

DataBase

DataBase people

- Me - Calin CENAN
- You

DataBase site

- <http://users.utcluj.ro/~calin/DataBase/>

Test

- Why study databases ... ?
- myths ...
- What are *you* want to learn during course and laboratory work on database topic

DataBase

- database technology still plays key role in information management
- database theory and technology are important components in center of information systems

- meet curriculum requirements
- help students developing database systems
- Database, design database, design database system (database system - integrated project that consists of various technologies)
- how database can be used to process information

- **autovit.ro** last transaction (Naspers) value **4.7 mil. Euro**
- Site revenue > 630.000 euro / year
- maintenance < 100.000 euro
- DataBase = value = \$ (a lot of \$)

- Goal
- TextBooks
- Definition
 - Data Model - Relational
- Importance
- Relation theory

DataBase Course's GOAL

- to present introduction to database and database management systems
 - organize information in DBMS
 - retrieve it efficiently
- concentrates on *relational* systems, which are by far the dominant type of DBMS today

DataBase Course Schedule

- Monday 16 – 18 room P03

DataBase Practice GOAL

- use DataBase Management System
- implement databases
 - MS Access
 - SQL Server
 - MySQL
- ***Structured Query Language SQL***
- implement databases applications

DataBase Laboratory Work Schedule

- Tuesday 8 – 20 room 214 Observator group 1, 4, 3
- Thursday 10 – 14 room 214 Observator group 2

Course Textbooks

- **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, Ed. Albastra 2001
- Baze de date
- Baze de date și gestiunea tranzacțiilor
- Utilizarea sistemului SQL Server (SQL 7.0, SQL 2000)

Course Textbooks

- Sams Teach Yourself SQL in 10 Minutes
- SQL For Dummies
- Lynn Beighley, Head First SQL - Your Brain on SQL, O'Reilly, 2009
- <https://www.w3schools.com/sql/>
- ***https://sqlzoo.net/***

Course Textbooks

- %SQL Server%
- %MySQL%
- %Access%

Course

PART I: DATABASE CONCEPTS

- Introduction to Database Systems
 - Basic Concepts and Definitions
 - Data Versus Information
 - Database, DataBase Management System
- Database System Architecture
 - Schemas
 - ANSI-SPARC Database Architecture: Internal, Conceptual, External
 - **Data Independence**
 - Data Models
 - Entity-Relationship (E-R) Data Model

PART II: RELATIONAL MODEL

- Relational Algebra and Calculus
- Relational Query Languages
 - Codd's Rules
 - **Structured Query Language (SQL)**
- Entity-Relationship (ER) Model

PART III: DATABASE DESIGN

- Functional Dependency, Decomposition
- Normalization

PART IV: PHYSICAL, COMMERCIAL DATABASES

- Physical Data Organization
 - Indexing
- COMMERCIAL DATABASES
 - Microsoft SQL Server
 - MySQL
 - php, web database applications
 - Microsoft Access

PART V

- Query Processing and Optimization
- Transaction Processing and Concurrency Control
- Database Recovery System
 - Backup & Restore
- Database Security

PARTS for Advanced Hardcore DataBase Design 4th year

- Object-Oriented Databases
- Object-Relational Database
- Parallel Database Systems
- Distribution Database Systems
- Decision Support Systems (DSS)
- Data Warehousing and Data Mining
- Emerging Database Technologies
(NoSQL)

Laboratory

- 1 week Access
- 2-3 weeks Access Query, SQL, SQLzoo
- 1-2 weeks Access forms, reports, ...
- 2 weeks SQL Server
- 2 weeks MySQL
- 2 weeks DataBase design
- 2 weeks DataBase Applications

Curricula Syllabus

- [http://cs.utcluj.ro/csd/pics/plan_invatamant
/Fise_an2_3.cs_18-19.pdf](http://cs.utcluj.ro/csd/pics/plan_invatamant/Fise_an2_3.cs_18-19.pdf)

- 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.*

You can safely skip the rest of the slides about

DATABASE = ROMANTIC

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!

- *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
 - E. Cebuc, M. Ivan, M. Joldos, T. Dadarlat ...
 - ...
- ...

- DataBase vs rest of (Object Oriented) Computer Science

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

categories

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

- | CategoryID | CategoryName | Description |
|------------|---------------------------------------|---|
| 1 | Beverages | Soft drinks, coffees, teas, beers, and ales |
| 2 | Condiments
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	Maria Anders Germany	Sales Representative 030-0074321	030-0076545	Obere Str. 57 Berlin
•	ANATR	Ana Trujillo Emparedados y helados México D.F.	05021	Ana Trujillo Mexico	Owner (5) 555-4729	Avda. de la Constitución 2222 (5) 555-3745
•	ANTON	Antonio Moreno Taquería D.F.	05023	Antonio Moreno Mexico	Owner (5) 555-3932	Mataderos 2312
•	AROUT	Around the Horn London	NULL	Thomas Hardy WA1 1DP	Sales Representative (171) 555-7788	120 Hanover Sq. (171) 555-6750
•	BERGS	Berglunds snabbköp Luleå	NULL	Christina Berglund S-958 22	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
•	BLONP	Blondesddsl père et fils Strasbourg	NULL	Frédérique Citeaux 67000	Marketing Manager 88.60.15.31	24, place Kléber
•	BOLID	Bólido Comidas preparadas Madrid	NULL	28023	Martín Sommer Spain	Owner (91) 555 22 82
•	BONAP	Bon app' 13008	France	91.24.45.40	Owner 91.24.45.41	12, rue des Bouchers
•	BOTTM	Bottom-Dollar Markets Tswassen	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 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)

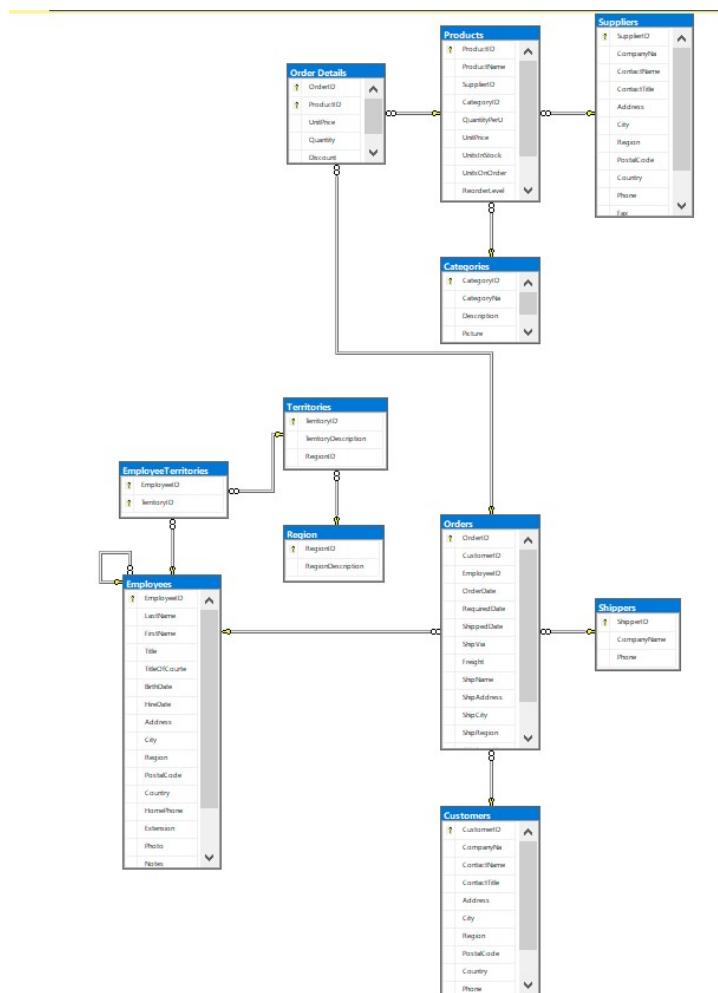
orders

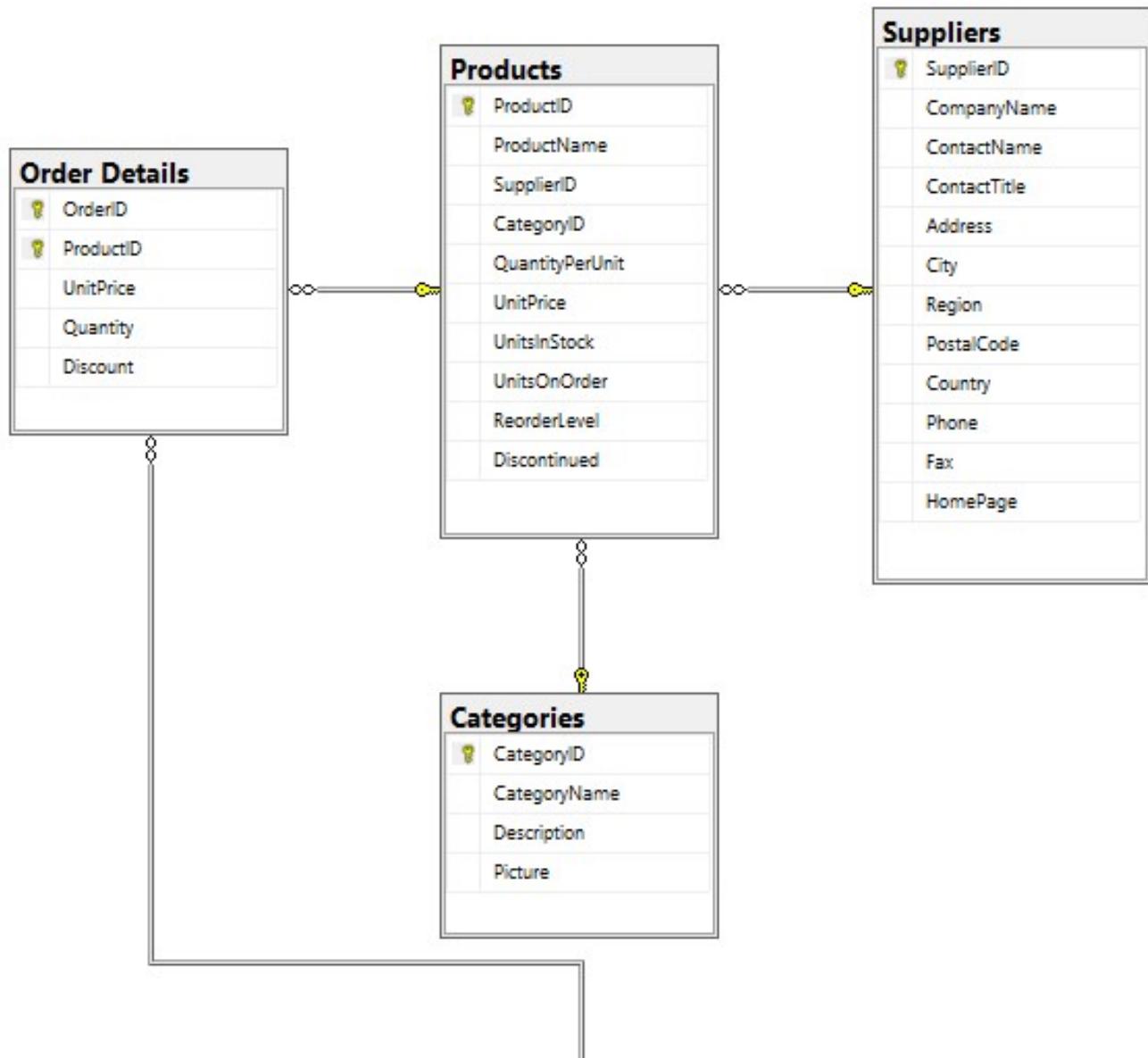
- OrderID, integer, Primary Key
- CustomerID, varchar(5); EmployeeID, integer - Foreign Keys
- OrderDate, date
- RequiredDate, date; ShippedDate, date
- ShipVia, integer Foreign Key
- Freight, double
- ShipName, varchar(40)
- ShipAddress, varchar(60)
- ShipCity, varchar(15)
- ShipRegion, varchar(15)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

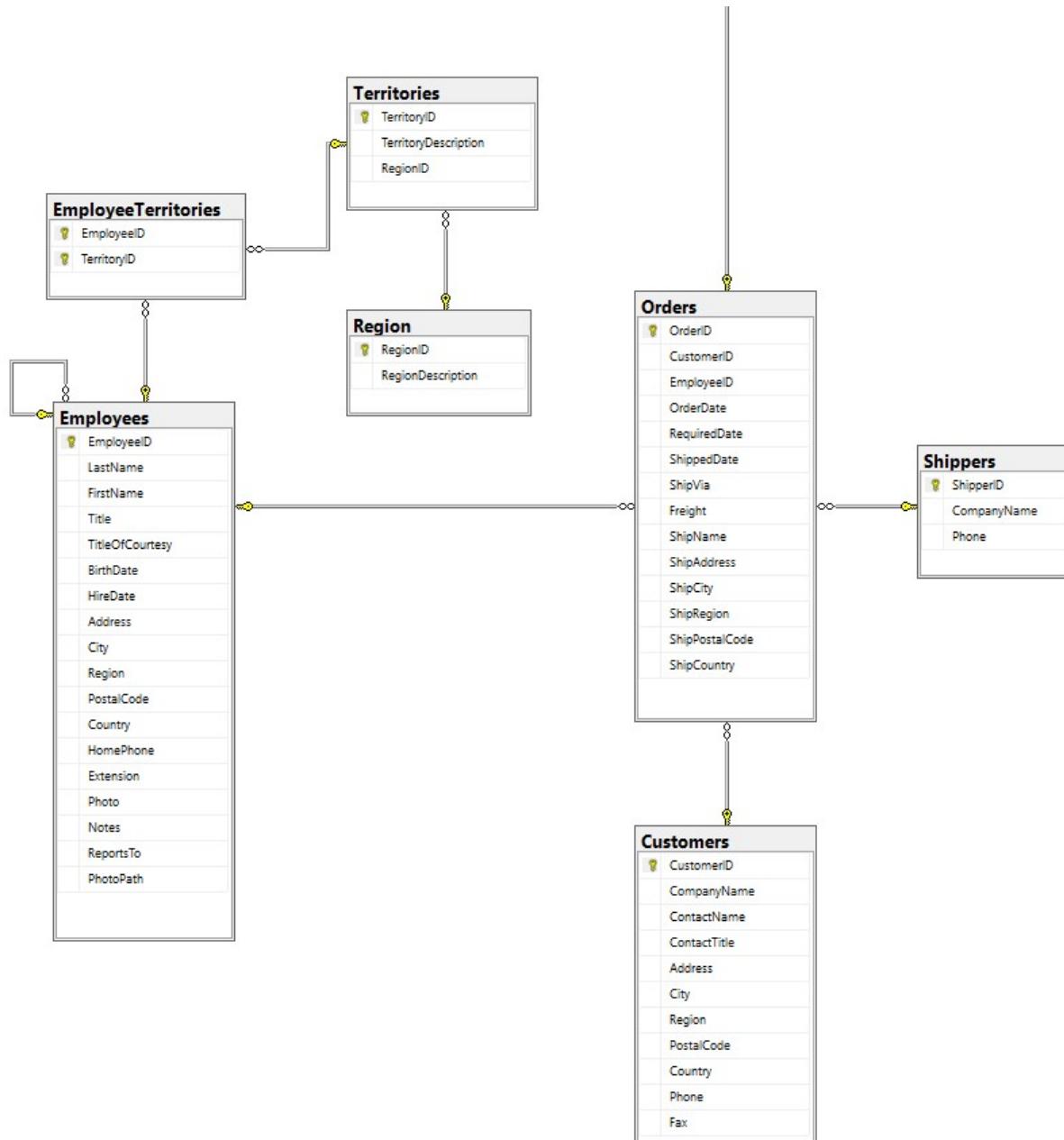
order_details

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

Suppliers of Beverages AND Condiments

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

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

Gotcha ?!

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

Suppliers of Beverages AND Condiments

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

Suppliers of Beverages AND Condiments

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

Suppliers of Beverages OR Condiments

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

use parentheses to group conditions together in WHERE clause

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

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

group by and having Clauses

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

- SELECT Title, COUNT(EmployeeID)
 - FROM Employees GROUP BY Title
-
- Title (No column name)
 - Inside Sales Coordinator 1
 - Sales Manager 1
 - Sales Representative 6
 - Vice President, Sales 1

- SELECT Title, COUNT(EmployeeID)
 - FROM Employees
 - GROUP BY Title
 - HAVING COUNT(EmployeeID) > 1
-
- Title (No column name)
 - Sales Representative 6

order by Clause

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

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

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

Sorting, Group By via Expressions

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

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

Sorting via Numeric Placeholders

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

FILTER

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

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

Condition

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

Condition

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

Condition

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

Condition

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

Condition

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

Condition

- with *in* operator, you can write single condition no matter how many expressions are in set
- along with writing your own set of expressions you can use subquery to generate set for you
- sometimes you want to see whether particular expression exists within set of expressions, and sometimes you want to see whether expression does *not* exist
- can use *not in* operator

Condition

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

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

Condition

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

Condition

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

Condition

- `SELECT FirstName, LastName, Title FROM Employees WHERE EmployeeID IN (`
- `SELECT DISTINCT ReportsTo FROM Employees)`
- `SELECT FirstName, LastName, Title FROM Employees WHERE`
- `Title LIKE '%president%' OR`
- `Title LIKE '%manager%'`

Null

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

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

- SELECT FirstName, LastName, Title FROM Employees
- WHERE ReportsTo !=2
- account for possibility that some rows might contain null in ReportsTo column
- SELECT FirstName, LastName, Title FROM Employees
- WHERE ReportsTo !=2 OR ReportsTo IS NULL

Null Values and Three Valued Logic

- Any comparison with *null* returns *unknown*
 - E.g. $5 < \text{null}$ or $\text{null} < \text{null}$ or $\text{null} = \text{null}$
- Three-valued logic using the truth value *unknown*:
 - OR: $(\text{unknown} \text{ or } \text{true}) = \text{true}$, $(\text{unknown} \text{ or } \text{false}) = \text{unknown}$
 $(\text{unknown} \text{ or } \text{unknown}) = \text{unknown}$
 - AND: $(\text{true} \text{ and } \text{unknown}) = \text{unknown}$, $(\text{false} \text{ and } \text{unknown}) = \text{false}$,
 $(\text{unknown} \text{ and } \text{unknown}) = \text{unknown}$
 - NOT: $(\text{not unknown}) = \text{unknown}$
 - “*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
 UnitPrice UnitsInStock UnitsOnOrder ReorderLevel
- 1 Chai 1 10 boxes x 20 bags 18.00 39 0 10
- 2 Chang 1 24 - 12 oz bottles 19.00 17 40 25
- 3 Aniseed Syrup 2 12 - 550 ml bottles 10.00 13 70
 25
- 4 Chef Anton's Cajun Seasoning 2 48 - 6 oz jars 22.00
 53 0 0
- 5 Chef Anton's Gumbo Mix 2 36 boxes 21.35 0 0
 0
- 6 Grandma's Boysenberry Spread 2 12 - 8 oz jars 25.00
 120 0 25
- 7 Uncle Bob's Organic Dried Pears 7 12 - 1 lb pkgs. 30.00
 15 0 10
- 8 Northwoods Cranberry Sauce 12 - 12 oz jars 40.00 6
 0 0
- ...

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

Cartesian Product

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

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

Cartesian Product

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

Inner Joins

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

Inner Joins

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

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

Inner Joins

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

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

Joining Three or More Tables

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

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

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

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

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

Using Subqueries As Tables in JOIN

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

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

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

Performance

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

Using Same Table Twice: Self-Joins

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

Using Same Table Twice: Self-Joins

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

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

Equi-Joins vs Non-Equi-Joins

- all of queries shown thus far have employed ***equi-joins***, meaning that values from two tables must match for join to succeed
- ***equi-join*** always employs an equals sign ON Products.CategoryID =Categories.CategoryID
- for example, let's say that managers has decided to have tennis tournament for all their people and you have been asked to create list of all pairings

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

Non-Equi-Joins

- `SELECT s1.FirstName + ' ' + s1.LastName,`
- `' vs. ', s2.FirstName + ' ' + s2.LastName`
- `FROM Employees s1 INNER JOIN Employees s2
ON s1.EmployeeID != s2.EmployeeID`
- Join Employees table to itself and return all rows where EmployeeID don't match (since person can't play tennis against himself)

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

SETS

Set Operators

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

union Operator

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

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

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

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

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

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

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

intersect Operator

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

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

except Operator

- Find Customers who Ordered Products from all Categories

CREATE VIEW CustomCateg AS

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

SELECT * FROM CustomCateg

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

- SELECT CategoryName FROM CustomCateg
WHERE CompanyName = 'Alfreds Futterkiste'
- CategoryName
 - 1. Beverages
 - 2. Condiments
 - 3. Dairy Products
 - 4. Produce
 - 5. Seafood

- SELECT CategoryName FROM CustomCateg WHERE CompanyName = 'QUICK-Stop'
- CategoryName
 - 1. Beverages
 - 2. Condiments
 - 3. Confections
 - 4. Dairy Products
 - 5. Grains/Cereals
 - 6. Meat/Poultry
 - 7. Produce
 - 8. Seafood

- SELECT Categories.CategoryName FROM Categories EXCEPT
- SELECT CategoryName FROM CustomCateg WHERE CompanyName = 'Alfreds Futterkiste'
- CategoryName
 1. Confections
 2. Grains/Cereals
 3. Meat/Poultry

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

- result it is the \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 2,147,483,647s 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 Administrator 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, vchar_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 vchar_fld = 'This is a piece of extremely long, too long varchar data';`
- *ERROR : Data too long for column 'vchar_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 ('Tirgomodina', 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 = DATEFROMMPARTS (2017,
MONTH(OrderDate), DAY(OrderDate)) ,`
- `RequiredDate = DATEFROMMPARTS (2017,
MONTH(RequiredDate), DAY(RequiredDate)) ,`
- `ShippedDate = DATEFROMMPARTS (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 SQL Server Management Studio window with two tabs: 'Results' and 'Messages'. The 'Results' tab is selected and displays a table with three columns: ID, Value, and another unnamed column. The data consists of nine rows, each containing values for ID, Value, and the unnamed column. The unnamed column is highlighted with a dashed border around its cell.

