-add CPU and memory

- SQL Server supports hot-add memory and CPU, meaning that if underlying hardware is capable of dynamically adding these resources, SQL Server can take advantage of them without requiring a restart
- there are number of key restrictions on using this feature, fully described in MS Resources

Networking components

- network I/O isn't likely to contribute to performance bottlenecks
- tuning disk I/O, memory, and CPU resources is far more likely to yield bigger performance increases
- network-related settings take into account
 - maximizing switch speed
 - building fault tolerance into network cards

Gigabit switches

- almost all servers come preinstalled with one or more gigabit network connections
- speed of network card is only as good as the switch port it's connected to

NIC teaming

- increase network bandwidth and provide fault tolerance at network level
- involves two or more physical network interface cards (NICs) used as a single logical NIC
- cards operate at the same time to increase bandwidth
- if one fails, other continues operating
- for further fault tolerance, each card is ideally connected to a separate switch
 - network cards offer autosense mode that permits self-configuration
 - it's not uncommon to get it wrong, artificially limiting throughput, so NIC settings (speed and duplex) should be manually configured

Planning and Installation

- Storage system sizing
- Physical server design
- **Installing SQL Server 2008**
- Failover clustering

Note on Windows 7 32 bits

- SQL Server should be the first program installed after operating system
 - before anything else .NET related

Services

- SQL Server
- SQL Server Agent
- SQL Server Analysis Services
- SQL Server Reporting Services
- SQL Server Integration Services

Service accounts

- specified during installation
- need to be created in advance
- Domain accounts
 - can use local server accounts
 - better choice as they enable access other instances and domain resources
- Separate accounts for each service

Service accounts

- Nonprivileged accounts
 - do not be members of domain administrators or local administrator groups
 - installation will grant service accounts necessary permissions to file system and registry
 - additional permissions, such as access to directory for data import/export purposes, should be manually granted for maximum security
- Additional account permissions
 - Perform Volume Maintenance Tasks, required for Instant Initialization
 - Lock Pages in Memory, required for 32-bit AWE-enabled systems and recommended for 64-bit systems
- Password expiration
 - shouldn't have any password expiration policies

Additional checks and considerations prior install

- Collation
 - to determine how characters are sorted and compared
 - by default, SQL Server setup will select collation to match server's Windows collation
 - inconsistent collation selection is a common cause of various administration problems
- Storage configuration
 - must ensure partitions are offset and formatted with 64K allocation unit size

Additional checks and considerations prior install

- Directory creation
 - specify locations for database data, log files, backup files, and tempdb data
 - for maximum performance, create each of these directories on partitions that are physically separate from each other
 - should be created before installation
- Network security
 - secured behind firewall, and unnecessary network protocols such as NetBIOS, SMB, should be disabled

Additional checks and considerations prior install

- Windows version
 - requires at least Win Server 2003 SP2 as prerequisite
 - shouldn't be installed on primary or backup domain controller - server should be dedicated to database
- Server reboot
 - SQL Server won't install if there are pending reboots
- WMI - Windows Management Instrumentation
 - WMI service must be installed and working properly before SQL Server can be installed

Windows Server 2008

- ideal underlying operating system for SQL Server is Windows Server 2008
- networking stack is substantially faster
- Enterprise and Data Center editions provides virtualization opportunities
- more clustering options

Default and named instances

- multiple instances (copies) of SQL Server can be installed on one server
 - control amount of memory and CPU resources granted to each instance
- selection between named instance and default instance
 - there can only be a single default instance per server
 - supports installation of up to 50 named instances

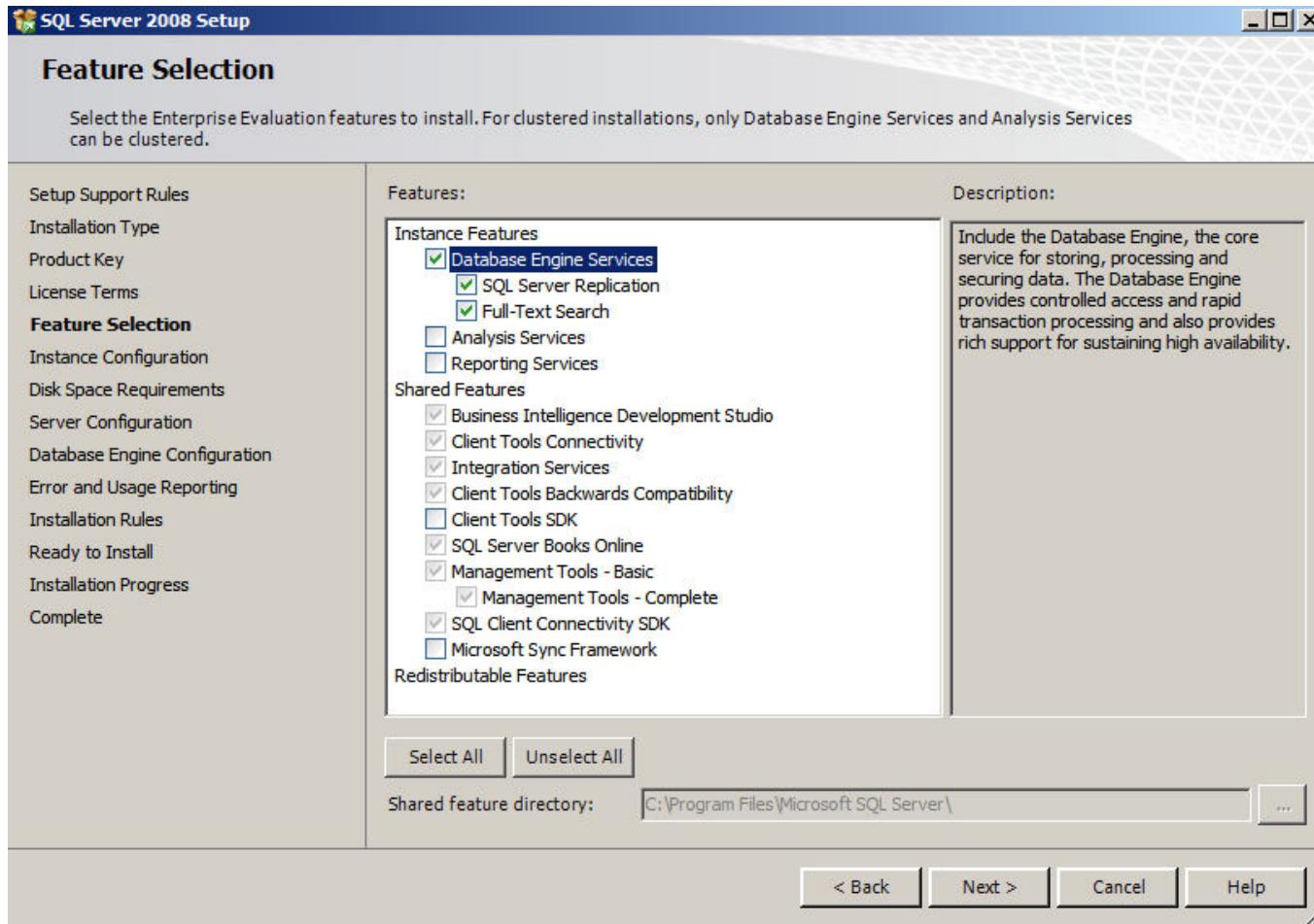
Start installation; run setup.exe from SQL Server DVD

- begins with check on installed versions of Windows Installer and .NET Framework
- if required versions are missing, setup process offers choice to install them
- after these components are verified (or installed), setup begins with SQL Server Installation Center

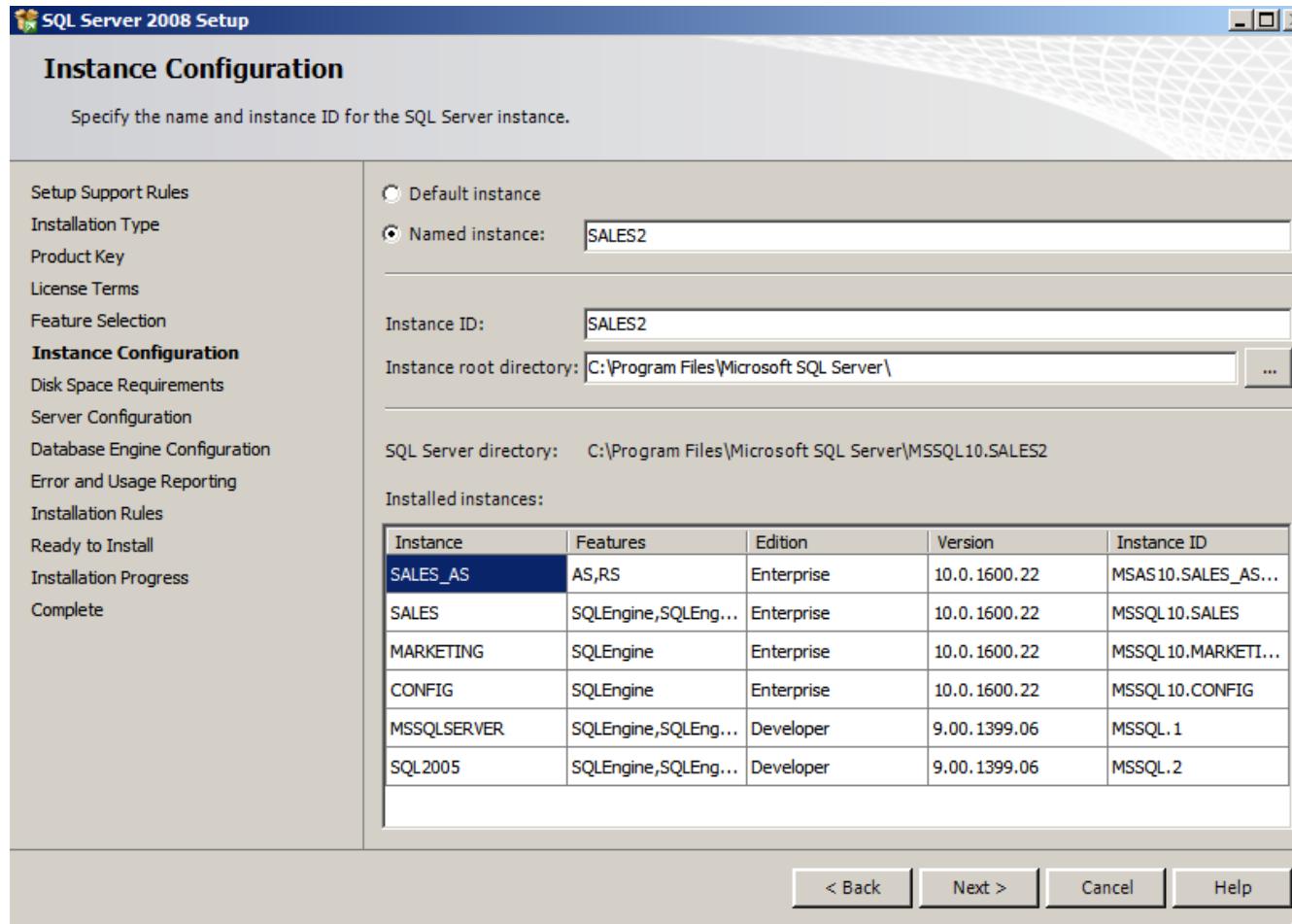
SQL Server Installation Center



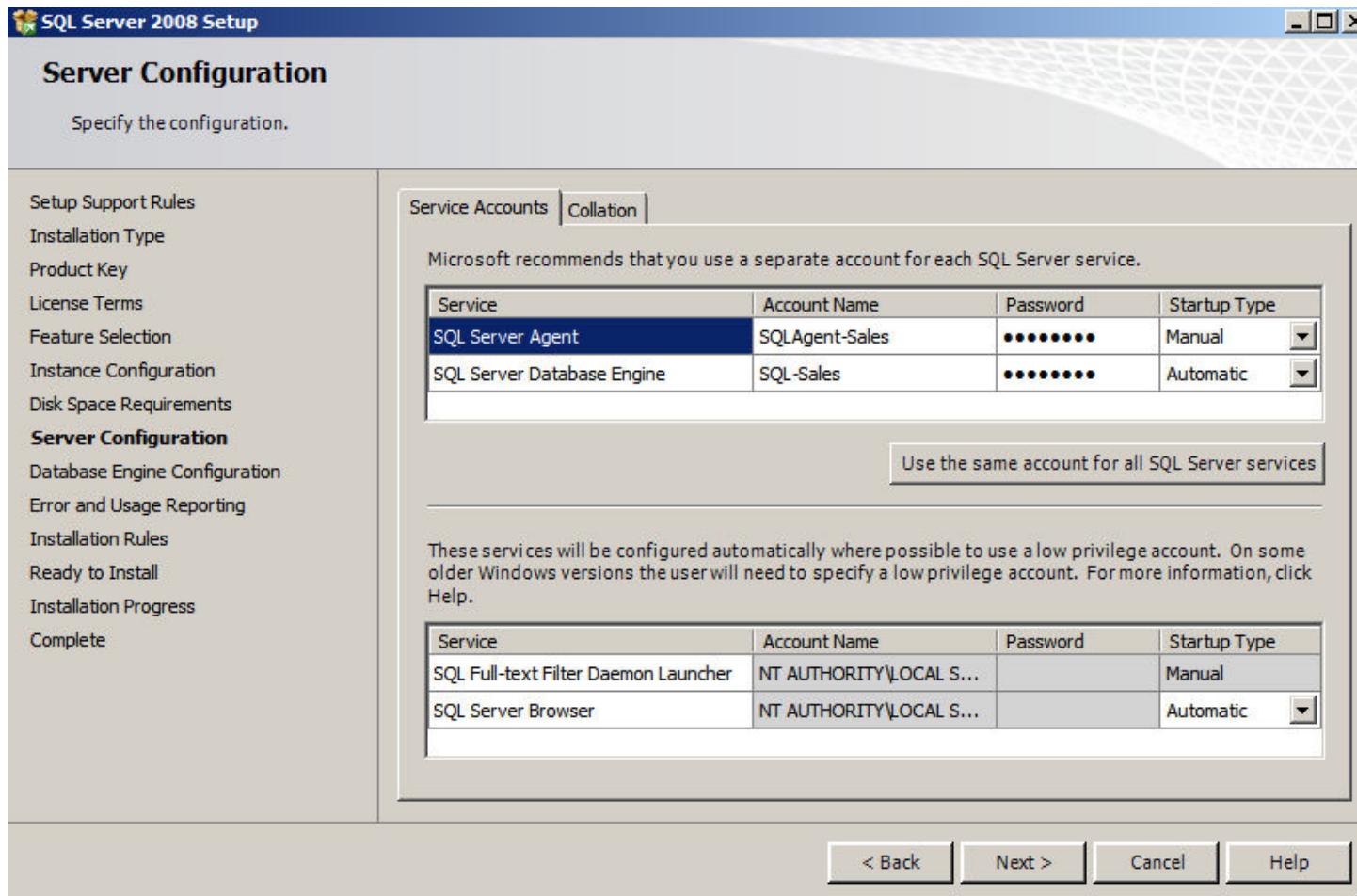
Features screen



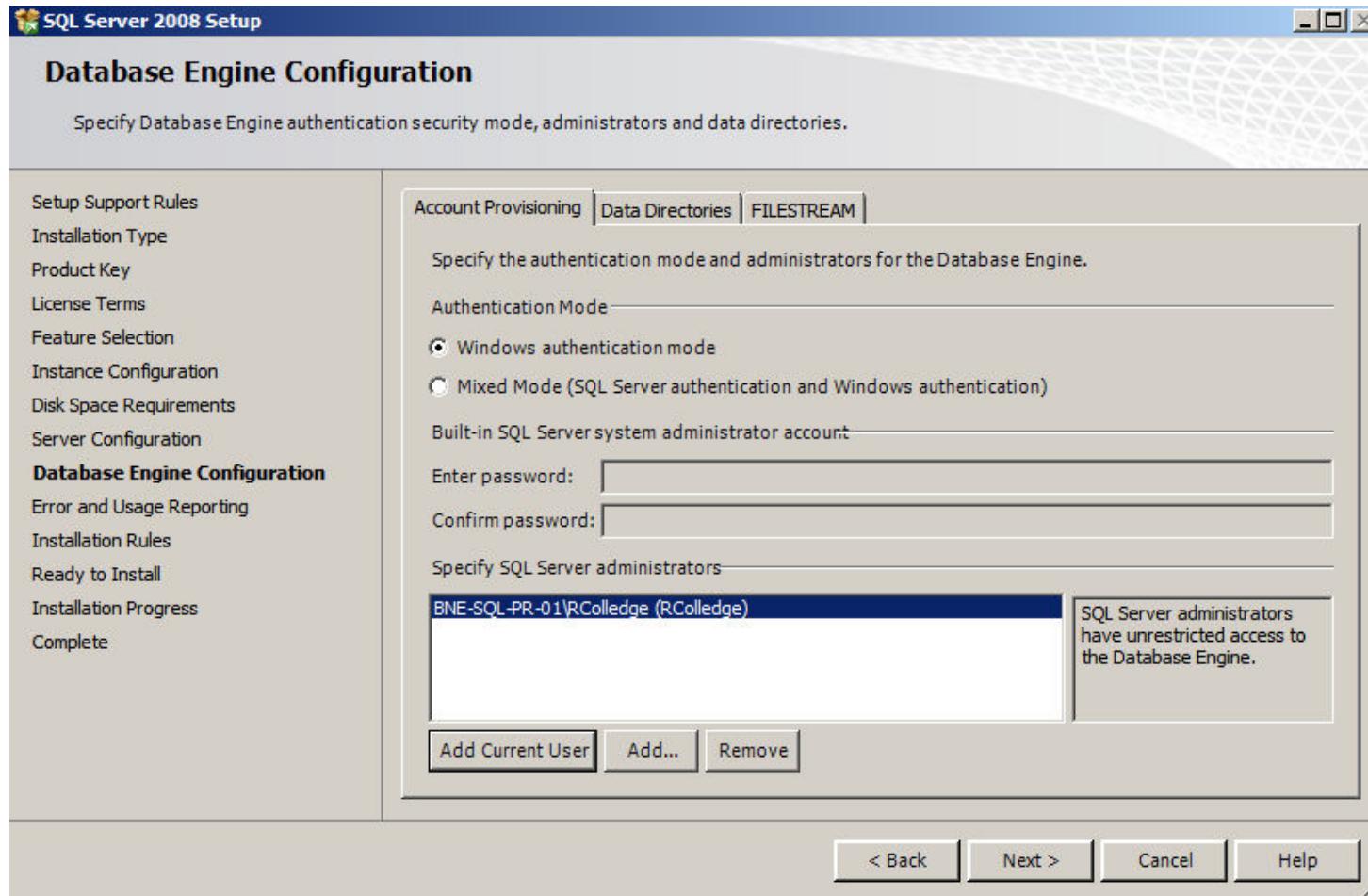
Select between default and named instance



Select service account and startup type for each service

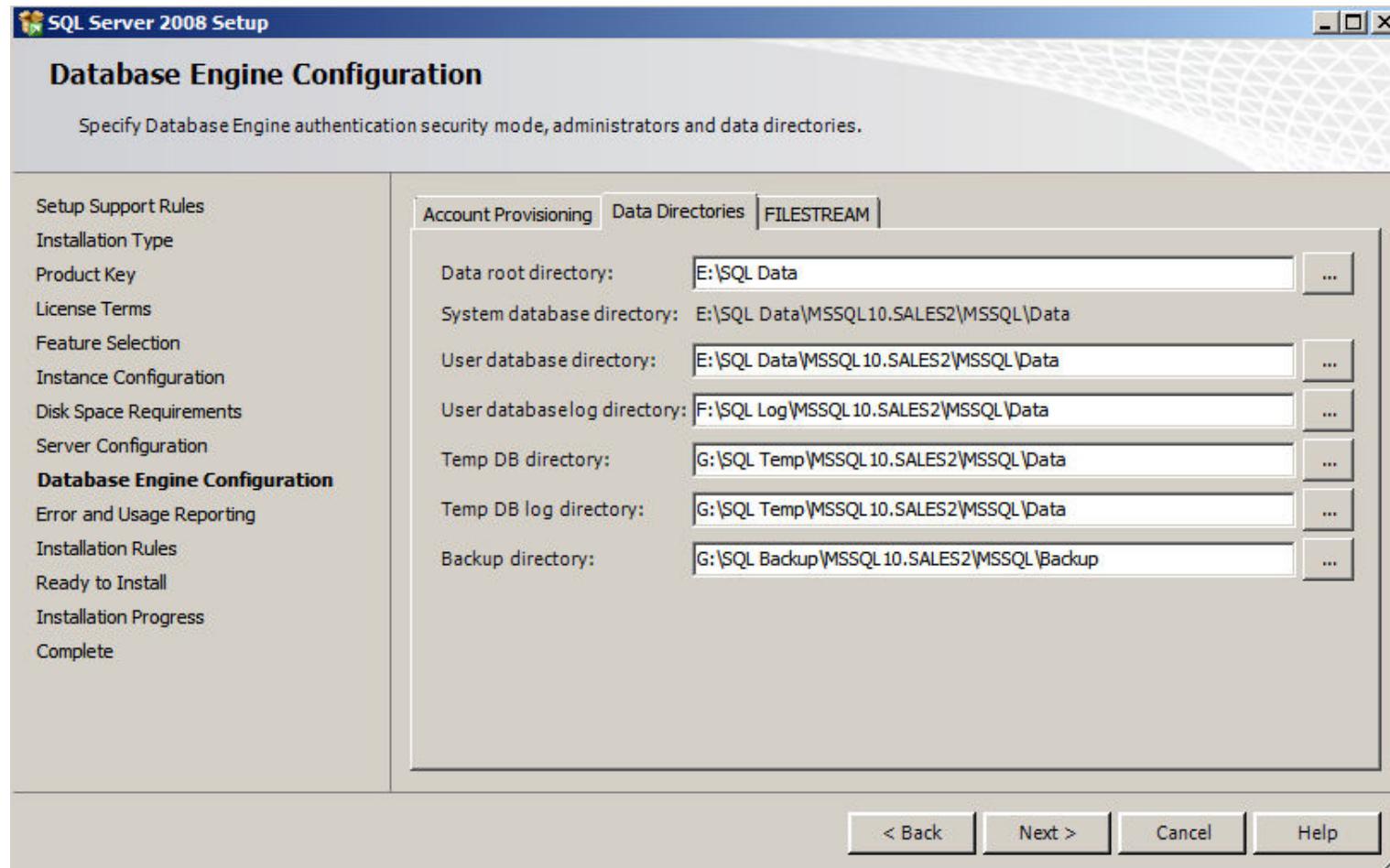


Select authentication mode



- Preferably mixed mode authentication
- Remember password for System Administrator
- Make at least current user administrator
- Make another user administrator

Specify directories for data, logs, backup, tempdb



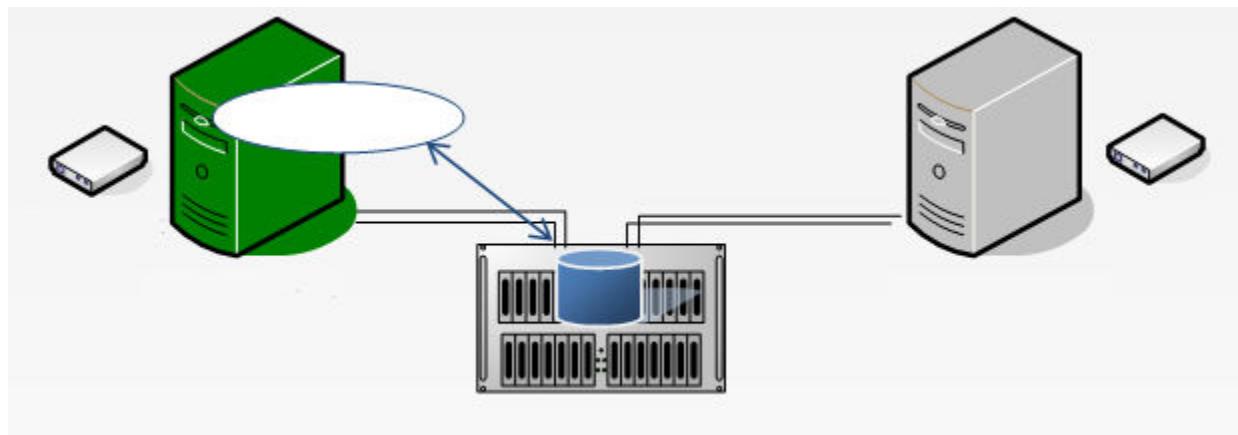
Upgrading to SQL Server 2008

- Microsoft released the SQL Server 2008 Upgrade Technical Reference Guide
- 490 pages - available for free download
- Upgrade Advisor wizard lets you inspect a SQL Server 2000 or 2005 instance before upgrading to detect issues
- upgrade Developer on laptop – simple
- upgrade existing, working, Enterprise on server - complicated

Planning and Installation

- Storage system sizing
- Physical server design
- Installing and upgrading SQL Server 2008
- **Failover clustering**

Failover clustering



Failover clustering

- commonly used high availability technique for SQL Server implementations
- in the event of failure of primary **server**, SQL Server instance automatically fails over to secondary server (in approx. 90 sec.)
- in either case, SQL instance will continue to be accessed using same virtual server name

CONFIGURATION

DataBase Design course notes 06
SQL Server Administration

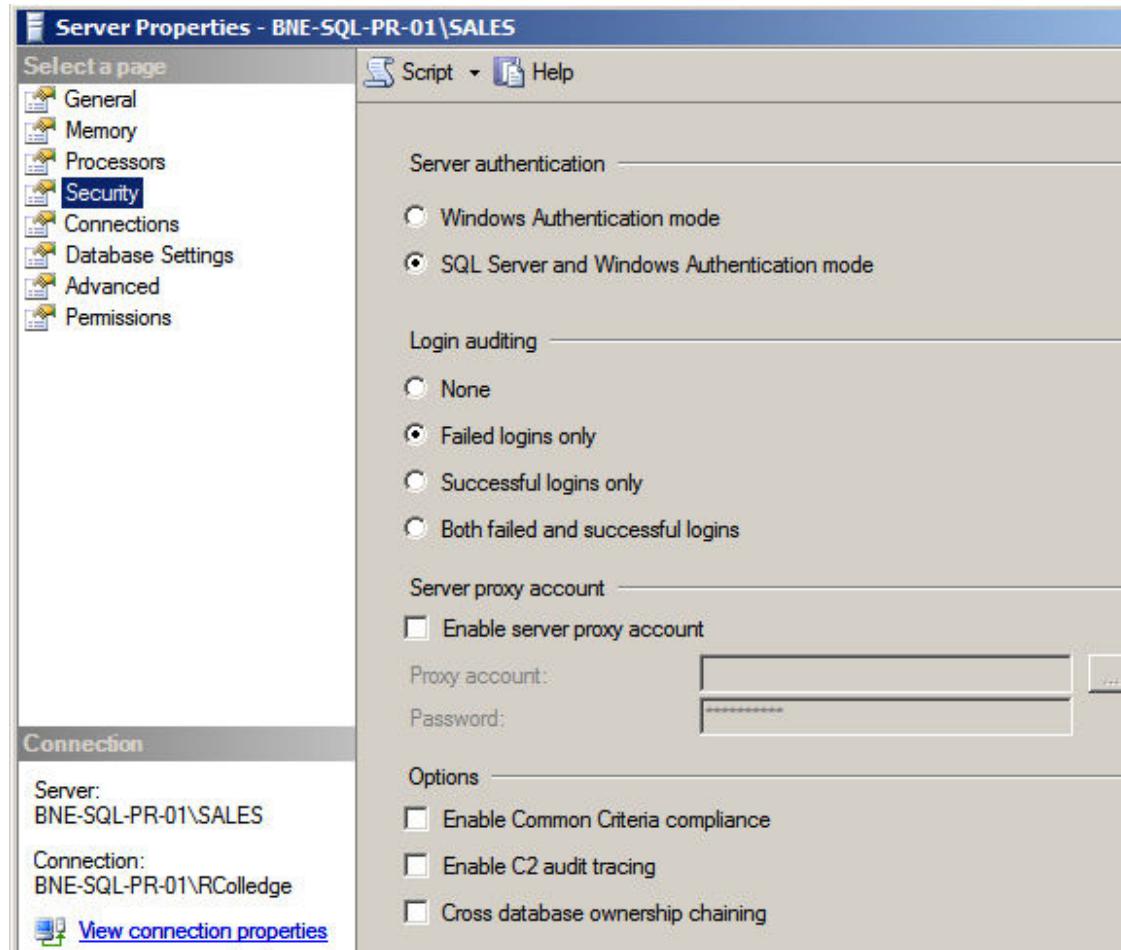
Configuration

- **Security**
- Configuring SQL Server
- Policy-based management
- Data management

Authentication mode

- Windows Authentication Mode
- SQL Server and Windows Authentication
 - commonly called Mixed Mode
- Change setting at any future point using `sp_configure`

Change setting Authentication mode



Windows Authentication mode

- It's the most secure
 - passwords are never sent across network,
 - expiration and complexity policies can be defined
- Account creation and management is centralized
- Windows accounts can be placed into groups, with permissions assigned at group level

- Unfortunately, many application vendors rely on SQL Server authentication
- in the worst examples, hard-code SA as the username in the connection properties
 - some with blank password!

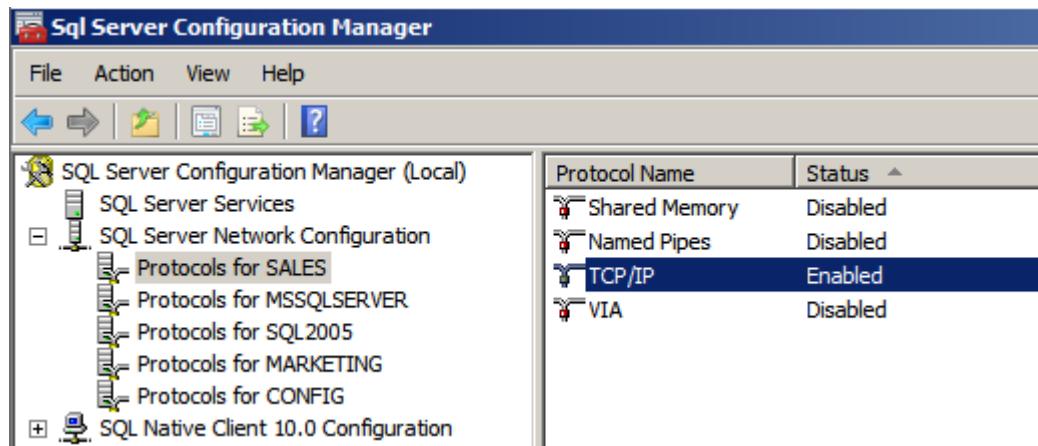
Networking Protocols

- Shared Memory
 - enabled by default on all editions
 - used purely for local connections
- Named Pipes
 - enabled by default for local connections only
 - with network connectivity over named pipes disabled

Networking Protocols

- TCP/IP
 - enabled by default for Enterprise,
 - disabled for Developer, Express
 - most widely used network protocol
 - provides better security and performance
- VIA
 - specialized protocol developed for use with specific hardware
 - disabled by default for all installations

SQL Server Configuration Manager



TCP ports

- Dynamic ports present a problem for firewall configuration
 - attempt to secure SQL Server instance behind firewall by only opening specific port number will obviously fail if port number changes, courtesy of dynamic port

TCP ports

- Static ports are best (and most secure) choice
 - avoid using ports currently (and commonly) used by other services and applications
 - IANA registration database,
<http://www.iana.org/assignments/port-numbers> resource lists registered port numbers for common applications, as well as “safe” ranges to use
- default instance listent on port 1433

Network encryption

- SQL Server 2008 introduces feature called Transparent Data Encryption (TDE)
 - when enabled, TDE automatically encrypts and decrypts data as it's read from and written to database without need for any application changes
- even with TDE enabled, other than initial login credentials, network transmission is unencrypted
- meaning packet sniffers could be used to intercept data
- for maximum data security, network transmission of SQL Server data can be encrypted using either Internet Protocol Security (IPSec) or Secure Sockets Layer (SSL)

Windows and DBA privilege separation

- in most cases, DBAs don't need to be local administrators to do their job
- not only that, they shouldn't be
- equally true, Windows administrators shouldn't be SQL Server sysadmins

Security in SQL Server

- Principal
- an entity requesting access to SQL Server object
- Securable
- for example, a user (principal) connects to database and runs select command against table (securable)
- database application with hundreds or thousands of principals and securables

Database Roles

- contains users (Windows or SQL Server logins) and is assigned permissions to objects (schema, tables, views, stored procedures, and so forth) within database

Create Role

- USE DataBase
- CREATE ROLE [DataBaseRole]
- -- Assign permissions to the role
- GRANT EXECUTE ON ... TO [DataBaseRole]
- GRANT SELECT ON ... TO [DataBaseRole]

Assigning logins to database role

- CREATE LOGIN [DataBaseLogin]
 - FROM WINDOWS WITH
 - DEFAULT_DATABASE=
- -- Create Database Users mapped to the Logins
- CREATE USER ... FOR LOGIN [DataBaseLogin]
- -- Assign the Users Role Membership
- EXEC sp_addrolemember [DataBaseRole],
[DataBaseLogin]

User/schema separation

- database schemas are distinct namespace
- each database user is created with default schema (dbo is default) and is used by SQL Server when no object owner is specified in commands
- objects within schema can be transferred to another schema
- ability to grant or deny permissions at schema level permits both powerful and flexible permissions structures
- sensitive tables could be placed in their own schema
- with only selected users granted access
- schema permissions can be granted to database roles

Fixed Database Roles

- `db_owner` role, which is highest level of permission in database
- commonly used fixed database roles are `db_datareader` and `db_datawriter` roles
 - used to grant read and add/delete/modify permissions respectively to all tables within database

SQL injection

- var city;
- city = request.form ("shippingCity");
- var sql = "select * from orders where ShipCity = " + city + """;
- intention of this code is that city variable will be populated with something

SQL injection

- what would happen if following value was entered in the shippingCity form field
- Cluj-Napoca'; select * from creditCards - -
- semicolon character marks end of command
- rest of input will run as separate query
- adding comments (--) characters to end of input, ensure any code added to end will be ignored
 - increasing chances of running injected code

Injection vulnerability analysis

- Microsoft Source Code Analyzer for SQL Injection tool
- described and downloadable from <http://support.microsoft.com/kb/954476>
- can be used to analyze and identify weaknesses in ASP pages that may be exploited as part of SQL injection attack

SQL injection

- more in the domain of application development
- well-established practices to prevent injection attacks

Prevent SQL injection attacks

- All user input should be validated by application before being submitted to SQL Server
 - ensure it doesn't contain any escape or comment characters: ' , ; and –
- Transact SQL statements shouldn't be dynamically built with user input appended
 - stored procedures should be used that accept input parameters
- Applications should anticipate not only injection attacks but also attacks that try to crash the system
 - for example, user supplying large binary file (MPEG, JPEG, and so forth) to form field designed to accept username or other
- Input should be validated at multiple levels

Configuration

- Security
- **Configuring SQL Server**
- Policy-based management
- Data management

Memory configuration

- most versions of SQL Server 2008 can address the amount of memory supported by underlying operating system
- 32-bit editions of SQL Server 2008 are constrained to 2GB of RAM unless configured with special settings

/3GB

- of the 4GB of RAM that a 32-bit system can natively address, 2GB is reserved by Windows, leaving applications with maximum of 2GB
- /3GB option in boot.ini file to limit Windows to 1GB of memory, enabling SQL Server to access up to 3GB

/PAE and AWE

- using Address Windowing Extensions (AWE) with /PAE option.
- Intel introduced 36-bit Physical Address Extensions (PAEs) in Pentium Pro
 - extra 4 bits enable applications to acquire physical memory above 4GB (up to 64GB) as non-paged memory dynamically mapped in 32-bit address space

/PAE and AWE

- Memory above 4GB can only be used by SQL Server data cache
- Analysis Services and Integration Services components aren't able to utilize memory accessed using PAE/AWE
- Unlike flat 64-bit environment, there's some overhead in mapping into AWE memory space in 32-bit systems

64-bit memory management

- full system RAM can be accessed by all SQL Server components without any additional configuration
- optional memory configuration for 64-bit systems is setting Lock Pages in Memory
 - beneficial in order to prevent Windows from paging out SQL Server's memory
 - certain actions such as large file copies can lead to Windows paging, or trimming, SQL Server's memory
 - leads to sudden dramatic reduction in performance

Setting minimum and maximum memory values

- when SQL Server starts, it acquires enough memory to initialize
- beyond which it acquires and releases memory as required
- values control upper limit to which SQL Server will acquire memory (maximum), and point at which it will stop releasing memory back to OS (minimum)

Setting minimum and maximum memory values

- By default, SQL Server's minimum and maximum memory values are 0 and 2,147,483,647, respectively
- allow SQL Server to cooperate with Windows and other applications
- on small systems with single SQL Server instance, default values will probably work fine
- on larger systems we need to give this a bit more thought

Amount of Memory to leave

- possible components require RAM:
 - Windows
 - Drivers for host bus adapter (HBA) cards, tape drives,
 - Antivirus software
 - Backup software
 - Microsoft Operations Manager (MOM) agents, or other monitoring software

CPU configuration

- When an instance of SQL server starts, it's created as an operating system process.
- Unlike simple application that performs series of serial tasks on single CPU, SQL Server is a complex application that must support hundreds or even thousands of simultaneous requests - SQL Server process creates *threads*
- Threads are assigned and balanced across available CPUs
- If thread is waiting on completion of task such as disk request, can schedule execution of other threads
- combination of multithreaded architecture and support for multi-CPU servers allows to support large number of simultaneous requests

Boost SQL Server Priority

- Threads created in Windows are assigned priority from 1 to 31
 - thread priority 0 reserved for operating system use
- Waiting threads are assigned to CPUs in priority order - higher-priority threads are assigned for execution ahead of lower-priority
- By default, SQL Server threads are created with “normal” priority level of 7
 - assigned and executed in timely manner without causing any stability issues

Boost SQL Server Priority option

- runs SQL Server threads at priority level of 13, higher than most other applications
 - sounds like a setting that should always be enabled
 - should only be used in rare circumstances
 - enabling this option has resulted in problems, such as not being able to shut down server and various other stability issues

Optionally, can define number of threads
for greater control

Default worker threads created

Number of CPUs	32-bit	64-bit
1–4	256	512
8	288	576
16	352	704
32	480	960

Server Properties - BNE-SQL-PR-01\SALES

Select a page Script Help

General
 Memory
 Processors
 Security
 Connections
 Database Settings
 Advanced
 Permissions

Enable processors

Processor	Processor Affinity	I/O Affinity
CPU0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Automatically set processor affinity mask for all processors
 Automatically set I/O affinity mask for all processors

Threads

Maximum worker threads:

Boost SQL Server priority
 Use Windows fibers (lightweight pooling)

Connection

Server: BNE-SQL-PR-01\SALES
Connection: BNE-SQL-PR-01\RColledge

[View connection properties](#)

- When system load is very high and all available threads are assigned to running tasks, system may become unresponsive until thread becomes available
- in such situations, dedicated administrator connection (DAC) can be used to connect to server and perform troubleshooting tasks

Maximum Degree of Parallelism

- controls the maximum number of CPUs that can be used in executing a single task
 - for example, large query may be broken up into different parts, with each part executing threads on separate CPUs - parallel query
- OLTP systems, use maximum MAXDOP setting of 8, including systems with access to more than 8 CPU cores
 - effort to split and rejoin query across more than 8 CPUs often outweighs benefits of parallelism

Server Properties - BNE-SQL-PR-01\SALES

Select a page

- General
- Memory
- Processors
- Security
- Connections
- Database Settings
- Advanced**
- Permissions

Script Help

Filestream Access Level Full access enabled

Allow Triggers to Fire Others True

Blocked Process Threshold 20

Cursor Threshold -1

Default Full-Text Language 1033

Default Language English

Full-Text Upgrade Option Import

Max Text Replication Size 65536

Optimize for Ad hoc Workloads False

Scan for Startup Procs False

Two Digit Year Cutoff 2049

Network Packet Size 4096

Remote Login Timeout 20

Cost Threshold for Parallelism 5

Locks 0

Max Degree of Parallelism 0

Query Wait -1

Connection

Server: BNE-SQL-PR-01\SALES

Connection: BNE-SQL-PR-01\RColledge

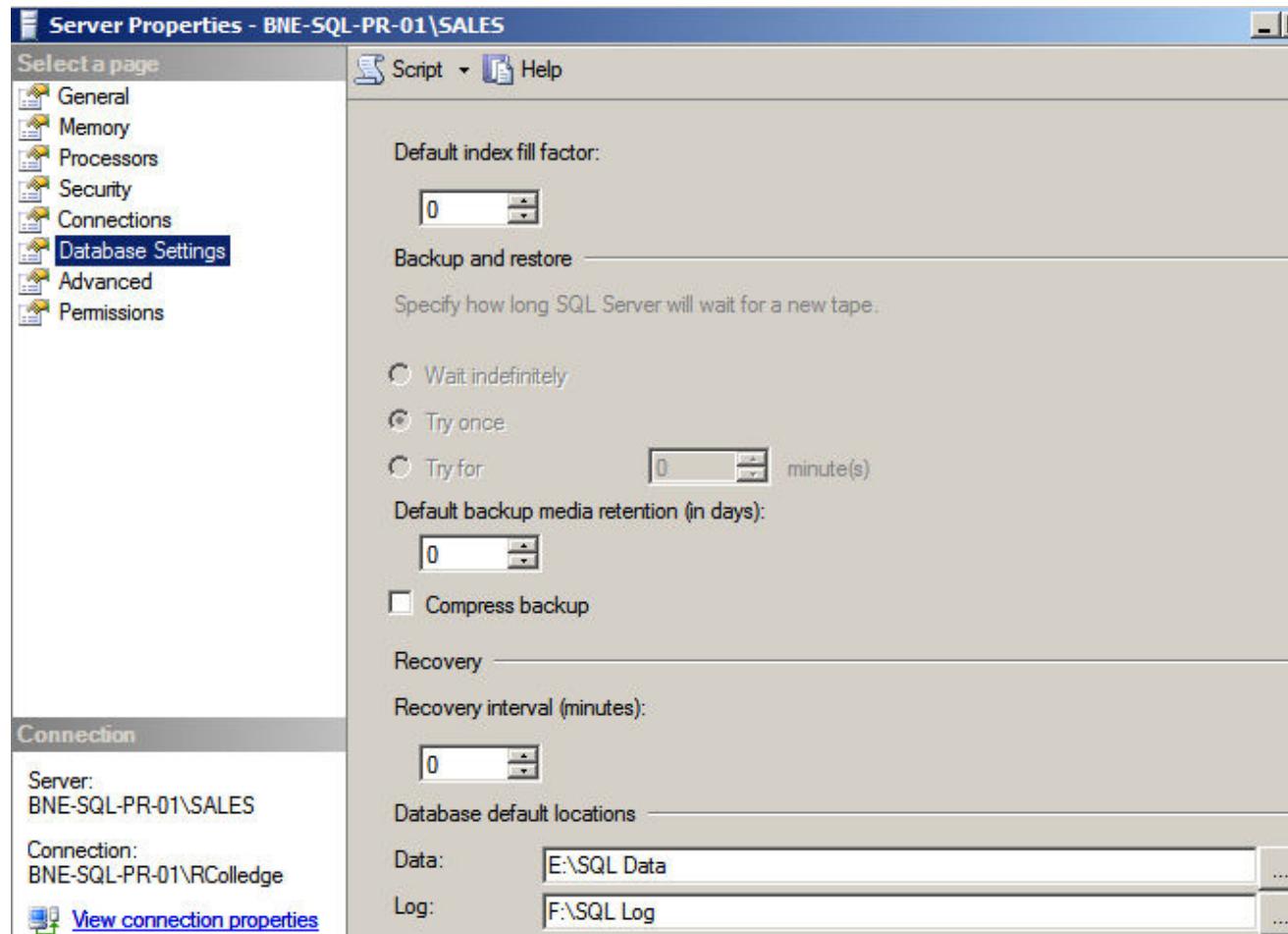
Server configuration - Fill factor

- when index is created or rebuilt, numeric value between 0 and 100, determines how full each index page will be
- benefits of full pages are that less I/O is required to fulfill an index scan/seek
- downside comes when data is inserted, full page needs to be split in order to allow new data to be inserted in index order
- best fill factor is determined by rate of data modification
- database with high volume of updates may benefit from lower fill factor

Server configuration - Fill factor

- fill factor value is only used when index is first created or rebuilt
- after that, page will fill/split as a natural consequence of data inserts and updates
- default server fill factor of 0 (equivalent to 100) - like all other configuration settings, default value for recovery interval is best value in almost all cases

Server configuration



Server configuration - Locks

- By default, SQL Server reserves sufficient memory for a lock pool consisting of 2,500 locks
- when number of used locks reaches 40% of size of the (non-AWE) buffer pool, SQL Server will consider lock escalation
 - will convert number of row locks to single page or table lock - therefore reducing memory impact
- when 60% of buffer pool is used for locks, new lock requests will be denied, resulting in error

Server configuration - Query Wait

- when query is submitted for execution, first checks to see if there's already cached query plan it can reuse
- if no such plan exists, new plan needs to be created
- avoiding this process through query parameterization is key performance tuning goal
- estimated cost of plan determines length of time that SQL Server will wait for resources before query times out

Server configuration

Query Governor Cost Limit

- SQL Server estimates cost of query
- comparing it to Query Governor Cost Limit
- if enabled, Query Governor Cost Limit option is used to prevent queries from running whose estimated cost exceeds configured value, specified in seconds
- By default, this option is disabled

- *Hint – design views - there is already plan to execute query*
- *view 20-30% better then ad-hoc query*

Server configuration

User Connection

- By default, SQL Server will allow an unlimited number of user connections, within the constraints of its available resources
- Setting non-zero value allows control over the number of concurrent connections
 - once number of connections is exceeded (bearing in mind user or application can have multiple connections), new connections will be denied with the exception of dedicated administrator connection

Server Properties - BNE-SQL-PR-01\SALES

Select a page

Connections

Maximum number of concurrent connections (0 = unlimited):

Use query governor to prevent long-running queries

Default connection options:

- implicit transactions
- cursor close on commit
- ansi warnings
- ansi padding
- ANSI NULLS
- arithmetic abort
- arithmetic ignore

Connection

Server: BNE-SQL-PR-01\SALES

Connection: BNE-SQL-PR-01\RColledge

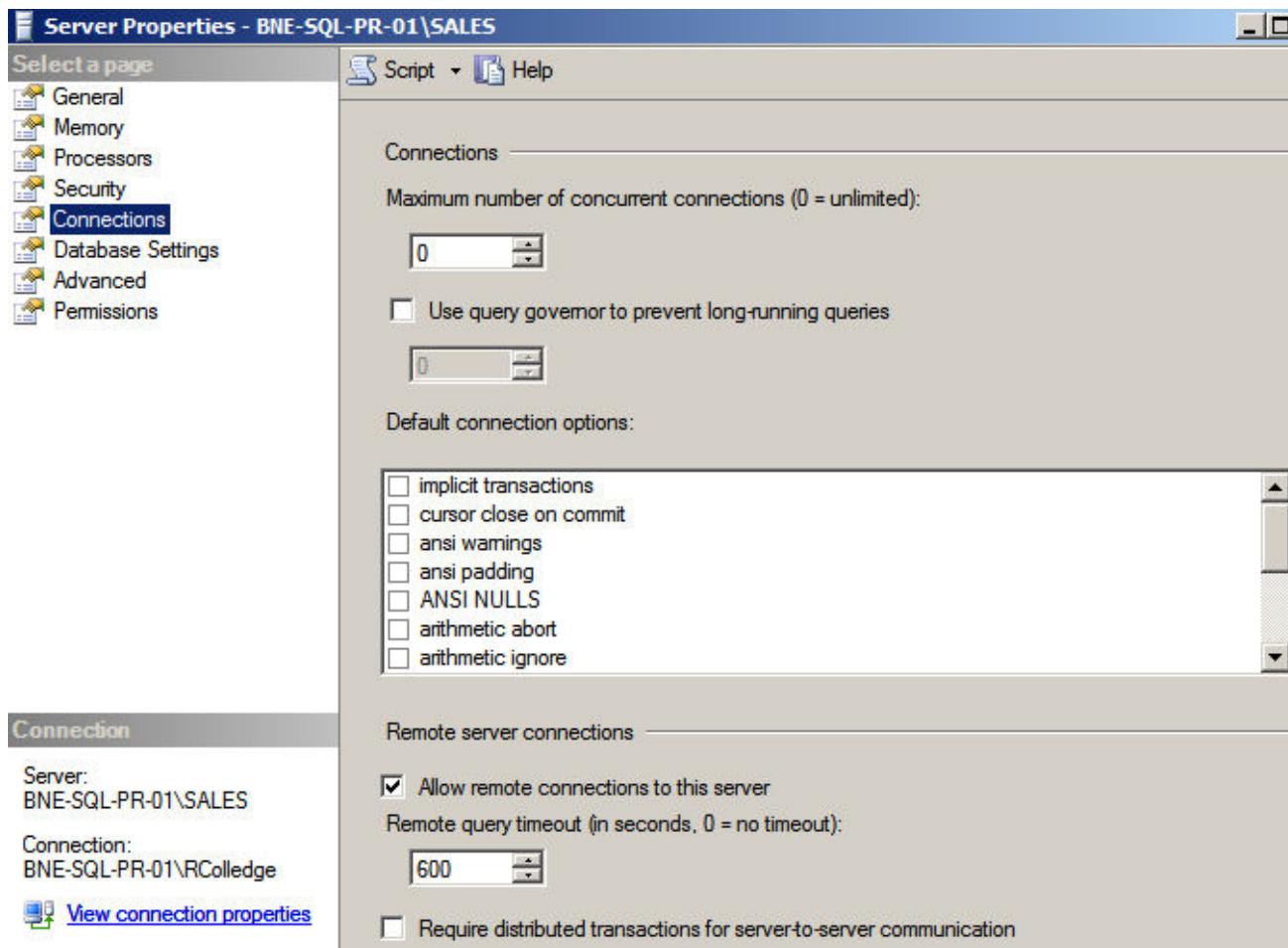
[View connection properties](#)

Remote server connections

Allow remote connections to this server

Remote query timeout (in seconds, 0 = no timeout):

Require distributed transactions for server-to-server communication



Operating system configuration

Running services

- very easy during installation to select all features on
 - chance that they may be required in future
- as result, people often end up with Analysis Services, Reporting Services, Integration Services, ...
 - create Windows Services that run on startup
- there are other Windows Services that may be running that are possibly not required
 - which ones are candidates for disabling?
 - but IIS is worth special mention
- Disabling nonessential services good from performance perspective as well as effective security practice

Operating system configuration

Processor scheduling

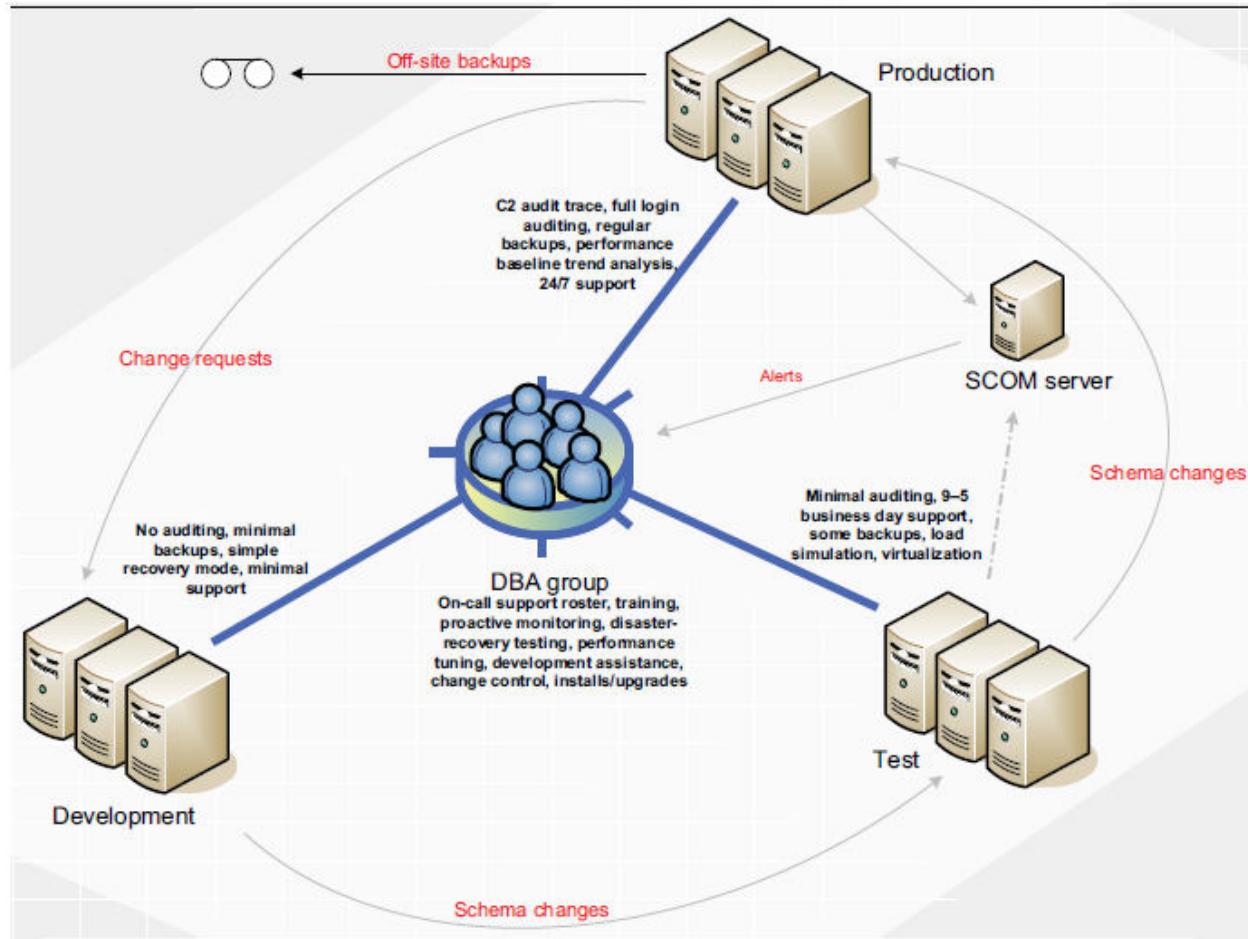
- accessed via Control Panel, advanced options of System Properties let you choose between adjusting processor resources to favor foreground applications or background services
- as background service, SQL Server obviously benefits from this option being set to Background Services

Configuration

- Security
- Configuring SQL Server
- **Policy-based management**
- Data management

- scripting technologies
 - SQL Server Management Objects (SMOs)
 - PowerShell
- Systems Center Operations Manager (SCOM) typically used for monitoring disk space and event logs
- **Enterprise DBA challenges**

typical enterprise environment



Enterprise DBA challenges

- Production systems should be secured with least privilege principle
- in contrast with development environments in which changes originate
 - when developed code reaches test and production systems, certain functions often fail as a result of security differences
- coordinate environment configuration across enterprise

Enterprise DBA challenges

- Databases in production environments (should) use full recovery model along with regular transaction log backups
- in development and test environments that don't perform transaction log backups, full recovery model may cause transaction log to consume all available disk space
- must match environments with backup profiles and recovery model settings

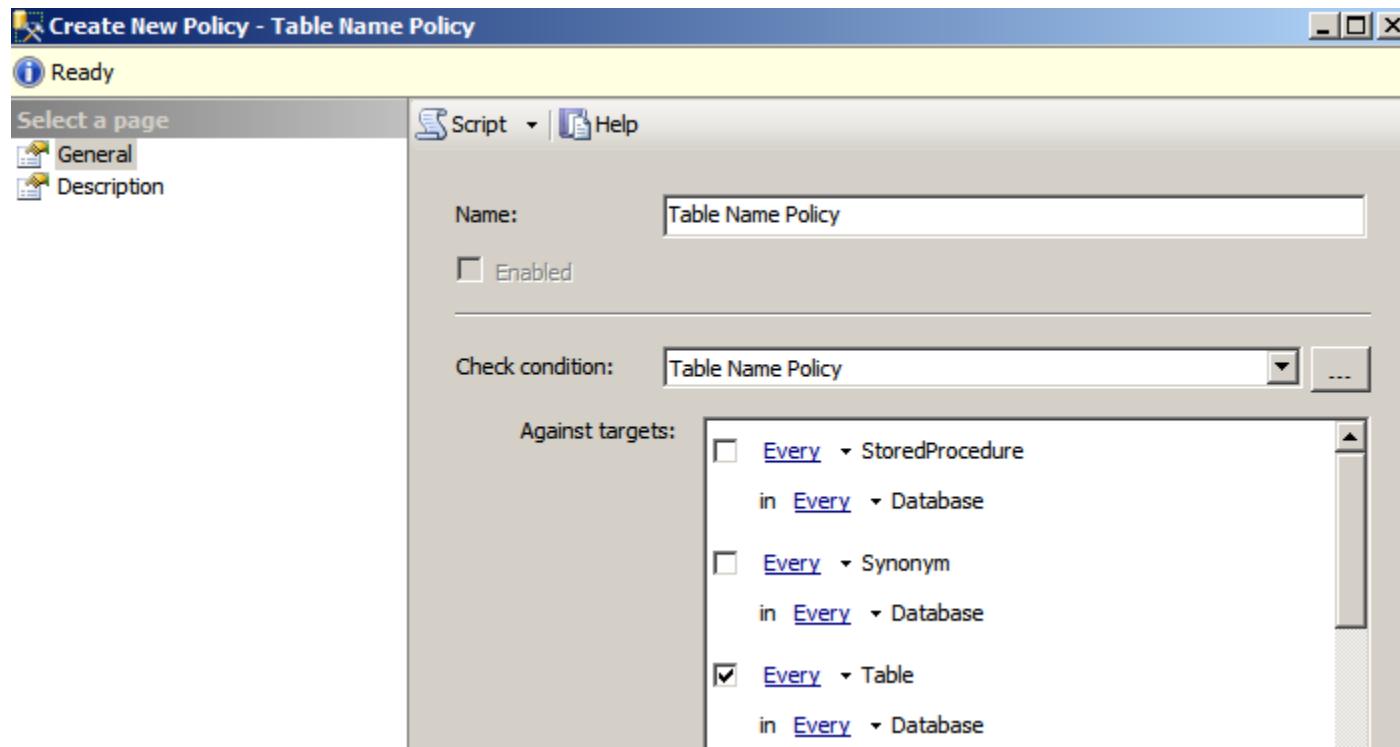
Enterprise DBA challenges

- In sites with a range of DBMS products that require support, it's often the case that requirement for a broad range of skills prevents expertise in any one area, making correct and consistent configuration even more difficult
- In poorly configured environments, time taken to troubleshoot highly visible production problems often prevents important proactive maintenance required for ongoing environment performance, security, and stability

Policy-based management

Target

- entity managed by a policy
- depending on policy, targets may be SQL Server instances, databases, tables, and so forth
- in next example, target chosen for table name policy is every table in every database



Policy-based management

Facet

- name given to a group of configurable properties that are appropriate for a certain number of targets
- in next example Configuration facet, applicable to Server target, contains properties such as
 - DatabaseMailEnabled
 - CLRIntegrationEnabled
 - XPCmdShellEnabled

Facet Properties - Surface Area Configuration

Ready

Select a page:

- General
- Dependent Policies
- Dependent Conditions

Script Help

Description: Surface area configuration for features of the Database Engine. Only the features required by your application should be enabled. Disabling unused features helps protect your server by

Applicable target types: Server

Properties:

OleAutomationEnabled	The OLE Automation extended stored procedures (XPs) allow Transact-SQL batches, (▲)
RemoteDocEnabled	A dedicated administrator connection (DAC) allows an administrator to connect to a se
ServiceBrokerEndpointActive	Service Broker provides queuing and reliable messaging for the Database Engine. Ser
SoapEndpointsEnabled	The SOAP endpoint can be in either a started, stopped or disabled state. Returns TRL
SqlMailEnabled	SQL Mail supports legacy applications that send and receive e-mail messages from the
WebAssistantEnabled	Web Assistant stored procedures, which generate HTML files from SQL Server databa
XPCmdShellEnabled	xp_cmdshell creates a Windows process that has same security rights as the SQL Ser

Connection: BNE-SQL-PR-01\SALES [BNE-SQL-PR-01\RColledge]
[View connection properties](#)

Progress: Ready

Policy-based management

Condition

- created to specify required state of one or more facet properties
- in next example, condition contains the required state of ten properties belonging to Surface Area Configuration facet

Open Condition - Surface Area Configuration for Database Engine 2008 Features

Ready

Select a page:

- General
- Description
- Dependent Policies

Script Help

Name: Surface Area Configuration for Database Engine 2008 Features

Facet: Surface Area Configuration

Expression:

AndOr	Field	Operator	Value
>	@AdHocRemoteQueriesEnabled	=	False
AND	@ClrIntegrationEnabled	=	False
AND	@DatabaseMailEnabled	=	False
AND	@OleAutomationEnabled	=	False
AND	@RemoteDادEnabled	=	False
AND	@ServiceBrokerEndpointActive	=	False
AND	@SoapEndpointsEnabled	=	False
AND	@SqlMailEnabled	=	False
AND	@XPCmdShellEnabled	=	False

* Click here to add a clause

AdHocRemoteQueriesEnabled

The OPENROWSET and OPENDATASOURCE functions support ad hoc connections to remote data sources without linked or remote servers. Enable these functions only if your applications and scripts call them.

Connection

LNE-SQL-PR-01\GALES
[LNE-SQL-PR-01]\Colledge]

[View connection properties](#)

Configuration

- Security
- Configuring SQL Server
- Policy-based management
- **Data management**

Database file

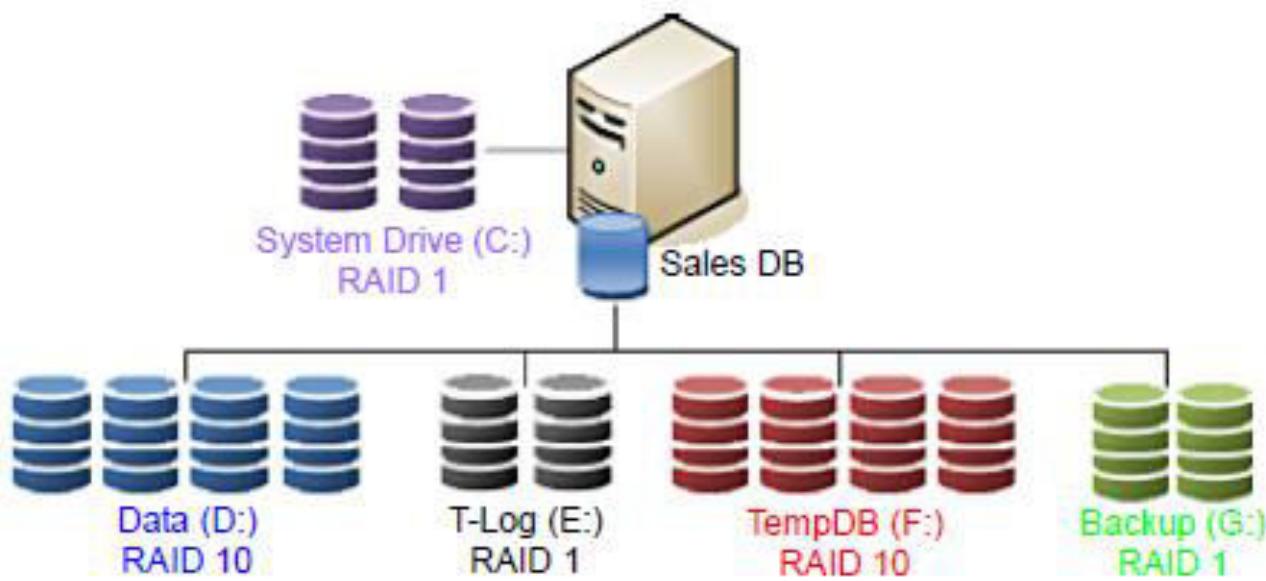
- Primary data file
 - by default only data file, contains system tables and information on all files within database
 - by default file has an .mdf extension
- Secondary data file
 - usually have an .ndf extension
 - optional files that can be added to database for performance and/or administrative benefits
 - database can contain one or more secondary files

Database file

- Filegroups
 - every database contains primary filegroup
 - containing at least primary data file
 - possibly all secondary data files
 - unless other filegroups are created and used
 - logical containers group one or more data files
- Transaction log file
 - typically using .ldf extension
 - records details of each database modification
 - used for transaction log, replication, database mirroring, and recovery

Volume separation

- By default, database is created with single data and transaction log file
- unless specified, both of these files will be created in same directory
 - with default size and growth rates inherited from model database
- important database file configuration task, particularly for databases with direct-attached storage, is to provide separate physical RAID-protected disk volumes for data, transaction log, tempdb, and backup files



Transaction Log file

- Unlike random access to data files, transaction logs are written sequentially
- if disk is dedicated to single database's transaction log, disk heads can stay in position writing sequentially
 - increasing transaction throughput
- disk that stores combination of data and transaction logs won't achieve same levels of throughput
 - given that disk heads will be moving between conflicting requirements of random data access/updates and sequential transaction log entries
- for database applications with high transaction rates, separation of data and transaction logs is crucial

Backup files

- common (and recommended) technique, is to back up databases to disk files and archive disk backup files at later point in day
- most optimal method for doing this is to have dedicated disk(s) for purpose of storing backups

Backup files

- Disk protection
 - consider case where database files and backup files are on the same disk
 - should disk fail, both database and backups are lost
- Increased throughput
 - substantial performance gains come from multiple disks working in unison
 - during backup, disks storing database data files are dedicated to reading files, and backup disks are dedicated to writing
 - having both data and backup files on same disk will substantially slow process

Backup files

- Cost-effective
 - backup disks may be lower-cost, higher-capacity SATA disks
 - with data disks being more expensive, RAID-protected SCSI or SAS disks
- Containing growth
 - last thing you want is situation where backup consumes all space on data disk
 - effectively stopping database from being used

TempDB

- By providing dedicated disks for tempdb, impact on other databases will be reduced while increasing performance for databases heavily reliant on it

Windows

- SQL data files shouldn't be located on same disks as Windows system and Program Files
- best way of ensuring this is to provide dedicated disks for SQL Server data, log, backups and tempdb

Multiple data files

- common discussion point on database file configuration is based on number of data files that should be created for database
- for example, should 100GB database contain single file, four 25GB files, or some other combination?

Performance

- common performance-tuning recommendation is to create one file per CPU core
 - for example, SQL Server instance with access to two quad-core CPUs should create eight database files
- having multiple data files is certainly recommended for tempdb database
- it isn't necessarily required for user databases

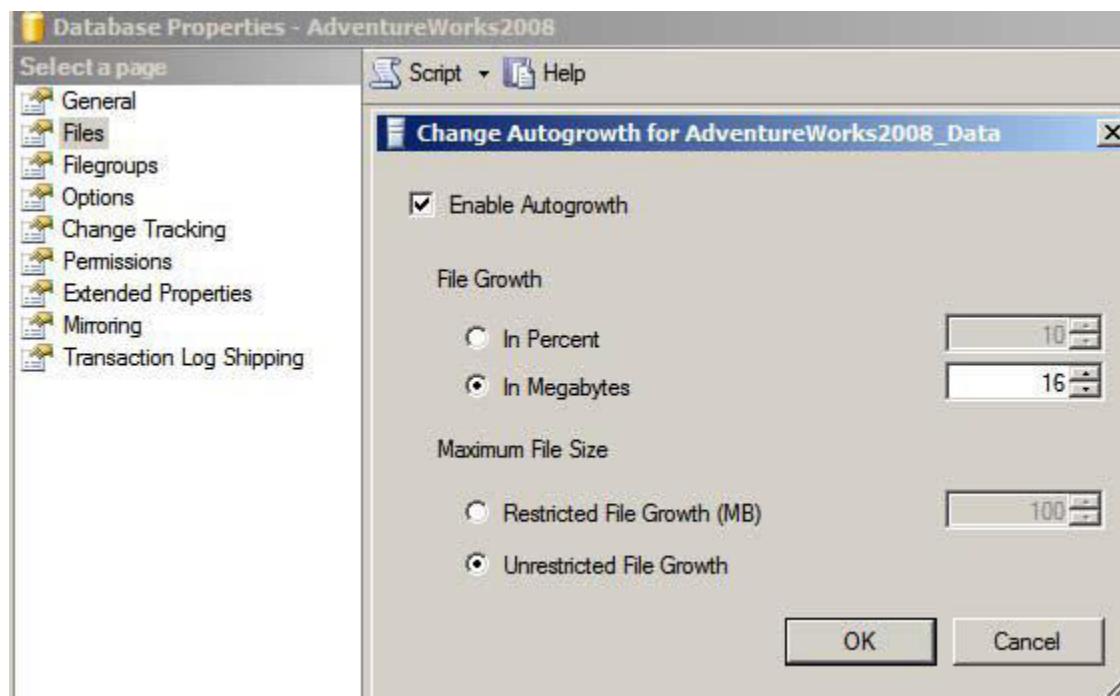
Performance

- one file per CPU core suggestion useful in avoiding allocation contention issues
- tempdb database used for creation of short-term objects, by all databases within instance
- potentially very large number of objects being allocated
- using multiple files enables contention on single allocation bitmap to be reduced, resulting in higher throughput

Manageability

- Consider database configured with single file stored on 1TB disk partition with database file currently 900GB
 - migration project requires database to be moved to new server that has been allocated three 500GB drives
 - obviously 900GB file won't fit into any of three new drives
- various ways of addressing this problem, but avoiding it by using multiple smaller files is arguably the easiest
- multiple smaller files enable additional flexibility in overcoming number of other storage-related issues
- if disk drive is approaching capacity, it's much easier (and quicker) to detach a database and move one or two smaller files than it is to move single large file

- transaction log files are written to in a sequential manner
- although it's possible to create more than one transaction log file per database, there's no benefit in doing so



- SQL Server offers features that enable databases to continue running with very little administrative effort
- such features often come with downsides
- Enable Autogrowth option
- enables database file to automatically expand when full
- Despite lower administration overhead, option should not be used in
- place of database presizing and proactive maintenance routines

- every time the file grows
- all activity on file is suspended until growth operation is complete

Filegroups

- logical containers for database disk files
- default configuration for new database is single filegroup called primary
 - contains one data file in which all database objects are stored
- common performance-tuning recommendation is to create tables on filegroup and indexes on another
 - with each filegroup containing files on dedicated disks
 - for example
 - Filegroup 1 (tables) contains files on RAID volume containing 10 disks
 - Filegroup 2 (indexes) contains files on separate RAID volume, also containing 10 disks

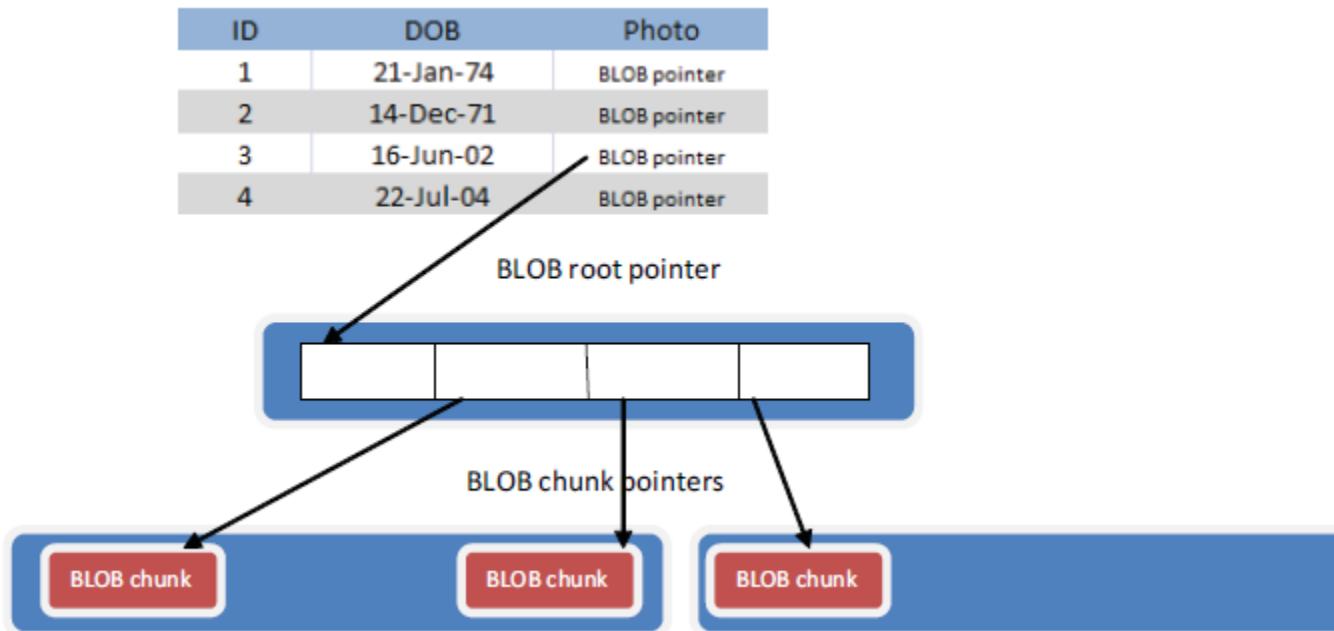
BLOB storage with FileStream

- binary large objects (BLOBs) such as video, images, or documents (PDFs, docs, and so forth)
 - store BLOB object within database in *varbinary(max)* column
 - store BLOBs in file system files, with link to file (hyperlink/path) stored in table column
- SQL Server 2008 introduces method FileStream
- lets you combine benefits of both of previous methods while avoiding their drawbacks

BLOBS in database

- storage engine designed and optimized for storage of normal relational data
- fundamental design component is 8K page size
- all but smallest BLOBs exceed this size
- to get around the 8K limitation, SQL Server breaks BLOB up into 8K chunks and stores them in B-tree structure
 - with pointer to root of tree stored in record's BLOB column

BLOBS in database



BLOBS in database

- -- Insert a jpg file into table using OPENROWSET
- INSERT INTO clients (ID, DOB, Photo)
- SELECT 1, '19 Jan 1964', BulkColumn
- FROM OPENROWSET (Bulk 'F:\photos\client_1.jpg',
SINGLE_BLOB) AS blob
- BLOBS are transactionally consistent
- for databases with large numbers of BLOBs, or even
moderate amounts of very large BLOBs, database size
can become massive and difficult to manage
- performance can suffer

BLOBS in file system

	id	dob	photolink
1	1	1977-01-18	f:\photos\client1.jpg
2	2	1905-12-01	f:\photos\client2.jpg
3	3	1967-03-22	f:\photos\client3.jpg
4	4	1991-09-30	f:\photos\client4.jpg

BLOBS in file system

- Windows NTFS is much better at file storage than SQL Server
- data in database is no longer transactionally consistent with BLOB files
- database backups aren't guaranteed to be synchronized with BLOBS
 - unless database is shut down for period of backup
 - which isn't an option for any 24/7 system

BLOB with FileStream data

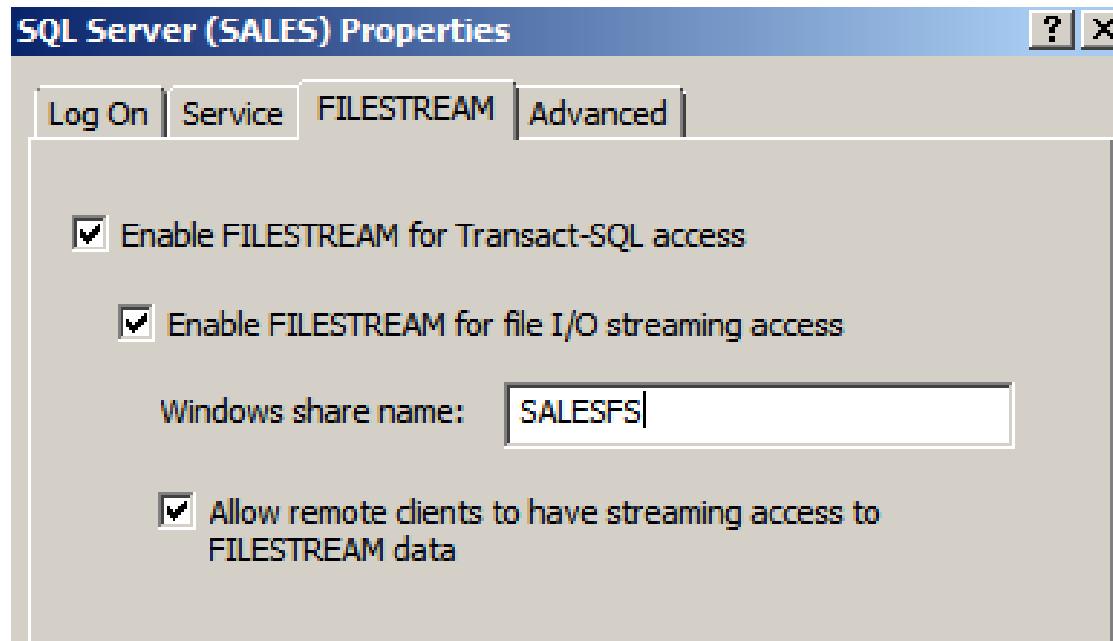
- BLOBs can be stored in file system
 - size of each BLOB is limited only by NTFS volume size limitations
 - overcomes 2GB limit
- Full transactional consistency exists between BLOB and database
- BLOBs are included in backup and restore operations

BLOB with FileStream data

- BLOB objects are accessible via both T-SQL and NTFS streaming APIs
- Superior streaming performance is provided for large BLOB types such as MPEG video
- Windows system cache is used for caching BLOB data, thus freeing up SQL Server buffer cache required for previous in-database BLOB storage techniques

Enable FileStream

SQL Server Configuration Manager



Using FileStream

- ensure there is FileStream filegroup
- Create database with SalesFileStreamFG filegroup by specifying CONTAINS FILESTREAM
- use directory name to specify location of FileStream data
- for optimal performance and minimal fragmentation, disks storing FileStream data should be formatted with 64K allocation unit size, and be placed on disk(s) separate from both data and transaction log files

Using FileStream

- -- Create a database with a FILESTREAM filegroup
- CREATE DATABASE [SALES] ON PRIMARY
 - (NAME = Sales1
 - , FILENAME = 'M:\MSSQL\Data\salesData.mdf')
 - , FILEGROUP [SalesFileStreamFG] CONTAINS
FILESTREAM
 - (NAME = Sales2
 - , FILENAME = 'G:\FSDATA\SALES')
 - LOG ON
 - (NAME = SalesLog
 - , FILENAME = 'L:\MSSQL\Data\salesLog.Idf')

Using FileStream

- -- Create a table with a FILESTREAM column
- CREATE TABLE Sales.dbo.Customer(
• [CustomerId] INT IDENTITY(1,1) PRIMARY KEY
• , [DOB] DATETIME NULL
• , [Photo] VARBINARY(MAX) FILESTREAM NULL
• , [CGUID] UNIQUEIDENTIFIER NOT NULL
 ROWGUIDCOL UNIQUE DEFAULT NEWID()
•) FILESTREAM_ON [SalesFileStreamFG];

Using FileStream

- `INSERT INTO Sales.dbo.Customer (DOB, Photo)`
- `VALUES ('21 Jan 1975', CAST ('{Photo}' as varbinary(max)));`
- no obvious correlation between database records and FileStream file or directory names
- it's not the intention of FileStream to enable direct access to resulting FileStream data using Windows Explorer

FileStream Limitations

- Database mirroring can't be enabled on databases containing FileStream data
- Database snapshots aren't capable of including FileStream data
- FileStream data can't be encrypted
- Depending on BLOB size and update pattern, you may achieve better performance by storing BLOB inside database
 - particularly for BLOBs smaller than 1MB
 - and when partial updates are required (for example, when you're updating small section of large document)

DBA OPERATIONS

DataBase Design course notes 06
SQL Server Administration

Backup and recovery

DBA OPERATIONS

Backup types

- Unless you're a DBA, you'd probably define a database backup as a complete copy of a database at a given point in time
- While that's one type of database backup, there are many others

Consider a very large database used 24/7

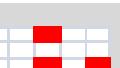
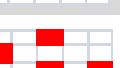
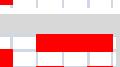
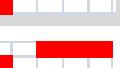
- How long does backup take
 - what impact does it have on users?
- Where are the backups stored
 - what is the media cost?
- How much of database changes?
- If the database failed partway through day
 - how much data would be lost if only recovery point was previous night's backup?