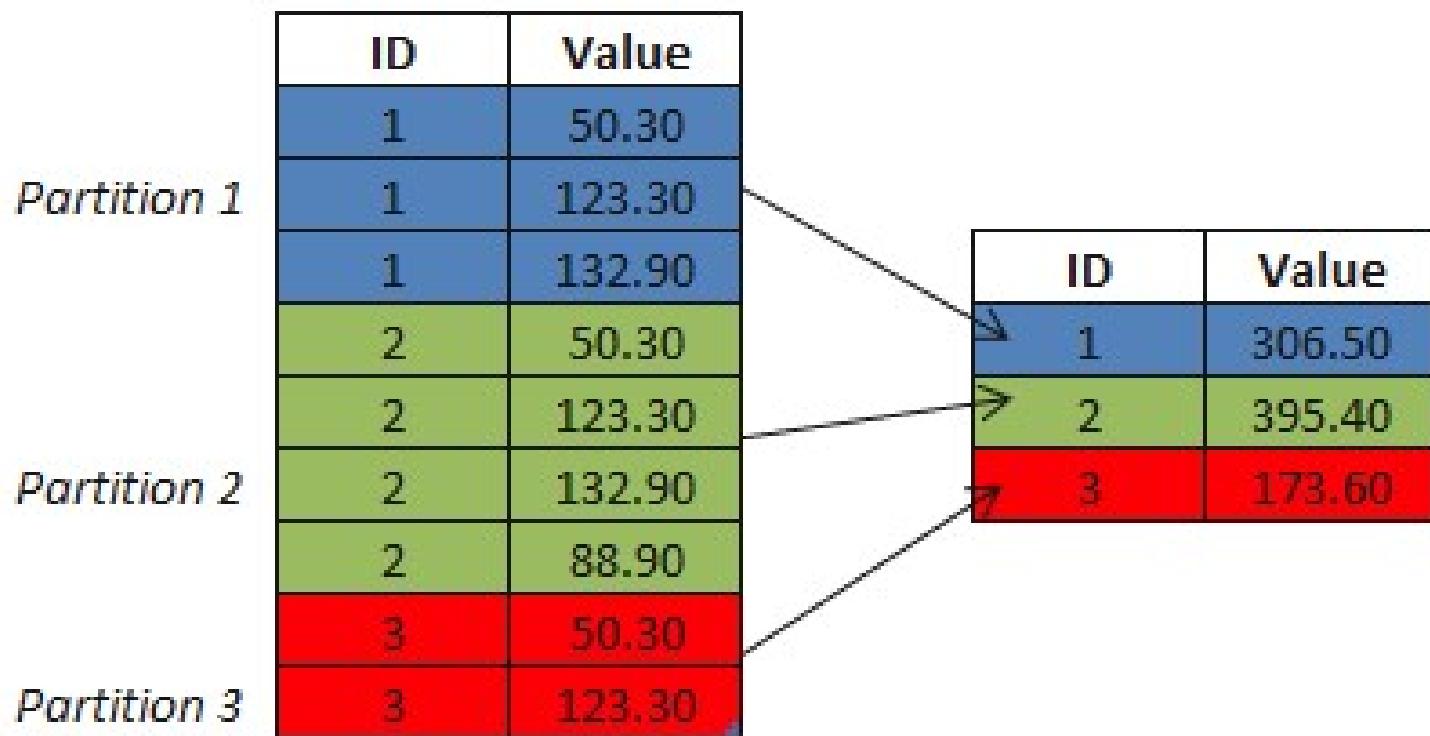
|   | ID | Value  | (No column name) |
|---|----|--------|------------------|
| 1 | 1  | 50.30  |                  |
| 2 | 1  | 123.30 |                  |
| 3 | 1  | 132.90 |                  |
| 4 | 2  | 50.30  |                  |
| 5 | 2  | 123.30 |                  |
| 6 | 2  | 132.90 |                  |
| 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 SQL Server Management Studio window with two tabs: 'Results' and 'Messages'. The 'Results' tab is selected and displays a table with three columns: ID, Value, and another unnamed column. The data consists of three rows, each containing values for ID, Value, and the unnamed column. The unnamed column is highlighted with a dashed border around its cell.

|   | 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 | 86.890.000  | 2        |  | 3  | 50.30  | 173.60 | 86.890.000  | 2        |
| 3  | 123.30 | 173.60 | 86.890.000  | 2        |  | 3  | 123.30 | 173.60 | 86.890.000  | 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

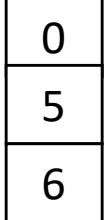
# Set-Comparison Operators

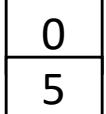
- SQL also supports op ANY and op ALL, where op is one of the arithmetic comparison operators {<,<=,=,<>,>=,>}
- SOME is also available, but it is just a synonym for ANY

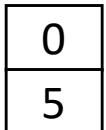
# Definition of Some (ANY) Clause

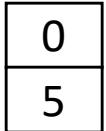
- $F <\text{comp}> \text{some } r \Leftrightarrow \exists t \in r \text{ s.t. } (F <\text{comp}> t)$

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

( $5 < \text{some}$   ) = true  
(read: 5 < some tuple in the relation)

( $5 < \text{some}$   ) = false

( $5 = \text{some}$   ) = true

( $5 \neq \text{some}$   ) = true (since  $0 \neq 5$ )

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

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

# Definition of All (ALL) Clause

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

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

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

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

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

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

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
- 1      18.00
- 2      19.00
- 3      10.00
- 4      22.00
- 5      21.35
- 6      25.00
- 7      30.00
- 8      40.00
- 9      97.00
- 10     31.00
- 11     21.00
- 12     38.00

- ProductID      AvgPrice
- 1      17.2434
- 2      17.5583
- 3      9.3902
- 4      20.8052
- 5      19.4669
- 6      24.4019
- 7      29.4416
- 8      36.9892
- 9      92.9157
- 10     29.8385
- 11     19.6912
- 12     37.4034

- UPDATE Products
- SET Products.UnitPrice =
- (SELECT POD.AvgPrice) FROM (
- SELECT OrderDetails.ProductID,  
SUM(OrderDetails.Quantity\*OrderDetails.UnitPri  
ce) / 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  
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 Coordinator      Laura 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.*

# Introduction

- in today's competitive environment, data (or information) and its efficient management is the most critical business objective of an organization
- database system is a tool that simplifies the above tasks of managing data and extracting useful information in a timely fashion
- databases have become integral component of our everyday life

# Data

- defined as known fact that can be recorded and that have implicit meaning
- raw or isolated facts from which required information is produced
- distinct pieces of information, usually formatted in special way
- binary computer representations of stored logical entities - can exist in variety of forms

# Data

- single piece of data represents single fact about something in which we are interested
- for an academic organization, it may be the fact that Calin CENAN employee (or social security) number is 106519, or that largest course attendance is at DataBases Course or that E-Mail address of one of key lecturer is Calin.Cenan@cs.utcluj.ro

# Data

- usually there are many facts (data) to describe something of interest to us
- data is also known as plural of datum, which means single piece of information
- in practice, data is used as both singular and plural form of word
- term is often used to distinguish machine readable information from human readable (textual) information

# Enterprise (organizational) data

- *Operational data* - stored in operational systems throughout organization (both internal and external) systems.
- *Reconciled data* - stored in data warehouse and data store
  - intended as single, authoritative source for all decision support applications
- *Derived data* - stored in each of data mart
  - limited and summarized data warehouse, selected, formatted and aggregated for end-user decision support applications

# Information

- <https://www.youtube.com/watch?v=nqwx2XFb1fQ>
- What's the difference ?
- One is my name. The other is not.

# Information

- is processed, organized or summarized data
- collection of related data that when put together, communicate meaningful and useful message to recipient who uses it, to make decision or to interpret data to get meaning
- data are processed to create information, which is meaningful to recipient
- *who is attending DataBases course*

# Data Warehouse

- collection of data designed to support management in decision-making process
- subject-oriented, integrated, time-variant, non-updatable collection of data used in support of management decision-making processes and business intelligence

# DataBase Data Warehouse

- student Floriana grade on DataBases topic
- student Floriana credits
- how many students have between 27 and 37 credits
- management decision
  - more examinations
  - no more examinations

# Data Warehouse

- present coherent picture of business conditions at single point of time
- unique kind of database, which focuses on business intelligence, external data and time-variant data
- process, where organizations extract meaning and inform decision making from their informational assets through the use of data

# Metadata

- also called *data dictionary* is data about data; also called *system catalog*
- data that describes objects in database and makes easier for those objects to be accessed or manipulated
- describes database structure, constraints, applications, authorization, sizes of data types, ...
- often used as integral tool for information resource management

# Data dictionary (also called information repositories)

- mini DataBase Management Systems that stores metadata, attribute names and definitions
- repository of information about database that documents data elements of database
- integral part of DataBase Management Systems (DBMS)
- aid database administrator in management, user view definitions as well as their use

# Data dictionary

- descriptions of schema
- information on physical database design, storage structures, file & record sizes
- database users, responsibilities & access rights
- descriptions of database transactions and relationships to users and data items referenced by them
- usage statistics such as frequencies of queries and transactions and access counts to different portions of database
- *relationships among entities*

# Data Item or Fields

- smallest unit of data that has meaning to its user, called also *data element*
- represented in database by value
- names, telephone numbers, bills amount, address, ...
- molecules of database (there are atoms and sub-atomic particles composing each molecule (bits and bytes)), but they do not convey any meaning so are of little concern to users
- may be used to construct other, more complex structures

# Record

- collection of logically related fields or data items, with each field possessing fixed number of bytes and having fixed data type - consists of values for each field
- occurrence of named collection of zero, one, or more than one data items or aggregates; data items are grouped together to form records
- grouping of data items can be achieved through different ways to form different records for different purposes
- are retrieved or updated using programs

# Files

- collection of related sequence of records
- in many cases, all records are of same record type (each record having an identical format)
- if every record has same size file is said to be *fixed-length records*
- if different records have different sizes, is said to be *variable-length records*

- data dictionary contains the following:
  - Entities
  - Attributes
  - Relationships
  - Keys

# Entities

- real physical object or an event; the user is interested in keeping track of
- any item about which information is stored
- for example Calin CENAN is real living person and employee of Technical University of Cluj-Napoca - is entity for which company (sinu) is interested in keeping track
- Dacia Logan model car CJ-10-SQL is real physical object and an entity
- employee owning car (and Access to Observatory parking) it is an event and it is another entity

# Entity Set

- collection of entities of same type, for example “all” of company’s employees
- record describes entity and file describes *entity set*

# Attributes

- property or characteristic (field) of an entity
- name, personal identification number (PIN = Cod Numeric Personal), position, ... all are his attributes
- height, weight, color of eyes, sense of humor are characteristics but not attributes (are not stored in database)
- values in all fields are attributes

# Relationships

- associations or ways that different entities relate to each other
- for example the relationship between employees and departments, Rodica POTOLEA is member of Computer Science dept. and Liviu MICLEA is member of Automation dept.

# Keys

- data item (or field) computer uses to identify record in database system
- single attribute or combination of attributes of entity set that is used to identify one or more entities, instances of set
- various types of keys:
  - Primary Key
  - Super key
  - Alternate Key

# Primary Key

- used to uniquely identify record - also called entity identifier, for example, PIN in EMPLOYEE file, MIN-NO in CAR file
- when more than one data item is used to identify record, it is called *concatenated key*, for example, both FirstName, LastName and Initial ...

# Super key

- includes any number of attributes that possess uniqueness property (not just the minimal number of such attributes like for PK)
- for example, if we add additional attributes to primary key, resulting combination would still uniquely identify an instance of entity set
- Primary Key is minimum super key

# Alternate Key

- or also called secondary key used to identify all records, which have certain property
- attribute or combination of attributes that classifies entity set on particular characteristic
- Primary Key it is an Alternate Key chosen by DBA to actually identify entities
- PIN vs. FirstName + LastName+ Initial vs. ...
- choose PIN because it is integer number & shorter

- Relationships could be of following types:
  - one-to-one (1:1) relationship
  - one-to-many (1:m) relationships
  - many-to-many (n:m) relationships
- 
- for example, one department have many faculty staff, it is a one-to-many (1:m) relationships

# DataBase

- database is defined as collection of logically related data stored together that is designed to meet the information needs of an organization
- organized in such a way that computer program can quickly select desired pieces of data

# DataBase

## can further be defined as

- collection of interrelated data stored together without harmful or unnecessary redundancy
- serves multiple computer program applications in which each user has his own view of data (data is protected from unauthorized access by security mechanism and concurrent access to data is provided with recovery mechanism)
- stores data independent of programs and changes in data storage structure or access strategy do not require changes in accessing programs or queries
- has structured data to provide foundation for growth and controlled approach is used for adding new data, modifying and restoring data

# DataBase

<https://en.wikipedia.org>

- organized collection of data, stored and accessed electronically

# DataBase

<https://en.wikipedia.org>

- DataBase Management System (DBMS) is software that interacts with end users, applications, and database itself to capture and analyze data; allows definition, creation, querying, update, and administration of databases
- database generally stored in DBMS-specific format
- often term "database" used to loosely refer to any of DBMS, database system or application associated

# Database Definition

- **database** - collection of data, typically describing activities of one or more related organizations
- **DataBase Management System**, or **DBMS** - software designed to assist in maintaining and using collections of data
- software that manages and controls access to database
- **database application** - program that interacts with database at some point in its execution

# Database

- **database** - collection of data, typically describing activities of one or more related organizations
- **database** - shared collection of logically related data and its description, designed to meet the information needs of an organization.

# Database example

- university database might contain information about:
  - entities such as students, faculty, courses, and classrooms
  - relationships between entities, such as students' enrollment in courses, faculty teaching courses, and the use of rooms for courses

# Database: Functional Definition

- Database is a stored data collection having the following characteristics:
  - assures **data independence**
  - assures **accesses** (possibly shared access) to large volumes of data

# DataBase Management System (DBMS)

- software designed to assist in maintaining and utilizing large collections of data
- software system that enables users to define, create, maintain, and control access to database
- software package used to store, manage, and analyze data

# DataBase Management System (DBMS)

- alternative to using a DBMS is to use ad hoc approaches that do not carry over from one application to another
  - store data in files and write application-specific code to manage it

# Database application

- computer program that interacts with database by issuing an appropriate request (typically an SQL statement) to DBMS
- **SQL – Structured Query Language** is common interface between language, technology and database
- Architecture
  - Client – Server
  - Web application

# Database Application Program

- maintain database and generate information
- written in programming language
- opposed to file-based approach, physical structure and storage of data are now managed by DBMS

# Database – Formal Definition

- A database is an *ordered collection of related data elements* intended to meet *the information needs* of an organization and designed to be (possible) *shared* by multiple users

# Ordered collection

- collection of data elements, not just a random assembly of data structures
- put together deliberately with proper order
- linked together in the most logical manner

# Related data elements

- not disjointed structures without any relationships among them
- related among themselves
- pertinent to particular organization

# Information needs

- collection of data elements in database is there for a specific purpose
- to satisfy and meet information needs of organization

# DataBase

- designed, built and populated with data for specific purpose
- has intended group of users and some preconceived applications in which users are interested
  - has some source from where data is derived
  - some degree of interaction with events in real world
  - audience that is actively interested in contents
- can be of any size and of varying complexity
- created & maintained by DataBase Management System

# DataBase consist of

1. Data items
2. Relationships
3. Constraints
4. Schema

# DataBase consist of

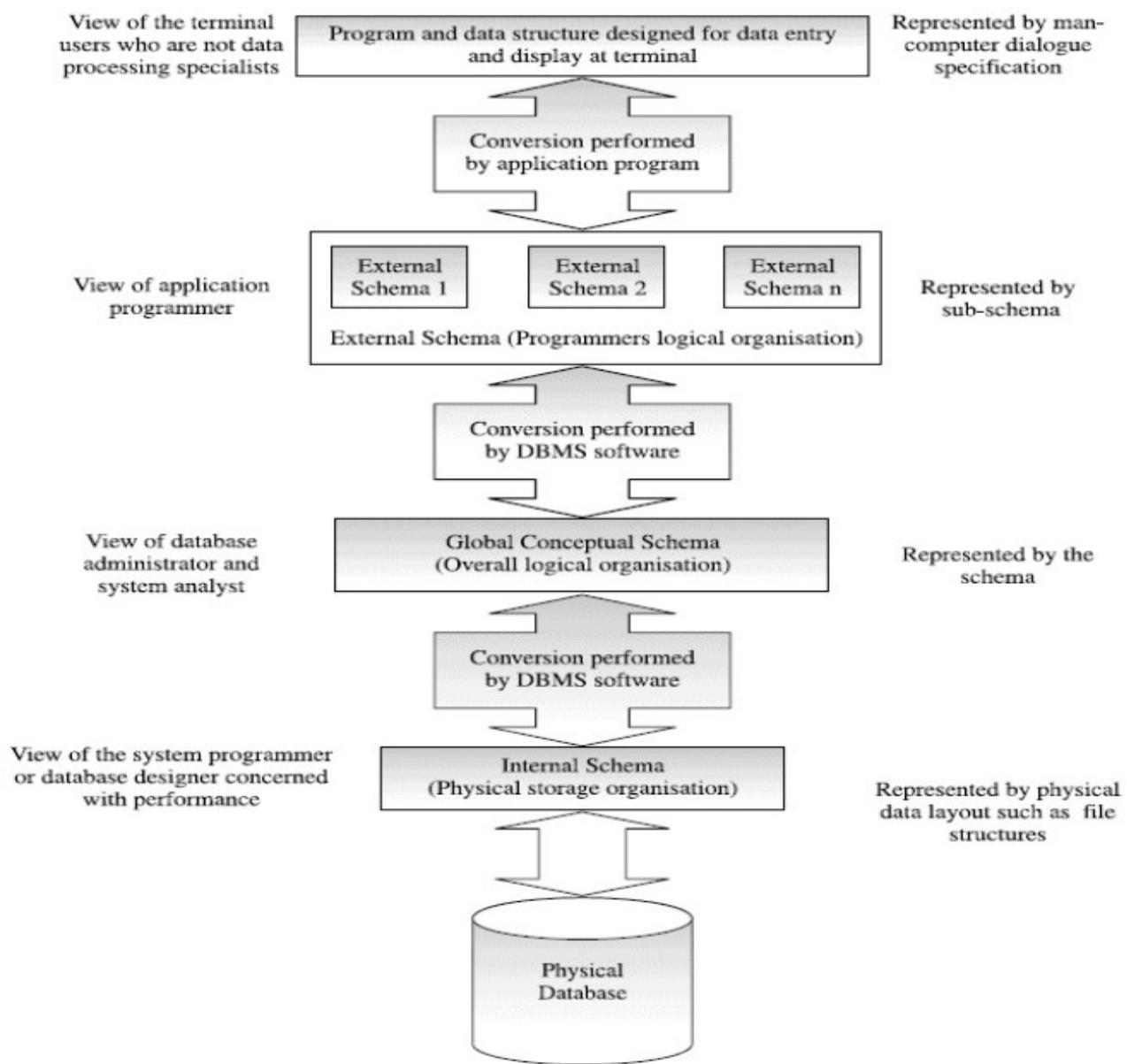
- *Data* (or *data item*) is distinct piece of information
- *Relationships* represent correspondence between various data elements
- *Constraints* are predicates that define correct database states
- *Schema* describes organization of data and relationships within database

# DataBase Schema

- defines various views of database for use of various system components of DBMS and for application security
- separates physical aspect of data storage from logical aspects of data representation

# DataBase three independent levels

1. Physical storage organization or internal schema layer
2. Logical organization or global conceptual schema layer
3. External schema layer for programmers' logical organization



# DataBase three independent levels

1. *internal schema* defines how and where data are organized in physical data storage
  2. *conceptual schema* defines stored data structure in terms of the database model used
  3. *external schema* defines view of database for particular users
- 
- DataBase Management System provides for accessing database while maintaining required correctness and consistency of stored data

# DataBase Management System

- also called database system, or database engine is a generalized software system for manipulating databases
- basically a computerized record-keeping system; which stores information and allows users to add, delete, change, retrieve and update that information (Create, Read, Update, Delete – CRUD operations)
- provides for simultaneous use of database by multiple users

# DataBase Management System

- also a collection of programs that enables users to create and maintain database; general purpose software system that facilitates process of defining (specifying data types, structures and constraints), constructing (storing data on media) and manipulating (querying to retrieve specific data, updating to reflect changes and generating reports from data) for various applications

# DataBase Management System

## typically has three components

1. Data description language (DDL) - allows users to define database: specify data types & data structures, constraints on data; translates schema into object schema, thereby creating logical and physical layout of database
2. Data manipulation language (DML) and query facility - allows users to insert, update, delete and retrieve data; provides general query facility through Structured Query Language
3. Software for controlled access of database - provides controlled access to database; preventing unauthorized user trying to access database, providing concurrency control to allow shared access, recovery control system to restore database to previous consistent state following failure, ...

# DataBase Administrator

- someone at senior level in organization understands data and needs
- whose job is to decide what data should be stored in database and establish policies for maintaining and dealing with that data
- what information is to be stored, identifies entities of interest, decides content of database at abstract level
- process known as *logical* or *conceptual database design* - overall understanding of system

# DataBase Administrator

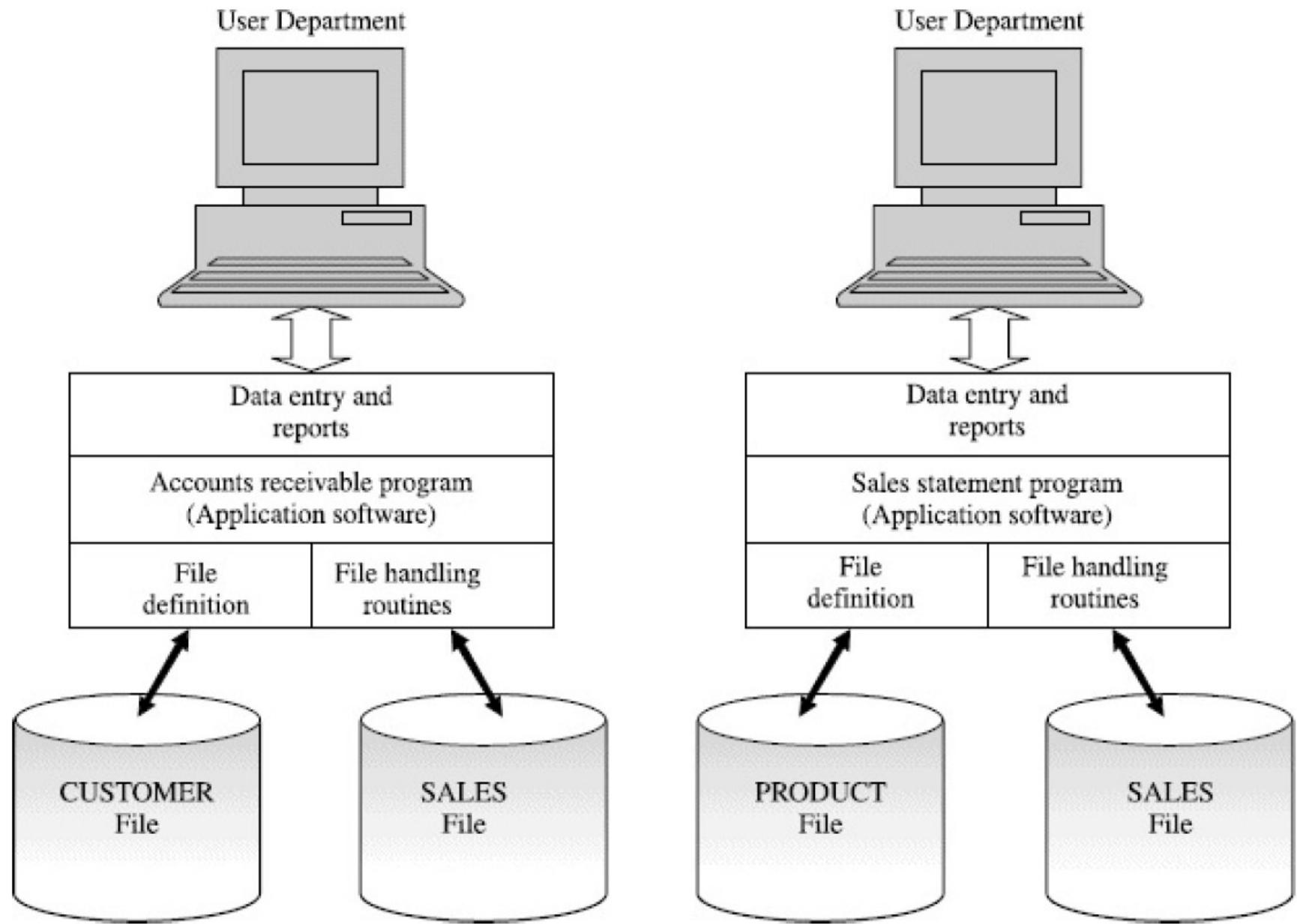
- provides necessary technical support for implementing policy decisions of databases; responsible for overall control of system at technical level; oversees and manages all resources; responsible for authorizing access, for coordinating and monitoring its use and for acquiring software and hardware resources as needed; accountable for security system, appropriate response time and ensuring adequate performance

# DataBase Administrator Functions and Responsibilities

- defining conceptual schema and database creation
- storage structure and access-method definition
- granting authorization to the users
- physical organization modification
- routine maintenance
- job monitoring

# FILE-ORIENTED SYSTEM VS. DATABASE SYSTEM

- File-oriented systems were an early attempt to computerize manual filing system
- systems performed normal record-keeping functions, they were called *data processing (DP) systems*
- with assistance of DP department, files were used for number of different applications by user departments
- with programs which accomplishes specific task of practical value in business (called *application program or software*), developed & designed to meet specific needs of particular requesting department or user group



- it can be seen from the above examples that there is significant amount of duplication of data storage in different departments

# Disadvantages of File-oriented System

- Data redundancy (or duplication)
- Data inconsistency (or loss of data integrity)
- Program-data dependence
- Inadequate data manipulation capabilities
- Excessive programming effort
- Limited data sharing
- Poor data control
- Security problems

# Data redundancy (or duplication):

- each department used their own independent application programs and special files of data; resulted into duplication of same data
- is wasteful and requires additional or higher storage space, costs extra time and money, and ***requires increased effort to keep all files up-to-date***

# Data inconsistency (or loss of data integrity)

- data redundancy also leads to *data inconsistency* (or loss of *data integrity*)
- since either data formats or values may be inconsistent, may no longer agree
- over period of time, such discrepancies can cause serious degradation in quality of information contained in data files and can also affect accuracy of reports and performance of business

# Program-data dependence

- physical structure, storage of data files and records are defined within each application program
- program contains detailed file description for files
- as a consequence, any change for file structure requires changes to file description for all programs that access file
- difficult to even locate all programs affected by such changes; subject to error when making changes

# Poor data control

- there is no centralized control at data element
- common for data field to have multiple names defined by various departments and depending on file
- could lead to different meanings in different context
- leads to poor data control, resulting in big confusion

# Limited data sharing

- each application has its own private files and users have little opportunity to share data outside their own applications
- to obtain data from several incompatible files in separate systems will require major programming effort

# Inadequate data manipulation capabilities

- file-oriented systems do not provide strong connections between data in different files and therefore its data manipulation capability is very limited

# Excessive programming effort

- high interdependence between program and data in file-oriented system
- new applications often requires number of other data fields that may not be available in existing file; programmer had to rewrite code for definitions for needed data fields from existing file as well as definitions of all new data fields
- each new application required that programmers essentially start from scratch by designing new file formats and descriptions and then write file access logic for each new program

# Security problems

- since applications programs are added to file-oriented system in an ad hoc manner, it was difficult to enforce such security system

# Data dependence

- physical structure and storage of data files and records are defined in application code
- changes to existing structure are difficult to make
- all programs that access file must be modified to conform to new file structure

# **DataBase Approach**

- previously mentioned limitations of file-based approach can be attributed to
- (1) definition of the data is embedded in application programs, rather than being stored separately and independently
- (2) there is no control over access and manipulation of data beyond that imposed by application programs

# Data Abstraction

- definition of data is separated from application programs - internal definition and users see only external definition
- can change internal definition without affecting users, provided that external definition remains
- if new data structures are added or existing structures are modified, then application programs are unaffected, provided that they do not directly depend upon what has been modified

# Driving Force for Database Systems

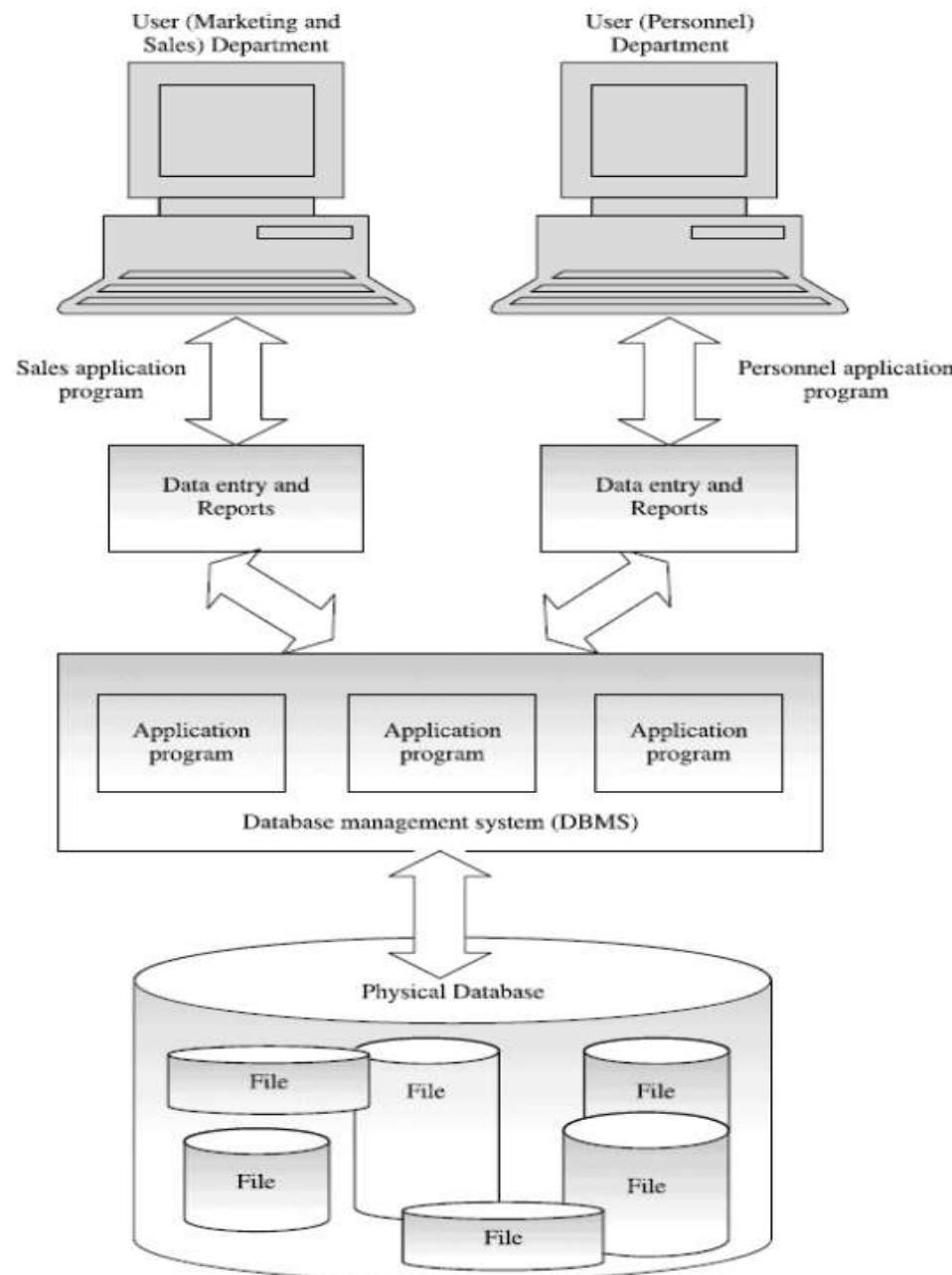
- Information as a Corporate Asset
  - *autovit.ro sell 4.7 mil. Euro*
- Explosive Growth of Computer Technology
- Escalating Demand for Information
- Inadequacy of Earlier Data Systems

# problems inherent in file-oriented systems make using database system very desirable

- Data redundancy (or duplication)
- Data inconsistency (or loss of data integrity)
- Program-data dependence
- Inadequate data manipulation capabilities
- Excessive programming effort
- Limited data sharing
- Poor data control
- Security problems

# database approach

- database system consists of logically related data stored in a single data dictionary
- change in way end user data are stored, accessed and managed
- emphasizes integration and sharing of data throughout organization
- overcome disadvantages of file-oriented system; eliminate problems related with data redundancy and data control by supporting integrated and centralized data structure



# Database System Environment

- refers to organization of components that define and regulate collection, storage, management and use of data within database
- consists of:
  1. Data
  2. Hardware
  3. Software
  4. Users (People)

# Data

- most important component of database system
- database are both *integrated* and *shared* in system ( users can access same piece of data concurrently (at same time))

# Hardware

- physical devices, servers and client computers

# Software

- application programs together with operating system
- DataBase Management System (DBMS) - comprises all requests from users to access database
- application software to solve specific common problems
- written in programming language such as C, C++, Visual Basic, Java, COBOL, Ada, Pascal, php, Python, using SQL embedded

# Users

- people interacting with database system
  - DataBase Administrators (DBAs)
  - database designers
  - application programmers who write database application programs
  - end users who interact with system

# DataBase Administrators

- database are corporate resources that must be managed like any other resource
- Data Administrator
- DataBase Administrator

# Data Administrators

- responsible for management of data resource, including database planning; development and maintenance of standards, policies and procedures; and conceptual/logical database design
- consults with and advises senior managers, ensuring that direction of database development will ultimately support corporate objectives

# DataBase Administrators

- responsible for physical realization of database, including physical database design and implementation, security and integrity control, maintenance of operational system, and ensuring satisfactory performance of applications for users
- DBA more technically oriented than role of DA
- requiring detailed knowledge of target DBMS and system environment
- in some organizations there is no distinction

# Database Designers

- in large database design projects, we can distinguish between two types of designer:
- logical database designers
- physical database designers

# Logical Database Designers

- concerned with identifying data, relationships and constraints
- must have thorough and complete understanding of organization's data and any constraints on this data (sometimes called **business rules**)
- these constraints describe main characteristics of data as viewed by organization

# Logical Database Designers

- must involve all prospective database users in development of data model
- split work into two stages:
- conceptual database design, which is independent of implementation details, such as target DBMS, application programs, program languages, or any other physical considerations
- logical database design, which targets specific data model, such as relational

# Physical Database Designers

- decides how logical database design is to be physically realized
- mapping logical database design into set of tables and integrity constraints
- selecting specific storage structures and access methods for data to achieve good performance
- designing any security measures required on data

# Application developers

- application programs to provide required functionality for end-users must be implemented
- work from specification produced by systems analysts
- programs contains statements that request DBMS to perform some operation on database
- programs written in programming language

# End-Users

- are the “clients” of database, which has been designed and implemented and is being maintained to serve their information needs; can be classified according to way they use the system:
- Naïve users
- Sophisticated users

# Naïve End-Users

- typically unaware of DBMS
- access database through specially written application programs that attempt to make operations as simple as possible
- invoke database operations by entering simple commands or choosing options from menu
- means that they do not need to know anything about database or DBMS

# Sophisticated End-Users

- familiar with structure of database and facilities offered by DBMS
- may use high-level query language such as SQL to perform required operations
- may even write application programs for their own use

# Advantages of DBMS

## due to centralized management and control

- Minimal data redundancy
- **Program-data independence**
- Efficient data access
- Improved data sharing
- Improved data consistency
- Improved data integrity
- Improved security
- Increased productivity of application development
- Enforcement of standards
- Economy of scale
- Balance of conflicting requirements
- Improved data accessibility and responsiveness
- Increased concurrency
- Reduced program maintenance
- Improved backup and recovery services
- Improved data quality

# Minimal data redundancy

- views of different user groups are integrated
- unnecessary duplication of data are avoided
- facts are ideally recorded in only one place in database
- data storage requirement is effectively reduced
- sometimes there are sound business and technical reasons for maintaining multiple copies of same data, for example, to improve performance, relationships and so on...
- in database system redundancy can be controlled - DBMS is aware of it, assumes responsibility for propagating updates and ensuring that multiple copies are consistent

# Program-data independence

- separation of data from application programs is called data independence
- allows for changes at one level of database without affecting other levels - changes are absorbed by mappings between levels
- allows organization's data to change and evolve (within limits) without changing application programs that process the data

# Efficient data access

- DBMS use variety of sophisticated techniques to store and retrieve data efficiently

# Improved data sharing

- data belonging to entire organization (all departments)
- existing application programs can share data
- new application programs can be developed on existing data - to share same data and add only data that is not currently stored
  - rather than having to define all data requirements again

# Improved data consistency

- corollary to redundancy
- when data is duplicated and changes made at one site are not propagated to other site, it results into inconsistency
- database supplies incorrect or contradictory information to its users
- redundancy is removed or controlled, chances of having inconsistency data is also removed and controlled - inconsistencies are avoided

# Improved data integrity

- means that data contained in database is both accurate and consistent
- integrity usually expressed in terms of *constraints*, which are consistency rules that database system should not violate
- database system ensures that adequate checks are incorporated
- $\text{Price} > 0$ ,  $1 \leq \text{Month} \leq 12$ , ensure that if there is reference to certain object, that object must exist
- for example, user is not allowed to transfer fund from nonexistent saving to checking account

# Improved security

- protection of database from unauthorized users
- proper (centralized) access procedure is followed

# Increased productivity of applications

- DBMS provides many of standard functions that application programmer would normally have to write in file-oriented application
  - provides all low-level file-handling routines
  - provide high-level environment consisting of productivity tools, such as forms and report generators, to automate some of activities of database design and simplify development of database applications
- results in increased productivity of programmer and reduced development time and cost

# Enforcement of standards

- with central control defines and enforces necessary standards
- standards can be defined for data formats to facilitate exchange of data between systems, naming conventions, display formats, report structures, terminology, documentation standards, update procedures, access rules and so on
- including business rules

# Economy of scale

- permits consolidation of data and applications
- thus reduces amount of wasteful overlap between activities of data-processing personnel in different departments
- combined low cost budget is required (instead of accumulated large budget)
- reduces overall costs of operation and management

# Balance of conflicting requirements

- knowing overall requirements of organization (instead of requirements of individual users), resolves conflicting requirements of various users and applications
- structure the system to provide overall service that is best for organization
- chose best structure and access methods to get optimal performance for response-critical operations, while permitting less critical applications to continue to use database (with relatively slower response)

# Improved data accessibility&responsiveness

- provide query languages and or report writers that allow users to obtain required information
  - without requiring programmer to write some software to extract this information from database

# Increased concurrency

- DBMSs manage concurrent databases access and prevents problems of loss of information or loss of integrity

# Reduced program maintenance

- in file-oriented environments, descriptions of data and logic for accessing data are built into individual application programs; as a result, changes to data formats and access methods inevitably result in need to modify application programs
- in database environment, data are more independent of application programs

# Improved backup&recovery services

- DBMS provides facilities for recovering from hardware or software failures through its backup and recovery subsystem
- database is restored to state it was before

# Improved data quality

- database system provides number of tools and processes to improve data quality

# HISTORICAL PERSPECTIVE OF DATABASE SYSTEMS

- during 1960s, US President, J.F. Kennedy initiated project “Moon Landing”
- project expected to generate large volume of data and there was no system available at that time - file-based system was unable to handle such voluminous data
- DataBase systems were first introduced during this time to handle such requirements

- North American Aviation (now known as Rockwell International), which was prime contractor for project, developed a software known as Generalized Update Access Method (GAUM) to meet voluminous data processing demand of project
- based on concept that smaller components come together as parts of larger components, and so on, until final product (man on moon) is assembled
- structure confirmed to up-down tree and was named ***hierarchical structure***

- in mid-1960s, the first general purpose DBMS was designed by Charles Bachman at General Electric, and was called *Integrated Data Store (IDS)* - formed basis for ***network*** data model
- Bachman was first recipient of computer science equivalent of Nobel Prize, called *Association of Computing Machinery (ACM) Turing Award*, for work in database area (1973)

- network data model was standardized by *Conference of Data Systems Languages (CODASYL)*, comprising representatives of US government and world of research, and business - it strongly influenced database systems
- CODASYL formed *Data Base Task Force (DBTG)* in 1967 to define standard specifications for an environment that would allow database creation and data manipulation

- IBM joined the North American Aviation to develop GAUM into what is known as *Information Management System (IMS)* DBMS, released in 1969
- More than 50 years later, old IBM slogan, “The world depends on it,” still holds true. Many of the largest corporations in the world rely on the system to run their everyday business. Indeed, more than 95 percent of the top Fortune 1000 companies use IMS to process more than 50 billion transactions a day and manage 15 million gigabytes of critical business data. And IBM continues to add new features to IMS to adjust to the changing IT world.

# Tale of Vern Watts IMS inventor

[http://vcwatts.org/ibm\\_story.html](http://vcwatts.org/ibm_story.html)

- from 1956, until 2009 Vern Watts worked on one single company, IBM, and only one single project, IMS database
- like ...?

# Tale of Vern Watts IMS inventor

[http://vcwatts.org/ibm\\_story.html](http://vcwatts.org/ibm_story.html)

- from 1956, until 2009 Vern Watts worked on one single company (IBM) and only one single project IMS database
- like ...?
- Er Bimbo de Oro (The Golden Boy), L'Ottavo Re di Roma (The Eighth King of Rome), Er Pupone (The Big Baby), Il Capitano (The Captain), and Il Gladiatore (The Gladiator), Francesco Totti spent his entire career at Roma

- IMS restricted to management of hierarchies of records to allow use of serial storage devices (magnetic tape were market requirement at that time)
- still the main hierarchical DBMS for most large mainframe computer installations
- SABRE system for making airline reservations was jointly developed by American Airlines and IBM around same time - today is being used to power popular web-based travel services

- hierarchical model - structured data as directed tree with root at top and leaves at bottom
- network model - structured data as directed graph without circuits, slight generalization that allowed to represent certain real-world data structures more easily
- during 1970s hierarchical and network DBMS were developed to cope with increasingly complex data structures that were extremely difficult to manage with conventional file processing methods
- approaches are still being used by organizations and are called *legacy systems*.

- drawbacks with hierarchical and network:
  - queries against data were difficult to execute, normally requiring program written by expert programmer who understood what could be a complex navigational structure of data
  - data independence is very limited so that programs are not insulated from changes to data formats
  - widely accepted theoretical foundation is not available

- in 1970, Edgar Codd, at IBM's San Jose Research Laboratory, wrote landmark paper (Codd, E. F. (1970). "A relational model of data for large shared data banks". Communications of the ACM. 13 (6): 377) proposing new data representation framework called ***relational data model*** and non-procedural ways of querying data
- received widespread commercial acceptance and diffusion during 1980s, sparked development of several DBMSs based on relational model (starting with Oracle ), along with rich body of theoretical results that placed database field on a firm foundation

- in relational model, all data are represented in the form of tables and simple language called *Structure Query Language (SQL)* is used for data retrieval
- simplicity of relational model, possibility of hiding implementation details completely from programmer and ease of access for non-programmers, solved major drawbacks of first-generation DBMSs
- Codd won 1981 ACM's Turing Award for his work

- relational model was not used in practice initially because of its perceived performance disadvantages and remained academically interesting for users
- in 1980s, IBM initiated *System R* project that developed techniques for construction of an efficient relational database system
- resulted in relational model consolidating its position as dominant DBMS paradigm

- this led to development of fully functional relational database product, called Structured Query Language / Database System
- in late 1980s, early 1990s, SQL was standardized and was adopted by American National Standards Institute (ANSI) and International Standard Organizations (ISO)

- now there are several relational DBMSs for both mainframe and PC environments for commercial applications, such as Ingress from Computer Associates, Informix, ORACLE, IBM DB2, Access and FoxPro from Microsoft, Paradox from Corel Corporation, InterBase from Borland and R-Base
- PostgreSQL – academic, evolved at the University of California, Berkeley in 1982
- MySQL, Microsoft SQL Server, Access

- in late 1990s, new era of computing started, such as client / server computing, data warehousing, and Internet applications
- advances were made in many areas of database systems
- complex data types and multimedia data (including graphics, sound images and video) became increasingly common; an *object-oriented database (OODBMS)* and *object-relational databases (ORDBMS)* were introduced during this period to cope with these

- the late 1990s also saw rise of Internet, three-tier client-server architecture, and demand to allow corporate databases to be integrated with Web applications
- the late 1990s saw development of **XML (eXtensible Markup Language)**, which has had a profound effect on many aspects of IT
- XML 1.0 ratified by W3C - XML becomes integrated with DBMS products and native XML databases are developed

- + 2000 DB vendors provide for newer data types
- Spatial databases introduced
- Databases support ERP (Enterprise Resource Planning), DW (Data Warehouse), DM (Data Mining) applications
- Big Data
- No SQL

- in response to increasing complexity of database applications “new” systems have emerged
- object-oriented DBMS and object relational DBMS
- Oracle (Java)
- MS SQL Server (.NET)
- new generation NoSQL DBMS

# Data Models

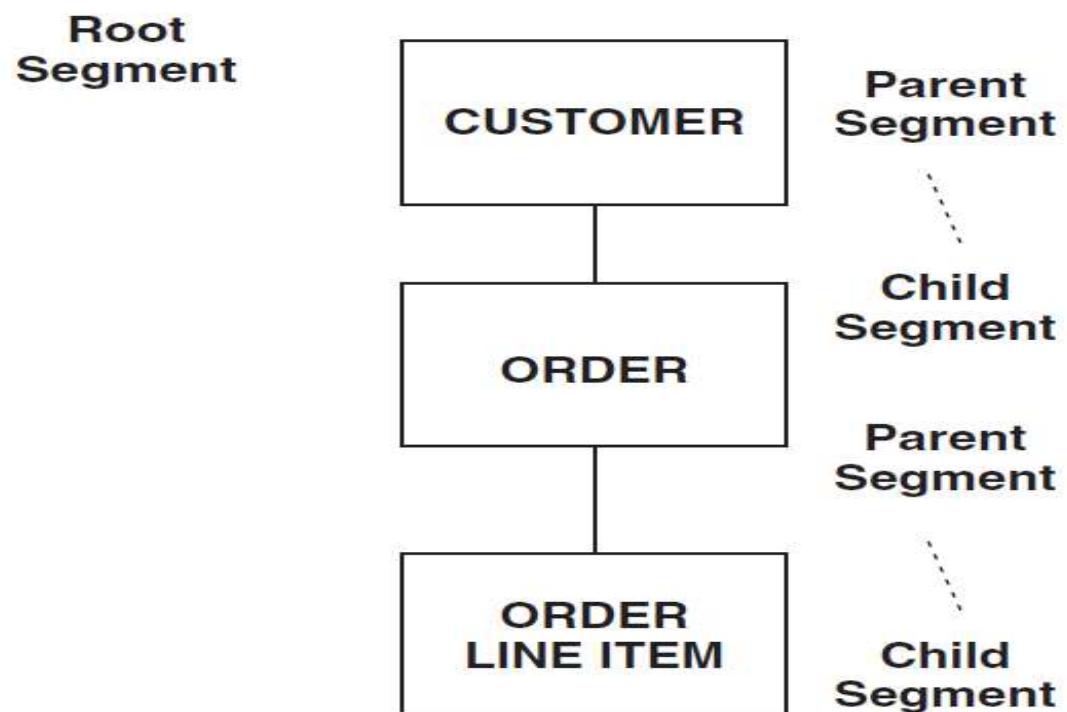
# Data Models

- represents data requirements of an organization
- can diagrammatically show data model with symbols and figures
- data for an organization reside in database
- when designing database - first create data model
- would represent real-world data requirements
- would show arrangement of data structures

# Data Models

- Hierarchical
- Network
- Relational
- Object – Relational

# Hierarchical



**Relationship links  
through physical  
pointers**

# Hierarchical

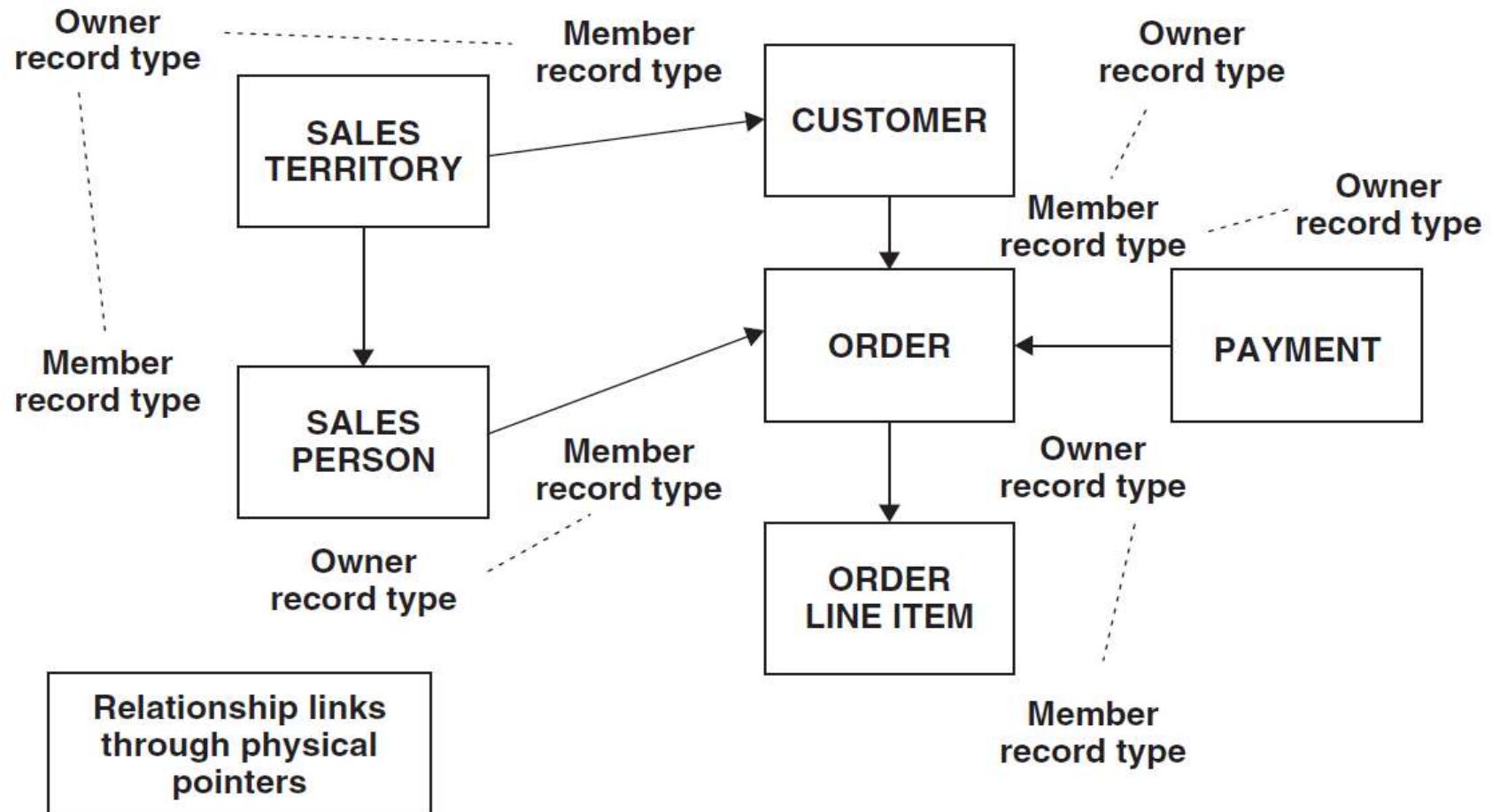
- *Levels* - each data structure representing business object is at one of hierarchical levels
- *Parent-Child Relationships* - relationship between each pair of data structures at levels next to each other is a parent-child relationship
- CUSTOMER is parent data segment whose child is ORDER data segment
- to separate orders into phone orders and mail orders - CUSTOMER may have PHONE ORDER and MAIL ORDER as two child segments

# Hierarchical

- *Root Segment* - data segment at top level of hierarchy is known as root data segment (as in an inverted tree)
- *Physical Pointers* - orders of particular customer linked in implementation of hierarchical data model by means of physical pointers or physical storage addresses embedded in database
- forward or backward pointers / physical pointers link parent – child and records of same segment type

- hierarchical data model represents well any business data that inherently contains levels one below other
  - in real world, most data structures do not conform to hierarchical arrangement
- customers placing orders and making payments, salespersons being assigned, and salespersons being part of sales territories – data elements cannot be arranged in hierarchy
- relationships cross over among data elements as though they form a network

# Network



# Network

- *Levels* - no hierarchical levels exist in network; lines in network data model simply connect appropriate data structures wherever necessary without restriction of connecting only successive levels as in the hierarchical model
- *Record Types* each data structure is known as a record type
  - CUSTOMER record type represents data content of all customers
  - ORDER record type represents data content of all orders