Full backup

- simplest, most well understood type of database backup
- can perform backups while database is in use and is being modified by users
- such backups are known as online backups
- when full backup is restored, changes since the full backup are lost

Differential backup

- full backups on nightly basis may not be possible (or desirable) for variety of reasons
 - if only small percentage of database changes on daily basis
 - full nightly backup are questionable, considering storage costs and users impact
- differential backup includes all database changes since last full backup

Day	Backup Type	Includes ...	Contents
Sunday	Full	Everything	
Monday	Differential	Changes since Sunday	
Tuesday	Differential	Changes since Sunday	
Wednesday	Differential	Changes since Sunday	
Thursday	Differential	Changes since Sunday	
Friday	Differential	Changes since Sunday	
Saturday	Differential	Changes since Sunday	

Differential backup

- when restoring differential backup, corresponding full backup, known as the base backup, needs to be restored with it
- if we needed to restore database on Friday morning, full backup from Sunday, along with differential backup from Thursday night, would be restored

- Frequent transaction log backups reduce exposure to data loss
 - If transaction log disk is completely destroyed, then all changes since last log backup will be lost

Disaster recovery plan

- considers wide variety of potential disasters
 - from small events such as corrupted log files and accidentally dropping a table, right through to large environmental disasters such as fires and earthquakes
- well-documented and well-understood backup and restore plan

Disaster recovery plan

- for disk backups, retention period is dependent on backup model
 - for example, if weekly full, nightly differential system is in place, then weekly backup would need to be retained on disk for whole week for use with previous night's differential backup
 - if disk space allows, then additional backups can be retained on disk as appropriate

Online restores

- Consider very large database in use 24/7
- advantages of using multiple filegroups is that we're able to back up individual filegroups instead of entire database
- minimizes user impact of backup operation
- enables online piecemeal restores, whereby parts of database can be brought online and available for user access while other parts are still being restored
- in contrast, traditional restore process would require users to wait for entire database to restore before being able to access it

Database snapshots

- common step in deploying changes to database is to take backup of database prior to change
- backup can then be used as rollback point if the change / release is deemed failure
- consider very large database
 - how long would backup and restore take either side of the change?
- Database snapshots can be used
 - process typically taking only a few seconds

High availability with

Database Mirroring

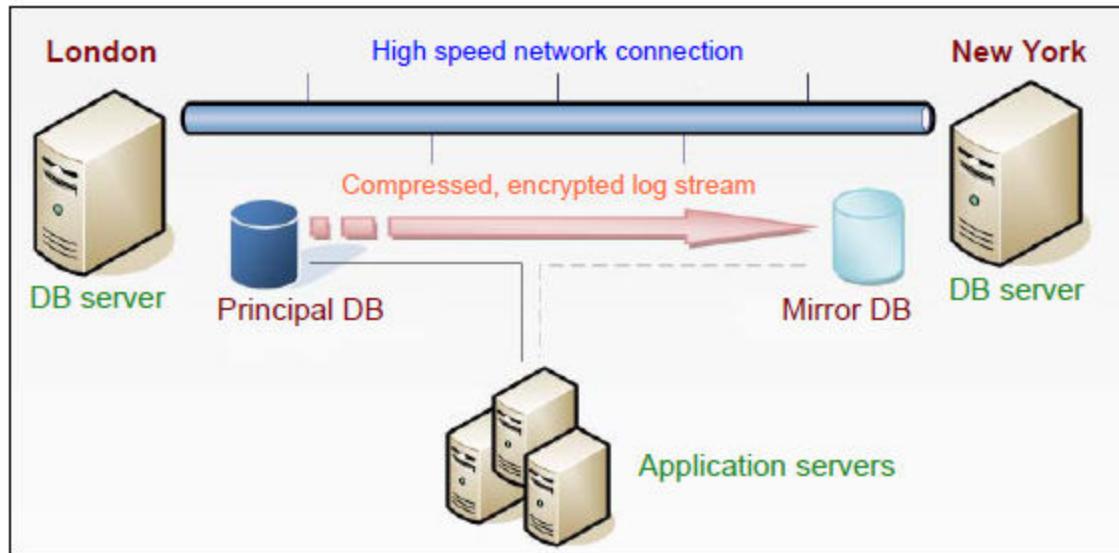
DBA OPERATIONS

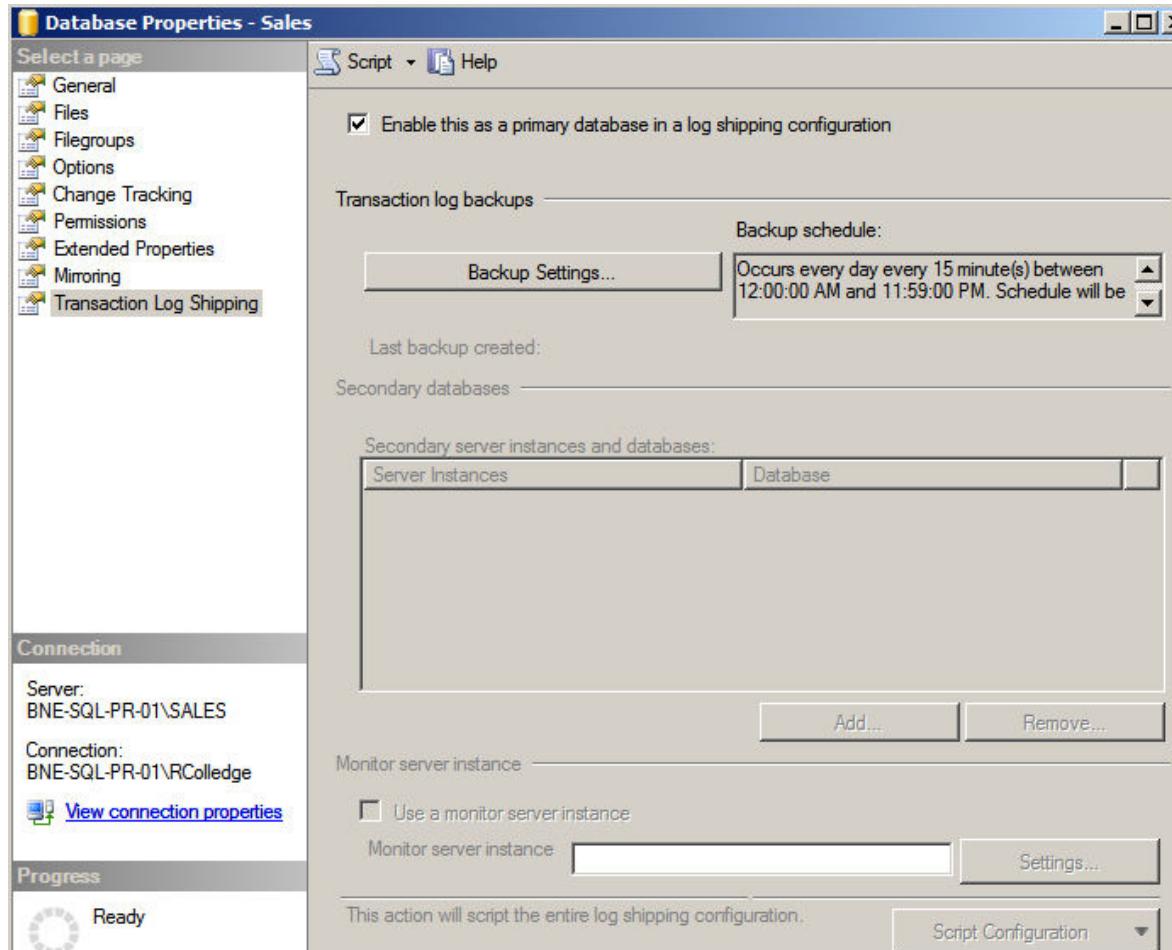
High-availability

- refers to any system or mechanism put in place to ensure the ongoing availability of a SQL Server instance in the event of a planned or unplanned outage
- Failover Clustering
- Log shipping
- Database Mirroring

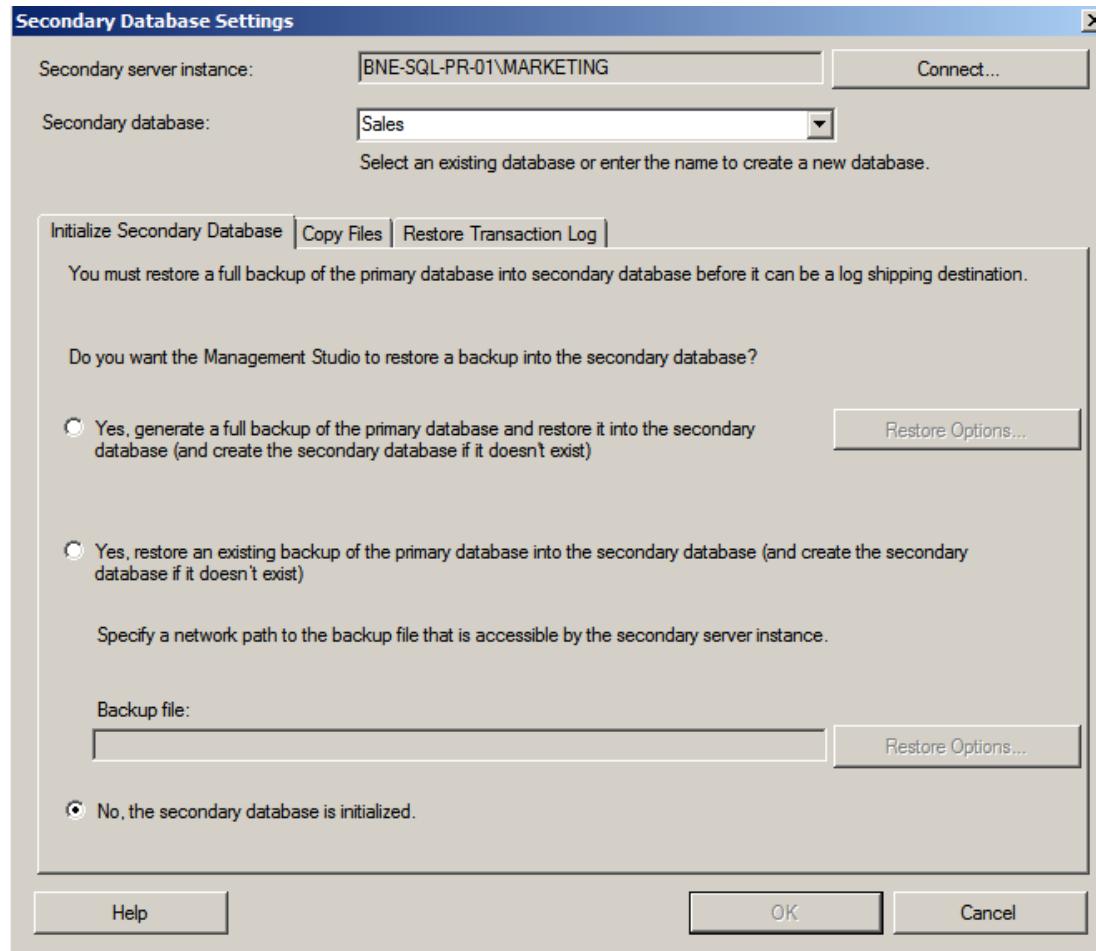
Database mirroring

- servers in database mirroring session use transaction log to move transactions between *principal* server and *mirror* server
- movement of transactions can be performed synchronously, guaranteeing that mirror is an exact copy of principal





DataBase Design course notes 06
SQL Server Administration

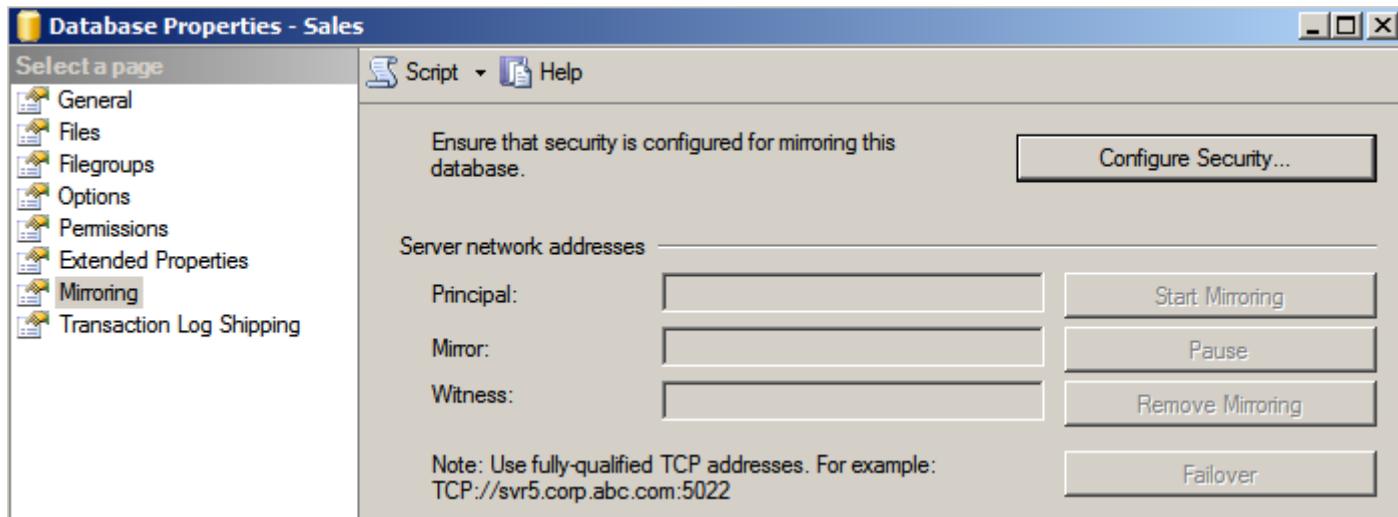


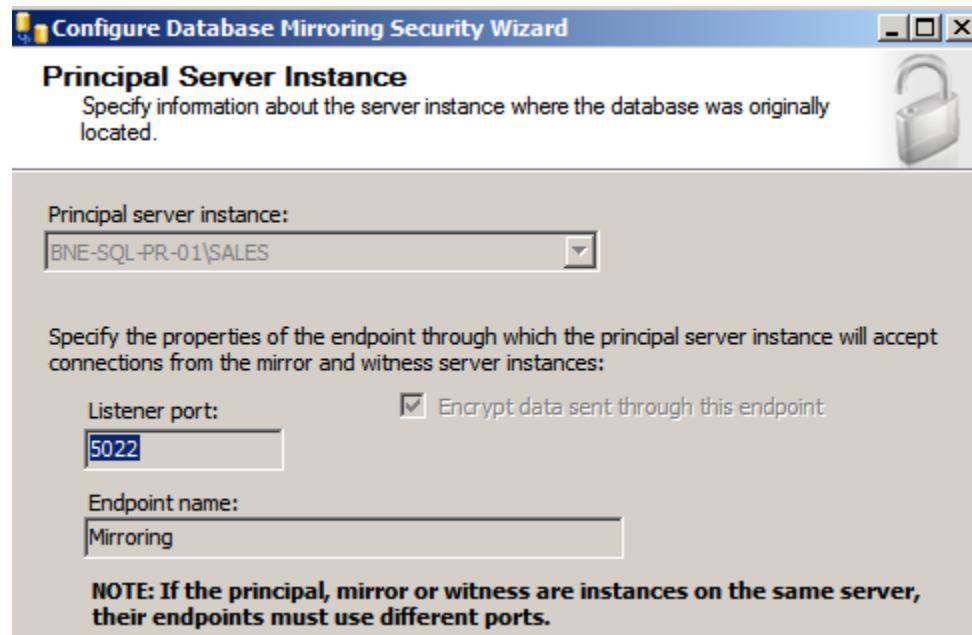
Failover and role reversal

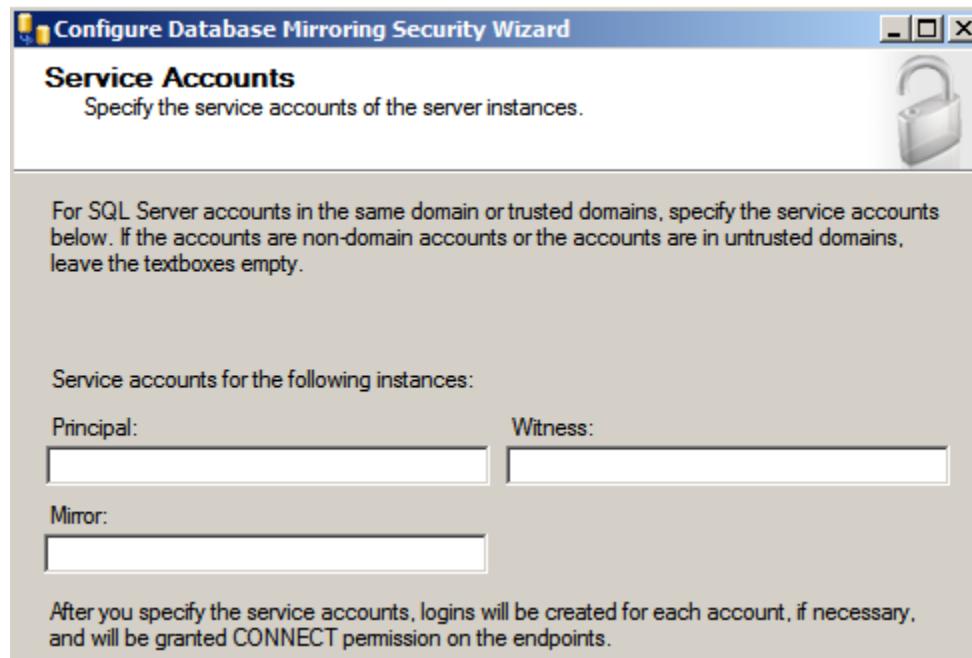
- if primary instance in log shipping solution fails (or a manual failover is required for planned maintenance), failover process is largely a manual effort, requiring you to back up, copy, and restore outstanding logs to secondary instance

Mirroring modes

- synchronous mirroring session (High safety)
- asynchronous (high performance)
 - transaction is committed on principal as soon as it's sent to mirror
 - used when transaction performance at principal is of prime concern







Mirroring session states

- Synchronizing
 - mirror DB is catching up on outstanding transactions
- Synchronized
 - mirror DB has caught up
- Disconnected
 - mirror partners have lost contact
- Suspended Caused by pausing or failover
 - No logs are sent to the mirror DB
- Pending failover
 - Temporary state at principal during failover

Index

design and maintenance

DBA OPERATIONS

Physical database design

- quality of database is number one concern
- assume that we've done our job in logical and implementation phases and that data quality is covered
- slow and right is always better than fast and wrong
 - like to get paid week early, but only half your money?
- obvious goal of computer system is to do things right and fast
- nothing we do in physical database design should affect data quality

Physical database design

- tuning your database structures
- must maintain balance between doing too much and doing too little
- if you don't use indexes enough, searches will be slow, as the query processor could have to read every row of every table for every query
- if you don't use too many indexes modifying data could take too long, as indexes have to be maintained

Indexes

- indexes allow database engine to perform fast, targeted data retrieval rather than simply scanning though the entire table
- well-placed index can speed up data retrieval by orders of magnitude
- indexing your data effectively requires a sound knowledge of how that data will change over time and the volume of data that you expect to be dealing with

Index structure

- index is an object that DBMS can maintain to optimize access to physical data in table
- can build index on one or more columns of table
- in essence, an index works on same principle as index of book
 - it organizes data from column (or columns) of data in a manner that's conducive to fast, efficient searching, so you can find row or set of rows without looking at entire table
 - provides means to jump quickly to specific piece of data

Basic index structure

- rather than just starting on page one each time you search the table and scanning through until you find what you're looking for
- even worse, unless DBMS knows exactly how many rows it is looking for, it has no way to know if it can stop scanning data when one row had been found
- also, like index of book, an index is separate entity from actual table being indexed

- indexes implemented using balanced tree (B-tree) structure
- index made up of index pages structured
 - each index page contains first value in a range and pointer to next lower page in index
 - last level in index is referred to as leaf page, which contains actual data values that are being indexed, plus either data for row or pointers to data

Clustered Indexes

- physically orders pages of data table
- leaf pages of clustered indexes are data pages of table
- each of data pages is then linked to next page in a doubly linked list
- leaf pages of clustered index are actual data pages
- data rows in table are sorted according to columns used in index

Clustered Indexes

- for clustered indexes that aren't defined as unique, each record has a 4-byte value (commonly known as a unifier) added to each value in the index where duplicate values exist
- can have only a single clustered index on a table
 - because table cannot be ordered in more than one direction

Non-Clustered Indexes

- are fully independent of underlying table
- completely separate from data, and on leaf page, there are pointers to go to data pages
- each leaf page in a nonclustered index contains some form of pointer to rows on data page, known as row locator

terms

- *scan*
- unordered search, scans leaf pages of index looking for value
- all leaf pages would be considered
- *seek*
- ordered search, in that index pages are used to go to a certain point in index and then scan is done on a range of values
- for unique index, this would always return a single value
- *lookup*
- clustered index is used to look up value for nonclustered index

Using Clustered Indexes

- primary key
- surrogate key (identity) as clustering key is great
 - small key (4 bytes integer), always unique value, generally monotonically increasing
- using GUID for surrogate key is becoming vogue, but be careful
- GUIDs are 16 bytes random values, generally aren't monotonically increasing, new GUID could sort anywhere in list of other GUIDs
- clustering on random value is generally horrible for inserts
 - because if you don't leave spaces on each page for new rows, you are likely to have page splitting
 - active system, constant page splitting can destroy your system

Using Clustered Indexes

- range queries
- data that's accessed sequentially
- queries that return large result sets

Using NonClustered Indexes

- alternate keys
- foreign keys

Domain tables

- to enforce domain using a table, rather than using scalar value with constraint

Domain tables

productType

productTypeCode

description

product

productCode

name

description

productTypeCode

Domain tables

- there are a small number of rows in productType
- unlikely that an index on product.productTypeCode column would be of any value in a join
- general advice is that tables of this sort don't need an index on foreign key values, by default

Domain tables

UserStatus

UserStatusCode
Description

User

UserId
Name
Description
UserStatusCode

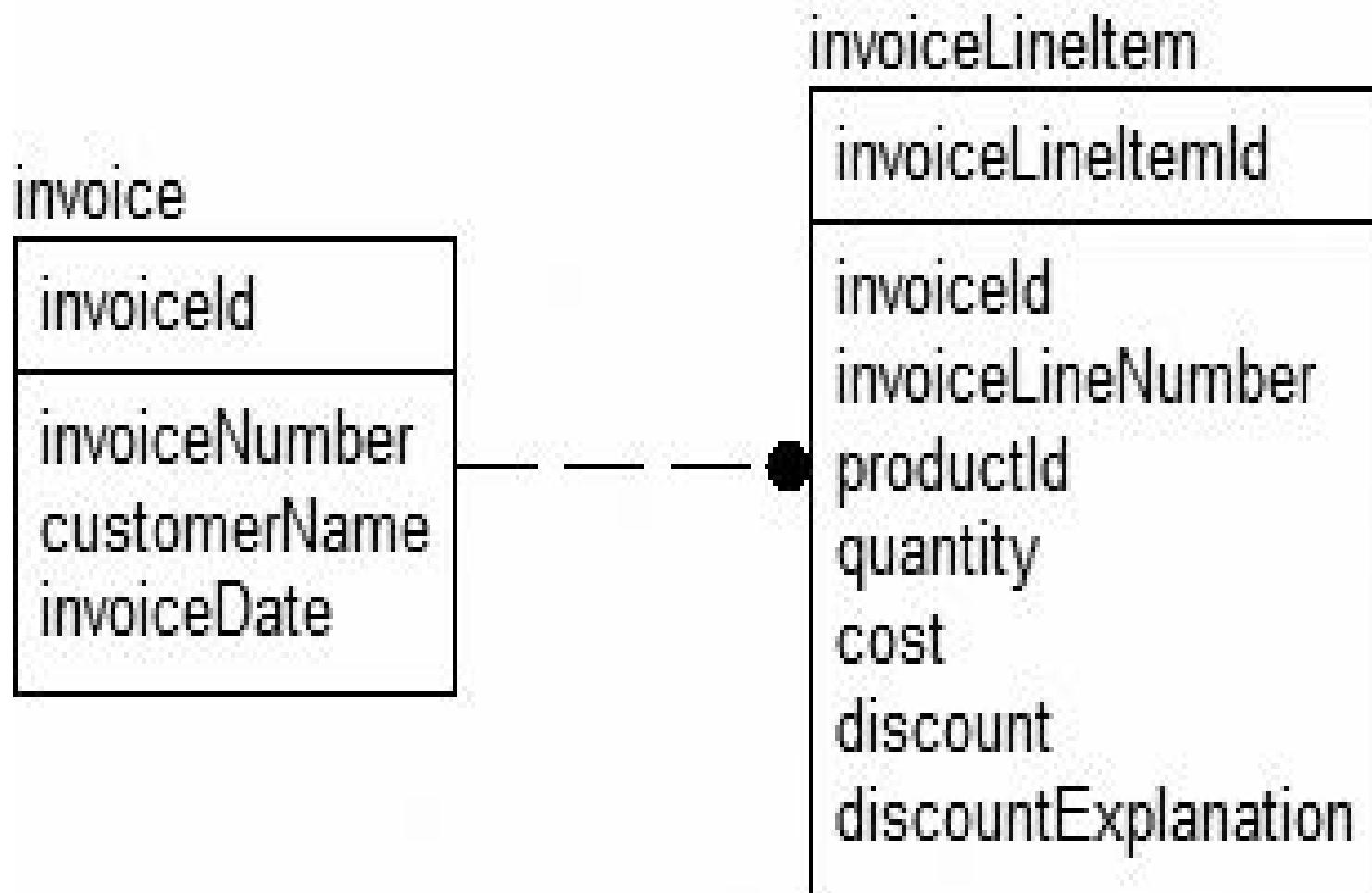
Domain tables

- most users would be with active status
- when user deactivated, you might need to do some action for that user
- since the number of inactive users would be far fewer than active users, it might be useful to have an index on userStatusCode column

Ownership Relationships

- ownership relationship to implement multivalued attributes of an object
- most of the time when the parent row is retrieved, the child rows are retrieved as well
- essential to have an index on the invoiceLineItem.invoicId column

Ownership Relationships



Many-to-Many Relationship

- needs to be an index on the two migrated keys from the two parent tables

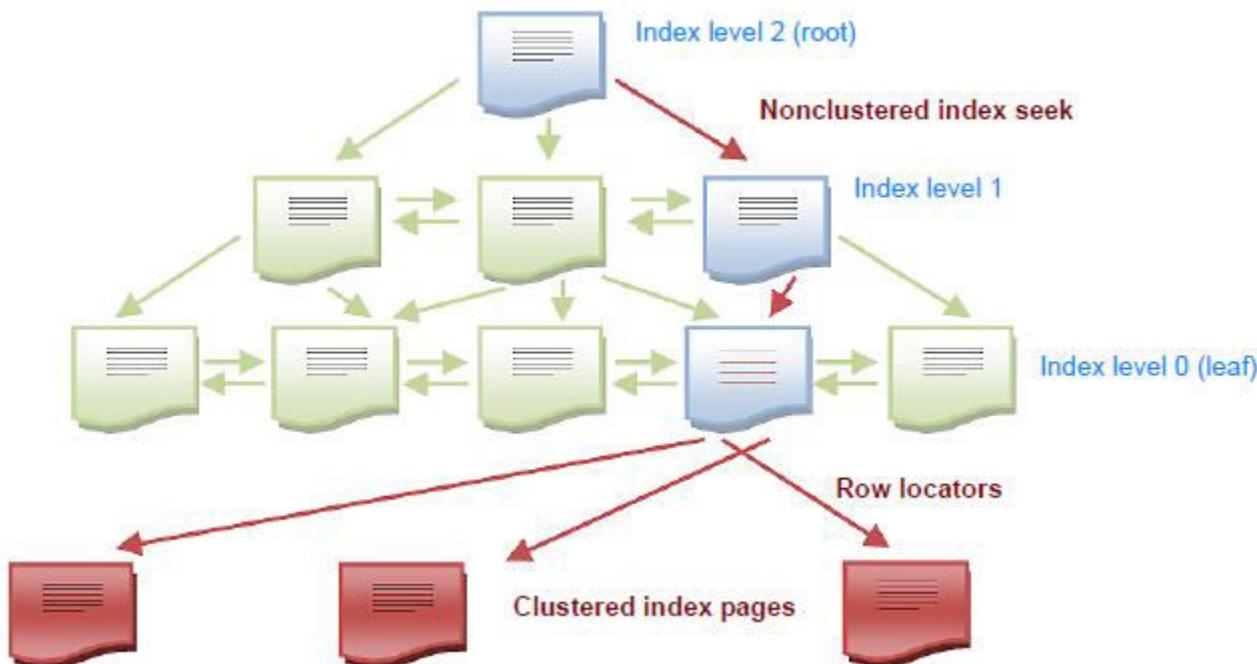
Best Practices

- apply UNIQUE constraints in all places where they make logical sense
- there are few reasons to add indexes to tables without testing
 - caveat can be foreign key indexes
- choose clustered index keys wisely
- keep indexes as thin as possible
 - consider several thin indexes rather than one monolithic index

Index

- possible for tables to be created without any indexes
- such tables are known as **heaps**.
- recommend that all tables be created with physical order, achieved by creating **clustered index**
- after create clustered index on column, data within table is physically ordered
- default, primary key constraint will be created as clustered index

nonclustered index lookup will traverse B-tree until it reaches leaf node
row locator used to locate data page



- `SELECT * FROM client WHERE SSN = '191-422-3775'` -- Query A
- `SELECT * FROM client WHERE surname = 'Smith'` -- Query B
- in Query A, index lookup on SSN will return single row
- in Query B, we're looking for people with surname of Smith, could return thousands of matching Rows
- table has two nonclustered indexes to support queries
 - one on SSN and other on surname
- first query, SSN index seek would return one row, with single key lookup required on clustered index to return remaining columns
- second query may return thousands of instances of Smith, each of which requires key lookup
- each of resultant Smith key lookups will be fulfilled from different physical parts of table, requiring thousands of random I/O operations

- depending on number of rows to be returned, it may be much faster to ignore index and sequentially scan table's clustered index
- despite reading more rows, overall cost of single, large, sequential I/O operation may be less than thousands of individual random I/Os

Statistics

- when indexes are first created, SQL Server calculates and stores statistical information on column values in index
- when evaluating how query will be executed, that is, whether to use index or not, SQL Server uses statistics to estimate likely cost of using index compared to other alternatives
- *selectivity* of index seek is used to define estimated percentage of matched rows compared to total number of rows in table

clustered index

- best candidates for clustered index
- columns that change infrequently
 - stable column value avoids need to maintain nonclustered index row locators
- columns that are narrow
 - limit size of each nonclustered index
- columns that are unique
 - avoid need for uniqueifier.

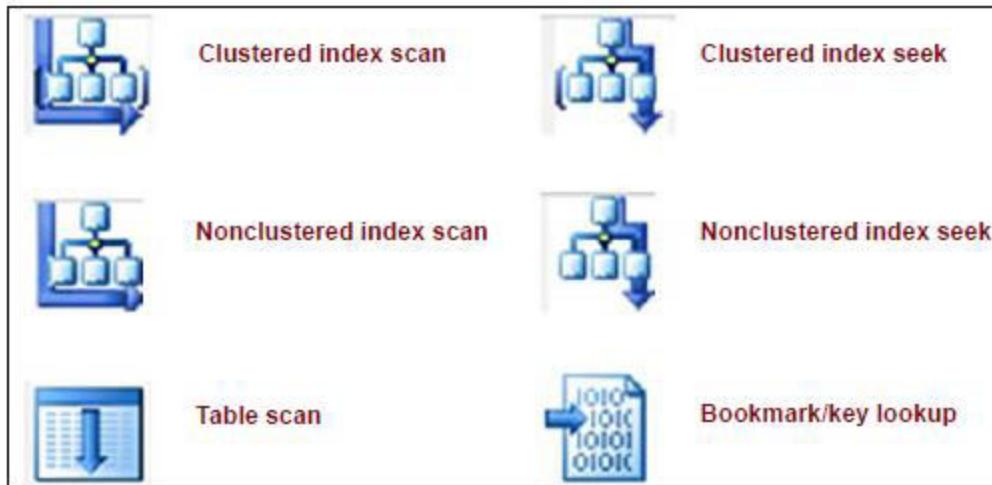
clustered index

- Using IDENTITY property to create surrogate key meets desired attributes for clustered index

Range scan, Sort

- tables that are frequently used in range-scanning operations, clustering on column(s) used in range scan can provide a big performance boost
- queries that select large volumes of sorted data often benefit from clustered indexes on column used in ORDER BY clause
- with data already sorted in clustered index, sort operation is avoided, boosting performance

icons used in graphical execution plans



- common indexing approach is to carpet bomb database with indexes in the hope that performance will (eventually) improve
- such an approach fail, usually ends in tears with accumulated performance and maintenance costs of unnecessary indexes eventually having paralyzing effect

Identifying indexes to drop/disable

- Indexes that are either not used or used infrequently not only consume additional space, but they also lengthen maintenance routines and slow performance, given the need to keep them updated in line with base table data

sys.dm_db_index_usage_stats

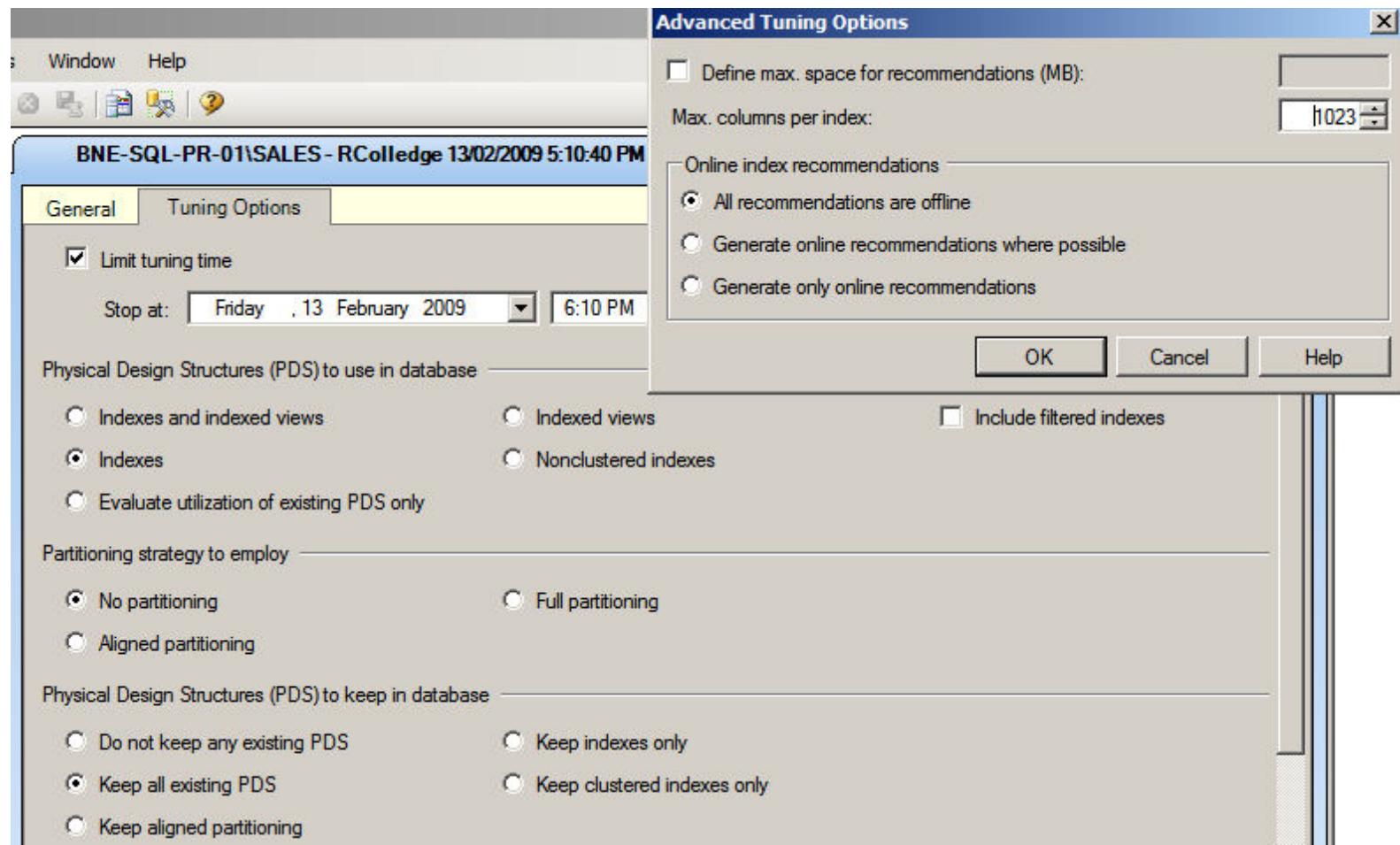
- returns information on how indexes are being used
- for each index, counts are kept on number of times index has been scanned, updated, used for lookup or seek purposes

`sys.dm_db_missing_index`

- When query optimizer uses suboptimal query plan, it records details of missing indexes that it believes are optimal for query
- access these details for all queries

Database Engine Tuning Advisor

- accessible in Performance Tools folder of Microsoft SQL Server 2008 program group
- analyzes workload from T-SQL file



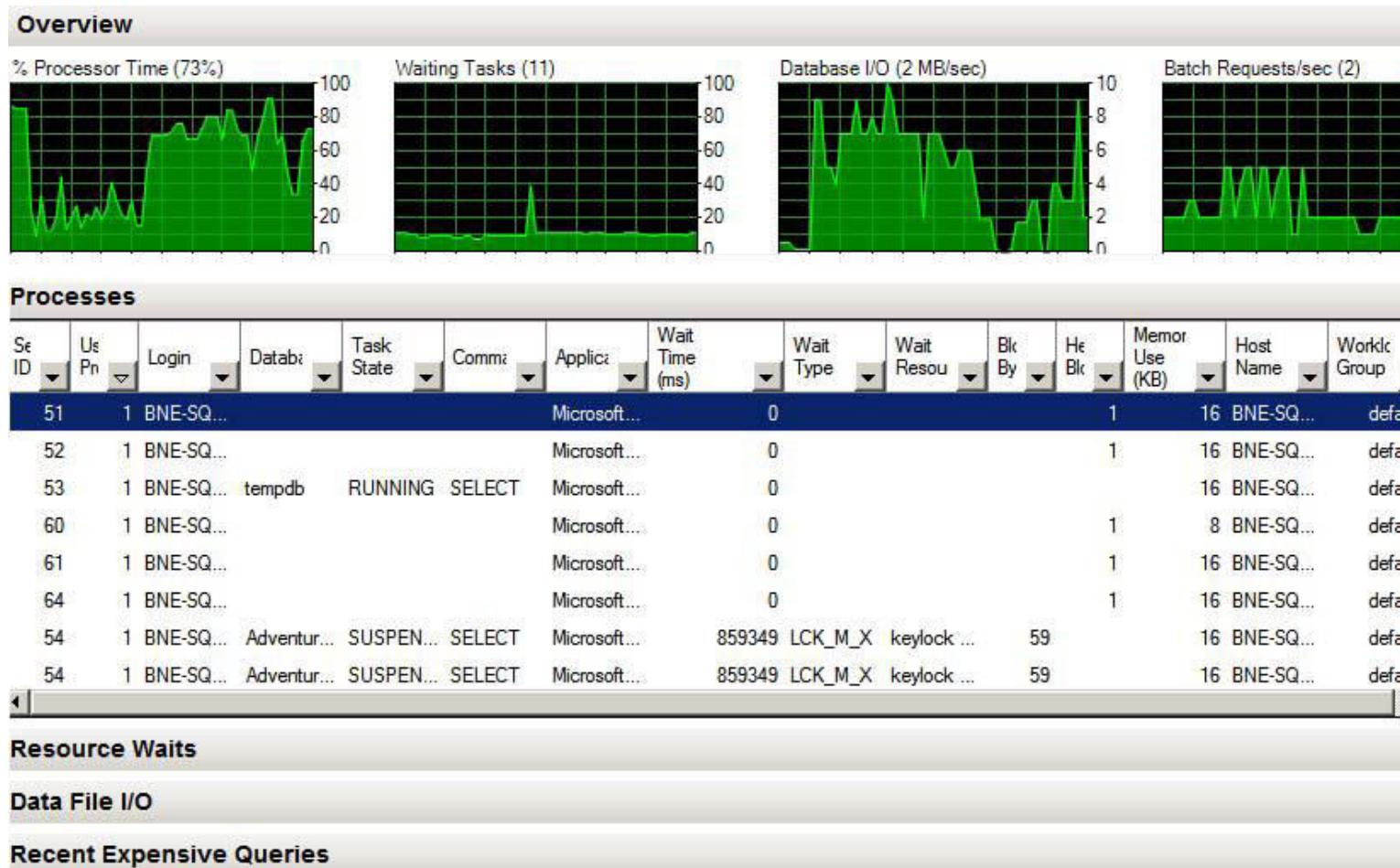
sp_updatestats

- runs UPDATE STATISTICS against all user tables using ALL keyword
- updating all statistics maintained for each user table

Monitoring and automation

DBA OPERATIONS

Activity Monitor



Activity Monitor

- Processes
- Resource Waits
- Data File I/O
- Recent Expensive Queries
 - sort results in ascending or descending order by CPU usage, physical/logical reads/sec, duration, plan count, and executions/min

SQL Server Agent

- component of SQL Server responsible for scheduled execution of tasks defined within jobs
 - creating maintenance plan will automatically create SQL Server Agent jobs to execute tasks contained within subplans

Monitoring and automation

DBA OPERATIONS

Data Collector

DBA OPERATIONS

DataBase Design course notes 06
SQL Server Administration

Data Collector

- controls collection and upload of information from SQL Server instance to management data warehouse using combination of SQL Server Agent jobs and SQL Server Integration Services (SSIS)
- Disk Usage
- Query Statistics
- Server Activity

Disk Usage Summary

Disk Usage Collection Set

on BNE-SQL-PR-01\SALES at 8/13/2008 1:19:50 PM



This report provides an overview of the disk space used for all databases on the server and growth trends for the data file and the log file for each database for the last 13 collection points between 8/12/2008 9:55:35 PM and 8/12/2008 10:09:57 PM.

Database Name	Database				Log			
	Start Size (MB)	Trend	Current Size (MB)	Average Growth (MB/Day)	Start Size (MB)	Trend	Current Size (MB)	Average Growth (MB/Day)
AdventureWorks2008	180.06		500.06	320	2.00		354.00	352
master	4.00		4.00	0	1.25		1.25	0
MDW	100.00		100.00	0	10.00		10.00	0
model	2.25		2.25	0	0.75		0.75	0
msdb	17.06		18.81	1.75	2.75		2.75	0
tempdb	15.69		33.88	18.188	2.25		2.25	0

Query Statistics History

Selected time range: 12/08/2008 9:55:17 PM to 12/08/2008 10:55:17 PM

Query

[Edit Query Text:](#)

```
select *  
from Sales.Store with (xlock)
```

Query Execution Statistics

Average CPU (ms) per Execution:	501.5
Average Duration (ms) per Execution:	4,003.4
Average Physical Reads per Execution:	0
Average Logical Writes per Execution:	0

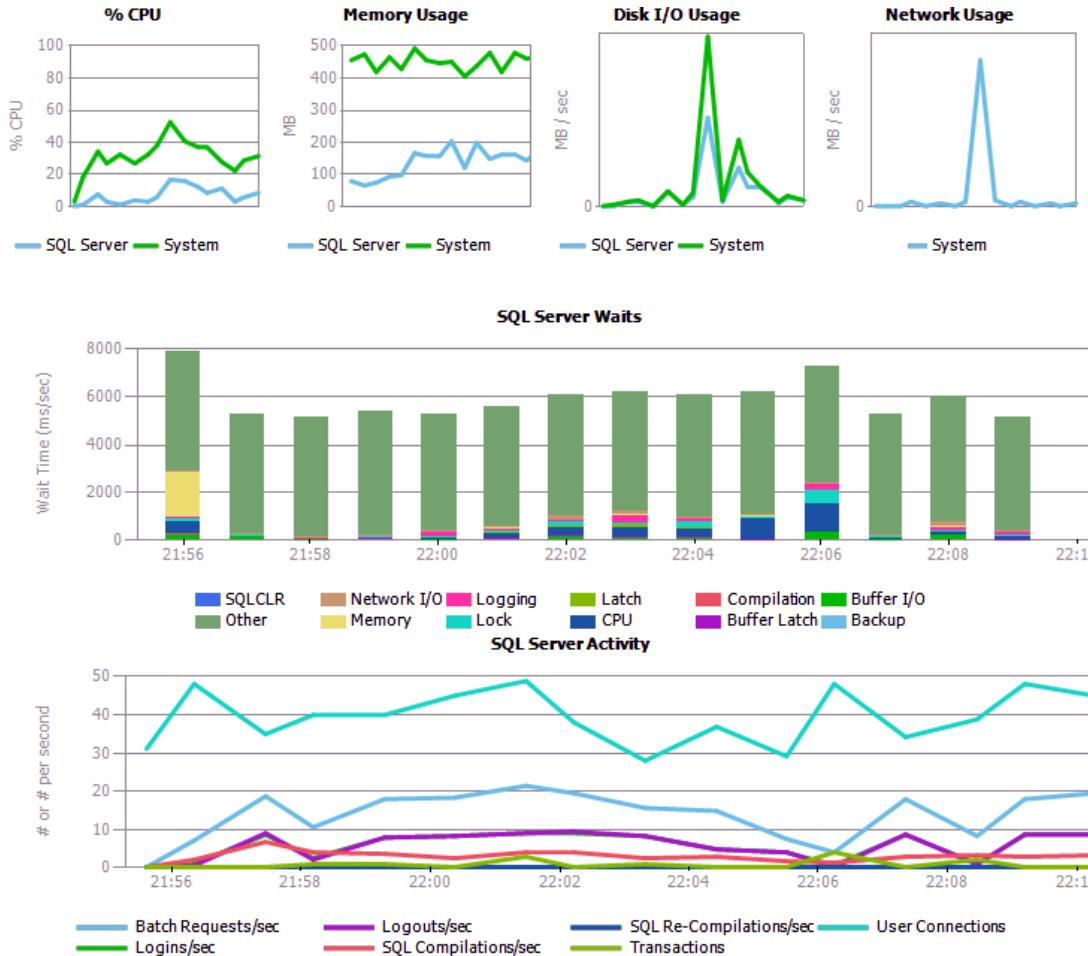
Total CPU (sec):	1.0
Total Duration (sec):	8.0
Total Physical Reads:	0
Total Logical Writes:	0
Total Executions:	2

Average Executions per Min:	0
Average CPU (ms) per Sec:	0
Average Duration (ms) per Sec:	2
Average Physical Reads per Sec:	0
Average Logical Writes per Sec:	0

Query Plan Count:

[View sampled waits for this query](#)

Server Activity History



Performance-tuning
Resource Governor
methodology
DBA OPERATIONS

Category	Waits	Queues	DMVs/DMFs	Additional notes and ideal values
CPU	Signal Wait % SOS_SCHEDULER_YIELD CXPACKET	% Processor Time % User Time % Privileged Time Interrupts/sec Processor Queue Length Context Switches/sec	sys.dm_os_wait_stats sys.dm_exec_query_stats sys.dm_exec_sql_text sys.dm_exec_cached_plans sys.dm_osSchedulers	Signal waits < 25%. High % privileged time may indicate hardware/driver issue. If Context Switches/sec > 20,000, consider fibre mode. CXPACKET waits on OLTP systems < 5%. Consider MAXDOP 1 and index analysis.
Memory	PAGEIOLATCH_*	Page Life Expectancy Buffer Cache Hit Ratio Pages/sec Page Faults/sec Memory Grants Pending CheckPoint Pages/sec Lazy Writes/sec Readahead Pages/sec	sys.dm_os_wait_stats sys.dm_os_memory_clerks	Page Life Expectancy > 500. Sudden page life drops may indicate poor indexing.
Disk IO	PAGEIOLATCH_* ASYNC/I/O_COMPLETION WRITLOG LOGMGR	Disk Seconds/Read and Write Disk Reads and Writes/sec % Disk Time Current and Avg. Disk Queue Length Log Flush Wait Time Log Flush Waits/sec Average Latch Wait Time Latch Waits/sec Total Latch Wait Time	sys.dm_os_wait_stats sys.dm_exec_query_stats sys.dm_exec_sql_text sys.dm_io_virtual_file_stats	Ensure storage tested with SQLIO/SIM before production implementation. tempdb separation. Disk Layout best practices including autogrow/shrink and t-log separation. Disk Sec/read and write <20 ms. Transaction log disk < 10ms. Disk queue length < 2 per disk in volume.
Network	NET_WAITFOR_PACKET	Bytes Received/sec Bytes Sent/sec Output Q length Dropped/Discarded Packets	sys.dm_os_wait_stats	Consider number of application round trips. Switched gigabit network connections.
Blocking	LOCK_*	Lock Waits/sec Lock Wait Time	sys.dm_os_wait_stats sys.dm_db_index_operational_stats	Consider SQL Profiler's blocked process report and deadlock graphs. Check transaction length, isolation levels and optimistic locking.
Indexing	Refer disk and memory waits	Forwarded Records/sec Full Scans/sec Index Searches/sec Pages Splits/sec	sys.dm_os_wait_stats sys.dm_db_index_operational_stats	Page Life Expectancy > 500. Sudden page life drops may indicate poor indexing.
Compilation	RESOURCE_SEMAPHORE_ QUERY_COMPILE	SQL Compilations/sec SQL Recompilations/sec Batch Requests/sec Auto Param Attempts/sec Failed Auto Param Attempts/sec Cache Hit Ratio (Plan Cache)	sys.dm_os_wait_stats sys.dm_exec_cached_plans sys.dm_exec_sql_text	Parameterized queries with sp_ExecuteSQL. Consider forced parameterization (after acknowledging possible downsides). Consider "optimize for ad hoc workloads" (after acknowledging possible downsides). Pay attention to large plan cache on 64-bit systems. Plan reuse on an OLTP system should be > 90%.

DataBase Design course notes 06

SQL Server Administration

DataBase Design

Transactions, Locking, Blocking, and
Deadlocking

Transactions

- define a single unit of work
 - can include one or more statements
- are either committed or rolled-back as group
- all-or-none functionality helps prevent partial updates or inconsistent data states
- when one part of interrelated process is rolled back or cancelled without rolling back or reversing all of other parts of interrelated processes

Atomicity, Consistency, Isolation, Durability

- Atomicity - means that transactions are an all-or-nothing entity - carrying out all steps or none at all
- Consistency - ensures that data is valid both before and after transaction
- data integrity maintained (foreign key references, ...) and internal data structures need to be in valid state

Atomicity, Consistency, Isolation, Durability

- Isolation - requirement that transactions not be dependent on other transactions that may be taking place concurrently (either at same time or overlapping)
- one transaction can't see another transaction's data that is in an intermediate state, but instead sees data as it was either before transaction began or after
- Durability - means that transaction's effects are permanent after transaction has committed
- any changes will survive system failures

Atomicity

- the whole transaction becomes persistent (COMMIT) in database or nothing (ROLLBACK)
- COMMIT or ROLLBACK can be explicitly executed by the user or by database engine
- most SQL engines default to ROLLBACK

Atomicity

- try to insert 1million rows into table and 1 row violated referential constraint, then whole set would be rejected, automatic ROLLBACK
- SAVEPOINT or CHECKPOINT options
 - sets savepoints during execution and lets transaction perform a local rollback to the checkpoint

Consistency

- when transaction starts, database is in consistent state, when it becomes persistent database is in a consistent state
- all data integrity constraints, relational integrity constraints, and any other constraints are true

Isolation

- one transaction is isolated from all other
- also called serializability
 - way to guarantee is serial execution
- system has to decide how to interleave transactions to get same effect

Durability

- database stored on durable media, if program is destroyed database persists
- database can be restored to a consistent state
- log files and backup procedures figure into this property, as well as disk writes done during processing

Bank Transaction

- BEGIN TRANSACTION
- UPDATE Accounts
 - SET valueAccount = valueAccount – (m + commisFee) WHERE AccountOwner = X
- UPDATE commissionAccount
 - SET valueAccount = valueAccount + commisFee WHERE AccountOwner = $Bank$
- UPDATE Accounts
 - SET valueAccount = valueAccount + m WHERE AccountOwner = Y
- END TRANSACTION

Concurrency Control

- multiple users who want to access it at the same time in their own sessions
- leads us to concurrency control
- concurrency control is part of transaction handling that deals with how multiple users access shared database

Transactions types

- in SQL Server 2005
 1. Autocommit
 2. Explicit
 3. Implicit

Autocommit

- default behavior
- each separate statement you execute is automatically committed after it is finished

Implicit

- SET IMPLICIT_TRANSACTIONS ON / OFF
- new transaction is automatically created (opened) when one of following statements is executed: ALTER TABLE, FETCH, REVOKE, CREATE GRANT, SELECT, DELETE, INSERT, TRUNCATE TABLE, DROP, OPEN, UPDATE
- remains open until either ROLLBACK or COMMIT statement is issued

Explicit

- transactions that you define
- recommended mode of operation when performing data modifications for database application
- explicitly control which modifications belong to single transaction, as well as actions that are performed if an error occurs

- BEGIN TRANSACTION - sets starting point of explicit transaction
- ROLLBACK TRANSACTION - restores original data modified by transaction, and brings data back to state it was in at start of transaction
- COMMIT TRANSACTION - ends transaction if no errors were encountered and makes changes permanent
- resources held by transaction are freed

- BEGIN DISTRIBUTED TRANSACTION - transaction to be managed by Microsoft Distributed Transaction Coordinator running locally and remotely
- SAVE TRANSACTION -issues savepoint within transaction
- @@TRANCOUNT - returns number of active transactions for connection

- CREATE PROCEDURE transfer_funds (
 @from_account INT, @to_account INT,
 @amount FLOAT)
- UPDATE account_balance SET balance = balance -
 @amount WHERE account_id = @from_account;
- ...
- UPDATE account_balance SET balance = balance +
 @amount WHERE account_id = @to_account;
- INSERT INTO journal (from_acount_id,
 to_account_id, transfer_amount) VALUES
 (@from_account, @to_account, @amount);

Example One

- CREATE PROCEDURE transfer_funds (
 @from_account INT, @to_account INT,
 @amount FLOAT)
- UPDATE account_balance SET balance = balance -
 @amount WHERE account_id = @from_account;
- ...
- UPDATE account_balance SET balance = balance +
 @amount WHERE account_id = @to_account;
- INSERT INTO journal (from_acount_id,
 to_account_id, transfer_amount) VALUES
 (@from_account, @to_account, @amount);

Example Two

- BEGIN TRANSACTION
- UPDATE account_balance SET balance = balance - @amount WHERE account_id = @from_account
- UPDATE account_balance SET balance = balance + @amount WHERE account_id = @to_account;
- INSERT INTO journal (from_acount_id, to_account_id, transfer_amount) VALUES (@from_account, @to_account, @amount);
- COMMIT TRANSACTION

Example Three

- SELECT COUNT(*) BeforeCount FROM Department
- DECLARE @Error INT
- BEGIN TRANSACTION
- INSERT Department (Id, Name) VALUES (121, 'Accounting')
- SET @Error = @@ERROR
- IF (@Error<> 0) GOTO Error_Handler
- INSERT Department (Id, Name) VALUES (121, 'Engineering')
- SET @Error = @@ERROR
- IF (@Error <> 0) GOTO Error_Handler
- COMMIT TRANSACTION

Example Three

- Error_Handler:
- IF @Error <> 0
- BEGIN
- ROLLBACK TRANSACTION
- END
- SELECT COUNT(*) AfterCount FROM Department

Example Three

- BeforeCount
- 19
- (1 row(s) affected)
- (1 row(s) affected)
- Msg 2601, Level 14, State 1, Line 14 - Cannot insert duplicate key row in object 'Department' with unique index
- The statement has been terminated
- AfterCount
- 19
- (1 row(s) affected)

Locking

- ensure integrity of data by not allowing concurrent updates to same data
- help prevent concurrency problems from occurring
 - user attempts to read data that another is modifying
 - user modify data that another is reading
 - user modify data that another transaction is trying to modify
- placed against resources

Lock modes

- Shared lock - issued during read, non-modifying queries; allow data to be read, but not updated by other processes while being held
- Intent lock - create lock queue, designating the order of connections and their associated right to update or read resources; show future intention of acquiring locks on specific resource

Lock modes

- Update lock - acquired prior to modifying data; when row is modified, lock is escalated to exclusive lock; if not modified, it is downgraded to shared lock; prevents deadlocks if two connections hold Shared (S) lock on resource, and attempt to convert to Exclusive (X) lock, but cannot because they are each waiting for the other transaction to release Shared (S) lock
- Exclusive lock - issues lock on resource that bars any kind of access (reads or writes); issued during INSERT, UPDATE, or DELETE statements
- Schema modification - issued when DDL statement is executed

Lock modes

- Schema stability - issued when query is being compiled; keeps DDL operations from being performed on table
- Bulk update - issued during bulk-copy operation; performance is increased but table concurrency is reduced
- Key-range - protect range of rows (based on index key); for example, protecting rows in an UPDATE statement with range of dates from '1/1/2005' to '12/31/2005'; prevents row inserts into date range that would be missed by current data modification

- lockable resources vary in granularity, from small (at row level) to large (entire database)
- small grain locks allow for greater database concurrency, because users can execute queries against specified unlocked rows
- each lock requires memory, so thousands of individual row locks can also affect performance
- larger grained locks reduce concurrency, but take up fewer resources

Lock Resources

- RID - Row Identifier, single table row
- Key - index row lock helping prevent phantom reads (also called Key-range lock) uses both range and row component
- Page - referring to an 8KB data or index page
- Extent - allocation unit of eight 8KB data or index pages
- HOPT - heap (table without clustered index) or B-tree
- Allocation unit - set of related pages grouped by data type, for example data rows, index rows, and large object data rows

Lock Resources

- Table - entire table, data, and indexes locked
- Object - database object, for example view, stored procedure, function
- File - database file
- DB - entire database lock.
- application - an application-specified resource
- Metadata - system metadata

- Not all lock types are compatible with each other
- for example, no other locks can be placed on resource that has already been locked Exclusive lock - other transaction must wait
- resource locked by Update can only have Shared lock placed on it by another transaction
- resource locked by Shared can have other Shared or Update locks placed on it

- Locks are allocated and escalated automatically
- escalation means that finer grain locks (row or page locks) are converted into coarse-grain table locks
- initialize escalation when single statement has more than 5,000 locks on single table or index, or if amount of locks on exceeds available memory

Interactions between Concurrent Transactions

- Dirty reads - occur while transaction is updating row, and second transaction reads the row before first transaction is committed; if original update rolls back, data read by second transaction is not the same
- Nonrepeatable reads - occur when transaction is updating data while second transaction is reading same data, both before and after change; data retrieved from first query does not match second query (this presumes that second transaction reads data twice: once before, and once after)

Interactions between Concurrent Transactions

- Phantom reads - occur when transaction retrieves set of rows once, another transaction inserts or deletes row from that same table, and first transaction re-executes query again only to find row that wasn't there before, or see that row is no longer returned in consecutive result sets; “phantom” is missing or new
- Lost updates - occurs when two transactions update a row's value, and transaction to last update row “wins”; thus the first update is lost

SQL Standard Isolation Levels

Isolation Level	Dirty Read	Non-Repeatable Read	Phantom Read
READ UNCOMMITTED	YES	YES	YES
READ COMMITTED	NO	YES	YES
REPEATABLE READ	NO	NO	YES
SERIALIZABLE	NO	NO	NO

SET TRANSACTION ISOLATION LEVEL

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SNAPSHOT
- SERIALIZABLE

READ COMMITTED

- default behaviour of SQL Server
- uncommitted data modifications can't be read
- Shared locks are used during query
- data cannot be modified by other processes while query is retrieving data
- data inserts and modifications to same table are allowed by other transactions, so long as rows involved are not locked by first transaction

READ UNCOMMITTED

- the least restrictive isolation level
- issuing no locks on data selected by transaction.
- provides highest concurrency but lowest amount of data integrity
- data that you read can be changed while you read it (these reads are known as “dirty reads”), or new data can be added or removed that would change your original query results
- allows you to read data without blocking others but with danger of reading data “in flux” that could be modified during read itself
 - including reading data changes from transaction that ends up getting rolled back
- for relatively static and unchanging data, can potentially improve performance - not issue unnecessary locking on accessed resources

REPEATABLE READ

- dirty and nonrepeatable reads are not allowed
- achieved by placing Shared locks on all read resources
- new rows that may fall into range of data returned by your query can, however, still be inserted by other transactions

SERIALIZABLE

- most restrictive setting
- Range locks are placed on data based on search criteria used to produce result set
- ensures that actions such as insertion of new rows, modification of values, or deletion of existing rows that would have been returned within original query and search criteria are not allowed

SNAPSHOT

- allows to read a transactionally consistent version of data as it existed at *beginning of transaction*
- data reads do not block data modifications
 - however, SNAPSHOT session will not detect changes being made

- possible for a user to fail to complete transaction for reasons other than hardware failing
 - deadlocks
 - livelock

Deadlocks

- possible for a user to fail to complete transaction for reasons other than hardware failing
- two or more users hold resources that others need, neither party will surrender
- DBA can kill one or more of sessions involved and roll back their work

Livelock

- involves users who are waiting for a resource, never get it because other users keep grabbing it.
- none of other users hold permanently, but as group they never free it
- DBA can kill one or more sessions
- DBA can raise priority of livelocked session so that can seize resources

Blocking

- occurs when one transaction in database session is locking resources that one or more other session transactions wants to read or modify
- short term blocking is usually acceptable, depending on your application requirements
- poorly designed applications can cause long term blocking
- unnecessarily keeping locks on resources and blocking other sessions from reading or updating

long term blocking can happen

- without proper indexing, blocking issues can grow
 - excessive row locks on table without index can cause to acquire table lock, blocking out other transactions
- applications that open transaction then request user feedback or interaction while transaction stays open
 - usually when end user is allowed to enter data in GUI while transaction remains open
- transactions that BEGIN and then look up data that could have been referenced prior to transaction starting

long term blocking can happen

- queries that use locking hints inappropriately
 - for example if application uses only few rows, but uses table lock hint instead
- application uses long-running transactions that update many rows or many tables within one transaction
 - chunking large updates into smaller update transactions can help improve concurrency

SET LOCK_TIMEOUT timeout_period

- when transaction is being “blocked,” this means it is waiting for lock on resource to be released
- specifies how long blocked statement should wait for lock to be released before returning an error
- timeout period is number of milliseconds before locking error will be returned

Deadlocking

- occurs when one user session (Session 1) has locks on resource that another user session (Session 2) wants to modify, and Session 2 has locks on resources that Session 1 needs to modify
- neither Session 1 nor Session 2 can continue until other releases locks
- database server chooses one of the sessions in deadlock as “deadlock victim”
 - has its session killed and transactions rolled back

- Identifying deadlocks with Trace Flag
 - using DBCC TRACEON, DBCC TRACEOFF, and DBCC TRACESTATUS
- SQL Profiler
 - other methods for troubleshooting deadlocks

Setting Deadlock Priority

- can increase query session's chance of being chosen as deadlock victim by using
- **SET DEADLOCK_PRIORITY**
 - LOW
 - NORMAL
 - HIGH
 - numeric-priority

2 Basic Classes of Concurrency Control

- SQL standards do not say *how* are to achieve these results
- Optimistic
- Pessimistic

Pessimistic Concurrency Control

- transactions are expected to conflict
- design system to avoid problems before start
- use locks
 - flags placed in database that give exclusive access to schema object to user

Pessimistic Concurrency Control

level of locking

- whole database
 - for system maintenance work
- table level
 - performance suffer, users must wait for common tables to become available

Pessimistic Concurrency Control

level of locking

- row level
 - other users can get to the rest
 - best possible shared access
 - huge number of flags to process and performance will suffer
- page locking
 - performance depends statistic distribution
 - generally the best compromise

Optimistic Concurrency Control

- based on idea that transactions are not very likely to conflict with each other
- need to design system to handle problems as exceptions after occur
- uses timestamp to track copies of data

Logical Concurrency Control

- based on idea that machine can analyze predicates in the queue of waiting queries and processes on a purely logical level and determine which of the statements can be allowed to operate on the database at the same time

EXAMPLE OF A TRANSACTION

CREATE TABLE Student

- CREATE TABLE student(
- student_id int IDENTITY(1,1) NOT NULL,
- stud_name *nvarchar*(50) NOT NULL,
- stud_status *varchar*(50) NOT NULL,
- CONSTRAINT [PK_student] PRIMARY KEY CLUSTERED
- ([student_id] ASC)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
-) ON [PRIMARY]

INSERT Student

- `insert student(stud_name, stud_status)`
- `values (N'Călin CENAN', N'studențesc')`

- `insert student(stud_name, stud_status)`
- `values ('Ioana Călina CENAN', 'studentesc')`

SELECT * FROM Student

- student_id stud_name stud_status
- 1 Călin CENAN studentesc
- 2 Ioana Calina CENAN studentesc

- cannot insert Unicode in VarChar
- could insert Unicode in NVarChar
- should prefix char. string with N

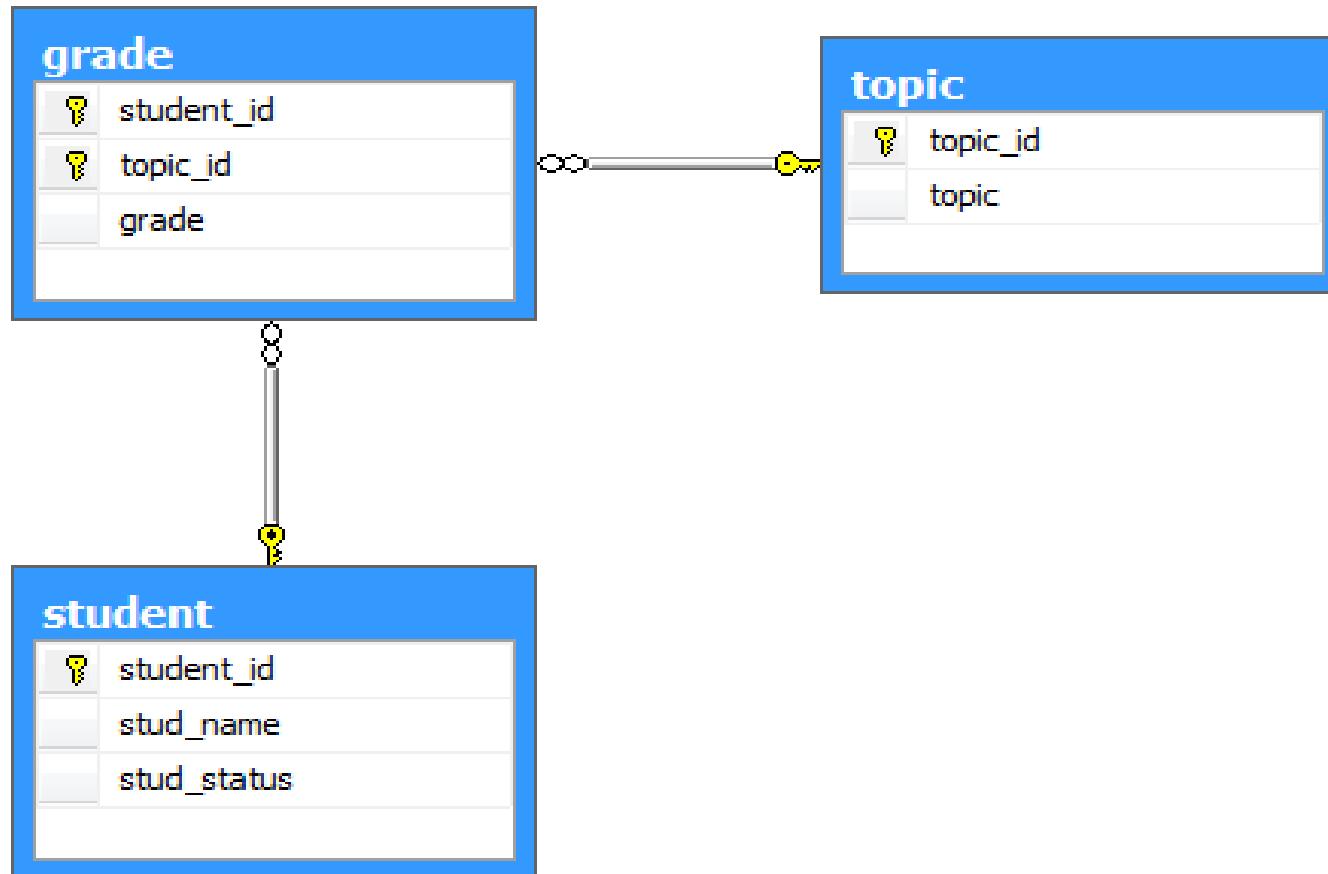
CREATE TABLE Topic

- CREATE TABLE topic (
- topic_id int IDENTITY(1,1) NOT NULL,
- topic varchar(50) NOT NULL,
- CONSTRAINT [PK_topic] PRIMARY KEY CLUSTERED
- (
- [topic_id] ASC
-)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
-) ON [PRIMARY]

CREATE TABLE Grade

- CREATE TABLE grade (
- student_id int NOT NULL,
- topic_id int NOT NULL,
- grade int NULL,
- CONSTRAINT [PK_grade] PRIMARY KEY CLUSTERED
- ([student_id] ASC,
- [topic_id] ASC
-)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
• IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
• ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
-) ON [PRIMARY]

DataBase Diagram



CREATE VIEW View_All

- CREATE VIEW View_All
- AS
- SELECT dbo.student.student_id,
dbo.student.stud_name, dbo.student.stud_status,
dbo.topic.topic, dbo.grade.grade
- FROM dbo.grade INNER JOIN
- dbo.student ON dbo.grade.student_id =
dbo.student.student_id INNER JOIN
- dbo.topic ON dbo.grade.topic_id =
dbo.topic.topic_id

SQL Code

- SELECT * FROM View_All
- GO
- UPDATE grade SET grade = 10 WHERE student_id = 1 AND topic_id = 1
- GO
- SELECT * FROM View_All
- GO

- DECLARE @Var INTEGER;
- DECLARE @DDate DATETIME;
- SET @Var = 0;
- SET @DDate = DATEADD(s, 10, GETDATE());
- WHILE GETDATE() < @DDate
 - BEGIN
 - SET @Var = @Var + 1
 - END;

- UPDATE student SET stud_status = 'Abslov. Ing.' WHERE student_id = 1
- GO
- SELECT * FROM View_All
- GO

1st UpDate

- SELECT * FROM View_All
- GO
- UPDATE grade SET grade = 10 WHERE student_id = 1 AND topic_id = 1
- GO
- SELECT * FROM View_All
- GO

Delay

- DECLARE @Var INTEGER;
- DECLARE @DDate DATETIME;
- SET @Var = 0;
- SET @DDate = DATEADD(s, 10, GETDATE());
- WHILE GETDATE() < @DDate
 - BEGIN
 - SET @Var = @Var + 1
 - END;

2nd UpDate

- UPDATE student SET stud_status = 'Absolv.
Ing.' WHERE student_id = 1
- GO
- SELECT * FROM View_All
- GO

View_All Before Updates – Valid States

- student_id stud_name stud_status topic grade
- 1 Călin CENAN Studentesc Licence 5

View_All Between Updates - Invalid States

- student_id stud_name stud_status topic grade
- 1 Călin CENAN Studentesc Licence 10

View_All After Updates – Valid States

- student_id stud_name stud_status topic grade
- 1 Călin CENAN Abslov. Ing. Licence 10

Make a TRANSACTION

- BEGIN TRANSACTION
- SQL Code
- COMMIT TRANSACTION

Check our Work

- BEGIN TRANSACTION
- SQL Code
 - increase Delay (after 1st update), from 10 sec to 1000 sec
 - time enough to produce a failure in the system
 - like pressing 3 key salute Ctrl-Alt-Del
- COMMIT TRANSACTION

View_All After Failed TRANSACTION

- student_id stud_name stud_status topic grade
- 1 Călin CENAN Studentesc Licence 5
- instead of
- View_All After 1st Update and Before 2nd Update
- student_id stud_name stud_status topic grade
- 1 Călin CENAN Studentesc Licence 10
- View_All After 1st & 2nd Update
- student_id stud_name stud_status topic grade
- 1 Călin CENAN Abslov. Ing. Licence 10

function bytea_import(p_path text, p_result out bytea)

- create or replace function bytea_import(p_path text, p_result out bytea)
- language plpgsql as \$\$
- declare
- l_oid oid;
- r record;
- begin
- p_result := "";
- select lo_import(p_path) into l_oid;
- for r in (select data
- from pg_largeobject
- where loid = l_oid
- order by pageno) loop
- p_result = p_result || r.data;
- end loop;
- perform lo_unlink(l_oid);
- end;\$\$;

```
insert into my_table(bytea_data)
select bytea_import('/my/file.name');
```

- ```
insert into my_table(bytea_data) select
bytea_import('/my/file.name');
```

# DataBase Design

DataBase Design

# Design DataBase Structures

- DataBase Design focuses on how database structure will be used to store and manage end-user data
- data modeling - first step in designing database: refers to process of creating specific data model for determined problem domain
- *database model* used to refer to implementation of *data model* in specific database system

# Design DataBase Structures

- data model is representation, usually graphical, of more complex real-world data structures; is abstraction of complex real-world object or event
- model's main function is to help you understand real-world environment
- data model represents data structures and their characteristics, relations, constraints, transformations, ... with purpose of supporting specific problem domain

# Data model

- organizes data elements and standardizes how data elements relate to one another
- Conceptual
- Logical
- Physical

# Conceptual Data Model

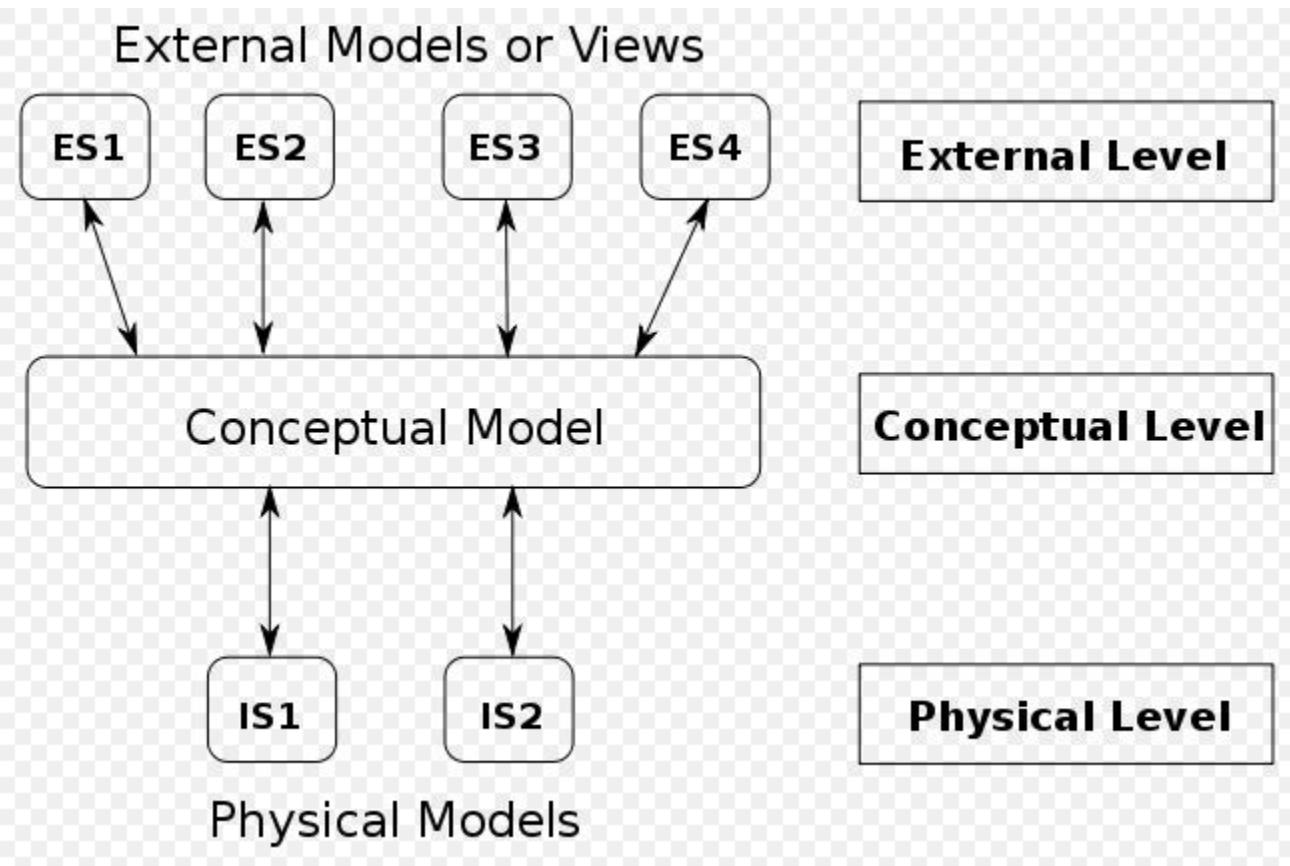
- Includes high-level data constructs
- Non-technical names, executives and managers at all levels can understand data basis
- Uses general high-level data constructs created in non-technical terms
- May not be normalized

# Logical Data Model

- Includes entities (tables), attributes (columns/fields) and relationships (keys)
- Uses business names for entities & attributes
- Is independent of technology (platform, DBMS)
- Is normalized to fourth normal form (4NF)

# Physical Data Model

- Includes tables, columns, keys, data types, validation rules, database triggers, stored procedures, domains, and access constraints
- Uses more defined and less generic specific names for tables and columns and company defined standards
- Includes primary keys and indices for fast data access
- May be de-normalized to meet performance requirements - based on nature of database –
  - Online Transaction Processing (OLTP) or Operational Data Store (ODS)  
it is usually not de-normalized
  - Online Analytical Processing (OLTP), Datawarehouses, de-normalization is common



# conceptual data model

- represent abstract structure of domain of information
- often diagram, used in business processes to capture things of importance to an organization and how they relate to one another
- basis of physical data model

# logical data model

- based on structures identified in preceding conceptual data model, since this describes semantics of information context

# Importance of Data Models

- facilitate interaction among designer, applications programmer, and end user
- improved understanding of organization
- communication tool
- applications created to manage data and to help transform data into information
  - but data are viewed in different ways by different people
- to create good database first create good, appropriate data model

# Data Model – Basic Building Blocks

- entities
- attributes
- relationships
- constraints
- *how to identify entities, attributes, relationships, and constraints? first step identify business rules for problem domain you are modeling*

# Data Model – Basic Building Blocks

## entity

- **entity** is person, place, thing, object or event about which data will be collected and stored
- represents particular type of object in real world, which means is “distinguishable” - each occurrence is unique and distinct
- may be physical objects, such as customers or products
- may also be abstractions, such as flight routes or musical concerts

# Data Model – Basic Building Blocks

## attribute

- **attribute** is characteristic of entity
- for example, CUSTOMER entity would be described by attributes such as customer last name, first name, phone number, address, and credit limit
- are equivalent of fields in file systems

# Data Model – Basic Building Blocks

## relationship

- **relationship** describes association among entities
- for example, relationship exists between customers and agents that can be described as an agent can serve many customers, and each customer may be served by one agent
- three types of relationships: one-to-many (1:M or 1..\*), many-to-many (M:N or \*..\*), and one-to-one (1:1 or 1..1)

# Data Model – Basic Building Blocks

## One-to-many (1:M or 1..\*) relationship

- painter creates many different paintings, but each is painted by only one painter
- painter (“one”) is related to paintings (“many”)
- relationship “PAINTER paints PAINTING” is 1:M
- customer (“one”) may generate many invoices, but each invoice (“many”) is generated by only single customer
- “CUSTOMER generates INVOICE” relationship would also be 1:M

# Data Model – Basic Building Blocks

## Many-to-many (M:N or \*..\*) relationship

- employee may learn many job skills, each job skill may be learned by many employees
- “EMPLOYEE learns SKILL” is M:N
- student can take many classes and each class can be taken by many students
- “STUDENT takes CLASS” is M:N

# Data Model – Basic Building Blocks

## One-to-one (1:1 or 1..1) relationship

- retail company's management structure may require that each of its stores be managed by single employee
- in turn, each store manager, who is an employee, manages only single store
- “EMPLOYEE manages STORE” is 1:1

# Data Model – Basic Building Blocks

## constraint

- **constraint** is restriction placed on data
- are important because they help to ensure data integrity
- normally expressed in form of rules
- for example:
  - employee's salary must have values that are between 6,000 and 350,000
  - student's grades must be between 0.00 and 10.00.
  - each class must have one and only one teacher

# Business Rule

- brief, precise, and unambiguous description of policy, procedure, or principle within specific organization
- apply to *any* organization - business, government unit, religious group, research laboratory - that stores and uses data to generate information
- business rules describe main and distinguishing characteristics of data *as viewed by organization*

# Business Rule

- set stage for proper identification of entities, attributes, relationships, and constraints
- names are used to identify objects
- noun in business rule will translate into entity in model
- verb (active or passive) that associates nouns will translate into relationship among entities
- should consider that relationships are bidirectional
- *naming convention*

# Models

- represent schools of thought as to what database is, what it should do, types of structures that it should employ, and technology that would be used to implement structures
- models are called data models, as are graphical data models
- next section traces evolution of data models and gives overview of major data models in roughly chronological order

# Models

1. 1960-70 – file system
2. 1970 – hierarchical and network
3. 1975 – relational
4. 1985 – object oriented / object relational
5. 1995 – XML, hybrid DataBase Management Systems
6. 2010 – emerging models, NoSQL; key-value store, column store, big data, ...