# Network

- *Relationships* - expresses relationships between two record types by designating one as the owner record type and the other as the member record type
- each member type with its corresponding owner record type is known as set
- set represents relationship between an owner and a member record type

# Network

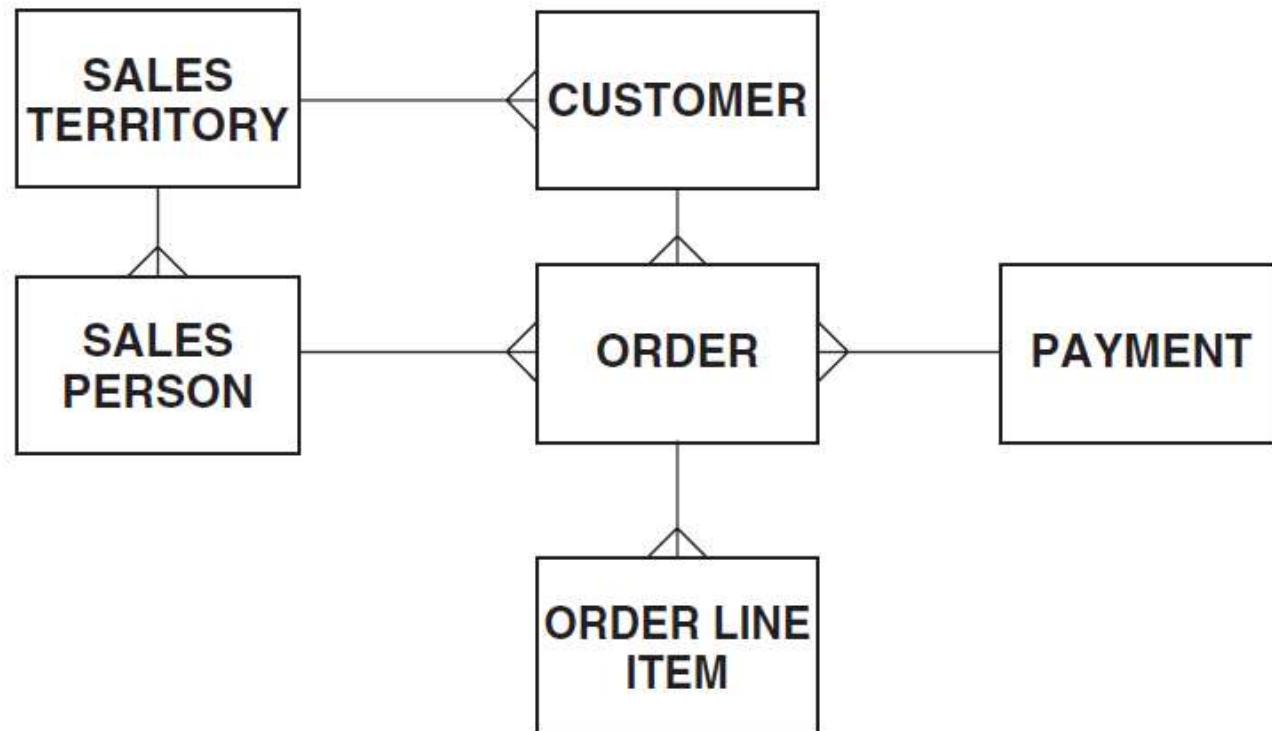
- *Multiple Parents* - for ORDER there are two parents or owner records, namely, CUSTOMER and PAYMENT
- for one occurrence of CUSTOMER, one or more occurrences of ORDER exist
- for one occurrence of PAYMENT there are one or more occurrences of ORDER
- by definition, hierarchical data model cannot represent this kind of data arrangement with two parents for one child data structure

# Network

- *Physical Pointers* – related occurrences of two different record types in network model are connected by physical pointers or physical storage addresses embedded within database
- physical pointers link occurrences of an owner record type with corresponding occurrences of member record type
- within each record type itself individual occurrences may be linked to one another by means of forward and backward pointers

- in both of these models need physical pointers to connect related data occurrences
- have to rewrite physical addresses in data records every time you reorganize data, move data to different storage
- relational model establishes connections between related data occurrences by means of logical links implemented through foreign keys

# Relational



**Relationship links NOT through physical pointers, but by foreign keys**

# Relational

- *Levels* - no hierarchical levels are present; lines simply indicate relationships between appropriate data structures wherever necessary without restriction of connecting only successive levels (as in hierarchical)
- *Relations or Tables* - model consists of relations; a relation is two-dimensional table of data observing relational rules - for example:
  - CUST relation represents data content of all customer
  - ORDER relation represents data content of all orders

# Relational

- *Relationships* - consider relationship between CUSTOMER and ORDER
- for each customer one or more orders may exist
- customer occurrence must be connected to all related order occurrences
- foreign key field included in ORDER data structure
- in each of order occurrences relating to certain customer, foreign key contains identification of that customer

# Relational

- *Relationships* - consider relationship between CUSTOMER and ORDER
- when look for all orders for particular customer, you search through foreign key field of ORDER and find those order occurrences with identification of that customer in foreign key field
- *No Physical Pointers* - unlike hierarchical or network, relational model establishes relationships between data structures by means of foreign keys and not by physical pointers

# Relational

- *Relationships* - consider relationship between CUSTOMER and ORDER
- when look for all orders for particular customer, you search through foreign key field of ORDER and find those order occurrences with identification of that customer in foreign key field

# Object Relational

- demand for information continues to grow, organizations need database systems that allow representation of complex data types, user-defined sophisticated functions, and user-defined operators for data access.
- present viable solutions for handling complex data types; combines ability of object technology to handle advanced types of relationships with features of data integrity, reliability, and recovery found in relational

# DATABASE LANGUAGES

- Data definition language (DDL)
- Storage definition language (SDL)
- View definition language (VDL)
- Data manipulation language (DML)
- Programming language, with embedded SQL
- Transactions

- in practice, there are not separate languages, instead they simply form parts of single database language and comprehensive integrated language is used such as widely used Structured Query Language (SQL)
- SQL represents combination of these, as well as statements for constraints specification

# Data Definition Language (DDL)

- used to specify database conceptual schema
- supports definition of database objects (or data element)
- describe and name entities, attributes and relationships
  - together with associated integrity and security constraints
- theoretically, different DDLs are defined for each schema in three-level schema-architecture (for conceptual, internal and external schemas)

# Data Storage Definition Language (DSDL)

- used to specify internal schema in database
- mapping between conceptual schema (as specified by DDL) and internal schema (as specified by DSDL) may be specified in either one of these languages
- storage structure and access methods used by database system is specified
- define implementation details of database schemas, which are usually hidden from user

# View Definition Language (VDL)

- used to specify user's views (external schema) and their mappings to conceptual schema
- in most of DBMSs, DDL is used to specify both conceptual and external schemas
- there are two views of data:
  - one is *logical view* of data - form that programmer perceives to be in
  - other is *physical view* - reflects way that data is actually stored on disk (or other storage devices)

# Data Manipulation Language (DML)

- is mechanism that provides set of operations to support basic data manipulation operations on data held in database
- used to retrieve data stored in database, express database queries and updates - communicate with DBMS
- part of DML that provides data retrieval is called *query language*

[https://ro.wikipedia.org/wiki/  
Daniela\\_Crudu Educație](https://ro.wikipedia.org/wiki/Daniela_Crudu_Educație)  
Universitatea Spiru Haret (2009-2012)

- Create - add (or insert) records to database
- Read - retrieve data and/or records from database
- Update - modify data and/or record in database
- Delete - delete records from database

# Transaction managements

- all work that logically represents single unit is called *transaction*
- sequence of database operations that represents logical unit of work is grouped together as single transaction and access database and transforms it from one state to another

# Transaction managements

- when DBMS does '*commit*', changes made by transaction are made permanent
- if changes are not be made permanent, transaction can be '*rollback*' and database will remain in its original state

# Transaction managements

- when updates are performed on database, we need some way to guarantee that set of updates will succeed all at once or not at all
- transaction (BEGIN TRANSACTION ... END TRANSACTION) ensures that all work completes or none of it affects database
- necessary in order to keep database in consistent state
- for example, transaction might involve transferring money from bank saving to checking account

# Transaction

- has four properties, called *ACID*:
  - Atomicity
  - Consistency
  - Isolation
  - Durability

# Transaction

- *Atomicity* means that either all work of transaction or none of it is applied
- other operations can only access any rows involved transactional access either before transaction occurs or after transaction is complete, but never while transaction is partially complete
- *Consistency* means that transaction's work will represent correct (or consistent) transformation of database's state
- *Isolation* requires that transaction not to be influenced by changes made by other concurrently executing transactions
- *Durability* means that work associated with successfully completed transaction is applied to database and is guaranteed to survive system or media failures



# Database Need

- amount of information available exploding
- value of data as organizational asset is widely recognized
  - ability to manage this vast amount of data
  - quickly find information relevant
- need for increasingly powerful and flexible data management systems

# Database Importance

- there is not real application without a kind of database
- great number of DBMS packages can be found on software market, for all types of computers and processing technologies
- DBMS can be found at the top 3 of most needed, requested, sold and used products

# Mathematics

- always strive for elegance and orthogonality - dislike exceptions
- system is said to be designed in an *orthogonal* way if its set of components that together make up whole system capability are *non-overlapping* and *mutually independent*
  - each capability should be implemented by only one component
  - one component should only implement one capability of system

# Mathematics

- well-separated and independent components ensure that there are no side effects: using or even changing one component does not cause side effects in another area of system
- formal disciplines, most relevant ones in application of mathematics to field of databases
- Set theory, Relation theory

# *Mathematics*

- *Everything should be made as simple as possible, but not simpler.*
  - Albert Einstein (1879–1955)

# Relational model

- Professionals in any discipline need to know the foundations of their field
- Database Professional - need to know theory of relations
- is not product-specific;
  - rather, it is concerned with principles

# Principles, not Products

- principle
  - source, root, origin; which is fundamental; essential nature; theoretical basis
- principles endure
- by contrast, products and technologies, change all the time

- E. F. Codd, at the time researcher at IBM; late in 1968 that Codd, a mathematician, first realized that discipline of mathematics could be used to inject some solid principles and rigor into field of database

# original definition of relational model

- E. F. Codd
  - *Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks*, IBM Research Report RJ599, 1969.
  - *A Relational Model of Data for Large Shared Data Banks*, CACM 13, No. 6, 1970.
- C. J. Date
  - *An Introduction to Database Systems*, 8<sup>th</sup> Ed. Boston, Mass.: Addison-Wesley, 2004.
  - first edition of book was published in 1975

# Database Development Process

# Database Design Paradigm Shift

- structure of database is determined during **database design**
- to produce system that will satisfy organization's information needs requires different approach from that of file-based systems, where work was driven by application needs of individuals
- for database approach to succeed, organization now has to think of data first and application second – this change in approach is referred to as *paradigm shift*

# Database Design Paradigm Shift

- system to be acceptable to end-users, database design activity is crucial
- poorly designed database will generate errors that may lead to bad decisions, which may have serious repercussions for organization
- well-designed database produces system that provides correct information for decision making process to succeed in an efficient way

- before database can be built, must understand how database will achieve its goals; verify that database will solve business problems
- design is necessary
- well-designed database is based on good **understanding of business process**
- first step in development of database is to identify business requirements and goals

- What are the business requirements for future database?
- What information needs to be stored in database?
- How can the data be stored?
- How can the data be presented to end users?

- next step is to **understand relationships**
- database designers translate data model to database tables with integrity constraints: specify tables and columns to organize business data into a well-defined **database structure**

- designers can use data model to verify if future database will meet database requirements
- usually, several modifications are needed
- might take some time to make sure that database design and business partners are all satisfied

- next step **implement database** with DBMS package; choose proper DBMS
- database designers or database administrators (DBAs) **create database** tables and populate them with business data
- test database; further modify database to meet users' requirements

- once database is ready database application developers **create database application** objects to assist users in accessing database
- database should enforce necessary **security** measures

- for large applications database must be partitioned into multiple parts, and **partition database** distributed to multiple computers
- to support business decision making, database should also provide **data analysis tools**, such as OLAP, data mart or data warehouse, and data mining
- analyzer can store history data for fast information search and identify trend and patterns hidden in daily business operation data

# Type of DataBases

# Type of Databases

- *How Databases Are Used*
  - Production Database
  - Decision Support System
  - Mass Deployment

# Production Database

- Support business functions
- Online transaction processing
- Usage includes CRUD activities (Create, Read, Update, Delete)
- Features include concurrency, security,
- transaction processing, security; OnLine Transaction Processing (OLTP) systems

# Decision Support System

- Used for analysis, querying, and reporting
- Generally read-only
- Features include query tools and custom applications
- Data warehouse and OnLine Analytical Processing (OLAP) systems

# Mass Deployment

- Intended for single user environments
- Workstation / mobile versions of database products
- Ease of use important
- Features include report and application generation capabilities

# Type of Databases

- *Where Data Are Used*
  - Centralized database
  - Distributed database
    - Homogeneous databases
    - Heterogeneous databases

# DataBase Market

# DataBase Market

- the five leading commercial relational database vendors by revenue
- Oracle (48.8%)
- IBM (20.2%)
- Microsoft (17.0%)
- SAP including Sybase (4.6%)
- Teradata (3.7%)

# DataBase Market

- the three leading open source implementations are MySQL, PostgreSQL, and SQLite
- MariaDB is a prominent fork of MySQL prompted by Oracle's acquisition of MySQL AB

# DataBase Market

- the percentage of database sites using any given technology were (site may deploy multiple)
- Oracle Database - 70%
- Microsoft SQL Server - 68%
- MySQL (Oracle Corporation) - 50%
- IBM DB2 - 39% IBM Informix - 18%
- SAP Sybase Adaptive Server Ent. - 15%
- SAP Sybase IQ - 14%
- Teradata - 11%

# Review Questions

# Review Questions

- list five government sectors that use database systems
- discuss each of the following terms:
  - data; database
  - database management system
  - database application program
  - data independence
  - security; integrity; views
- describe role of DBMS database approach

# Review Questions

- discuss why knowledge of DBMS is important for DBA
- describe main characteristics of database approach and contrast it with file-based
- describe five components of DBMS environment and discuss how they relate to each other
- discuss the three generations of DBMS

# Review Questions

- discuss roles of following personnel in database environment:
  - data administrator
  - database administrator
  - logical database designer
  - physical database designer
  - application developer
  - end-users
- why are views important aspect of database

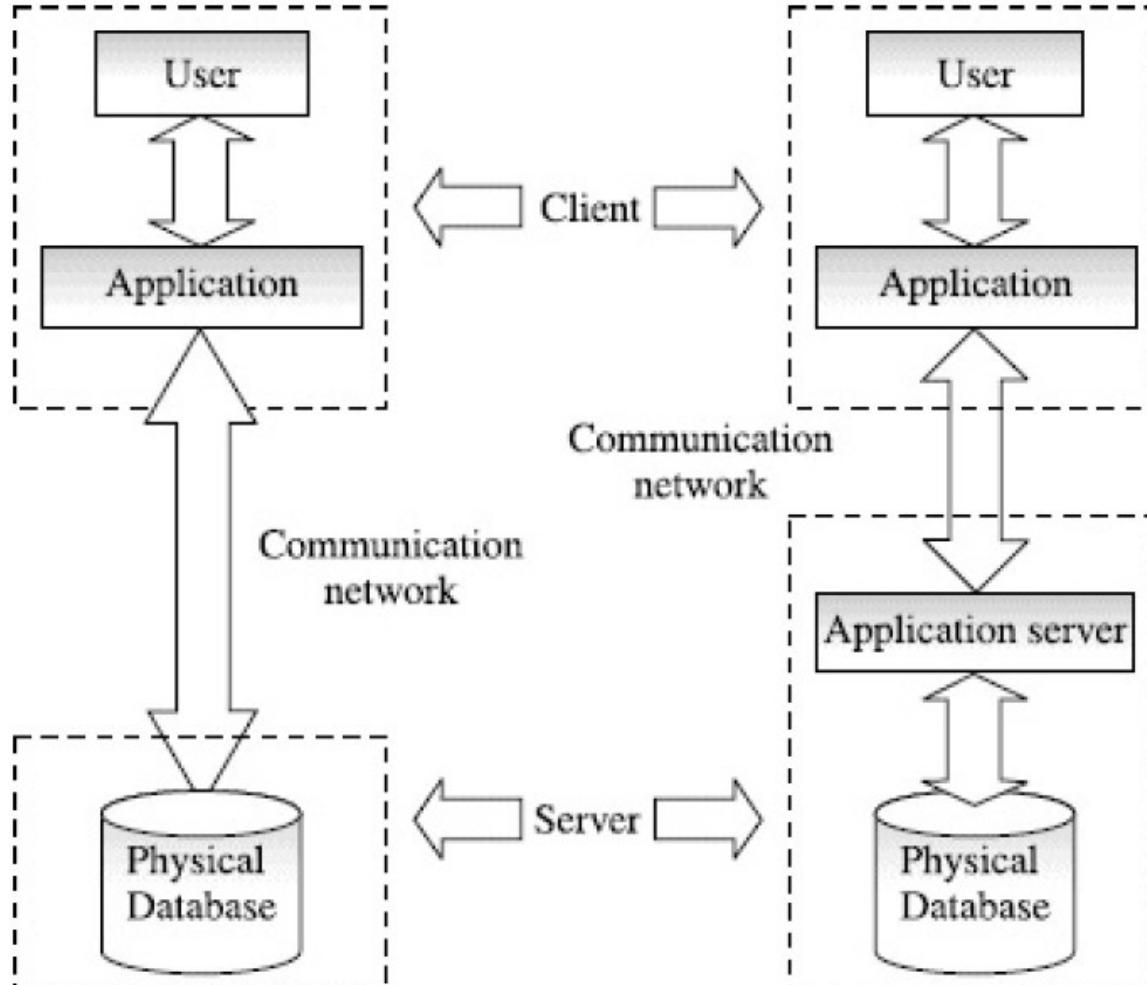
# Review Questions

- Why is business process information important?
- Describe functions of database in business process
- What is metadata?
- Explain functions of ODBC and ADO
- Name commonly used database components
- Describe tables and their structures

- 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.*

# Database System Architecture

# DataBase System Architecture



- organization requires accurate&reliable data -> maintains records for its varied operations by building appropriate database models (capturing essential properties of objects and relationship)
- users look for abstract view of data they are interested in
- since database is shared resource, each user may require different view of data
- we need to develop architecture for database systems - framework in which structure is described

- when database is designed to meet information needs of organization *schema* of database becomes most important concern
- data in changes frequently, while schema remain the same over long time
- Schema consist of types of entities and relationships among these
- DBMS should do the translation between logical (users' view) organization and physical organization of data in database

# Schema

- gives the names of entities and attributes
- specifies relationship among them
- database can have several schemas  
partitioned according to levels of abstraction

# Logical / Physical Schema

- logical schema - concerned with exploiting data structures, to make scheme understandable to computer; important as programs use it to construct applications
  - database definition language (DDL)
- physical schema – deals with manner in which conceptual database shall get represented in computer as stored database; hidden beneath logical schema and can usually be changed easily without affecting application programs
  - database storage definition language (DSDL)

# Subschema

- subset of schema and inherits same property
- refers to application programmer's (user's) view of data
- defines portion of database as “seen” by application programs

- different application programs can have different view of data; individual application programs can change their respective subschema without effecting subschema views of others
- application programs are not concerned about physical organization of data; physical organization of data in database can change without affecting application programs
- subschemas also act as unit for enforcing controlled access to database

# Instances

- when schema framework is filled in the data item values (contents of database at any point) referred to as an *instance* of database
- also called state of database or snapshot

# Three-Level ANSI-SPARC DataBase Architecture

- in 1971, Database Task Group (DBTG) appointed by Conference on Data Systems and Languages (CODASYL), produced a proposal for general architecture for database systems
- two-tier architecture with system view called schema and user views called subschemas
- in 1975, ANSI-SPARC (American National Standards Institute — Standards Planning and Requirements Committee) produced three-tier architecture with system catalog
- architecture of most commercial DBMSs available today is based to some extent on this proposal

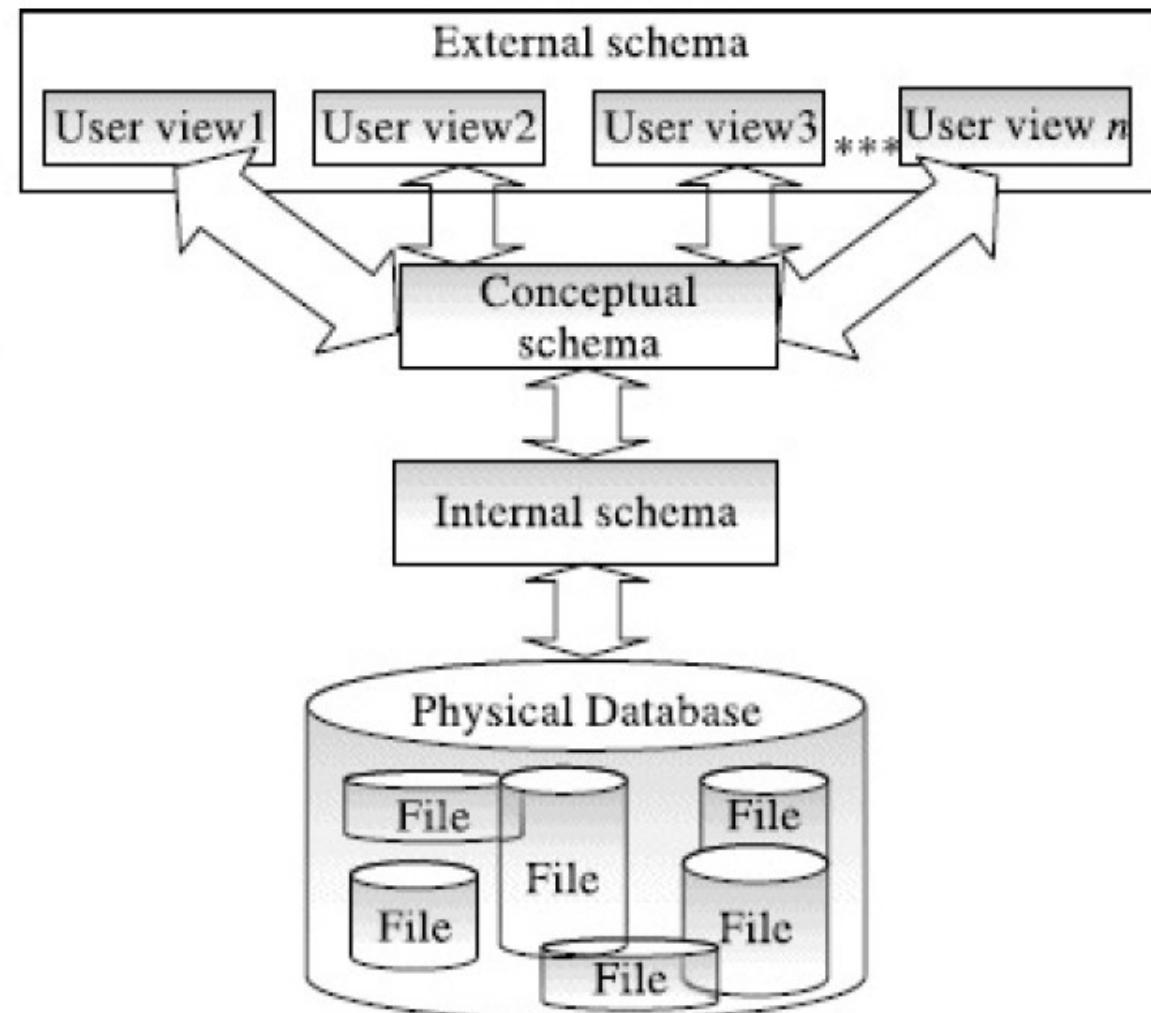
# Three-Level ANSI-SPARC DataBase Architecture

External level  
(defined by user or application program)

Conceptual level  
(defined by DBA)

Internal level  
(defined by DBA)

Physical level



# Three-Level ANSI-SPARC DataBase Architecture consists of

1. Internal level
2. Conceptual level
3. External level

- data abstracted in three levels corresponding to three views (namely internal, conceptual and external views)
- lowest level of abstraction of data contains description of actual method of storing data
- third level is highest level of abstraction seen by user or application program

# Internal Level

- physical representation of database on computer
- indicates how data will be stored
- describes data and file structures (definition of stored records, method of representing data fields), indexing and hashing schemes and access methods to be used on storage devices
- covered by internal level to achieve routine performance and storage space utilization

# Internal Level concerned with

- storage space allocation for data
- record descriptions for storage with stored sizes for data items
- record placement
- data compression and data encryption techniques

# Physical DataBase Design

- process arriving at good internal (or physical) schema

# Conceptual Level

- all database entities and relationships
- provides community view of database, as seen by DBA
- *conceptual schema* defines conceptual view also called *logical schema*
- contains method of deriving objects in conceptual view from objects in internal view
- supports each external view, in that any data available to user must be derived from conceptual level
- level must not contain any storage-dependent details

# Conceptual Level concerned with

- all entities, their attributes and their relationships
- constraint on data
- semantic information about data
- checks to retain data consistency and integrity
- security information