# Hierarchical Model

- manage large amounts of data for complex manufacturing projects
- model's basic logical structure is represented by upside-down tree - hierarchical structure contains levels, or segments
- segment is equivalent of file system's record type
- within hierarchy, higher layer is perceived as parent of segment directly beneath it, which is called child
- depicts set of one-to-many (1:M) relationships between parent and its children segments

# Network Model

- created to represent complex data relationships more effectively than hierarchical model, to improve database performance, and to impose database standard (CODASYL)
- user perceives network database as collection of records in 1:M relationships - network model allows record to have more than one parent
- definitions of standard database (CODASYL) *concepts* that emerged with network model are still used by modern data models

# DataBase Concepts

- schema is conceptual organization of entire database as viewed by database administrator
- subschema defines portion of database “seen” by application programs that actually produce desired information from data within database
- Data Manipulation Language (DML) defines environment in which data can be managed and is used to work with data in database
- Data Definition Language (DDL) enables database administrator to define schema components

- lack of ad hoc query capability put heavy pressure on programmers to generate code required to produce even simplest reports
- provided limited data independence
  - any structural change in database could still produce havoc in all application programs
- because of disadvantages of hierarchical and network models, they were largely replaced by **relational data model** in 1980s

# Object-Oriented Model

- both data and their relationships are contained in single structure known as object
- basis for Object-Oriented DBMS
- reflects very different way to define and use entities
- like relational model's entity, object is described by its factual content - *unlike* entity, object includes information about relationships between facts, information about relationships with other objects
- therefore, facts within object are given greater meaning
- semantic data model - *semantic* indicates meaning

# Object-Oriented Model

- development has allowed object also to contain all *operations* that can be performed on it
  - such as changing its values, finding specific data, printing data
- because objects include data, various types of relationships, and operational procedures, object becomes self-contained, thus making it basic building block for autonomous structures

# Object-Oriented Model Components

- object is abstraction of real-world entity
  - object represents only one occurrence of entity
- attributes describe properties of object

# Object-Oriented Model Components

- objects that share similar characteristics grouped in *classes*
- class is collection of similar objects with shared structure (attributes) and behavior (*methods*)
- class resembles relational model's entity set
- class is different from entity set in that it contains set of procedures known methods
- class's method represents actions, are equivalent of procedures in traditional programming languages
- in Object Oriented terms, methods define object's behavior

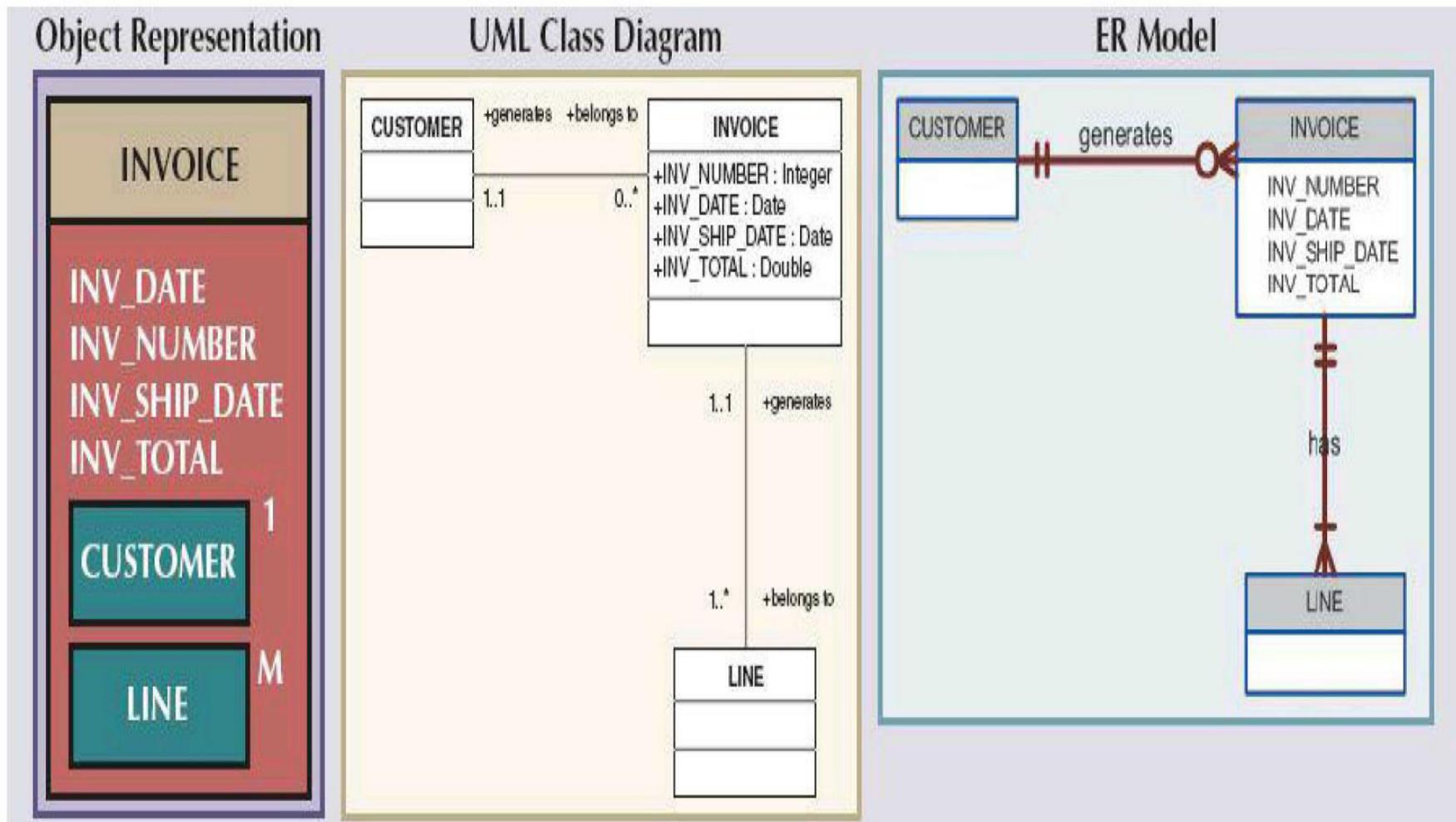
# Object-Oriented Model Components

- classes organized in hierarchy
- inheritance is ability of object within class hierarchy to inherit attributes and methods of classes above it

# Object-oriented models typically depicted using Unified Modeling Language diagrams

- UML is language based on OO concepts that describes set of diagrams and symbols use to graphically model system
  - to illustrate main concepts consider simple invoicing problem

# comparison of OO, UML, and ER models



# Object / Relational and XML

- added semantics of OO model allowed for richer representation of complex objects
- enabled applications to support increasingly complex objects in innovative ways
- facing demand to support more complex data representations, relational model's main vendors evolved model further and created Extended Relational data model
- gave birth to new generation of relational databases that support OO features such as objects (encapsulated data and methods), extensible data types based on classes, and inheritance
- Object/Relational DataBase Management System
  - PostgreSQL, Oracle, SQL Server

- OO DBMS popular in niche markets such as computer-aided drawing/computer-aided manufacturing (CAD/CAM), geographic information systems (GIS), communications, multimedia,
  - which require support for more complex objects

# Object / Relational and XML

- use of complex objects received boost with Internet revolution - realized potential to access, distribute, and exchange critical business information
- within this environment, Extensible Markup Language (XML) emerged as de facto standard for efficient and effective exchange of structured and unstructured data
- XML databases emerged to manage unstructured data within native XML format
- O/R DBMSs added support for XML-based documents within their relational data structure
- due to its robust foundation in broadly applicable principles, relational model is easily extended to include new classes of capabilities, such as objects and XML

- although relational and object/relational databases address most current data processing needs, new generation of databases has emerged to address some very specific challenges found in Internet-era organizations
- Emerging Data Models: Big Data and NoSQL

# Emerging Data Models: Big Data & NoSQL

- rapid pace of data growth is top challenge for organizations
  - with system performance and scalability as next biggest challenges
- *Big Data* refers to movement to find new and better ways to manage large amounts of Web-generated data and derive business insight from it, while simultaneously providing high performance and scalability at reasonable cost
- relational approach does not always match needs of organizations with Big Data challenges

# Emerging Data Models: Big Data & NoSQL

- organizations are turning to NoSQL databases to mine wealth of information hidden in mountains of Web data and gain competitive advantage
- relational databases remain preferred and dominant databases to support most day-to-day transactions and structured data analytics needs
- object/relational databases serve 98% of market needs
- for remaining 2%, NoSQL databases are an option

# NoSQL Databases

- *not based on relational model*, hence name NoSQL
- supports distributed database architectures
- provides high scalability, high availability, and fault tolerance
- supports big data, very large amounts of sparse data
- geared toward performance rather than transaction consistency

# Database design

- part science and part art
- science to normalization
- but determining scope of entity is an art form
  - best learned from working with a broad repertoire of databases.
- foundation of every datacentric application

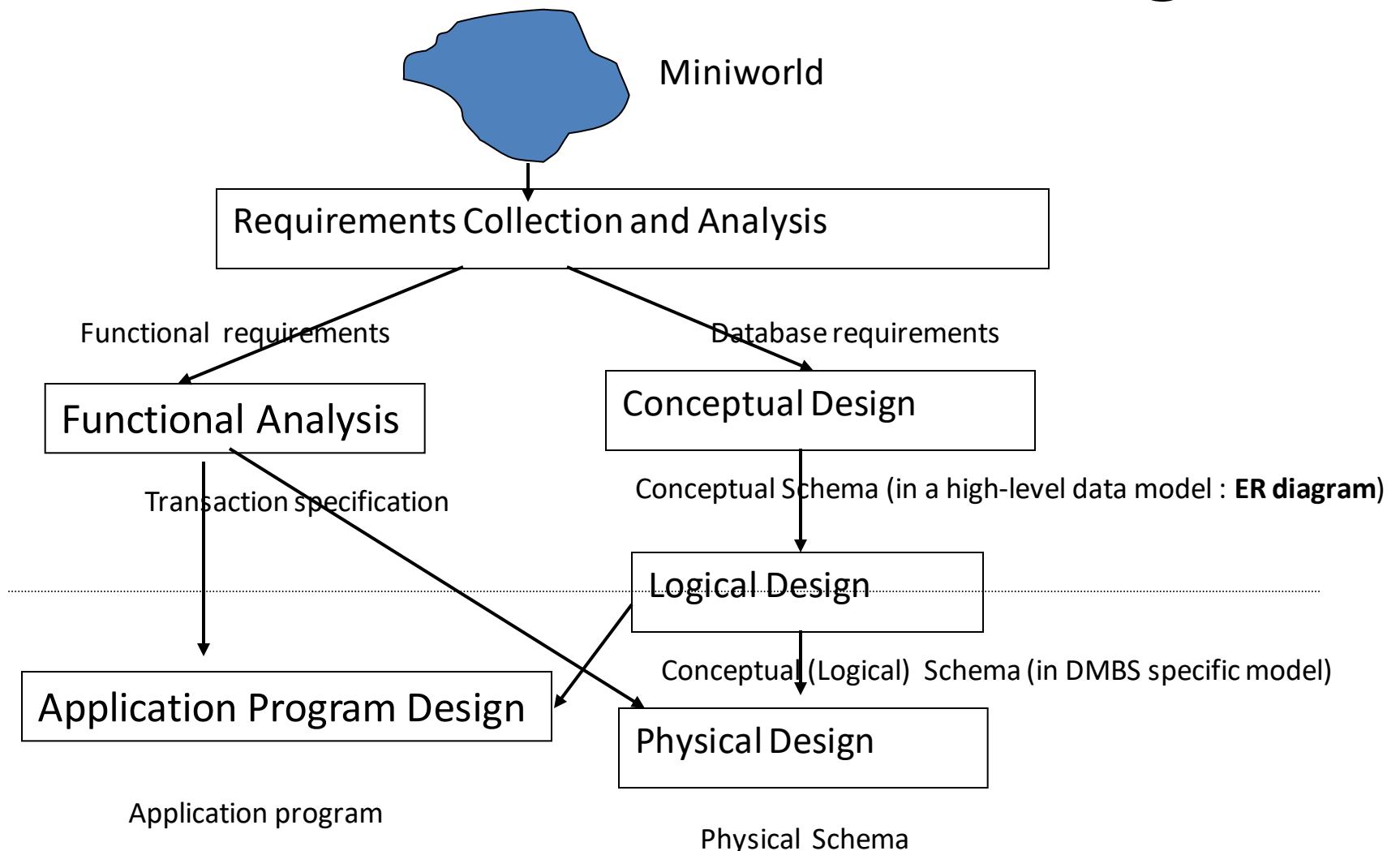
# Database design

- good database design makes the data obvious and easy to query and sets up the developer for success with efficient set based queries
- no amount of code can compensate for a poor database design or add features missing from the database

# Database design

- user interfaces come and go, but data lasts for generations of application languages
- today's database schema errors will be cursed by programmers not yet born using languages and tools not yet invented
- worth spending a little extra time to polish your database design

# Overview of Database Design



- too often when to build a system that requires data storage, reaction is to start thinking in terms of how to fulfill immediate need
- little regard is given to future data needs, and even less is given to impact design will have on future business needs, reporting requirements, and, most crucial, integrity of data

- obvious things are commonly missed, and late in project you have to go back and tweak (and re-tweak) design
- too often, too much time is spent deciding how to build a system as quickly (and cheaply!) as possible, and too little time is spent considering desired outcome
- backward compatibility makes it harder and harder to actually make changes to your systems as more users exist
- maintenance programming is far more expensive (an order of magnitude less fun) than initially creating system

# Database design process

## four distinct phases

- Conceptual
- Logical
- Implementation
- Physical

# Conceptual

- sketch of database that you will get from initial requirements gathering and customer information
- attempt to identify what user wants
- find out about business process, scope, business rules that will govern use of data
- capture this information in conceptual data model consisting of set of high-level entities and interactions between them

# Logical

- refinement of work done in conceptual phase
- transforming conceptual design into full-fledged relational database design that will be foundation for implementation design
- define required set of entities, relationships between them, attributes of each entity, and domains of these attributes (i.e., sort of data attribute holds and range of valid values)

# Implementation

- adapt logical model for implementation in the host relational database management system

# Physical

- create model where implementation data structures are mapped to physical storage
- more or less performance tuning/optimization phase
- indexes, disk layouts, and so on, come into play, and not before this

# Conceptual database design

- define organizational and user data requirements of the system
- discovering and documenting a set of entities and relationships between them
- discovering and documenting business rules that define how data can and will be used and also the scope of the system

# Conceptual database design

- should capture fundamental sets of data that are required to support business processes and users' needs
- entity discovery is at the heart of this process
- entities correspond to nouns that are fundamental to business processes
- business statement such as the following
- ***People place orders to buy products.***
- identify three conceptual entities and begin to understand how they interact

- entity is not the same thing as a table
- table is an implementation-specific SQL construct
- sometimes entity will map directly to table
- some conceptual entities will be too abstract to ever be implemented
- sometimes they will map to two or more tables

- before starting database design process
- need to explore a few core relational database concepts

# Relational Model

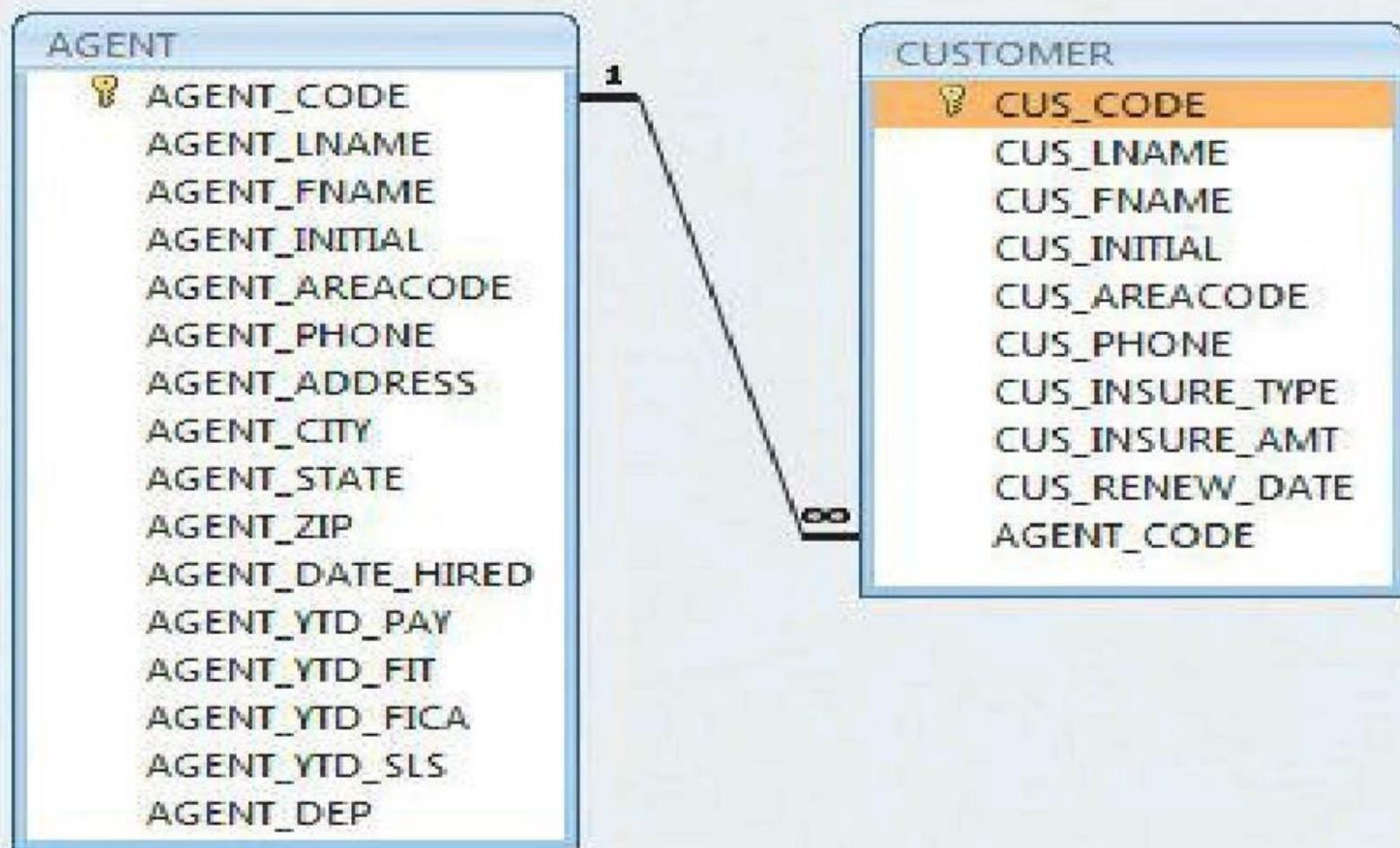
- introduced by E. F. Codd of IBM in his landmark paper “A Relational Model of Data for Large Shared Databanks” (Communications of the ACM, June 1970)
- set the stage for genuine database revolution
- foundation is mathematical concept known as **relation**
- computers at that time lacked power to implement relational model - Oracle

# Relational DataBase Management Systems

- hide complexities of relational model from user
- manages all of physical details, while user sees relational database as collection of tables in which data are stored - user can manipulate and query data in way that seems intuitive and logical through Structured Query Language

# Relational Diagram

- representation of relational database's entities, attributes within those entities, and relationships between those entities



- AGENT and relationship type, 1:M (MS Access, employs infinity symbol (  $\infty$  ) to indicate “many” side) in this example, CUSTOMER
- AGENT can have many CUSTOMERS
- AGENT represents “1” side because each CUSTOMER has only one AGENT

- from end-user perspective, any relational database application involves three parts
  1. user interface
  2. set of tables stored in database
  3. SQL database “engine”

# end-user interface

- allows end user to interact with data (by automatically generating SQL code)
- interface is product of software vendor's idea of meaningful interaction with data
- design your own customized interface with application generators

# collection of tables stored in database

- all data are perceived to be stored in tables
- tables simply “present” data to end user
- each table is independent
- rows in different tables are related by common values in common attributes

# DataBase Engine

- hidden from end user, SQL database engine executes all queries, or data requests
- is part of DBMS software
- end user uses SQL to create table structures and to perform data access and table maintenance
- engine processes all user requests
- SQL is said to be declarative language that tells what must be done but not how

# Entity Relationship ER Model

- become widely accepted standard for data modeling
- graphical representation of entities and their relationships in database structure
- normally represented in Entity Relationship Diagram which uses graphical representations to model database components

# Entity Relationship ER Model

## components

- **Entity** defined as anything about which data will be collected and stored
- represented in ERD by rectangle, also known as entity box
- name of the entity, noun, is written in center of rectangle
- generally written in capital letters and in singular form
- usually, when applying ERD to relational model, entity is mapped to relational table
- each row in relational table is known as entity instance or entity occurrence

# Entity Relationship ER Model

## components

- **Entity** defined as anything about which data will be collected and stored
- represented in ERD by rectangle, also known as entity box
- name of the entity, noun, is written in center of rectangle
- generally written in capital letters and in singular form
- usually, when applying ERD to relational model, entity is mapped to relational table
- each row in relational table is known as entity instance or entity occurrence

- collection of like entities is known as entity set
- technically speaking, ERD depicts entity sets
- ERD designers use word entity as substitute for entity set

# Entity Relationship ER Model

## components

- each entity consists of set of **Attributes** that describes particular characteristics of entity
- for example, entity EMPLOYEE will have attributes such as Social Security number, last name, and first name...

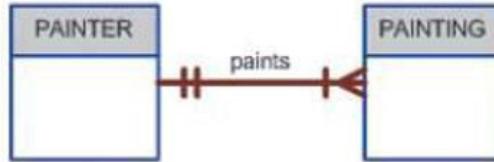
# Entity Relationship ER Model components

- **Relationships** describe associations among data
- most relationships describe associations between two entities
- three types of data relationships were illustrated: one-to-many (1:M), many-to-many (M:N), and one-to-one (1:1)
- uses term connectivity to label relationship types
- name of relationship is usually an active or passive verb
  - for example, PAINTER *paints* many PAINTINGs, EMPLOYEE *learns* many SKILLs, and EMPLOYEE *manages* STORE

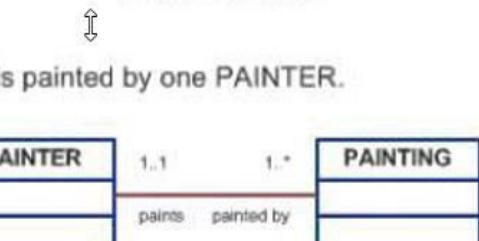
*Chen Notation*



*Crow's Foot Notation*

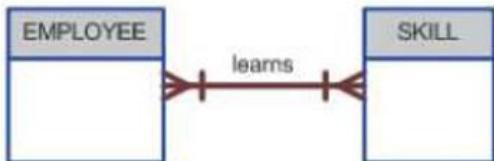


*UML Notation*

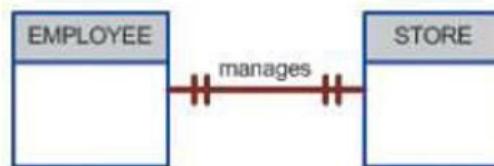


A One-to-Many (1:M) Relationship: a PAINTER can paint many PAINTINGS; each PAINTING is painted by one PAINTER.

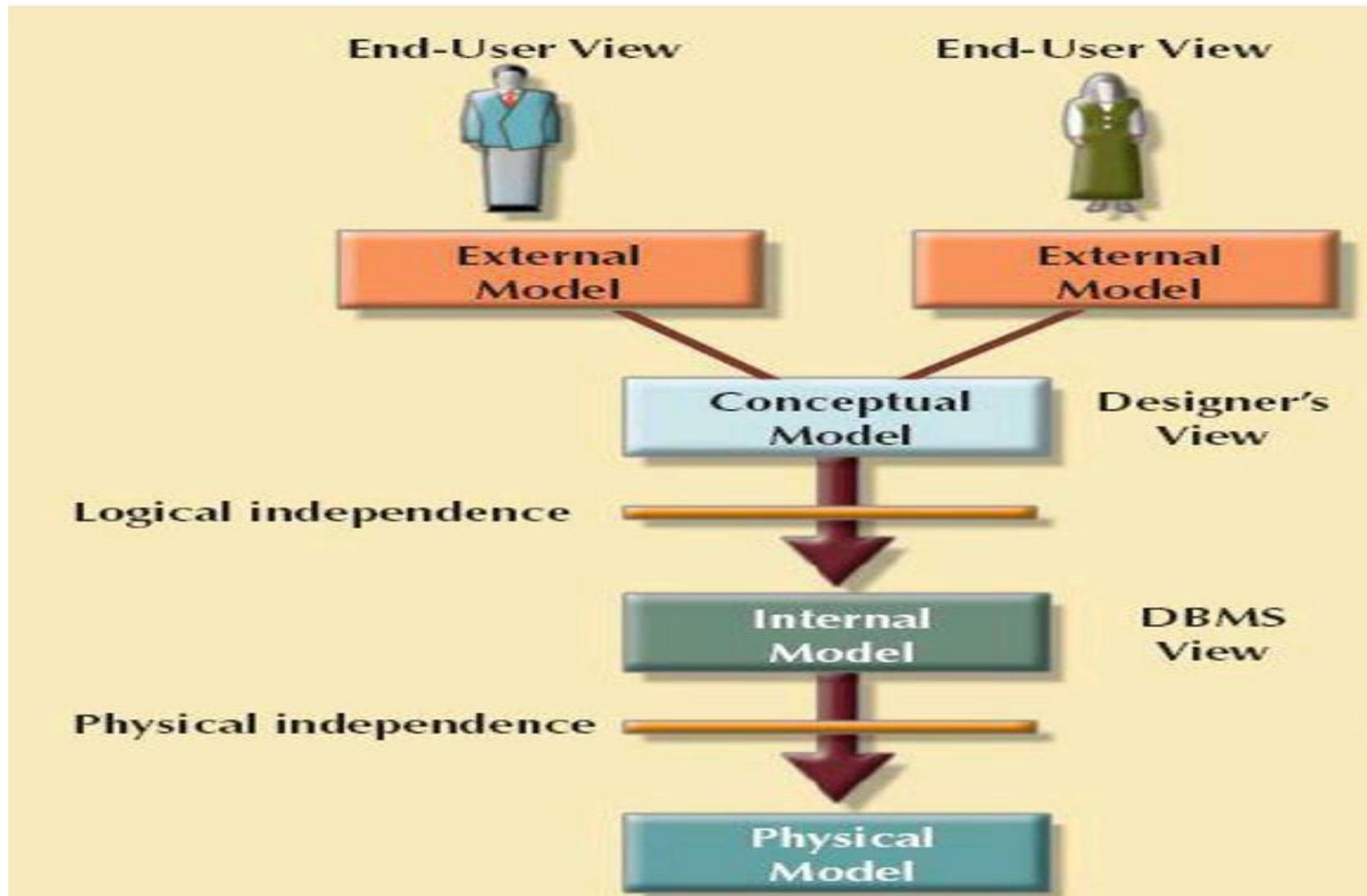
A Many-to-Many (M:N) Relationship: an EMPLOYEE can learn many SKILLS; each SKILL can be learned by many EMPLOYEES.



A One-to-One (1:1) Relationship: an EMPLOYEE manages one STORE; each STORE is managed by one EMPLOYEE.



# Data abstraction levels



# Relational DataBase Model

# Relational Model

- introduced by E. F. Codd in 1970, based on predicate logic and set theory
- predicate logic provides framework in which assertion (statement of fact) can be verified as either true or false
- set theory is mathematical science that deals with sets used as basis for data manipulation in relational model

# Relational Model

1. logical data structure represented by relations
2. set of integrity rules to enforce that data are and remain consistent over time
3. set of operations that defines how data are manipulated

# Relational Model

- enable to view data *logically* rather than *physically*
- because relation is mathematical construct, end users find it much easier to think of relation as table - table is two-dimensional structure composed of rows and columns
- think of table as *persistent* representation of logical relation
- table contains entity set, group of related entity occurrences
- *relation - entity set - table*

# Characteristics of Relational Table

1. table is two-dimensional structure composed of rows and columns
2. table row (tuple) represents single entity occurrence in entity set
3. table column represents attribute, column has distinct name
4. row & column intersect represents single (atomic) data value
5. values in column must conform to same data format, column has specific range of values known as attribute domain
6. order of rows and columns is immaterial – in set order of elements is irrelevant
7. table must have attribute or combination of attributes that uniquely identifies each row – set elements are unique

# Keys

- used to ensure that each row in table is uniquely identifiable
- used to establish relationships among tables and to ensure integrity of data
- consists of one or more attributes that determine other attributes
- role of key is based on concept of determination

# Functional Dependence

- state in which knowing value of one attribute makes it possible to determine value of another
- in database environment is not normally based on formula but on relationships among attributes
- means that value of one or more attributes determines value of one or more other attributes
- standard notation for representing relationship between STU\_NUM and STU\_LNAME is:
- $\text{STU\_NUM} \rightarrow \text{STU\_LNAME}$
- Student's No. ID and Student's Last Name

# Functional Dependence

- attribute whose value determines another called *determinant*
- attribute whose value is determined by others called *dependent*
- STUDENT table  $\text{STU\_NUM} \rightarrow$
- $(\text{STU\_LNAME}, \text{STU\_FNAME}, \text{STU\_INIT}, \text{STU\_DOB}, \text{STU\_HRS}, \text{STU\_CLASS}, \text{STU\_GPA})$

# Student Table

| STU_NUM | STU_LNAME | STU_FNAME | STU_INIT | STU_DOB     | STU_HRS | STU_CLASS | STU_GPA | STU_TRANSFER | DEPT_CODE | STU_PHONE | PROF_NUM |
|---------|-----------|-----------|----------|-------------|---------|-----------|---------|--------------|-----------|-----------|----------|
| 321452  | Bowser    | William   | C        | 12-Feb-1975 | 42      | Sr        | 2.84    | No           | BIOL      | 2134      | 205      |
| 324257  | Smithson  | Anne      | K        | 15-Nov-1981 | 81      | Jr        | 3.27    | Yes          | CIS       | 2256      | 222      |
| 324258  | Brewer    | Juliette  |          | 23-Aug-1969 | 36      | Sr        | 2.26    | Yes          | ACCT      | 2256      | 228      |
| 324269  | Oblonski  | Walter    | H        | 16-Sep-1976 | 66      | Jr        | 3.09    | No           | CIS       | 2114      | 222      |
| 324273  | Smith     | John      | D        | 30-Dec-1958 | 102     | Sr        | 2.11    | Yes          | ENGL      | 2231      | 199      |
| 324274  | Katinga   | Raphael   | P        | 21-Oct-1979 | 114     | Sr        | 3.15    | No           | ACCT      | 2267      | 228      |
| 324291  | Robertson | Gerald    | T        | 08-Apr-1973 | 120     | Sr        | 3.87    | No           | EDU       | 2267      | 311      |
| 324299  | Smith     | John      | B        | 30-Nov-1986 | 15      | Fr        | 2.92    | No           | ACCT      | 2315      | 230      |

|                     |                                                        |
|---------------------|--------------------------------------------------------|
| <b>STU_NUM</b>      | = Student number                                       |
| <b>STU_LNAME</b>    | = Student last name                                    |
| <b>STU_FNAME</b>    | = Student first name                                   |
| <b>STU_INIT</b>     | = Student middle initial                               |
| <b>STU_DOB</b>      | = Student date of birth                                |
| <b>STU_HRS</b>      | = Credit hours earned                                  |
| <b>STU_CLASS</b>    | = Student classification                               |
| <b>STU_GPA</b>      | = Grade point average                                  |
| <b>STU_TRANSFER</b> | = Student transferred from another institution         |
| <b>DEPT_CODE</b>    | = Department code                                      |
| <b>STU_PHONE</b>    | = 4-digit campus phone extension                       |
| <b>PROF_NUM</b>     | = Number of the professor who is the student's advisor |

# Keys

- composite key composed of more than one attribute
- attribute that is part of key is called key attribute
- superkey is key that can uniquely identify any row in table; functionally determines every attribute
- candidate key is specific type of superkey - minimal - without any unnecessary attributes
- table can have many different candidate keys
- they are eligible options from which designer will choose when selecting primary key
- primary key is candidate key chosen to be primary means by which rows of table are uniquely identified

# composite primary keys

- identifiers of composite entities, in which each primary key combination is allowed only once in M:N relationship
- identifiers of weak entities, in which weak entity has strong identifying relationship with parent entity

# Entity integrity rule

- integrity rule – check all entity
- condition in which each row (entity instance) in table has its own unique identity
- to ensure entity integrity, primary key has requirements
  1. all of values in primary key must be unique
  2. no attribute in Primary Key can contain null

# natural identifiers

- is real-world, generally accepted identifier used to distinguish - uniquely identify - real-world objects
- is familiar to users and forms part of their day-to-day business vocabulary

- *function of primary key is to guarantee entity integrity, not to “describe” entity*
- Primary Keys and Foreign Keys are used to implement relationships among entities
  - implementation is done mostly behind scenes, hidden from end users

# Desirable Primary Key Characteristics

- Unique values
- Nonintelligent
- No change over time
- Preferably single attribute
- Preferably numeric
- Security-compliant

# Desirable Primary Key Characteristics

- Unique values: PK must uniquely identify each entity instance; must be able to guarantee unique values; it cannot contain nulls
- Nonintelligent: PK should not have embedded semantic meaning other than to uniquely identify each entity instance; attribute with embedded semantic meaning is probably better used as descriptive characteristic of entity than as identifier

# Desirable Primary Key Characteristics

- No change over time: if attribute has semantic meaning, it might be subject to updates, which is why names do not make good primary keys; if PK is subject to change, FK values must be updated, thus adding to database work load
- PK should be permanent and unchangeable
- Preferably single attribute: should have minimum number of attributes possible (irreducible) desirable but not required
- Single-attribute Primary Keys simplify implementation of Foreign Keys; having multiple attribute PK can cause PK of related entities to grow through possible addition of many attributes, thus adding to database work load and making (application) coding more cumbersome

# Desirable Primary Key Characteristics

- Preferably numeric: unique values can be better managed when they are numeric, because database can use internal routines to implement counter-style attribute that automatically increments values with addition of each new row; most database systems include ability to use special constructs, such as AutoNumber, Increment, Identity, Sequence to support self-incrementing primary key attributes
- Security-compliant: must not be composed of any attribute(s) that might be considered security risk or violation; for example, using Social Security number as PK in EMPLOYEE table is not a good idea

# Surrogate Primary keys

- in some instances natural Primary Key doesn't exist in real world or existing might not be suitable
- standard practice is to create **surrogate key** PK created by database to simplify identification of entity instances
- has no meaning in user's environment - exists only to distinguish one entity instance from another
- it has no intrinsic meaning - values for it can be generated by DBMS to ensure that unique values are always provided (Globally Unique IDentifier )

- if you use surrogate key, must ensure that Alternate Key, the candidate key, performs properly through use of “unique index” and “not null” constraints

# Missing Values (NULLs)

- in database, there must exist some way to say that the value of a given column is not known or that the value is irrelevant
- often, a value outside of legitimate actual range
- Codd's third rules states the following:
- *NULL values (distinct from empty character string or string of blank characters or zero) supported in RDBMS for representing missing information in systematic way, independent of data type.*

# NULL

- is absence of any data value
- could represent unknown attribute value, known, but missing, attribute value, “not applicable” condition, ...

# properties of NULL

- any value concatenated with NULL is NULL.
- all math operations with NULL will return NULL
- logical comparisons can get tricky when NULL is introduced

# Referential integrity rule

- relationships between tables are implemented through common attributes as form of controlled redundancy
- for example next figure shows PRODUCT and VENDOR tables that are linked through common attribute, VEND\_CODE
- VEND\_CODE is referred to as Foreign Key in PRODUCT table - is Primary Key of another table to create common attribute
- Primary Key of VENDOR, VEND\_CODE, placed in PRODUCT table; VEND\_CODE Foreign Key in PRODUCT

**Table name: PRODUCT**

**Primary key: PROD\_CODE**

**Foreign key: VEND\_CODE**

**Database name:**

| PROD_CODE | PROD_DESCRPT                    | PROD_PRICE | PROD_ON_HAND | VEND_CODE |
|-----------|---------------------------------|------------|--------------|-----------|
| 001278-AB | Claw hammer                     | 12.95      | 23           | 232       |
| 123-21UUY | Houselite chain saw, 16-in. bar | 189.99     | 4            | 235       |
| QER-34256 | Sledge hammer, 16-lb. head      | 18.63      | 6            | 231       |
| SRE-657UG | Rat-tail file                   | 2.99       | 15           | 232       |
| ZZX/3245Q | Steel tape, 12-ft. length       | 6.79       | 8            | 235       |

link

**Table name: VENDOR**

**Primary key: VEND\_CODE**

**Foreign key: none**

| VEND_CODE | VEND_CONTACT       | VEND_AREACODE | VEND_PHONE |
|-----------|--------------------|---------------|------------|
| 230       | Shelly K. Smithson | 608           | 555-1234   |
| 231       | James Johnson      | 615           | 123-4536   |
| 232       | Annelise Crystall  | 608           | 224-2134   |
| 233       | Candice Wallace    | 904           | 342-6567   |
| 234       | Arthur Jones       | 615           | 123-3324   |
| 235       | Henry Ortozo       | 615           | 899-3425   |

# Referential integrity rule

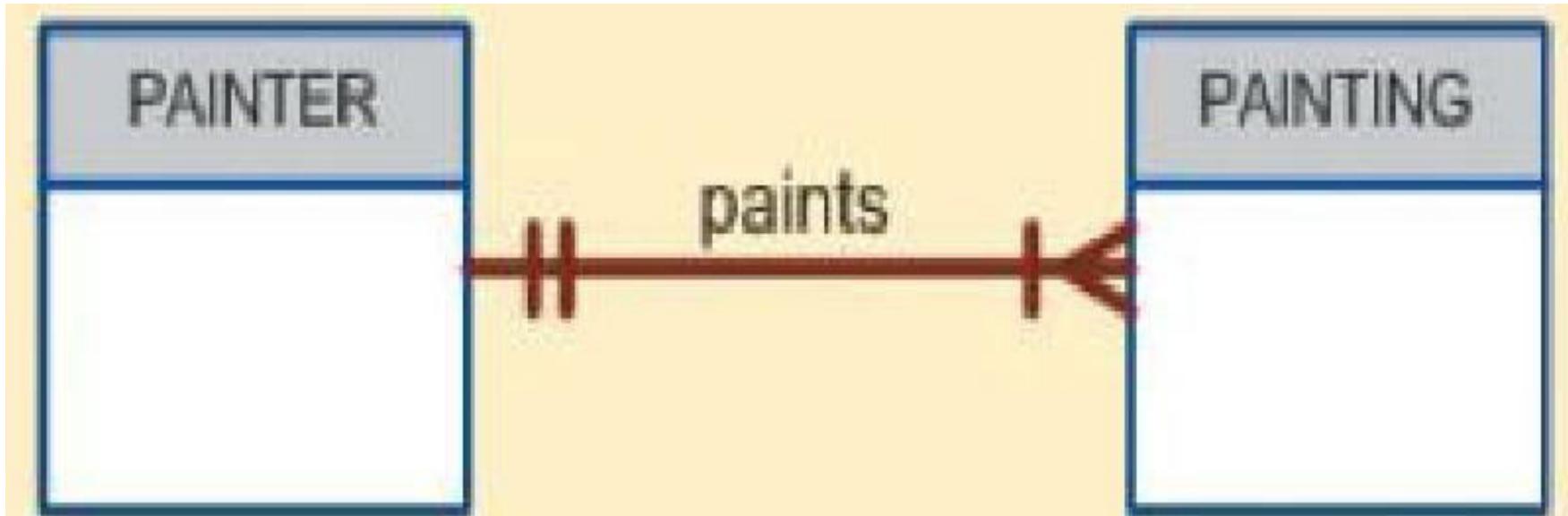
- every Foreign Key entry must either be null or valid value in Primary Key of related table
- every vendor referred to by row in PRODUCT table is valid vendor

- Alternate Key – Candidate Key not primary
- Secondary Key used strictly for data retrieval purposes - does not necessarily yield unique

# Relationships

- 1:M relationship is relational modeling ideal; should be norm in any relational database design
- 1:1 relationship should be rare
  - exception hierarchies
- M:N relationships cannot be implemented as such in relational model
  - can be changed into two 1:M relationships

# 1:M relationship “paints” between PAINTER and PAINTING



# PAINTER and PAINTING tables

Table name: PAINTER

Primary key: PAINTER\_NUM

Foreign key: none

Database

| PAINTER_NUM | PAINTER_LNAME | PAINTER_FNAME | PAINTER_INITIAL |
|-------------|---------------|---------------|-----------------|
| 123         | Ross          | Georgette     | P               |
| 126         | Itero         | Julio         | G               |

Table name: PAINTING

Primary key: PAINTING\_NUM

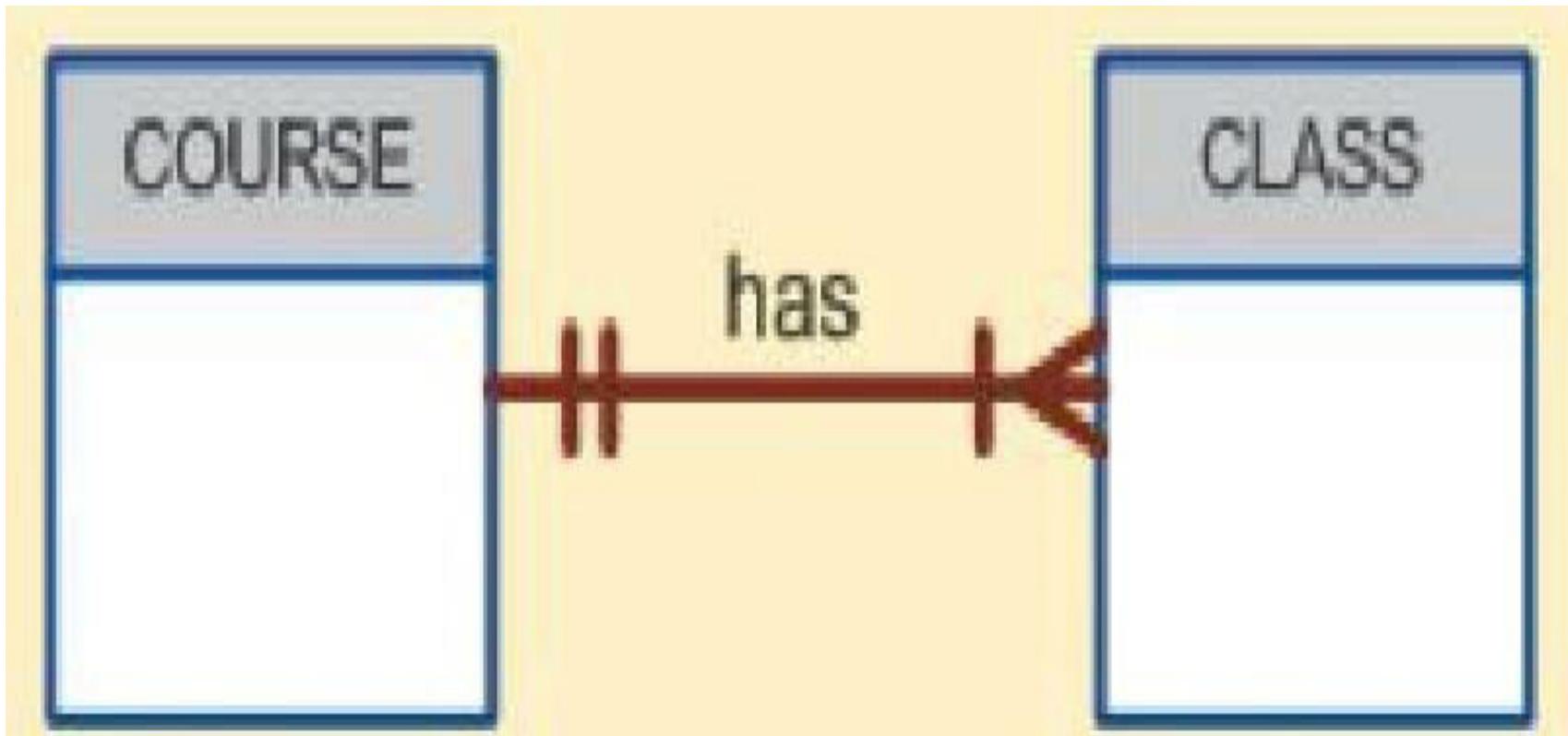
Foreign key: PAINTER\_NUM

| PAINTING_NUM | PAINTING_TITLE           | PAINTER_NUM |
|--------------|--------------------------|-------------|
| 1338         | Dawn Thunder             | 123         |
| 1339         | Vanilla Roses To Nowhere | 123         |
| 1340         | Tired Flounders          | 126         |
| 1341         | Hasty Exit               | 123         |
| 1342         | Plastic Paradise         | 126         |

- one-to-many (1:M) relationship implemented in relational model by putting Primary Key of “1” side in table of “many” side as Foreign Key

- students in typical college or university will discover that each COURSE can generate many CLASSes but that each CLASS refers to only COURSE
- 1 course DataBase Design – 2 classes in Romanian and English
- each COURSE can have many CLASSes
- each CLASS references only one COURSE

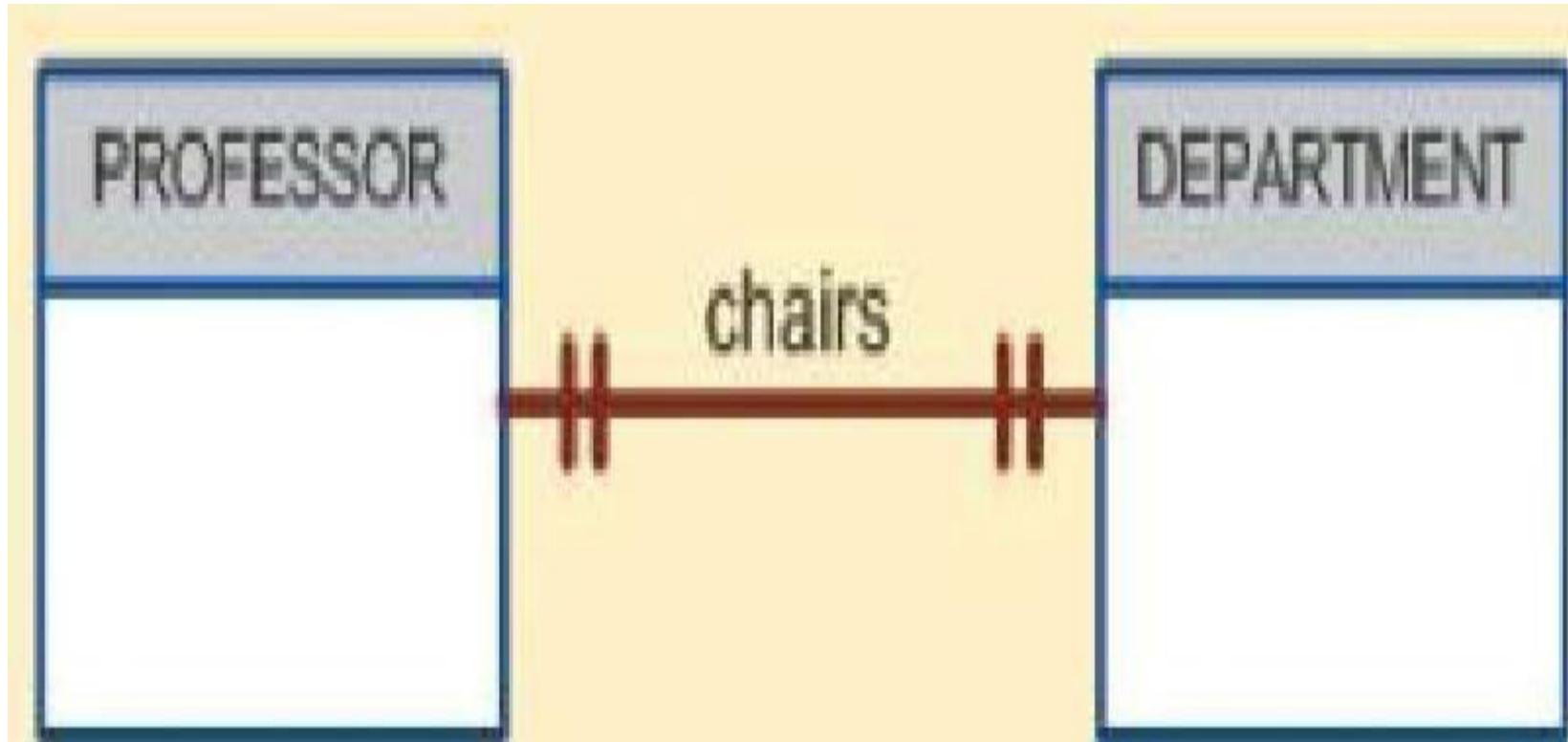
# Entity Relationship Model for 1:M relationship COURSE has CLASS



# 1:1 Relationship

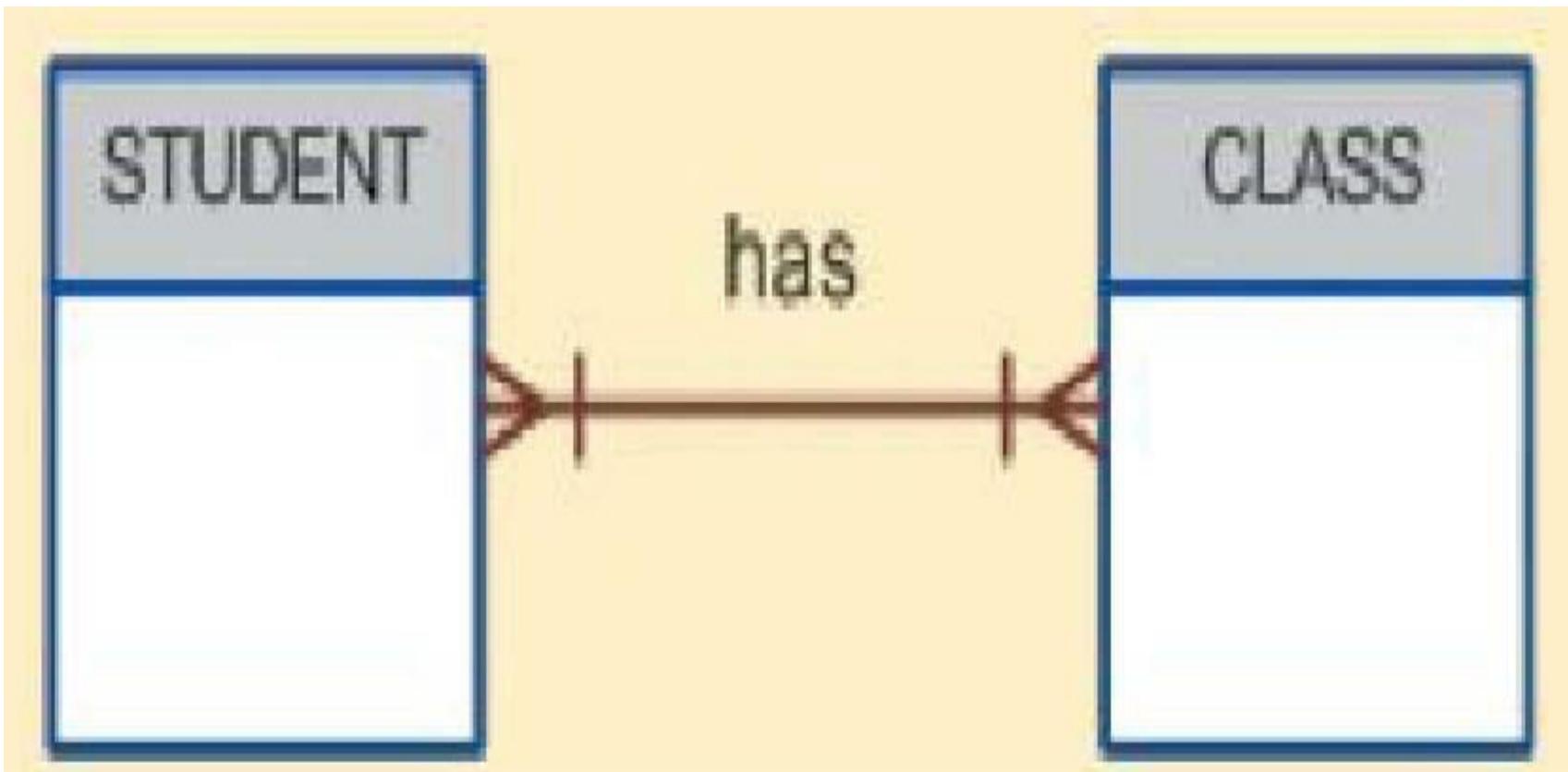
- for example, one department chair, a professor, can chair only one department, and one department can have only one department chair
- entities PROFESSOR and DEPARTMENT thus exhibit a 1:1 relationship
- treated as special case of 1:M relationship in which “many” side is restricted to single occurrence

# 1:1 relationship between PROFESSOR and DEPARTMENT



- existence of 1:1 relationship sometimes means that entity components were not defined properly - could indicate that two entities actually belong in same table
- use of 1:1 relationship ensures that two entity sets are not placed in same table

# ERModel M:N relationship between STUDENT and CLASS



# Student Enrollment

- each CLASS can have many STUDENTS
- each STUDENT can take many CLASSes
- problems (redundancies) inherent in many-to-many relationship avoided by creating ***composite entity*** (bridge, associative, link entity)
- such a table is used to link tables that were originally related in M:N relationship, composite entity structure includes - as Foreign Keys - at least Primary Keys of tables that are to be linked
- designer has two main options: use combination of those FK or create new PK

# Converting M:N relationship into two 1:M relationships

**Table name: STUDENT**

**Primary key: STU\_NUM**

**Foreign key: none**

| STU_NUM | STU_LNAME |
|---------|-----------|
| 321452  | Bowser    |
| 324257  | Smithson  |

**Table name: ENROLL**

**Primary key: CLASS\_CODE + STU\_NUM**

**Foreign key: CLASS\_CODE, STU\_NUM**

| CLASS_CODE | STU_NUM | ENROLL_GRADE |
|------------|---------|--------------|
| 10014      | 321452  | C            |
| 10014      | 324257  | B            |
| 10018      | 321452  | A            |
| 10018      | 324257  | B            |
| 10021      | 321452  | C            |
| 10021      | 324257  | C            |

**Table name: CLASS**

**Primary key: CLASS\_CODE**

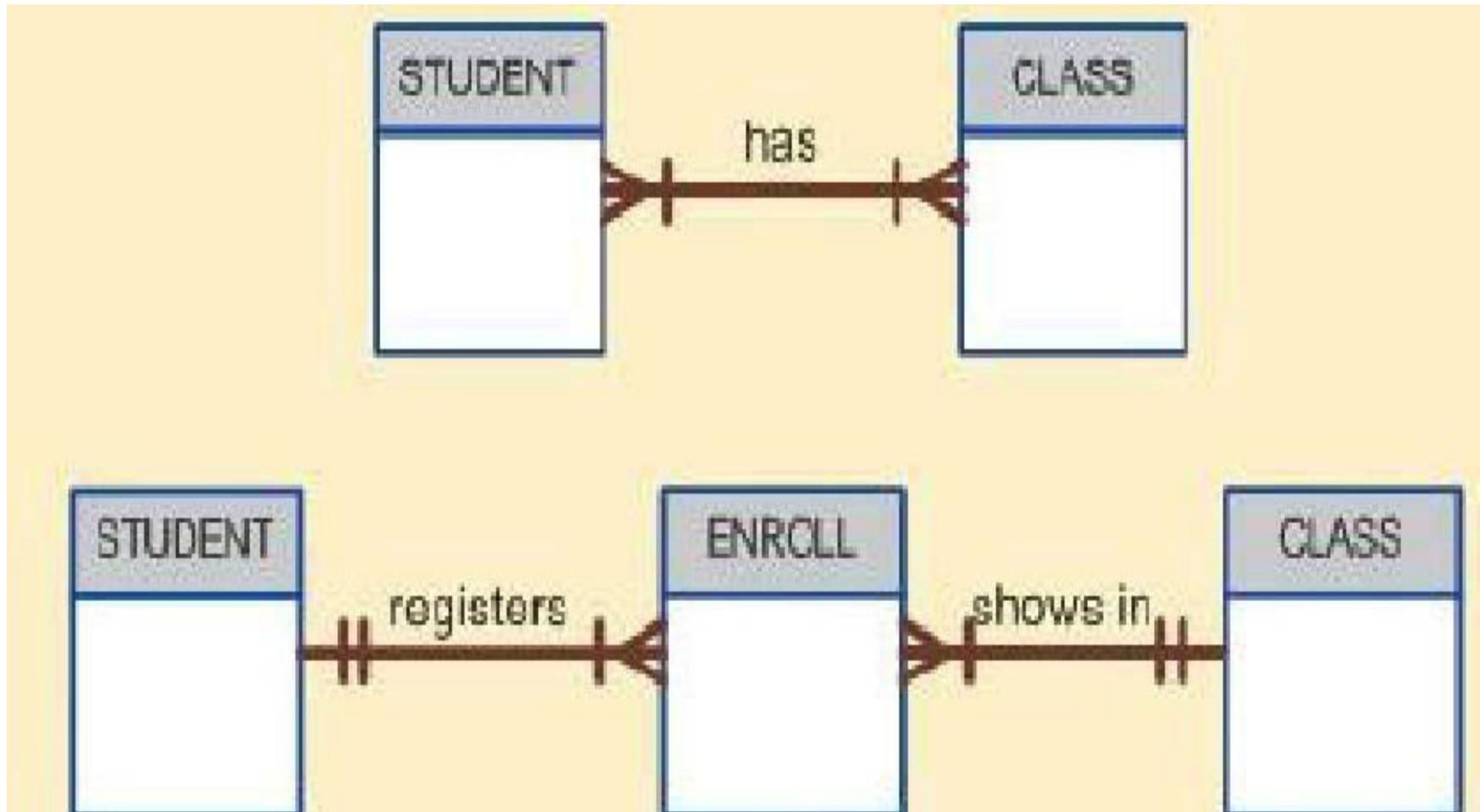
**Foreign key: CRS\_CODE**

| CLASS_CODE | CRS_CODE | CLASS_SECTION | CLASS_TIME         | CLASS_ROOM | PROF_NUM |
|------------|----------|---------------|--------------------|------------|----------|
| 10014      | ACCT-211 | 3             | TTh 2:30-3:45 p.m. | BUS252     | 342      |
| 10018      | CIS-220  | 2             | MWF 9:00-9:50 a.m. | KLR211     | 114      |
| 10021      | QM-261   | 1             | MWF 8:00-8:50 a.m. | KLR200     | 114      |

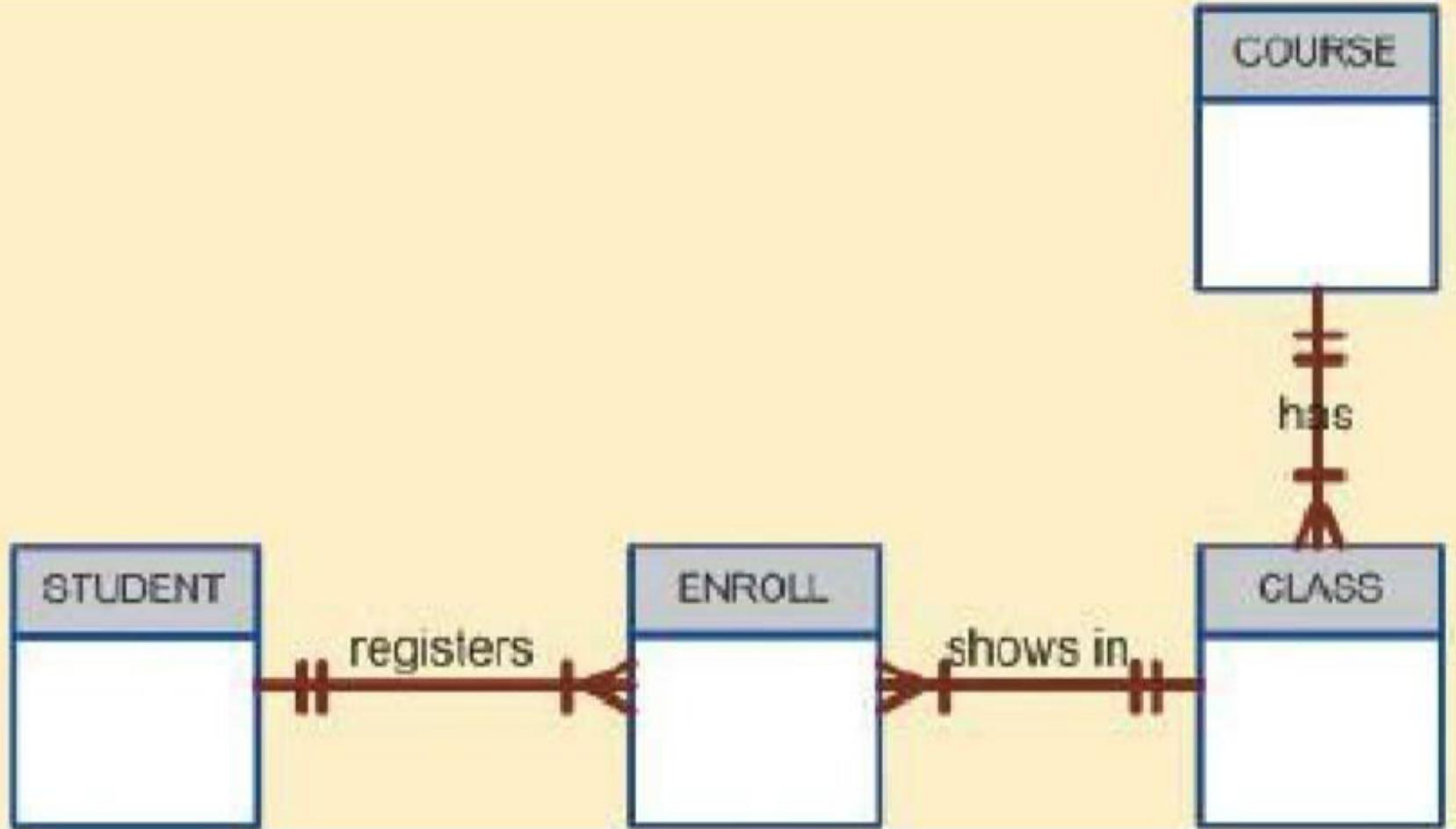
# Linking Table

- ENROLL table links two tables, STUDENT and CLASS is implementation of composite entity
- in addition to linking attributes, composite table ENROLL can also contain such relevant attributes as grade earned in course – attributes describe relationship

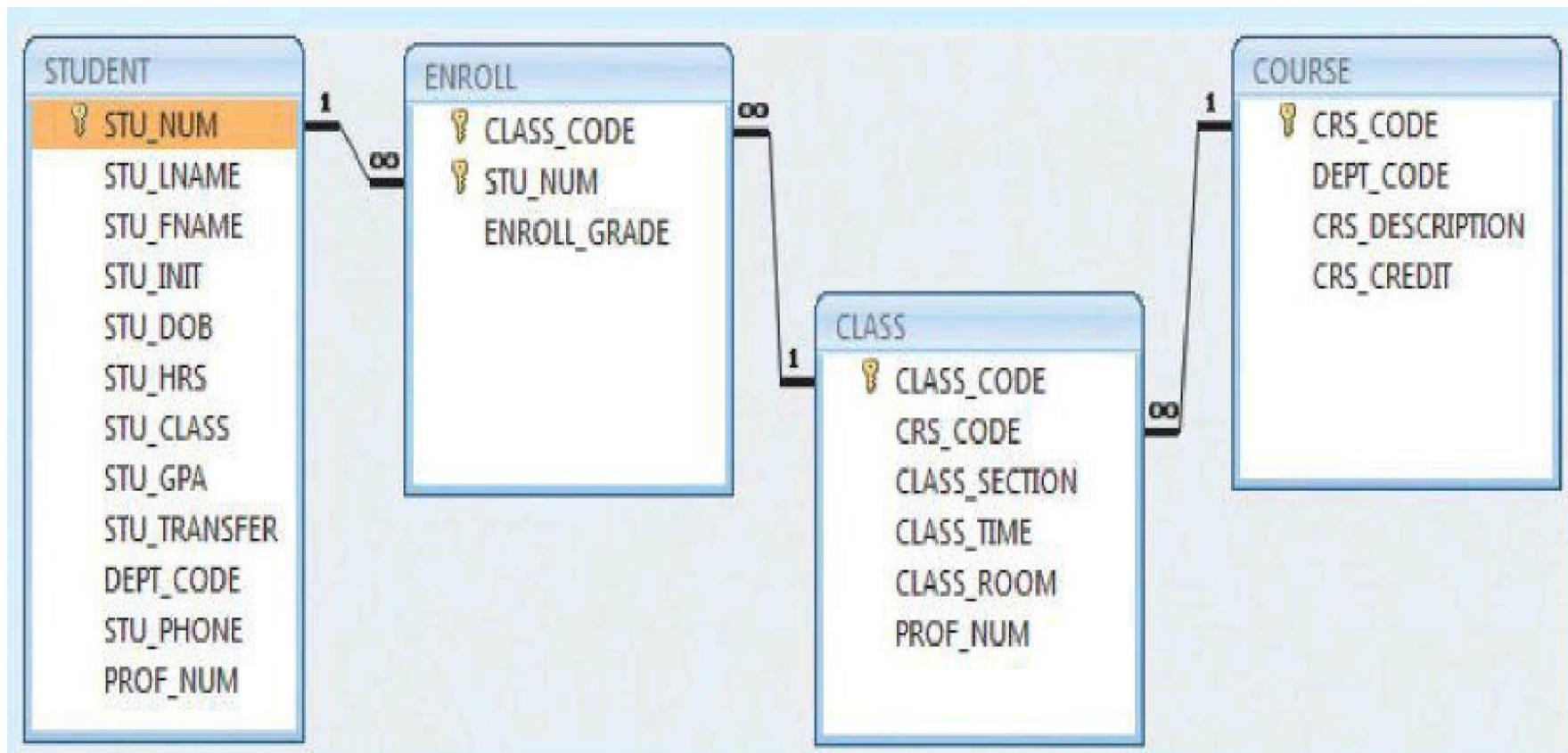
# Converting M:N relationship into two 1:M relationships



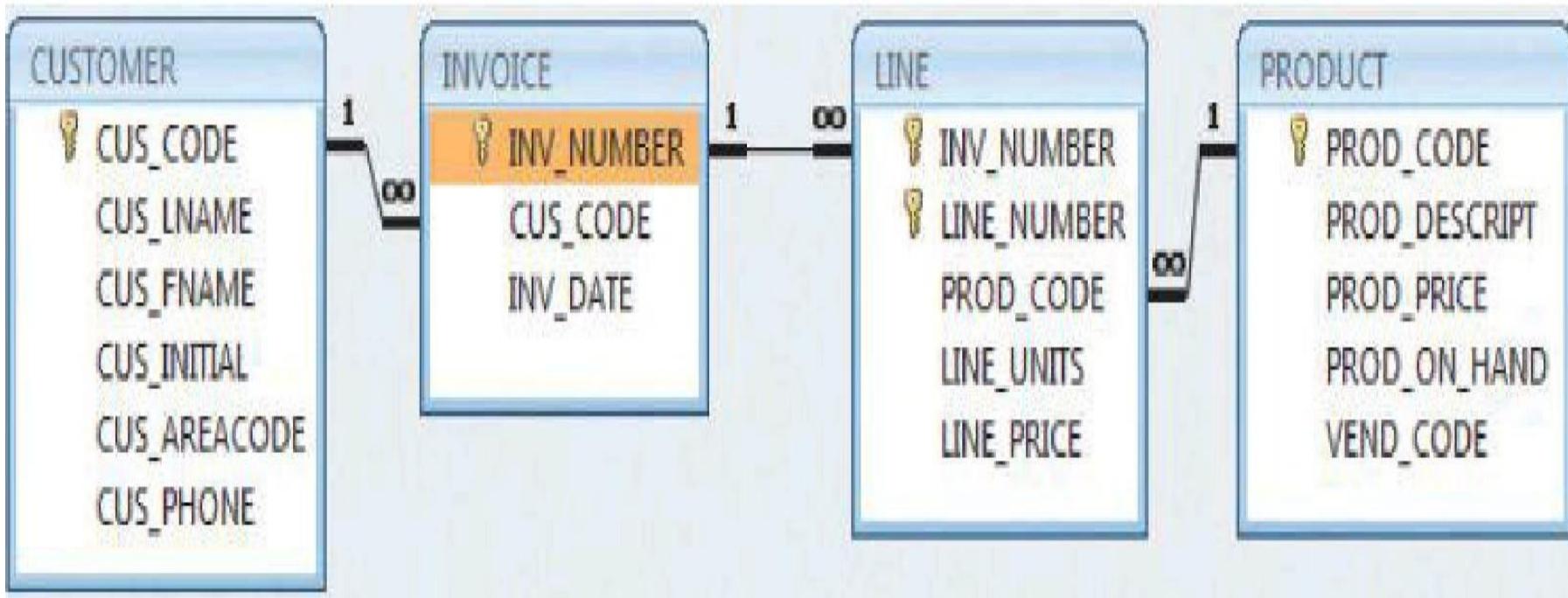
# Entity Relationship Model



# Relational Diagram



# Relational diagram for invoicing system



# Codd's Relational DataBase Rules

- rules to define relational database system
- database products meet minimum relational standards
- *dominant database vendors do not fully support all rules*

# Codd's Relational DataBase Rules

1. Information: all information in Relational Database must be logically represented as column values in rows within tables
2. Guaranteed access: every value in table is guaranteed to be accessible through combination of table name, primary key value, and column name
3. Systematic treatment of nulls: nulls must be represented and treated in systematic way, independent of data type

# Codd's Relational DataBase Rules

4. Dynamic online catalog based on relational model: metadata must be stored and managed as ordinary data, in tables within database; must be available to authorized users using standard database relational language
5. Comprehensive data sublanguage: relational database may support many languages; must support one well-defined, declarative language as well as data definition, view definition, data manipulation (interactive and by program), integrity constraints, authorization, and transaction management
6. View updating: any view that is theoretically updatable must be updatable through system

# Codd's Relational DataBase Rules

7. High-level insert, update, and delete: database must support set-level inserts, updates, and deletes
8. Physical data independence: application programs and ad hoc facilities are logically unaffected when physical access methods or storage structures are changed
9. Logical data independence: application programs and ad hoc facilities are logically unaffected when changes are made to table structures that preserve original table values (changing order of columns or inserting columns)

# Codd's Relational DataBase Rules

10. Integrity independence: all relational integrity constraints must be definable in relational language and stored in system catalog, not at application level
11. Distribution independence: end users and application programs are unaware of and unaffected by data location (distributed vs. local)
12. Nonsubversion: if system supports low-level access to data, users must not be allowed to bypass integrity rules of database

# Entity Relationship Modeling

# Entity Relationship Modeling

- conceptual models such as Entity Relationship Modeling can be used to understand and design data requirements of organization independent of database type (hierarchical, network, relational, object-oriented, ...)
- conceptual models are used in conceptual design of databases
- study mostly relational database model - relational model is used to explain ER and way to develop database designs

# Modeling languages

- Integration Definition for Information Modeling (IDEF1X)
- Information Engineering
- Chen Entity Relationship Model (ERD) methodology
- diagram capabilities built into SQL Server Management Studio
- [SQL Power Architect](#) Community Edition

# Entity Relationship Diagrams

- Entity Relationship Diagrams represents conceptual database; depict entities, attributes, relationships
- Chen ERD notation favors conceptual modeling
- Crow's Foot, IDEF1X, IE notations favors more implementation-oriented approach
- UML notation can be used for both conceptual and implementation modeling

# Entities

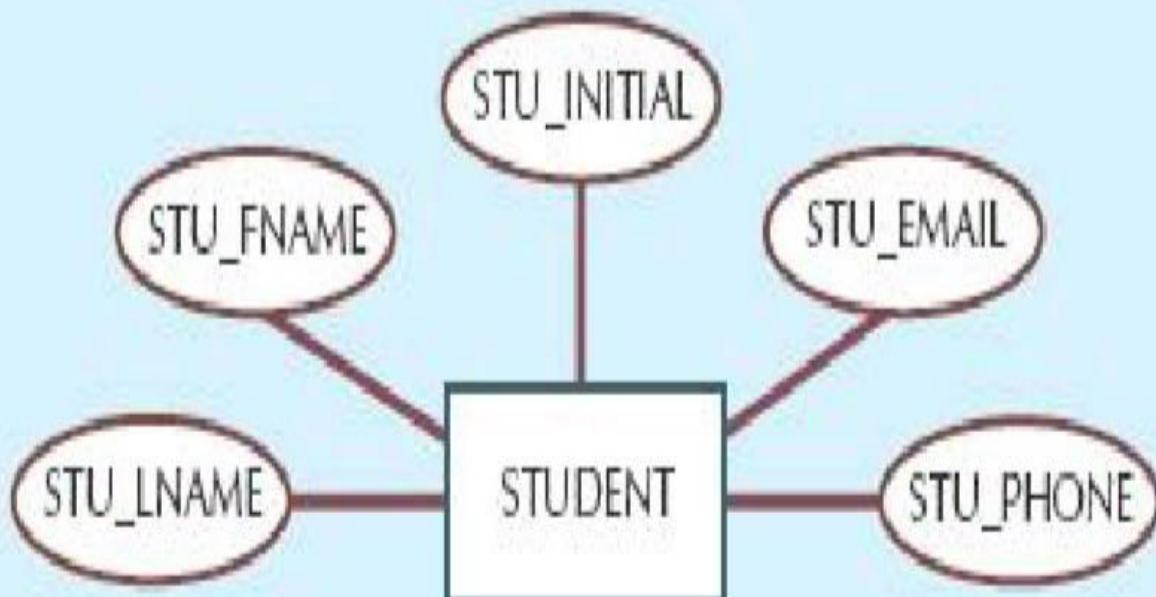
- entity is object of interest to user
- refers to entity set and not to single entity occurrence - corresponds to table
- represented by rectangle that contains entity's name - usually noun written in all capital letters

# Attributes

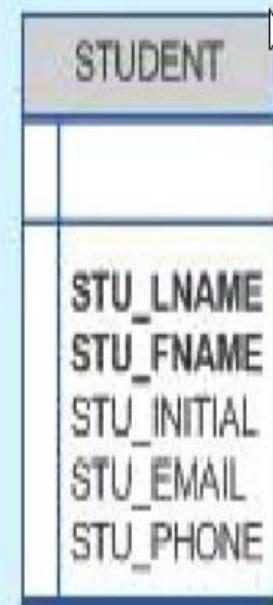
- are characteristics of entities
- in Chen notation are represented by ovals and connected to entity rectangle with line; each oval contains name of attribute it represents
- in Crow's Foot notation, attributes are written in attribute box below entity rectangle
- Chen representation consumes more space, software vendors have adopted Crow's Foot attribute display

# attributes of STUDENT entity

Chen Model



Crow's Foot Model



# Mandatory / Optional attributes

- Mandatory (Required) attribute is attribute that must have value; cannot be left empty
- Optional attribute is attribute that does not require value; therefore, it can be left empty
- two boldfaced attributes in Crow's Foot notation indicate that data entry will be required
- STU\_LNAME and STU\_FNAME require data entries because all students are assumed to have names
- students might not have middle name, and perhaps do not yet have phone number and e-mail address

# Domain

- set of possible values for attribute
- for example, domain for grade point average (GPA) is written (0, 10), domain for gender consists of M or F (or some other equivalent code), ...
- attributes may share domain
- for instance, student address and professor address share same domain of all possible addresses

# Identifiers (Primary Keys)

- attributes that uniquely identify each entity instance
- mapped as table's Primary Key (PK) or Alternate Key (AK) - are underlined in diagram
- **relational schema** frequently used shorthand notation for table structure uses following format:
- TABLE NAME (*KEY\_ATTRIBUTE 1, KEY\_ATTRIBUTE 2, ATTRIBUTE 3, ... ATTRIBUTE N*)
- for example: CAR (CAR\_VIN, MODEL\_CODE, CAR\_YEAR, CAR\_COLOR)

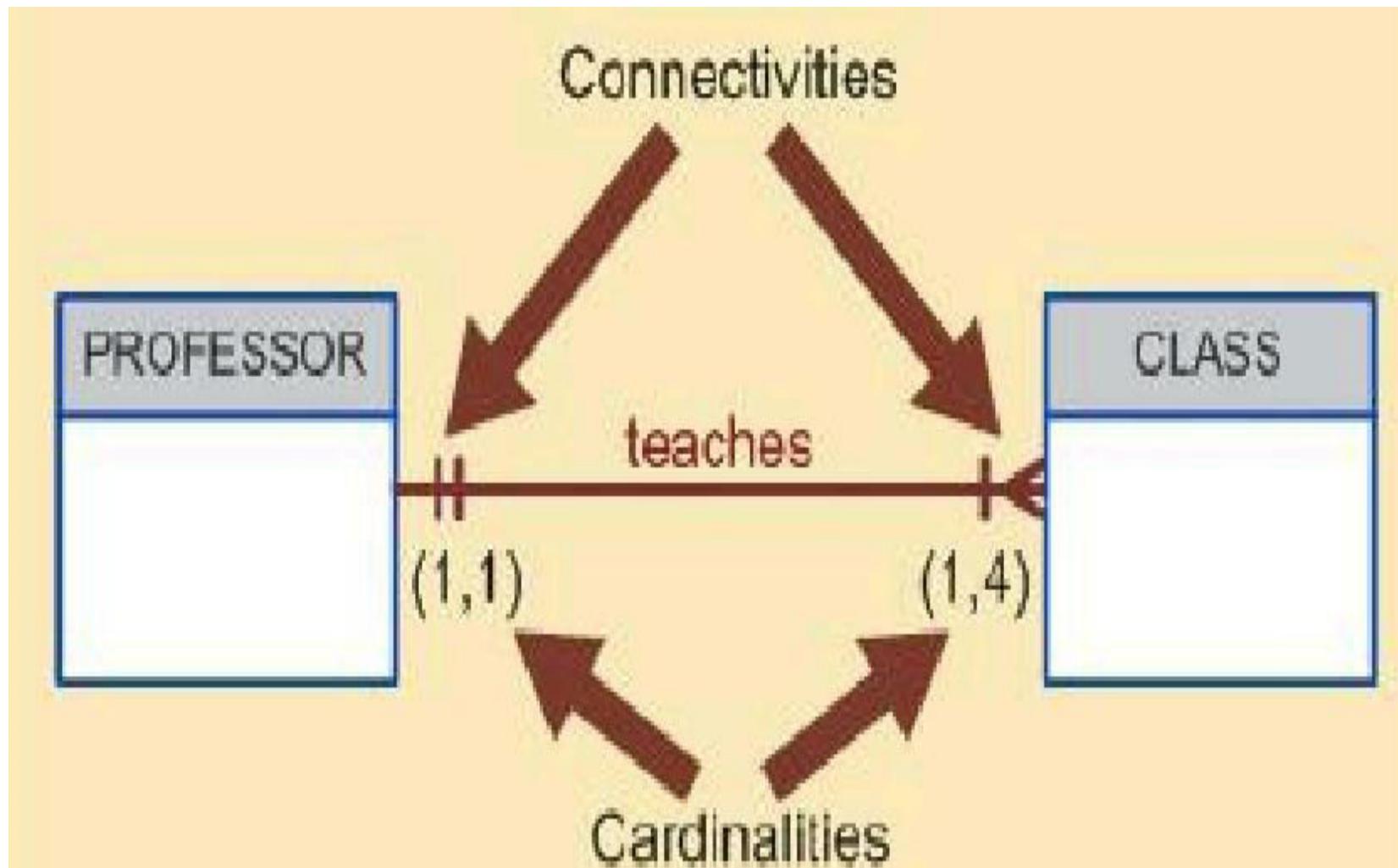
# Relationship

- participants: entities that are in relationship
- relationship name: active or passive verb
- always operate in both directions
  - a CUSTOMER may generate many INVOICES
  - each INVOICE is generated by one CUSTOMER
- this relationship can be classified as 1:M

# Connectivity and Cardinality

- connectivity: used to describe relationship classification
- cardinality: expresses minimum and maximum number of entity occurrences associated with one occurrence of related entity
- established by business rules
- in ERD indicated by placing appropriate numbers beside entities, using format (x,y): first value represents minimum number of associated entities, while second value represents maximum number

# Connectivity and Cardinality



# Existence-Dependent

- entity is existence-dependent if it can exist in database only when it is associated with another related entity occurrence
- if it has mandatory Foreign Key attribute that cannot be null
- for example, if employee wants to claim one or more dependents for tax-withholding purposes, relationship “EMPLOYEE claims DEPENDENT” would be appropriate; in that case, DEPENDENT entity is clearly existence-dependent on EMPLOYEE entity because it is impossible for dependent to exist apart from EMPLOYEE in database

# Existence-Independent

- entity can exist apart from all of its related entities
- referred to as a **strong entity** or regular entity
- for example, suppose that company uses parts to produce its products and some of those parts are produced in-house and other parts are bought from vendors; possible for PART to exist independently from VENDOR in relationship “PART is supplied by VENDOR” because at least some of the parts are not supplied by vendor - therefore, PART is existence-independent from VENDOR

# Relationship Strength

- concept based on how PK of related entity is defined
- sometimes FK also is PK component in related entity
- relationship strength decisions affect primary key arrangement in database design

# Weak (Non-Identifying) Relationships

- exists if PK of related entity does not contain PK=FK component of parent entity
- for example, suppose 1:M relationship between COURSE and CLASS:
- COURSE (**CRS\_CODE**, DEPT\_CODE, CRS\_DESCRIPTION, CRS\_CREDIT)
- CLASS (**CLASS\_CODE**, CRS\_CODE, CLASS\_SECTION, CLASS\_TIME, ROOM\_CODE, PROF\_NUM)
- PK of parent entity is only FK in child entity
- CLASS Primary Key did not inherit Primary Key component from COURSE entity

| COURSE |                 |
|--------|-----------------|
| PK     | <u>CRS_CODE</u> |
|        | DEPT_CODE       |
|        | CRS_DESCRIPTION |
|        | CRS_CREDIT      |



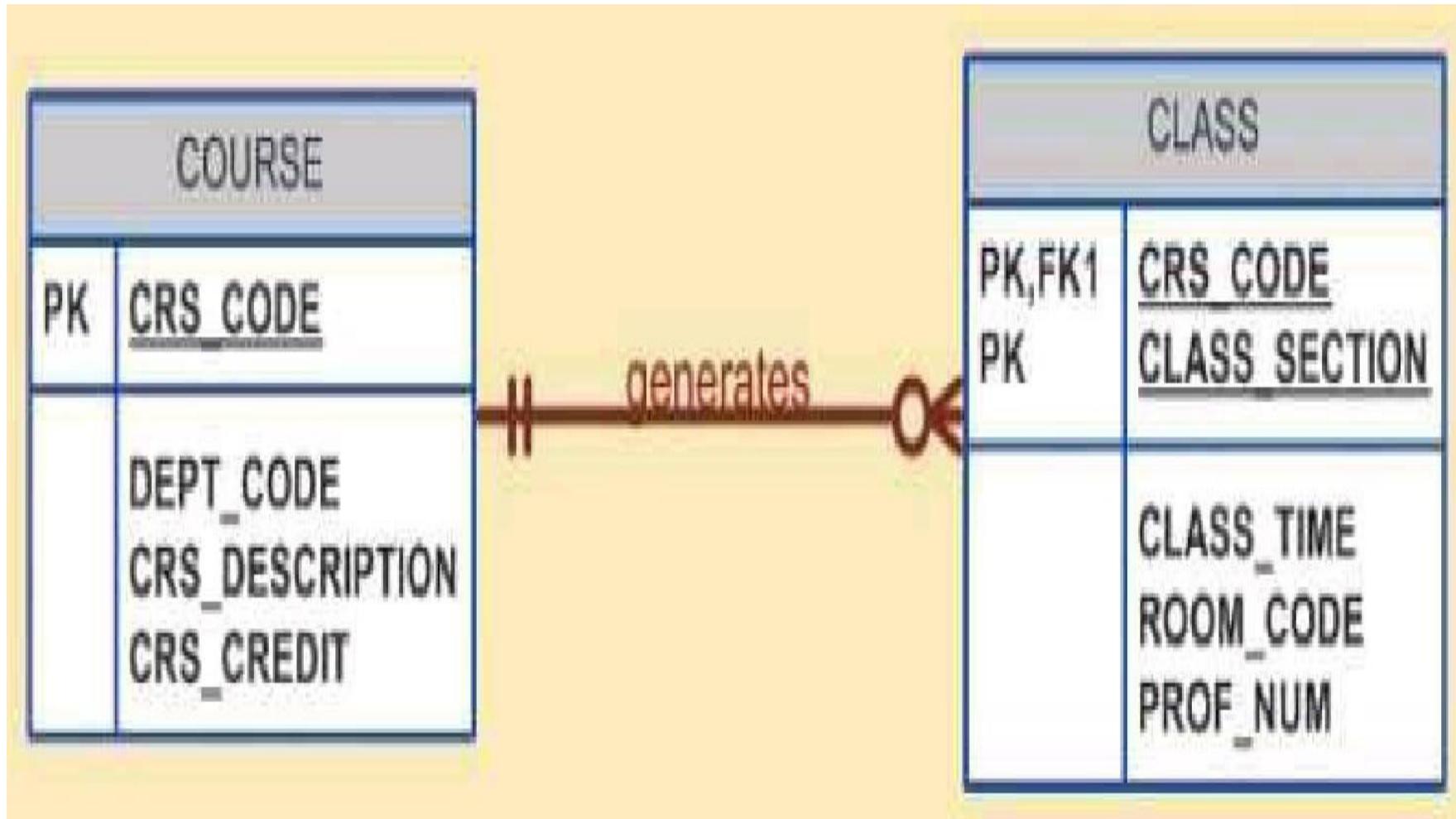
| CLASS |                   |
|-------|-------------------|
| PK    | <u>CLASS_CODE</u> |
| FK1   | <u>CRS_CODE</u>   |
|       | CLASS_SECTION     |
|       | CLASS_TIME        |
|       | ROOM_CODE         |
|       | PROF_NUM          |

# Strong (Identifying) Relationships

- exists when PK of related entity contains PK=FK component of parent entity

# Strong (Identifying) Relationships

- for example, suppose 1:M relationship between COURSE and CLASS:
- COURSE (**CRS\_CODE**, DEPT\_CODE, CRS\_DESCRIPTION, CRS\_CREDIT)
- CLASS (**CRS\_CODE**, **CLASS\_SECTION**, CLASS\_TIME, ROOM\_CODE, PROF\_NUM)
- CLASS entity Primary Key is composed of CRS\_CODE and CLASS\_SECTION;
- Primary Key of parent entity is Foreign Key and Primary Key component in child entity
- CLASS PK inherit PK component from COURSE entity - CRS\_CODE in CLASS is also FK to COURSE entity



- relationship between COURSE and CLASS is strong or weak depends on how CLASS entity's Primary Key is defined
- **Weak Entities**
  - entity is existence-dependent; it cannot exist without entity with which it has relationship
  - entity has Primary Key that is partially or totally derived from parent entity in relationship

# Relationship Participation

- participation in entity relationship is either optional or mandatory
- must consider bidirectional nature of relationship when determining participation

# Optional participation

- means that one entity occurrence does not require corresponding entity occurrence in particular relationship
- some courses do not generate class - entity occurrence (row) in COURSE table does not necessarily require existence of corresponding entity occurrence in CLASS table
- CLASS entity is considered to be optional to COURSE entity
- in Crow's Foot notation is shown by drawing small circle (O) on side of optional entity
- indicates that its minimum cardinality is 0

# Mandatory participation

- means that one entity occurrence requires corresponding entity occurrence in particular relationship
- if no optionality symbol is depicted it is assumed to exist in mandatory relationship with related entity
- indicates that minimum cardinality is at least 1 for mandatory entity

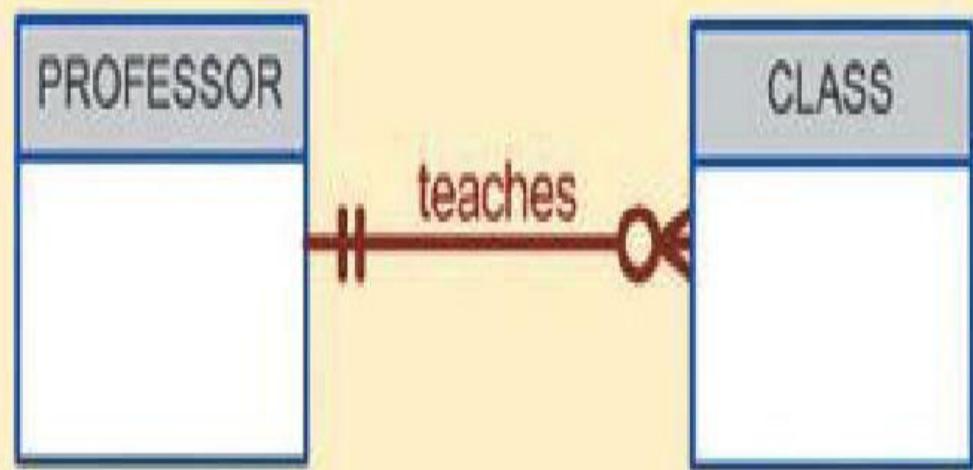
# Relationship Degree

- indicates number of entities or participants associated with relationship
- ***unary*** when association is maintained within single entity
- ***binary*** when two entities are associated
- ***ternary*** when three entities are associated
- ***higher degrees*** exist, they are rare and are not specifically named

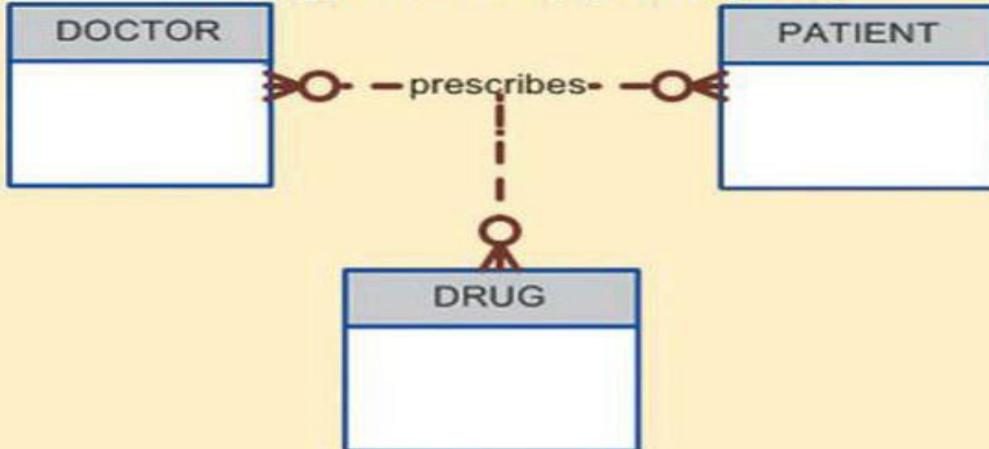
## Unary relationship



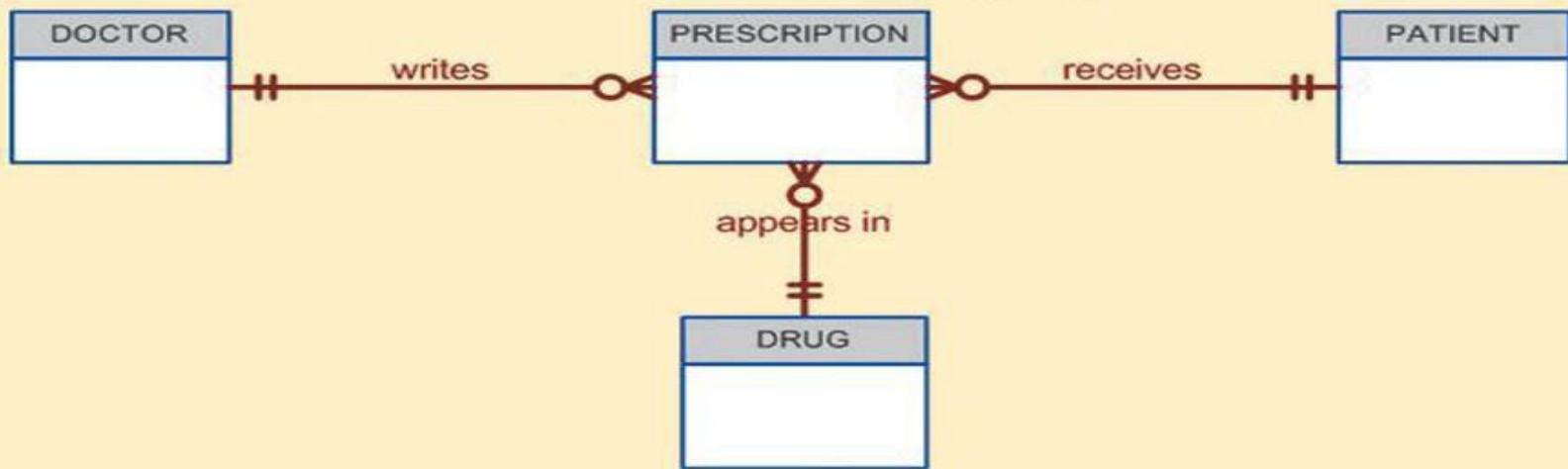
## Binary relationship



### Ternary relationship (Conceptual)



### Ternary relationship (Logical)



# Recursive relationship

- is one in which relationship can exist between occurrences of same entity set - found within unary relationship
- for example, 1:M unary relationship can be expressed by “an EMPLOYEE may manage many EMPLOYEES, and each EMPLOYEE is managed by one EMPLOYEE.”

# Developing ER diagram

- process of database design is iterative rather than linear or sequential
- create detailed narrative of organization's description of operations
- identify business rules based on description of operations
- identify main entities and relationships
- develop initial Entity Relationship Diagram
- identify attributes and identifiers that adequately describe entities
- revise and review ER diagram

# Naming conventions

- names should be limited in length (database-dependent size)

# Entity names

- should be nouns that are familiar to business and should be short and meaningful
- should document abbreviations, synonyms, and aliases for each entity
- should be unique within model
- for composite entities, may include combination of abbreviated names of entities linked through composite entity

# Attribute names

- should be unique within entity
- should use entity abbreviation as prefix
- should be descriptive of characteristic
- should use suffixes such as `_ID`, `_NUM`, or `_CODE` for PK attribute
- should not be reserved word
- should not contain spaces or special characters such as `@`, `!`, or `&`

# Relationship names

- should be active or passive verbs that clearly indicate nature of relationship

# Entities

- each entity should represent single subject
- each entity should represent set of distinguishable entity instances
- granularity of entity instance should be clearly defined
- Primary Key should be clearly defined and support selected data granularity

# Attributes

- should be simple and single-valued (atomic data)
- should document default values, constraints, synonyms, and aliases
- derived attributes should be clearly identified and include source(s)
- should not be redundant unless this is required for transaction accuracy, performance, or maintaining history

# Relationships

- should clearly identify relationship participants
- should clearly define participation, connectivity, and document cardinality

# Entity Relationship Model

- should be validated against expected processes: inserts, updates, and deletes
- should evaluate where, when, and how to maintain history
- should not contain redundant relationships except as required
- should minimize data redundancy to ensure single-place updates
- should conform to minimal data rule: all that is needed is there, and all that is there is needed

# Extended Entity Relationship Model

- enhanced Entity Relationship Model
- adding more semantic constructs to
- entity supertypes, subtypes
- entity clustering

# **Advance Data Modeling**

## **Specialization Generalization Hierarchies**

# **ENTITY SUPERTYPES AND SUBTYPES**

# Entity Supertypes and Subtypes

- because most employees possess wide range of attributes data modelers must find ways to group employees based on their characteristics
- for instance, retail company could group employees as salaried and hourly, while university could group employees as faculty, staff, and administrators

- grouping of employees into various *types* provides benefits:
  1. avoids unnecessary nulls in attributes when some employees have characteristics that are not shared by other employees
  2. enables particular employee type to participate in relationships that are unique to that type

- for example use aviation business, which employs pilots, mechanics, secretaries, accountants, database managers, and many other types of employees
- pilots share certain characteristics with other employees, such as last name and hire date
- many pilot characteristics are not shared by other employees: special requirements flight hour restrictions, flight checks, periodic training
- pilots participate in some relationships that are unique to their qualifications: only certified employees = pilots can participate in “employee flies airplane” relation

- if all Up - employee characteristics and special attributes were stored in single EMPLOYEE entity, would have lot of nulls or lot of needless dummy entries; special pilot attributes such as EMP\_LICENSE, EMP\_RATINGS, and EMP\_MED\_TYPE will generate nulls for employees who are not pilots

- if all Down - employee characteristics and special attributes were stored in many PILOTS, MECHANICS, SECRETARIES, ACCOUNTATNTS entities, would have to make a UNION to access common attributes to find all employees salary

- PILOT entity stores only attributes that are unique to pilots
- EMPLOYEE entity stores attributes that are common to all employees
- PILOT is *subtype* of EMPLOYEE
- EMPLOYEE is *supertype* of PILOT
- **entity supertype** is generic entity type that is related to one or more **entity subtypes** (each contain their own unique attributes, characteristics)

# criteria help designer determine when to use subtypes and supertypes:

1. there must be different, identifiable kinds or types of entity in user's environment
2. different kinds or types of instances should each have one or more attributes that are unique to that kind or type of instance

# entity supertypes and subtypes are related in specialization / generalization hierarchy

- pilots identifiable kind of employee and have unique attributes that other employees do not possess, it is appropriate to create PILOT as subtype of EMPLOYEE
- mechanics and accountants also each have attributes that are unique to them, respectively, and clerks do not
- MECHANIC and ACCOUNTANT would also be legitimate subtypes of EMPLOYEE because they are identifiable kinds of employees and have unique attributes
- CLERK would *not* be acceptable subtype of EMPLOYEE because it only satisfies one criteria - it is an identifiable kind of employee - but none of attributes are unique to just clerks

# specialization hierarchy

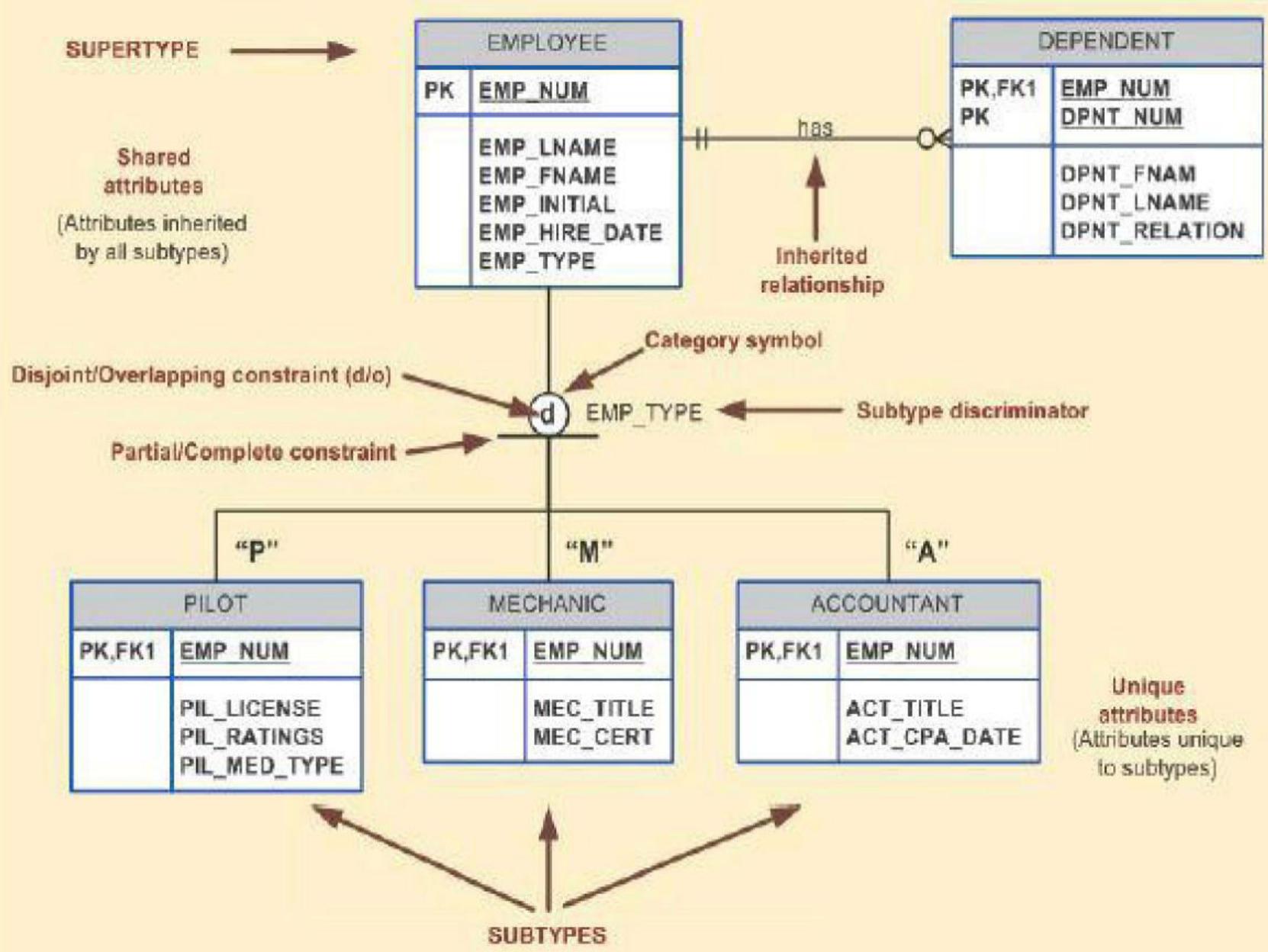
- reflects **1:1 relationship** between EMPLOYEE and its subtypes
- PILOT subtype occurrence is related to one instance of EMPLOYEE supertype
- MECHANIC subtype occurrence is related to one instance of EMPLOYEE supertype

# specialization hierarchy

- relationships depicted within specialization hierarchy are sometimes described in terms of “is-a” relationships
- pilot *is a* employee, mechanic *is a* employee, and accountant *is a* employee
- subtype can exist only within context of supertype

# specialization hierarchy

- support attribute inheritance
- define special supertype attribute known as subtype discriminator
- define disjoint/overlapping constraints and complete/partial constraints



# Inheritance

- enable entity subtype to inherit attributes and relationships of supertype
- pilots, mechanics, and accountants all inherit employee number, last name, first name, middle initial, and hire date from EMPLOYEE entity
- pilots have unique attributes; same is true for mechanics and accountants
- *all entity subtypes inherit their primary key attribute from their supertype - EMP\_NUM attribute is Primary Key for each of subtypes*

# Inheritance

- entity subtypes inherit all relationships in which supertype entity participates
- EMPLOYEE entity supertype participate in 1:M relationship with DEPENDENT entity
- through inheritance, all subtypes also participate in that relationship

# Subtype Discriminator

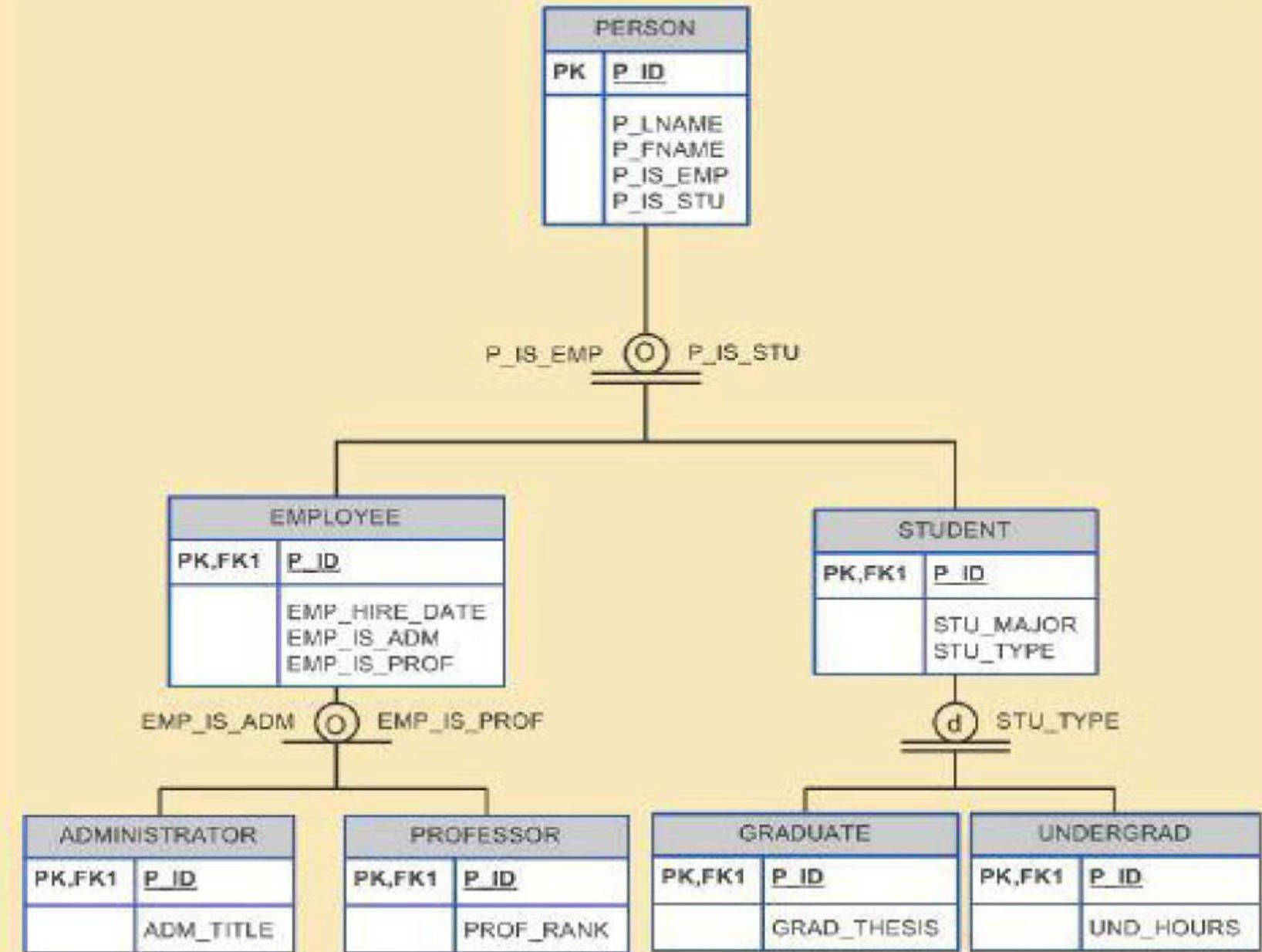
- attribute in supertype entity that determines to which subtype supertype occurrence is related
- common practice to show subtype discriminator and its value for each subtype in ER diagram
- supertype is related to PILOT subtype if EMP\_TYPE has value of “P”; if EMP\_TYPE value is “M,” supertype is related to MECHANIC subtype; if EMP\_TYPE value is “A” supertype is related to ACCOUNTANT subtype

# Disjoint subtypes, NonOverlapping subtypes

- business rules dictates that employee cannot belong to more than one subtype at a time; that is, employee cannot be pilot and mechanic at same time
- each entity instance of supertype can appear in only one of subtypes

# Overlapping subtypes

- if business rule specifies that employees can have multiple classifications, EMPLOYEE supertype may contain *overlapping* job classification subtypes
- subtypes that contain nonunique subsets of supertype entity set; that is, each entity instance of supertype may appear in more than one subtype
- for example, in university environment, person may be employee, student, or both; employee may be professor as well as administrator



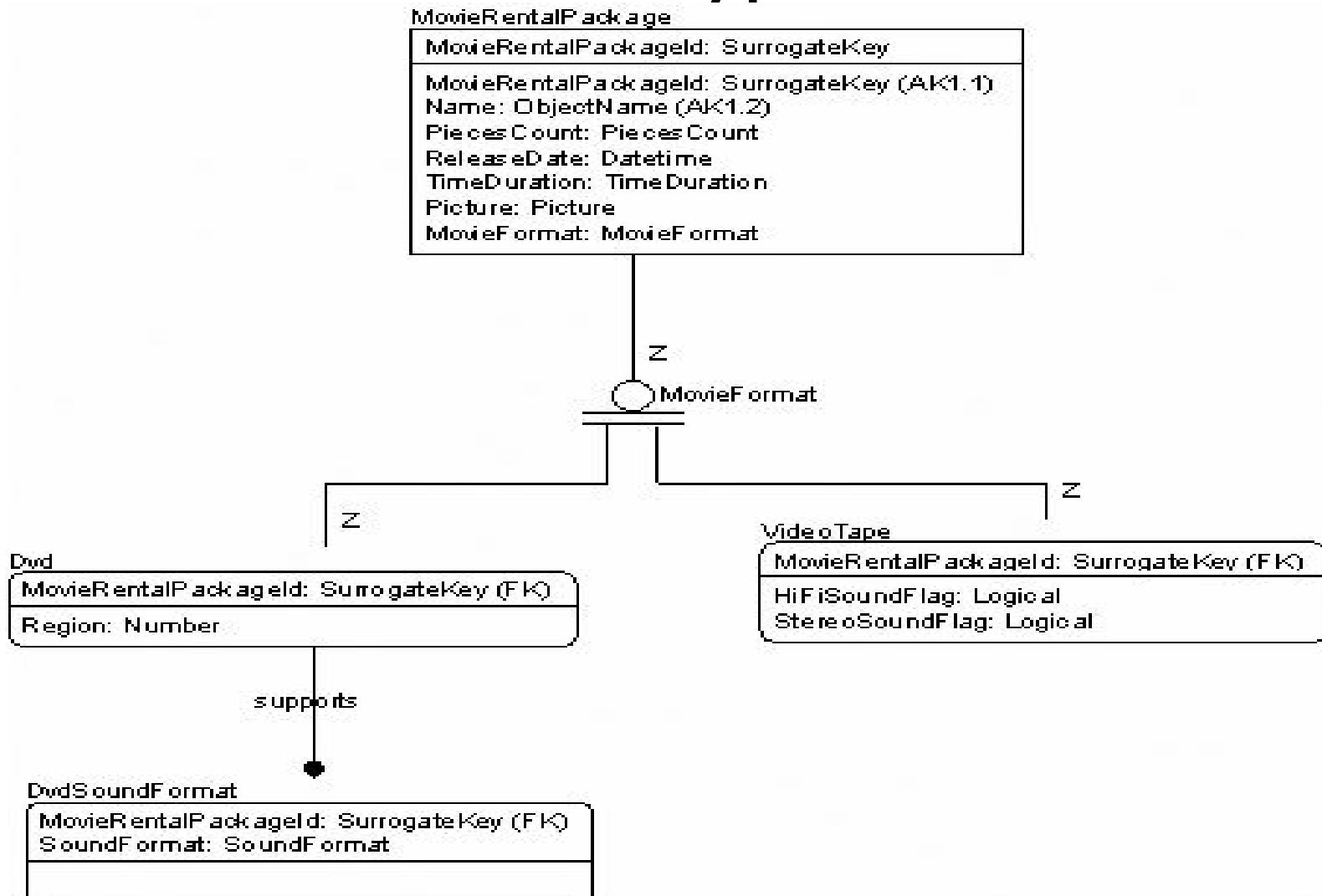
# Completeness constraint

- specifies whether each entity supertype occurrence must also be member of at least one subtype - can be partial or total
- partial completeness means that not every supertype occurrence is member of subtype; some supertype occurrences may not be members of any subtype
- total completeness means that every supertype occurrence must be member of at least one subtype

# Dealing with Subtypes

- rolling up subtypes
- leaving as separate tables
- rolling down generic types

# Subtypes



# Subtypes

- to create new media rental, you have to create rows in at least two tables
  - MovieRentalPackage and VideoTape or Dvd
- to see list of all items available for rental, write moderately complex query
  - joins MovieRentalPackage with VideoTape and union this with another join between MovieRentalPackage and Dvd

# Rolling up subtypes

## Movie Rental Package

MovieRentalPackageId: Surrogate Key

MovieId: Surrogate Key (FK) (AK1.1)

Name: ObjectName (AK1.2)

PiecesCount: Pieces Count

ReleaseDate: Datetime

TimeDuration: Time Duration

Picture: Picture

MovieFormat: Movie Format

DvdRegion: Number

supports

## Movie Rental Package Sound Format

SoundFormat: SoundFormat

MovieRentalPackageId: Surrogate Key (FK)

# Rolling up subtypes

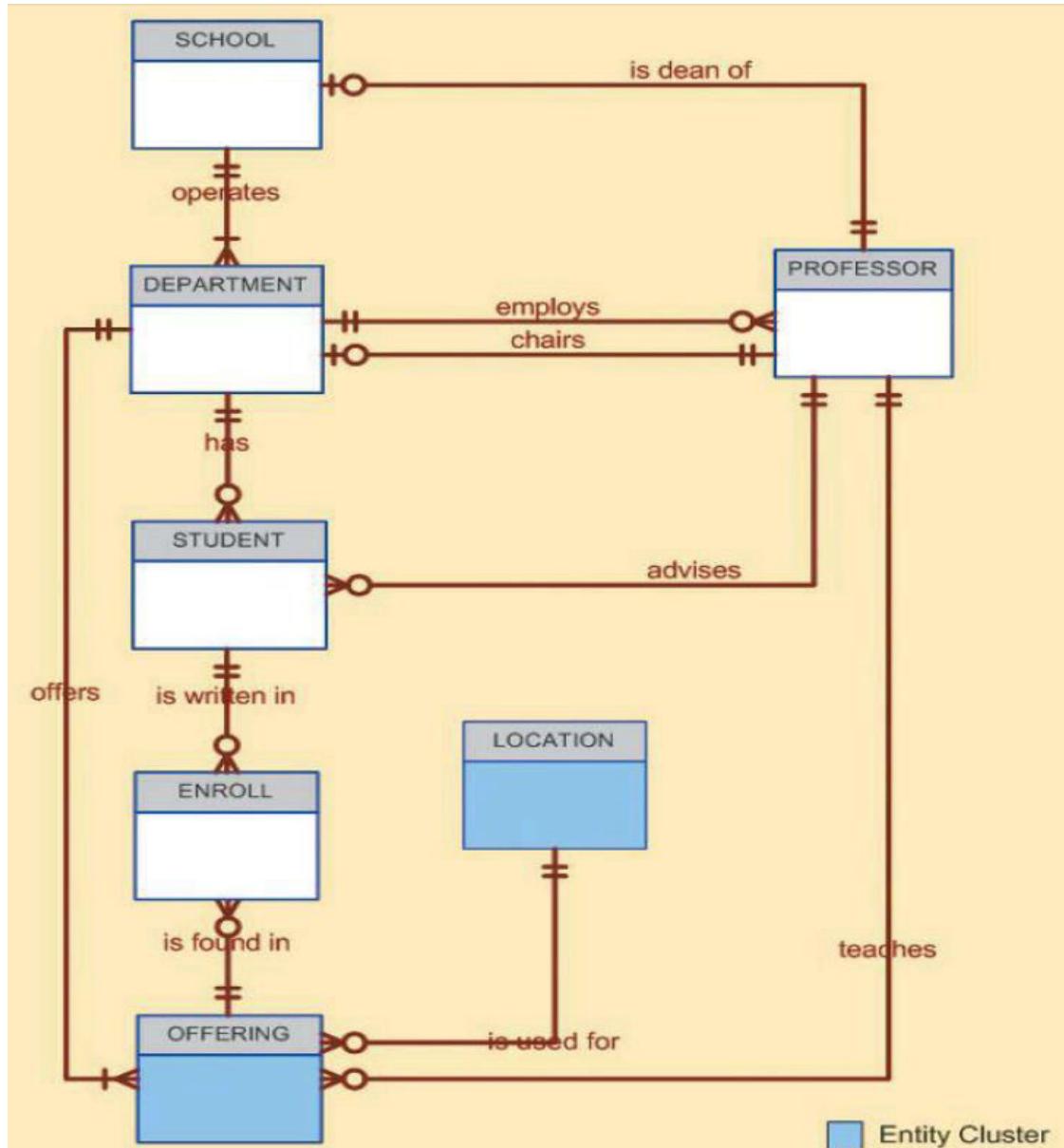
- when MediaFormat is 'VideoTape', DvdRegion value doesn't apply and must be set to NULL
- formats won't be applicable to 'VideoTape' but apply to 'Dvd'
- structure becomes less and less self-documenting, requiring more code to keep everything straight, requiring constraints

# Entity Clustering

- developing an ER diagram entails discovery of possibly hundreds of entity and their respective relationships that crowd diagram to point of making it unreadable and inefficient as communication tool
- can use entity clusters to minimize number of entities shown

# entity cluster

- virtual entity type used to represent multiple entities and relationships in ER diagram
- formed by combining multiple interrelated entities into single, abstract entity object
- considered “virtual” or “abstract” in sense that it is not actually entity in final ER diagram
  - temporary used to represent multiple entities and relationships, with purpose of simplifying and thus enhancing readability

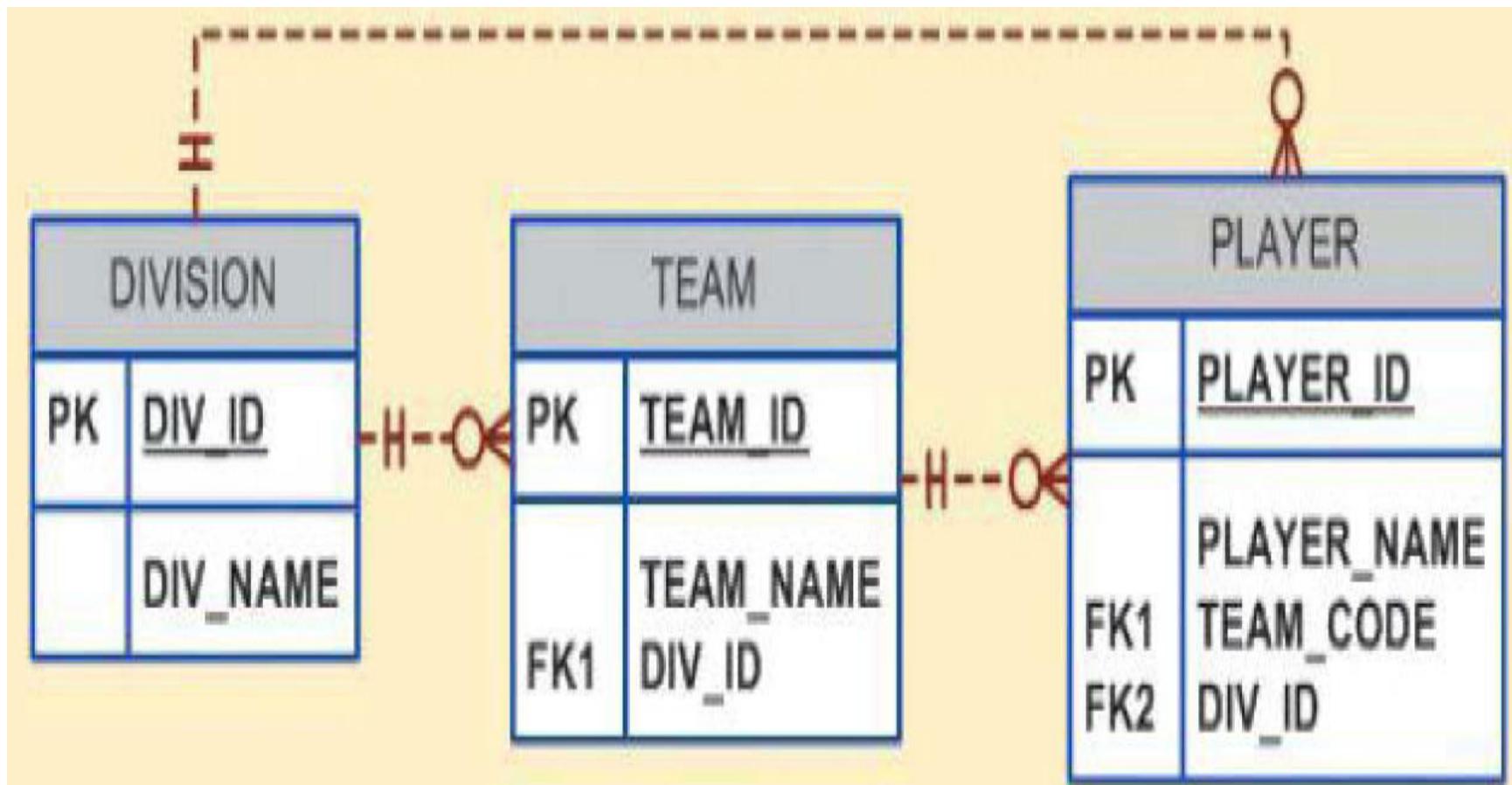


- illustrates use of entity clusters based on college example – ER diagram contains two entity clusters:
- OFFERING, groups COURSE and CLASS
- LOCATION, groups ROOM and BUILDING
- does not show attributes for entities

- due to miscommunication or incomplete understanding of business rules or processes, it is not uncommon to **misidentify relationships** among entities
- ER diagram may contain design trap - occurs when relationship is improperly or incompletely identified and is therefore represented in way that is not consistent with real world

# Redundant Relationships

- occur when there are multiple relationship paths between related entities
- main concern is that they remain consistent across model
- note transitive 1:M relationship between DIVISION and PLAYER through TEAM
- relationship that connects DIVISION and PLAYER is redundant, for all practical purposes



# Normalization

## **NORMAL STRUCTURES**

# Normalization

- is process for evaluating and correcting table structures to minimize data redundancies, reducing likelihood of data anomalies
- involves assigning attributes to tables based on concept of Functional Determination
- works through series of stages called Normal Forms
- first normal form (1NF), second normal form (2NF), and third normal form (3NF) - from structural point of view, 2NF is better than 1NF, and 3NF is better than 2NF
- for most purposes in business database design, 3NF is as high as you need to go in normalization process

# **IMPLEMENTING THE DESIGN**

- CREATE TABLE  
[<database>.] [<schema>.] <tablename>
- (
- <column specification>
- )

- <columnName> <datatype> [<NULL specification>] [IDENTITY [(seed,increment)]]
- --or
- <columnName> AS <computed definition>)

- CREATE TABLE Inventory.MovieFormat (
- MovieFormatId int NOT NULL
- *CONSTRAINT PKInventory\_MovieFormat  
PRIMARY KEY CLUSTERED,*
- Name varchar(20) NOT NULL
- )

- [CONSTRAINT constraintname] UNIQUE  
[CLUSTERED | NONCLUSTERED]
- ALTER TABLE Inventory.Genre
- ADD CONSTRAINT AKInventory\_Genre\_Name  
UNIQUE NONCLUSTERED (Name)

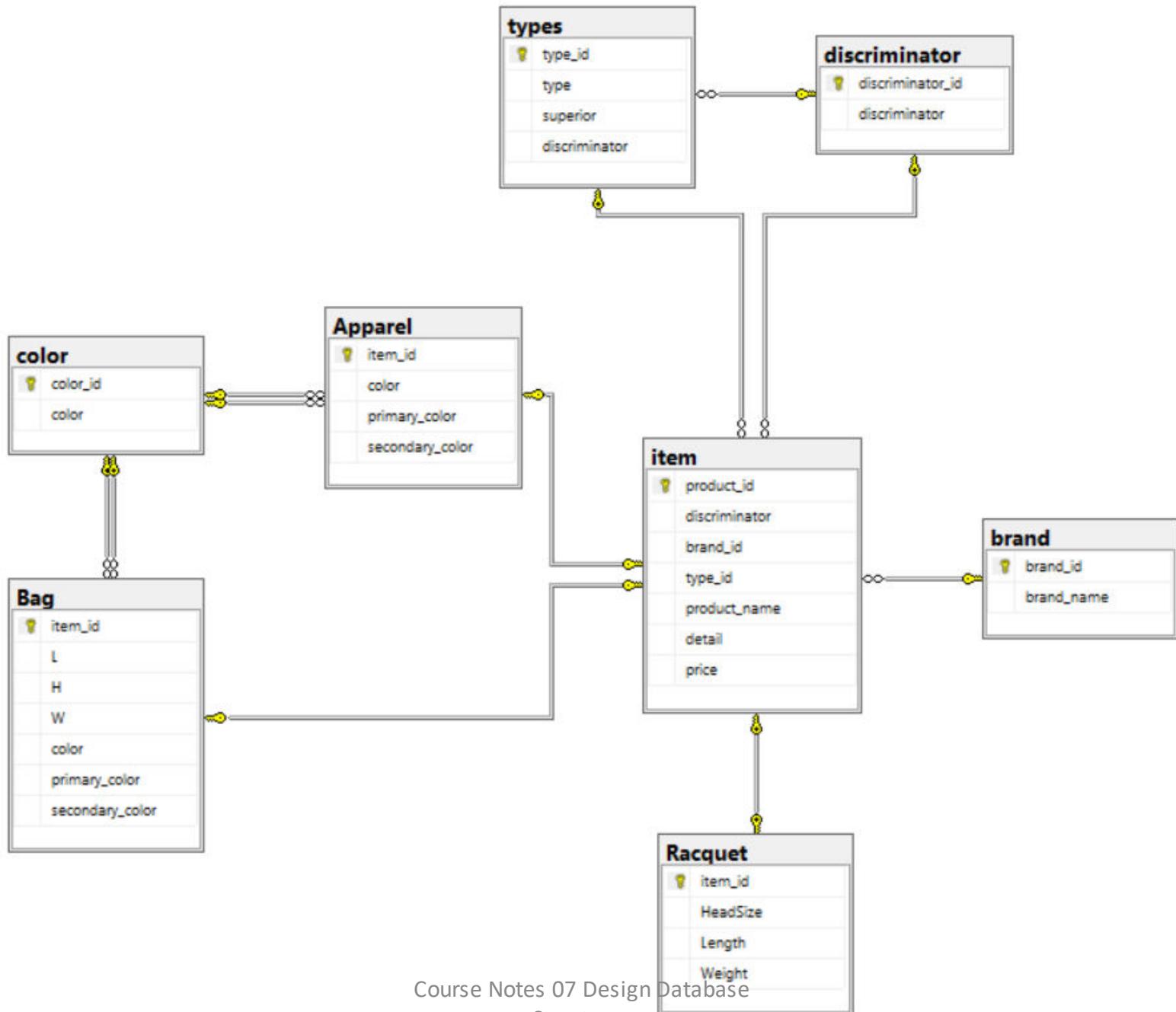
- ALTER TABLE <tableName>
  - ADD [ CONSTRAINT <DefaultName> ]
  - DEFAULT <constantExpression>
  - FOR <columnName>

- [CONSTRAINT <constraintName>]
- FOREIGN KEY REFERENCES <referenceTable> (<referenceColumns>)
- [ON DELETE <NO ACTION | CASCADE | SET NULL | SET DEFAULT> ]
- [ON UPDATE <NO ACTION | CASCADE | SET NULL | SET DEFAULT> ]

- **CASCADE**
  - ON DELETE, delete child rows; ON UPDATE, change child rows to match updated values
- **SET NULL**
  - ON DELETE or UPDATE of key values, set child row values to NULL, even if the value is updated to a valid value
- **SET DEFAULT**
  - ON DELETE or UPDATE, set child values to a default value, much like SET NULL, but useful for columns that cannot be set to NULL

# Tennis **WAREHOUSE**

- only presentation ... no orders



# General entity

- **item – product**
- product name, details, price

# Special entities

- **Racquet, Apparel, Bag, ...**
- specific attributes:
- Racquet:
  - Head size, Length, Weight, ...
- Apparel:
  - Color, ...
- Bag:
  - L, H, W, ...

# Relationships

- Brand – Product item
- general relationship – all have brand name
- Color – Apparel, Color – Bag
- specific relationship – only apparel and bag have colors

# color attribute

- racquet have no color
- apparel and bag have at most 3 color:
- marketing color
  - try to imagine the best sunset – it is the same color but green or blue
- primary & secondary color in relationship with color entity – used to filter data, in searching

# discriminator

- RAcquet
- APparel
- BAg
- ...

# Type Entity

- id
- type
- superior

# Type Entity

## type superior

- 3 Racquet Tennis Bags Bag
- 6 Racquet Tennis Bags Bag
- Club Tennis Bags (Non-Racquet) Bag
- Tennis Backpacks Bag
- Travel Bags with Wheels Bag

# Type Entity

## type superior

- Men's Apparel                      Apparel
- Women's Apparel                  Apparel
- Polos        Men's                  Apparel
- Crew & V-Neck Tops              Men's Apparel
- T-Shirts                              Men's Apparel
- Sleeveless Tops                    Men's Apparel
- Long Sleeve Tops                 Men's Apparel
- Shorts                                Men's Apparel

# Type Entity

## type superior

- Tanks Women's Apparel
- Polos Women's Apparel
- T-Shirts Women's Apparel
- Shorts Women's Apparel
- Dresses Women's Apparel
- Under Garments Women's Apparel
- Panties Women's Apparel
- Bras Women's Apparel
- ...

# View Types

- SELECT dbo.types.type\_id, dbo.types.type,  
types\_1.type AS superior
- FROM dbo.types INNER JOIN
  - dbo.types AS types\_1 ON
  - dbo.types.superior = types\_1.type\_id

# View product items

- ```
SELECT      dbo.item.product_id,
            dbo.brand.brand_name, dbo.types.type,
            dbo.item.product_name, dbo.item.detail,
            dbo.item.price,
            dbo.discriminator.discriminator, types_1.type
              AS superior
```
- ```
FROM dbo.discriminator INNER JOIN
```

# View product items

- ```
SELECT      ... FROM dbo.discriminator INNER JOIN
              – dbo.item ON dbo.discriminator.discriminator_id =
                           dbo.item.discriminator INNER JOIN
              – dbo.brand ON dbo.item.brand_id =
                           dbo.brand.brand_id INNER JOIN
              – dbo.types ON dbo.item.type_id = dbo.types.type_id
                           AND dbo.discriminator.discriminator_id =
                           dbo.types.discriminator LEFT OUTER JOIN
              – dbo.types AS types_1 ON dbo.types.superior =
                           types_1.type_id
```

brand_name type product_name price discriminator superior

1. Babolat Racquet PLAY Pure Aero 349.00 RA
NULL
2. Adidas Club Tennis Bags (Non-Racquet) Defender II
Duffel Bag Medium Black/White 31.99 BA Bag
3. Babolat 6 Racquet Tennis Bags Club Line Blue 39.95 BA
Bag
4. Nike Crew & V-Neck Tops Summer Rafa AeroReact
Challenger 100.00 AP Men's Apparel
5. Wilson Tanks Spring Core Classic 29.99 AP
Women's Apparel
6. Nike Bras Summer Streak Swoosh 35.00 AP
Women's Apparel

View RAcquet

- `SELECT dbo.item.product_id,
 dbo.item.discriminator, dbo.brand.brand_name,
 dbo.item.product_name, dbo.item.price,
 dbo.Racquet.HeadSize, dbo.Racquet.Length,
 dbo.Racquet.Weight`
- `FROM dbo.item INNER JOIN
 – dbo.Racquet ON dbo.item.product_id =
 dbo.Racquet.item_id INNER JOIN
 – dbo.brand ON dbo.item.brand_id =
 dbo.brand.brand_id`

View RAcquet

- product_id discriminator brand_name
product_name price HeadSize Length
Weight
- 5 RA Babolat PLAY Pure Aero 349.00
100 27 11.3

UnSolved issues

- product item – size
- specific relationships:
- racquet – size 1, 2, 3, 4, 5
- apparel – size S, M, L, XL, XXL, ...
- bag does not have different sizes

UnSolved issues

- orders

UnSolved issues

- collections for apparel

UnSolved issues

- *apparel: same product item comes in different color variants*

UnSolved issues

- types and discriminator are little a bit redundant ...
- but still consider that we truly need type entity and discriminator it is useful in relation of product items with particular tables entities Racquet, Apparel, Bag, ...

UnSolved issues

- Unique indexes ...

Solved issues

- CASCADE DELETE from product items to Racquet, Apparel, Bag

Populate Database

- I use MS Access
- create ODBC source to already created MS SQL SERVER DataBase
- link tables
- use lookup
- consult Access materials

DataBase Design

DataBase Design

Design DataBase Structures

- DataBase Design focuses on how database structure will be used to store and manage end-user data
- data modeling - first step in designing database: refers to process of creating specific data model for determined problem domain
- *database model* used to refer to implementation of *data model* in specific database system

Design DataBase Structures

- data model is representation, usually graphical, of more complex real-world data structures; is abstraction of complex real-world object or event
- model's main function is to help you understand real-world environment
- data model represents data structures and their characteristics, relations, constraints, transformations, ... with purpose of supporting specific problem domain

Importance of Data Models

- facilitate interaction among designer, applications programmer, and end user
- improved understanding of organization
- communication tool
- applications created to manage data and to help transform data into information
 - but data are viewed in different ways by different people
- to create good database first create good, appropriate data model

Data Model – Basic Building Blocks

- entities
- attributes
- relationships
- constraints
- *how to identify entities, attributes, relationships, and constraints? first step identify business rules for problem domain you are modeling*

Business Rule

- brief, precise, and unambiguous description of policy, procedure, or principle within specific organization
- apply to *any* organization - business, government unit, religious group, research laboratory - that stores and uses data to generate information
- business rules describe main and distinguishing characteristics of data *as viewed by organization*

Business Rule

- set stage for proper identification of entities, attributes, relationships, and constraints
- names are used to identify objects
- noun in business rule will translate into entity in model
- verb (active or passive) that associates nouns will translate into relationship among entities
- should consider that relationships are bidirectional
- *naming convention*

Functional Dependence

- state in which knowing value of one attribute makes it possible to determine value of another
- in database environment is not normally based on formula but on relationships among attributes
- means that value of one or more attributes determines value of one or more other attributes
- standard notation for representing relationship between STU_NUM and STU_LNAME is:
- $\text{STU_NUM} \rightarrow \text{STU_LNAME}$
- Student's No. ID and Student's Last Name

Functional Dependence

- attribute whose value determines another called *determinant*
- attribute whose value is determined by others called *dependent*
- STUDENT table $\text{STU_NUM} \rightarrow$
- $(\text{STU_LNAME}, \text{STU_FNAME}, \text{STU_INIT}, \text{STU_DOB}, \text{STU_HRS}, \text{STU_CLASS}, \text{STU_GPA})$

Student Table

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT	STU_DOB	STU_HRS	STU_CLASS	STU_GPA	STU_TRANSFER	DEPT_CODE	STU_PHONE	PROF_NUM
321452	Bowser	William	C	12-Feb-1975	42	Sr	2.84	No	BIOL	2134	205
324257	Smithson	Anne	K	15-Nov-1981	81	Jr	3.27	Yes	CIS	2256	222
324258	Brewer	Juliette		23-Aug-1969	36	Sr	2.26	Yes	ACCT	2256	228
324269	Oblonski	Walter	H	16-Sep-1976	66	Jr	3.09	No	CIS	2114	222
324273	Smith	John	D	30-Dec-1958	102	Sr	2.11	Yes	ENGL	2231	199
324274	Katinga	Raphael	P	21-Oct-1979	114	Sr	3.15	No	ACCT	2267	228
324291	Robertson	Gerald	T	08-Apr-1973	120	Sr	3.87	No	EDU	2267	311
324299	Smith	John	B	30-Nov-1986	15	Fr	2.92	No	ACCT	2315	230

STU_NUM	= Student number
STU_LNAME	= Student last name
STU_FNAME	= Student first name
STU_INIT	= Student middle initial
STU_DOB	= Student date of birth
STU_HRS	= Credit hours earned
STU_CLASS	= Student classification
STU_GPA	= Grade point average
STU_TRANSFER	= Student transferred from another institution
DEPT_CODE	= Department code
STU_PHONE	= 4-digit campus phone extension
PROF_NUM	= Number of the professor who is the student's advisor

Normalization

NORMAL STRUCTURES

Normalization

- is process for evaluating and correcting table structures to minimize data redundancies, reducing likelihood of data anomalies
- involves assigning attributes to tables based on concept of Functional Determination
- works through series of stages called Normal Forms
- first normal form (1NF), second normal form (2NF), and third normal form (3NF) - from structural point of view, 2NF is better than 1NF, and 3NF is better than 2NF
- for most purposes in business database design, 3NF is as high as you need to go in normalization process

- higher Normal Form more relational join operations you need to produce specified output; more resources are required by database system to respond to end-user queries
- in normalization terminology, any attribute that is at least part of key is known as *prime attribute* instead of key attribute and *nonprime attribute*, or nonkey attribute, is not part of any candidate key

Normalization of Database Tables

- Entity Relationship (ER) Modeling, yields good table structures
- it is possible to create poor table structures even in good database design
- how do you recognize a poor table structure, and how do you produce a good table?
- The answer to both questions involves normalization
- Normalization is a process for evaluating and correcting table structures to minimize data redundancies, thereby reducing the likelihood of data anomalies

Normalization

- works through a series of stages called normal forms
- first normal form (1NF), second normal form (2NF), and third normal form (3NF)
- from structural point of view, 2NF is better than 1NF, and 3NF is better than 2NF
- for most purposes in business database design, 3NF is as high as you need

Denormalization

- produces a lower normal form; that is, a 3NF will be converted to a 2NF
- price you pay for increased performance through denormalization is greater data redundancy

- any attribute that is at least part of a key is known as a prime attribute
- nonprime attribute is not part of any candidate key
- concept of keys is central to the discussion of normalization

- consider the simplified database activities of a construction company that manages several building projects
- each project has its own project number, name, employees assigned to it, ...
- each employee has an employee number, name, and job classification, such as engineer, computer technician, ...
- company charges its clients by billing hours spent on each contract
- hourly billing rate is dependent on employee's position

Report generated

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.8
		101	John G. News	Database Designer	105.00	19.4
		105	Alice K. Johnson *	Database Designer	105.00	35.7
		106	William Smithfield	Programmer	35.75	12.6
		102	David H. Senior	Systems Analyst	96.75	23.8
18	Amber Wave	114	Annelise Jones	Applications Designer	48.10	24.6
		118	James J. Frommer	General Support	18.36	45.3
		104	Anne K. Ramoras *	Systems Analyst	96.75	32.4
		112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
		104	Anne K. Ramoras	Systems Analyst	96.75	48.4
		113	Delbert K. Joenbrood *	Applications Designer	48.10	23.6
		111	Geoff B. Wabash	Clerical Support	26.87	22.0
		106	William Smithfield	Programmer	35.75	12.8
25	Starflight	107	Maria D. Alonzo	Programmer	35.75	24.6
		115	Travis B. Bawangi	Systems Analyst	96.75	45.8
		101	John G. News *	Database Designer	105.00	56.3
		114	Annelise Jones	Applications Designer	48.10	33.1
		108	Ralph B. Washington	Systems Analyst	96.75	23.6
		118	James J. Frommer	General Support	18.36	30.5
		112	Darlene M. Smithson	DSS Analyst	45.95	41.4

- an employee can be assigned to more than one project
- each project includes only a single occurrence of any one employee

- project number (PROJ_NUM) is apparently intended to be a primary key or at least a part of a PK, but it contains nulls
 - PROJ_NUM + EMP_NUM will define each row
- table entries invite data inconsistencies
 - for example, JOB_CLASS value “Elect. Engineer” might be entered as “Elect.Eng.” in some cases, “El. Eng.” in others, and “EE” in still others

data redundancies yield the following anomalies:

- Update anomalies
- Insertion anomalies
- Deletion anomalies

Update anomalies

- Modifying the JOB_CLASS for employee number 105 requires (potentially) many alterations, one for each EMP_NUM = 105.

Insertion anomalies

- Just to complete a row definition, a new employee must be assigned to a project
- If the employee is not yet assigned, a phantom project must be created to complete the employee data entry

Deletion anomalies

- Suppose that only one employee is associated with a given project
- If that employee leaves the company and the employee data are deleted, the project information will also be deleted
- To prevent the loss of the project information, a fictitious employee must be created just to save the project information

well-formed relations

- objective of normalization is to ensure that each table conforms to the concept of *well-formed relations*, that is, tables that have the following characteristics:

- Each table represents a single subject
 - for example, course table will contain only data that directly pertain to courses
 - similarly, student table will contain only student data
- No data item will be unnecessarily stored in more than one table (tables have minimum controlled redundancy)
 - the reason for this requirement is to ensure that the data are updated in only one place

- All nonprime attributes in a table are dependent on the primary key, the entire primary key and nothing but the primary key
 - reason for this requirement is to ensure that the data are uniquely identifiable by primary key value
- each table is void of insertion, update, or deletion anomalies
 - to ensure integrity and consistency of data

Functional dependence

- attribute B is fully functionally dependent on attribute A if each value of A determines one and only one value of B
- Example: PROJ_NUM → PROJ_NAME
 - read as “PROJ_NUM functionally determines PROJ_NAME”
 - attribute PROJ_NUM is known as “determinant” attribute, and attribute PROJ_NAME is known as “dependent” attribute

Functional dependence

- Attribute A determines attribute B (that is, B is functionally dependent on A) if all of the rows in table that agree in value for attribute A also agree in value for attribute B

Step 1: Eliminate the Repeating Groups

- Start by presenting the data in a tabular format, where each cell has a single value and there are no repeating groups.
- To eliminate the repeating groups, eliminate the nulls by making sure that each repeating group attribute contains an appropriate data value.