# Conceptual DataBase Design

- process of arriving at good conceptual schema
- choice of relations (entities) and field (or data item) for each relation, is not always obvious

# External Level

- is user's view of database, portions of concern to user or application program are included in form that is familiar for that user
- any number of user views, even identical, may exist for given conceptual view of database
- includes only those entities, attributes and relationships that user is interested in
- different views may have different representations of same data

# External Level

- *external schema* describes each external view
- consists of definition of logical records and relationships in external view
- also contains method of deriving objects
- allow data access to be customized at level of individual users or groups of users
- given database has exactly one internal or physical schema and one conceptual schema but, it may have several external schemas, each tailored to particular group of users

# Advantages of Three-tier Architecture

- main objective is to isolate each user's view of database from the way database is physically stored or logically represented
- each user is able to access same data but have different customized view as per their own needs; can change way views data and this change does not affect other users
- user is not concerned about physical data storage details - user's interaction with database is independent of physical data storage

# Advantages of Three-tier Architecture

- internal structure of database is unaffected by changes to physical storage organization, such as changeover to new storage device
- database administrator (DBA) is able to change conceptual structure and database storage structures without affecting user's view

# Data Independence

- major objective of implementing DBMS
- defined as immunity of application programs to change in logical and physical representation and access techniques
- characteristics of database system to change schema at one level without having to change schema at next higher level
- application programs do not depend on any one particular physical representation or access technique
- insulates application programs from changes in way data is structured and stored

# Data Independence

- Physical data independence
- Logical data independence

# Physical Data Independence

- immunity of conceptual (or external) schemas to changes in internal schema
- conceptual schema insulates users from changes in physical storage of data
  - such as using different file organizations or storage structures, using different storage devices, modifying indexes or hashing algorithms
- physical storage could be changed without necessitating change in conceptual view or any of external views; change is absorbed by conceptual/internal mapping

# Logical Data Independence

- immunity of external schemas (application programs) to changes in conceptual schema
- users shielded from changes in logical structure of data
  - such as addition and deletion of entities, addition and deletion of attributes, or addition and deletion of relationships, without changing existing external schemas (having to rewrite application programs)
- only view definition and mapping need be changed; application programs that refers to external schema must work as before, after conceptual schema undergoes logical reorganization

# mappings

- process of transforming requests and results between the three levels
- between internal, conceptual and external schemas
- Conceptual/Internal mapping
- External/Conceptual mapping

# Conceptual/Internal Mapping

- defines correspondence between conceptual view and stored database
- specifies how conceptual records and fields are presented at internal level
- enables DBMS to find actual records in physical storage that constitute logical record in conceptual schema, together with any constraints to be enforced on operations)

# Conceptual/Internal Mapping

- allows any differences in entity names, attribute names, attribute orders, data types, ... to be resolved
- in case of any change in structure of stored database, conceptual/internal mapping is also changed, so that conceptual schema can remain invariant
- effects of changes to database storage structure are isolated below conceptual level in order to preserve physical data independence

# External/Conceptual Mapping

- defines correspondence between external view and conceptual view
  - among records and relationships of external and conceptual views
- enables to map names in user's view to relevant part of conceptual schema
- there could be several mappings between external and conceptual levels
- conceptual/internal mapping is the key to physical data independence
- external/conceptual mapping is the key to the logical data independence

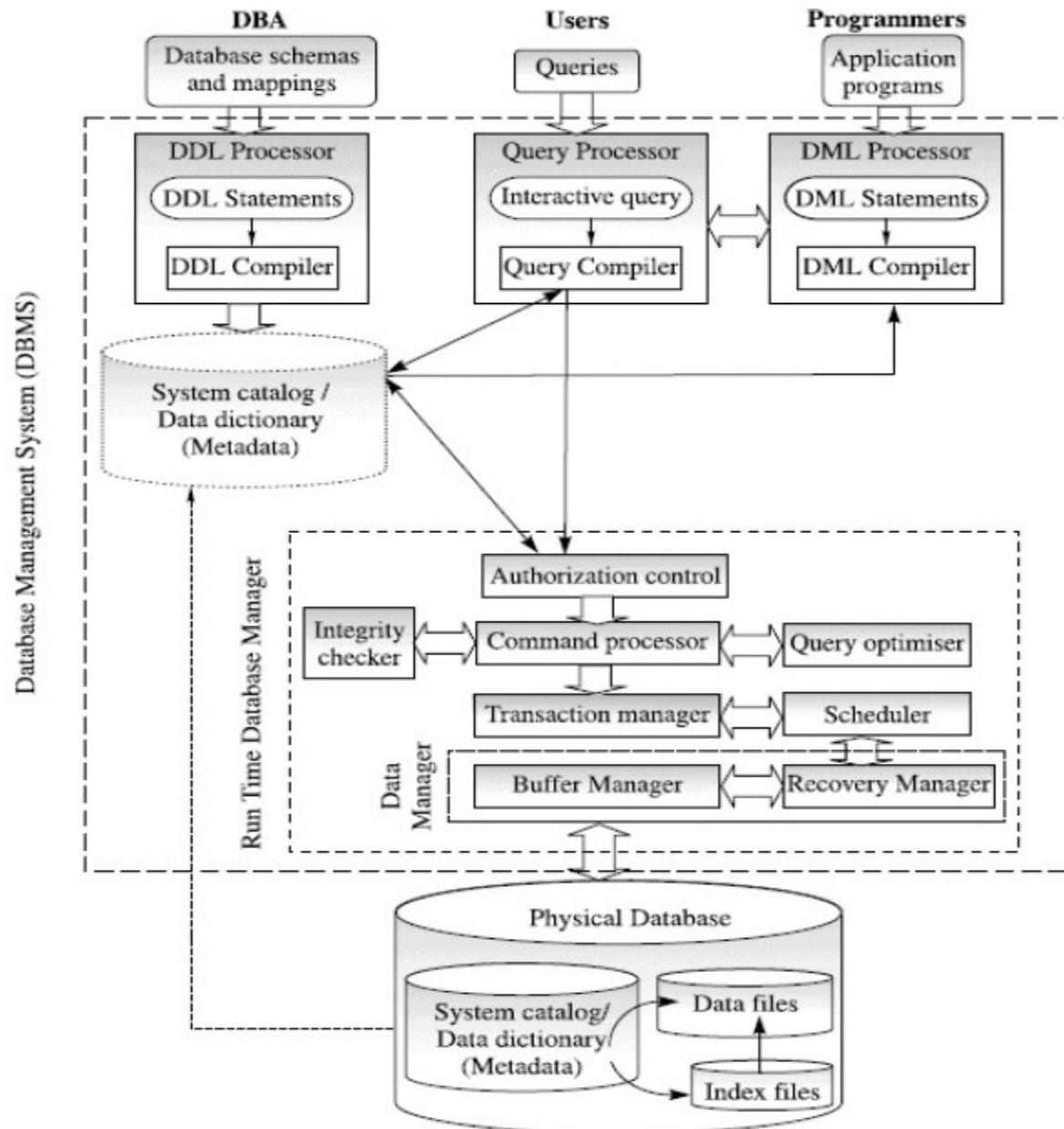
# External/Conceptual Mapping

- information about mapping are included in system catalog of DBMS
- when schema is changed at some level, schema at next higher level remains unchanged; only mapping between levels is changed
- data independence is accomplished

# Structure, Components, Functions of DBMS

# Structure of a DBMS

- typical structure of DBMS with components and relationships between them
- DBMS partitioned into several modules
- some of functions of DBMS are supported by operating systems (OS) to provide basic services
  - physical data and system catalog are stored on physical disk, access to disk is controlled primarily by OS, therefore, interface with OS must be taken into account



# Execution Steps of DBMS

1. users issue query using particular database language, for example, SQL commands
2. passed query is presented to query optimizer, which uses information about how data is stored to produce efficient execution plan for evaluating query
3. DBMS accepts users SQL commands and analyses them
4. DBMS produces query evaluation plans
  - external schema for user, corresponding external/conceptual mapping, conceptual schema, conceptual/internal mapping, storage structure definition
5. DBMS executes plans against physical database and returns answers to users

- DBMS supports concurrency and crash recovery by carefully scheduling users requests and maintaining log of all changes to database with components such as transaction manager, buffer manager, and recovery manager

# Components of DBMS

- Query processor
- Run time database manager
  - Authorization control
  - Command processor
  - Integrity checker
  - Query optimizer
  - Transaction manager
  - Scheduler
  - Data manager
- DML processor
- DDL processor

# Query processor

- transforms users queries into series of low-level instructions directed to run time database manager
- interpret online user's query and convert it into efficient series of operations in form capable of being sent to run time data manager for execution
- uses data dictionary to find structure of relevant portion of database and uses this information in modifying query and preparing optimal plan to access database

# Run time database manager

- central software component of DBMS
- interfaces with user-submitted application programs and queries
- handles database access at run time
- converts operations in user's queries coming directly or indirectly via application program from user's logical view to physical file system
- accepts queries and examines external and conceptual schemas to determine what conceptual records are required to satisfy users request
- places call to physical database to perform request
- enforces constraints to maintain consistency and integrity of data, as well as its security; also performs backing and recovery operations
- sometimes referred to as *database control system*

# Run time database manager / database control system components

- authorization control - checks that user has necessary authorization to carry out required operation
- command processor - processes queries passed by authorization control module
- integrity checker - checks for necessary integrity constraints for all requested operations that changes database
- query optimizer - determines optimal strategy for query execution; uses information on how data is stored to produce efficient execution plan for evaluating query

# Run time database manager / database control system components

- transaction manager - performs required processing of operations it receives from transactions; ensures that transactions request and release locks according to suitable locking protocol and schedules execution of transactions
- scheduler - responsible for ensuring that concurrent operations on database proceed without conflicting with one another; controls relative order in which transaction operations are executed
- data manager - responsible for actual handling of data in database
  - recovery manager - ensures that database remains in consistent state in presence of failures; responsible for transaction commit and abort operations, maintaining log, and restoring system to consistent state after crash
  - buffer manager - responsible for transfer of data between main memory and secondary storage (disk); brings in pages from disk to main memory as needed in response to read user requests (sometimes referred as cache manager)

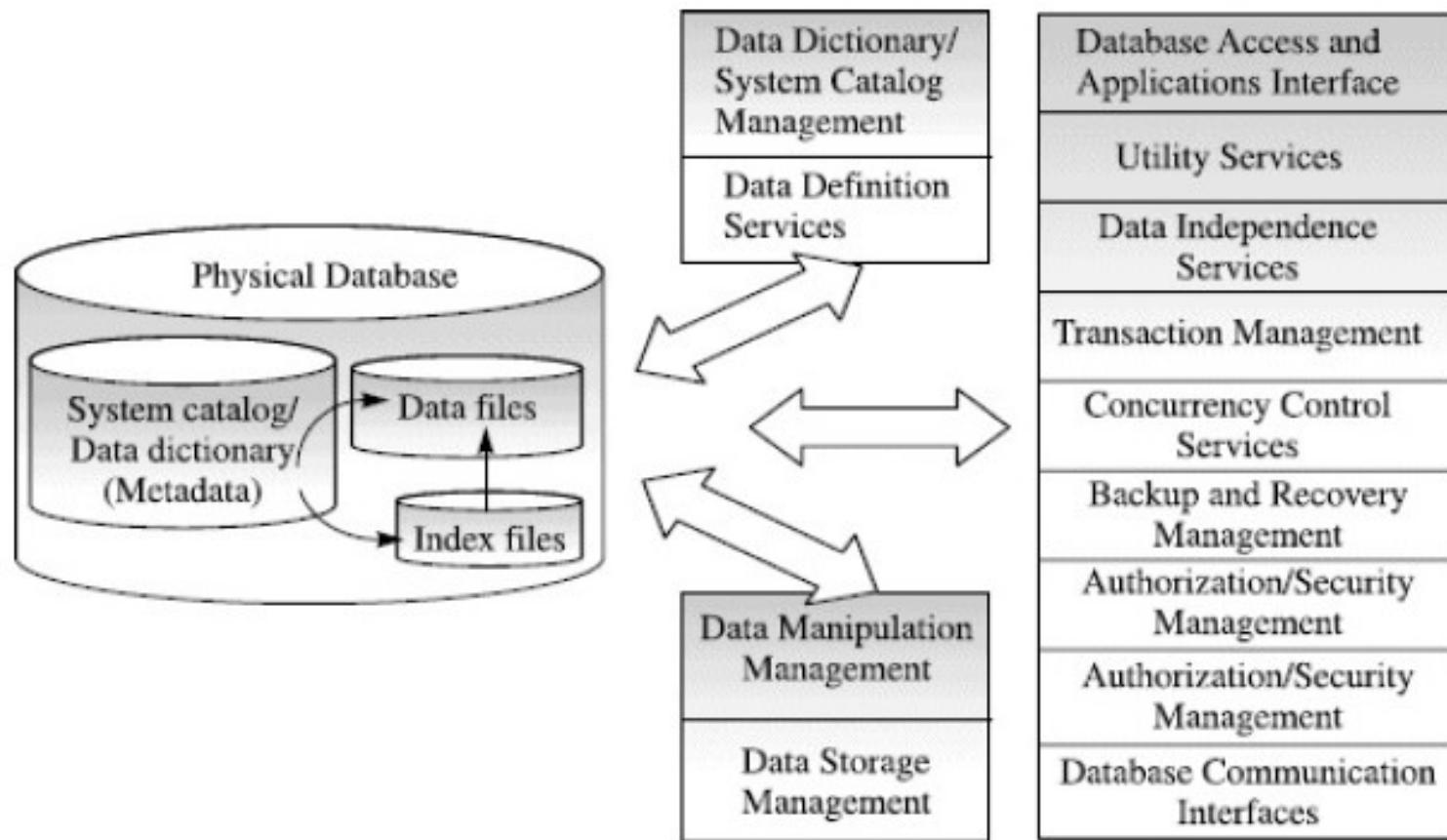
# DML processor

- using DML compiler, converts DML statements embedded in application program into standard function calls in host language
- compiler converts DML statements written in host programming language into object code for database access
- processor must interact with query processor to generate appropriate code

# DDL processor

- using DDL compiler, converts DDL statements into set of tables containing metadata; these tables contain metadata concerning database and are in form that can be used by other components of DBMS; tables are then stored in system catalog while control information is stored in data file headers
- DDL compiler processes schema definitions, specified in DDL and stores description of schema (metadata) in DBMS system catalog
- system catalog includes information such as names of data files, data items, storage details of each data file, mapping information amongst schemas, and constraints

# Functions and Services of DBMS



# Data Storage Management

- DBMS creates complex structures required for data storage in physical database, provides mechanism for management of permanent storage of data
- internal schema defines data should be stored by storage management mechanism and storage manager interfaces with operating system to access physical storage
- relieves users from task of defining and programming physical data characteristics
- DBMS provides not only for data, but also for related data entry forms or screen definitions, report definitions, data validation rules, procedural code, structure to handle pictures, sounds and video formats, and so on

# Data Definition Services

- DBMS accepts data definitions such as external schema, conceptual schema, internal schema, and all the associated mappings
- converts them to appropriate object form using DDL processor component for each of various data definition languages (DDLs)

# Data Dictionary

## System Catalog Management

- DBMS provides data dictionary or system catalog in which descriptions of data items are stored and which is accessible to users; repository of information describing data
- data about data or metadata
- all schemas and mappings, all security and integrity constraints are stored
- automatically created by DBMS and consulted frequently to resolve user requests

# Data Manipulation Management

- DBMS furnishes users with ability to retrieve, update and delete existing data
- includes DML processor component

# Authorization / Security Management

- DBMS protects database against unauthorized access - mechanism to ensure that only authorized users can access database
- enforces user security and data privacy within database
- security rules determine which users can access database, which data items each user may access and which data operations (read, add, delete and modify) user may perform

# Database Communication Interfaces

- user's requests for database are transmitted to DBMS in the form of communication messages
- DBMS provides special communication routines designed to allow database to accept user requests within computer network environment
- response to user is transmitted back from DBMS in form of such communication messages
- DBMS integrates with *data communication manager (DCM)*, which controls such message transmission activities

# Backup and Recovery Management

- DBMS provides mechanisms for backing up data periodically and recovering from different types of failures
- prevents loss of data; ensures that aborted or failed transactions do not create any adverse effect
- recovery mechanisms of DBMSs make sure that database is returned to consistent state after transaction fails or aborts due to system crash

# Concurrency Control Services

- DBMSs support sharing of data among multiple users
- must provide mechanism for managing concurrent access to database
- DBMS ensure that database is kept in consistent state and that the integrity of data is preserved
- ensures that database is updated correctly when multiple users are updating database concurrently

# Transaction Management

- transaction is series of database operations, carried out by single user or application program, which accesses or changes contents of database
- DBMS must provide mechanism to ensure either that all updates corresponding to given transaction are made or that none of them is made

# Integrity Services

- database integrity refers to correctness and consistency of stored data (important in transaction-oriented database system)
- DBMS must provide means to ensure that both data in database and changes to data follow certain rules to maximizes data consistency
- data relationships stored in data dictionary are used to enforce data integrity
- various types of integrity mechanisms and constraints may be supported to ensure that data values are valid, that operations performed on those values are valid and database remains in consistent state

# Utility Services

- DBMS provides set of utility services used by DBA and database designer to create, implement, monitor and maintain database
- help DBA to administer database effectively

# Database Access and Application Programming Interfaces

- provide interface to enable applications to use DBMS services; provide data access via structured query language (SQL)
- DBMS query language contains two components:
  - data definition language (DDL) defines structure in which data are stored
  - data manipulation language (DML) allows users to extract data
- DBMS provides data access to application programmers via procedural languages such as C, PASCAL, COBOL, Visual BASIC, .NET, php, Java and others

# Data Independence Services

- DBMS must support independence of programs from actual structure of database

# Data Models

# Model

- is an abstraction that concentrates essential aspects of organization's applications while ignores details
- is representation of real world objects and events and their associations
- *data model* is mechanism that provides this abstraction for database application

# Data Model

- represents organization - provides basic concepts and notations to allow database designers and users unambiguously and accurately communicate their understanding of organizational data
- data modelling used for representing entities of interest and their relationships in database
- conceptual method of structuring data

# Data Model

- collection of mathematically well-defined concepts that help organization to express static and dynamic properties of data intensive applications
- consists of:
  - static properties, for example, objects, attributes and relationships
  - integrity rules over objects and operations
  - dynamic properties, for example, operations or rules defining database states

# Data Model

- can be classified into:
  1. record-based data models
    - Hierarchical data model.
    - Network data model.
    - Relational data model.
  2. object-based data models
    - Entity-relationship.
    - Object-oriented.
- DBMS support a single data model

# Hierarchical Data Models

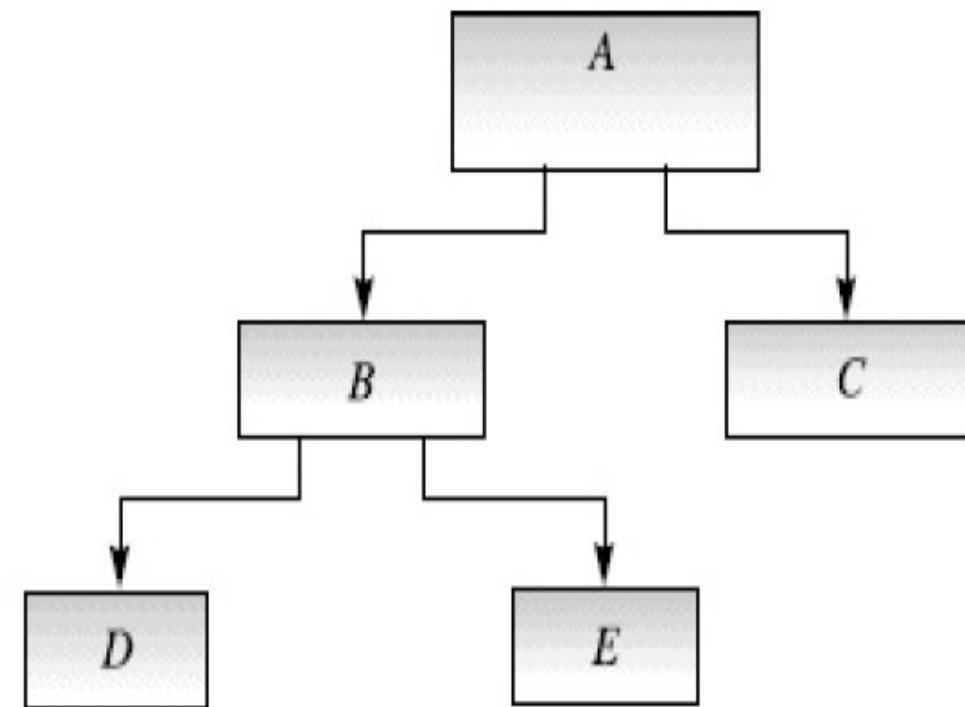
- represented by upside-down tree
- user perceives as hierarchy of segments
- segment equivalent of file system's record type
- relationship between files or records forms a hierarchy

# Hierarchical Data Models

Level-0: Root Parent  
(node)

Level-1: Root children  
(segment)

Level-2: Segments  
(Level-1 children)



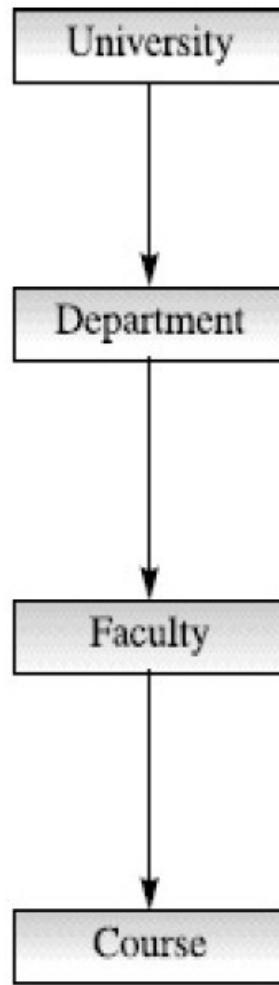
# Hierarchical Data Models

- tree defined as set of nodes such that there is one specially designated node called root (node)
- remaining nodes are portioned into disjoint sets and perceived as children of segment above them
- each disjoint set in turn is a tree

# Hierarchical Data Models

- can represent one-to-many relationship between two entities where the two are respectively parent and child
- nodes of tree represent record types

- hierarchical path that traces parent segments to child segments, beginning from left, defines the tree
- hierarchical path for segment ‘E’ can be traced as ABDE, tracing all segments from root starting at leftmost segment
- left-traced path is known as *preorder traversal* or *hierarchical sequence*

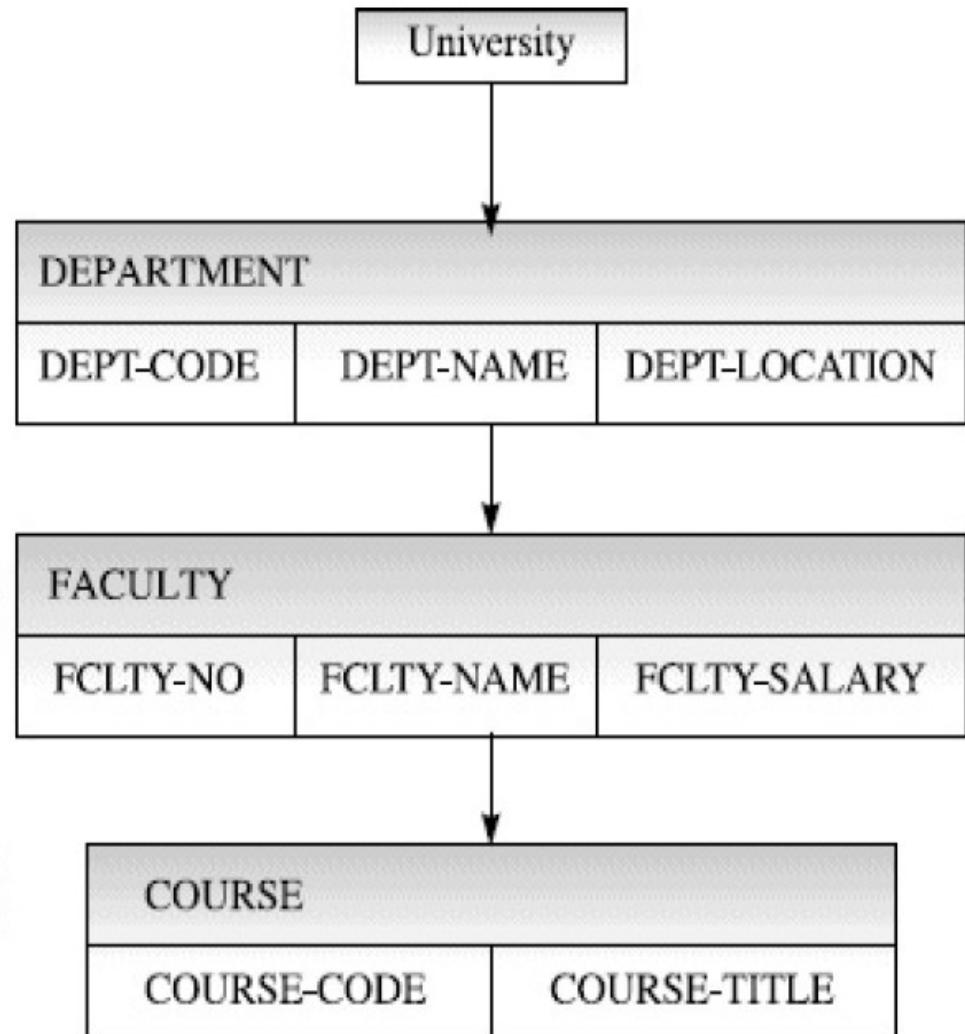


(a) University hierarchy

Level-0: Root node  
(Department is the parent of faculty)

Level-1: Root segment  
(Faculty is the child of department and parent of course)

Level-2: Root segment  
(Faculty is the child of department and parent of course)



(b) Records with hierarchical relationship

# Hierarchical Data Models

- one of oldest database models used by enterprise
- IBM *Information Management System (IMS)* became world's leading hierarchical database

# advantages

- simplicity
- data sharing
- data security
- data independence
- data integrity
- efficiency
- available expertise
- tried business applications

# disadvantages

- implementation complexity
- inflexibility
- database management problems
- lack of structural independence
- application programming complexity
- implementation limitation: many common relationships do not conform to one-to-many
- no standards
- extensive programming efforts

# Network Data Model

- similar to hierarchical model except that record can have multiple parents
- has three basic components: record type, data items (or fields) and links
- relationship is called *set* in which each set is composed of at least two record types - first record type is *owner* equivalent to parent and second record type is *member* equivalent to child - connection between owner and its member records is identified by link to which database designers assign *set-name*; this set-name used to retrieve and manipulate data

# Network Data Model

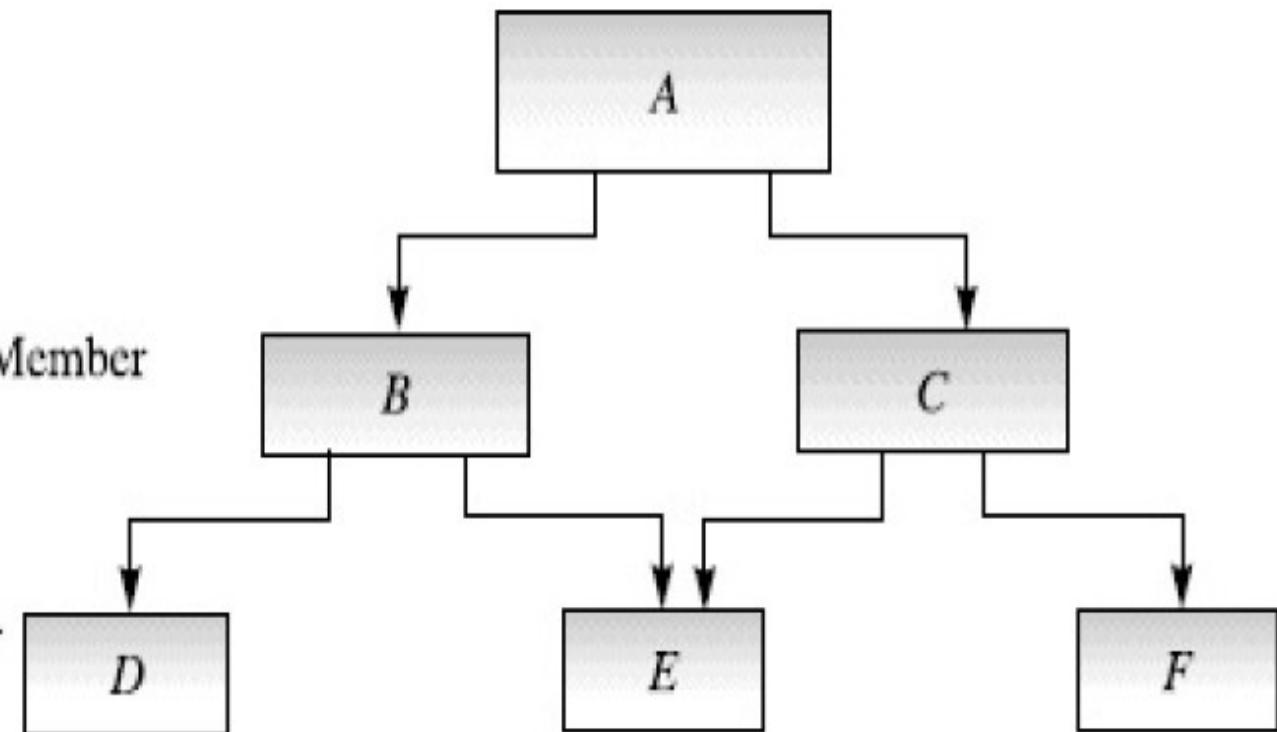
- links between owners and their members indicate access paths in network models and are typically implemented with pointers
- member can appear in more than one set and thus may have several owners, and therefore, it facilitates many-to-many relationships

# Network Data Model

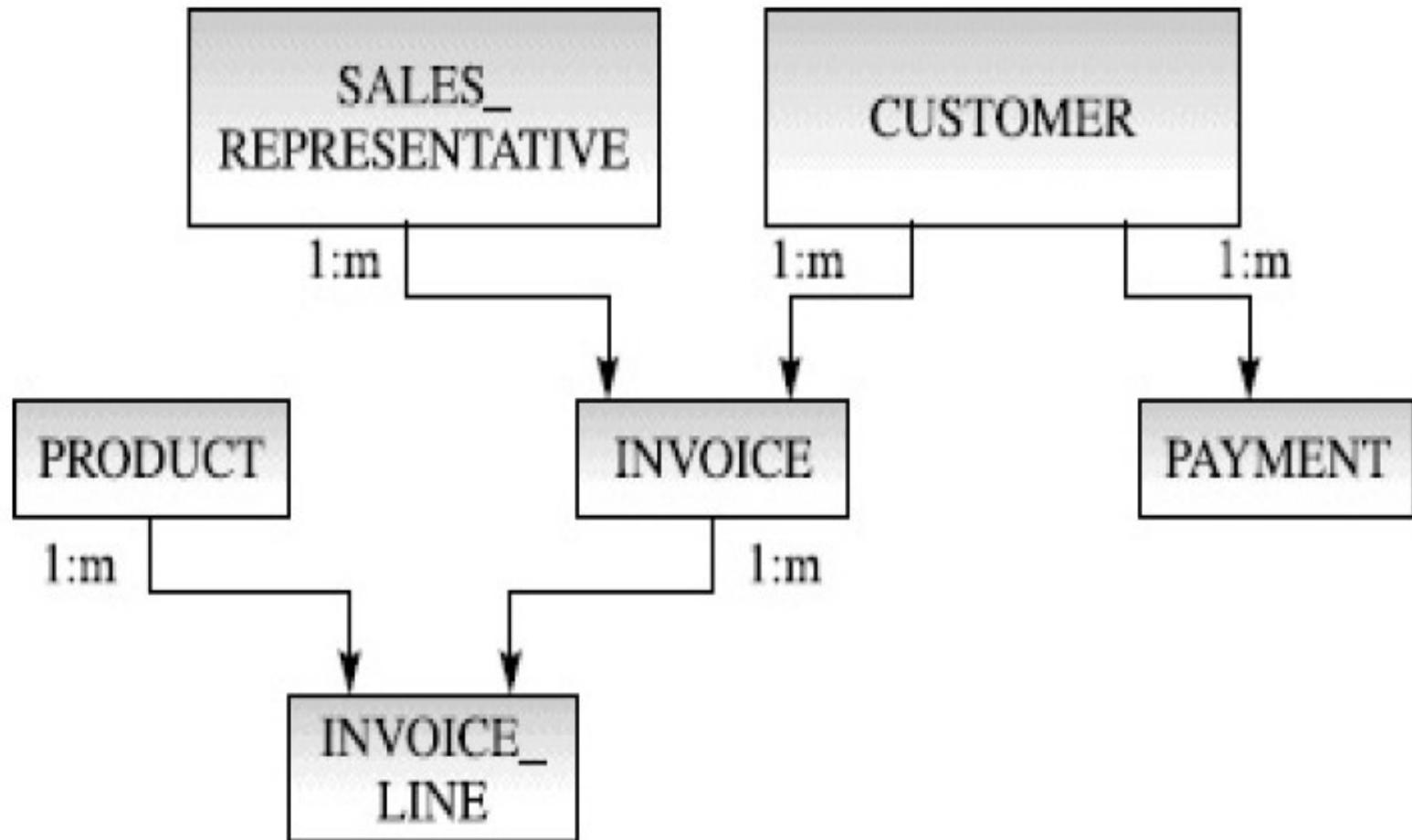
Level- 0: Owner

Level- 1: Owner/Member

Level- 2: Member



# Network Data Model for sales organization



# advantages

- simplicity
- facilitating more relationship types
- superior data access
- database integrity
- data independence
- database standards

# disadvantages

- system complexity
- absence of structural independence
- not a user-friendly

# Relational Data Model

- *Relational Database Management System (RDBMS)* performs same basic functions
- relational data model simplifies user's view of database using simple tables instead of more complex tree and network structures
- in Relational Data Model database it is collection of tables (also called relations)
- each table is matrix of series of rows and columns
- tables are related by sharing common entity characteristic

# *advantages*

- Simplicity - is even simpler than hierarchical and network models; frees designers from actual physical data storage details, thereby allowing them to concentrate on logical view of database
- structural independence - does not depend on navigational data access system; changes in database structure do not affect data access

# *advantages*

- ease of design, implementation, maintenance and uses - provides both structural independence and data independence; it makes database design, implementation, maintenance and usage much easier
- flexible and powerful query capability - provides very powerful, flexible, and easy-to-use query facilities; structured query language (SQL) capability makes ad hoc queries a reality

# disadvantages

- **hardware overheads** - need more powerful computing hardware and data storage devices to perform RDMS-assigned tasks
  - tend to be slower than other database systems
  - with rapid advancement in computing technology (1970-80) disadvantage of being slow is getting faded
- **easy-to-design capability leading to bad design** - easy-to-use feature of relational database results into untrained people generating queries and reports without much understanding and giving much thought to need of proper database design; with growth of database, poor design results into slower system, degraded performance and data corruption

# disadvantages

- recent disadvantages (in the Internet era)
  - big data
  - unstructured data

# Entity-Relationship (E-R) Data Model

- is a logical database model, which has logical representation of data for enterprise, business
- is a collection of objects of similar structures called entity set
- relationship between entity sets is represented on basis of number of entities from entity set that can be associated with number of entities of another set
  - such as one-to-one (1:1), one-to-many (1:n), or many-to-many (n:n) relationships

# E-R Data Model diagram

- E-R diagram is shown graphically
- E-R diagram has become widely accepted data model used for designing of relational databases

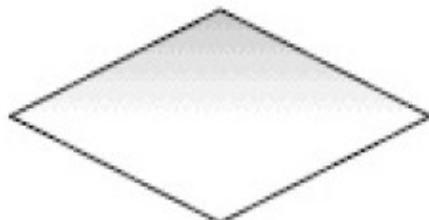
# Building blocks (symbols) of diagram of E-R Data Model



Entity set



Attributes



Relationship



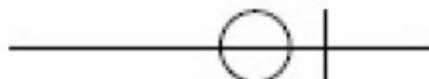
Mandatory One

---

Link between attribute  
and entity set



Mandatory Many

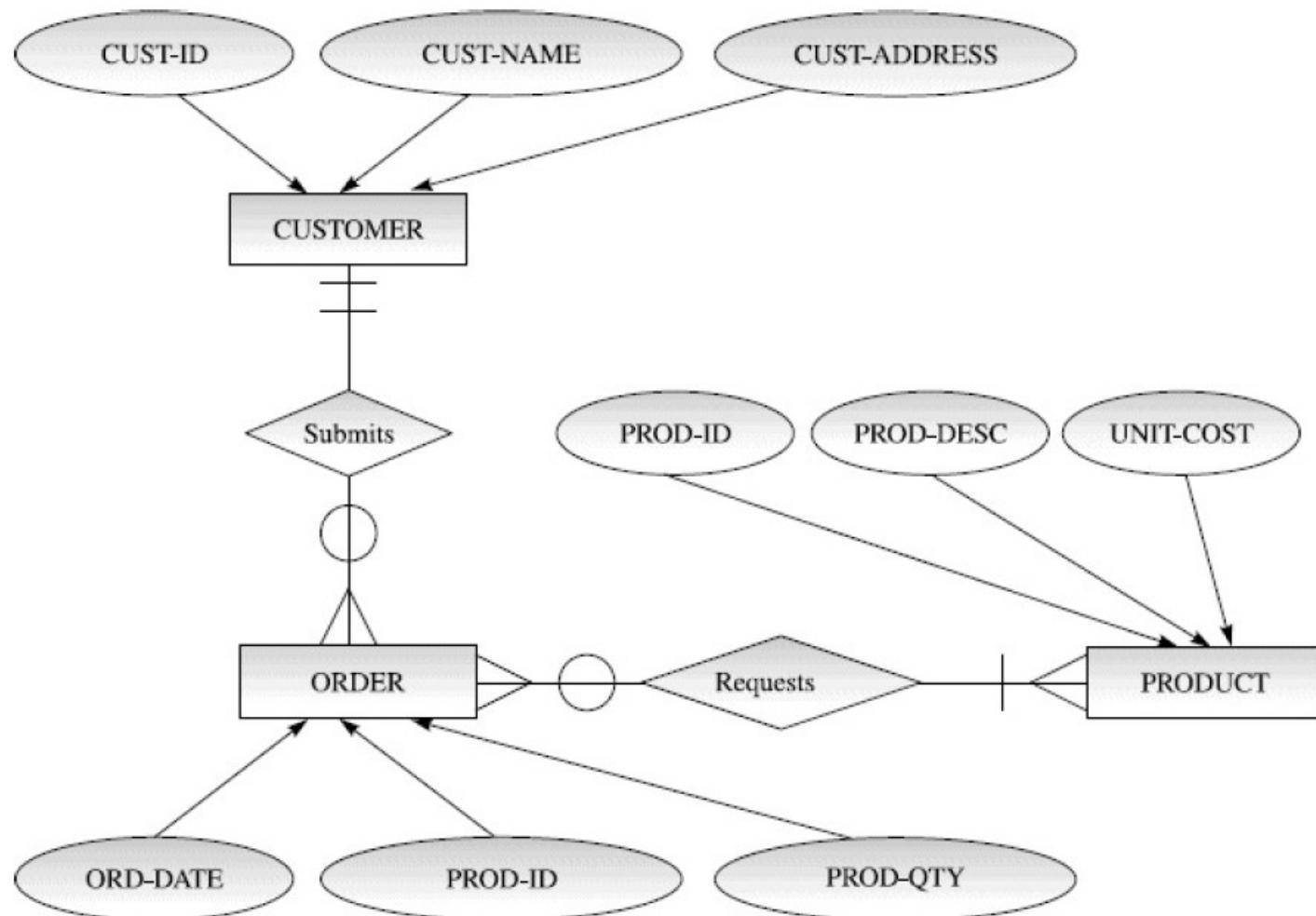


Optional One



Optional Many

# E-R Data Model diagram for product sales organization



# *advantages*

- straightforward relational representation
- easy conversion for E-R to other data model
- graphical representation for better understanding

# *disadvantages*

- no industry standard for notation
- popular only for high-level database design

# Object-Oriented Data Model

- logical data model that captures semantics of objects supported in OOP
- persistent and sharable collection of defined objects
- has the ability to model complete solution
- database models represent entity and class; class represents both object attributes as well as behavior of entity
- maintains relationships through *logical containment*

# Object-Oriented Data Model

- structure is highly variable, unlike traditional databases (such as hierarchical, network or relational), it has no single inherent database structure
- structure for any given class could be anything programmer finds useful (for example, linked list, set, array, ...); may contain varying degrees of complexity, making use of multiple types and multiple structures

# advantages

- capable of handling a large variety of data types
- combining object-oriented programming with database technology
- improved productivity
- improved data access

# disadvantages

- no precise definition
- difficult to maintain
- not suited for all applications

# Review Questions

# Review Questions

- Describe the three-tier ANSI-SPARC architecture. Why do we need mappings between different schema levels? How do different schema definition languages support this architecture?
- Discuss the advantages and characteristics of the three-tier architecture.
- Discuss the concept of data independence and explain its importance in a database environment.
- What is logical data independence and why is it important?
- What is the difference between physical data independence and logical data independence?

# Review Questions

- How does the ANSI-SPARC three-tier architecture address the issue of data independence?
- Explain the difference between external, conceptual and internal schemas. How are these different schema layers related to the concepts of physical and logical data independence?
- Describe the structure of a DBMS.
- Describe the main components of a DBMS.
- Explain the structure of DBMS.

# Review Questions

- What is a transaction?
- How does the hierarchical data model address the problem of data redundancy?
- What do you mean by a data model? Describe the different types of data models used.
- Explain the following with their advantages and disadvantages:
  - Hierarchical database model
  - Network database model
  - Relational database model
  - E-R data models
  - Object-oriented data model.

# Review Questions

- Define the following terms:
  - Data independence
  - Query processor
  - DDL processor
  - DML processor.
  - Run time database manager.
- How does the hierarchical data model address the problem of data redundancy?
- Describe the basic features of the relational data model. Discuss their advantages, disadvantages and importance to the end-user and the designer.

- 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.*

# **Relational Model**

# Objectives

- origins of relational model
- terminology of relational model
- how tables are used to represent data
- connection between mathematical relations and relations in relational model
- properties of database relations

# Objectives

- how to identify candidate, primary, alternate, and foreign keys
- meaning of entity integrity and referential integrity
- purpose and advantages of views in relational systems

# Historical perspective

- Relational Database Management System (RDBMS) has become dominant data-processing software in use today, with an estimated total software revenue worldwide of US\$ 20-30 billion
- software represents second generation of DBMSs based on proposed *relational data model* in 1970, Edgar Codd, at IBM's San Jose Research Laboratory

# Historical perspective

- in relational model, all data is logically structured within relations (tables); each ***relation*** has name and is made up of named ***attributes*** (columns) of data; each ***tuple*** (row) contains one value per attribute
- great strength of relational model is this simple logical structure; behind this structure is sound theoretical foundation that is lacking in first generation of DBMSs (**network & hierarchical**)

# Mathematics

- always strive for elegance and orthogonality - dislike exceptions
- system is said to be designed in an *orthogonal* way if its set of components that together make up whole system capability are *non-overlapping* and *mutually independent*
  - each capability should be implemented by only one component
  - one component should only implement one capability of system

# Mathematics

- well-separated and independent components ensure that there are no side effects: using or even changing one component does not cause side effects in another area of system
- formal disciplines, most relevant ones in application of mathematics to field of databases
- Set theory

# Mathematics

- *Everything should be made as simple as possible, but not simpler.*
  - Albert Einstein (1879–1955)

# Codd relational model's objectives

- to allow high degree of data independence; application programs must not be affected by modifications to internal data representation, particularly by changes to file organizations, record orderings, or access paths
- to provide substantial grounds for dealing with data semantics, consistency, and redundancy problems; introduced concept of ***normalized*** relations that have no repeating groups
- to enable expansion of set-oriented data manipulation languages

# 1 - Prototype relational DBMS System R

- at IBM's San José Research Laboratory in California - prototype relational DBMS System R, which was developed during the late 1970s
- project was designed to prove practicality of relational model by providing an implementation of its data structures and operations
- proved to be excellent source of information about implementation concerns such as transaction management, concurrency control, recovery techniques, query optimization, data security and integrity, human factors, and user interfaces,

# led to two major developments

- development of a structured query language called SQL, which has since become formal International Organization for Standardization (ISO) and de facto standard language for RDBMS
- production of various commercial relational DBMS products during late 1970s and 1980s: IBM DB2 and Oracle

## 2 - INGRES

- development of relational model INGRES (Interactive Graphics Retrieval System) project at University of California at Berkeley
- involved development of prototype RDBMS, with research concentrating on same overall objectives as System R project - led to an academic version of INGRES, which contributed to general appreciation of relational concepts, and spawned commercial products INGRES

# Terminology and structural concepts of relational model

- relational model is based on mathematical concept of ***relation***, which is physically represented as ***table***
- we explain terminology and structural concepts of the relational model

# Relation

- is a table with columns and rows
- RDBMS requires only that database be perceived by user as tables
- perception applies only to logical structure of database: external and conceptual levels of ANSI-SPARC architecture
- does not apply to physical structure of database, which can be implemented using variety of storage structures

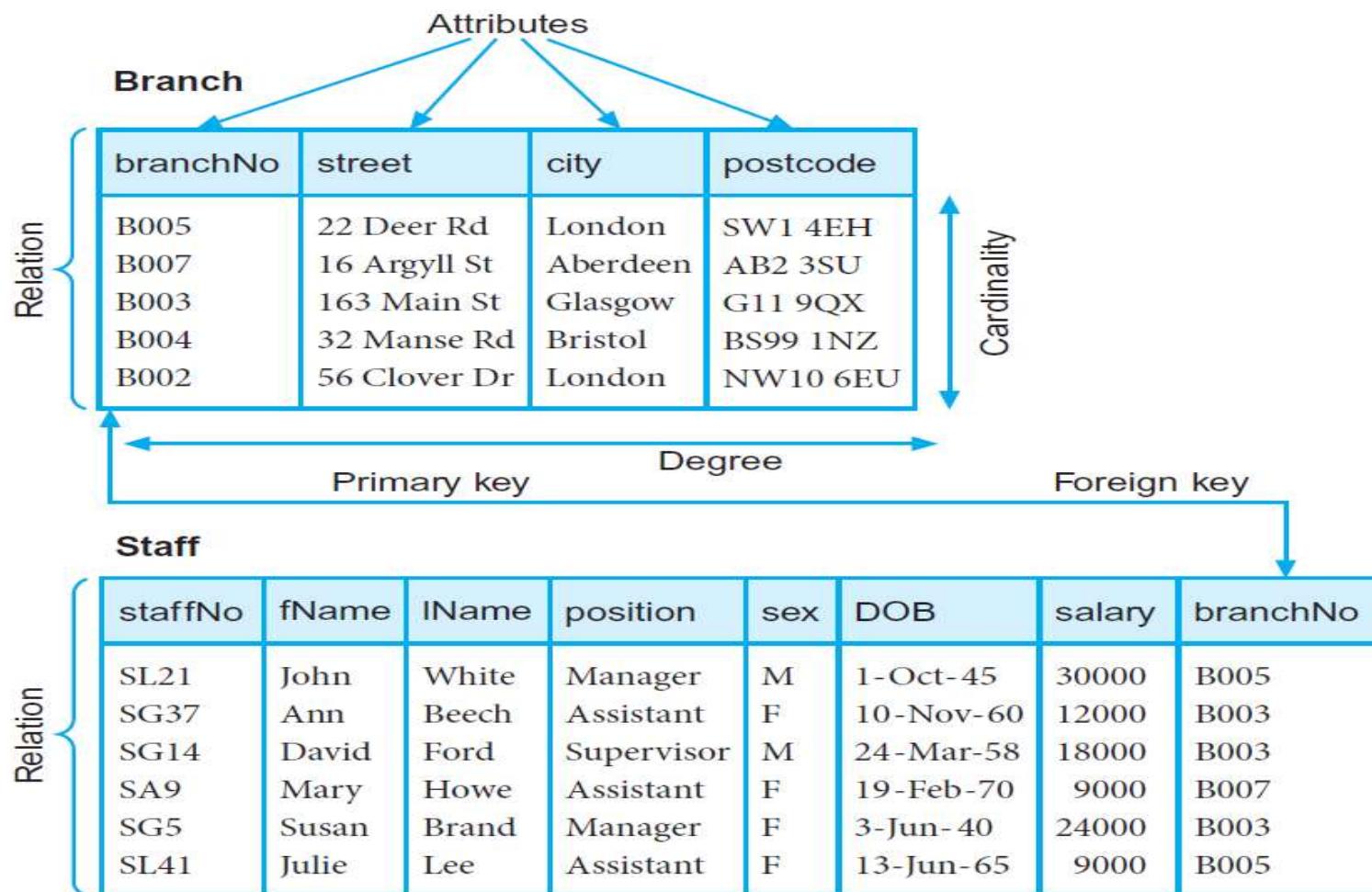
# Attribute

- is named column of relation
- relations used to hold information about objects to be represented in database as a two-dimensional table in which rows correspond to individual records and columns correspond to attributes
- attributes can appear in any order and relation will still be the same - therefore will convey same meaning

# Domain

- is set of allowable (possible) values for one or more attributes
- may be distinct for each attribute, or two or more attributes may be defined on same domain
- typically there will be values in domain that do not currently appear as values in corresponding attribute

# Instances of Branch & Staff relation



# Domains for some attributes

| Attribute | Domain Name   | Meaning                                | Domain Definition                              |
|-----------|---------------|----------------------------------------|------------------------------------------------|
| branchNo  | BranchNumbers | The set of all possible branch numbers | character: size 4, range B001-B999             |
| street    | StreetNames   | The set of all street names in Britain | character: size 25                             |
| city      | CityNames     | The set of all city names in Britain   | character: size 15                             |
| postcode  | Postcodes     | The set of all postcodes in Britain    | character: size 8                              |
| sex       | Sex           | The sex of a person                    | character: size 1, value M or F                |
| DOB       | DatesOfBirth  | Possible values of staff birth dates   | date, range from 1-Jan-20,<br>format dd-mmm-yy |
| salary    | Salaries      | Possible values of staff salaries      | monetary: 7 digits, range<br>6000.00-40000.00  |