Table in first normal form 1NF

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.8
15	Evergreen	101	John G. News	Database Designer	105.00	19.4
15	Evergreen	105	Alice K. Johnson *	Database Designer	105.00	35.7
15	Evergreen	106	William Smithfield	Programmer	35.75	12.6
15	Evergreen	102	David H. Senior	Systems Analyst	96.75	23.8
18	Amber Wave	114	Annelise Jones	Applications Designer	48.10	24.6
18	Amber Wave	118	James J. Frommer	General Support	18.36	45.3
18	Amber Wave	104	Anne K. Ramoras *	Systems Analyst	96.75	32.4
18	Amber Wave	112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
22	Rolling Tide	104	Anne K. Ramoras	Systems Analyst	96.75	48.4
22	Rolling Tide	113	Delbert K. Joenbrood *	Applications Designer	48.10	23.6
22	Rolling Tide	111	Geoff B. Wabash	Clerical Support	26.87	22.0
22	Rolling Tide	106	William Smithfield	Programmer	35.75	12.8
25	Starflight	107	Maria D. Alonzo	Programmer	35.75	24.6
25	Starflight	115	Travis B. Bawangi	Systems Analyst	96.75	45.8
25	Starflight	101	John G. News *	Database Designer	105.00	56.3
25	Starflight	114	Annelise Jones	Applications Designer	48.10	33.1
25	Starflight	108	Ralph B. Washington	Systems Analyst	96.75	23.6
25	Starflight	118	James J. Frommer	General Support	18.36	30.5
25	Starflight	112	Darlene M. Smithson	DSS Analyst	45.95	41.4

first normal form (1NF)

- describes tabular format in which:
- all of key attributes are defined
- there are no repeating groups in the table
- each row/column intersection contains only atomic values, one and only one value, not set of values
- all attributes are dependent on primary key
- all relational tables satisfy 1NF requirements

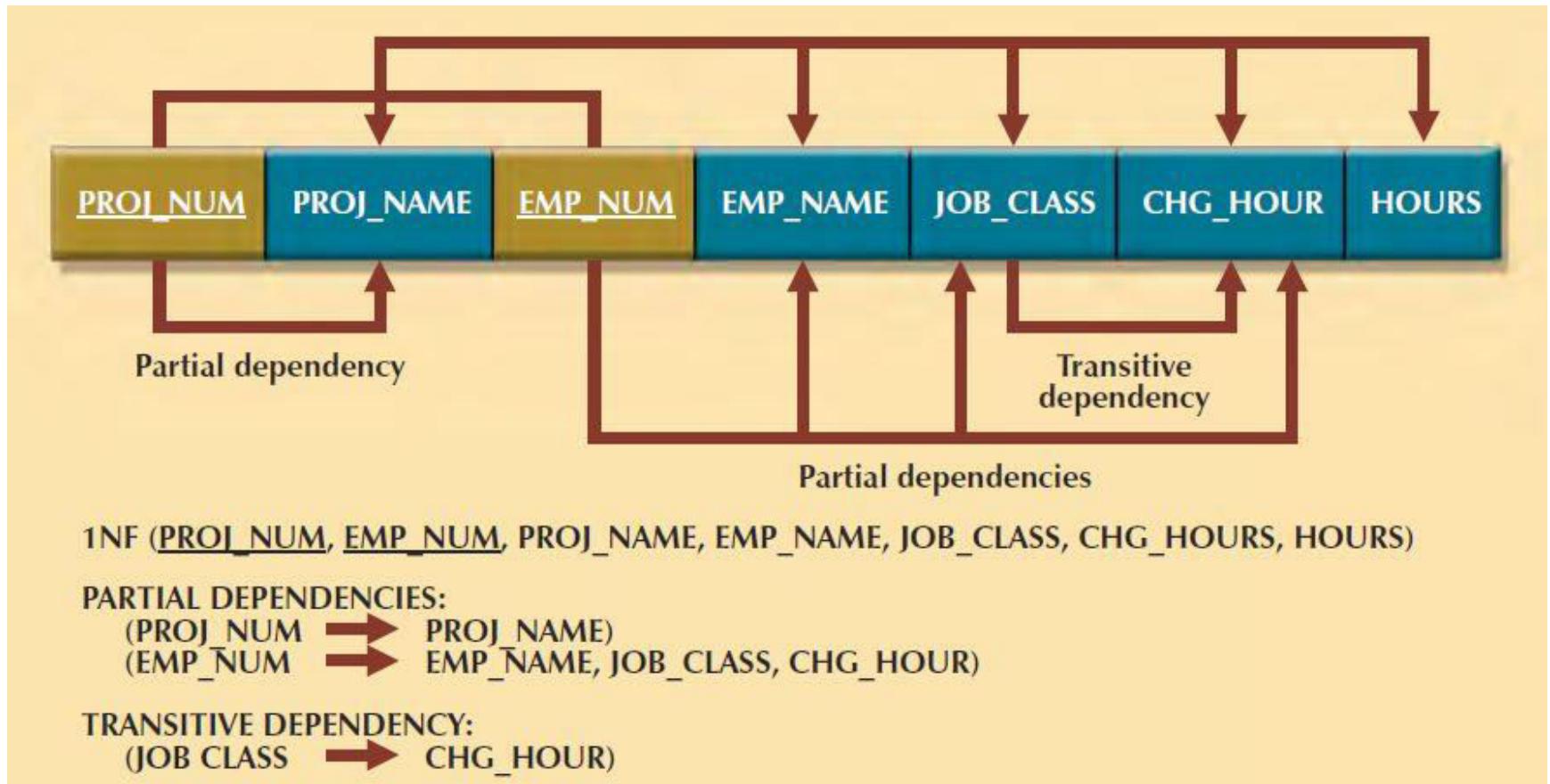
Step 2: Identify the Primary Key

- that will uniquely identify any attribute value must be composed of a combination of PROJ_NUM and EMP_NUM
- identification of PK in Step 2 means that you have already identified following dependency:
- PROJ_NUM, EMP_NUM → PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOUR, HOURS

Step 3: Identify All Dependencies

- PROJ_NUM → PROJ_NAME
- EMP_NUM → EMP_NAME, JOB_CLASS,
CHG_HOUR
- JOB_CLASS → CHG_HOUR

Dependency Diagram



Dependency Diagram

1. primary key attributes are bold, underlined, and shaded in different color.
2. arrows above attributes indicate all desirable dependencies, that is, dependencies that are based on PK; entity's attributes are dependent on *combination of PROJ_NUM and EMP_NUM*
3. arrows below indicate less desirable dependencies; Two types of such dependencies exist:

1. *Partial dependencies* - you need to know only PROJ_NUM to determine the PROJ_NAME; that is, PROJ_NAME is dependent on only part of PK; and you need to know only EMP_NUM to find EMP_NAME, JOB_CLASS, and CHG_HOUR. Dependency based on only a part of a composite primary key is a partial dependency
2. Transitive dependencies - CHG_HOUR is dependent on JOB_CLASS; neither CHG_HOUR nor JOB_CLASS is prime attribute (at least part of key) condition is transitive dependency. Transitive dependency is a dependency of one nonprime attribute on another. Transitive dependencies still yield data anomalies

Data Redundancies ... Anomalies

- occur because row entry requires duplication of data
- if Alice K. Johnson submits her work log, user would have to make multiple entries during course of a day
 - for each entry, EMP_NAME, JOB_CLASS, and CHG_HOUR must be entered each time, even though attribute values are identical for each row entered
- duplicate effort inefficient, helps create anomalies: nothing prevents from typing slightly different versions of employee name, position, hourly pay
- data anomalies violate relational database's integrity and consistency rules

Conversion to Second Normal Form

- Converting to 2NF is done only when the 1NF has a composite primary key.
- If the 1NF has a single-attribute primary key, then the table is automatically in 2NF.

Step 1: Make New Tables to Eliminate Partial Dependencies

- for each component of primary key that acts as determinant in a partial dependency, create a new table with copy of that component as primary key
- while these components are placed in new tables, it is important that they also remain in original table as well - they will be foreign keys for the relationships that are needed to relate these new tables to original table

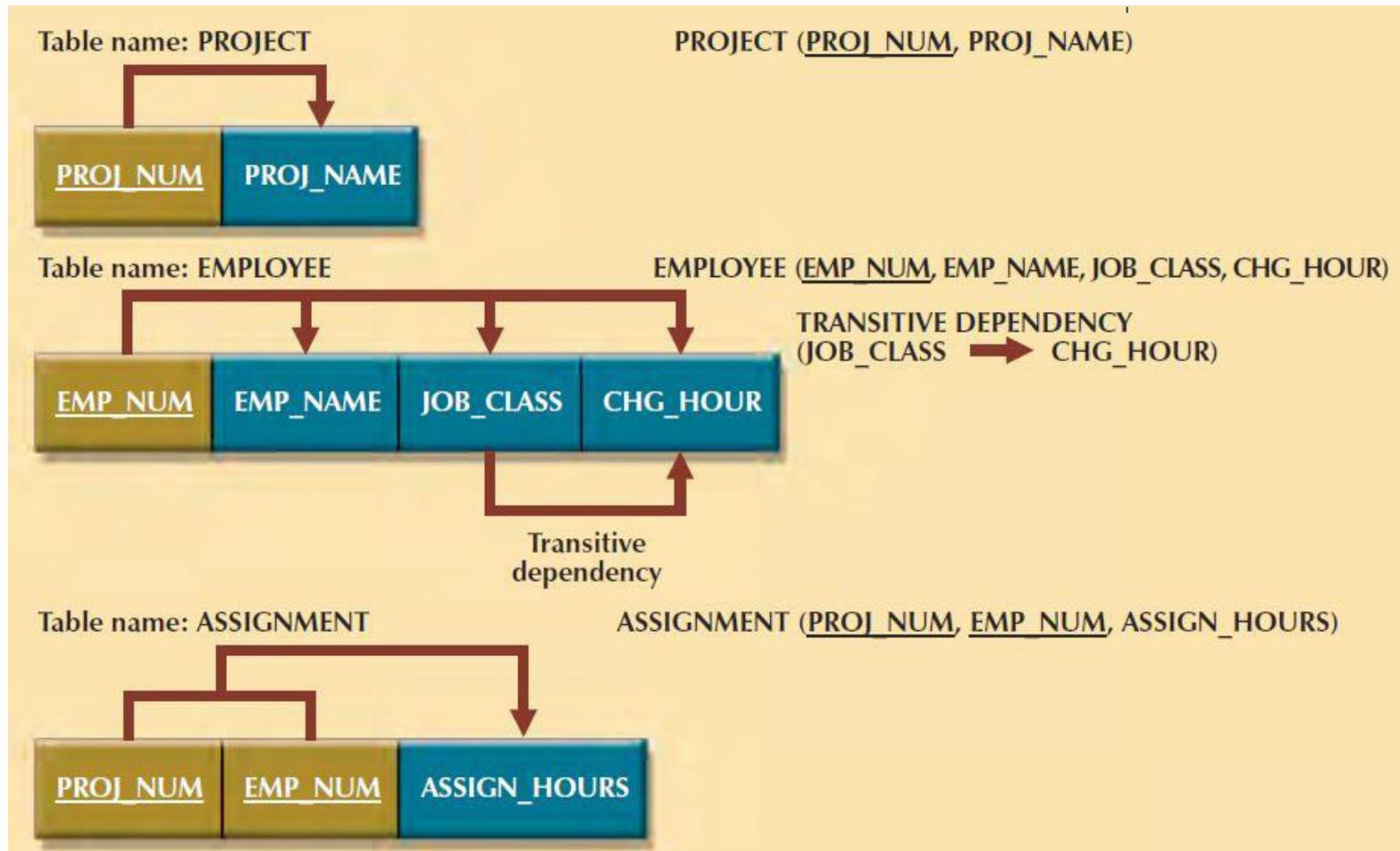
- PROJ_NUM
- EMP_NUM
- PROJ_NUM EMP_NUM
- each component will become key in new table
- original table is now divided into three tables
(PROJECT, EMPLOYEE, and ASSIGNMENT)

Step 2: Reassign Corresponding Dependent Attributes

- dependencies for original key components are found by examining arrows below dependency diagram
- attributes that are dependent in a partial dependency are removed from original table and placed in new table with its determinant
- any attributes that are not dependent in partial dependency will remain in original table

- PROJECT (PROJ_NUM, PROJ_NAME)
- EMPLOYEE (EMP_NUM, EMP_NAME,
JOB_CLASS, CHG_HOUR)
- ASSIGNMENT (PROJ_NUM, EMP_NUM,
ASSIGN_HOURS)

Second normal form (2NF)



second normal form (2NF)

- table is in second normal form (2NF) when:
- it is in 1NF
- and
- it includes no partial dependencies; that is, no attribute is dependent on only portion of primary key

- Because number of hours spent on each project by each employee is dependent on both PROJ_NUM and EMP_NUM in ASSIGNMENT table, you leave those hours in ASSIGNMENT table
- ASSIGNMENT table contains composite primary key composed of attributes PROJ_NUM and EMP_NUM
- primary key/foreign key relationships have been created

- most of the anomalies discussed earlier have been eliminated
- for example, if you now want to add, change, or delete a PROJECT record, you need to go only to PROJECT table and make the change to only one row
- transitive dependency can generate anomalies
- for example, if charge per hour changes for job classification held by many employees, that change must be made for each of those employees
- if you forget to update some of employee records that are affected by charge per hour change, different employees with same job description will generate different hourly charges

Conversion to Third Normal Form

Step 1: Make New Tables to Eliminate Transitive Dependencies

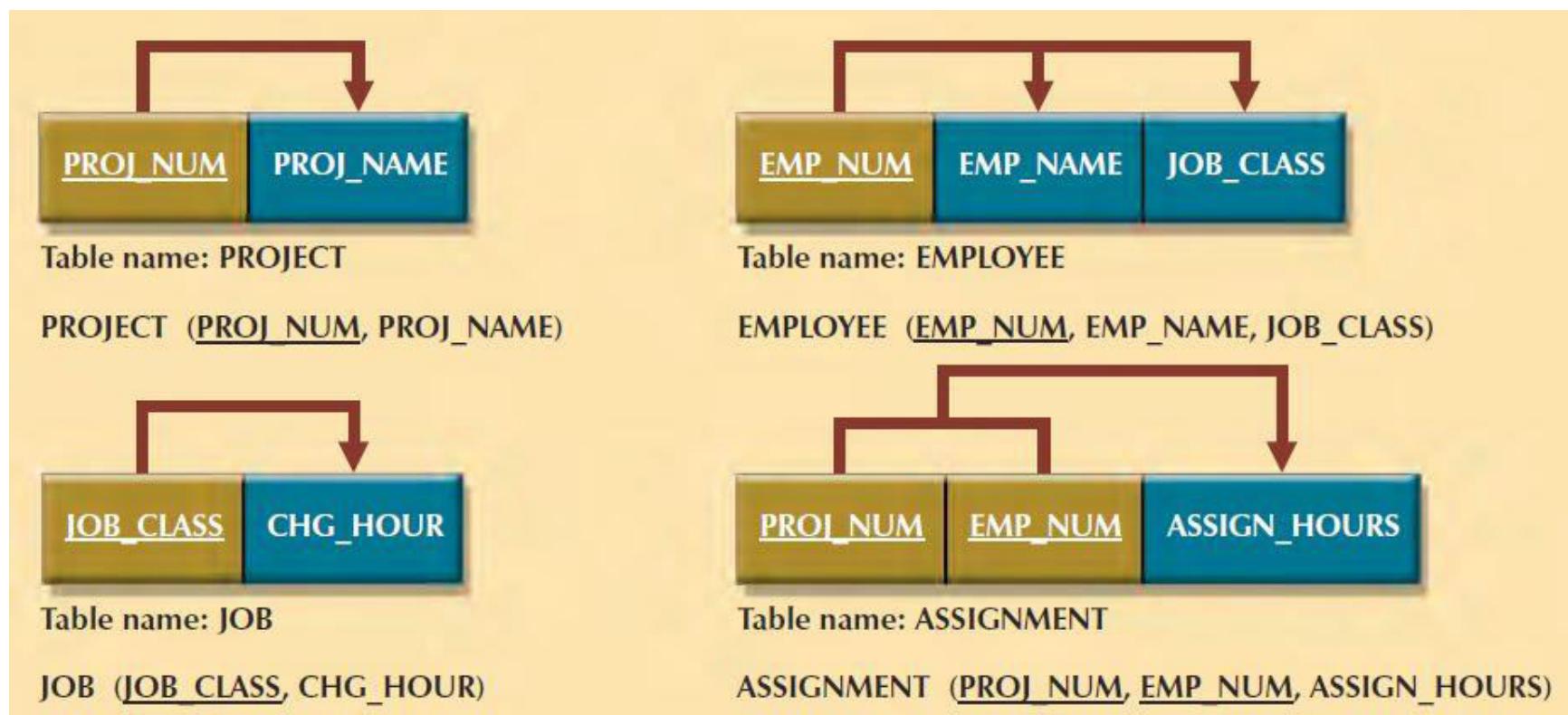
- for every transitive dependency, write a copy of its determinant as primary key for new table
- determinant is any attribute whose value determines other values within a row
- if you have three different transitive dependencies, you will have three different determinants
- determinant remain in original table to serve as foreign key

- determinant for this transitive dependency as:
- JOB_CLASS

Step 2: Reassign Corresponding Dependent Attributes

- identify the attributes that are dependent on each determinant identified
- place dependent attributes in new tables with their determinants and remove them from their original tables

Third normal form (3NF)



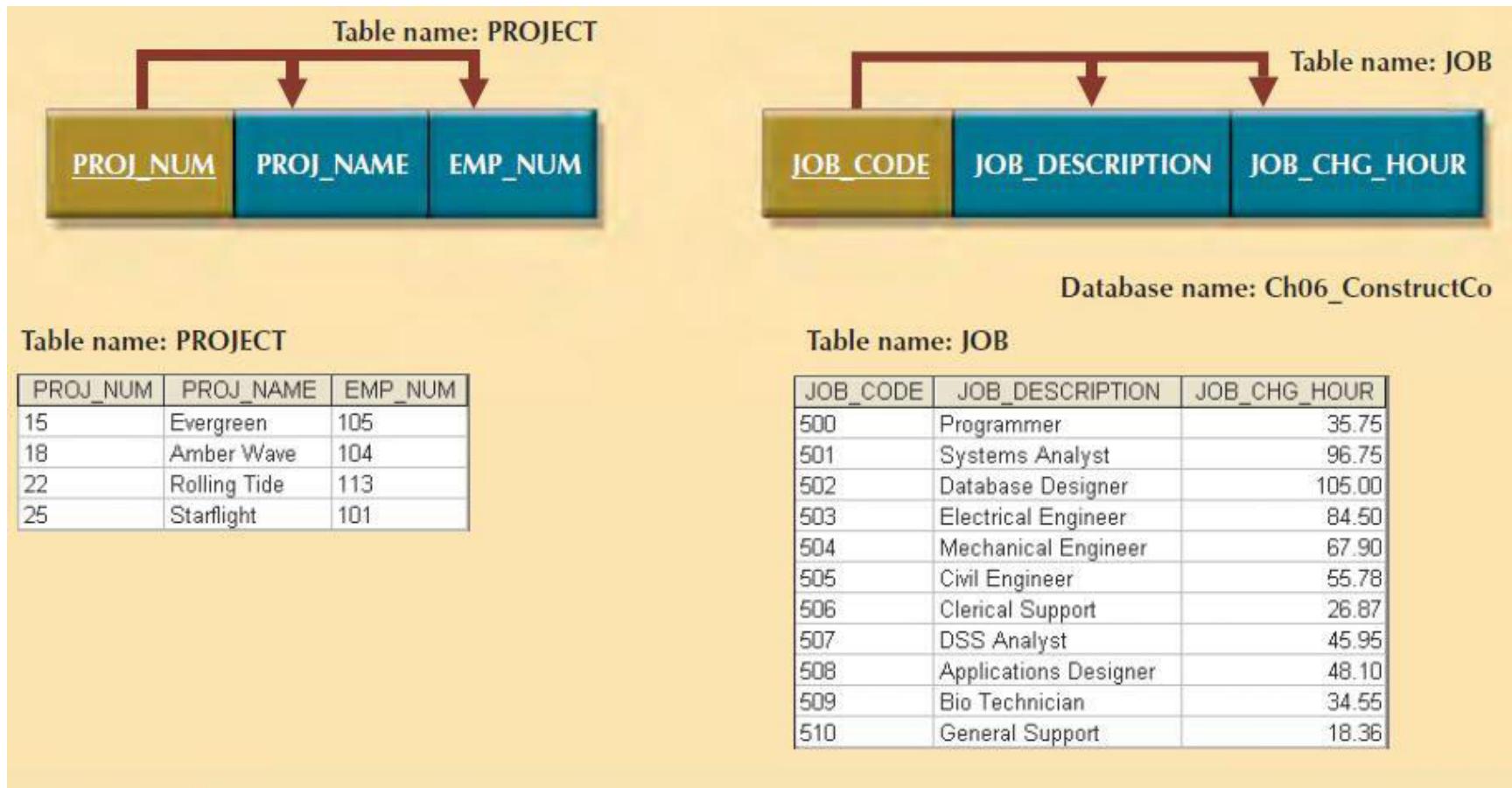
third normal form (3NF)

- table is in third normal form (3NF) when:
- it is in 2NF
- and
- it contains no transitive dependencies

- PROJECT (PROJ_NUM, PROJ_NAME)
- EMPLOYEE (EMP_NUM, EMP_NAME, JOB_CLASS)
- JOB (JOB_CLASS, CHG_HOUR)
- ASSIGNMENT (PROJ_NUM, EMP_NUM, ASSIGN_HOURS)

- technique is the same, ...
- it is imperative that 2NF be achieved before moving on to 3NF;
- be certain to resolve partial dependencies before resolving transitive dependencies

Improving Design



Completed Database

Table name: ASSIGNMENT

ASSIGN_NUM	ASSIGN_DATE	PROJ_NUM	EMP_NUM	ASSIGN_HOURS	ASSIGN_CHG_HOUR	ASSIGN_CHARGE
1001	04-Mar-10	15	103	2.6	84.50	219.70
1002	04-Mar-10	18	118	1.4	18.36	25.70
1003	05-Mar-10	15	101	3.6	105.00	378.00
1004	05-Mar-10	22	113	2.5	48.10	120.25
1005	05-Mar-10	15	103	1.9	84.50	160.55
1006	05-Mar-10	25	115	4.2	96.75	406.35
1007	05-Mar-10	22	105	5.2	105.00	546.00
1008	05-Mar-10	25	101	1.7	105.00	178.50
1009	05-Mar-10	15	105	2.0	105.00	210.00
1010	06-Mar-10	15	102	3.8	96.75	367.65
1011	06-Mar-10	22	104	2.6	96.75	251.55
1012	06-Mar-10	15	101	2.3	105.00	241.50
1013	06-Mar-10	25	114	1.8	48.10	86.58
1014	06-Mar-10	22	111	4.0	26.87	107.48
1015	06-Mar-10	25	114	3.4	48.10	163.54
1016	06-Mar-10	18	112	1.2	45.95	55.14
1017	06-Mar-10	18	118	2.0	18.36	36.72
1018	06-Mar-10	18	104	2.6	96.75	251.55
1019	06-Mar-10	15	103	3.0	84.50	253.50
1020	07-Mar-10	22	105	2.7	105.00	283.50
1021	08-Mar-10	25	108	4.2	96.75	406.35
1022	07-Mar-10	25	114	5.8	48.10	278.98
1023	07-Mar-10	22	106	2.4	35.75	85.80

Table name: EMPLOYEE



Table name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE
101	News	John	G	08-Nov-00	502
102	Senior	David	H	12-Jul-89	501
103	Arbough	June	E	01-Dec-97	503
104	Ramoras	Anne	K	15-Nov-88	501
105	Johnson	Alice	K	01-Feb-94	502
106	Smithfield	William		22-Jun-05	500
107	Alonzo	Maria	D	10-Oct-94	500
108	Washington	Ralph	B	22-Aug-89	501
109	Smith	Larry	W	18-Jul-99	501
110	Olenko	Gerald	A	11-Dec-96	505
111	Wabash	Geoff	B	04-Apr-89	506
112	Smithson	Darlene	M	23-Oct-95	507
113	Joenbrood	Delbert	K	15-Nov-94	508
114	Jones	Annelise		20-Aug-91	508
115	Bawangi	Travis	B	25-Jan-90	501
116	Pratt	Gerald	L	05-Mar-95	510
117	Williamson	Angie	H	19-Jun-94	509
118	Frommer	James	J	04-Jan-06	510

Higher Level Normal Forms

- Tables in 3NF will perform suitably in business transactional databases.
- However, there are occasions when higher normal forms are useful
- special case of 3NF, known as Boyce-Codd normal form(BCNF)
- fourth normal form (4NF)

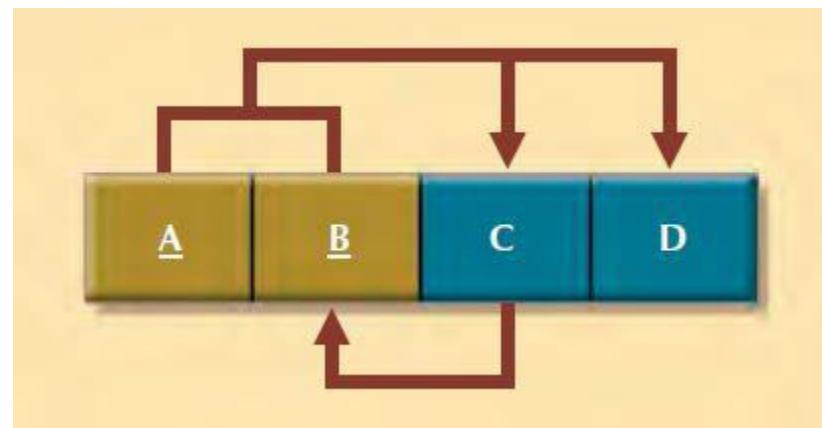
Boyce-Codd normal form (BCNF)

- when every determinant in table is candidate key
- when table contains only one candidate key, the 3NF and the BCNF are equivalent

Boyce-Codd normal form (BCNF)

- table is in 3NF when it is in 2NF and there are no transitive dependencies
- nonkey attribute is determinant of key attribute
- condition does not violate 3NF, yet it fails to meet the BCNF requirements because BCNF requires that every determinant in the table be candidate key

table that is in 3NF but not in BCNF



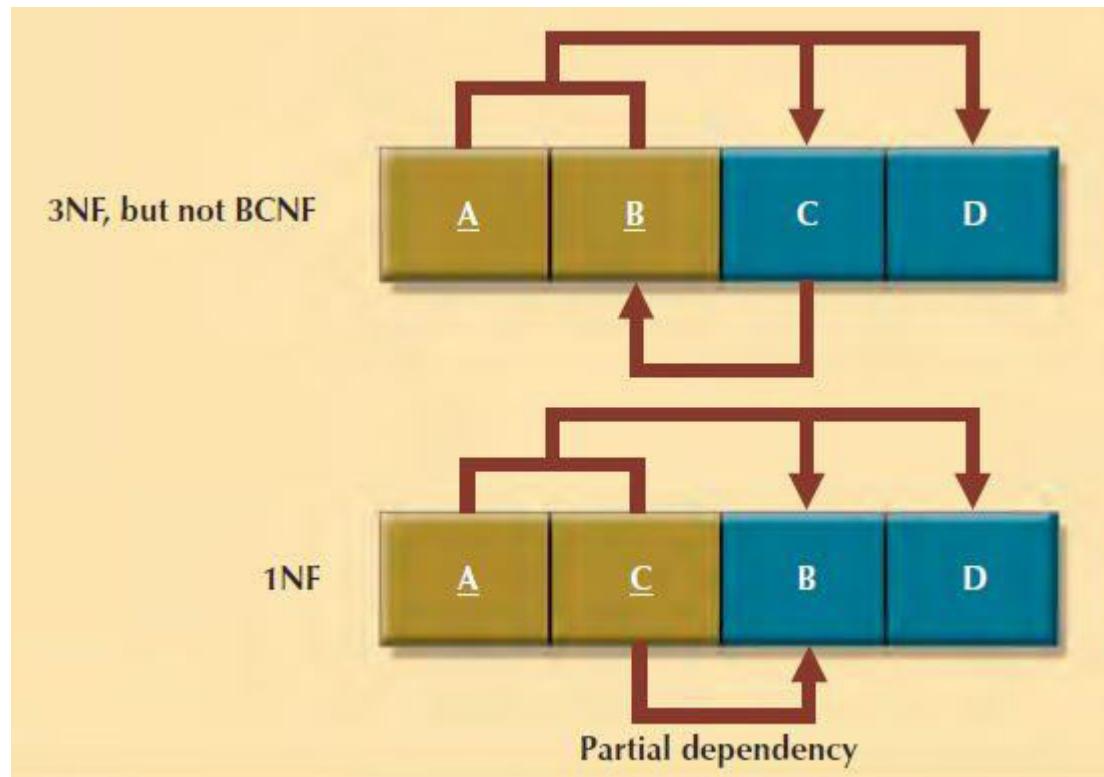
functional dependencies

- $A + B \rightarrow C, D$
- $A + C \rightarrow B, D$
- $C \rightarrow B$
- two candidate keys: $(A + B)$ and $(A + C)$
- has no partial dependencies, nor does it contain transitive dependencies
- $C \rightarrow B$ indicates that nonkey attribute determines part of the PK and that dependency is not transitive or partial because dependent is prime attribute

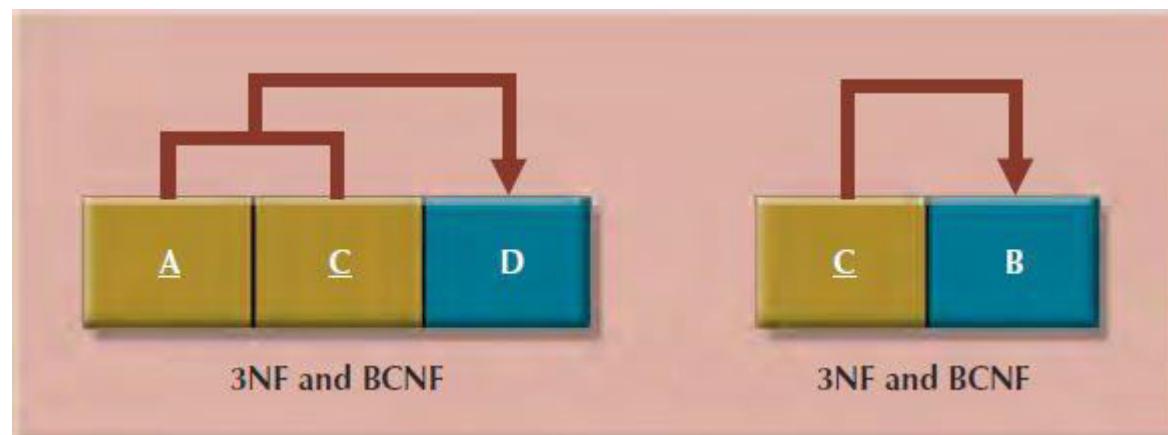
convert into BCNF

- change the primary key to A + C
- appropriate action because dependency C → B means that C is, in effect, superset of B
- at this point, table is in 1NF because it contains partial dependency, C → B
- next, follow standard decomposition procedures to produce results

Decomposition to BCNF



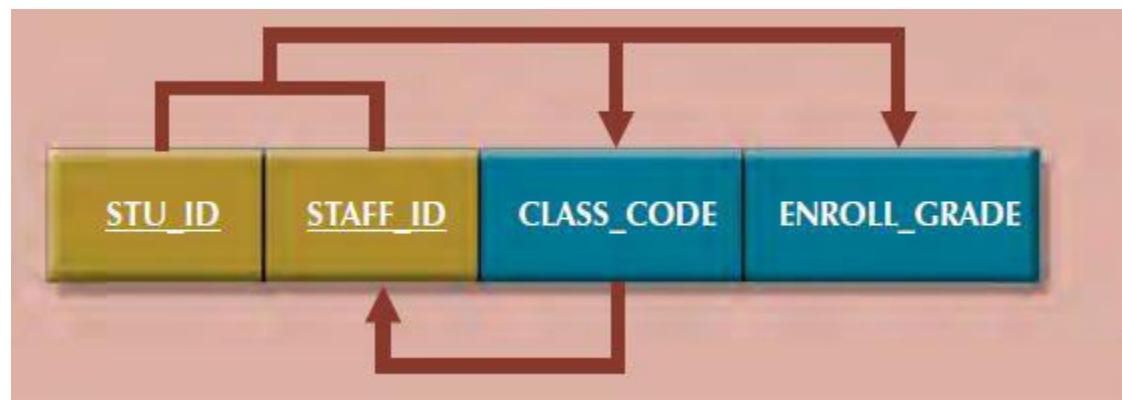
Decomposition to BCNF



Sample Data for a BCNF Conversion

STU_ID	STAFF_ID	CLASS_CODE	ENROLL_GRADE
125	25	21334	A
125	20	32456	C
135	20	28458	B
144	25	27563	C
144	20	32456	B

- each CLASS_CODE identifies class uniquely, course might generate many classes
 - for example, course labeled INFS 420 might be taught in two classes (sections), each identified by unique code to facilitate registration
 - CLASS_CODE 32456 might identify INFS 420, class section 1
 - CLASS_CODE 32457 might identify INFS 420, class section 2
 - CLASS_CODE 28458 might identify QM 362, class section 5
- student can take many classes
- staff member can teach many classes, but each class is taught by only one staff member



- $\text{STU_ID} + \text{STAFF_ID} \rightarrow \text{CLASS_CODE},$
 ENROLL_GRADE
- $\text{CLASS_CODE} \rightarrow \text{STAFF_ID}$

- table represented by this structure has major problem, because it is trying to describe two things: staff assignments to classes and student enrollment information
- dual-purpose table structure will cause anomalies
- if different staff member is assigned to teach class 32456, two rows will require updates, thus producing an update anomaly
- if student 135 drops class 28458, information about who taught that class is lost, thus producing a deletion anomaly



forth normal form (4NF)

- multivalued attributes exist
- Consider possibility that an employee can have multiple assignments and can also be involved in multiple service organizations
- Suppose employee 10123 does volunteer work for Red Cross and United Way; same employee might be assigned to work on three projects: 1, 3, and 4

Table name: VOLUNTEER_V1

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	1
10123	UW	3
10123		4

Table name: VOLUNTEER_V2

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	
10123	UW	
10123		1
10123		3
10123		4

Table name: VOLUNTEER_V3

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	1
10123	RC	3
10123	UW	4

- attributes ORG_CODE and ASSIGN_NUM each may have many different values
- situation is referred to as multivalued dependency
- one key determines multiple values of two other attributes and those attributes are independent of each other
- one employee can have many service entries and many assignment entries
- one EMP_NUM can determine multiple values of ORG_CODE and multiple values of ASSIGN_NUM
- ORG_CODE and ASSIGN_NUM are independent

- if versions 1 and 2 are implemented, tables are likely to contain quite a few null values; do not even have a viable candidate key
- version 3 at least has a PK, but it is composed of all of attributes in the table; meets 3NF requirements, yet it contains many redundancies that are clearly undesirable

set of tables in 4NF

Table name: PROJECT

PROJ_CODE	PROJ_NAME	PROJ_BUDGET
1	BeThere	1023245.00
2	BlueMoon	20198608.00
3	GreenThumb	3234456.00
4	GoFast	5674000.00
5	GoSlow	1002500.00

Table name: ASSIGNMENT

ASSIGN_NUM	EMP_NUM	PROJ_CODE
1	10123	1
2	10121	2
3	10123	3
4	10123	4
5	10121	1
6	10124	2
7	10124	3
8	10124	5

Table name: EMPLOYEE

EMP_NUM	EMP_LNAME
10121	Rogers
10122	O'Leery
10123	Panera
10124	Johnson

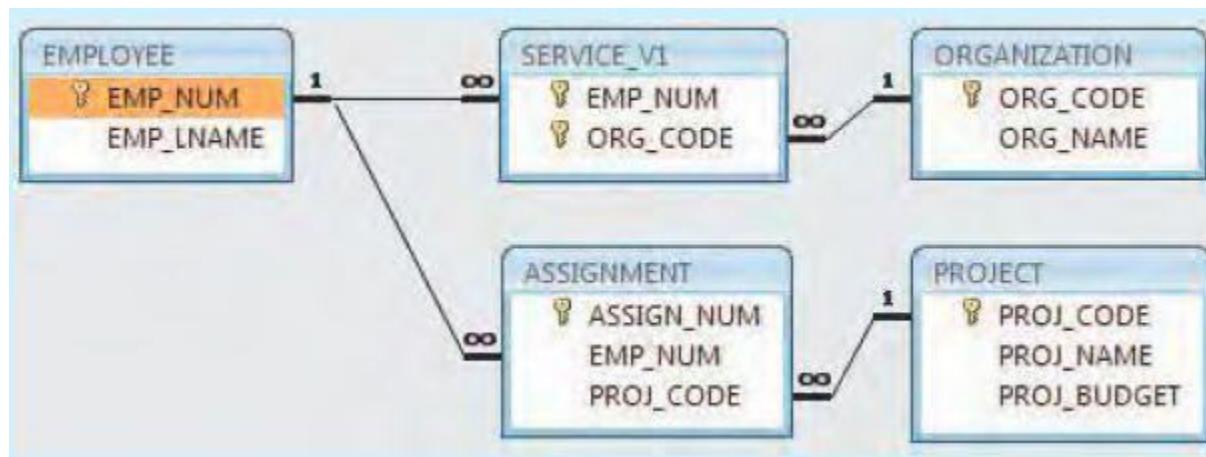
Table name: ORGANIZATION

ORG_CODE	ORG_NAME
RC	Red Cross
UW	United Way
WF	Wildlife Fund

Table name: SERVICE_V1

EMP_NUM	ORG_CODE
10123	RC
10123	UW
10123	WF

relational diagram



fourth normal form (4NF)

- table is in fourth normal form (4NF) when it is in 3NF and has no multivalued dependencies
- ***Normalization should be part of the design process***

DataBase Design

Hit List

Sites

- Sites like emag.ro, piata-az.ro, ...
- Lot of products from a lot of different categories with a lot of different features
- Dynamics – products change, product categories change (tablet are a new addition, MP3 maybe will disappear ...), features change (touchscreen, fingerprint scanner, ...)
- Processor: 286, 386, 486, ... , Pentium 2, 3, ... i3, i5, i7

Sites

- Goal of such sites it is to obtain a lot of hits

emag.ro

The screenshot shows the eMAG.ro website interface for mobile phones. At the top, there's a navigation bar with links for File, Edit, View, History, Bookmarks, Tools, and Help. Below that is a browser header with the URL www.emag.ro/telefoane-mobile/c?ref=bc. The main content area features a banner with a child and the text "Participă!". Below the banner, there are filters for "PRODUSE" (In Stock, All Products, New, Liquidation, Resigned), "Promoții săptămânii" (Weekend Promotions), "Resigilate" (Liquidation), "Lichidări de stoc" (Stock Liquidation), "Card Cadou" (Gift Card), and a competition for an iPhone 5C. The category path is Telephone & Tablete > Telefoane Mobile > Telefoane Mobile. On the right, there are social media icons for Google+ (696) and Facebook (424). The main search area is titled "Alege Tehnologii:" (Choose Technology) and shows categories like Smartphones (286), Dual SIM (143), 4G (87), GPS (277), Touchscreen (298), NFC (105), and Wi-Fi (301). A red badge indicates up to -17% off. Below this, there are filters for price ("Pret (Lei)" with options from Sub 100 to Peste 3000) and producer ("Producători" with options for Samsung, Nokia, LG, Sony, Allview, HTC, Apple, Gsmart, and BlackBerry). The main product grid displays five smartphones: Samsung I9300 GALAXY S3, 16GB, Blue; Nokia 520 Lumia, Black; Samsung S7580 Trend Plus, Black; Samsung S7580 Trend Plus, Pure White; and Allview A4 You, Dual SIM, Black. Each phone has a star rating and review count below it.

emag.ro – mobile phones

- *Search filters on:*
- Product status
- Price interval
- Manufacturer
- Technology
- Display diagonal (in inch)
- Operating System
- Camera
- Internal memory

emag.ro

The screenshot shows a Mozilla Firefox browser window displaying the emag.ro website. The page is for a Samsung I9300 Galaxy S3 smartphone. The main content area is titled 'Specificări' (Specifications) and lists various technical details under several categories: General, Foto video, Alimentare, Comunicare, Memorie, and Multimedia.

General

Smartphone:	Da
Touchscreen:	Da
Tastatura QWERTY:	Nu
Tip tastatura:	Standard
Retea:	3G: 850 / 900 / 1900 / 2100 2G: 850 / 900 / 1800 / 1900
Dual Sim:	Nu
Model Procesor:	Exynos 4212 Quad Core
Procesor (MHz):	1400
Culoare:	Albastru
Dimensiuni (W x H x D mm):	70.6 x 136.6 x 8.6
GPS:	Da
Greutate (g):	133
Sistem de operare:	Android v4.0 (Ice Cream Sandwich)
Tip SIM:	MicroSIM

Foto video

Rezolutie camera (Mp):	8
Rezolutii foto:	3264 x 2448
Blitz integrat:	LED
Rezolutie video:	1920 x 1080
Camera secundara:	1.9 Mp
Caracteristici foto/video:	Senzor BSI Zero shutter lag

Alimentare

Durata conștiință:	2G: 21 ore 3G: 11 ore
Durata în regim de așteptare:	2G: 590 ore 3G: 790 ore
Capacitate (mAh):	2100
Tip baterie:	Li-Ion

Comunicare

Mesagerie:	SMS / MMS / IM
Browsing:	Da
E-mail:	Da

Memorie

Slot memorie (tip):	MicroSD (pana la 64 GB)
Memorie RAM:	1 GB
Memorie internă:	16 GB

Multimedia

FM radio:	Stereo cu RDS
Redare video:	MP4/DivX/XviD/WMV

www.piata-az.ro

- it is similar with emag.ro and with other hit list sites

www.piata-az.ro – Real Estate

PIAȚA 

Cauta după cuvinte cheie Search icon

Home icon Cont me

Categorii anunțuri ➔

Vanzare apartament 3 camere, Marasti, zona BRD, confort sporit 75 000 EUR



Welt Imobiliare va propune spre vanzare un apartament de 3 camere confort sporit, situat la parterul unui bloc de 4 etaje, în Marasti, zona BRD. Apartamentul are o suprafață utilă de 78 de mp, la care se adaugă o boxă la subsol, care figurează de asemenea în CF. În apropierea apartamentului este disponibil un loc de parcare. Finisajele sunt preponderent moderne, inclusiv centrala termică, parchet din lemn, termopane. Imobilul este reabilitat termic. Pentru alte detalii și vizionari va rugam să ne contactați!

<http://www.weltimobiliare.ro/vanzare-apartament-3-camere-cluj-napoca-marasti-id-14629.html>

Share icons: Twitter, Email, Print, Facebook, Plus 0

Contact:
0737.040012
lucian.rogojan@weltimobiliare.ro

**Lucian Rogojan**
Consilier Imobiliar
B-dul 21 Decembrie nr. 24

Tip Oferta:	Vand	Tert:	Agentie Imobiliara
Etaj:	Parter	Vechime imobil:	mai vechi de 2000
Nr. bai:	1	Nr. balcoane:	Nespecificat
Negociabil:	Da	Grad de finisare:	Nespecificat
Centrala Termica:	Da	Compartim.:	Decomandat
Confort:	3	Loc parcare:	Da
TV cablu:	Da	Loc in pivnita:	Da
Agentie:	Welt Imobiliare	Nr. camere:	3
Strada:	-	Suprafață:	78mp

www.piata-az.ro – Real Estate

Categorii anunturi ➔

Filtreaza anunturile după:

Valuta
EUR

Pret
0 - 23398800

Titlu/Descriere

Nr. anunturi per pagina 20

Judet Cluj

Oras Cluj-Napoca

Cartier

Tip Oferta Vand

Tert

Nr. camere

Suprafata

Anunturi apartamente de vanzare

Ultima actualizare: 18.04.2014 20:59

Au fost gasite 5975 anunturi care corespund filtrarii in Imobiliare > Apartamente

◀ ▶

Ap. 2 camere Floresti 17.900 18900

Apartamente cu CFI 1, 2, 3 camere, direct 28500

Ansamblu EUROPA - apartamente 3 si 0

◀ ▶

◀ precedente 1 2 3 4 5 6 7 8 ... 298 299 urmatoare »

Vand apartamente 1, 2, 3 si 4 camere ...
Constructor, vand apartamente cu 1, 2, 3 si 4 camere in Floresti, str. Eroilor nr 128. Pret apartamente [...] Detalii ➔

450 EUR/mp 18.04.2014 20:45

Vazare apartament 2 camere etaj inter...
RE/MAX Grup de Lux vinde apartament 2 camere, suprafata utila de 52,57 mp, situat la etajul 1/4 intr-un [...] Detalii ➔

52 000 EUR 18.04.2014 18:06

www.piata-az.ro – Real Estate

- *Filter on*
- Price
- Location, location, location
- Type
- Third person
- No. of rooms
- Area
- Age

www.piata-az.ro – Auto-Moto

Categorii anunturi ➔

focus 2004 inmatriculat

2 550 EUR



impecabil tehnic si estetic distributie schimbata facturi doveditoare folii omologate rar roti vara iarna 3m parbriz incalzit itp 2016 accept orice fel de test in reprezentanta taxa nerecuperata ATENTIE KM REALI PRET NEDISCUTABIL

18.04.2014
23:29
Cluj-Napoca
Cluj
vizualizari: 112

[Twitter](#) [Email](#) [Print](#) [Facebook](#) + 0

Tip Oferta:	Vand	Carburant:	Diesel
Negociabil:	Nu	Numar usi:	4-5
Marca:	Ford	Aer conditionat:	Nu
Model:	Focus	Climatronic:	Nu
Stare autoturism:	Inmatriculat in RO	Tip caroserie:	Hatchback
An fabricatie:	2004	Norma de poluare:	Euro 4
Rulaj:	149000km	Transmisie:	Manuala
Capacitate :	1800cmc	Istoric:	Fara accident
Extraurban:	5	Tractiune:	Fata
Jante Al:	Da	Alarma:	Da
ABS:	Nu	Radio:	Da
CD:	Da	Casetofon:	Da
Servo directie:	Da	Servo frana:	Da
Inchidere centralizata:	Da	Tapiterie piele:	Da
Airbag dreapta:	Da	Airbag sofer:	Da
Oglinzi incalzite:	Da	Oglinzi electrice:	Da

Contact:
0747766629
bogdan_7862@yahoo.com

Contacteaza prin email

Numele tau *

Telefonul tau

Email-ul tau *

Mesajul tau * Sunt interesat de oferta dumneavoastra, va rog sa ma contactati

www.piata-az.ro – Auto-Moto

Categorii anunturi ➔

Filtreaza anunturile după:

Valuta
EUR ↴

Pret
0 - 33500

Ultima actualizare: 18.04.2014 23:29

Au fost gasite 973 anunturi care corespund filtrarii in Auto-Moto-Velo > Autoturisme

Titlu/Descriere

Judet
Cluj ↴

Nr. anunturi per pagina 20 ↴

« precedente 1 2 3 4 5 6 7 8 ... 48 49 urmatoare »

focus 2004 inmatriculat 2 550 EUR
18.04.2014 23:29
impecabil tehnic si estetic distributie schimbata
facturi doveditoare folii omologate rar roti vara roti
iarna 3m parbriz incalzit [...]
[Detalii ➔](#)

cumpar cu plata pe loc autoturisme 1 EUR
18.04.2014 23:29
cumpar cu plata pe loc in lei sau valuta urmatoarele
marci de autoturisme : bmw! opel! ford! mercedes!
[...]
[Detalii ➔](#)

Marca
-- ↴

Model
-- ↴

Stare autoturism
-- ↴

An fabricatie
1980 2012

www.piata-az.ro – Auto-Moto

- *Filter on*
- Price
- Location, location, location
- Type
- Manufacturer, Model (model feature which depend on another feature (manufacturer))
- Status
- Age, Mileage
- No. of doors
- Fuel

Feature values

- fixed list of values
 - which can change
 - Fuel: diesel, petrol, ...
 - hybrid, electric, ...
- integer, float
 - could have unit of measurement / or not
 - Area (m^2 , km^2) / No. of rooms, doors
- text

Design Data Driven Application

- Design a general template
- Show everything in this template

Categories

- Hierarchy
- Order of category in hierarchy

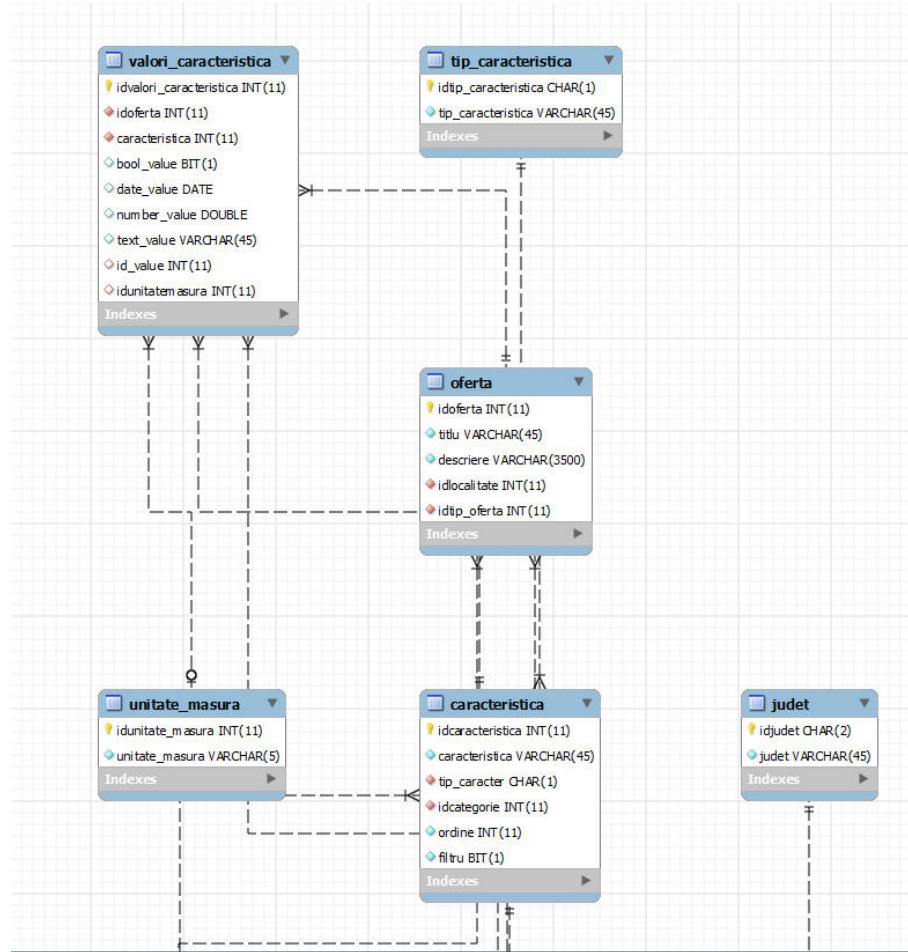
Location, Location, Location

- important
- design database structure to model the location structure: country, county, city
 - area (neighborhood, quartier)
- will be able to search location

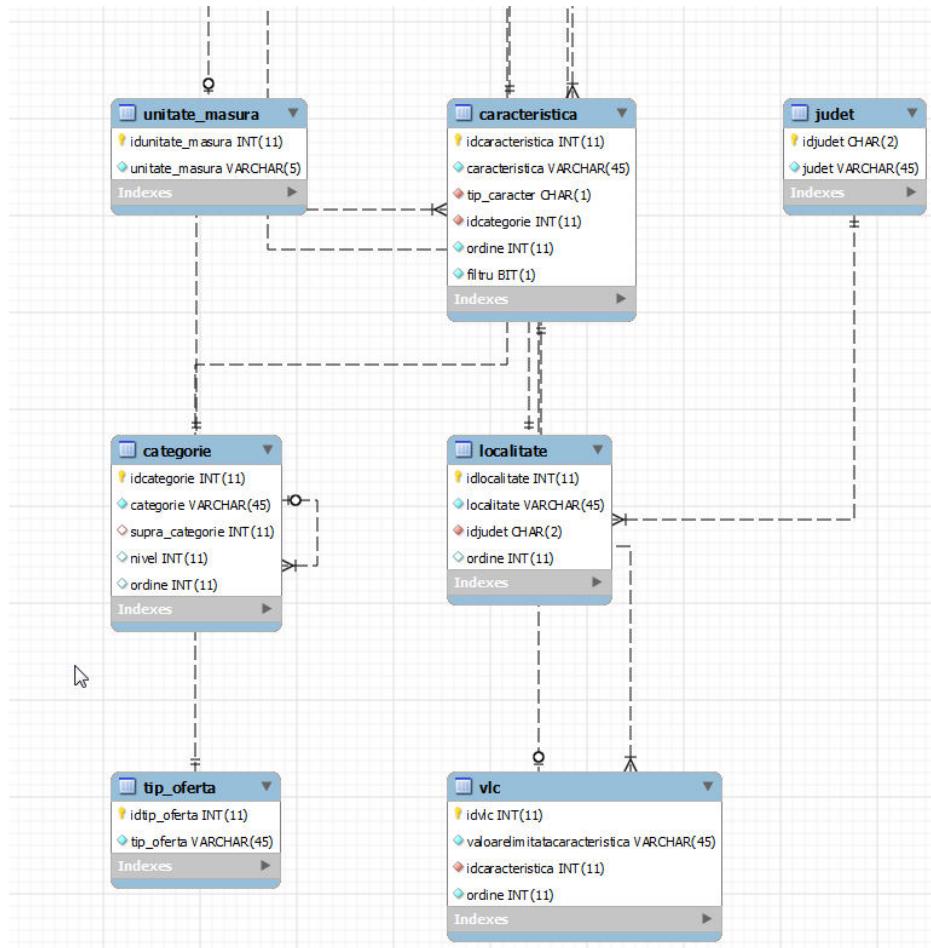
Feature

- Fixed list of values stored in the database
 - with order (integer) and filter (bool true or false)
- will be able to search feature value

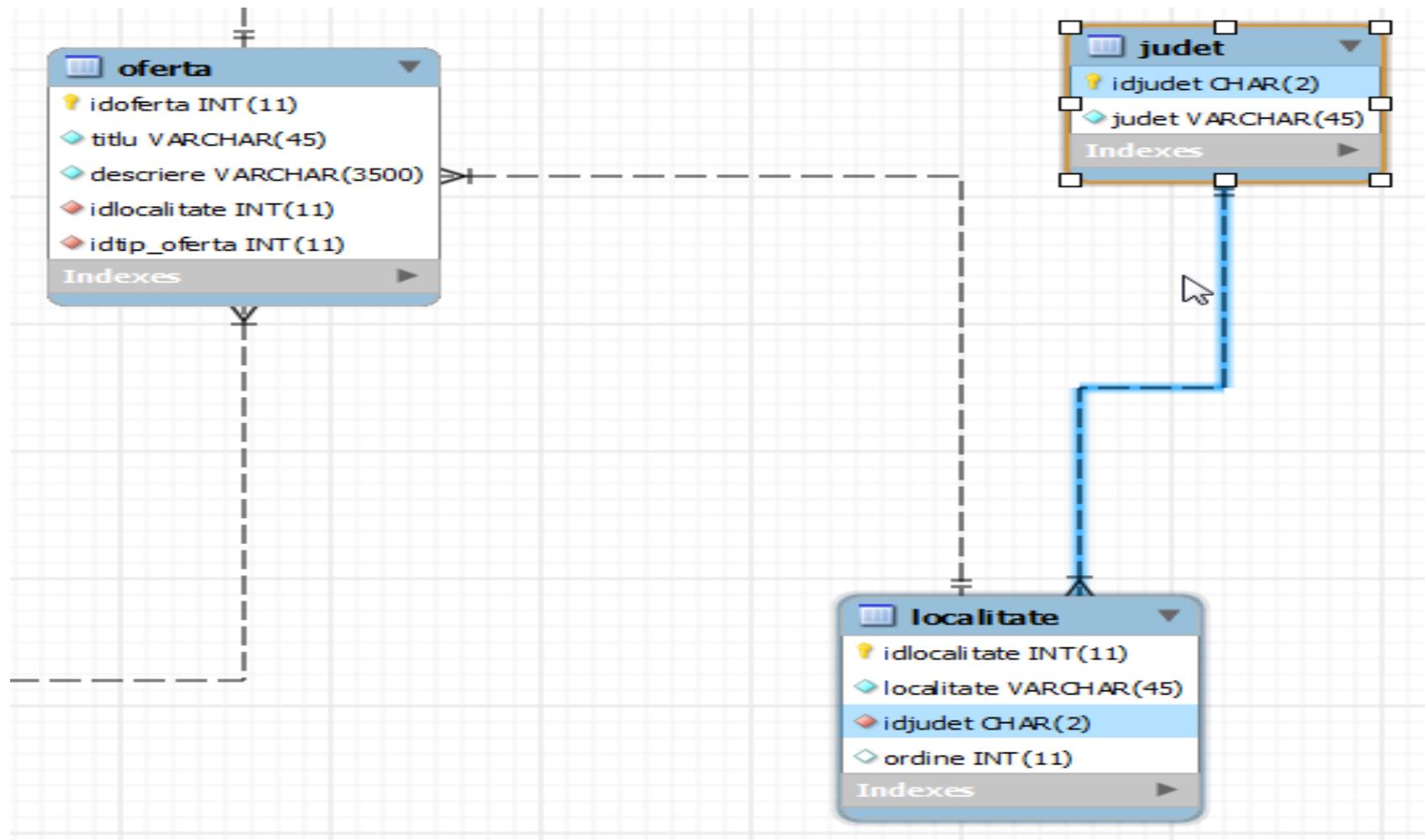
DataBase Diagram



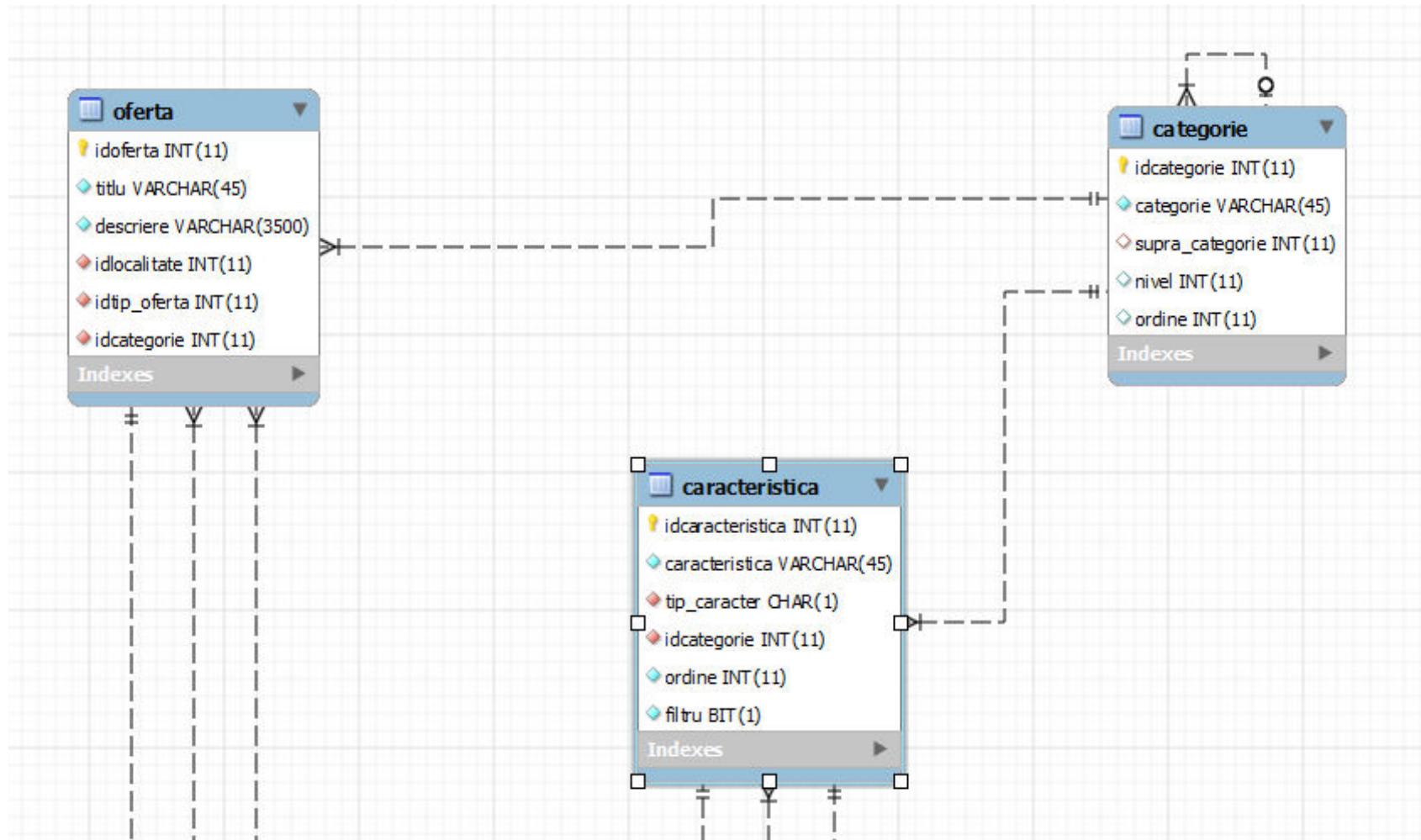
DataBase Diagram



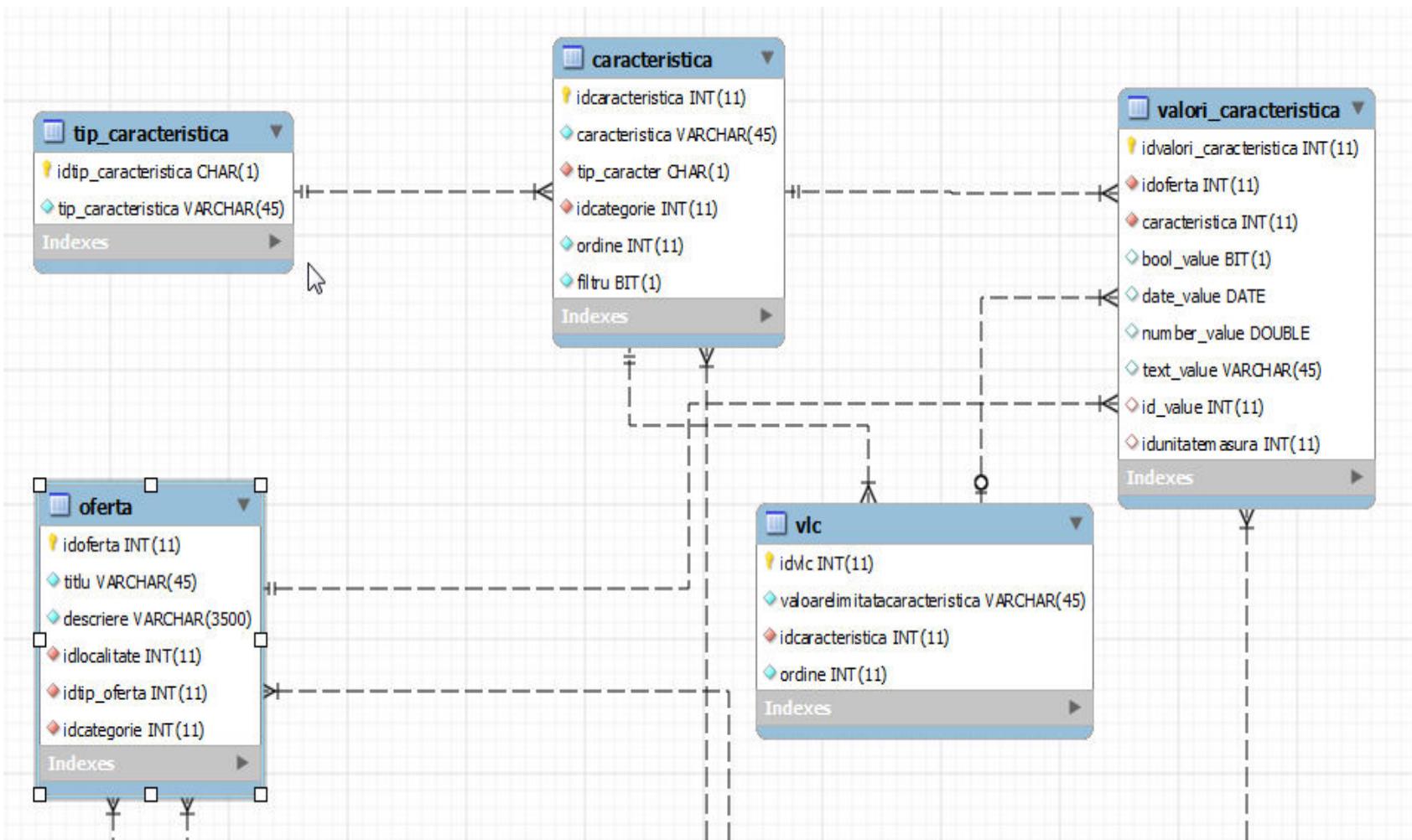
DB Diagram - Location



DB Diagram - Categories



DB Diagram - Feature



```
SELECT idcategorie, categorie, supra_categorie,  
nivel, ordine FROM categorie ORDER BY nivel,  
supra_categorie, ordine;
```

1	Imobiliare	1	1
5	Prestari Servicii	1	2
8	Auto-Moto- Velo	1	3
2	Garsoniere	2	1
3	Apartamente	2	2
4	Case, vile	2	3
	Resurse umane		
6	- consiliere- servicii psihologi	2	1
7	Amenajari interioare	2	2
9	Autoturisme	2	1

```
SELECT idlocalitate, localitate, ordine, judet
FROM judet JOIN localitate ON judet.idjudet =
localitate.idjudet ORDER BY judet, ordine
```

7	Brasov	1	BRASOV
1	Cluj-Napoca	1	CLUJ
2	Campia Turzii	2	CLUJ
3	Dej	3	CLUJ
4	Gherla	4	CLUJ
5	Huedin	5	CLUJ
6	Turda	6	CLUJ

DataBase

- SELECT idcaracteristica,
caracteristica,tip_caracteristica.tip_caracteristica,
categorie.categorie, caracteristica.ordine,
filtru FROM caracteristicaJOIN
tip_caracteristica ON
caracteristica.tip_caracter =
tip_caracteristica.idtip_caracteristicaJOIN
categorie ON caracteristica.idcategorie =
categorie.idcategorie ORDER BY categorie,
ordine

DataBase

1	Stare autoturism	Valori Limitate	Autoturisme	1	1
2	An fabricatie	Numeric	Autoturisme	2	1
3	Rulaj	Numeric	Autoturisme	3	1
4	Istoric	Valori Limitate	Autoturisme	4	1

DataBase

- SELECT C.idcaratteristica, C.caratteristica,
V.valoarelimitatacaratteristica, V.ordine
- FROM caratteristica C JOIN vlc V ON
C.idcaratteristica = V.idcaratteristica
- ORDER BY caratteristica, ordine

DataBase

4	Istoric	Nespecificat	1
4	Istoric	Fara accident	2
4	Istoric	Cu accident anterior	3
1	Stare autoturism	Inmatriculabil	1
1	Stare autoturism	Inmatriculat in RO	2
1	Stare autoturism	Neinmatriculabil	3
1	Stare autoturism	Preluare de leasing	4

View NValor

- SELECT valori_caracteristica.idoferta AS idoferta,
- caracteristica.caracteristica AS caracteristica,
- valori_caracteristica.number_value AS VValue,
- unitate_masura.unitate_masura AS UM
- FROM valori_caracteristica JOIN caracteristica ON
valori_caracteristica.caracteristica =
caracteristica.idcaracteristica JOIN unitate_masura
ON valori_caracteristica.idunitatemasura =
unitate_masura.idunitate_masura
- WHERE caracteristica.tip_caracter = 'N'

View NValor

1

An fabricatie

2006

1

Rulaj

164254

km

View VValor

- SELECT valori_caracteristica.idoferta AS idoferta,
- caracteristica.caracteristica AS caracteristica,
- vlc.valoarelimitatacaracteristica AS Vvalue
- FROM valori_caracteristica JOIN caracteristica ON valori_caracteristica.caracteristica = caracteristica.idcaracteristica JOIN vlc ON valori_caracteristica.id_value = vlc.idvlc AND valori_caracteristica.caracteristica = vlc.idcaracteristica
- WHERE caracteristica.tip_caracter = 'V'

View VValor

1	Stare autoturism	Inmatriculat in RO
1	Istoric	Fara accident

Offer

- SELECT idoferta, titlu, descriere, localitate,
tip_oferta.tip_oferta, categorie.categorie
- FROM piata.oferta JOIN localitate ON
oferta.idlocalitate = localitate.idlocalitate
- JOIN tip_oferta ON oferta.idtip_oferta =
tip_oferta.idtip_oferta
- JOIN categorie ON oferta.idcategorie =
categorie.idcategorie

```
SELECT idoferta, titlu, descriere, localitate,  
tip_oferta.tip_oferta, categorie.categorie FROM
```

- 1 ford focus titaniu taxa zero !!!
- Interior de piele, faruri adaptable la viraje, proiectoare ceata, senzori ploaie, senzori lumina, parbriz si luneta incalzite, 4 x geamuri electrice, oglinzi el si incalzite. inchidere centralizata, 2 chei tip briceag, dublu climatronic, sistem audio Sony, volan de ...
- Cluj-Napoca
- Vand
- Autoturisme

Feature

- SELECT * FROM Nvalori
- UNION
- SELECT *, NULL AS UM FROM Vvalori
- WHERE idoferta = 1

SELECT idoferta, caracteristica, Value, UM

1	An fabricatie	2006	
1	Rulaj	164254	km
1	Stare autoturism	Inmatriculat in RO	
1	Istoric	Fara accident	

Examples of Normal Form Problems

- given relation R with four attributes $ABCD$
 - sets of FDs, assuming those are the only dependencies that hold for R
- a) Identify the candidate key(s) for R
 - b) Identify the best normal form that R satisfies (1NF, 2NF, 3NF, or BCNF)
 - c) If R is not in BCNF, decompose it into a set of BCNF relations that preserve the dependencies

$$C \rightarrow D, C \rightarrow A, B \rightarrow C$$

- Candidate keys: B
- R is in 2NF but not 3NF
- CA, CB, CD

$(1) C \rightarrow D, (2) C \rightarrow A, (3) B \rightarrow C$

- Step 1
- (0) - PK = ABCD

$$(1)C \rightarrow D, (2)C \rightarrow A, (3)B \rightarrow C$$

- Step 2
- (1) - PK = ABC
 - eliminate D from $(1)C \rightarrow D$
- (2) - PK = BCD
 - eliminate A from $(2)C \rightarrow A$
- (3) - PK = ABD
 - eliminate C from $(3)B \rightarrow C$

$(1)C \rightarrow D, (2)C \rightarrow A, (3)B \rightarrow C$

- Step 3
- (1) - PK = ABC
 - (12) – PK = BC
 - (13) – PK = AB
- (2) - PK = BCD
 - (21) – PK = BC
 - (23) – PK = BD
- (3) - PK = ABD
 - (31) – PK = AB
 - (32) – PK = BD

$$(1) C \rightarrow D, (2) C \rightarrow A, (3) B \rightarrow C$$

- Step 4
- (12) – PK = BC
 - (123) = PK = B
- (13) – PK = AB
 - (132) = PK = B
- (23) – PK = BD
 - (231) – PK = B
- Primary Key is B

$$B \rightarrow C, D \rightarrow A$$

- Candidate keys: BD
- R is in 1NF but not 2NF
- AD, BCD
 - AD, BC, BD

$$ABC \rightarrow D, D \rightarrow A$$

- Candidate keys: ABC, BCD
- R is in 3NF but not BCNF.
- not in BCNF since $D \rightarrow A$ and D is not a key
 - if we split up R as AD, BCD we cannot preserve the dependency $ABC \rightarrow D$
 - no BCNF decomposition

$$A \rightarrow B, BC \rightarrow D, A \rightarrow C$$

- Candidate keys: A
- R is in 2NF but not 3NF (because of FD: $BC \rightarrow D$)
- $BC \rightarrow D$ violates BCNF since BC does not contain a key
- split up R as in: BCD, ABC

$$AB \rightarrow C, AB \rightarrow D, C \rightarrow A, D \rightarrow B$$

- Candidate keys: AB, BC, CD, AD
- R is in 3NF but not BCNF (because of the FD: $C \rightarrow A$)
- $C \rightarrow A$ and $D \rightarrow B$ both cause violations
- decompose into: AC, BCD
 - this does not preserve $AB \rightarrow C$ and $AB \rightarrow D$
 - and BCD is still not BCNF because $D \rightarrow B$
- need to decompose further into: AC, BD, CD
 - when we attempt to revive lost functional dependencies by adding ABC and ABD , we see that these relations are not in BCNF form
- there is no BCNF decomposition

Guide to Modern Database & NoSQL movement

NoSQL

- Until present, like most new and upcoming technologies, NoSQL is shrouded in mist of fear, uncertainty, and doubt - world of developers probably divided into:
 - Those who love it
 - Those who deny it
 - Those who ignore it

NoSQL

- No + SQL = *No relational* (NonRel)
- used today as umbrella term for all databases that don't follow RDBMS
- represents class of products and collection of diverse, and sometimes related, concepts about data storage and manipulation that often relate to large data sets accessed and manipulated on Web scale

Challenges of RDBMS

- RDBMS assumes well-defined structure in data, data is dense and is largely uniform
- properties of data can be defined up front and interrelationship are well established and systematically referenced
- RDBMS can certainly deal with some irregularities and lack of structure but ...