# tuple

- is row of relation
- elements of relation are rows or tuples in table
- tuples can appear in any order and relation will still be the same - therefore convey same meaning
- structure of relation, together with specification of domains and any other restrictions on possible values, is called its intension; usually fixed - tuples are called extension (or state) of relation, which changes over time

# Degree

- is number of attributes it contains
- the Branch relation has four attributes or degree four – means that each row of table is a four-tuple, containing four values
- degree of a relation is property of *intension* of relation

# Cardinality

- is number of tuples it contains
- changes as tuples are added or deleted; is property of *extension* of relation and is determined from particular instance of relation at any given moment

# Relational database

- collection of normalized relations with distinct relation names
- consists of relations that are appropriately structured - refer to this as *normalization*

# Alternative terminology

- third set of terms is sometimes used; stems from fact that, physically, RDBMS may store each relation in a file

# Alternative terminology

| Formal term | Logical | Physical |
|-------------|---------|----------|
| Relation    | Table   | File     |
| Attribute   | Column  | Field    |
| Tuple       | Row     | Record   |

# **MATHEMATICAL RELATIONS**

# Sets and Elements

- set is a collection of objects, and those objects are called the elements of the set
- set is fully characterized by its distinct elements, and nothing else but these elements

# Sets and Elements

- elements of set don't have any ordering
- sets don't contain duplicate elements
- two sets A and B, are the same if each element of A is also an element of B and each element of B is also an element of A
- $\emptyset$  the empty set

# Sets and Elements

- set theory has a mathematical symbol  $\in$  that is used to state that some object is an element of some set
- $x \in S$
- $x \notin S \leftrightarrow \neg(x \in S)$

# Methods to specify sets

- enumerative method
  - $S1 := \{1, 2, 3, 5, 7, 11, 13, 17, 19, 23\}$
- predicative method
  - $S2 := \{x \in S \mid P(x)\}$ 
    - let  $P(x)$  be a given predicate with exactly one parameter named  $x$  of type  $S$
- substitutive method
  - $S3 := \{x^2 \mid x \in N\}$

# Cardinality

- sets can be finite and infinite
- when dealing with databases all your sets are finite by definition
- for every finite set  $S$ , cardinality is defined as number of elements of  $S$

# Subsets, Supersets

- A is a subset of B (B is a superset of A) if every element of A is an element of B
- $A \subseteq B$
- A is proper subset of B (B is proper superset of A) if A is subset of B and  $A \neq B$
- $A \subset B$

# Union, Intersection, Difference

- union  $A \cup B = \{ x \mid x \in A \vee x \in B\}$
- intersection  $A \cap B = \{ x \mid x \in A \wedge x \in B\}$
- difference  $A - B = \{ x \mid x \in A \wedge x \notin B\}$

# (Binary) Relation

- two sets,  $D_1$  and  $D_2$ , where  $D_1 = \{2, 4\}$  and  $D_2 = \{1, 3, 5\}$ ; Cartesian product of these two sets, written  $D_1 \times D_2$ , is set of all ordered pairs such that first element is a member of  $D_1$  and second element is a member of  $D_2$  -  $D_1 \times D_2 = \{(2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5)\}$
- any subset of this Cartesian product is a relation

# Example of relation $R$

- $R = \{(2, 1), (4, 1)\}$
- $R = \{(x, y) \mid x \in D_1, y \in D_2, \text{ and } y = 1\}$
- easily extend notion to three sets; let  $D_1$ ,  $D_2$ , and  $D_3$  be sets; Cartesian product  $D_1 \times D_2 \times D_3$  of these three sets is set of all ordered triples such that first element is from  $D_1$ , second is from  $D_2$  and third element is from  $D_3$
- any subset of Cartesian product is relation

# N-ary Relation

- order of coordinates of ordered pair is important
- order of tuples it is not important – it is subset of Cartesian – subset of a set is a set – order of element in a set it is not important
- can extend three sets to a more general concept of n-ary relation

# Relation

- define general relation on  $n$  domains
- let  $D_1, D_2, \dots, D_n$  be  $n$  sets
- Cartesian product is defined as:
- $D_1 \times D_2 \times \dots \times D_n = \{(d_1, d_2, \dots, d_n) \mid d_1 \in D_1, d_2 \in D_2, \dots, d_n \in D_n\}$  and is usually written as  $\prod_{i=1}^n D_i$
- any set of  $n$ -tuples from this Cartesian product is relation on  $n$  sets

# DataBase Relations

- note that in defining these relations we have to specify sets, or domains, from which we choose values
- Relation schema: named relation defined by set of attribute and domain name pairs
- let  $A_1, A_2, \dots, A_n$  be attributes with domains  $D_1, D_2, \dots, D_n$
- then set  $\{A_1:D_1, A_2:D_2, \dots, A_n:D_n\}$  is relation schema

# DataBase Relations

- relation  $R$  defined by relation schema is set of mappings from attribute names to their corresponding domains
- relation  $R$  is set of  $n$ -tuples:
- $(A_1:d_1, A_2:d_2, \dots, A_n:d_n)$  such that  $d_1 \in D_1$ ,  $d_2 \in D_2, \dots, d_n \in D_n$
- each element in  $n$ -tuple consists of attribute and value for that attribute

# DataBase Relations

- normally we list attribute names as column headings and write out tuples as rows having form  $(d_1, d_2, \dots, d_n)$ , where each value is taken from appropriate domain
- Relation is any subset of Cartesian product of domains of attributes
- table is simply physical representation of such relation

# Branch and Staff relations

## Branch

| branchNo | street       | city     | postCode |
|----------|--------------|----------|----------|
| B005     | 22 Deer Rd   | London   | SW1 4EH  |
| B007     | 16 Argyll St | Aberdeen | AB2 3SU  |
| B003     | 163 Main St  | Glasgow  | G11 9QX  |
| B004     | 32 Manse Rd  | Bristol  | BS99 1NZ |
| B002     | 56 Clover Dr | London   | NW10 6EU |

## Staff

| staffNo | fName | IName | position   | sex | DOB       | salary | branchNo |
|---------|-------|-------|------------|-----|-----------|--------|----------|
| SL21    | John  | White | Manager    | M   | 1-Oct-45  | 30000  | B005     |
| SG37    | Ann   | Beech | Assistant  | F   | 10-Nov-60 | 12000  | B003     |
| SG14    | David | Ford  | Supervisor | M   | 24-Mar-58 | 18000  | B003     |
| SA9     | Mary  | Howe  | Assistant  | F   | 19-Feb-70 | 9000   | B007     |
| SG5     | Susan | Brand | Manager    | F   | 3-Jun-40  | 24000  | B003     |
| SL41    | Julie | Lee   | Assistant  | F   | 13-Jun-65 | 9000   | B005     |

- it shows for example that employee John White is manager with salary of 30,000, who works at branch (branchNo) B005, which, from first table, is at 22 Deer Rd in London
- there is relationship between Staff and Branch: branch office *has* staff
- there is no explicit link between these two tables; it is only by knowing that attribute branchNo in Staff have same value as branchNo of Branch

# Branch relation

- has attributes branchNo, street, city, and postcode, each with its domain
- is any subset of Cartesian product of domains, or any set of four-tuples in which first element is from domain BranchNumbers, second is from domain StreetNames, ...

# Branch relation instance

- one of tuples is: or more correctly
- $\{(B005, 22\text{ Deer Rd, London, SW1 4EH})\}$
- $\{\text{branchNo: B005, street: 22 Deer Rd, city: London, postcode: SW1 4EH}\}$
- we refer to this as *relation instance*
- Branch table is convenient way of writing out all tuples that form relation at specific moment in time

# Relational database schema

- set of relation schemas, each with distinct name
- in same way that relation has schema, so too does relational database
- if  $R_1, R_2, \dots, R_n$  are set of relation schemas, then we can write *relational database schema*, or simply *relational schema*,  $R$ , as:  $R = \{R_1, R_2, \dots, R_n\}$

# Properties of Relations

- relation has name that is distinct from all other relation names in relational schema
- each cell of relation contains exactly one atomic (single) value; relations do not contain repeating groups
- relation that satisfies this property is said to be *normalized* in *first normal form*

# Properties of Relations

- each attribute has distinct name
- value of attribute are all from same domain
- each tuple is distinct; there are no duplicate tuples
- order of attributes has no significance
- order of tuples has no significance
- most of properties specified result from properties of mathematical relations

# Relational Keys

- ... there are no duplicate tuples ...
- therefore, we need to identify one or more attributes (*relational keys*) that uniquely identifies each tuple in relation

# Superkey

- attribute, or set of attributes, that uniquely identifies tuple within relation
- may contain additional attributes that are not necessary for unique identification
- interested in identifying superkeys that contain only minimum number of attributes necessary for unique identification

# Candidate Key

- superkey such that no proper subset is superkey within relation
- *Uniqueness* – in each tuple of  $R$ , values of  $K$  uniquely identify that tuple
- *Irreducibility* - no proper subset of  $K$  has uniqueness property
- when key consists of more than one attribute, we call it *composite key*
- there may be several candidate keys for relation

# Candidate Key

- consider Branch relation
- given value of city, we can determine several branch offices (London has two branch offices)
- attribute cannot be a candidate key
- company allocates each branch office unique branch number we can determine at most one tuple, so that *branchNo* is candidate key
- similarly, postcode is also candidate key

# Candidate Key

- consider relation *Viewing*, which contains information relating to properties viewed by clients
- relation comprises client number (*clientNo*), property number (*propertyNo*), date of viewing (*viewDate*) and, optionally, comment (*comment*)

# Candidate Key

- combination of *clientNo* and *propertyNo* identifies at most one tuple, so for *Viewing* together form (composite) candidate key
- if we take into account possibility that client may view property more than once, then we could add *viewDate* to composite key

# Candidate Key

- instance of relation cannot be used to prove that attribute or combination of attributes is CK
- fact that there are no duplicates for values that appear at particular moment in time does not guarantee that duplicates are not possible
- presence of duplicates in instance can be used to show that attribute combination is not CK
- identifying candidate key requires that we know the “real-world” meaning of attribute(s) involved; can decide whether duplicates are possible

# Primary Key

- candidate key that is selected to identify tuples uniquely within relation (by database designer)
- relation has no duplicate tuples; always possible to identify each row uniquely
- relation always has primary key
- in worst case, entire set of attributes could serve as primary key; usually some smaller subset is sufficient to distinguish tuples

# Alternate Key

- candidate keys that are not selected to be primary key
- for *Branch* relation, if we choose *branchNo* as primary key, *postcode* would then be an alternate key
- for *Viewing* relation, there is only one candidate key, comprising *clientNo* and *propertyNo*, so these attributes would automatically form primary key

# Foreign Key

- attribute, or set of attributes, within one relation that matches candidate key of some (possibly same) relation
- when attribute appears in more than one relation, its appearance usually represents relationship between tuples of two relations
- for example, inclusion of *branchNo* in both *Branch* and *Staff* relations is quite deliberate and links each branch to details of staff working at that branch

# Foreign Key

- in *Branch* relation, *branchNo* is primary key
- in *Staff* relation, *branchNo* attribute exists to match staff to branch office they work in
- in *Staff* relation, *branchNo* is *foreign key*
- we say that attribute *branchNo* in *Staff* relation *targets* primary key attribute *branchNo* in home relation, *Branch*

# Representing Relational Database Schemas

- *Branch (branchNo, street, city, postcode)*
- *Staff (staffNo, fName, lName, position, sex, DOB, salary, branchNo)*
- *PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)*
- *Client (clientNo, fName, lName, telNo, prefType, maxRent, eMail)*

# Representing Relational Database Schemas

- *PrivateOwner (ownerNo, fName, lName, address, telNo, eMail, password)*
- *Viewing (clientNo, propertyNo, viewDate, comment)*
- *Registration (clientNo, branchNo, staffNo, dateJoined)*

# Representing Relational Database Schemas

- common convention for representing relation schema is to give name of relation followed by attribute names in parentheses; primary key is underlined
- *conceptual model*, or *conceptual schema*, is set of all such schemas for database

# Instance of rental database

**Branch**

| branchNo | street       | city     | postcode |
|----------|--------------|----------|----------|
| B005     | 22 Deer Rd   | London   | SW1 4EH  |
| B007     | 16 Argyll St | Aberdeen | AB2 3SU  |
| B003     | 163 Main St  | Glasgow  | G11 9QX  |
| B004     | 32 Manse Rd  | Bristol  | BS99 1NZ |
| B002     | 56 Clover Dr | London   | NW10 6EU |

**Staff**

| staffNo | fName | IName | position   | sex | DOB       | salary | branchNo |
|---------|-------|-------|------------|-----|-----------|--------|----------|
| SL21    | John  | White | Manager    | M   | 1-Oct-45  | 30000  | B005     |
| SG37    | Ann   | Beech | Assistant  | F   | 10-Nov-60 | 12000  | B003     |
| SG14    | David | Ford  | Supervisor | M   | 24-Mar-58 | 18000  | B003     |
| SA9     | Mary  | Howe  | Assistant  | F   | 19-Feb-70 | 9000   | B007     |
| SG5     | Susan | Brand | Manager    | F   | 3-Jun-40  | 24000  | B003     |
| SL41    | Julie | Lee   | Assistant  | F   | 13-Jun-65 | 9000   | B005     |

**PropertyForRent**

| propertyNo | street        | city     | postcode | type  | rooms | rent | ownerNo | staffNo | branchNo |
|------------|---------------|----------|----------|-------|-------|------|---------|---------|----------|
| PA14       | 16 Holhead    | Aberdeen | AB7 5SU  | House | 6     | 650  | CO46    | SA9     | B007     |
| PL94       | 6 Argyll St   | London   | NW2      | Flat  | 4     | 400  | CO87    | SL41    | B005     |
| PG4        | 6 Lawrence St | Glasgow  | G11 9QX  | Flat  | 3     | 350  | CO40    |         | B003     |
| PG36       | 2 Manor Rd    | Glasgow  | G32 4QX  | Flat  | 3     | 375  | CO93    | SG37    | B003     |
| PG21       | 18 Dale Rd    | Glasgow  | G12      | House | 5     | 600  | CO87    | SG37    | B003     |
| PG16       | 5 Novar Dr    | Glasgow  | G12 9AX  | Flat  | 4     | 450  | CO93    | SG14    | B003     |

# Instance of rental database

**Client**

| clientNo | fName | IName   | telNo         | prefType | maxRent | eMail                  |
|----------|-------|---------|---------------|----------|---------|------------------------|
| CR76     | John  | Kay     | 0207-774-5632 | Flat     | 425     | john.kay@gmail.com     |
| CR56     | Aline | Stewart | 0141-848-1825 | Flat     | 350     | astewart@hotmail.com   |
| CR74     | Mike  | Ritchie | 01475-392178  | House    | 750     | mritchie01@yahoo.co.uk |
| CR62     | Mary  | Tregear | 01224-196720  | Flat     | 600     | maryt@hotmail.co.uk    |

**PrivateOwner**

| ownerNo | fName | IName  | address                       | telNo         | eMail             | password |
|---------|-------|--------|-------------------------------|---------------|-------------------|----------|
| CO46    | Joe   | Keogh  | 2 Fergus Dr, Aberdeen AB2 7SX | 01224-861212  | jkeogh@lhh.com    | *****    |
| CO87    | Carol | Farrel | 6 Achray St, Glasgow G32 9DX  | 0141-357-7419 | cfarrel@gmail.com | *****    |
| CO40    | Tina  | Murphy | 63 Well St, Glasgow G42       | 0141-943-1728 | tinam@hotmail.com | *****    |
| CO93    | Tony  | Shaw   | 12 Park Pl, Glasgow G4 0QR    | 0141-225-7025 | tony.shaw@ark.com | *****    |

**Viewing**

| clientNo | propertyNo | viewDate  | comment        |
|----------|------------|-----------|----------------|
| CR56     | PA14       | 24-May-13 | too small      |
| CR76     | PG4        | 20-Apr-13 | too remote     |
| CR56     | PG4        | 26-May-13 |                |
| CR62     | PA14       | 14-May-13 | no dining room |
| CR56     | PG36       | 28-Apr-13 |                |

**Registration**

| clientNo | branchNo | staffNo | dateJoined |
|----------|----------|---------|------------|
| CR76     | B005     | SL41    | 2-Jan-13   |
| CR56     | B003     | SG37    | 11-Apr-12  |
| CR74     | B003     | SG37    | 16-Nov-11  |
| CR62     | B007     | SA9     | 7-Mar-12   |

# Null

- represents value for attribute that is currently unknown or is not applicable for this tuple
- mean logical value “unknown”, value is not applicable to particular tuple, or that no value has yet been supplied
- way to deal with incomplete or exceptional data
- not same as zero or empty text string
- represents absence of value

# Null

- for example, in Viewing relation *comment* attribute may be undefined until potential renter has visited property and returned comment to agency
- without nulls, it becomes necessary to introduce false data to represent this state

# Null

- can cause implementation problems, arising from fact that relational model is based on first-order predicate calculus, which is two-valued or Boolean logic; allowing nulls means that we have to work with higher-valued logic, such as three-valued logic (true, false, null)
- Codd regarded nulls as an integral part of relational model

# Integrity Constraints

- relational integrity constraints ensure that data is accurate
- because every attribute has an associated domain, there are ***domain constraints*** that form restrictions on set of values allowed for attributes of relations
- ***integrity rules*** - constraints that apply to all instances of database: ***entity integrity*** and ***referential integrity***

# Integrity Constraints

- because every attribute has an associated domain, there are ***domain constraints*** that form restrictions on set of values allowed for attributes of relations
- ***integrity rules*** - constraints that apply to all instances of database: ***entity integrity*** and ***referential integrity***

# Entity integrity

- in base relation, no attribute of primary key can be null
- PK is minimal identifier used to identify tuples uniquely; if we allow null for any part of PK we are implying that not all attributes are needed
- for example, *branchNo* is PK of *Branch*, we should not be able to insert tuple with null
- Viewing (clientNo, propertyNo, ...) we should not be able to insert tuple into Viewing with null for clientNo or null for propertyNo or nulls for both

# Entity integrity

- using data of *Viewing* relation consider query  
“List all comments from viewings”
- produce unary relation consisting of attribute comment
- this attribute must be PK, but it contains nulls
- this relation is not base relation (derived relation), model allows primary key to contain nulls

# Referential integrity

- if foreign key exists in relation, either foreign key value must match a candidate key value of some tuple in its home relation or foreign key value must be wholly null

# Referential integrity

- for example, *branchNo* in *Staff* is FK targeting *branchNo* attribute in home relation, *Branch*
- possible to create *Staff* record with *branchNo* B025, if there is already *branchNo* B025 in *Branch* relation
- we should be able to create *Staff* record with null *branchNo* to allow for situation where member of staff has joined company but has not yet been assigned to particular branch office

# General constraints

- additional rules specified by users of database that define or constrain some aspect of organization
- for example, if an upper limit of 20 has been placed upon number of staff that may work at branch office
- level of support for general constraints varies from system to system

# General constraints

- *declarative integrity constraint* is statement about database that is always true
- *trigger* is procedural code that is automatically executed in response to certain events on particular table or view in database; mostly used for maintaining integrity of information

# Views

- in relational model has slightly different meaning than view from architecture
- rather than being entire external model of user's view, view is *virtual* or *derived relation* that does not necessarily exist in its own right, but may be dynamically derived from one or more base relations
- external model can consist of both base (conceptual-level) relations and views derived from base relations

# Views

- relation that appears to user to exist, can be manipulated as if it were base relation, but does not necessarily exist in storage (although its definition is stored in system catalog)
- contents of view are defined as query on one or more base relations
- operations on view are automatically translated into operations on relations from which it is derived

# Views

- dynamic – changes made to base relations that affect view are immediately reflected in view
- when users make permitted changes to (updating) view, these changes are made to underlying relations

# Views

- provides powerful and flexible security mechanism by hiding parts of database from certain users; users are not aware of existence of any attributes or tuples that are missing
- it permits users to access data in way that is customized to their needs, same data can be seen by different users in different ways
- it can simplify complex operations on base relations

# Views

- for example, if view defined as combination (join) of two relations users may now perform more simple operations on view, which will be translated by DBMS into equivalent operations
- some members of staff should see *Staff* tuples without salary attribute
- attributes may be renamed or order changed; user accustomed to calling *branchNo* attribute of branches by full name

# Views

- provides *logical data independence*; supports reorganization of conceptual schema
- for example, if new attribute is added to relation, existing users can be unaware of its existence if their views are defined to exclude it; if existing relation is rearranged or split up, view may be defined so that users can continue to see their original views

# Codd's rules

- Codd came up with rules database must obey in order to be regarded as relational
- hardly any commercial product follows all

# Rule 0: Foundation

- system must qualify as relational, as a database, and as a management system
- system must use its relational facilities (exclusively) to manage database
- foundation rule, which acts as base for all other 12 rules derive from this

# Rule 1: Information

- all information in database is to be represented in one and only one way, namely by values in column positions within rows of tables
- data stored in database, may it be user data or metadata, must be value of some table cell; everything in database must be stored in a table format

# Rule 2: Guaranteed Access

- all data must be accessible; essentially restatement of fundamental requirement for primary keys; every individual scalar value in database must be logically addressable by specifying name of table, name of column and primary key value
- every single data element (value) is guaranteed to be accessible logically with combination of table-name, primary-key (row), and attribute-name (column); no other means, such as pointers, can be used to access data

# Rule 3: Systematic Treatment of NULL Values

- DBMS must allow each field to remain null (or empty); must support representation of "missing information and inapplicable information" that is systematic, distinct from all regular values (0, "", ...), and independent of data type; manipulated by DBMS in systematic way
- NULL values in database must be given systematic and uniform treatment; interpreted as
  - data is missing, data is not known, or data is not applicable

# Rule 4: Active Online Catalog

- system must support online relational catalog that is accessible to authorized users by means of their regular query language; users must be able to access database's structure (catalog) using same query language that they use to access database's data
- structure description of entire database must be stored in online catalog, known as *data dictionary*, which can be accessed by authorized users; users can use same query language to access catalog which they use to access database itself

# Rule 5: Comprehensive Data Sub-Language

- system must support at least one relational language that
  - has linear syntax
  - can be used both interactively and within application programs
  - supports data definition operations (including view definitions), data manipulation operations (update and retrieval), security and integrity constraints, transaction management operations (begin, commit, and rollback)

# Rule 5: Comprehensive Data Sub-Language

- database can only be accessed using language having linear syntax that supports data definition, data manipulation, and transaction management operations; language can be used directly or by means of some application
- if database allows access to data without any help of this language, then it is considered violation

# Rule 6: View Updating

- all views those can be updated theoretically, must be updated by system.
- all views of database, which can theoretically be updated, must also be updatable by system

# Rule 7: High-Level Insert, Update, and Delete

- system must support set-at-a-time insert, update, and delete operators; data can be retrieved from relational database in sets constructed of data from multiple rows and/or multiple tables; insert, update, and delete operations should be supported for any retrievable set rather than just for single row in single table

# Rule 7: High-Level Insert, Update, and Delete

- database must support high-level insertion, updation, and deletion; this must not be limited to single row, that is, it must also support union, intersection and minus operations to yield sets of data records

# Rule 8: Physical Data Independence

- changes to physical level (how data is stored, whether in arrays or linked lists etc.) must not require change to application based on structure
- data stored in database must be independent of applications that access database; any change in physical structure of database must not have any impact on how data is being accessed by external applications

# Rule 9: Logical Data Independence

- changes to logical level (tables, columns, rows, ...) must not require change to application based on structure; more difficult to achieve
- logical data in database must be independent of its user's view (application); any change in logical data must not affect applications using it

# Rule 10: Integrity Independence

- integrity constraints must be specified separately from application programs and stored in catalog; it must be possible to change such constraints as and when appropriate without unnecessarily affecting existing applications
- database must be independent of application that uses it; all its integrity constraints can be independently modified without need of any change in application; makes database independent of front-end application & interface

# Rule 11: Distribution Independence

- distribution of portions of database to various locations should be invisible to users; existing applications should continue to operate
  - when distributed version is first introduced
  - when existing distributed data are redistributed
- end-user must not be able to see that data is distributed over various locations; users should always get impression that data is located at one site only;
- regarded as foundation of distributed database

# Rule 12: Non-Subversion Rule

- if system provides low-level (record-at-a-time) interface, then that interface cannot be used to subvert system; bypassing relational security or integrity constraint
- if system has an interface that provides access to low-level records, then interface must not be able to subvert system and bypass security and integrity constraints

# Review Questions

- discuss each of following concepts in context of relational data model: relation; attribute; domain; tuple; intension and extension; degree and cardinality
- describe term “normalized reaction”
- why are constraints so important in relational database
- discuss properties of relation

# Review Questions

- differences between CK & PK of relation
- explain what is meant by FK; how do FK relate to CK? give examples
- define two principal integrity rules for relational model; why it is desirable to enforce these rules
- define views; why are they important in database approach

# **Entity–Relationship Modeling**

# Objectives

- ***use Entity–Relationship (ER) modeling in database design***
- basic concepts associated with ER model: entities, relationships, and attributes
- identify and resolve problems with ER models (connection traps)