Challenges of RDBMS

- in context of massive sparse data sets with loosely defined structures, RDBMS appears a forced fit
- with massive data sets typical storage mechanism & access method get stretched
- denormalizing tables, dropping constraints, and relaxing transactional guarantee can help an RDBMS scale
 - but after these modifications an RDBMS starts resembling NoSQL products

Challenges of RDBMS

- Flexibility comes at a price
- NoSQL alleviates problems that RDBMS imposes and makes it easy to work with large sparse data, but in turn takes away power of transactional integrity and flexible indexing and querying
- one of features most missed in NoSQL is SQL, and product vendors are making all sorts of attempts to bridge this

History of NoSQL

- Google has, over past few years, built out massively scalable infrastructure for its search engine and other applications, created full mechanism that included distributed file system, column family oriented data store, distributed coordination system, and MapReduce-based parallel algorithm execution environment

History of NoSQL

- Lucene, first to develop opensource version
- Lucene developers joined Yahoo, where with help of host of other contributors, they created a parallel universe that mimicked all pieces of Google distributed computing stack
- opensource alternative is Hadoop
<http://hadoop.apache.org>
- somewhere toward first of its releases emerged idea of NoSQL - Hadoop laid groundwork for rapid growth of NoSQL

History of NoSQL

- year after Google papers had catalyzed interest in parallel scalable processing and nonrelational distributed data stores, Amazon decided to share some of its own success story - in 2007 Amazon presented its ideas of distributed highly available and eventually consistent data store named Dynamo
- endorsement of NoSQL from two leading web giants several new products emerged in this space
- in around 5 years NoSQL and related concepts for managing big data have become widespread and use cases have emerged including Facebook, Netflix, Yahoo, EBay, Hulu, IBM, and many more

Big Data

- currently, any data set over few terabytes is classified as big data
- typically size where data set is large enough to start spanning multiple storage units
- size at which traditional RDBMS techniques start showing first signs of stress

Scalability

- ability of system to increase throughput with addition of resources to address load increases
- vertical achieved either by provisioning large and powerful resource to meet additional demands
- horizontal can be achieved by relying on cluster of ordinary machines to work as unit, typically involves adding additional nodes to serve additional load

MapReduce

- parallel programming model that allows distributed processing on large data sets on a cluster of computers
- patented by Google, but ideas are freely shared and adopted in number of open-source implementations

MapReduce

- derives its ideas and inspiration from concepts in functional programming
- map function applies a function to each element in list
- reduce function applies function on all elements of data structure and produces single result or output
- simple idea of map and reduce has been extended to work on large data sets

MapReduce

- trivial example to explain flow
- collection of key/value pairs as follows:
- [{ "Student 1": "Grade 1"}, {"Student 2": "Grade 2"}, ..., {"Student n": "Grade n"}]
- simple map function on this collection could get names of all those who pass
- reduce function could work on this output to simply count number

Google's Bigtable

SORTED ORDERED COLUMN- ORIENTED STORES

- each unit of data can be thought of as set of key/value pairs, where unit itself is identified with primary identifier row-key
- units are stored in ordered-sorted manner

HBase

- <http://hbase.apache.org>
- created at Powerset (now part of Microsoft) in 2007; donated to Apache foundation before was acquired
- implemented in Java
- JRuby shell allows command-line access to store
- Thrift, Avro, REST, and protobuf clients
- Java API is available with distribution
- no native querying language; Hive (<http://hive.apache.org>) provides SQL-like interface
- Apache License
- Facebook, StumbleUpon, Hulu, Ning, Mahalo, Yahoo!, and others

Hypertable

- www.hypertable.org
- created at Zvents in 2007; now independent open-source project
- implemented in C++, uses Google RE2 regular expression library
- command-line shell available; Thrift interface supported
- language bindings have been created
- HQL (Hypertable Query Language) is SQL-like abstraction for querying data; adapter for Hive
- Open-Source License
- Zvents, Baidu (China's biggest search engine), Rediff (India's biggest portal)

Cloudata

- [www.cloudata.org/.](http://www.cloudata.org/)
- created by Korean developer named YK Kwon
- implemented in Java.
- command-line access is available; Thrift, REST, and Java API are available
- CQL (Cloudata Query Language) SQL-like query language
- Open-Source License

HashMap

KEY-VALUES STORES

- associative array simple data structure that can hold set of key/value pairs
- provide a very efficient algorithm for accessing data
- unique value in set can be easily looked up to access data

Couchbase CouchDB

- www.membase.org
- started in 2009 by NorthScale, later renamed Membase
- implemented in Erlang, C, and C++
- Memcached-compliant API with some extensions
- Open-Source License
- Zynga, NHN, and others

Kyoto Cabinet

- <http://fallabs.com/kyotocabinet/>
- implemented in C++
- APIs for C, C++, Java, C#, Python, Ruby, Perl, Erlang, OCaml, and Lua
- Open-Source License
- Mixi, blog posts and mailing lists suggest that there are many users but no public list is available

Redis

- [http://redis.io/.](http://redis.io/)
- project started in 2009 by Salvatore Sanfilippo created for his startup (<http://lloogg.com/>); still an independent project, author employed by VMware, who sponsor its development
- implemented in C
- access via Redis command-line interface and set of well-maintained client libraries for languages like Java, Python, Ruby, C, C++, Lua, Haskell, AS3, and more
- Open-Source License
- Craigslist

- open-source implementations of ideas proposed by Amazon Dynamo

Cassandra

- [http://cassandra.apache.org/.](http://cassandra.apache.org/)
- developed at Facebook and open sourced in 2008, donated to Apache foundation
- implemented in Java
- command-line access to store; Thrift interface and internal Java API exist
- clients for multiple languages including Java, Python, Grails, PHP, .NET. and Ruby; Hadoop integration supported
- Open-Source License
- Facebook, Digg, Reddit, Twitter, and others

Voldemort

- [http://project-voldemort.com/.](http://project-voldemort.com/)
- created by data and analytics team at LinkedIn in 2008
- implemented in Java
- provides for pluggable storage using either Berkeley DB or MySQL
- integrates with Thrift, Avro, and protobuf
- (<http://code.google.com/p/protobuf/>) interfaces; can be used in conjunction with Hadoop
- Open-Source License
- LinkedIn

Riak

- <http://wiki.basho.com/>
- created at Basho in 2008
- implemented in Erlang; also, uses bit of C and JavaScript
- interfaces for JSON (over HTTP) and protobuf clients
- libraries for Erlang, Java, Ruby, Python, PHP, JavaScript
- Open-Source License
- Comcast and Mochi Media

MongoDB / CouchDB

DOCUMENT DATABASES

- loosely structured sets of key/value pairs in documents, typically JSON (JavaScript Object Notation), and not *documents* (though these could be stored too)
- treat document as whole and avoid splitting a document into its constituent name/value pairs
- at collection level, this allows for putting together diverse set of documents into single collection
- allow indexing of documents on the basis of not only its primary identifier but also its properties

MongoDB

- www.mongodb.org
- created at 10gen
- implemented in C++
- JavaScript command-line interface; drivers exist for languages including C, C#, C++, Erlang, Haskell, Java, JavaScript, Perl, PHP, Python, Ruby, and Scala
- SQL-like query language
- Open-Source License
- FourSquare, Shutterfly, Intuit, Github, and more

CouchDB

- <http://couchdb.apache.org> / www.couchbase.com
- most authors are part of Couchbase, Inc.
- work started in 2005 and it was incubated into Apache in 2008
- implemented in Erlang with some C and JavaScript
- standard web tools and clients to access database
- Open-Source License
- Apple, BBC, Canonical, Cern, and more

Graph databases and XML data stores qualify as NoSQL

GRAPH DATABASES

- ACID-compliant graph database
- facilitates rapid traversal of graphs

Neo4j

- <http://neo4j.org>
- created at Neo Technologies in 2003
- implemented in Java
- command-line access to store is provided; REST interface also available
- client libraries for Java, Python, Ruby, Clojure, Scala, and PHP
- SPARQL protocol and RDF Query Language
- Open-Source License
- Box.net

Relational

- most common classic database pattern
- set-theory-based systems implemented as two-dimensional tables with rows and columns
- strictly enforce type are generally numeric, strings, dates, and uninterpreted blobs
- extensions such as array or cube

Good For:

- when layout of data is known in advance
- business problems are aptly modeled this way

Not-So-Good For:

- when data is highly variable or deeply hierarchical
- data problems that exhibit high degree of record-to-record variation will be problematic
- want database that makes less restrictions in advance on what you can put into it

Columnar

- like key-value database, values are queried by matching keys
- like relational, their values are groups of zero or more columns, though each row is capable of populating however many it wants
- columns are inexpensive to add versioning is trivial, and there is no real storage cost for unpopulated values

Good For:

- traditionally developed with horizontal scalability as primary design goal
- suited to “Big Data” problems, living on clusters of tens, hundreds, or thousands of nodes
- tend to have built-in support for features such as compression and versioning

- canonical example of good columnar data storage problem is indexing web pages
- pages on Web are highly textual (benefits from compression), somewhat interrelated, and change over time (benefits from versioning)

Not-So-Good For:

- different drawbacks; one thing they have in common is that it's best to design your schema based on how you plan to query data
- if data usage patterns can't be defined in advance - for example, fast ad hoc reporting - then columnar database may not be best fit

Key-Value

- simplest model; maps simple keys to (possibly) complex values like huge hash
- has most flexibility of implementation
- hash lookups are fast, (in the case of Redis, speed was its primary concern)
- hash lookups are also easily distributed (Riak took advantage of this fact for focusing on simple-to-manage clusters)
- simplicity can be a downside for any data with complex modeling requirements

Good For:

- little or no need to maintain indexes; often designed to be horizontally scalable, extremely fast, or both
- suited for problems where data are not highly related

Not-So-Good For:

- often lacking indexes and scanning capabilities, won't help you if you need to be able to perform queries on your data, other than basic CRUD (Create, Read, Update, Delete) operations

Document

- allow for any number of fields per object and even allow objects to be nested to any depth as values of other fields
- common representation of these objects is as JavaScript Object Notation (JSON)

- since documents don't relate to each other like relational databases, they are relatively easy to shard and replicate across several servers, making distributed implementations fairly common
- MongoHQ tends to tackle availability by supporting creation of datacenters that manage huge datasets for Web
- CouchDB focuses on being simple and durable, where availability is achieved by master-master replication of fairly autonomous nodes

Good For:

- suited to problems involving highly variable domains
- when you don't know in advance what exactly your data will look like
- often map well to object-oriented programming models – less impedance mismatch when moving data between database model and application model

Not-So-Good For:

- performing elaborate join queries on highly normalized relational database schemas
- document should generally contain most or all of relevant information required for normal use
- while in relational database you naturally normalize your data to reduce or eliminate copies that can get out of sync, with document databases, denormalized data is the norm

Graph

- focuses more on free interrelation of data than actual values
- growing in popularity for many social network applications
- other database styles group collections of like objects into common buckets, graph databases are more free-form

- queries consist of following edges shared by two nodes or, namely, traversing nodes
- growing straightforward social examples to occupy more nuanced use cases, such as recommendation engines, access control lists, and geographic data

Good For:

- tailor-made for networking applications
- prototypical example is social network, where nodes represent users who have various kinds of relationships to each other
- modeling this kind of data using any of other styles is often tough fit
- perfect matches for object-oriented system

Not-So-Good For:

- generally not suitable for network partitioning - don't scale out well
- likely it'll be one piece of larger system, with bulk of data stored elsewhere and only relationships maintained in graph

data is the new oil

- we sit upon vast ocean of data, yet until it's refined into information, it's unusable
- ease of collecting and ultimately storing, mining, and refining data out there starts with database you choose

facebook

- though social graph may seem to clearly function best, simply have far too much data to choose
- likely going to choose “Big Data” implementation, such as HBase or Riak
- force your hand into choosing columnar or key-value store

bank transactions

- relational database is clearly best option
- Neo4j also supports ACID transactions

- there are several more dimensions to consider when choosing database
- durability, availability, consistency, scalability, and security

Genre, Datatypes, Data Relations

- MongoDB - Document, Typed, None
- CouchDB - Document, Typed, None
- Riak - Key-value, Blob, Ad hoc (Links)
- Redis - Key-value Semi-typed, None
- PostgreSQL - Relational, Predefined and typed, Predefined
- Neo4j - Graph, Untyped, Ad hoc (Edges)
- HBase - Columnar, Predefined and typed, None

Replication, Sharding, Concurrency

- MongoDB – master-slave (via replica sets), Yes, Write lock
- CouchDB - master-master, Yes (with filters in BigCouch), Lock-free MVCC
- Riak - Peer-based master-master, Yes, Vector-clock
- Redis - master-slave, Add-ons (e.g., client), None
- PostgreSQL - master-slave, Add-ons (e.g., PL/Proxy), table/row writer lock
- Neo4j - master-slave, No, Write lock
- Hbase, master-slave, Yes via HDFS, Consistent per row

Shard

- scale horizontally without compromising functionality
- horizontal partitioning is database design principle whereby rows of database table are held separately, each partition forms part of shard, which may in turn be located on separate database server or physical location
- advantages to this partitioning approach
- tables are divided and distributed into multiple servers, total number of rows in each table in each database is reduced - reduces index size, which generally improves search performance

Shard

- can be placed on separate hardware, and multiple shards can be placed on multiple machines – enables distribution of database over large number of machines, which means that database performance can be spread out over multiple machines, greatly improving performance
- if database shard is based on some real-world segmentation of data it may be possible to infer appropriate shard membership easily and automatically, and query only relevant shard

Shard

- it has been done for long time by hand-coding (especially where rows have an obvious grouping)
- desire to support sharding automatically, both in terms of adding code support for it, and for identifying candidates to be sharded separately
- consistent hashing is one form of automatic sharding to spread large loads across multiple smaller services and servers
- where distributed computing is used to separate load between multiple servers (for performance or reliability reasons), shard approach may be useful

Main Differentiator, Weaknesses

- MongoDB - Easily query Big Data, Embedability
- CouchDB - Durable and embeddable clusters, Query-ability
- Riak - Highly available, Query-ability
- Redis - Very, very fast, Complex data
- PostgreSQL - Best of OSS RDBMS model, Distributed availability
- Neo4j - Flexible graph, BLOBs or terabyte scale
- HBase - Very large-scale, Hadoop infrastructure, Flexible growth, query-ability

CAP theorem

- proves that you can create a distributed database that is Consistent (writes are atomic and all subsequent requests retrieve new value), Available (database will always return value as long as single server is running), Partition tolerant (system will still function even if server communication is temporarily lost - that is, network partition)
 - but you can have only two at once

CAP theorem

- Now, for the sake of argument, imagine you are (like me) a passionate Carmen Serban fan and the date is 2008
- suddenly, at party celebrating release of Carmen's last album, tidal wave drags you out to sea.
- without any means of communication, you are effectively partitioned from rest of system (world) there you wait for four student years
- one morning in 2013 you are awakened by ... a captain has discovered you! after five years alone, captain asks : "How many albums does Carmen Serban have?"

CAP theorem

- can answer question with most recent value you have (which is now five years old) - you are Available
- can decline to answer question, knowing since you are Partitioned, your answer may not be Consistent with rest of world
 - captain won't get his answer, but state of world remains consistent (if he sails back home, he can get correct answer)
- role of queried node, you can either help keep world's data consistent or be available, but not both

CAP theorem

- story ends with captain rescuing you, and you return home to find any new album and live happily ever after
- as long as you remain on land, you needn't be Partition tolerant and can remain Available and Consistent

CAP theorem

- Riak allows clients to decide at request time what level of consistency they require
- Redis, PostgreSQL, and Neo4J are consistent and available (CA); they don't distribute data and so partitioning is not an issue (though arguably, CAP doesn't make much sense in non-distributed systems)

CAP theorem

- MongoDB and Hbase are generally consistent and partition tolerant (CP)
 - in the event of network partition, they can become unable to respond to certain types of queries - in practice, hardware failure is handled gracefully - other still-networked nodes can cover for downed server - but strictly speaking, in CAP theorem sense, they are unavailable
- CouchDB is available and partition tolerant (AP)
- even though two or more CouchDB servers can replicate data between them, CouchDB doesn't guarantee consistency between any two servers

CAP theorem

- most of these databases can be configured to change CAP type
- MongoDB can be CA, CouchDB can be CP, but here we've noted their default or common behaviors

Draft

PHP and MongoDB Web Development

MongoDB databases, collections, and documents

- MongoDB server hosts a number of databases
- databases act as containers of data and they are independent of each other
- contains one or more collections
- document stored in collection is unit of data
- document contains set of fields or key-value pairs

MongoDB databases, collections, and documents

- collection is set of documents, logically analogous to concept of table in relational database - unlike tables, you don't have to define structure of data to be stored
- document contains set of fields or key-value pairs - keys are strings, values can be of various types: strings, integers, floats, timestamps, can even store document as value of field in another document

Document

- structure as JSON or JavaScript Object Notation
- { _id : ObjectId("4db31fa0ba3aba54146d851a") }
- username : "CalinCenan"
- email : "Calin.Cenan@cs.utcluj.ro"
- age : 49
- is_admin : true
- created :"Apr 17 2013 " }
- value for first field, `_id`, is autogenerated,
MongoDB automatically generates an ObjectId
for each document you create in collection and
assigns it for that document

BSON - data exchange format for MongoDB

- when store this document in database, it is serialized into special binary encoded format known as BSON, short for binary JSON - default data exchange format
- more efficient than conventional formats such as XML and JSON, both in terms of memory consumption and processing time
- supports all data types supported by JSON (string, integer, double, Boolean, array, object, null) plus some special data types such as regular expression, object ID, date, binary data, and code

BSON - data exchange format for MongoDB

- programming languages such as PHP, Python, Java, have libraries that manage conversion of language-specific data structures (for example, associative array in PHP) to and from BSON

<http://www.mongodb.org/>

- MongoDB supports wide variety of platforms
- run on Windows (XP, Vista, and 7)
- various flavors of Linux (Debian/Ubuntu, Fedora, CentOS, and so on)
- and OS X running on Intel-based Macs

- bin/mongod - database process
- bin/mongos - sharding controller
- bin/mongo - database shell
- drivers - client drivers for most programming languages are available at mongodb.org
 - use shell ("mongo") for administrative tasks

to run single server database

- \$ mkdir /data/db
 - location where Mongo will store its data files
- \$./mongod
 - mongo javascript shell connects to localhost and test database by default
- \$./mongo
- > help
- > show dbs
 - local (empty)

Utilities

- bin/mongodump - dump tool for backups, snapshots, etc..
- bin/mongorestore - restore dump
- bin/mongoexport - export single collection to test (JSON, CSV)
- bin/mongoimport - import from JSON or CSV
- bin/mongofiles - utility for putting and getting files from MongoDB GridFS
- bin/mongostat - show performance statistics

Stopping MongoDB

- shutdown running MongoDB server
 - hitting Control + C in terminal
 - signal the server to do clean shutdown, flush, and close its data files
- from mongo interactive shell issue `shutdownServer()` command
- > `use admin`
 - switched to db admin
- > `db.shutdownServer()`

- > use myfirstdb
- commands to create documents in collection named movies
- > db.movies.insert({name:"Source Code", genre:"sci-fi", year:2011})
- > db.movies.insert({name:"Star Wars – a new hope", genre:"sci-fi ", year:1977})
- > db.movies.insert({name:"Office Space", genre:"comedy", year:1999})
- > db.movies.insert({name:"Terminator", genre:"action", year:1984})
- command returns all documents from movies collection:
- > db.movies.find()

Install PHP driver for MongoDB

- functional PHP environment installed, running on top of Apache web server
- download, copy `php_mongo.dll` file to PHP extension directory (ext inside your PHP installation)
- in `php.ini` file inside your PHP installation add following line
`extension=php_mongo.dll`

phpinfo()

mongo

MongoDB Support	enabled
Version	1.0.11

Directive	Local Value	Master Value
mongo.allow_empty_keys	0	0
mongo.allow_persistent	1	1
mongo.auto_reconnect	1	1
mongo.chunk_size	262144	262144
mongo.cmd	\$	\$
mongo.default_host	localhost	localhost
mongo.default_port	27017	27017
mongo.long_as_object	0	0
mongo.native_long	0	0
mongo.utf8	1	1

(mongodb://<hostname>:<port_number>)
default: localhost, on port 27107

- try{
- \$mongo = new Mongo();
 - //create a connection to MongoDB
- \$databases = \$mongo->listDBs();
 - //List all databases
- print_r(\$databases);
- \$mongo->close();
- } catch(MongoConnectionException \$e) {
 - /die(\$e->getMessage());
- }

CRUD (Create, Read, Update, Delete)
blog application
PHP and MongoDB web development

blogpost.php (styles.css)

Blog Post Creator

Title

Content

This is my first blog post in this blogging application. I hope this works! I plan to write more.

Save

blogpost.php

- <form action="blogpost.php" method="post">
- <h3>Title</h3>
- <p><input type="text" name="title" id="title"/></p>
- <h3>Content</h3>
- <textarea name="content" rows="20"></textarea>
- <p><input type="submit" name="btn_submit" value="Save"/></p>
- </form>

blogpost.php

- \$connection = new Mongo();
- \$database = \$connection->
 selectDB('myblogsite');
- \$collection = \$database->
 – selectCollection('articles');

blogpost.php

- construct array with user-submitted data, and pass this array as argument insert()
- \$article = array();
- \$article['title'] = \$_POST['title'];
- \$article['content'] = \$_POST['content'];
- \$collection->insert(\$article);
- insert() method stores data in collection
- \$article array automatically receives field `_id`, which is autogenerated unique ObjectId of inserted BSON document

Creating databases and collections implicitly

- we do not have to run any CREATE DATABASE or CREATE TABLE statement beforehand
- database and collection namespaces get created on the fly
- selectDB() either selects existing database or creates it implicitly if it is not there
- same thing goes for selectCollection()

Shortcut approach for selecting database/collection

- \$connection = new Mongo();
- \$collection = \$connection-> myblogsite->articles

'safe' inserts

- program control does not wait for response from MongoDB database
 - just signals MongoDB to insert document, and then immediately moves on to execute next statement in code
- if we want to wait for response of database we send an optional safe = true argument to insert() method (by default set to false)
- an optional timeout parameter, which specifies how long (in milliseconds) program will wait for response

'safe' inserts

- try {
- \$status = \$connection->insert(array('title'
 => 'Blog Title',
- 'content' => 'Blog Content'),
- array('safe' => True));
- echo "Insert operation complete";
- } catch (MongoCursorException \$e) {
- die("Insert failed ".\$e->getMessage());
- }

Benefits of safe inserts

- guards against all kinds of user level errors during inserts
 - for example, if we try to insert document with a non-unique `_id`, will raise an exception
- MongoDB may also run out of disk space when inserting documents
 - check against such potential pitfalls
- in multi-server setup, when data is being stored on multiple machines (replica set), ensures that data is written on at least one

Setting user generated _id

- we can set `_id` field of document explicitly
- useful for situations when want to give each document unique identifier derived from our own rule
- when setting `_id` in this way, we should always do 'safe' insert, ensures when there is more than one document in collection with same `_id`,
MongoCursorException is thrown

MongoDate object

- represents ISODate data type in BSON
- when instantiated, MongoDate gives current timestamp
- can use built-in strtotime() function in PHP to instantiate MongoDate objects
- `$timestamp = new MongoDate(strtotime('2011-05-05 12:00:00'));`
- `print date('g:i a, F j', $timestamp->sec); //prints 12 pm, May 05`

blogs.php

My Blogs

Hello World!

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi...

[Read more](#)

Learnt something important today

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi...

[Read more](#)

Blogging is fun!

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi...

[Read more](#)

- \$cursor = \$collection->find();
 - while (\$cursor->hasNext()):
 - \$article = \$cursor->getNext();
-
- \$article = \$collection->findOne(array('_id' => new Mongold(\$id)));
 - echo \$article['title'];
 - echo \$article['content'];

blog.php

My Blogs

Hello World!

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incident ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?

- `$cursor = $collection->find();`
- `find()` method returns `MongoCursor` object, that we can use to iterate through results
- `<?php while ($cursor->hasNext()):`
- `$article = $cursor->getNext(); ?>`
- `<h2><?php echo $article['title']; ?></h2>`
- `<p><?php echo substr($article['content'], 0, 200).'...'; ?></p>`
- `<a href="blog.php?id=<?php echo $article['_id']; ?>">Read more`
- `<?php endwhile; ?>`

- blog.php receives `_id` of article as HTTP GET parameter
- `findOne()` method on articles collection, send `_id` value as parameter used to retrieve single document
- unlike `find()` which we use to retrieve cursor of set of documents that we can iterate over
- `$id = $_GET['id'];`
- `$article = $collection->findOne(array('_id'=>`
- `new Mongold($id)));`

Mongo Query Language

- queries expressed as JSON objects
- passed as arguments to find() method
- `>db.movies.find({"genre":"sci-fi"})`
- `{ "_id": ObjectId("4db439153ec7b6fd1c9093ec"),`
- `"name": "Source Code",`
- `"genre": "sci-fi", "year": 2011 }`
- `$moviesCollection->find(array("genre": "sci-fi"));`

Mongo Query Language

- query is expressed as an associative array:
- //get all movies where genre == sci-fi
- \$moviesCollection->find(array("genre" : "sci-fi"));
- can also specify multiple query parameters:
- \$moviesCollection->find(array('genre' => 'comedy', 'year' => 2013));
- when we don't pass any query arguments to find(), it gets an empty array by default (empty JSON object in MongoDB) and matches all documents
- \$moviesCollection->find(); / \$moviesCollection->find(array())

MongoCursor object

- query performed with `find()`, returns cursor
- iterate over this cursor to retrieve all documents matched by query
- `$cursor = $movieCollection->find(array('genre' => 'action'));`
- `while ($cursor->hasNext()) {`
 - `$movie = $cursor->getNext();`
 - //do something with `$movie`
 - }

- `hasNext()` checks whether or not there are any more objects in cursor
- `getNext()` returns next object pointed by cursor, and then advances cursor
- documents do not get loaded into memory unless we invoke either `getNext()` or `hasNext()` on cursor, for keeping memory usage low

- can do this in foreach loop as well:
- `$cursor = $movieCollection->find(array('genre' => 'action'));`
- `if ($cursor->count() === 0) {`
 - `foreach ($cursor as $movie) {`
 - `//do something with $movie`
 - `}`
- `}`

- can use built-in `iterator_to_array()` function to change cursor returned by query into array
- programmers who find cursor approach a little too difficult prefer this trick
- downside of using `iterator_to_array()`, if size of data returned is very large, using `iterator_to_array()` may lead to major performance decrease, as PHP will try to load entire data into memory

- \$cursor = \$movieCollection->find({'genre': 'action'});
- \$array = iterator_to_array(\$cursor);
- if (!empty(\$array)) {
 - foreach(\$array as \$item){
 - //do something with \$item
 - }
- }

Conditional Queries

- //get all items with field 'x' greater than 100
- \$collection->find(array('x' => array('\$gt' => 100)));
- //get all items with field 'x' lower than 100
- \$collection->find(array('x' => array('\$lt' => 100)));
- //get all items with field 'x' greater than or equal to 100
- \$collection->find(array('x' => array('\$gte' => 100)));
- //get all items with field 'x' lower than or equal to 100
- \$collection->find(array('x' => array('\$lte' => 100)));

Conditional Queries

- //get all items with field 'x' between 100 and 200
- \$collection->find(array('x' => array('\$gte' => 100, '\$lte' => 200)));
- //get all items with field 'x' not equal to 100
- \$collection->find(array('x' => array('\$ne' => 100)));

- when using MongoDB conditional query operators in PHP (or any special operator that has \$ as a prefix), we must use operator within single quotes (') and not double quotes (")
- //Using single quotes (') is ok.
- \$collection->find(array('x' => array('\$gt' => 100)));
- //\\$ is escaped within double quotes (")
- \$collection->find(array('x' => array("\\$gt" => 100)));
- //this will cause an error
- \$collection->find(array('x' => array("\$gt" => 100)));

Dashboard

Title	Saved at	Action
Adding manpower to a late software ...	1:50 pm, May 13	View
Research supports a specific theory...	1:49 pm, May 13	View
Nature always sides with the hidden...	1:49 pm, May 13	View
Any given program, when running, is...	1:48 pm, May 13	View
Blogging is fun!...	6:48 pm, May 9	View

[Previous](#) 2 [Next](#)

Sort query results

- `sort()` method invoked on `MongoCursor` object sorts query results based on value of specified field
- `$cursor->sort(array('saved_at' => -1))`
 - // -1 means descending order
- `$cursor->sort(array('saved_at' => 1))`
 - // 1 means ascending order
- // sort on 'x' in ascending order and 'y' in descending order
- `$cursor->sort(array('x' => 1, 'y' => -1));`

- `count()` on `MongoCursor` object returns number of items in cursor
- `$cursor = $articleCollection->find();`
 - //gets all articles
- //skip first five articles in cursor
- `$cursor->skip(5);`
- //start iterating from sixth article
- `while($cursor->hasNext()) {`
 - `$cursor->getNext();`
- `limit()` enables to limit number of results returned

Update documents in MongoDB

- public bool MongoCollection::
- update (\$criteria, \$newobj, \$options)
- \$criteria is array that specifies document that is going to be updated
- database is queried with \$criteria to select document intended to be updated
- \$newobj is document (represented as an array) that is going to replace old document
- \$options specifies optional arguments to update()

Blog Post Editor

Title

The only perfect science is hind-sight

Content

Consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Save

edit.php

- \$article = array();
- \$article['title'] = \$_POST['title'];
- \$article['content'] = \$_POST['content'];
- \$article['saved_at'] = new MongoDate();
- \$articleCollection->update(array('_id' => new Mongoid(\$id)),
 \$article);

Optional arguments to update

- safe: same as doing 'safe' insert
- \$collection->update(\$criteria, \$newobj,
array('safe' => True));
- timeout: specifies timeout (in milliseconds)
on update operation
- \$collection->update(\$criteria, \$newobj,
array('safe' => True, 'timeout' => 100));

Optional arguments to update

- `multiple`: even if we get more than one document matched by criteria, updates just one of them - setting `multiple` to `True` switches this default behavior
- `$collection->update($criteria, $newobj, array('multiple' => True));`

Upsert

- MongoDB allows us to perform an interesting operation named Upsert, short for "Update if exists, INSERT otherwise"
- setting an optional upsert flag to True in update() method
- \$users->update(array('email' => 'alice@wonderland.com'),
array('firstname' => 'Alice', 'lastname'=>
'Liddell'), array('upsert' => True));

- alternate approach for updating documents in MongoDB is using `save()` method on `MongoCollection` object:
- `$collection->save($document);`
- difference between `update()` and `save()` is that when using `save()`, if document does not exist in database it gets created
- almost the same as doing `upsert=True`

- `$document = array('name' => 'Adam Smith', 'age' => 27);`
- `$collection->save($document);`
 - //inserts the object
- `$document['age'] = 31;`
- `$collection->save($document);`
 - //updates the object

set

- \$set allows us to set value of particular field of document
- \$articles->update(array('_id' => Mongoid('4dcd2abe5981'))),
- array('\$set' => array('title' => 'New Title')));

inc

- \$inc lets us increment value of field by specified number
- \$articles->update(array('_id' => Mongoid('4dcd2abe5981')),
array('\$set' => array('content' => 'New Content'),
'\$inc' => array('update_count' => 1)));

unset

- deleting fields with \$unset is just opposite of \$set can use it to remove field from document
- `$articles->update(array('_id' => MongoDB('4dcd2abe5981')), array('$unset' => array('title' => True))));`

rename

- renaming fields with \$rename
- modifier operator which can be used to change name of field in a document
- \$articles->update(array(),
• array('\$rename' => array('saved_at' =>
'created_at')),
• array('multiple' => True));

Dashboard

Title

Title	Date	Action
Research supports LIGO's specific theory...	1:45 pm, May 13	View Edit Delete
Nature always sides with the hidden...	1:49 pm, May 13	View Edit Delete
Any given program, when running, is...	1:48 pm, May 13	View Edit Delete
Blogging is fun!...	6:48 pm, May 9	View Edit Delete
Learnt something important today...	6:47 pm, May 9	View Edit Delete

[Previous](#) 2 [Next](#)

delete.php

- use remove() method to delete an article from database
- takes array as its parameter, which it uses to query document it is going to delete
- if multiple documents match query, all of them are deleted
- //delete document(s) from the movies collection where genre is drama
- **\$movies->remove(array('genre' => 'drama'));**

Optional arguments to remove

- safe:
- timeout:
- justOne:

relationships between documents

- real-world objects are related to each other to some degree
- embedding document within other document
- creating reference from one document to other

Embedded documents

- top-level document contains related document in itself
- for example, author document may have an address document embedded in it:

Embedded documents

- { "_id" : ObjectId("4dd491695072aefc456c9aca"),
- "username" : "AndreAgasii",
- "email" : "Andre.Agassi@atptennis.com",
- "fullname" : "Andre Kirk Agassi",
- "joined_at" : ISODate("2011-02-03 T03:41:29.703Z"),
- "address" : {
- "street" : "35 Sinai st.",
- "city" : "Caprica", "state" : "CC",
- "zipcode" : 512 }

Referenced documents

- document may also have reference to another document
- for example, an article document may refer to an author document by storing ID of author as field:

Referenced documents

- { _id : ObjectId("4dcd2abe5981aec801010000"),
- title : "The only perfect site is hind-site",
- content : "Loren ipsum dolor sit amet...",
- saved_at : ISODate('2012-04-08 T18:42:57.949Z'),
- author_id :
 ObjectId("4dd491695072aefc456c9aca")
- }

comment.php

Blogging is fun!

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Post your comment

Name

Email

Nice post! Keep it up.

- \$comment = array(
- 'name' => \$_POST['commenter_name'],
- 'email' => \$_POST['commenter_email'],
- 'comment' => \$_POST['comment'],
- 'posted_at' => new MongoDate());
- \$collection->update(array('_id' => new Mongold(\$id)),
- array('\$push' => array('comments' =>
- \$comments)));

- each top-level object in data model should have collection of its own
- if relation between two objects is such that one contains the other, the latter should be embedded in the former
- for example, article may contain one or more comments by embedding them in itself

Embedding gets preference

- embedded objects are more efficient in terms of performance tend to be faster
 - documents share disk space
 - embedded documents are loaded to memory when you load their container documents, whereas to get referenced document you have to hit database again
- this does not mean you should always go for embedded objects when designing data model

Querying embedded objects

- {
- name : "Angelina Jolie",
- address : {
- city : "Springfield",
- state : "Florida"
- }
- }
- {
- name : "Lara Croft",
- address : {
- city : "Miami",
- state: "Florida"
- }
- }

Querying embedded objects

- `$users->find(array('address' => array('city' => 'Springfield', 'state' => 'Florida')));`
- `$users->find(array('address.state' => 'Florida'));`
- `$users->find(array('address.city' => 'Springfield', 'address.state' => 'Florida'));`

Aggregation Queries

- document in collection will have following fields:
- title: string, represents title of article
- description: content of article, also string
- author: string representing name of author
- category: of article
- rating: integer between 1 and 10
- tags: array, contains between 1 to 5 distinct tags of article
- published_at: timestamp

- { "_id" : ObjectId("4dfb49545981ae0a02680700"),
- "title" : "Programmers will act rational when all other possibilities have been exhausted.",
- "author" : "Spock",
- "category" : "Programming",
- "rating" : 7,
- "tags" : ["security", "code", "howto"],
- "published_at" : ISODate("2013-05-007
T12:32:20Z") }

MapReduce

- functional programming paradigm to perform aggregation and batch processing of data
- map
 - breakdown task into smaller subtasks and execute them to produce intermediate results
- reduce
 - combine intermediate results and produce final output

- map() takes array as input and performs operation on each element on array
- reduce() takes result array of map() as input and combines all elements in that array into single element by performing some operation
- [1, 2, 3, 4, 5] - sum of squares of all these numbers
- map() takes array and applies $f(x) = x^2$ on each in array and produces output [1, 4, 9, 16, 25]
- reduce() takes this output array, sums all numbers in it, and outputs number 55 ($1+4+9+16+25 = 55$)

- Google demonstrated how MapReduce model could be used to process large datasets concurrently in cluster
- one machine in cluster assumes role of master node; it partitions input into smaller subtasks and distributes them among multiple worker nodes
- workers run in parallel to process subtasks and return results back to master - combines results and produces final output
- gain speed and scalability
- Google's proposed programming model was soon adopted by other

mongo

- var map = function() { emit(this.author, 1);
}

output of map

- {
- "Luke Skywalker" : [1],
- "Spock" : [1],
- "Han Solo" : [1, 1, 1],
- "James Kirk" : [1, 1],
-
- }

mongo

- var reduce = function(key, values) {
- ...var count = 0;
- ...for (var i = 0; i < values.length; i++){
- ... count += values[i];
- ... }
- ... return count;
- ... };

mongo

- db.runCommand({
- ... mapreduce: 'sample_articles',
- ... map: map,
- ... reduce: reduce,
- ... out: 'articles_per_author'
- ... })

mongo

- { "result" : "articles_per_author",
- "timeMillis" : 70,
- "counts" : {
- "input" : 1000,
- "emit" : 1000,
- "output" : 8 },
- "ok" : 1
- }

```
> db.articles_per_author.find()
```

- { "_id" : "Darth Vader", "value": 126 }
- { "_id" : "Han Solo", "value": 130 }
- { "_id" : "Hikaru Sulu", "value": 119 }
- { "_id" : "James Kirk", "value": 115 }
- { "_id" : "Leia Organa", "value": 134 }
- { "_id" : "Luke Skywalker", "value": 132 }
- { "_id" : "Nyota Uhura", "value": 127 }
- { "_id" : "Spock", "value": 117 }

MapReduce on subset of collection

- > db.runCommand({
- ... mapreduce: 'sample_articles',
- ... query: {category: 'Programming'},
- ... map: map,
- ... reduce: reduce,
- ... out: 'articles_per_author'
- ... })

Concurrency

- MapReduce jobs running on MongoDB server are single threaded
 - because of limitations imposed by current JavaScript engines
- developers at 10gen are looking for alternative ways to achieve parallelism of MapReduce jobs
- concurrency can be achieved by sharding database

Sharding

- process of partitioning data into multiple nodes, and is performed when volume of data becomes too large to be handled in single machine

tagcloud.php

Tag Cloud

algorithms benchmark career-advice code engineering
hacking hardware howto nosql
opensource optimization
presentation productivity
programming security software
testing tutorial version-control
version-control

- \$tags = iterator_to_array(\$db->selectCollection('tagcount')->find());
- function getBiggestTag(\$tags)
- {
- reset(\$tags);
- \$firstKey = key(\$tags);
- return (int)\$tags[\$firstKey]['value'];
- }

- \$biggestTag = getBiggestTag(\$tags);
- foreach(\$tags as &\$tag) {
- \$weight = floor(\$tag['value'] / \$biggestTag) * 100;
- switch(\$weight){
- case (\$weight < 10):
- \$tag['class'] = 'class1';
- break;.....
- case (\$weight >= 90):
- \$tag['class'] = 'class9';
- break; } }

aggregation using group()

- method on collection short-circuit approach for doing MapReduce similar to using GROUP BY in SQL
- key: specifies key or set of keys by which documents will be grouped
- initial: base aggregator counter, initial values before aggregation
- reduce: aggregates documents; takes two arguments, current document being iterated over, and aggregation counter

aggregation using group()

- optional arguments:
- cond: query object; documents matching this query will be used in grouping
- finalize: function that runs on each item in result set (before returning item); can either modify or replace returning item

avg_rating.php

Authors' Ratings

Author	Articles	Average Rating
Han Solo	25	6
Hikaru Sulu	21	6
Leia Organa	23	6
Luke Skywalker	23	6
James Kirk	10	6
Spock	14	7
Nyota Uhura	23	6
Darth Vader	22	5

- \$result = \$collection->group(\$key, \$initial,
new MongoCode(\$reduce),
- array(
- 'finalize' =>
- new MongoCode(\$finalize),
- 'condition' => \$condition
-)
-);

MapReduce vs group()

- group() returns result in single BSON object, and therefore has to be very small; cannot be applied on key having more than 10,000 distinct values
- group() method blocks entire database
- when aggregating over large dataset, MapReduce is preferred option

`distinct()`

- invoke method on collection, passing key as argument
- following command gives us all distinct values of category field in `sample_article` collection:
 - `> db.sample_articles.distinct('category')` [
 - "Programming", "Data Structures",
 - "Mathematics", "Operating System", ...]

count()

- used to count number of objects in collection
- takes document selector as parameter and returns number of documents matching
- <?php
- //get the number of articles written by Spock
- \$collection->count(array('author' => 'Spock'));

Using MongoDB with Relational Databases

... very often used for
Web analytics ...

Potential use cases

- Storing results of aggregation queries
- Data archiving
- Logging
- Storing entity metadata

Store results of aggregation

- results of expensive aggregation queries (COUNT, GROUP BY, and so on) can be stored in MongoDB database - application quickly get result from MongoDB without having to perform same query again, until result becomes stale (at which point the query will be performed and result will be stored again)
- don't need to know anything about structure of result data beforehand - rows returned by aggregation query could be stored as BSON documents

Data archiving

- as volume of data grows, queries and other operations on relational table increasingly take more time
- solution to this problem is to partition data into two tables: an Online table, which contains working dataset, and an Archival table that holds old data
- size of online table will remain, but archival table will grow larger

Data archiving

- when schema of online table changes, we will have to apply same changes to archive table
- could use MongoDB collection as archive because of its flexible schema

Logging

- asynchronous insert feature and Mongo query language (and MapReduce) makes MongoDB a better choice for logging
- log HTTP requests in MongoDB in Web Analytics using MongoDB

Storing entity metadata

- maps entities of domain into tables
- entities could be physical, real-world objects (users, products, and so on), or they could be something virtual (blog posts, categories, and tags)
- determine what pieces of information you need to store for each of these entities, and then you design database schema and define table structures

Storing entity metadata

- assume that we need to store some additional information for some of these entities
- don't know what kind of information we need to store, and they vary from one entity to another

Scale

- what makes MongoDB unique is their ability to efficiently handle arbitrarily nested, schemaless data documents and ability to scale across several servers, by replicating (copying data to other servers) or sharding collections (splitting collection into pieces) and performing queries in parallel
- promote availability

Replica Sets

- built to scale out, not to run stand-alone built for data consistency and partition tolerance
- sharding data has cost: if one part of collection is lost, whole thing is compromised
- deals with this implicit sharding weakness in simple manner: duplication
- rarely run single Mongo instance in production but rather replicate stored data across multiple services

- create data directories
- \$ mkdir ./mongo1 ./mongo2 ./mongo3
- default port is 27017 – start 3 servers, add replSet flag with name *book* and specify ports
- \$ mongod --replSet book --dbpath ./mongo2 --port 27011 --rest
- \$ mongod --replSet book --dbpath ./mongo2 --port 27012 --rest
- \$ mongod --replSet book --dbpath ./mongo3 --port 27013 --rest

- \$ mongo localhost:27011
- > rs.initiate({
- _id: 'book',
- members: [
- { _id: 1, host: 'localhost:27011' },
- { _id: 2, host: 'localhost:27012' },
- { _id: 3, host: 'localhost:27013' }
-])
- > rs.status()

object called rs (replica set)

- `status()` command will let us know when our replica set is running
- should see that one server outputs:
- [rs Manager] `replSet PRIMARY`
- and two servers will have following output:
- [rs_sync] `replSet SECONDARY`
- > `db.echo.insert({ say : 'Hello world!' })`

isMaster() function

- \$ mongo localhost:27013
- connecting to: localhost:27013/test
- > db.isMaster() {
- "setName" : "book",
- "ismaster" : false, "secondary" : true,
- "hosts" : ["localhost:27013",
"localhost:27012", "localhost:27011"],
- "primary" : "localhost:27012",
- "ok" : 1 }

- attempt to insert another value.
- > db.echo.insert({ say : '*is this thing on?*' })
- not master
- letting us know that we cannot write to secondary node; nor can you directly read from it - there is only one master per replica set, and you must interact with it

- problem is deciding who gets promoted when master node goes down
- deals with this by giving each mongod service a vote, and one with freshest data is elected new master
- shut down current master
- [ReplSetHealthPollTask] replSet info localhost:27012 is now down (or...[rs Manager] replSet can't see a *majority*, *will not try to elect self*

Mongo philosophy of server setups have an odd number of servers

- relaunch other servers and watch logs; when nodes are brought back up, they go into recovery state and attempt to resync data with new master
- if original master had data that did not yet propagate those operations are dropped
- write in Mongo replica set isn't considered successful until most nodes have copy of data
- sometimes, network connection between nodes is down Mongo dictates that *majority of nodes that can still communicate make up the network*

- consider five-node network, for example
- if connection issues split it into three node fragment and two-node fragment, larger fragment has clear majority and can elect master and continue servicing requests
- with no clear majority, quorum couldn't be reached

- some databases (e.g., CouchDB) are built to allow multiple masters, but Mongo is not, and so it isn't prepared to resolve data updates between them
- MongoDB deals with conflicts between multiple masters by simply not allowing them

Sharding

- safely and quickly handle very large datasets through horizontal sharding by value ranges
- rather than single server hosting all values in collection, some range of values are split (or in other words, sharded) onto other servers
- Mongo makes this easier by autosharding, managing this division for you

- launch couple of (nonreplicating) mongoDB servers
- special parameter necessary to be considered shard server
- \$ mkdir ./mongo4 ./mongo5
- \$ mongod --shardsvr --dbpath ./mongo4 --port 27014
- \$ mongod --shardsvr --dbpath ./mongo5 --port 27015

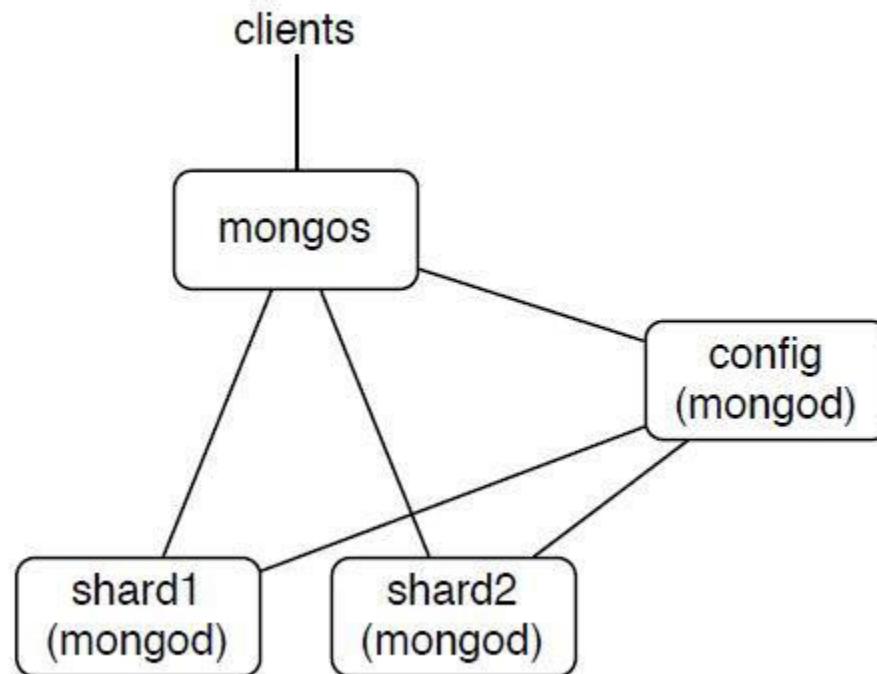
- config server (which is just a regular mongod) that keeps track of which server (mongo4 or mongo5) owns what values
- \$ mkdir ./mongoconfig
- \$ mongod --configsvr --dbpath
./mongoconfig --port 27016

- finally, we need to run fourth server called mongos, which is single point of entry for our clients – will connect to mongoconfig config server to keep track of sharding information stored there
- set it on port 27020 with chunkSize of 1
 - chunkSize is 1MB, which is smallest value
- just for our small dataset, so we can watch sharding take place

- \$ mongos --configdb localhost:27016 --chunkSize 1 --port 27020
- \$ mongo localhost:27020/admin
- > db.runCommand({ addshard : "localhost:27014" })
- { "shardAdded" : "shard0000", "ok" : 1 }
- > db.runCommand({ addshard : "localhost:27015" })
- { "shardAdded" : "shard0001", "ok" : 1 }

- > db.runCommand({ enablesharding : "test" })
- { "ok" : 1 }
- > db.runCommand({ shardcollection : "test.cities", key : {name : 1} })
- { "collectionsharded" : "test.cities", "ok" : 1 }

sharded cluster



GeoSpatial Queries

- core of geospatial secret lies in indexing
- special form of indexing geographic data called geohash that not only finds values of specific value or range quickly but finds nearby values quickly in ad hoc queries
- so to query, step 1 is to index data on location field; 2d index must be set on any two value fields, in our case hash (for example, { longitude: 1.48453, latitude:42.57205 }), but could easily have been an array ([1.48453, 42.57205])

- `db.cities.ensureIndex({ location : "2d" })`
- following will work only with nonsharded collections in current version of Mongo
- `> db.cities.find({ location : { $near : [45.52, -122.67] } }).limit(5)`
- should be patched in future versions for sharded collections

geoNear()

- > db.runCommand({geoNear : 'cities', near : [45.52, -122.67], num : 5, maxDistance : 1})
- {
- "ns" : "test.cities",
- "near" :
"100011000100010110100111000111110",
- "results" : [{
- "dis" : 0.007105400003747849,"obj" : {
- "_id" : ObjectId("4d81c216a5d037634ca98df6"),
- "name" : "Portland",

- `geoNear()` also helps with troubleshooting geospatial commands
- returns useful information such as distance from queried point, average and max distance of returned set, and index information

MongoDB's Strengths

- ability to handle huge amounts of data (and huge amounts of requests) by replication and horizontal scaling
- very flexible data model
- easy to use
 - similarity between Mongo commands and SQL database concepts
- mind share from former object-relational model (ORM) users

MongoDB's Weaknesses

- encourages denormalization of schemas (by not having any)
 - some developers find cold, hard constraints of relational database reassuring
- flexibility is generally not important if your data model is already fairly mature and locked down
- works best in large clusters, which can require some effort to design and manage

- excellent choice if you are currently using relational database to store your data through an ORM out of habit
- often recommend it to Model-View-Controller (MVC) developers, since they can then perform validations and field management through models at application layer

- adding new fields to document is as easy as adding new field to your data model, will happily accept the new terms
- much more natural answer to many common problem scopes for application-driven datasets than relational databases

Business Intelligence

Reporting Services

- SQL Server Reporting Services is one component of Microsoft's Business Intelligence platform

Business Intelligence (BI)

- refers to computer-based techniques used in spotting, digging, analyzing business data
- provide historical, current, and predictive views of business operations, common functions
 - reporting
 - online analytical processing, data mining
 - business performance management, benchmarking, predictive analytics

Business Intelligence

- aims to support better business decision-making
- can be called a decision support system (DSS)
- often BI applications use data gathered from a data warehouse or a data mart
 - however, not all data warehouses are used for business intelligence, nor do all business intelligence applications require a data warehouse

Life is filled with decisions

- Should I read these course notes slides or should I skip these
- first bit of business intelligence: Read this!

Data warehouse

- repository of an organization's electronically stored data
- designed to facilitate reporting and analysis
- definition focuses on data storage
- means to retrieve and analyze data, to extract, transform and load data, and to manage the data dictionary are also considered essential components of a data warehousing system

Extract, Transform, Load

- process in database usage and especially in data warehousing that involves
- Extracting data from outside sources
- Transforming it to fit operational needs
- Loading it into the end target (database or data warehouse)

problem
example

Report Classic Models

- company which buys collectable model cars, trains, trucks, buses, trains and ships directly from manufacturers and sells them to distributors across the globe

Offices

- Classic Models Inc. has 7 offices worldwide (San Francisco, Boston, NYC, Paris, Tokyo, Sydney, London)
- based on geography each office is assigned to a sales territory
 - APAC, NA, EMEA, JAPAN

Employees

- company has 23 employees
 - 6 Execs and 17 Sales Reps
- assigned to one of company's 7offices
- Sales Reps also assigned to a number of customers (distributors) in their respective geographies that they sell to
- Sales Rep reports to Sales Manager for his/her territory ... exceptions
- Execs don't work directly with customers

Customers

- Classic Models Inc. has 122 customers across the world

Products

- Classic Models Inc. sells 110 unique models which they purchase from 13 vendors
- for each product price at which product was purchased from vendor as well as Manufacturer Suggested Retail Price are provided
 - MSRP price is on average 45% (30% to 60%) above buyPrice

Product Lines

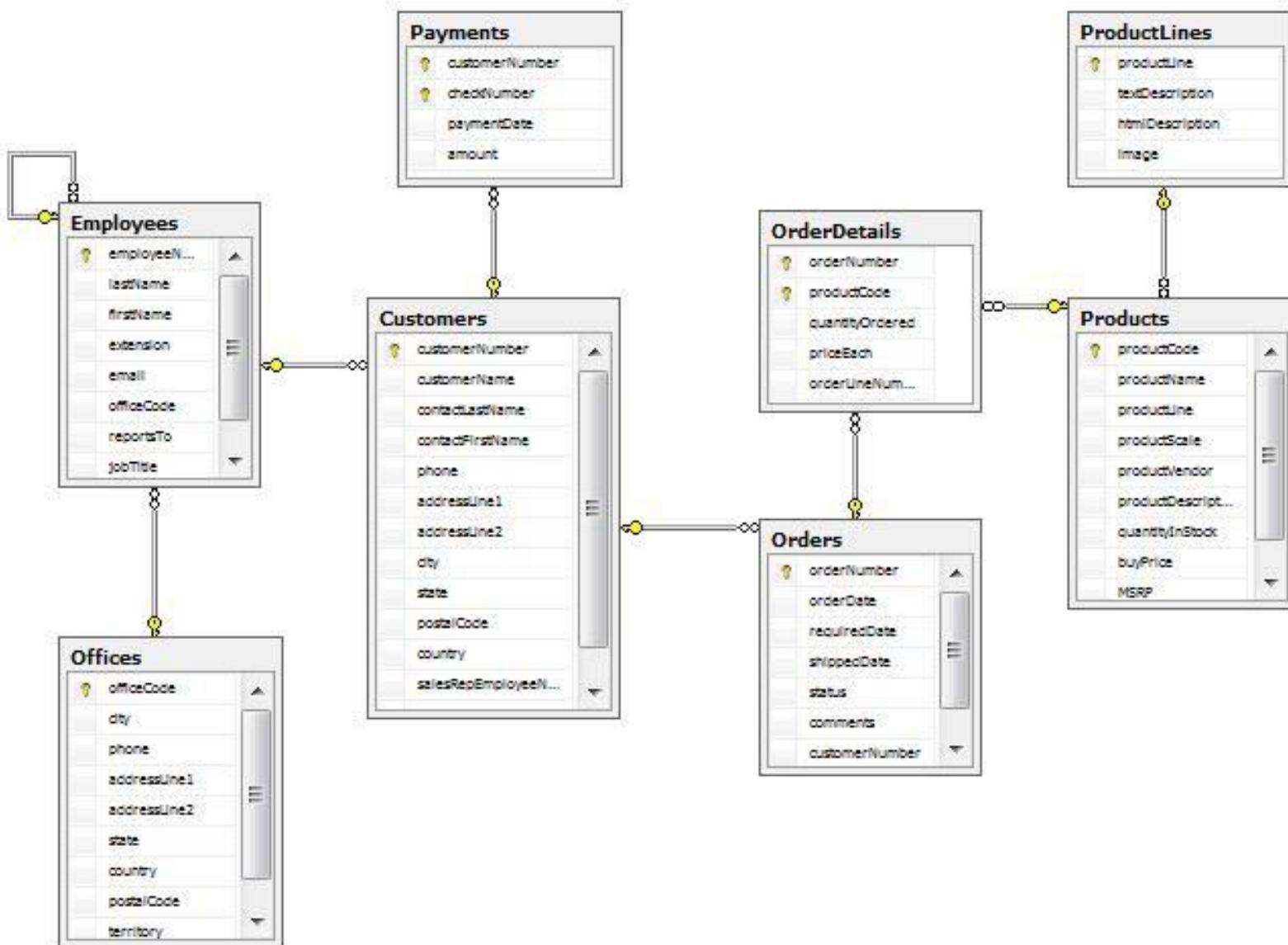
- models are classified as 7 distinct product lines
 - Classic Cars, Vintage Cars, Motorcycles, Trucks and Buses, Planes, Ships, Trains

Orders

- customers place their orders and expect to receive them approximately within 6 to 10 days
- once an order is placed it's assembled and shipped within 1 to 6 days (7-8 for Japan)
- can be in one of these states:
 - In Process, Shipped, Cancelled, Disputed, Resolved, On Hold

Order Details

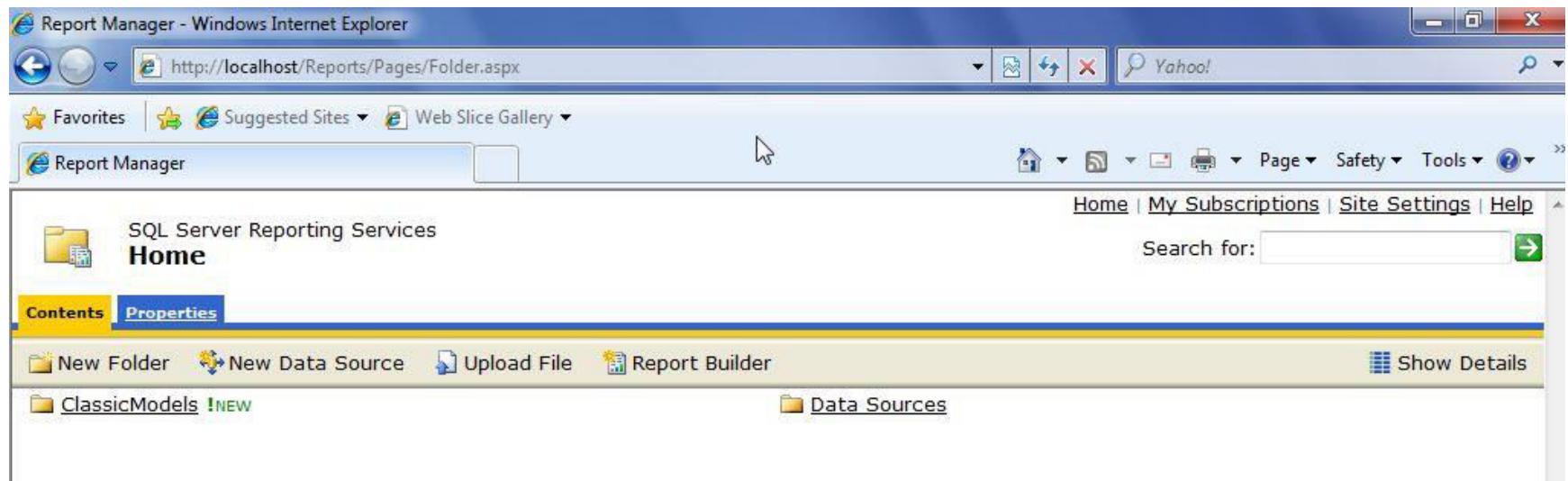
- each order contains an average of 9 unique products (order line items) with an average quantity of 35 per product
- reflects negotiated price per product
 - based on corresponding product's MSRP as well as quantity per product and maybe customer or other elements



Big Boss

- better management = business intelligence
- predict future based on past history
 - already stored in database
- what was the value added
 - by different dimensions

Report Manager



Dimensions

The screenshot shows a Microsoft Internet Explorer window titled "Report Manager - Windows Internet Explorer". The address bar displays the URL <http://localhost/Reports/Pages/Folder.aspx?ItemPath=%2fClassicModels&ViewMode=List>. The page content is the "Report Manager" interface for the "ClassicModels" folder under "SQL Server Reporting Services". The left sidebar shows a tree view with nodes: Home > **ClassicModels**, which contains sub-nodes: SalesEmployee !NEW, SalesGeographically !NEW, SalesProduct !NEW, SalesTime !NEW, and SalesVendor !NEW. The top navigation bar includes links for Home, My Subscriptions, Site Settings, Help, and a search bar. The bottom toolbar includes buttons for New Folder, New Data Source, Upload File, Report Builder, and Show Details.

Employee

The screenshot shows a Microsoft Internet Explorer window displaying a report from SQL Server Reporting Services. The title bar reads "Report Manager - Windows Internet Explorer" and the address bar shows the URL "http://localhost/Reports/Pages/Report.aspx?ItemPath=%2fClassicModels%2fSalesEmployee". The report itself is titled "Sales Employee" and displays a detailed table of sales data. The table has columns for Territory, Office, Employee, Customer, Order Date, Buy Price, Price / Item, and Quantity. The data is grouped by Territory and Office, with specific employees listed under each office. The report includes a header with navigation links like Home, My Subscriptions, Site Settings, and Help, and a search bar at the top right.

Territory	Office	Employee	Customer	Order Date	Buy Price	Price / Item	Quantity
APAC					19,857.3	32,865.27	12,878
EMEA					77,084.62	128,224.01	49,578
	4 Paris	Gerard Hernandez			51,952.28	86,624.44	33,881
		Loui Bondur			20,881.46	34,668.3	14,231
		Martin Gerard			9,543.25	16,225.21	6,180
			CAF Imports		6,291.76	10,582.34	4,180
			Corrida Auto Replicas, Ltd		745.7	1,249.62	468
				28 May 2003	1,738.3	3,064.57	1,161
				26 Jan 2004	961.59	1,623.71	611
				01 Nov 2004	315.97	623.43	241
					460.74	817.43	308
			Enaco Distributors		1,141.93	1,776.13	882
			Iberia Gift Imports, Corp.		744.26	1,298.02	589
			Vida Sport, Ltd		1,921.57	3,194	1,078
		Pamela Castillo			15,235.81	25,148.59	9,290

Employee - ?

- Employee
 - Customer
 - Date
- Employee
 - Date
 - Customer

Geographically

The screenshot shows a Microsoft Excel spreadsheet with data from the Northwind database. The data is organized into several columns: Territory, Country, Customer, Order Date, Buy Price, Price / Item, Quantity, Discount, and Value. The rows are grouped by Territory (APAC, EMEA, Japan, NA) and then by Country (Australia, New Zealand, Singapore). Within each country group, there are multiple rows for different customers, with their respective order dates, buy prices, and calculated values.

Territory	Country	Customer	Order Date	Buy Price	Price / Item	Quantity	Discount	Value
APAC				19,857.3	32,865.27	12,878	3,843.87	
	Australia			10,063.73	16,687.78	6,246	1,897.24	
		Anna's Decorations, Ltd		2,590.99	4,314.94	1,469	524.39	
			11 Sep 2003	828.76	1,374.9	430	164.05	
			04 Nov 2003	651.4	1,130.7	444	167.63	
			20 Jan 2005	558.84	898.11	256	91.84	
			09 Mar 2005	551.99	911.23	339	100.87	
		Australian Collectables, Ltd		1,069.46	1,816.84	705	221.35	
				3,149.21	5,159.19	1,926	564.63	
				900.36	1,528.57	545	152.3	
				2,353.71	3,868.24	1,601	434.57	
	New Zealand			7,845.78	13,003.07	5,396	1,605.81	
	Singapore			1,947.79	3,174.42	1,236	340.82	
EMEA				77,084.62	128,224.01	49,578	14,497.67	
Japan				7,651.64	12,635.69	4,923	1,405.37	
NA				58,916.55	98,220.45	38,137	10,769.56	

Products

Sales Product

Product Line	Product Name	Date	Buy Price	Price / Item	Quantity	Price	Value Added
Classic Cars			65,924.62	109,084.52	35,582	3,853,922.49	1,526,212.2
Motorcycles			18,254.99	31,348.93	12,778	1,121,426.12	469,255.3
Planes			16,675.4	26,989.94	11,872	954,637.54	365,960.71
	1900s Vintage Bi-Plane		959	1,726.39	940	58,434.07	26,239.07
	1900s Vintage Tri-Plane		1,014.44	1,896.02	1,009	68,276.35	31,720.28
		17 Feb 2003	36.23	71	26	1,846	904.02
		29 Apr 2003	36.23	71.73	29	2,080.17	1,029.5
		27 Jun 2003	36.23	61.58	46	2,832.68	1,166.1
		25 Aug 2003	36.23	71.73	33	2,367.09	1,171.5
		09 Oct 2003	36.23	66.65	33	2,199.45	1,003.86
		28 Oct 2003	36.23	68.1	48	3,268.8	1,529.76
		11 Nov 2003	36.23	66.65	27	1,799.55	821.34
		15 Nov 2003	36.23	64.48	33	2,127.84	932.25
		01 Dec 2003	36.23	70.28	39	2,740.92	1,327.95
		12 Jan 2004	36.23	68.1	40	2,724	1,274.8

Geography - Products

- report to illustrate the geography of sold products

Time

Sales by Time

Year	Quarter	Month	Order Date	Buy Price	Price / Item	Quantity	Price	Value Added
2003				57,567.79	95,687.83	36,439	3,317,348.39	1,320,622.94
2004				77,427.76	129,122.11	49,487	4,515,905.51	1,809,381.14
	Q1			13,341.98	22,274.89	8,694	799,579.31	319,596.74
		Q1		4,877.01	8,249.85	3,245	292,385.21	119,453.03
			02 Jan 2004	872.33	1,438.31	530	49,614.72	19,397.02
			09 Jan 2004	383.66	646.57	269	21,053.69	8,598.51
				85.68	129.2	39	5,038.8	1,697.28
				51.61	82.58	28	2,312.24	867.16
				64.58	97.4	20	1,948	656.4
				34.25	66.45	43	2,857.35	1,384.6
				26.3	56.55	36	2,035.8	1,089
				48.64	79.67	22	1,752.74	682.66
				39.83	90.52	33	2,987.16	1,672.77
				32.77	44.2	48	2,121.6	548.64
			12 Jan 2004	877.09	1,443.06	577	47,177.59	18,374.31
			15 Jan 2004	761.07	1,399.57	530	49,165.16	22,311.18

Time

- will always be a dimension in any business

Vendor

Sales Vendor

Vendor	Product Name	Buy Price	Price / Item	Quantity	Price	Value Added
Autoart Studio Design		11,169.21	20,988.33	7,702	736,928.03	344,926.37
Carousel DieCast Legends		11,666.07	18,734.9	8,735	667,190	251,702.34
Classic Metal Creations		15,191.08	25,957.87	9,678	934,554.42	384,438.06
	1928 British Royal Navy Airplane	1,868.72	2,746.46	972	94,885.37	30,014.09
	1938 Cadillac V-16 Presidential Limousine	577.08	1,127.18	955	38,449.09	18,766.54
	1949 Jaguar XK 120	1,181.25	2,018.21	949	76,670.02	31,829.77
	1952 Alpine Renault 1300	2,760.24	5,524.66	961	190,017.96	95,282.58
	1954 Greyhound Scenicruiser	727.44	1,364.12	955	46,519.05	21,708.15
	1956 Porsche 356A Coupe	2,654.1	3,463	1,052	134,240.71	30,829.11
	1957 Corvette Convertible	1,888.11	3,490.85	1,013	130,749.31	59,910.22
	1961 Chevrolet Impala	872.91	1,978.17	941	69,120.97	38,698.44
	1962 City of Detroit Streetcar	1,012.23	1,452.78	966	52,123.81	15,908.47
	1965 Aston Martin DB5	1,649	2,792.44	914	101,778.13	41,490.69
Exoto Designs		14,250.53	22,167.6	8,604	793,392.31	282,537.09
Gearbox Collectibles		14,165.46	23,984.34	8,352	828,013.76	338,223.61

Vendor

- based on this example and just this view it seems that ?

Vendor

- AutoArt Studio Design is the best vendor
 - cheap buy price, large value added, highest profitability
- Carousel Diecast Legends is the worst vendor
 - large buy price, small value added, lowest profitability

Dashboard Report

- report to senior management that provides at-a-glance perspective on current status of company in context of predetermined metrics
- depending on organization, those metrics may include cost, profitability, value added, time, requirements, risk, customer satisfaction, or other measures critical to management team
- provides management with quick understanding of current business posture, without detailed explanation of causes or solutions

Business Intelligence

Data Warehouse
Cub

Business intelligence

- Business intelligence is the delivery of accurate, useful information to the appropriate decision makers within the necessary timeframe to support effective decision making.

Transactional data

- information stored to track interactions, or business transactions, carried out by organization
- business transactions of an organization need to be tracked for that organization to operate
- organization needs to keep track of what it has done and what it needs to do

OnLine Transaction Processing

- when these transactions are stored on and managed by computers, we refer to OLTP
- OLTP systems record business interactions as they happen
- support day-to-day operation of organization
- optimized for efficiently processing and storing transactions - breaking data up into small chunks using rules of database normalization

Business intelligence measures

- are not designed to reflect events of one transaction, but to reflect net result of number of transactions over selected period of time
- are often aggregates of hundreds, thousands, or even millions of individual transactions
- designing system to provide these aggregates efficiently requires entirely different optimizations

- need to do is take information stored in OLTP systems and move it into different data store
- intermediate data store can then serve as source for our measure calculations
- when data is stored in this manner, it is referred to as *data mart*

Data Mart

- body of historical data in an electronic repository that does not participate in daily operations of organization
- used to create business intelligence
- usually applies to specific area of organization

Data Warehouse

- tend to be large, one-stop-shopping repositories where all historical data for organization would be stored
- data marts are smaller undertakings that focus on particular aspect of organization
- Business Intelligence
 - Data Warehouse, Data Mart, Cube

Extract, Transform, and Load (ETL) process

- extracts data from one or more OLTP systems, performs any required data consolidation and cleansing (removes inconsistencies and errors from transactional data so it has consistency necessary for use in data mart) to transform data into consistent format, and loads the cleansed data by inserting it into data mart

OnLine Analytical Processing

- type of system that would be tuned to needs of data analysts, online analytical processing, or OLAP, system
- enable users to quickly and easily retrieve information from data, usually data mart, for analysis
- OLAP systems present data using measures, dimensions, hierarchies, and cubes

Data Warehouse Design Cube

Data Mart Structure

- The data we use for business intelligence can be divided into four categories
 - measures
 - dimensions
 - attributes
 - hierarchies

Measure

- forms basis of everything we do with business intelligence - building blocks for effective decision making
- numeric quantity expressing some aspect of organization's performance
- information represented by this quantity is used to support or evaluate decision making and performance of organization
- can also be called a fact; tables that hold measure information known as *fact tables*

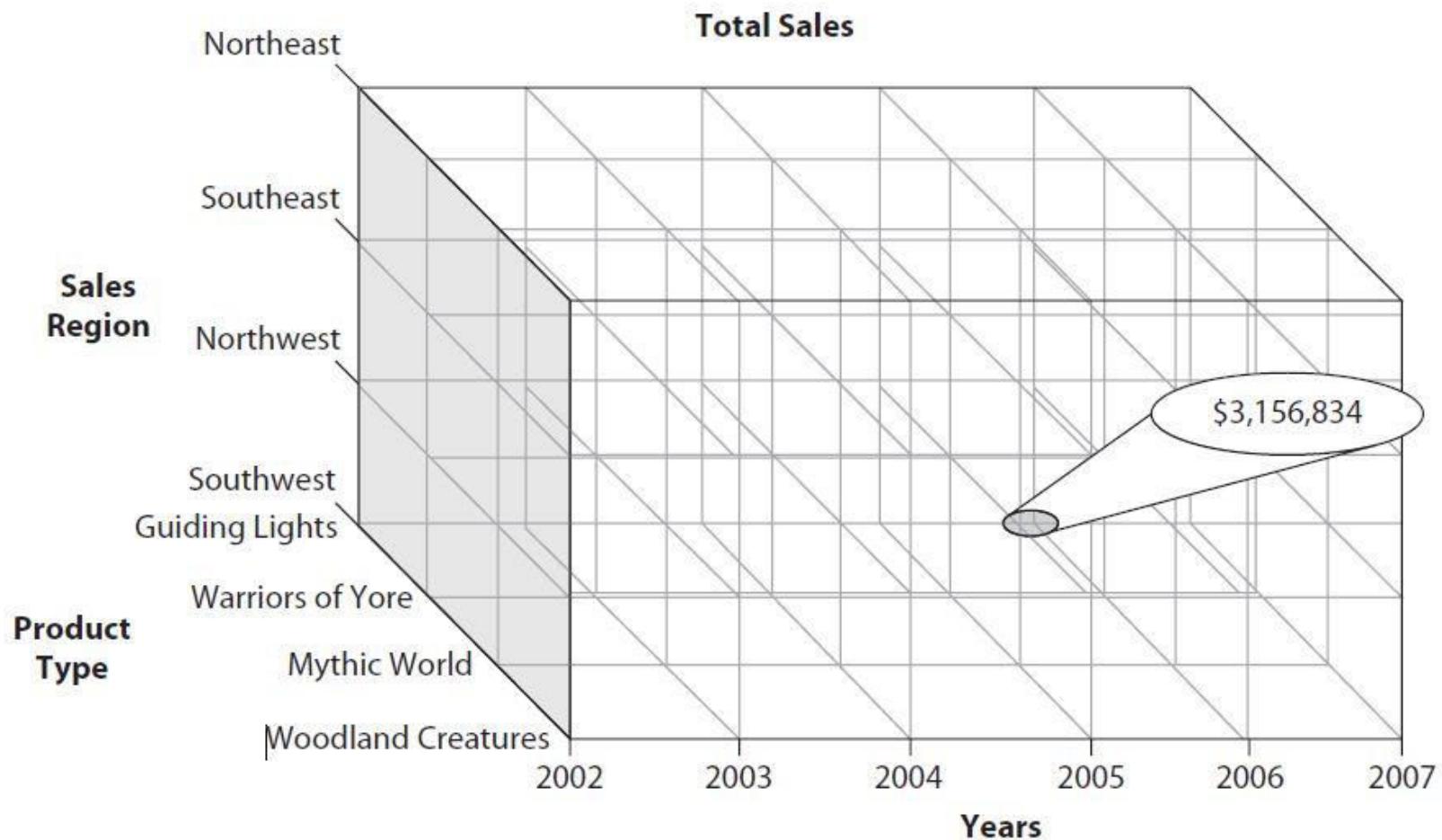
Dimension

- sales is an example of measure that is often required for effective decision making
- however, it is not often that decision makers want to see one single aggregate representing total sales for all products for all salespersons for entire lifetime of organization
- more likely to want to slice and dice this total into smaller parts

Dimension

- categorization used to spread out an aggregate measure to reveal its constituent parts
- used to facilitate this slicing and dicing

Cube



Cube

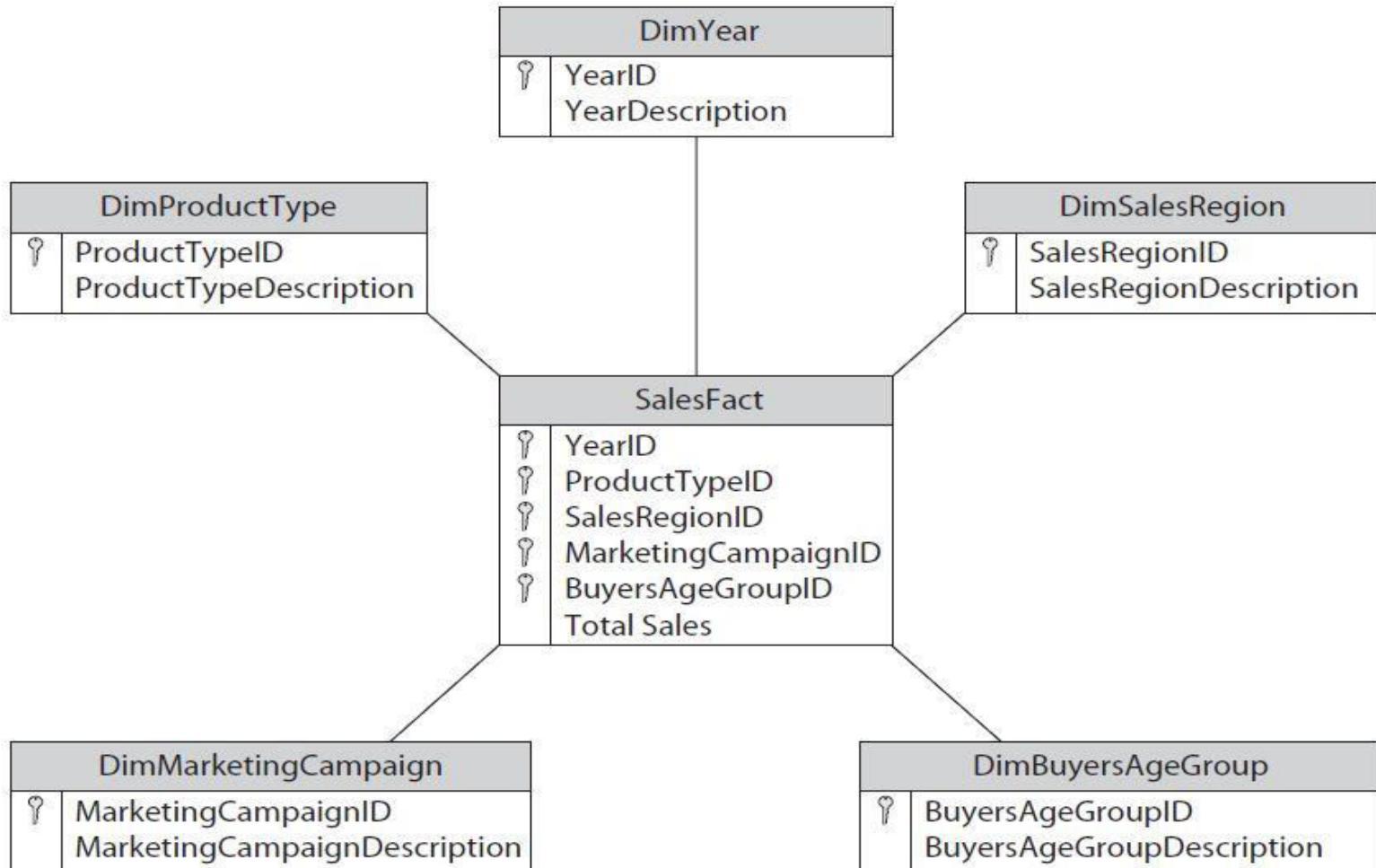
- can continue to spread out measure using additional dimensions, such as marketing campaign dimension and buyer's age
- becomes difficult to represent in illustration
- refer to these structures with four, five, or even more dimensions as cubes
- typical dimension: time, space, products

- measures and dimensions are stored in data mart in one of two layouts, or schemas
- star schema
- snowflake schema

Star Schema

- measures are stored in fact table
- dimensions are stored in dimension tables
- center of star is formed by fact table
- fact table has column for measure and column for each dimension containing foreign key for member of that dimension
- primary key for this table is created by concatenating all of foreign key fields
- fact table may contain multiple measures

Star Schema



Attributes

- additional information about dimension members
- information pertaining to dimension member that is not the unique identifier or description of member
- information about dimension that users likely want as part of their business intelligence output
- also used to store information that may be used to limit or filter records
- stored as additional columns in dimension tables

Hierarchies

- in many cases, dimension is part of larger structure with many levels
- structure known as hierarchy
- structure made up of two or more levels of related dimensions
- dimension at upper level of hierarchy completely contains one or more dimensions from next lower level

Hierarchies

- in star schema information about hierarchies is stored right in dimension tables
- primary key in each of dimension tables is at the lowest level of hierarchy
- fact table must be changed so its foreign keys point to lowest levels in hierarchies as well

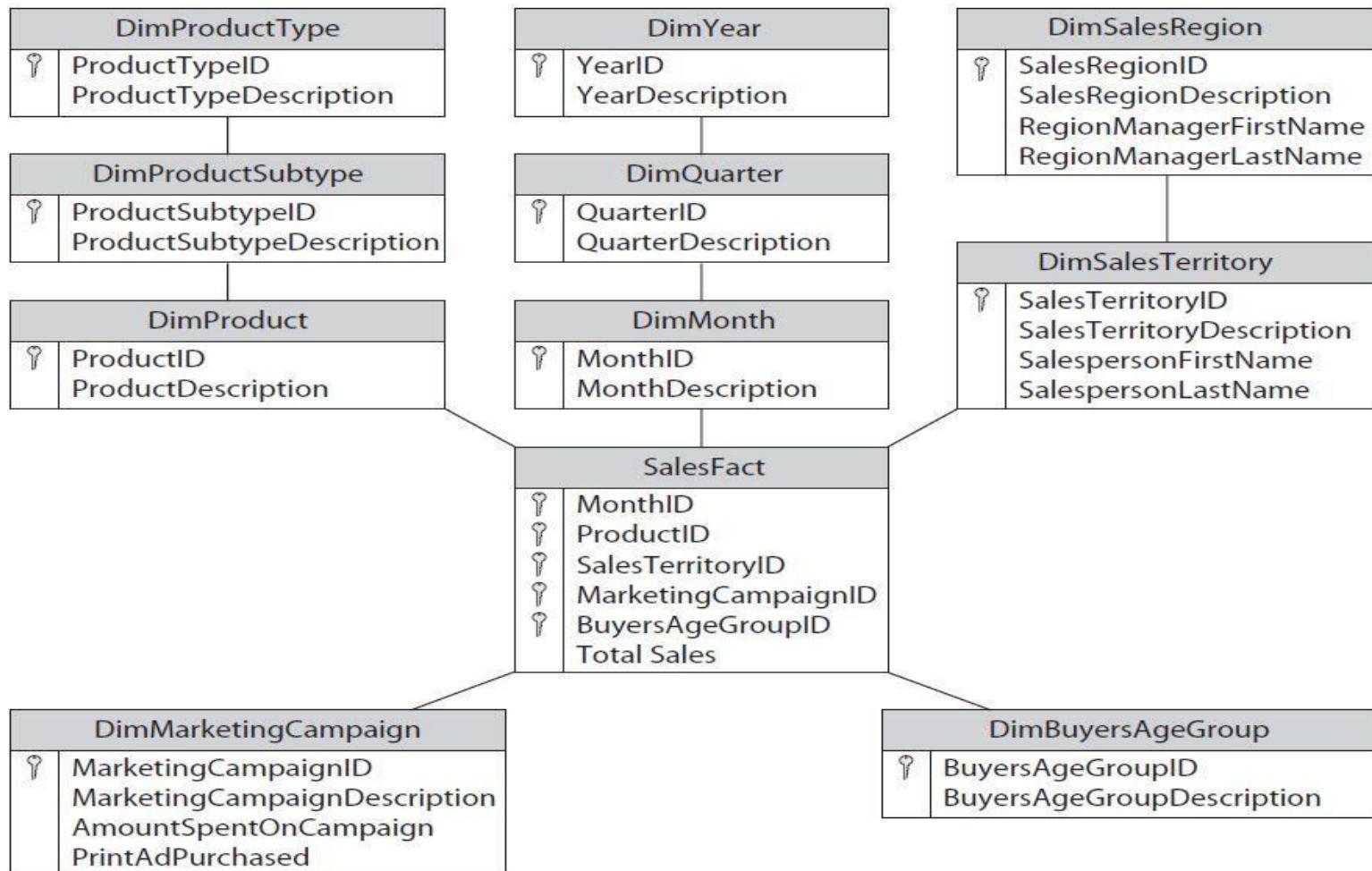
Hierarchies

- fact table will have one row (and one set of measures) for each unique combination of members at lowest level of all hierarchies
- measures for hierarchy levels above lowest level are not stored in data mart
- these measures have to be calculated by taking an aggregate of measures stored at lowest level
- for example, if user wants to see total sales for sales region, calculated by aggregating total sales for all of territories within region

Snowflake Schema

- represents hierarchies in a manner that is more familiar to those working with relational databases
- each level of hierarchy is stored in separate dimension table

Snowflake Schema



Snowflakes, Stars Analysis Services

- snowflake schema has all advantages of good relational design
 - does not result in duplicate data
- disadvantage of snowflake design is that it requires a number of table joins when aggregating measures at upper levels of hierarchy

Snowflakes, Stars Analysis Services

- in both snowflake and star schemas, we have to calculate aggregates on the fly when user wants to see data at any level above lowest level in each dimension
- in schema with large dimensions or with dimensions that have large number of members, can take significant amount of time

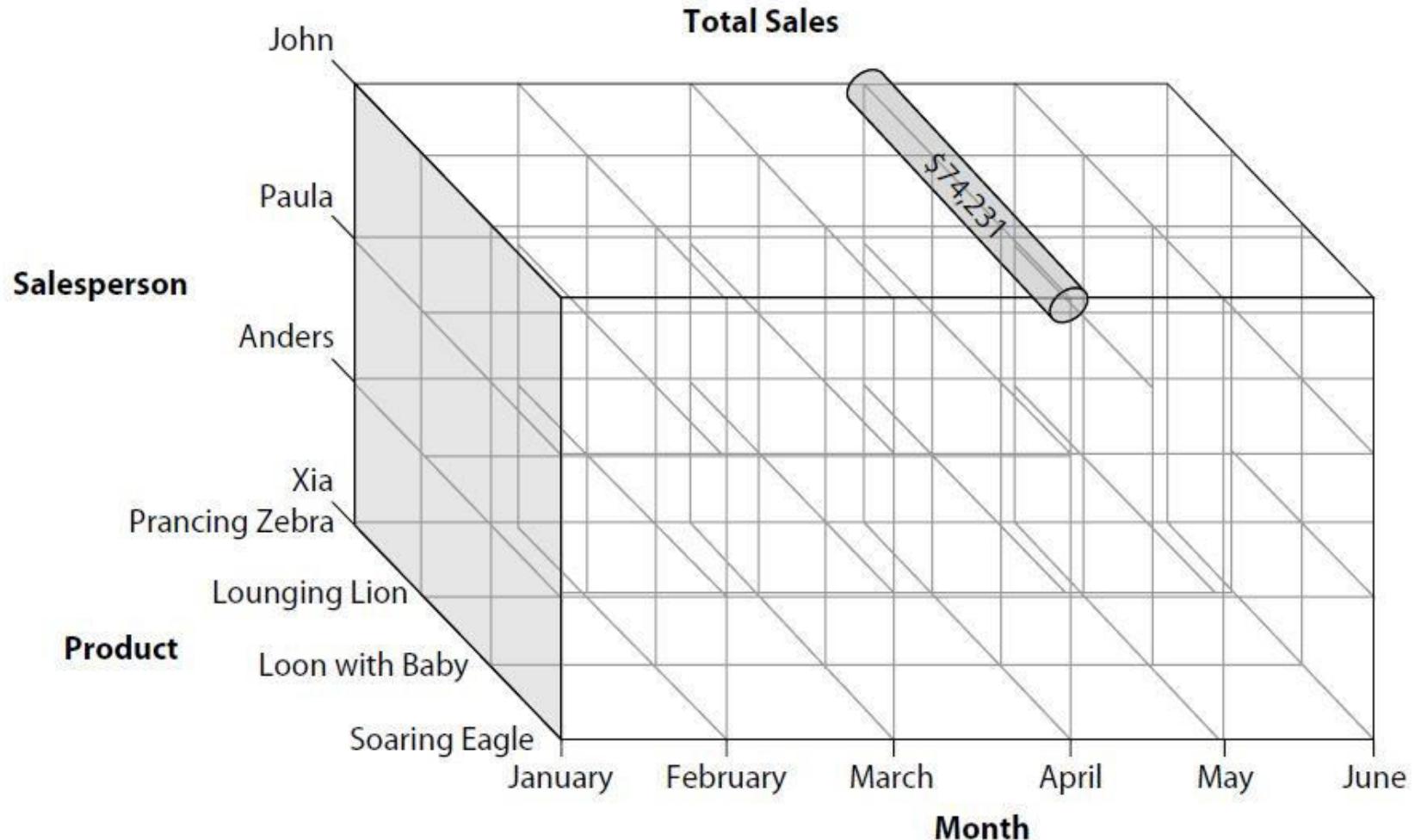
Cube

- structure that contains value for one or more measures for each unique combination of members of all its dimensions
 - detail, or leaf-level, values
- contains aggregated values formed by dimension hierarchies or when one or more of dimensions are left out

Aggregate

- value formed by combining values from given dimension or set of dimensions to create a single value
- often done by adding values together using sum aggregate, but other aggregation calculations can also be used

Sales cube with aggregation for John's total sales for April



Preprocessed Aggregates

- cubes require quite a few aggregate calculations as user navigates through
 - can slow down analysis
- to combat this, some or all of possible data aggregates in cubes are calculated ahead of time and stored within cube itself
- these stored values are known as *preprocessed aggregates*

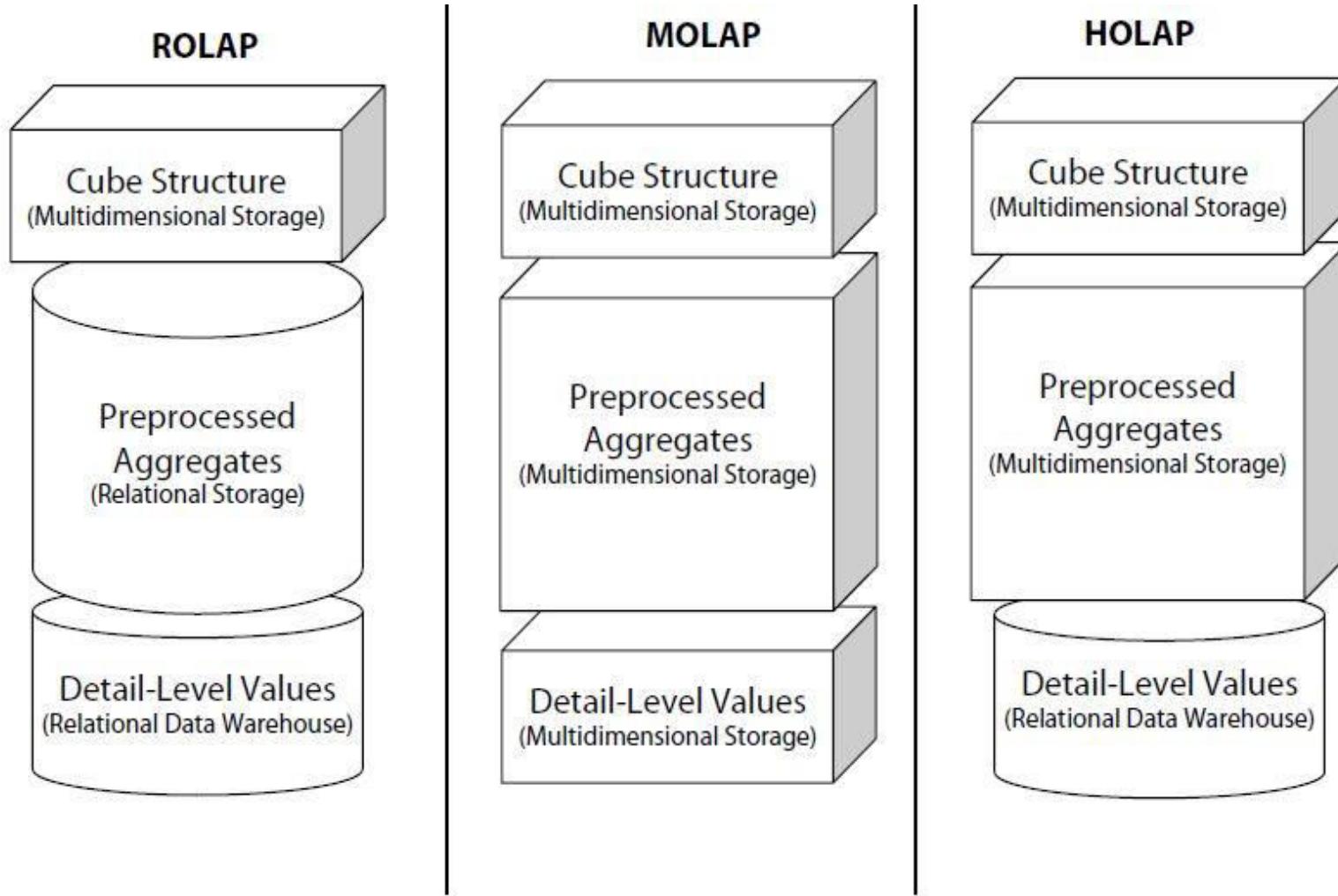
Multi-Dimensional Database

- structured around measures, dimensions, hierarchies, and cubes
 - rather than tables, rows, columns, relations
- natural way to store information used for *business intelligence*
- provides structure for storing pre-processed aggregates

Architecture

- key part of OLAP system is cube and preprocessed aggregates it contains
- OLAP systems typically use one of three different architectures for storing cube data

Architecture



Relational OLAP (ROLAP)

- stores cube structure in multidimensional database
- leaf-level measures are left in relational data mart that serves as source of cube
- preprocessed aggregates are also stored in relational database table
- when decision maker requests value of measure for set of dimension, ROLAP system first checks to determine whether dimension specify aggregate or leaf-level value
 - if aggregate is specified, value is selected from relational table
 - if leaf-level value is specified, value is selected from data mart

Relational OLAP (ROLAP)

- can store larger amounts of data than other OLAP architectures
- leaf-level values returned are always as up-to-date as data mart itself
 - does not add latency to leaf-level data
- disadvantage of ROLAP system is that retrieval of aggregate and leaf-level values is slower than other OLAP architectures

Multidimensional OLAP (MOLAP)

- also stores cube structure in multidimensional database
- both preprocessed aggregate values and copy of leaf-level values are placed in multidimensional database as well
- all data requests answered from multidimensional database, high responsive
- additional time required when loading MOLAP system - all leaf level data copied

Hybrid OLAP (HOLAP)

- Hybrid OLAP combines ROLAP and MOLAP storage

Unified Dimensional Model (UDM)

- Microsoft introduced new technology called *Unified Dimensional Model (UDM)*
- designed to provide all benefits of an OLAP system with multidimensional storage and preprocessed aggregates
- avoids number of drawbacks of more traditional OLAP systems

Unified Dimensional Model

- structure that sits over the top of data mart and looks exactly like an OLAP system to the end user - it does not require a data mart
- can build UDM over one or more OLTP systems
- can even mix data mart and OLTP system data in same UDM
- can even include data from databases from other vendors and Extensible Markup Language (XML)-formatted data

Unified Dimensional Model

- can define measures, dimensions, hierarchies, and cubes, either from star and snowflake schemas, or directly from relational database tables
- latter capability enables us to provide business intelligence without first having to build data mart

Data Sources

- UDM begins with one or more *data sources*
- stores information necessary to connect to database that provides data to UDM
- includes server name, database name, logon credentials, ...
- OLE DB providers are available for accessing different databases

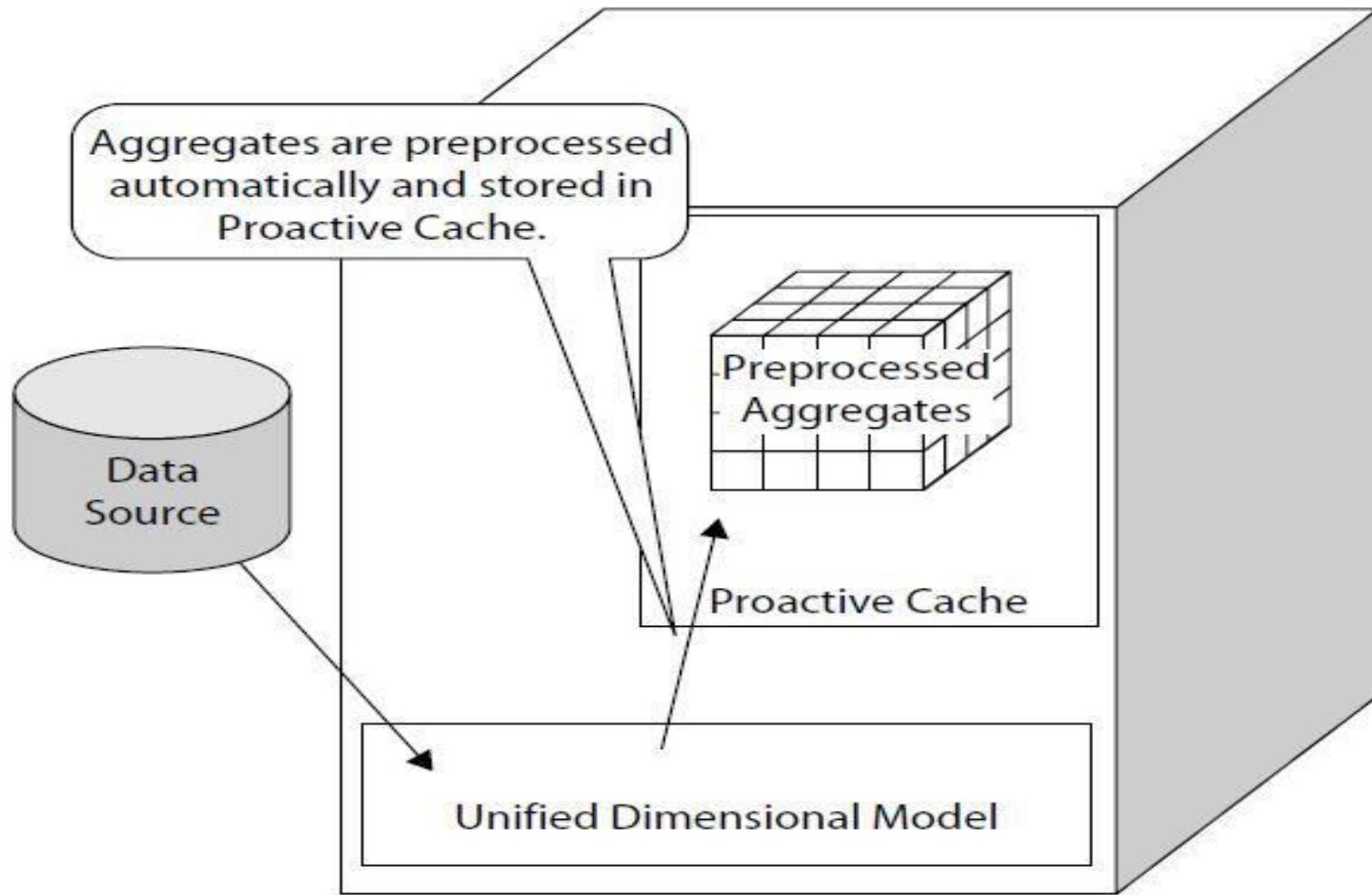
Data Views

- used to determine which tables and fields are utilized
- combine tables and fields from number of different data sources
- this multiple data source capability of data views is what puts “*Unified*” in *Unified Dimensional Model*

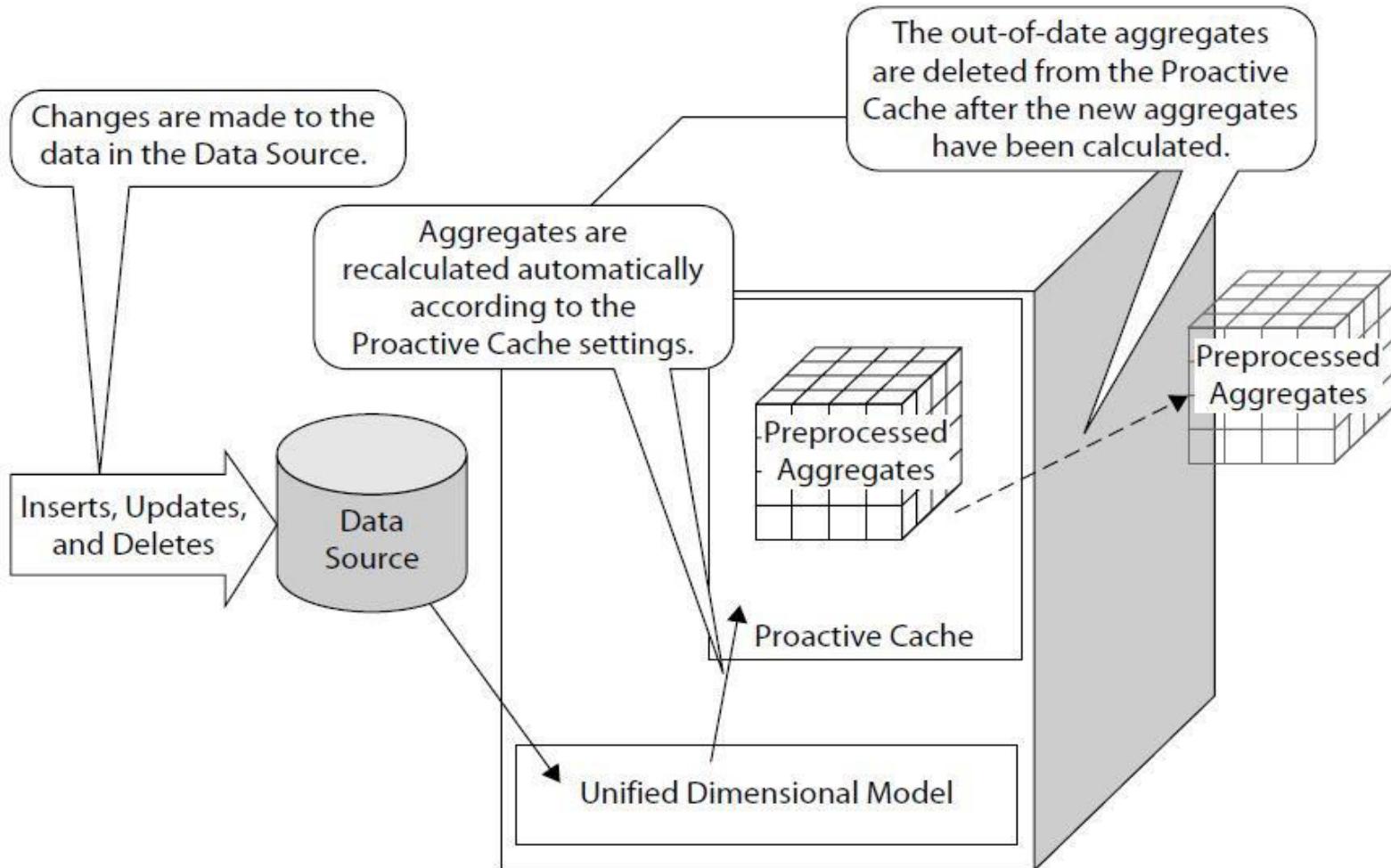
Proactive Caching

- to obtain same performance benefits of traditional OLAP systems, UDM uses *preprocessed aggregates*
- to facilitate high availability, these preprocessed aggregates are stored in *proactive cache*

Proactive Caching



Proactive cache deleted re-created in response to changes in data



XML Definitions

- definitions of all objects in UDM are stored as XML files
- act as source code for UDM

UDM Advantages

- OLAP built on transactional data
- extremely low latency
- ease of creation and maintenance

Design

- what are decision makers needs (analysis)
- what facts, figures, statistics, ... you need for effective decision making (measures, facts)
- how should this information be sliced and diced for analysis (dimensions)
- what additional information can aid in finding exactly what is needed (attributes)

Extract Transform Load (ETL)

Integration Services

Extract Transform Load (ETL)

- decision makers - data marts – must contain useful data - *Integration Services*
- enables us to change location of data by copying it from (OLTP) databases
- as it is being copied, we can also transform data from format required by OLTP systems to format required by OLAP systems

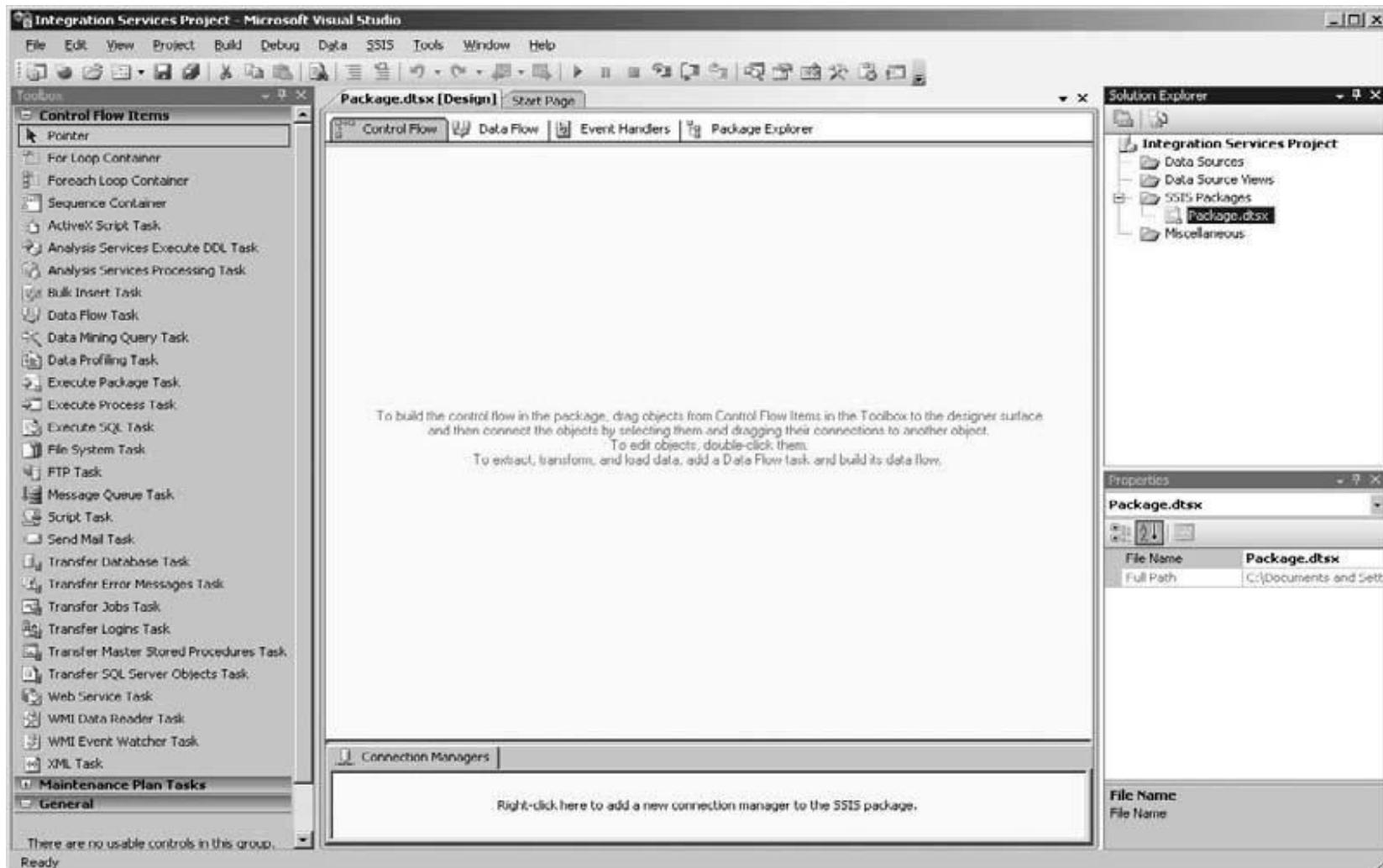
Integration Services

- create structures called packages, which are used to move data between systems
- packages contain data sources and data destinations
- Microsoft's approach to data transfer
- easy-to-use, flexible, capable, highly scalable Extract, Transform, Load ETL tool
- uses a drag-and-drop development style

Integration Services Package

- operation is defined by control flow
 - sequence of tasks that will be performed
- Integration Services packages are created using Integration Services project in Business Intelligence Development Studio

Integration Services Project

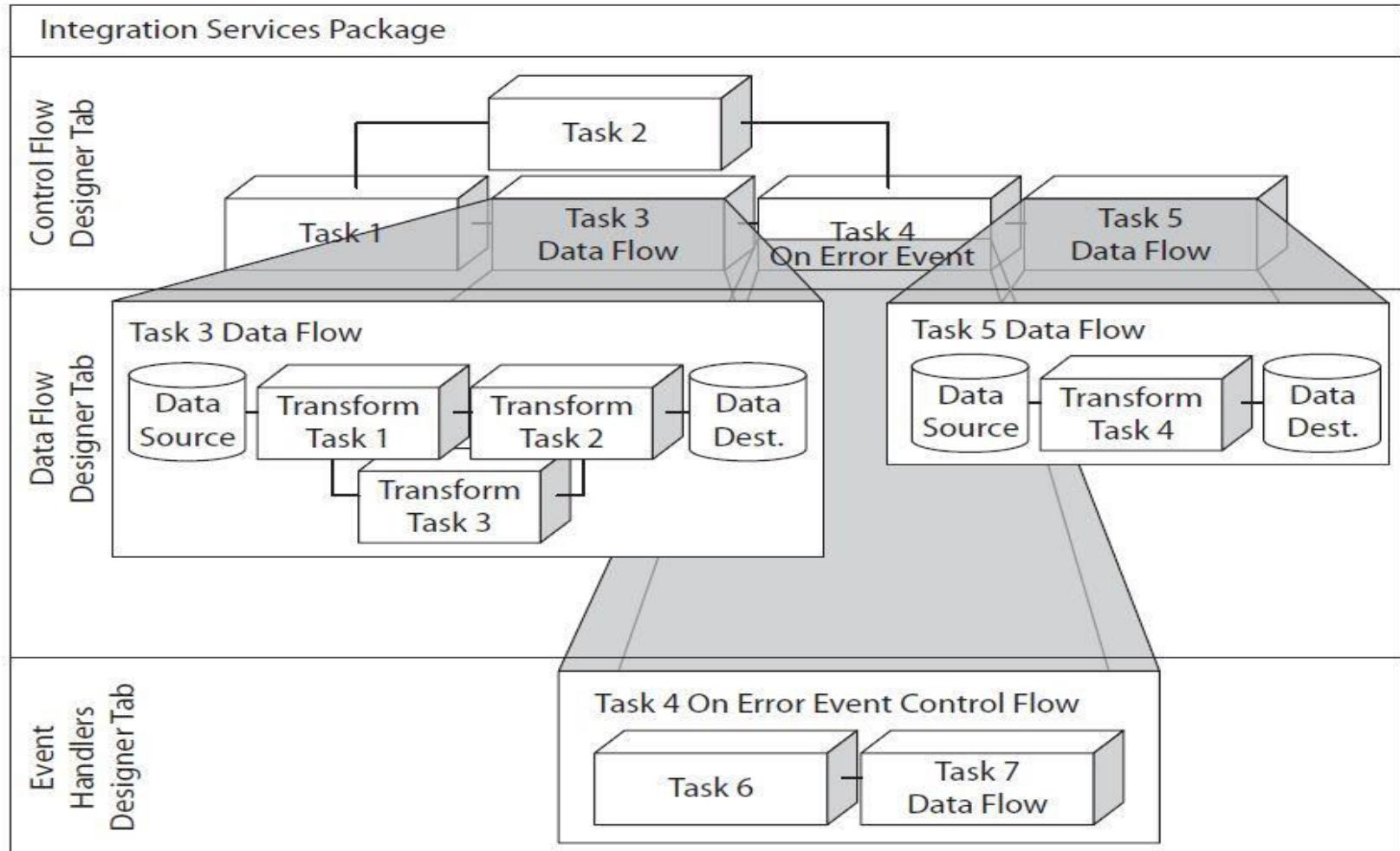


Integration Services Project design layout

- Designer Tab
 - Control Flow
 - Data flow
 - Event Handlers
 - Package Explorer
- Connections tray (special area at bottom of Designer window)
 - for defining Connection Managers

- event handler defined as control flow
- (however, event handler control flows are created on Event Handlers Designer tab rather than on Control Flow Designer tab)
- control flow
 - tasks
 - data source, transformation, data destination

Integration Services package structure



Control Flow Items

- For Loop Container, Foreach Loop Container
- Bulk Insert Task
- **Data Flow Task**
- Execute Package Task, Message Queue Task
- Send Mail Task
- Maintenance Plan Tasks, Back Up Database Task
- Execute SQL Server Agent Job Task
- Execute T-SQL Statement Task

Data Flow Task

- Toolbox is divided into three areas
 - data flow sources
 - data flow transformations
 - data flow destinations

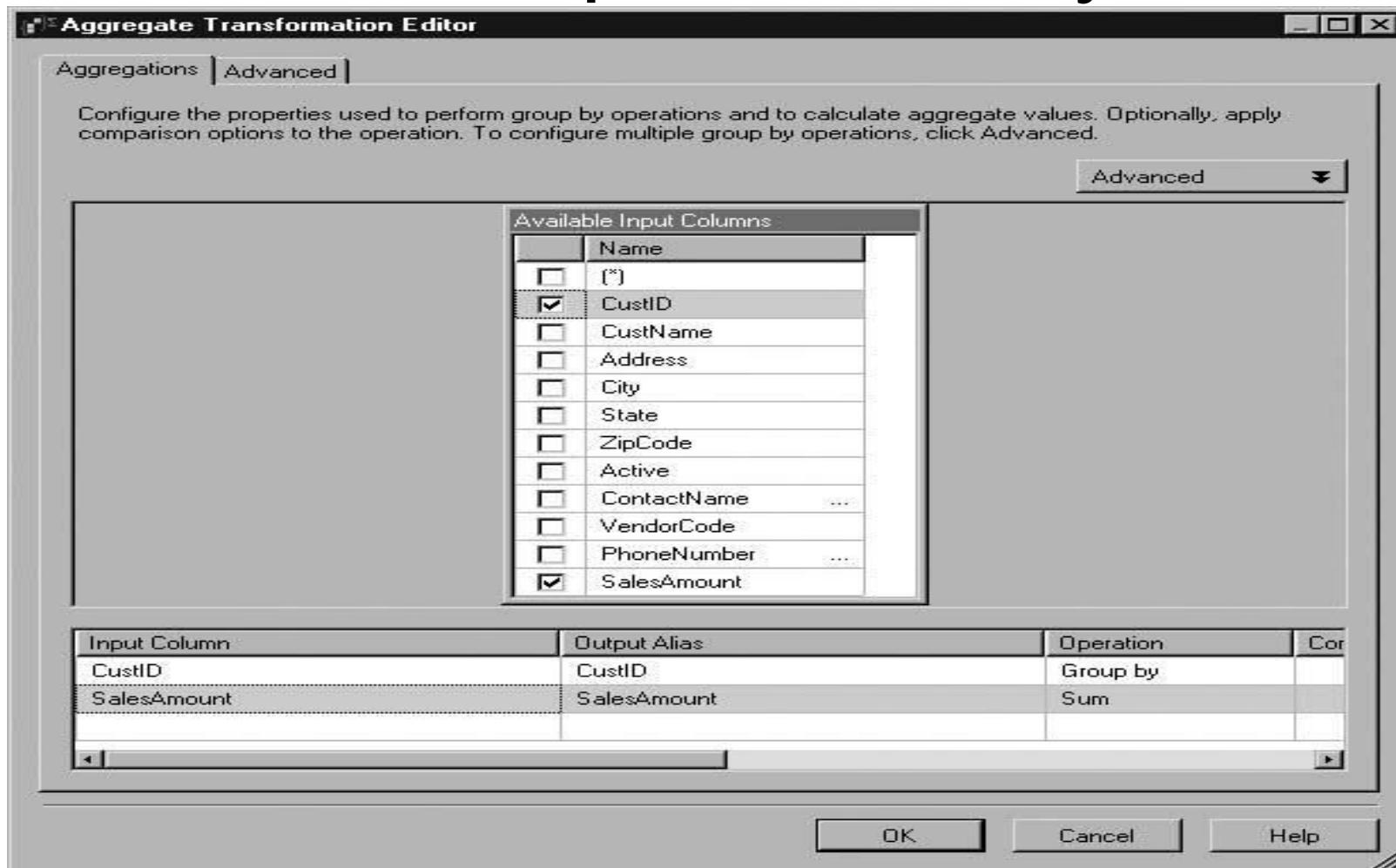
Data Flow Sources, Destinations

- ADO.NET Source
- Excel Source
- Flat File Source
- **OLE DB Source**
- Raw File Source
- XML Source

Data Flow Transformations - Aggregate

- select columns to participate in aggregation from Available Input Columns
- select Operation that is applied to each of selected columns
- column can either be used as group by or it can be aggregated
- Average
- Count; Count distinct
- Maximum; Minimum
- Sum

sum of SalesAmount for each customer represented by CustID



Data Flow Transformations - Audit

- lets us add columns that contain information about package execution to data flow
- audit columns can be stored in data destination and used to determine when package was run, where it was run, ...

Data Flow Transformations - Character Map

- enables us to modify contents of character-based columns
- modified column can be placed in place of original column, or it can be added as new column

Data Flow Transformations - Conditional Split

- enables us to split data flow into multiple outputs

Data Flow Transformations -

Copy Column

- lets us create new columns in data flow that are copies of existing columns

Data Flow Transformations - Data Conversion

- enables us to convert columns from one data type to another
- can either replace existing column or be added as new column

Data Flow Transformations - Derived Column

- enables us to create value derived from an expression
- expression can use values from variables and content of columns in data flow

Data Flow Transformations - Export Column

- lets us take content of text or image column and write it out to file
- pathname and filename are specified in different column in data flow
- content of text or image column can be written to different file for each row

Data Flow Transformations - Import Column

- lets us take content of set of files and insert it into text or image column in data flow
- pathname and filename are specified in different column in data flow
- content of different text or image file can be written to each row in data flow

Data Flow Transformations - Fuzzy Grouping

- enables us to find groups of rows in data flow based on non-exact matches
- often used to find possible duplicate rows based on names, addresses, ...
- for example, Ms. Kathy Jones, Ms. Kathryn Jones, and Ms. Cathy Jones may be three entries for same person
- transformation selects one of rows in group as best candidate for other rows to be combined into
- once groups and their model rows have been identified, we can use another transformation to combine any unique information from non-model rows into model row and delete non-model rows

Data Flow Transformations - Lookup

- works similarly to Fuzzy Lookup transformation
- difference is the Lookup transformation requires exact matches, rather than using similarity scores

Data Flow Transformations - Merge

- merges two data flows together
- to work properly, both input data flows must be sorted using same sort order

Data Flow Transformations – Merge Join

- enables us to merge two data flows together by executing an inner join, left outer join, or full outer join
- as with Merge transformation, both of input data flows must be sorted

Data Flow Transformations – Multicast

- lets us take single data flow and use it as input to several data flow transformations or data flow destinations

Data Flow Transformations – OLE DB Command

- enables us to execute SQL statement for each row in data flow
- question marks can be used to create parameterized query

Data Flow Transformations – Pivot

- enables us to take normalized data and change it into less normalized structure
- taking content of one or more columns in input data flow and using it as column names in output data flow
- data in these newly created columns is calculated by taking an aggregate of contents of another column from input data flow
 - number of rows from input may define single row in output

Data Flow Transformations – Unpivot

- enables us to take a denormalized data flow and turn it into normalized data

Data Flow Transformations – Row Count

- lets us determine number of rows in data flow
- count is then stored in package variable, can then be used in expressions to modify control flow or data flow in package

Data Flow Transformations – Script

- lets us create .NET code for execution as part of our data flow
- it can be used as data source, data destination, or data transformation

Data Flow Transformations – Slowly Changing Dimension

- enables us to use data flow to update information in slowly changing dimension of data mart

Data Flow Transformations – Sort

- enables us to sort rows in data flow

Data Flow Transformations – Term Extraction

- lets us extract list of words and phrases from column containing free form text
- identifies recurring nouns phrases in text, along with score showing frequency of occurrence for each word or phrase
- information can then be used to help discover content of unstructured, textual data

Data Flow Transformations – Term Lookup

- functions almost identically to Term Extraction transformation
- difference is Term Lookup starts with table of terms to look for in free form text,
 - whereas Term Extraction creates its own list on-the-fly

Data Flow Transformations – Union

- let us merge several data flows into single data flow
- any number of data flows can be unioned together
- only limitation is they all must be able to contribute fields to all of columns defined in output

Procedural Logic in Structured Query Language

PROBLEMS

DataBase Design course notes 08
Procedural Extension to SQL

1. Hotel

2. Genealogy:

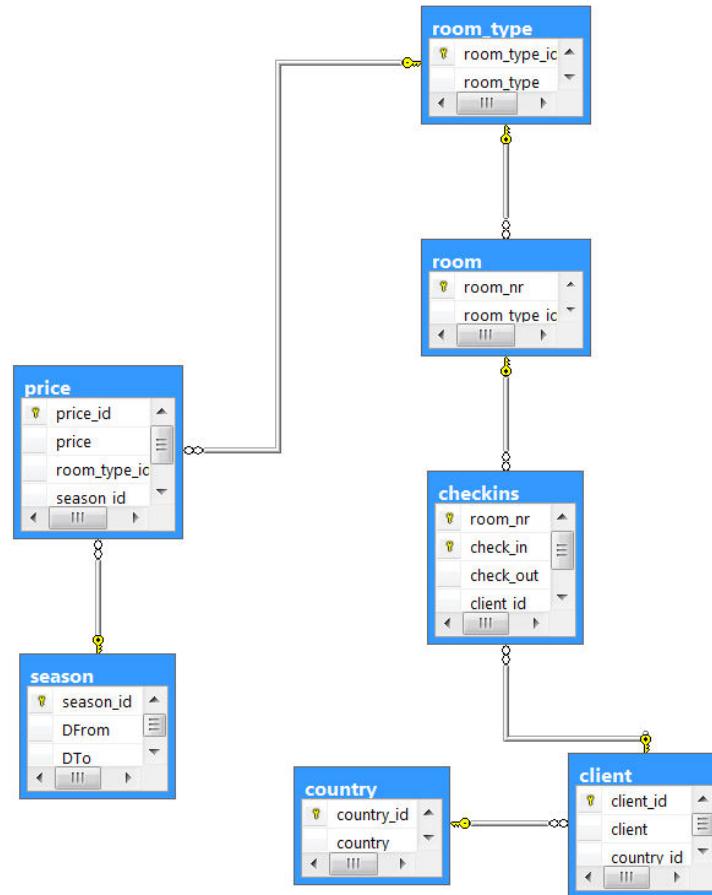
1. Math Genealogy

2. Queen

Hotel

- price depend on room type and season
 - season – begin date, end date
- tourist stay
 - check in date
 - check out date
 - Room
- *How much do they have to pay?*

DataBase Diagram



DataBase Design course notes 08
Procedural Extension to SQL

Room

- CREATE TABLE [dbo].[room] (
- [room_nr] [char](2) NOT NULL,
- [room_type_id] [int] NOT NULL,
- CONSTRAINT [PK_room] PRIMARY KEY CLUSTERED
- (
- [room_nr] ASC
-)) ON [PRIMARY]
- ALTER TABLE [dbo].[room] WITH CHECK ADD
CONSTRAINT [FK_room_room_type] FOREIGN
KEY([room_type_id]) REFERENCES [dbo].[room_type]
([room_type_id])

Room Type

- CREATE TABLE [dbo].[room_type] (
- [room_type_id] [int] IDENTITY(1,1) NOT NULL,
- [room_type] [varchar](50) NOT NULL,
- CONSTRAINT [PK_room_type] PRIMARY KEY
CLUSTERED
- (
- [room_type_id] ASC
-)) ON [PRIMARY]
- Room_Type – Alternate Key, Unique Index, Not NULL

Season

- CREATE TABLE [dbo].[season] ([season_id] [int] IDENTITY(1,1) NOT NULL, [DFrom] [date] NOT NULL, [DTo] [date] NULL,)
- CONSTRAINT [PK_season] PRIMARY KEY CLUSTERED ([season_id] ASC) ON [PRIMARY]
- ALTER TABLE [dbo].[season] WITH CHECK ADD CONSTRAINT [CK_season] CHECK (([DTo] IS NULL OR [DTo]>[DFrom]))

Price

depend on Season & Room Type

- CREATE TABLE [dbo].[price] (
- [price_id] [int] IDENTITY(1,1) NOT NULL,
- [price] [money] NOT NULL,
- [room_type_id] [int] NOT NULL,
- [season_id] [int] NOT NULL,
- CONSTRAINT [PK_price] PRIMARY KEY
CLUSTERED
- (
- [price_id] ASC
-)) ON [PRIMARY]

Price

depend on Season & Room Type

- ALTER TABLE [dbo].[price] WITH CHECK ADD CONSTRAINT [FK_price_room_type] FOREIGN KEY([room_type_id])
- ALTER TABLE [dbo].[price] WITH CHECK ADD CONSTRAINT [FK_price_season] FOREIGN KEY([season_id])
- ALTER TABLE [dbo].[price] WITH CHECK ADD CONSTRAINT [CK_price] CHECK (([price]>(0)))

View Room Price

- SELECT room_type.room_type, season.DFrom, season.DTo, price.price FROM
- room_type INNER JOIN
- price ON room_type.room_type_id = price.room_type_id INNER JOIN
- season ON price.season_id = season.season_id

View Room Price

room_type	DFrom	DTO	price
• single	2015-01-01	2015-02-27	100
• double	2015-01-01	2015-02-27	150
• single	2015-02-27	2015-03-15	120
• double	2015-02-27	2015-03-15	200
• single	2015-03-15	2015-04-10	100
• double	2015-03-15	2015-04-10	150
• single	2015-04-10	2015-04-15	150
• double	2015-04-10	2015-04-15	220

View Room Price

room_type	DFrom	DTO	price
single	2015-04-10	2015-04-15	150.00
double	2015-04-10	2015-04-15	220.00
single	2015-04-15	2015-04-30	110.00
double	2015-04-15	2015-04-30	160.00
single	2015-04-30	2015-05-05	150.00
double	2015-04-30	2015-05-05	250.00
single	2015-05-05	NULL	120.00
double	2015-05-05	NULL	170.00

Client

- CREATE TABLE [dbo].[client] (
- [client_id] [int] IDENTITY(1,1) NOT NULL,
- [client] [varchar](50) NOT NULL,
- [country_id] [int] NULL,
- CONSTRAINT [PK_client] PRIMARY KEY CLUSTERED
- (
- [client_id] ASC
-)) ON [PRIMARY]
- ALTER TABLE [dbo].[client] WITH CHECK ADD
CONSTRAINT [FK_client_country] FOREIGN
KEY([country_id])
- REFERENCES [dbo].[country] ([country_id])

Check Ins

- CREATE TABLE [dbo].[checkins] (
- [room_nr] [char](2) NOT NULL,
- [check_in] [date] NOT NULL,
- [check_out] [date] NULL,
- [client_id] [int] NOT NULL,
- CONSTRAINT [PK_checkins] PRIMARY KEY CLUSTERED
- ([room_nr] ASC, [check_in] ASC
-)) ON [PRIMARY]

Check Ins

- ALTER TABLE [dbo].[checkins] WITH CHECK ADD CONSTRAINT [FK_checkins_client] FOREIGN KEY([client_id]) REFERENCES [dbo].[client] ([client_id])
- ALTER TABLE [dbo].[checkins] WITH CHECK ADD CONSTRAINT [FK_checkins_room] FOREIGN KEY([room_nr]) REFERENCES [dbo].[room] ([room_nr])
- ALTER TABLE [dbo].[checkins] WITH CHECK ADD CONSTRAINT [CK_checkins] CHECK (([check_out] IS NULL OR [check_out]>[check_in]))

SQL Solution

- CREATE TABLE [dbo].[days](
 - [day] [datetime] NOT NULL,
- CONSTRAINT [PK_days] PRIMARY KEY CLUSTERED
 - ([day] ASC)) ON [PRIMARY]
- SELECT * FROM [hotel].[dbo].[days]
- 2000-01-01,
- 2000-01-02 , 2000-01-03, 2000-01-04, ... ,
- 2179-06-05

View CheckIns Day Price

- SELECT client.client, checkins.room_nr,
checkins.check_in, checkins.check_out, price.price,
days.day FROM
- checkins INNER JOIN
- room ON checkins.room_nr = room.room_nr INNER
JOIN
- room_type ON room.room_type_id =
room_type.room_type_id INNER JOIN
- client ON checkins.client_id = client.client_id INNER
JOIN

View CheckIns Day Price

- ...
- price ON room_type.room_type_id = price.room_type_id
INNER JOIN
- season ON price.season_id = season.season_id INNER JOIN
- days ON season.DFrom < days.day AND (days.day <= season.DTo OR
 season.DTo IS NULL) AND
checkins.check_in < days.day AND (days.day <= checkins.check_out OR
 checkins.check_out IS NULL)

```
SELECT * FROM VCheckInsDayPrice  
ORDER BY Client, Check_in, Day
```

• Traian Basescu	13	2015-01-11	2015-01-13	150.00	2015-01-12
• Traian Basescu	13	2015-01-11	2015-01-13	150.00	2015-01-13
• Angelina Jolie	16	2015-01-11	2015-01-16	150.00	2015-01-12
• Angelina Jolie	16	2015-01-11	2015-01-16	150.00	2015-01-13
• Angelina Jolie	16	2015-01-11	2015-01-16	150.00	2015-01-14
• Angelina Jolie	16	2015-01-11	2015-01-16	150.00	2015-01-15
• Angelina Jolie	16	2015-01-11	2015-01-16	150.00	2015-01-16
• Angelina Jolie	10	2015-04-01	2015-04-16	100.00	2015-04-02
• Angelina Jolie	10	2015-04-01	2015-04-16	100.00	2015-04-03
• Angelina Jolie	10	2015-04-01	2015-04-16	100.00	2015-04-04
• ..					
• Angelina Jolie	10	2015-04-01	2015-04-16	150.00	2015-04-15
• Angelina Jolie	10	2015-04-01	2015-04-16	110.00	2015-04-16

**SELECT * FROM VCheckInsDayPrice
ORDER BY Client, Check_in, Day**

• Emil Constantinescu	11	2015-02-02 2015-03-23 150.00	2015-02-03
• Emil Constantinescu	11	2015-02-02 2015-03-23 150.00	2015-02-04
• Emil Constantinescu	11	2015-02-02 2015-03-23 150.00	2015-02-05
• ...			
• Emil Constantinescu	11	2015-02-02 2015-03-23 150.00	2015-03-22
• Emil Constantinescu	11	2015-02-02 2015-03-23 150.00	2015-03-23
• Ion Iliescu	12	2015-01-11 2015-01-13 150.00	2015-01-12
• Ion Iliescu	12	2015-01-11 2015-01-13 150.00	2015-01-13
• Ion Iliescu	1	2015-04-01 2015-04-02 100.00	2015-04-02
• Ion Iliescu	1	2015-04-04 2015-05-05 100.00	2015-04-05
• Ion Iliescu	1	2015-04-04 2015-05-05 100.00	2015-04-06
• ..			
• Ion Iliescu	1	2015-04-04 2015-05-05 150.00	2015-05-04
• Ion Iliescu	1	2015-04-04 2015-05-05 150.00	2015-05-05

View Invoice Declarative

- SELECT client, room_nr, check_in, check_out, DATEDIFF(day, check_in, check_out) AS Days, SUM(price) AS Price
- FROM VCheckInsDayPrice
- GROUP BY client, room_nr, check_in, check_out, DATEDIFF(day, check_in, check_out)

View Invoice Declarative

clientroom_nr	check_in	check_out	Days	Price	
Angelina Jolie	10	2015-04-01	2015-04-16	15	1760.00
Angelina Jolie	16	2015-01-11	2015-01-16	5	750.00
Emil Constantinescu	11	2015-02-02	2015-03-23	49	8150.00
Ion Iliescu	1	2015-04-01	2015-04-02	1	100.00
Ion Iliescu	1	2015-04-04	2015-05-05	31	3750.00
Ion Iliescu	12	2015-01-11	2015-01-13	2	300.00
Traian Basescu	13	2015-01-11	2015-01-13	2	300.00

Procedural SQL Solution

- CREATE Scalar-valued Function
- Price per Day
- Price per Stay

FUNCTION PricePerDay

- (@PRoom_Type_Id int, @PDay DATE) RETURNS money
- DECLARE @ResultVar money;
- SELECT @ResultVar = price FROM price INNER JOIN room_type ON room_type.room_type_id = price.room_type_id INNER JOIN season ON season.season_id = price.season_id
- WHERE room_type.room_type_id = @PRoom_Type_Id AND season.DFrom < @Pday AND (@PDay <= season.DTo OR season.DTo IS NULL)
- RETURN @ResultVar;

FUNCTION PricePerStay

- (@PRoom CHAR(2), @PCheck_In DATE,
@PCheck_Out DATE) RETURNS MONEY
- DECLARE @PRoom_Type_Id INT;
- DECLARE @PDay DATE;
- DECLARE @ResultVar MONEY = 0;
- SELECT @PRoom_Type_Id = room_type_id FROM
room WHERE room.room_nr = @PRoom
- SET @PDay = DATEADD(day,1,@PCheck_In);
- WHILE @PDay <= @PCheck_Out

FUNCTION PricePerStay

- WHILE @PDay <= @PCheck_Out
- BEGIN
 - SET @ResultVar = @ResultVar +
dbo.PricePerDay(@PRoom_Type_Id, @PDay);
 - SET @PDay = DATEADD(day,1,@PDay);
- END
- RETURN @ResultVar

View Invoice Procedural

- ```
SELECT client.client, checkins.room_nr,
checkins.check_in, checkins.check_out,
DATEDIFF(day, checkins.check_in,
checkins.check_out) AS Days,
PricePerStay(checkins.room_nr,
checkins.check_in, checkins.check_out) AS
Price FROM
```
- checkins INNER JOIN
- client ON checkins.client\_id = client.client\_id

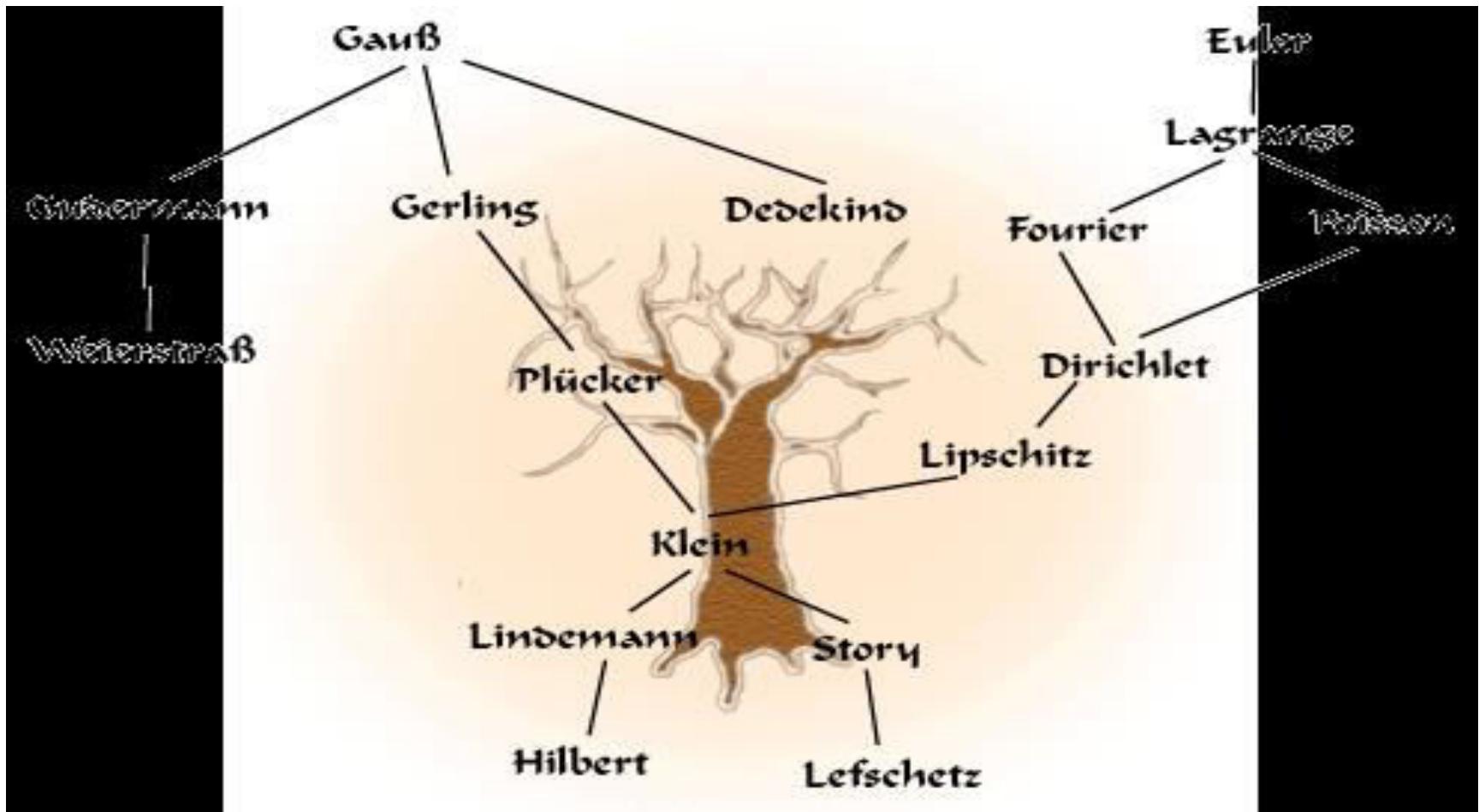
# View Invoice Procedural

|   | client room_nr      | check_in | check_out  | Days       | Price         |    |
|---|---------------------|----------|------------|------------|---------------|----|
| • | Angelina Jolie      | 10       | 2015-04-01 | 2015-04-16 | 15<br>1760.00 | 15 |
| • | Angelina Jolie      | 16       | 2015-01-11 | 2015-01-16 | 5<br>750.00   | 5  |
| • | Emil Constantinescu | 11       | 2015-02-02 | 2015-03-23 | 49<br>8150.00 | 49 |
| • | Ion Iliescu         | 1        | 2015-04-01 | 2015-04-02 | 1<br>100.00   | 1  |
| • | Ion Iliescu         | 1        | 2015-04-04 | 2015-05-05 | 31<br>3750.00 | 31 |
| • | Ion Iliescu         | 12       | 2015-01-11 | 2015-01-13 | 2<br>300.00   | 2  |
| • | Traian Basescu      | 13       | 2015-01-11 | 2015-01-13 | 2<br>300.00   | 2  |

# Genealogy

- The Mathematics Genealogy Project
- <http://genealogy.math.ndsu.nodak.edu/>

# Lab. Task – Math. Genealogy

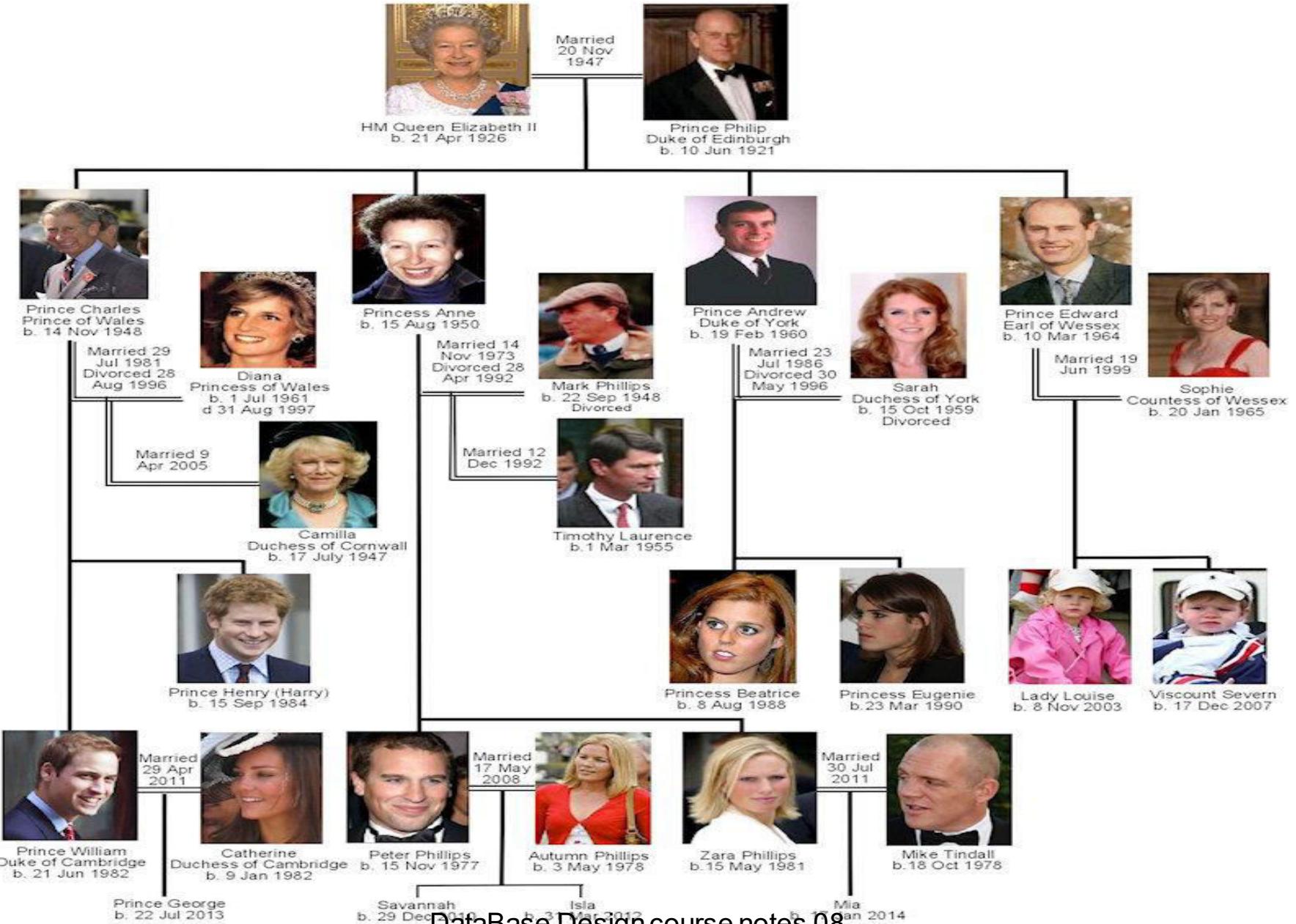


# Lab. Task – Math. Genealogy

- Design a DataBase
- Find a way to Count the number of descendants of a node
- Procedure / Function which make use of a **Cursor**
- Trigger ?
- $\text{Descendants}(\text{ Euler }) = 10$

# Genealogy

- **British Royal Family**
- <http://www.britroyals.com/>

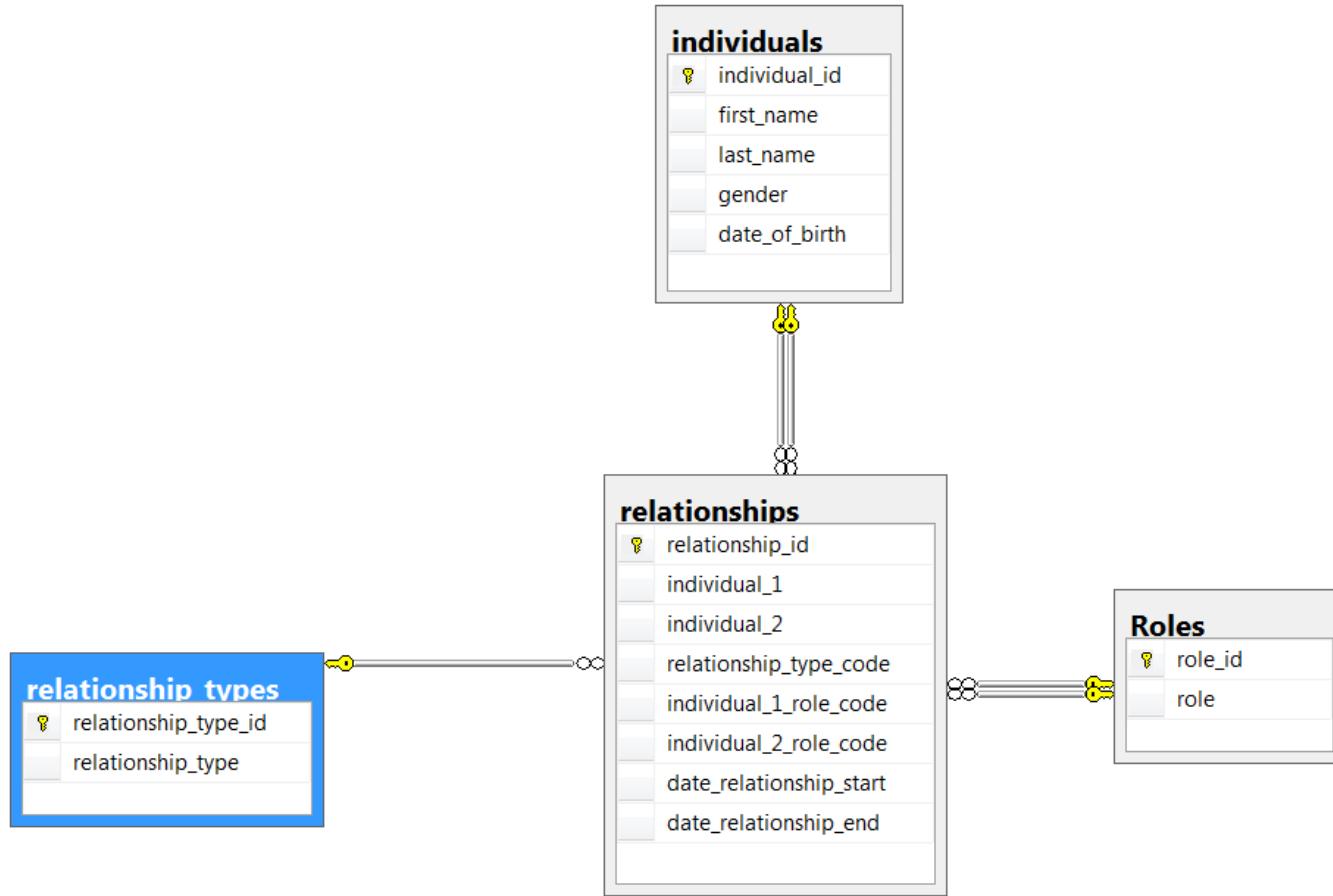


# Lab. Task – Queen Genealogy

- Design a DataBase
- Find a way to Count the number of descendants of a node
- Procedure / Function which make use of a **Cursor**
- Trigger ?
- Descendants(His Majesty Queen Elizabeth) = 10

- databaseanswers.org
- data\_models/genealogy/

# DataBase Diagram



**DataBase Design course notes 08**  
**Procedural Extension to SQL**

# Report

# Report

- document that can present numbers, text, and/or graphical information
- present information in coherent manner to business people
  - enable to understand what is happening in some aspect of business
  - enable to make decisions based on this information
- data sitting in organization's databases and transform it into information and eventually into kind of knowledge
  1. identify data
  2. create report

# Business Intelligence

- solutions integrate information from multiple data sources and realign data into structures that are ideal for reporting

# Microsoft Report Builder - report authoring tool

- **design surface** is the user interface used to create reports for Reporting Services
- primary design interface provided by Microsoft for creating reports is Visual Studio - hosts Report Designer
- allows user to select data source, build query dataset, layout report elements, preview, publish to report server
- saves developers from Structured Query Language (SQL), eXtensible Markup Language (XML), MultiDimensional eXpressions (MDX) if you're accessing data cubes, OLAP data stores

- publish to report server
- embedded in programs

# Report Builder



Untitled - Microsoft Report Builder

File Home Insert View

Run Paste Views Clipboard Font Paragraph Border Number Layout

Report Data

New Edit... Click to add title

Built-in Fields Parameters Images Data Sources Datasets

[&ExecutionTime]

No current report server. Connect

100%

A screenshot of the Microsoft Report Builder application window. The title bar says "Untitled - Microsoft Report Builder". The ribbon menu has tabs for File, Home, Insert, and View. The Home tab is selected, showing icons for Run, Paste, Views, Clipboard, Font, Paragraph, Border, Number, and Layout. On the left, there's a "Report Data" pane with a tree view showing "Built-in Fields", "Parameters", "Images", "Data Sources", and "Datasets". The main workspace has a placeholder text "Click to add title" in a large font. At the bottom of the workspace, there's a footer placeholder with the expression "[&ExecutionTime]". The bottom of the window shows "Row Groups" and "Column Groups" sections, and the status bar at the bottom says "No current report server. Connect" and "100%".

php

- koolreport.com
- reportico.org

# java – free open-source Business Intelligence reports

- [jaspersoft.com](http://jaspersoft.com)
  - [community.jaspersoft.com/](http://community.jaspersoft.com/)
- [eclipse.org/birt/](http://eclipse.org/birt/)
- [pentaho.com](http://pentaho.com) (hitachi vantara)
  - [community.hitachivantara.com/s/pentaho](http://community.hitachivantara.com/s/pentaho)
  - <https://sourceforge.net/projects/pentaho/>

# commercial

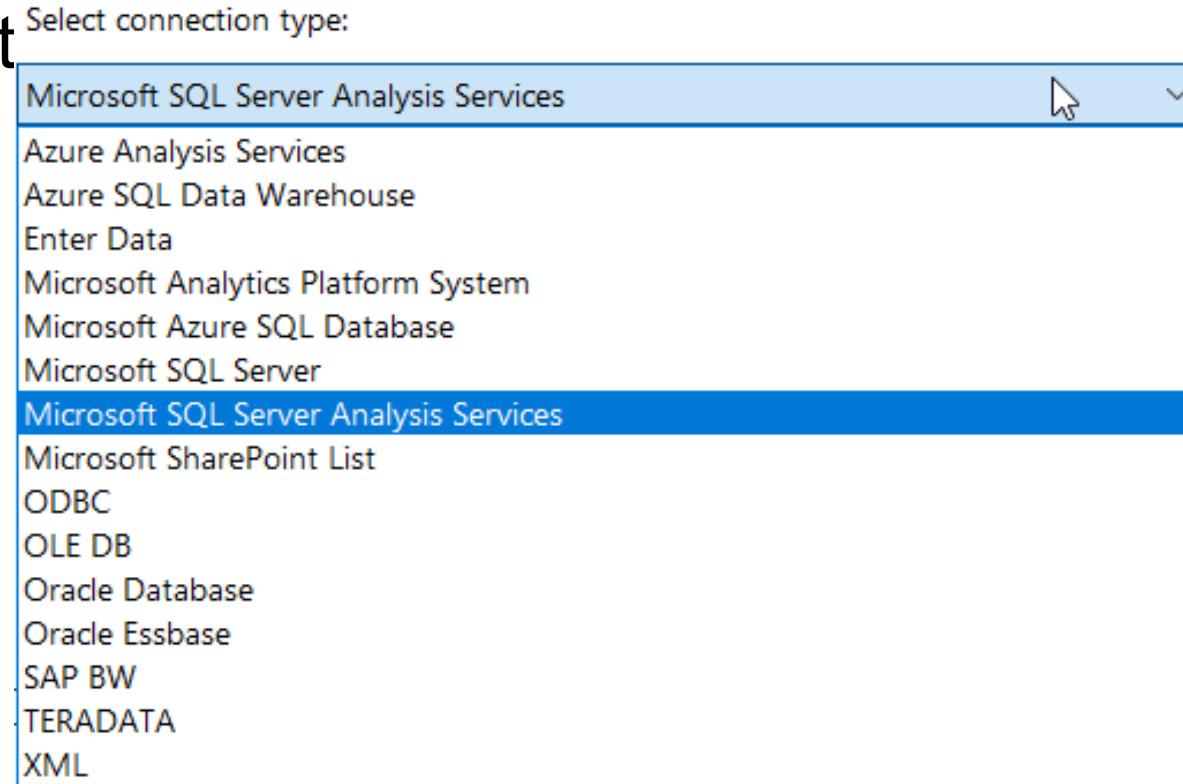
- tableau.com
- qlik.com
- ...

... cloud ...

- knowage-suite.com spagobi.org
- powerbi.microsoft.com

# Connect to data source

- Reporting Services allows access to any data source with ODBC (Open Database Connectivity) driver: allows multiple data sources t



# Create layout

- report items in toolbox of report controls available for building reports
- independent items not associated with particular data source: Textbox, Line, Rectangle, and Image report items
- controls to organize data for presentation *data regions*: List, Table, Matrix, Chart, and Subreport controls

# Plan Report

- format do you want the report to appear in
- how to deliver report

# report definition (.rdl) files

- create and modify paginated report definition (.rdl) files
- provides complete description of data source connections, queries used to retrieve data, expressions, parameters, images, text boxes, tables, other design-time elements that you include in report
- rendered at run time as processed report
- written in XML that conforms to XML grammar called Report Definition Language (RDL)

# Report Parts: data region, ..., map

- **data region** is object in report that displays data from report dataset
- report data can be displayed as numbers and text in table, matrix, or list; graphic in chart; and against geographic background in map
- tables, matrices, and lists are all based on tablix data region, which expands as needed to display all data from dataset
- **tablix** data region supports multiple row and column groups and both static and dynamic rows and columns
- **chart** displays multiple series and category groups in variety of formats
- **map** displays spatial data as map elements that can vary in appearance based on aggregated data from a dataset

# Nested data region; Linked data region

- can nest data regions within other data regions
- can link more than one data region to same dataset to provide different views of same data; for example, can show same data in table and in chart
- can author the report to provide interactive sort buttons on table

# tablix

- collectively, tables, matrices, and lists are referred to as tablix data regions
- templates are built on tablix data region, which is flexible grid that can display data in cells
  - in table and matrix templates, cells are organized into rows and columns, they have tabular layout and their data comes from single dataset built on single data source
- difference between tables and matrices is that tables can include only row groups, where matrices have row groups and column groups
- lists are different, support free-layout that and can include multiple peer tables or matrices, each using data from different dataset

| Sales Territory | Full Name             | 2003         | 2004         |
|-----------------|-----------------------|--------------|--------------|
| Northwest       | Pamela O Anzman-Woyle | \$900365.56  | \$1656492.56 |
|                 | David R Campbell      | \$1377431.33 | \$1930685.56 |
|                 | Tara A. Menna-Arryn   |              |              |
|                 | Total                 |              |              |

Table Report

| Sales by Area and Year |                | North America |             | 2003      | 2004         | Total        |
|------------------------|----------------|---------------|-------------|-----------|--------------|--------------|
|                        |                | US            | CA          |           |              |              |
| Clothing               | Caps           | \$9795.76     | \$2801.06   | \$9002.44 | \$1656492.56 | \$1656492.56 |
|                        | Tights         | \$51391.51    | \$8233.90   | \$5962    |              |              |
| Components             | Chains         | \$4972.00     | \$793.83    | \$152     |              |              |
|                        | Cranksets      | \$105100.64   | \$18515.00  | \$12365   |              |              |
|                        | Touring Frames | \$767130.33   | \$196141.85 | \$60345   |              |              |

Matrix Report

List Report

### Congratulations!

Michael G Blythe 275

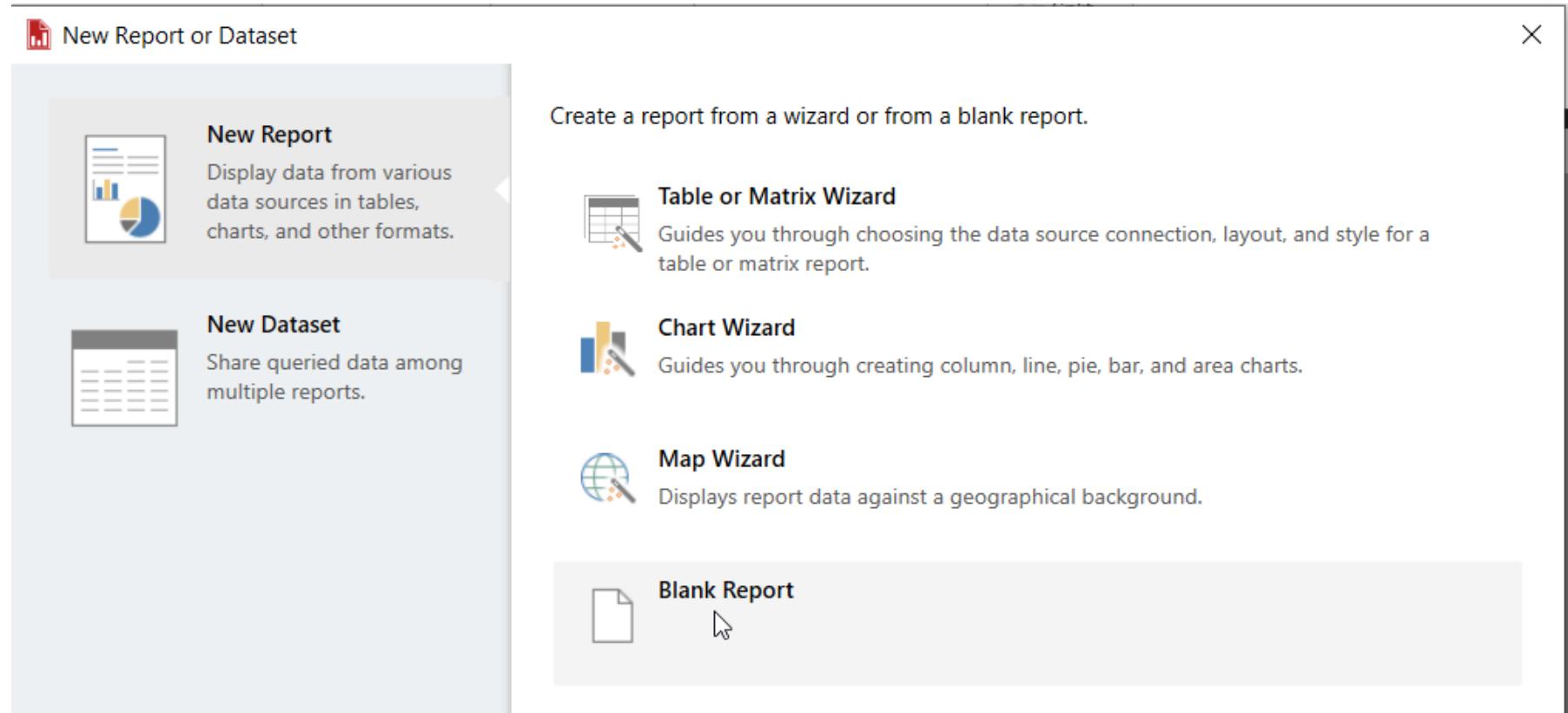
Hello Michael,  
Congratulations on an  
outstanding year!  
Your sales of \$1951005.00  
makes you the TOP  
Adventure Works sales  
person in 2004.  
Dot & Caley





# Create Basic Reports – with Report Builder

- open Report Builder
- File – New



# *dataset*

- represents report data that is returned from running SQL query on *data source*
- depends on data connection that contains information about data source; data itself is not included in report definition
- contains query command, field collection, parameters, filters, and data options that include case sensitivity and collation
- there are two types of datasets

1. Shared datasets - published on report server and can be used by multiple reports; must be based on shared data source; can be cached and scheduled by creating cache refresh plan
  2. Embedded datasets - defined in and used by single report
- recommend that you use shared datasets as much as possible; optimize query or cache query results to benefit report performance; make data access easier to manage, and help to keep reports and datasets more secure and more performant

# Add Data Source

Microsoft SQL Server



Azure Analysis Services

Azure SQL Data Warehouse

Enter Data

Microsoft Analytics Platform System

Microsoft Azure SQL Database

Microsoft SQL Server

Microsoft SQL Server Analysis Services

Microsoft SharePoint List

ODBC

OLE DB

Oracle Database

Oracle Essbase

SAP BW

TERADATA

XML

or

## Data Source Properties

X

General

Credentials

Change name, type, and connection options.

Name:

DataSource1

- Use a shared connection or report model
- Use a connection embedded in my report

Select connection type:

Microsoft SQL Server

Connection string:

Click here to type or paste a connection string

Build...



Test Connection

- Use single transaction when processing the queries

## Connection Properties

?

X

Data source:

Microsoft SQL Server (SqlClient)

Change...

Server name:

LAP-E570

Refresh

Log on to the server

 Use Windows Authentication Use SQL Server Authentication

User name:

Password:

 Save my password

Connect to a database

 Select or enter a database name:

ClassicModels

 Attach a database file:

Browse...

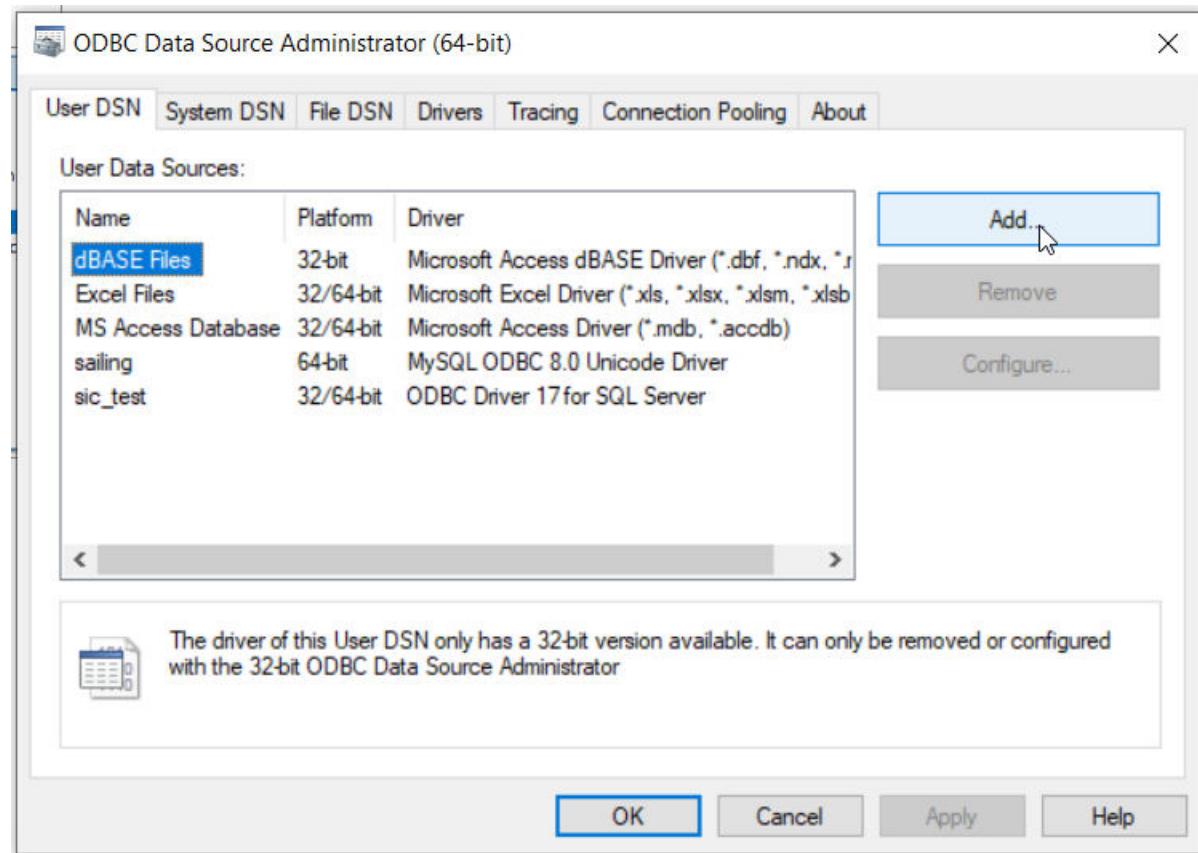
Logical name:

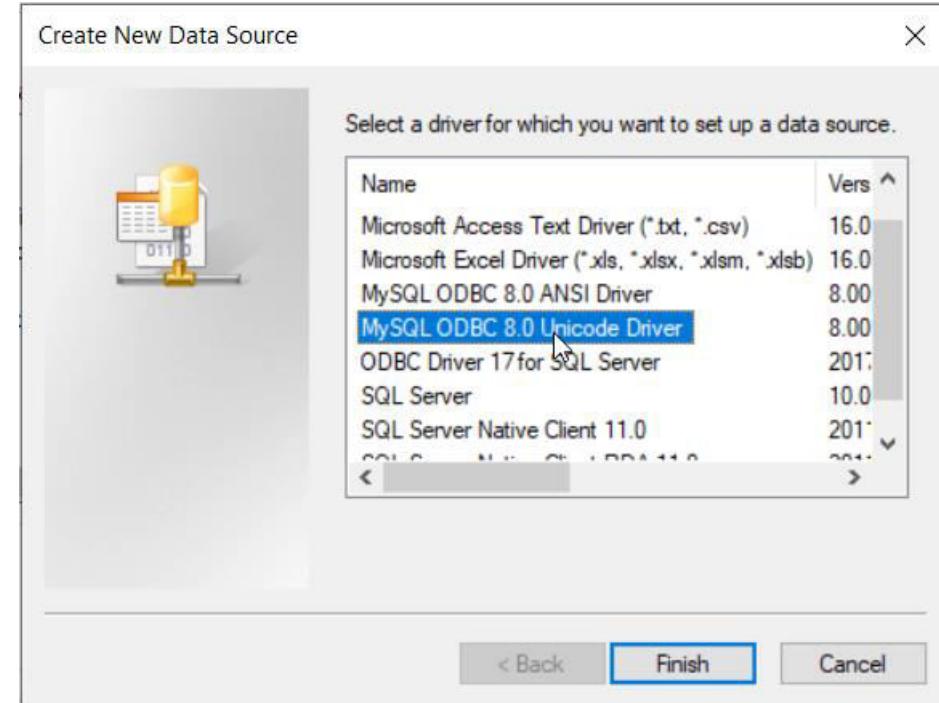
Advanced...

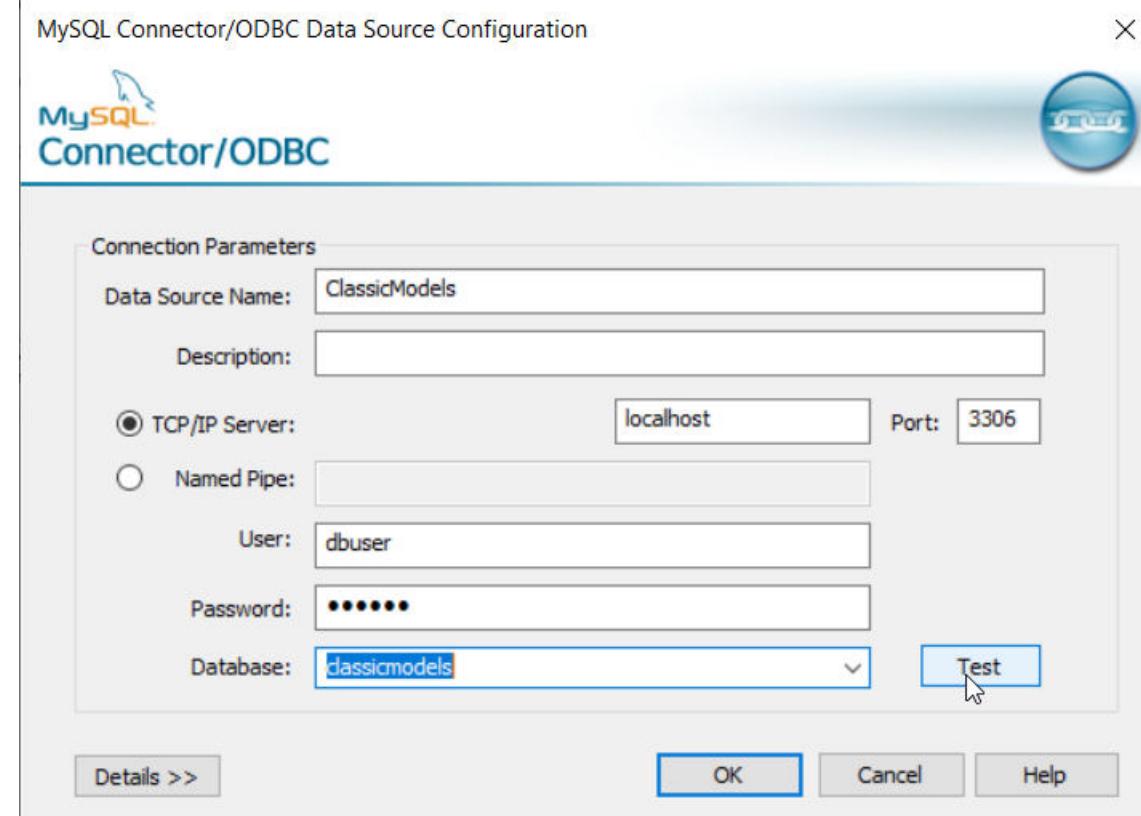
Test Connection

OK

Cancel







Report / CURSO Notes (V)

My1stReport.rdl - Microsoft Report Builder

File Home Insert View

Run Paste Views Clipboard Font Paragraph Border Number Layout

Report Data

New Edit... X ↑ ↓

- + Built-in Fields
- + Parameters
- + Images
- Data Sources
- DataSource1**
- Datasets

Click to add title

[&ExecutionTime]

A screenshot of the Microsoft Report Builder application. The window title is "My1stReport.rdl - Microsoft Report Builder". The ribbon menu includes "File", "Home", "Insert", and "View". The "Home" tab is selected, showing various toolbar icons for running reports, pasting, and viewing. Below the toolbar are sections for "Font", "Paragraph", "Border", "Number", and "Layout". A "Report Data" pane on the left lists report components: "Built-in Fields", "Parameters", "Images", "Data Sources" (with "DataSource1" selected), and "Datasets". The main workspace contains a large text box with the placeholder "Click to add title" and a smaller text box at the bottom right containing the expression "[&ExecutionTime]".

# Add DataSet

## Dataset Properties

X

|            |
|------------|
| Query      |
| Fields     |
| Options    |
| Filters    |
| Parameters |

Choose a data source and create a query.

Name:

DataSet1

- Use a shared dataset.  
 Use a dataset embedded in my report.

Data source:

DataSource1

New...

Query type:

- Text    Table    Stored Procedure

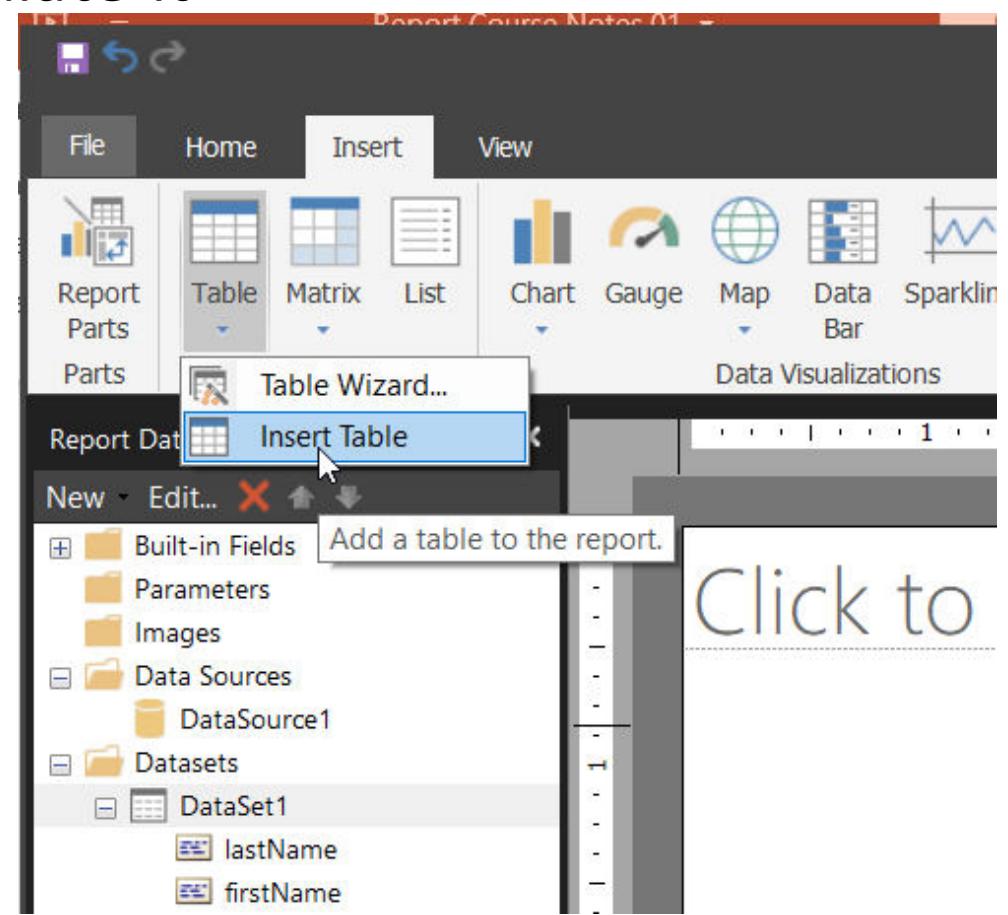
Query:

```
SELECT [lastName],[firstName]
 ,[extension],[email]
 ,Offices.city
 ,[jobTitle]
 ,[salary]
 FROM [ClassicModels].[dbo].[Employees]
 JOIN Offices ON Offices.officeCode = Employees.officeCode
```

- SELECT [firstName],[lastName]
- , [extension], [email]
- , Offices.city
- , [jobTitle]
- , [salary]
- FROM [ClassicModels].[dbo].[Employees]
- JOIN Offices ON Offices.officeCode = Employees.officeCode
- ORDER BY lastName, firstName

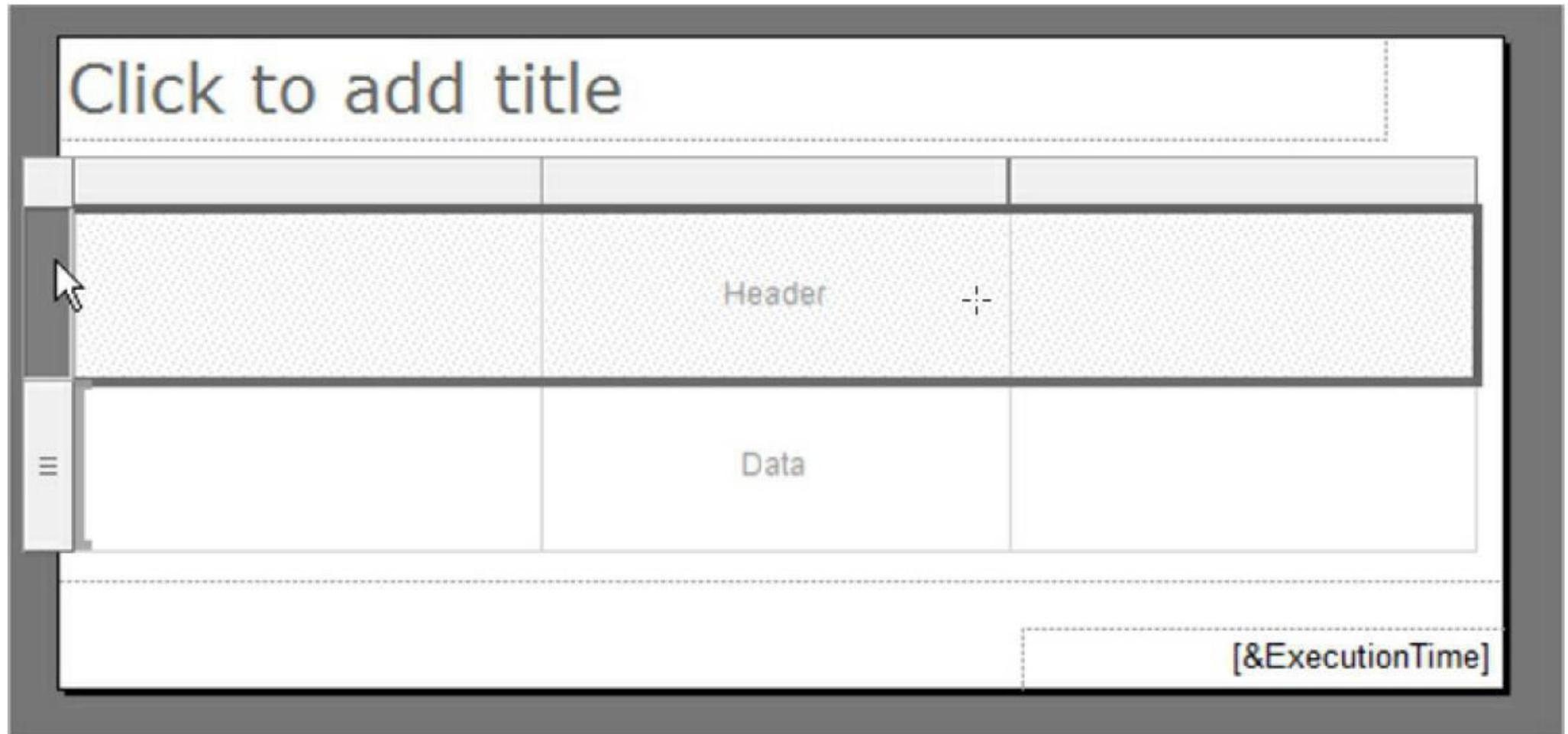
# Add Table

- place Table item on report and populate it
- Insert – Table – Insert Table
- drag and drop
- when release table is created
- occupy area defined
- by default, every cell in table
- occupied by empty text box

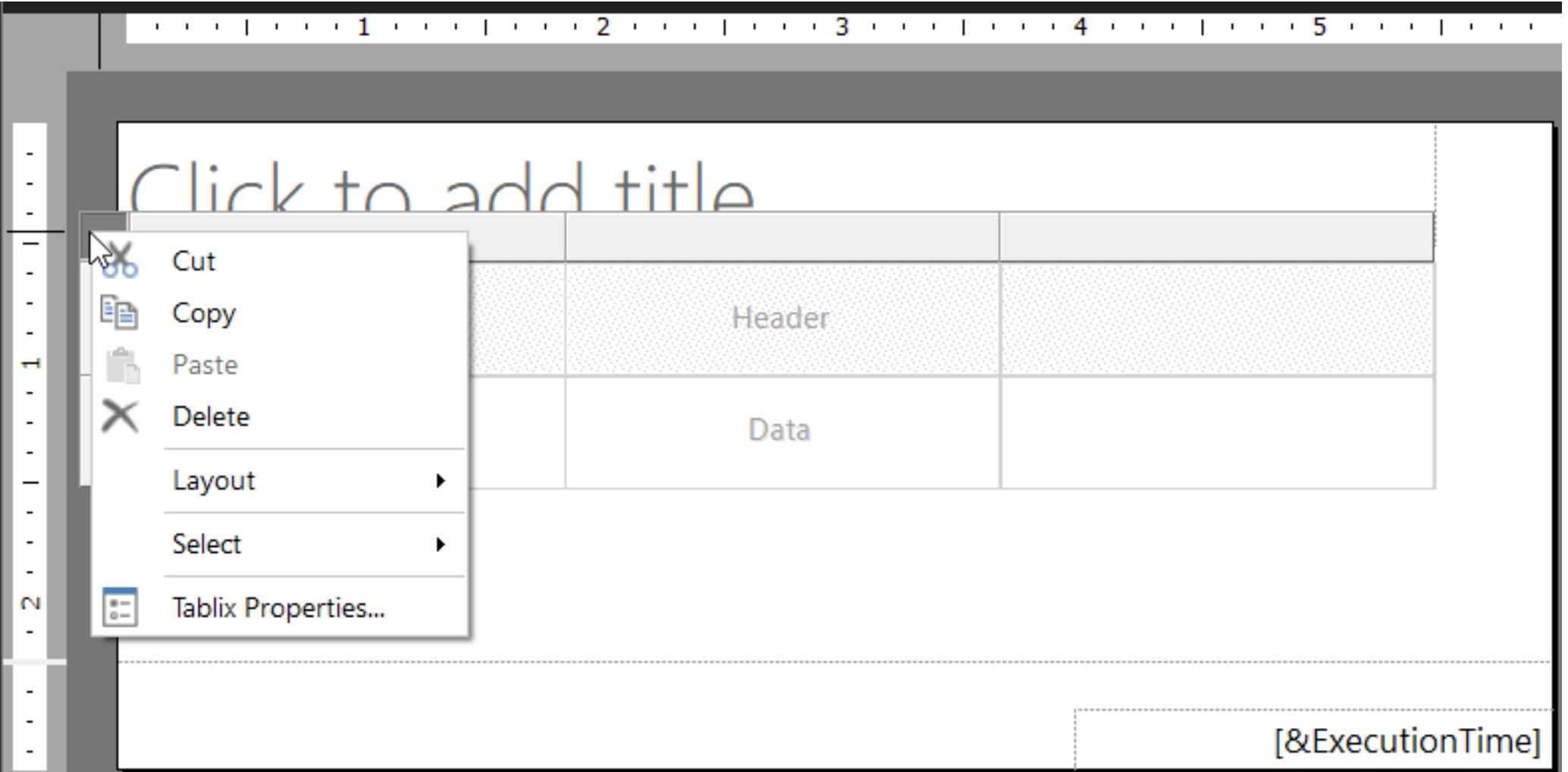


The screenshot shows the Microsoft Access Report Design View. At the top, there is a navigation bar with icons for back, forward, and search, followed by page numbers 1 through 5. Below the navigation bar is a title bar with the placeholder text "Click to add title". The main area contains a report structure with three sections: a header section labeled "Header" and a data section labeled "Data", both in blue font. A vertical column on the left side of the report body has numerical labels 1 and 2. In the bottom right corner of the report body, there is a placeholder box containing the text "[&ExecutionTime]". The entire report is enclosed in a black border.

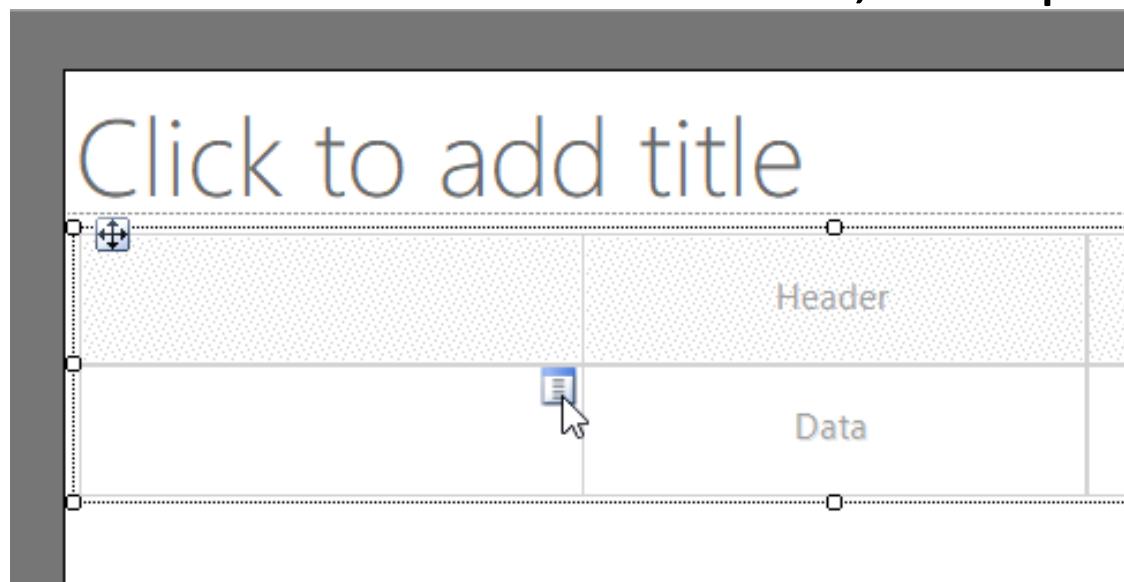
click any of gray rectangles in border to left of table  
selects row



click gray square in upper-left corner of border  
selects entire table – Tablix Properties



- hover cursor over table cell (lower-row table cell)
- small icon representing Field Selector will appear in upper-right corner of text box
- click icon Field Selector, list of fields defined in dataset, is displayed



select row by clicking gray rectangle in border to left of row – change format

The screenshot shows the Microsoft Report Builder interface with the title bar "My1stReport.rdl - Microsoft Report B". The ribbon menu is visible with tabs like File, Home, Insert, and View. The Home tab is selected, showing various toolbar icons for Run, Paste, Views, Clipboard, Font, Paragraph, Border, Number, and Layout. A "Report Data" pane on the left lists "Built-in Fields", "Parameters", "Images", "Data Sources" (DataSource1), and "Datasets" (DataSet1). DataSet1 is expanded to show fields: lastName, firstName, extension, email, city, jobTitle, and salary. In the main workspace, there is a table with three columns: "Name", "office City", and "Salary". The second row of the table contains the expressions "[lastName]", "[city]", and "[salary]". The first row is highlighted with a yellow selection bar on its left side, indicating it is selected for editing.

| Name       | office City | Salary   |
|------------|-------------|----------|
| [lastName] | [city]      | [salary] |

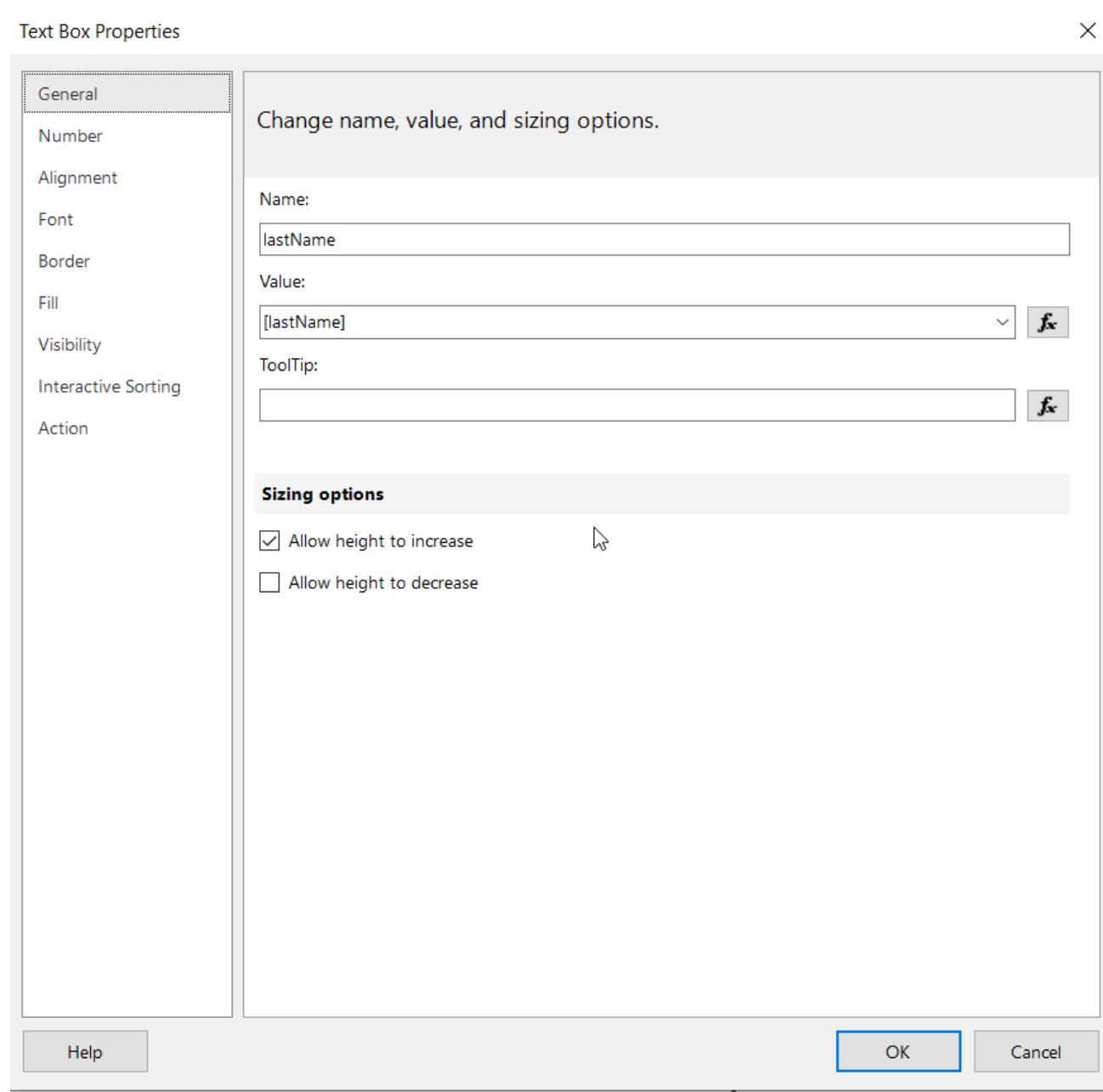
[&ExecutionTime]

A screenshot of the Microsoft Report Designer interface. At the top, there's a header bar with a title bar showing 'Report 01' and a ribbon with tabs 1 through 5. Below the header is a placeholder text 'Click to add title'. The main area contains a table with three columns: 'Name', 'office City', and 'Salary'. The 'Name' column contains a text box with the expression '[lastName]'. A context menu is open over this text box, listing options: Cut, Copy, Paste, Delete, Select (with a submenu), Expression..., and Text Box Properties... (which is highlighted). The 'Text Box' section of the menu is also visible. To the right of the table, another text box contains the expression '[&ExecutionTime]'. The background shows a dark grey design with some dashed lines.

| Name       | office City | Salary   |
|------------|-------------|----------|
| [lastName] | [city]      | [salary] |

[&ExecutionTime]

## Text Box Properties



Value:

[lastName]



ToolTip:

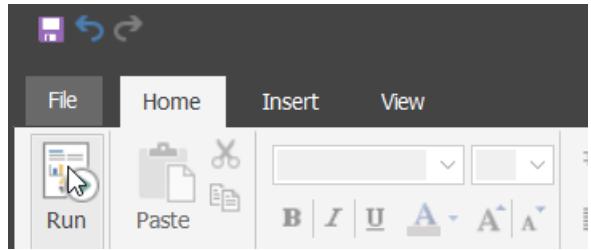
Expression

Set expression for: Value  
=Fields!firstName.Value&Fields!lastName.Value

| Category:                                                                                                                                                                                                                                                                                                                                                 | Item: | Values:                                                                                                                                                 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>Constants</li><li>Built-in Fields</li><li>Parameters</li><li>Fields (DataSet1)</li><li>Datasets</li><li>Variables</li><li>Operators<ul style="list-style-type: none"><li>Arithmetic</li><li>Comparison</li><li>Concatenation</li><li>Logical/Bitwise</li><li>Bit Shift</li></ul></li><li>Common Functions</li></ul> | <All> | <ul style="list-style-type: none"><li>lastName</li><li>firstName</li><li>extension</li><li>email</li><li>city</li><li>jobTitle</li><li>salary</li></ul> |

```
=Fields!firstName.Value & " "
Fields!lastName.Value
```

- =
- Fields (Dataset1)
  - firstName - Fields!firstName.Value
- Operators
  - Concatenation
    - & “ “ &
- Fields (Dataset1)
  - lastName - Fields!lastName.Value



# Home - Run

| <u>Name</u>   | <u>office City</u> | <u>Salary</u> |
|---------------|--------------------|---------------|
| Gerard Bondur | Paris              | 9,709         |
| Loui Bondur   | Paris              | 5,277         |
| Larry Bott    | London             | 7,996         |
| Anthony Bow   | San Francisco      | 7,941         |

# Number

Text Box Properties

General

Number

Alignment

Font

Border

Fill

Visibility

Interactive Sorting

Action

Set number and date formatting options.

Category:

- Default
- Number
- Currency
- Date
- Time
- Percentage
- Scientific
- Custom

Sample  
12,345

Use regional formatting

Decimal places:

Use 1000 separator (,)

Show values in: Thousands

Show zero as: -

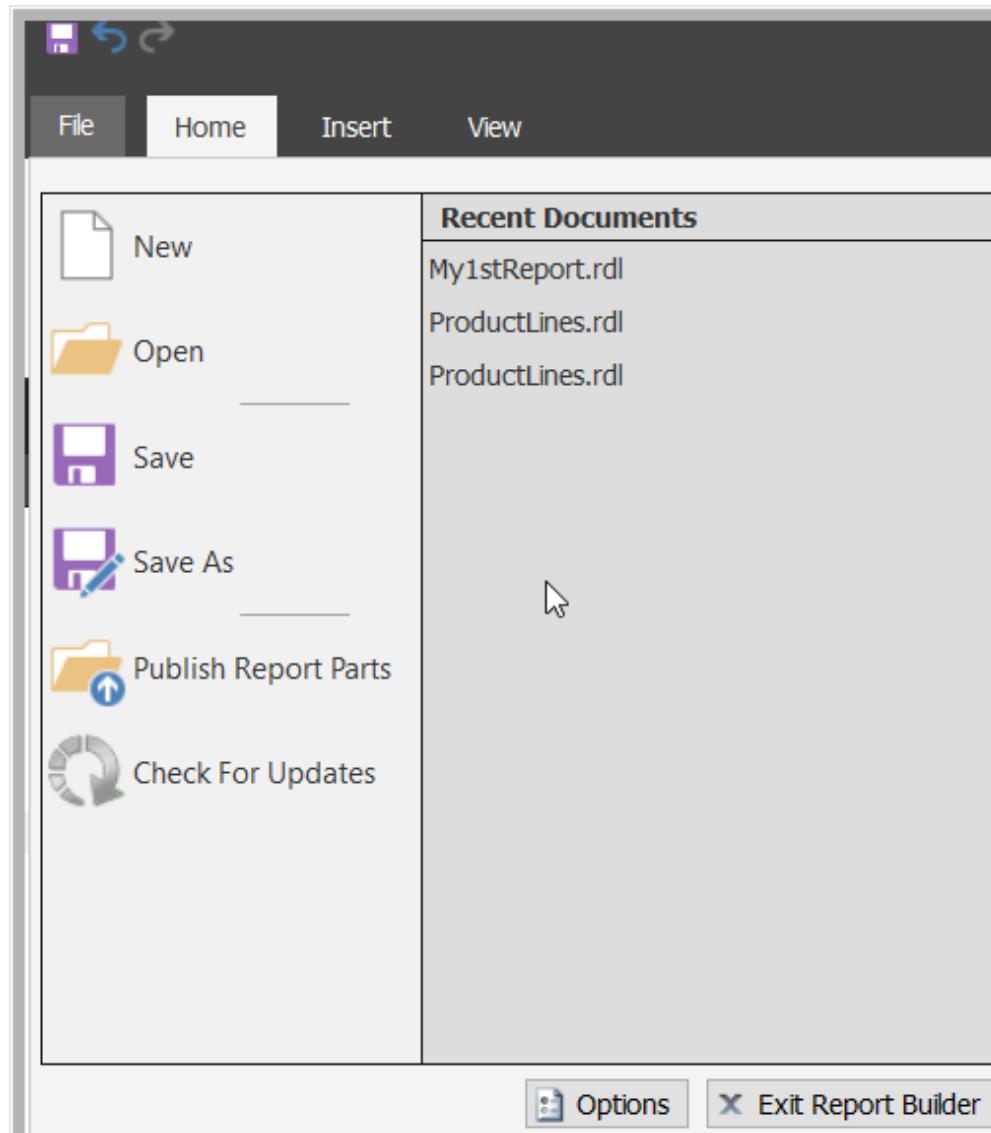
Negative numbers:

- (12,345)
- 12,345
- 12,345
- 12,345-
- 12,345 -

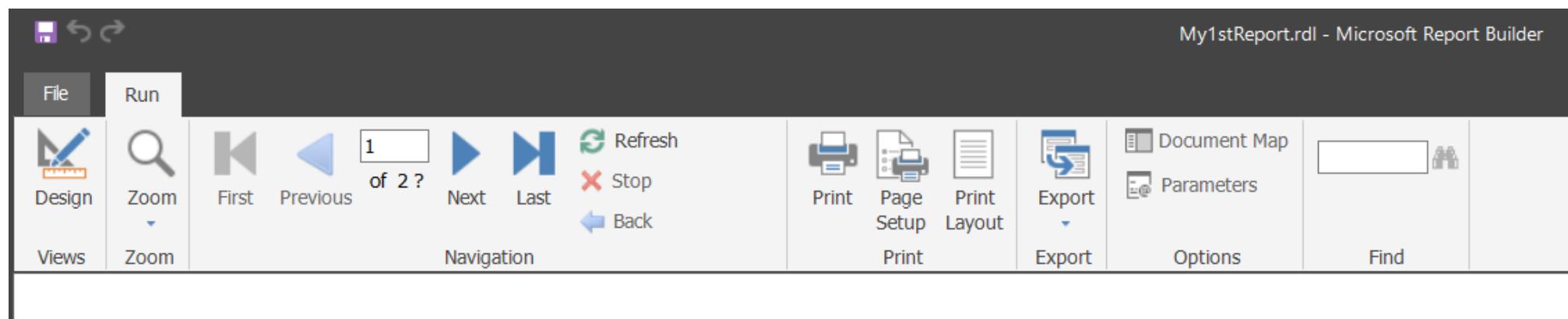
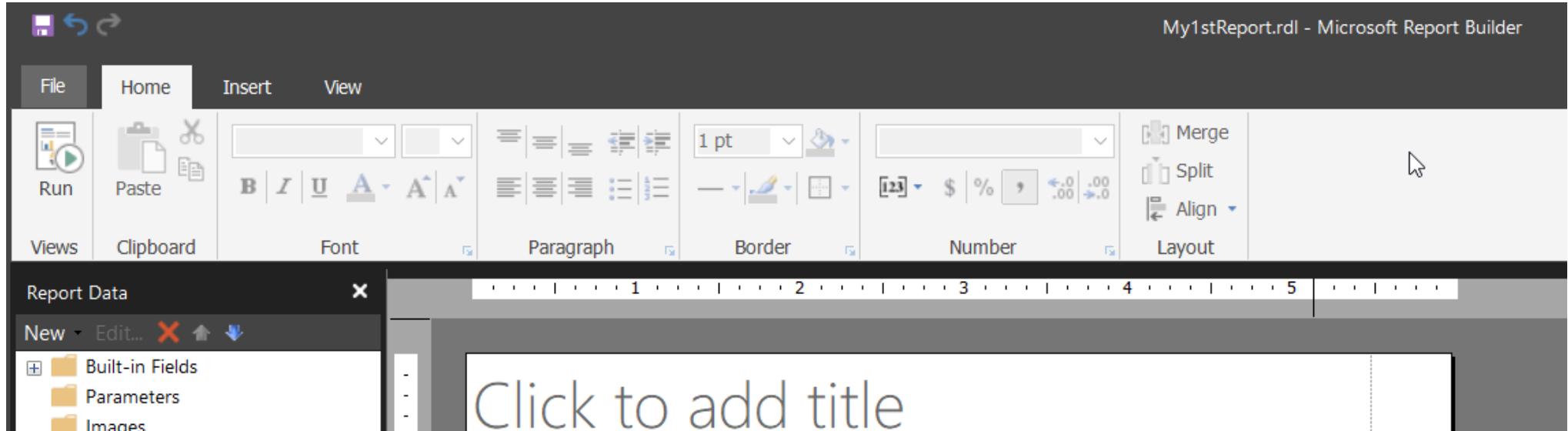
- Text – Left Justified
- Number – Right Justified

# Report Builder - Menu

# File



# Home Run

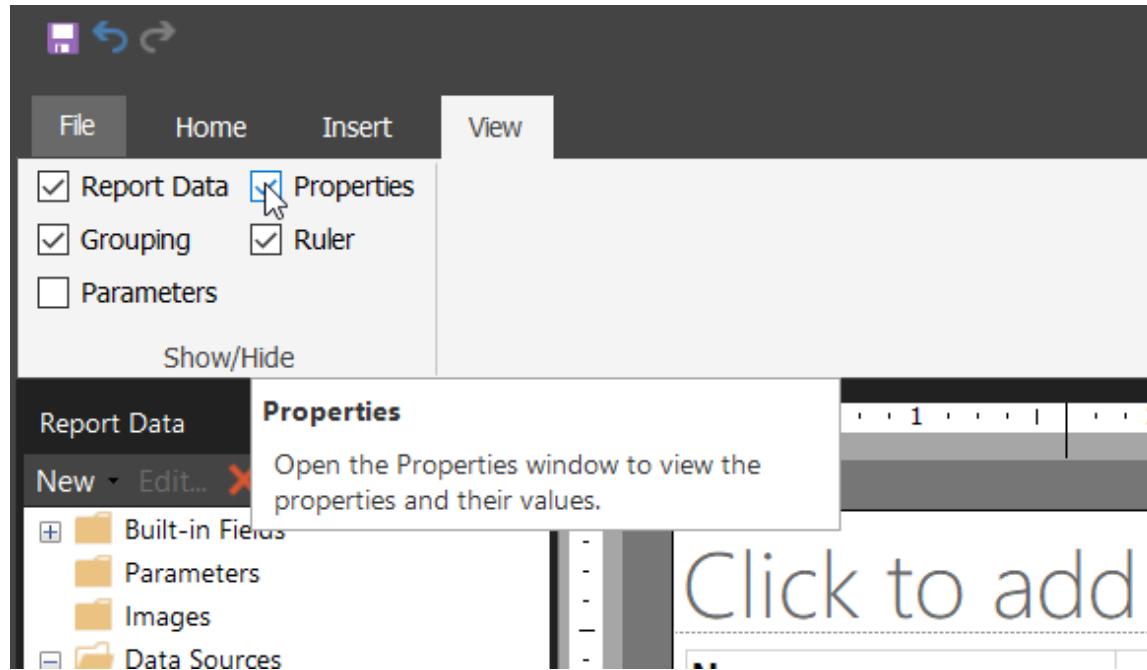


# Insert

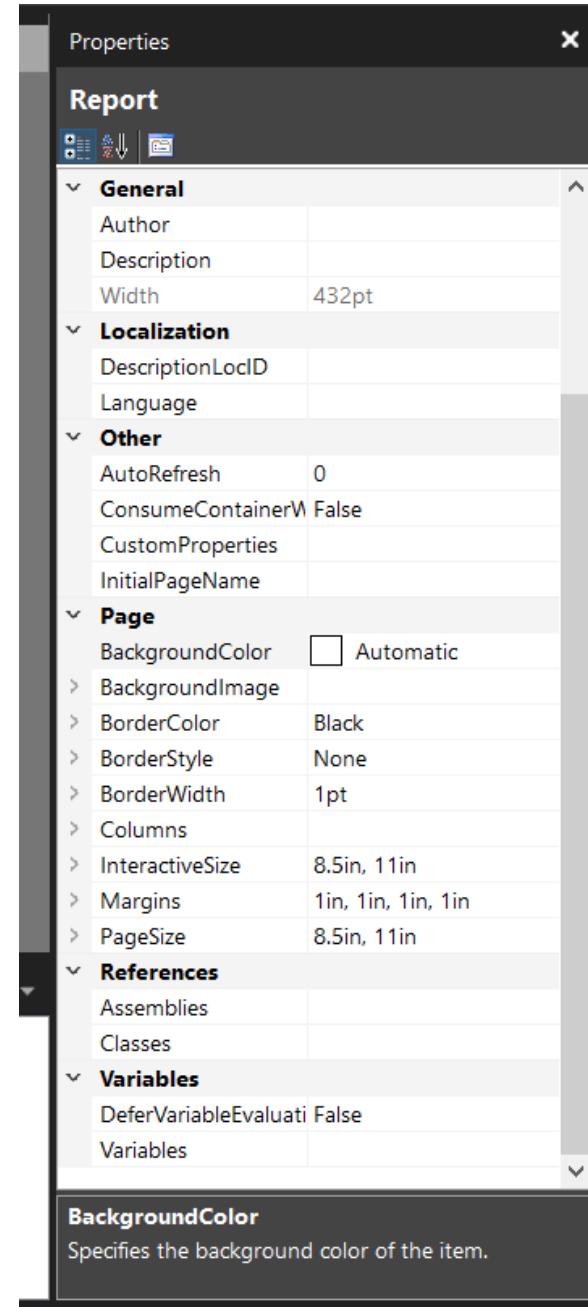
The screenshot shows the Microsoft Report Builder interface. The top navigation bar has tabs for File, Home, Insert (which is currently selected), and View. Below the ribbon is a toolbar with various icons for report parts like Report Parts, Table, Matrix, List, Chart, Gauge, Map, Data Bar, Sparkline, Indicator, Text Box, Image, Line, Rectangle, Subreport, Header, and Footer. On the left, there's a 'Report Data' pane with a tree view showing Built-in Fields, Parameters, Images, Data Sources (DataSource1), and Datasets (DataSet1). DataSet1 contains fields: lastName, firstName, extension, email, city, jobTitle, and salary. The main workspace shows a placeholder 'Click to add title' above a table. The table has three columns: Name, office City, and Salary. The 'Name' column contains the expression «Expr». The 'office City' column contains the expression [city]. The 'Salary' column contains the expression [salary]. In the bottom right corner of the workspace, there is a placeholder for execution time: [&ExecutionTime].

| Name   | office City | Salary   |
|--------|-------------|----------|
| «Expr» | [city]      | [salary] |

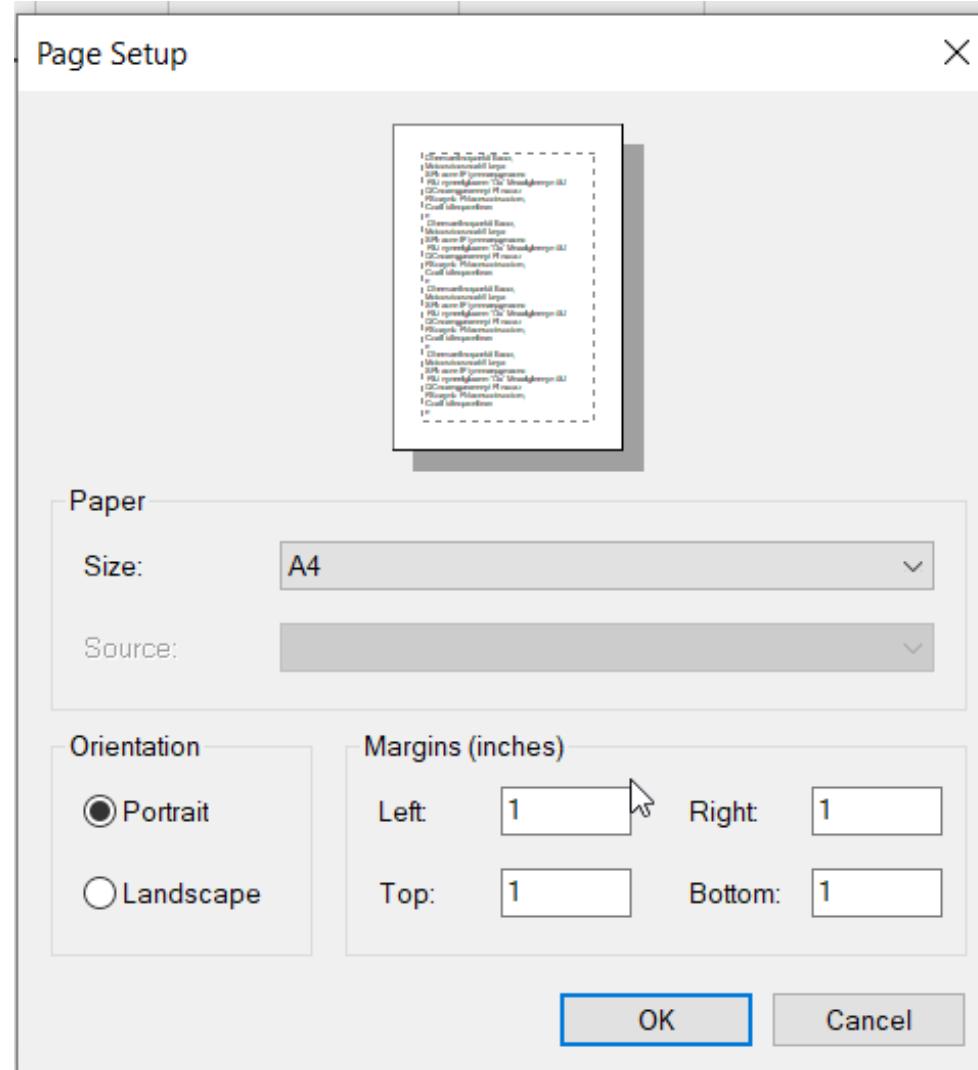
# View



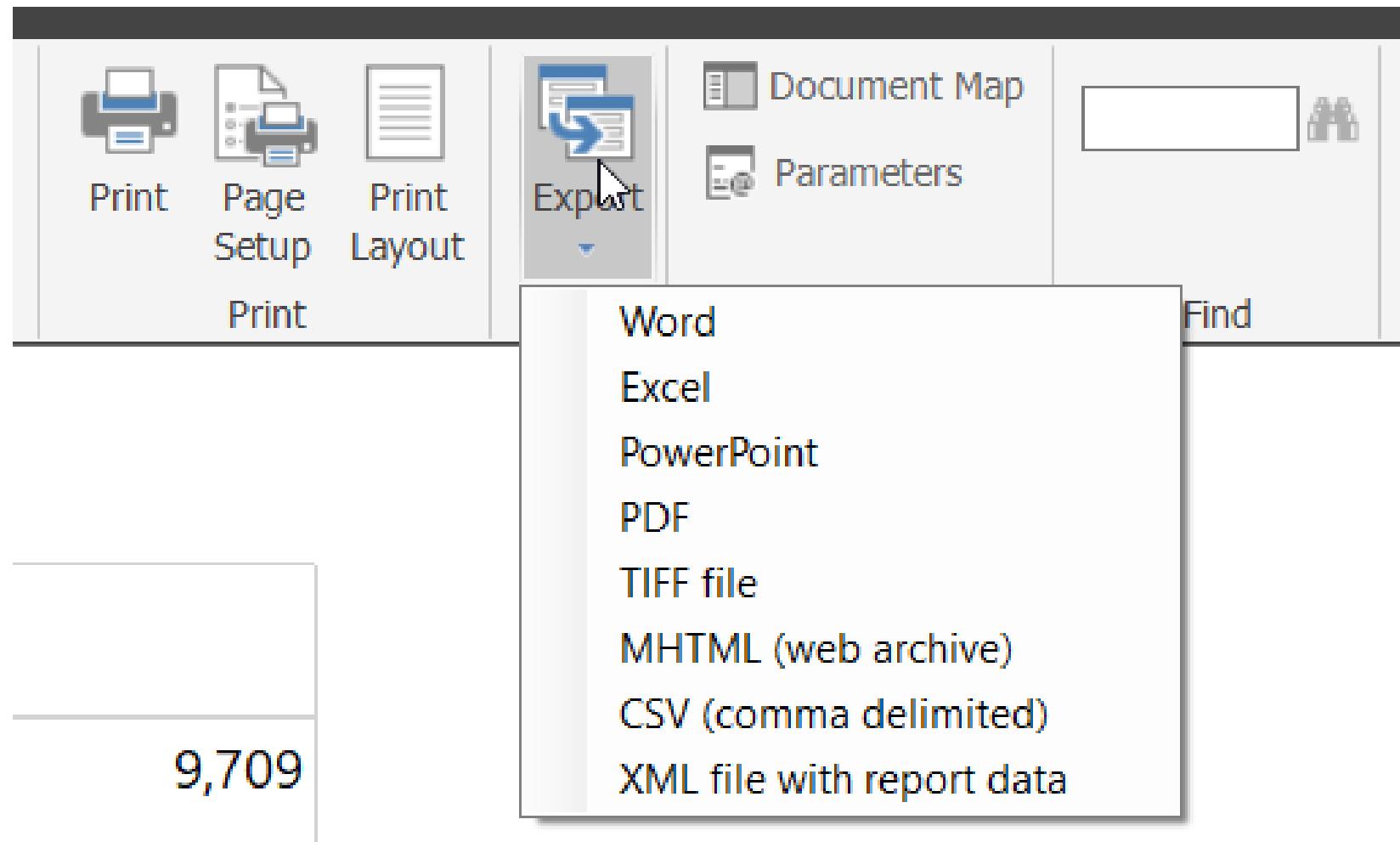
# Report Properties



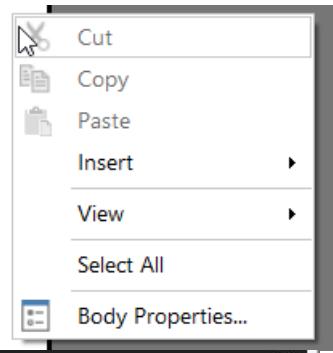
# Page Setup



# Export



# Report Body Properties



The screenshot shows the 'Report Body Properties' dialog box and the 'Properties' panel.

**Report Body Properties Dialog:**

- Fill:** No Color
- Border:** Black, None, 1pt
- Background image:** External
- Use this image:** (empty dropdown)

**Properties Panel:**

- Body** node selected.
- Border:**
  - BorderColor: Black
  - BorderStyle: None
  - BorderWidth: 1pt
- Fill:**
  - BackgroundColor: No Color
  - BackgroundImage: (empty)
- Position:**
  - Size: 6in, 2.25in

- 6" body size = 8" page size – 1 – 1 (left & right margin)
- on portrait orientation
- 9" on landscape orientation

# My1stReport

## Report Title

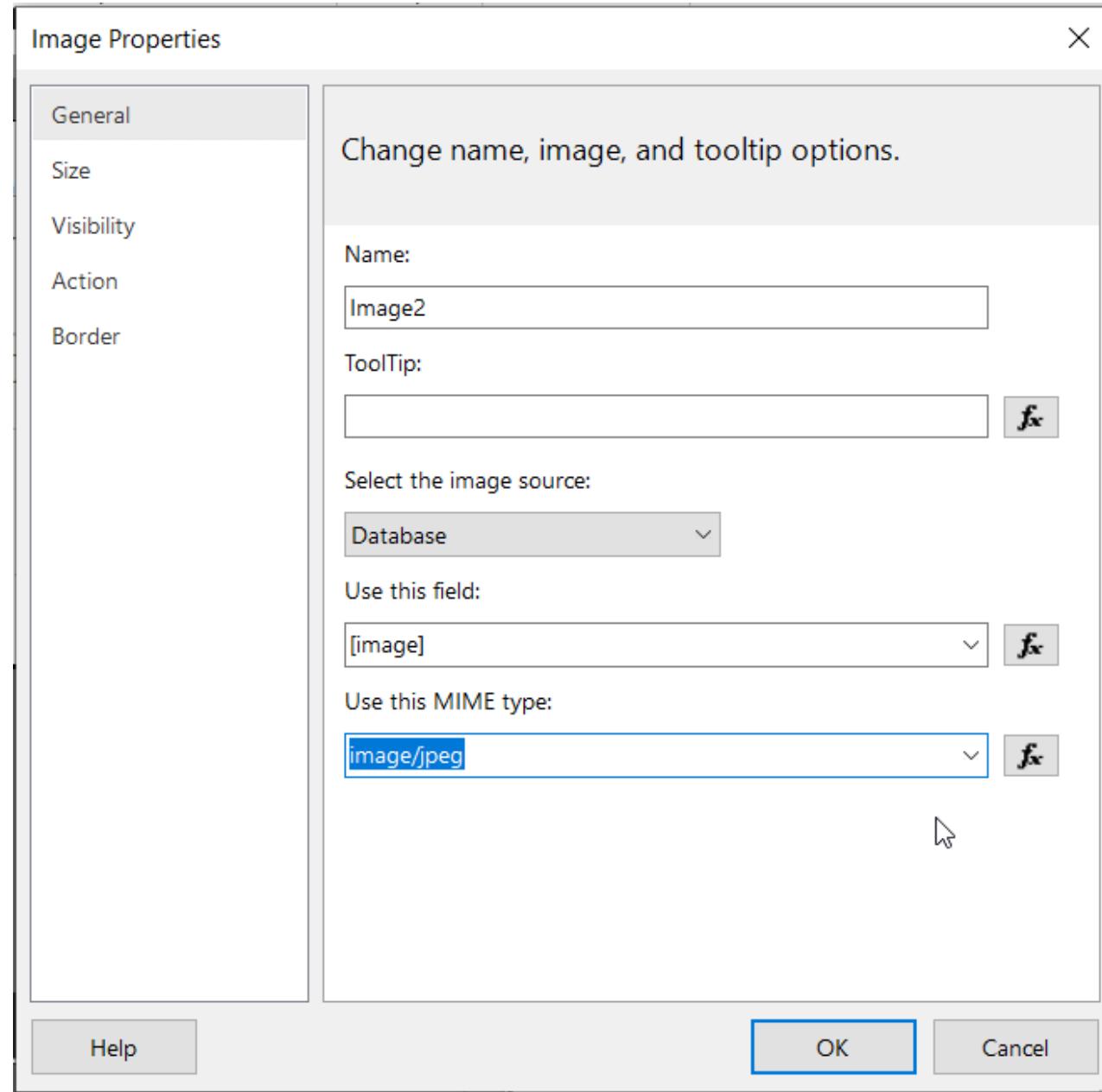
| <u>Name</u>     | <u>office City</u> | <u>Salary</u> |
|-----------------|--------------------|---------------|
| Gerard Bondur   | Paris              | 9,709         |
| Loui Bondur     | Paris              | 5,277         |
| Larry Bott      | London             | 7,996         |
| Anthony Bow     | San Francisco      | 7,941         |
| Pamela Castillo | Paris              | 6,107         |
| Jeff Firrelli   | San Francisco      | 5,751         |
| Julie Firrelli  | Boston             | 5,914         |
| Andy Fixter     | Sydney             | 7,078         |
| Martin Gerard   | Paris              | 5,512         |

# My2ndReport

# Report Title

| Name         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                       |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| Classic Cars | Attention car enthusiasts: Make your wildest car ownership dreams come true. Whether you are looking for classic muscle cars, dream sports cars or movie-inspired miniatures, you will find great choices in this category. These replicas feature superb attention to detail and craftsmanship and offer features such as working steering system, opening forward compartment, opening rear trunk with removable spare wheel, 4-wheel independent spring suspension, and so on. The models range in size from 1:10 to 1:24 scale and include numerous limited edition and several out-of-production vehicles. All models include a certificate of authenticity from their manufacturers and come fully assembled. |    |
| Motorcycles  | Our motorcycles are state of the art replicas of classic as well as contemporary motorcycle legends such as Harley Davidson, Ducati and Vespa. Models contain stunning details such as official logos, rotating wheels, working kickstand, front suspension, gear-shift lever, footbrake lever, and drive chain. Materials used include diecast and plastic. The models range in size from 1:10 to 1:50 scale and include numerous limited edition and several out-of-production vehicles. All models come fully assembled and ready for display in the home or office. Most include a certificate of authenticity.                                                                                                 |    |
| Planes       | Unique, diecast airplane and helicopter replicas suitable for collections, as well as home, office or classroom decorations. Models contain stunning details such as official logos and insignias, rotating jet engines and propellers, retractable wheels, and so on. Most come fully assembled and with a certificate of authenticity from their manufacturers.                                                                                                                                                                                                                                                                                                                                                   |  |
| Ships        | The perfect holiday or anniversary gift for executives, clients, friends, and family. These handcrafted model ships are unique, stunning works of art that will be                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |  |

# Insert - Image



# My3rdReport

Namejob TitleE Mailoffice CitySalary

## Report Title

|                 |                     |                                 |        |       |
|-----------------|---------------------|---------------------------------|--------|-------|
| Julie Firrelli  | Sales Rep           | jfirrelli@classicmodelcars.com  | Boston | 5,914 |
| Steve Patterson | Sales Rep           | spatterson@classicmodelcars.com | Boston | 5,277 |
| Larry Bott      | Sales Rep           | lbott@classicmodelcars.com      | London | 7,996 |
| Barry Jones     | Sales Rep           | bjones@classicmodelcars.com     | London | 9,268 |
| Foon Yue Tseng  | Sales Rep           | ftseng@classicmodelcars.com     | NYC    | 5,277 |
| George Vanauf   | Sales Rep           | gvanauf@classicmodelcars.com    | NYC    | 5,636 |
| Gerard Bondur   | Sale Manager (EMEA) | gbondur@classicmodelcars.com    | Paris  | 9,709 |
| Loui Bondur     | Sales Rep           | lbondur@classicmodelcars.com    | Paris  | 5,277 |
| Pamela Castillo | Sales Rep           | pcastillo@classicmodelcars.com  | Paris  | 6,107 |
| Martin Gerard   | Sales Rep           | mgerard@classicmodelcars.com    | Paris  | 5,518 |

| <u>Name</u>       | <u>job Title</u>     | <u>E Mail</u>                       | <u>office City</u> | <u>Salary</u> |
|-------------------|----------------------|-------------------------------------|--------------------|---------------|
| Leslie Thompson   | Sales Rep            | lthompson@classicmodelcars.com      | San Francisco      | 5,927         |
| Andy Fixter       | Sales Rep            | afixter@classicmodelcars.com        | Sydney             | 7,078         |
| Tom King          | Sales Rep            | tking@classicmodelcars.com          | Sydney             | 9,950         |
| Peter Marsh       | Sales Rep            | pmarsh@classicmodelcars.com         | Sydney             | 5,373         |
| William Patterson | Sales Manager (APAC) | wpatterson@classicmodelcars.co<br>m | Sydney             | 6,588         |
| Yoshimi Kato      | Sales Rep            | ykato@classicmodelcars.com          | Tokyo              | 9,806         |
| Mami Nishi        | Sales Rep            | mnishi@classicmodelcars.com         | Tokyo              | 7,609         |

- Report Properties
  - Page Size
- Insert Header

# My1stReport

- Table Grouping
- other Report Formatting

- click drop-down arrow
- in Row Groups pane; select
- Add Group | Parent Group

Report Title

| <u>Name</u>        | <u>office City</u> | <u>Salary</u>    |
|--------------------|--------------------|------------------|
| <<Expr>><br>[city] | [salary]           | [&ExecutionTime] |

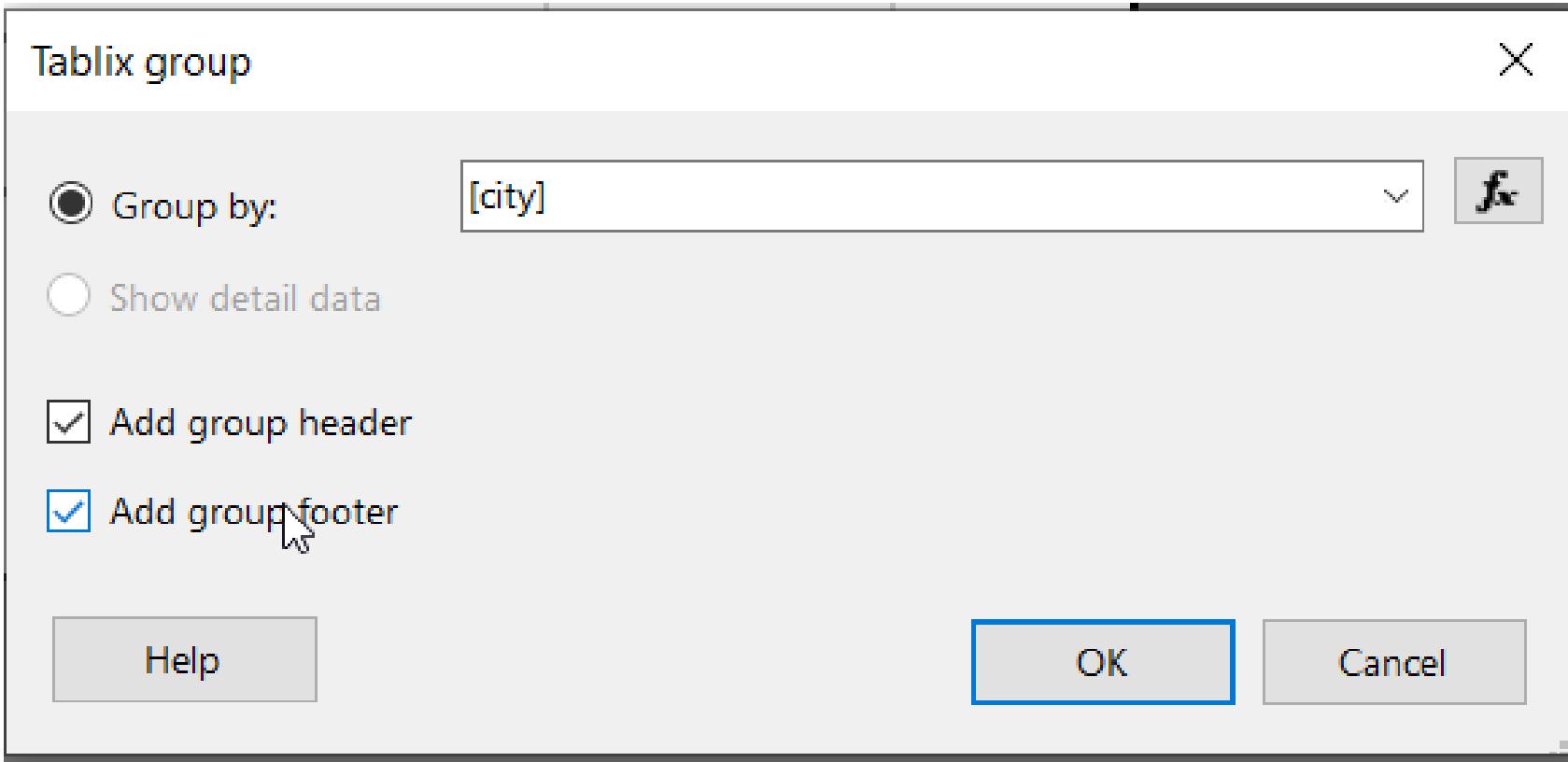
Report Title

| <u>Name</u> | <u>office City</u> | <u>Salary</u>    |
|-------------|--------------------|------------------|
| [city]      | [salary]           | [&ExecutionTime] |

Report 01

Report Title

| <u>Name</u>        | <u>office City</u> | <u>Salary</u>    |
|--------------------|--------------------|------------------|
| <<Expr>><br>[city] | [salary]           | [&ExecutionTime] |



# Report Title

|        |        |                      |               |
|--------|--------|----------------------|---------------|
| [city] | «Expr» | [jobTitle]           | [salary]      |
|        |        | <b>Total Salary:</b> | [Sum(salary)] |



[&ExecutionTime]

- do not confuse grouping in report with GROUP BY clause used in SQL

# Report Title

|        |                  |                     |        |
|--------|------------------|---------------------|--------|
| Boston |                  |                     |        |
|        | Julie Firrelli   | Sales Rep           | 5,914  |
|        | Steve Patterson  | Sales Rep           | 5,277  |
|        | Total Salary:    |                     | 11,191 |
| London |                  |                     |        |
|        | Larry Bott       | Sales Rep           | 7,996  |
|        | Barry Jones      | Sales Rep           | 9,268  |
|        | Total Salary:    |                     | 17,264 |
| NYC    |                  |                     |        |
|        | Foon Yue Tseng   | Sales Rep           | 5,277  |
|        | George Vanauf    | Sales Rep           | 5,636  |
|        | Total Salary:    |                     | 10,913 |
| Paris  |                  |                     |        |
|        | Gerard Bondur    | Sale Manager (EMEA) | 9,709  |
|        | Loui Bondur      | Sales Rep           | 5,277  |
|        | Pamela Castillo  | Sales Rep           | 6,107  |
|        | Martin Gerard    | Sales Rep           | 5,518  |
|        | Gerard Hernandez | Sales Rep           | 5,277  |
|        | Total Salary:    |                     | 31,888 |

# Report Title

|                                |                     |        |
|--------------------------------|---------------------|--------|
| Boston Total Salary of:        |                     | 11,191 |
| Julie Firrelli                 | Sales Rep           | 5,914  |
| Steve Patterson                | Sales Rep           | 5,277  |
| London Total Salary of:        |                     | 17,264 |
| Larry Bott                     | Sales Rep           | 7,996  |
| Barry Jones                    | Sales Rep           | 9,268  |
| NYC Total Salary of:           |                     | 10,913 |
| Foon Yue Tseng                 | Sales Rep           | 5,277  |
| George Vanauf                  | Sales Rep           | 5,636  |
| Paris Total Salary of:         |                     | 31,888 |
| Gerard Bondur                  | Sale Manager (EMEA) | 9,709  |
| Loui Bondur                    | Sales Rep           | 5,277  |
| Pamela Castillo                | Sales Rep           | 6,107  |
| Martin Gerard                  | Sales Rep           | 5,518  |
| Gerard Hernandez               | Sales Rep           | 5,277  |
| San Francisco Total Salary of: |                     | 39,805 |
| Anthony Bow                    | Sales Manager (NA)  | 7,941  |
| Jeff Firrelli                  | VP Marketing        | 5,751  |
| Leslie Jennings                | Sales Rep           | 6,149  |
| Diane Murphy                   | President           | 8,575  |

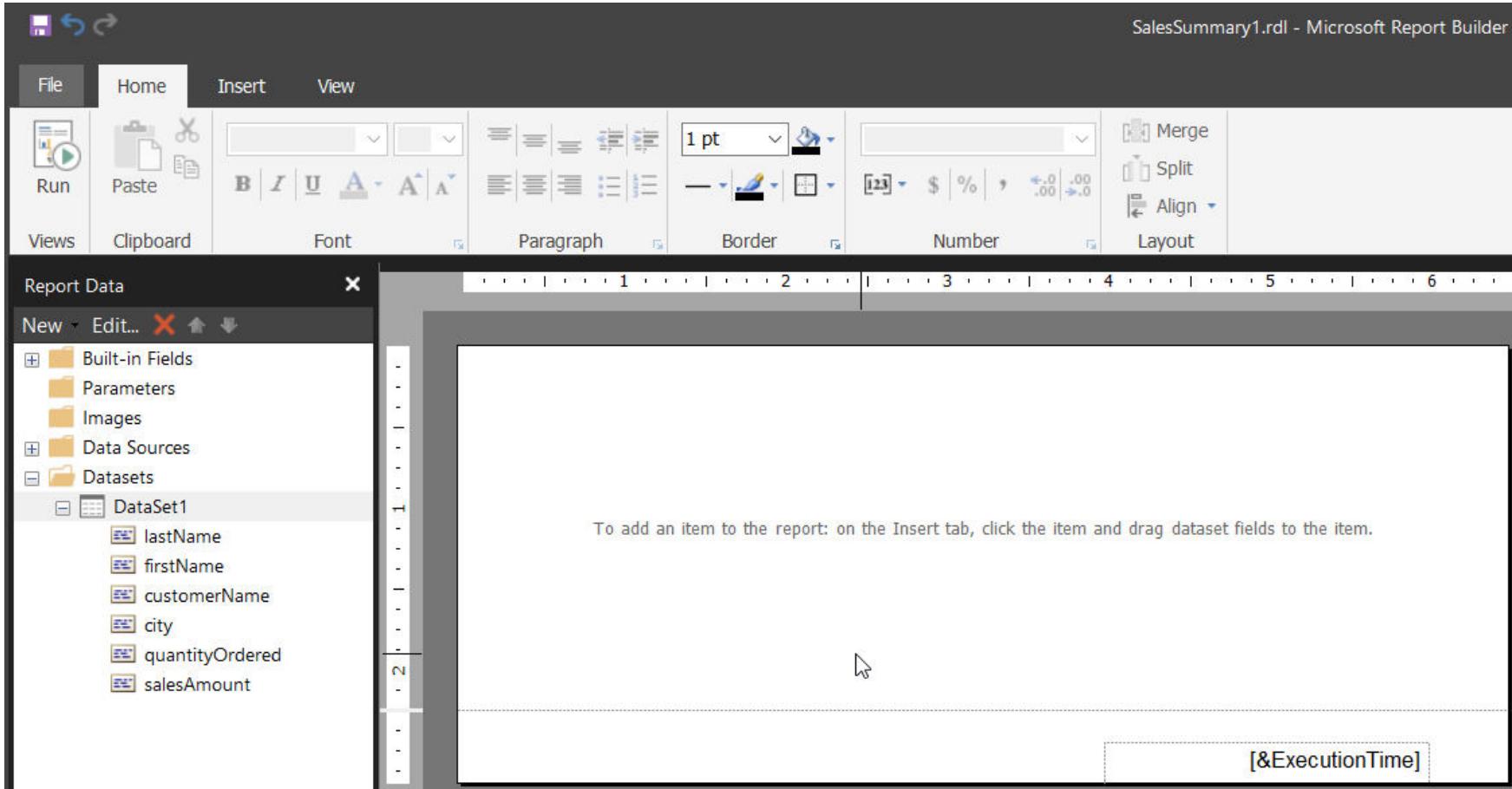
# data region - tablix

- for working with datasets: tablix, chart, gauge, map, sparkline
- tablix: table, matrix, list as data region items – there are simply templates for creating tablix data region
- each data region item has property called DataSetName - name of dataset used by data region

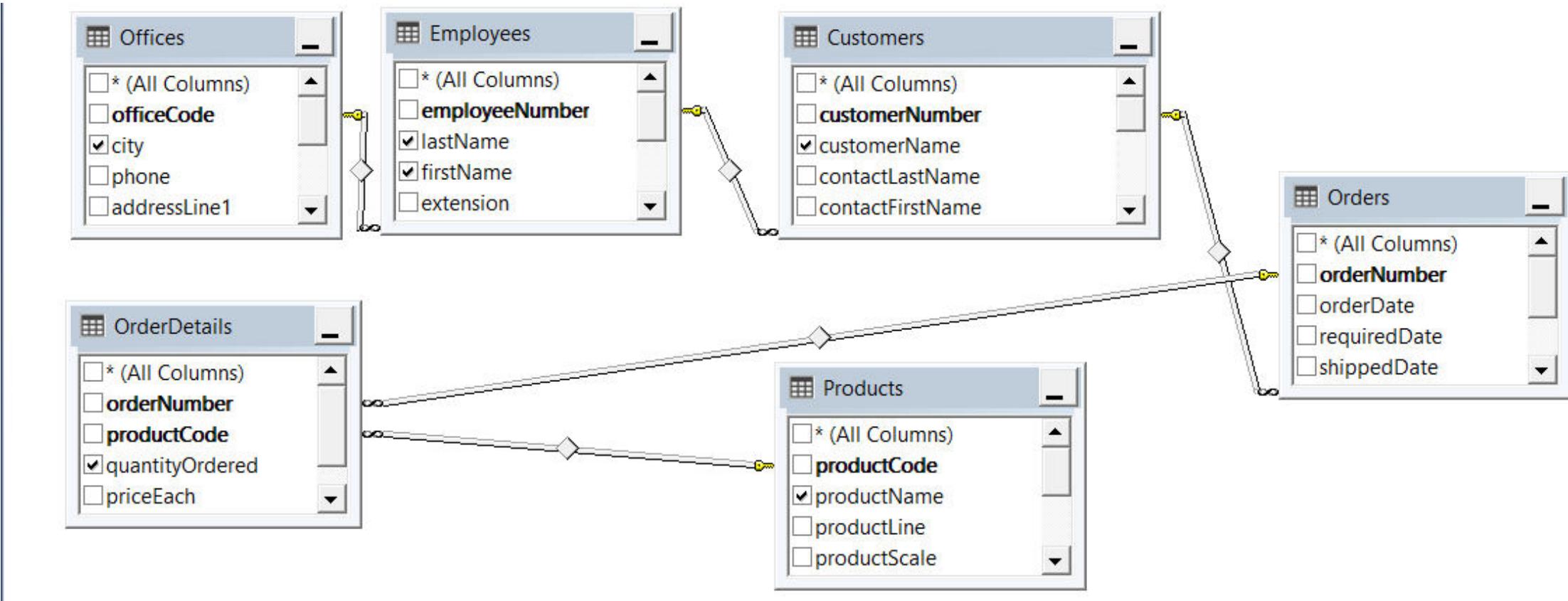
# Report

# Basic Report Design

# Sales Summary 01



- SELECT Offices.city,
- Employees.firstName, Employees.lastName,
- Customers.customerName, Products.productName,
- OrderDetails.quantityOrdered,
- OrderDetails.priceEach \* OrderDetails.quantityOrdered AS salesAmount
- FROM Offices
- JOIN Employees ON Offices.officeCode = Employees.officeCode
- JOIN Customers ON Employees.employeeNumber = Customers.salesRepEmployeeNumber
- JOIN Orders ON Customers.customerNumber = Orders.customerNumber
- JOIN OrderDetails ON Orders.orderNumber = OrderDetails.orderNumber
- JOIN Products ON OrderDetails.productCode = Products.productCode



Report Data

New Edit... X ↑ ↓

- Built-in Fields
- Parameters
- Images
- Data Sources
- Datasets
  - DataSet1
    - lastName
    - firstName
    - customerName
    - city
    - productName
    - quantityOrdered
    - salesAmount

Sales Summary

| City   | Employee | Customer       | Product          | Qua     |
|--------|----------|----------------|------------------|---------|
| [city] | «Expr»   | [customerName] | [productName]    | uantity |
|        |          |                | [&ExecutionTime] |         |

## Sales Summary

| City   | Employee        | Customer                     | Product                                   | Quantity | Sales Amount |
|--------|-----------------|------------------------------|-------------------------------------------|----------|--------------|
| Boston | Steve Patterson | Online Diecast Creations Co. | 1917 Grand Touring Sedan                  | 30       | \$4,080.00   |
| Boston | Steve Patterson | Online Diecast Creations Co. | 1911 Ford Town Car                        | 50       | \$2,754.50   |
| Boston | Steve Patterson | Online Diecast Creations Co. | 1932 Alfa Romeo 8C2300 Spider Sport       | 22       | \$1,660.12   |
| Boston | Steve Patterson | Online Diecast Creations Co. | 1936 Mercedes Benz 500k Roadster          | 49       | \$1,729.21   |
| London | Barry Jones     | Blauer See Auto, Co.         | 1932 Model A Ford J-Coupe                 | 25       | \$2,701.50   |
| London | Barry Jones     | Blauer See Auto, Co.         | 1928 Mercedes-Benz SSK                    | 26       | \$4,343.56   |
| London | Barry Jones     | Blauer See Auto, Co.         | 1939 Chevrolet Deluxe Coupe               | 45       | \$1,463.85   |
| London | Barry Jones     | Blauer See Auto, Co.         | 1938 Cadillac V-16 Presidential Limousine | 46       | \$2,040.10   |
| NYC    | Foon Yue Tseng  | Vitachrome Inc.              | 1937 Lincoln Berline                      | 39       | \$3,726.45   |