# Objectives

- **Specialization/Generalization**
- special types of entities known as superclasses and subclasses, and attribute inheritance
- present two main types of constraints on superclass/subclass relationships called participation and disjoint constraints
- show how to represent specialization / generalization in ER diagram

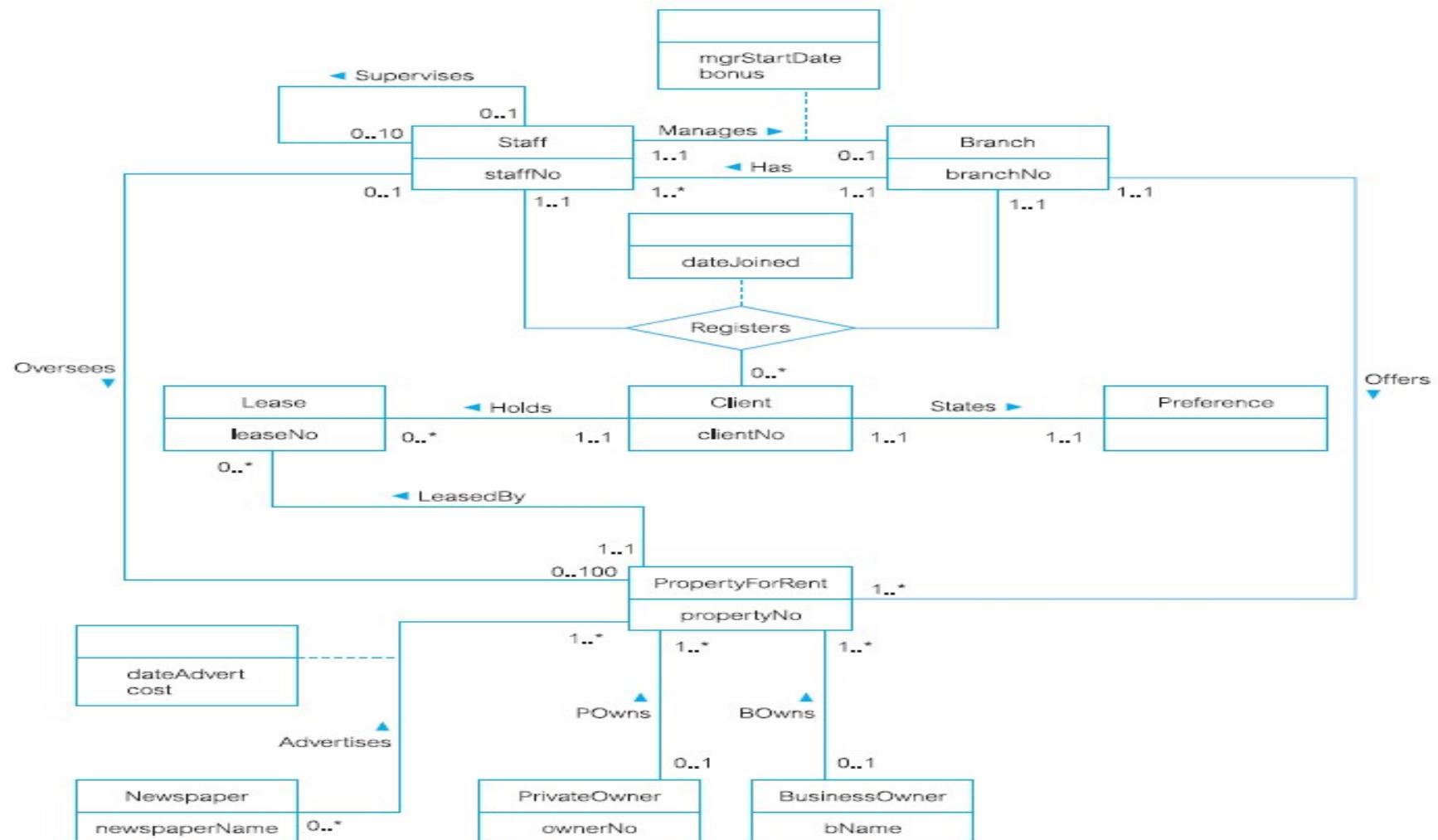
- designers, programmers, and end-users tend to view data & its use in different way
- common understanding that reflects how enterprise operates
- need model for communication - nontechnical and free of ambiguities
- Entity–Relationship (ER) model is one such example

- Entity–Relationship (ER) modeling is top-down approach to database design that begins by identifying important data called *entities* and *relationships*; add more details, such as information we want to hold about entities and relationships (*attributes*) and any *constraints*

# Entity Set

- group of objects with same properties, which are identified by enterprise as having an independent existence

# Entity–Relationship (ER) diagram of Branch view



# Entity

- independent existence can be objects with physical (or “real”) existence
- objects with conceptual (or “abstract”) existence

## **Physical existence**

- Staff
- Property
- Customer
- Part
- Supplier
- Product

## **Conceptual existence**

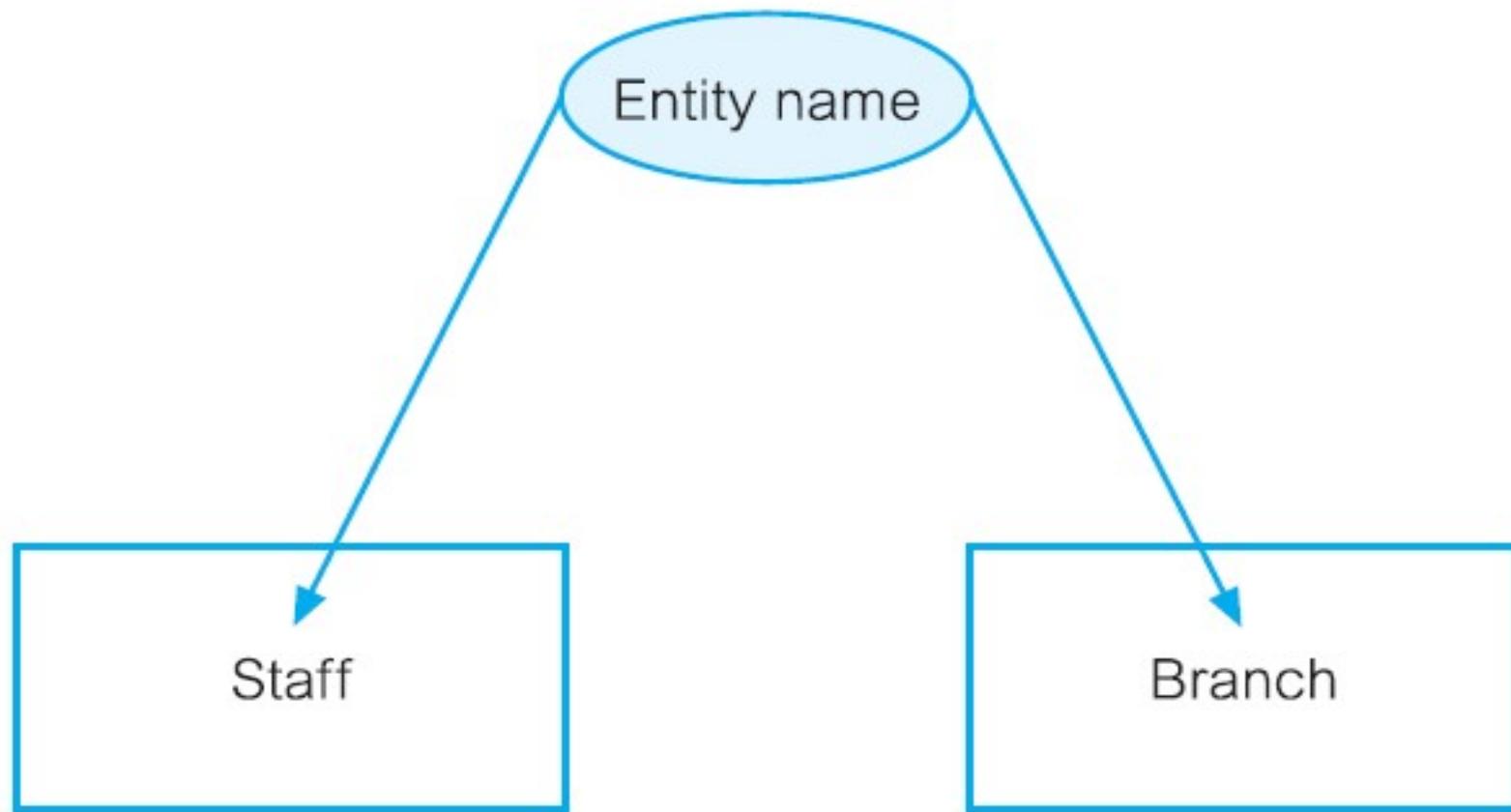
- Viewing
- Inspection
- Sale
- Work experience

# Entity occurrence

- uniquely identifiable object of entity set

- use terms “entity set” and “entity” or “entity occurrence”; use more general term “entity” where meaning is obvious
- identify each entity type by name and list of properties
- examples of entity sets: Staff, Branch, PropertyForRent, and PrivateOwner

# Diagram of Staff and Branch entity sets



# Relationship

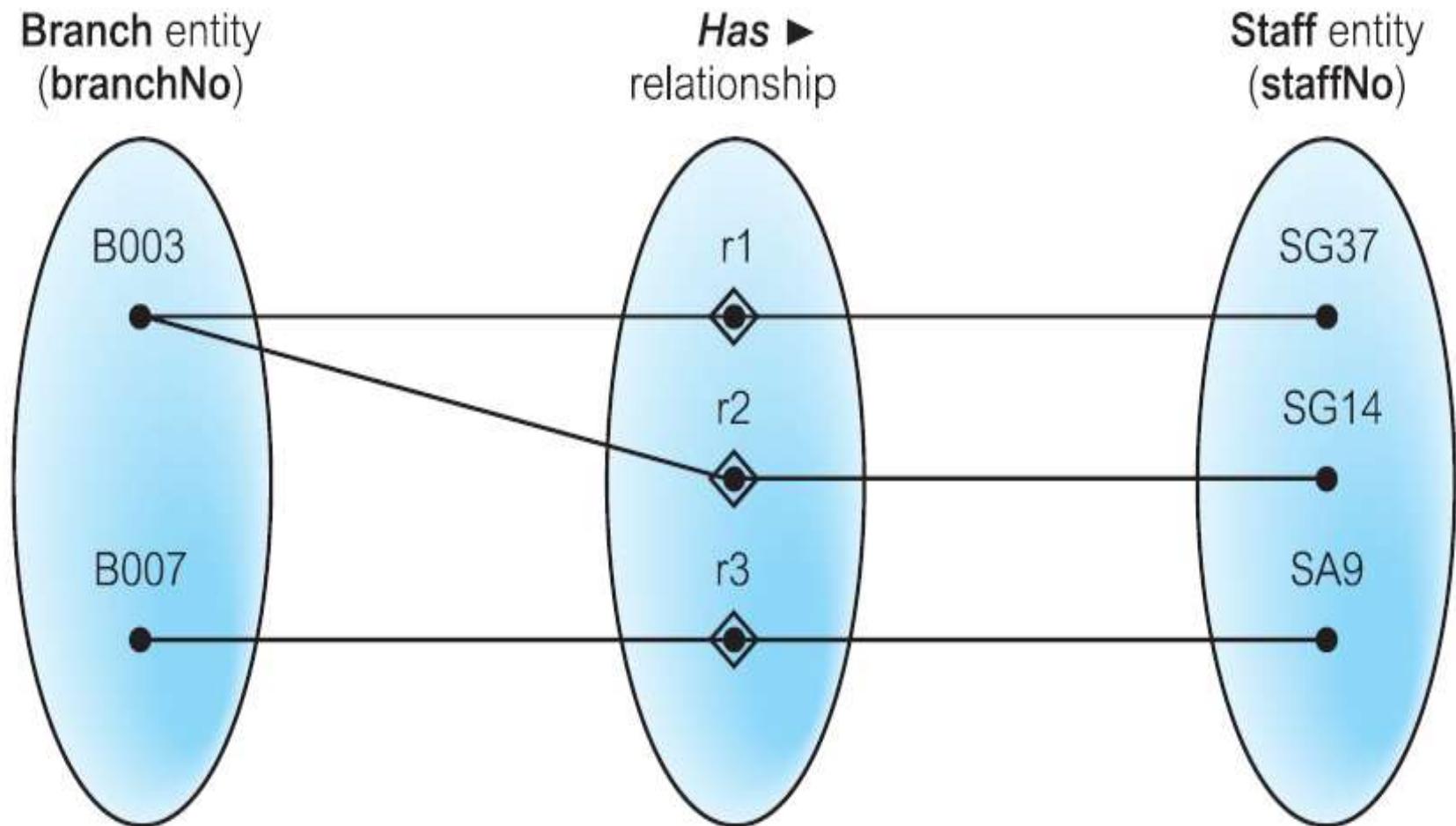
- set of meaningful associations among entity sets

# Relationship occurrence

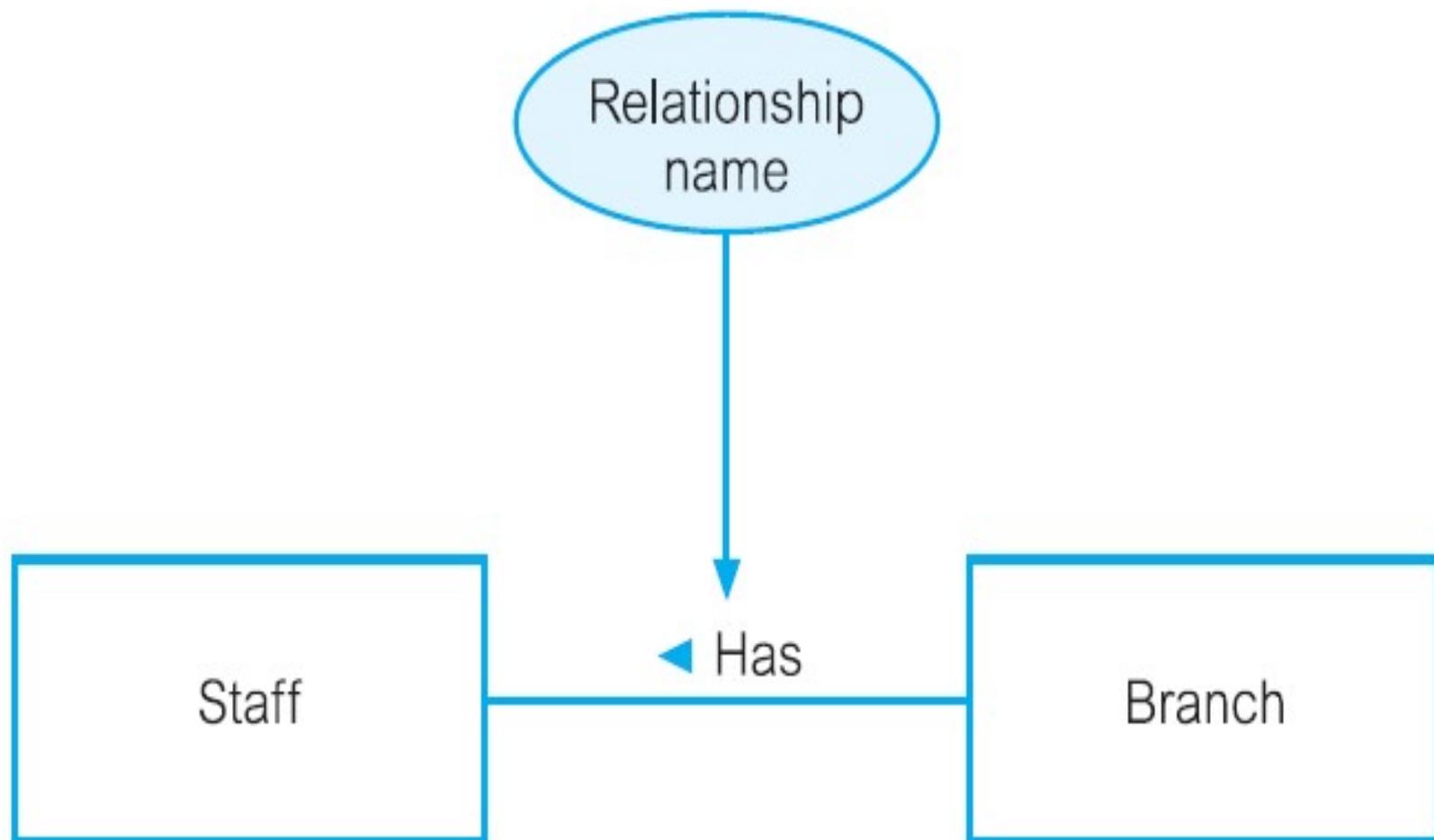
- uniquely identifiable association that includes one occurrence from each participating entity set

- relationship called *Has*, which represents association between Branch and Staff entities, that is Branch Has Staff
- each occurrence of *Has* relationship associates one Branch entity occurrence with one Staff entity occurrence

# occurrences of *Has* relationship



# Diagram of Branch *Has* Staff relationship



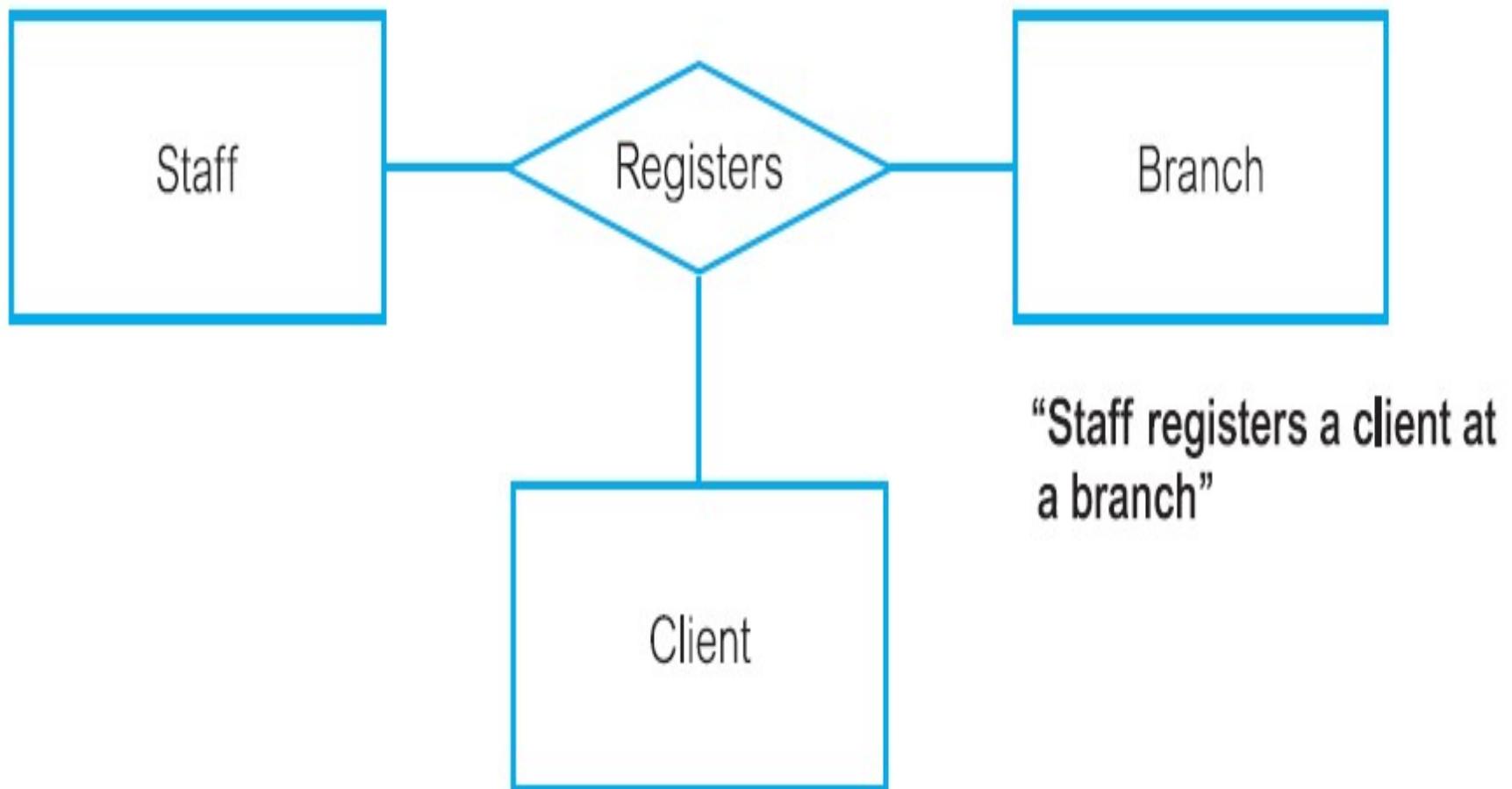
- each entity set is shown as rectangle, labeled with name of entity, which is normally singular noun
- relationship is named using verb
- entity name should be unique, whenever possible, relationship name should also be unique for given ER model
- relationship is labeled in one direction; name of relationship only makes sense in one direction

# Degree of Relationship

- number of participating entity sets in relationship
- entities involved in relationship are referred to as ***participants***
- number of participants is called ***degree***
- relationship of degree two is called ***binary***  
most common degree

- term “complex relationship” used to describe relationships with degrees higher than binary
- uses diamond to represent relationships with degrees higher than binary; name of relationship is displayed inside

# example of ternary relationship called *Registers*

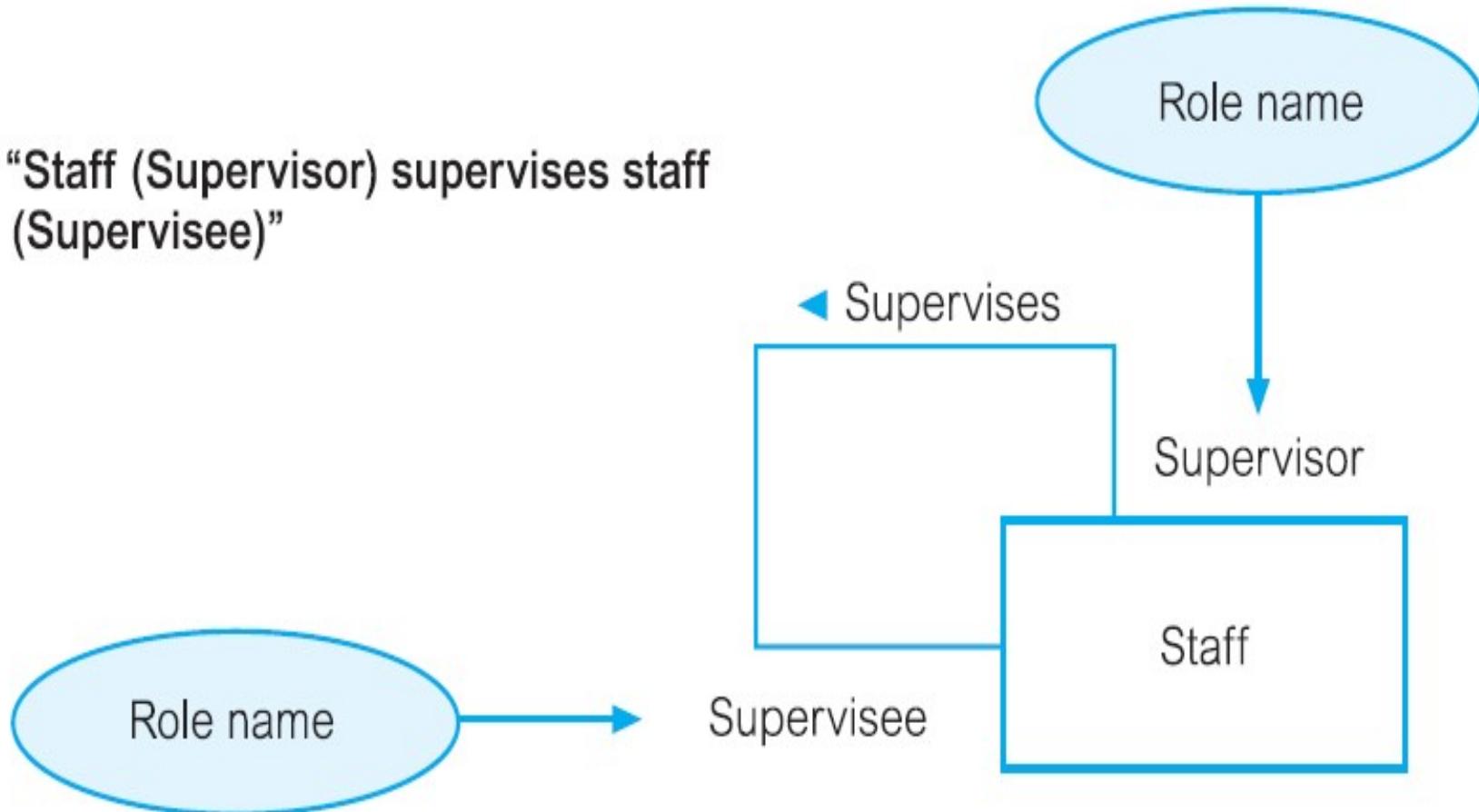


# Recursive Relationship

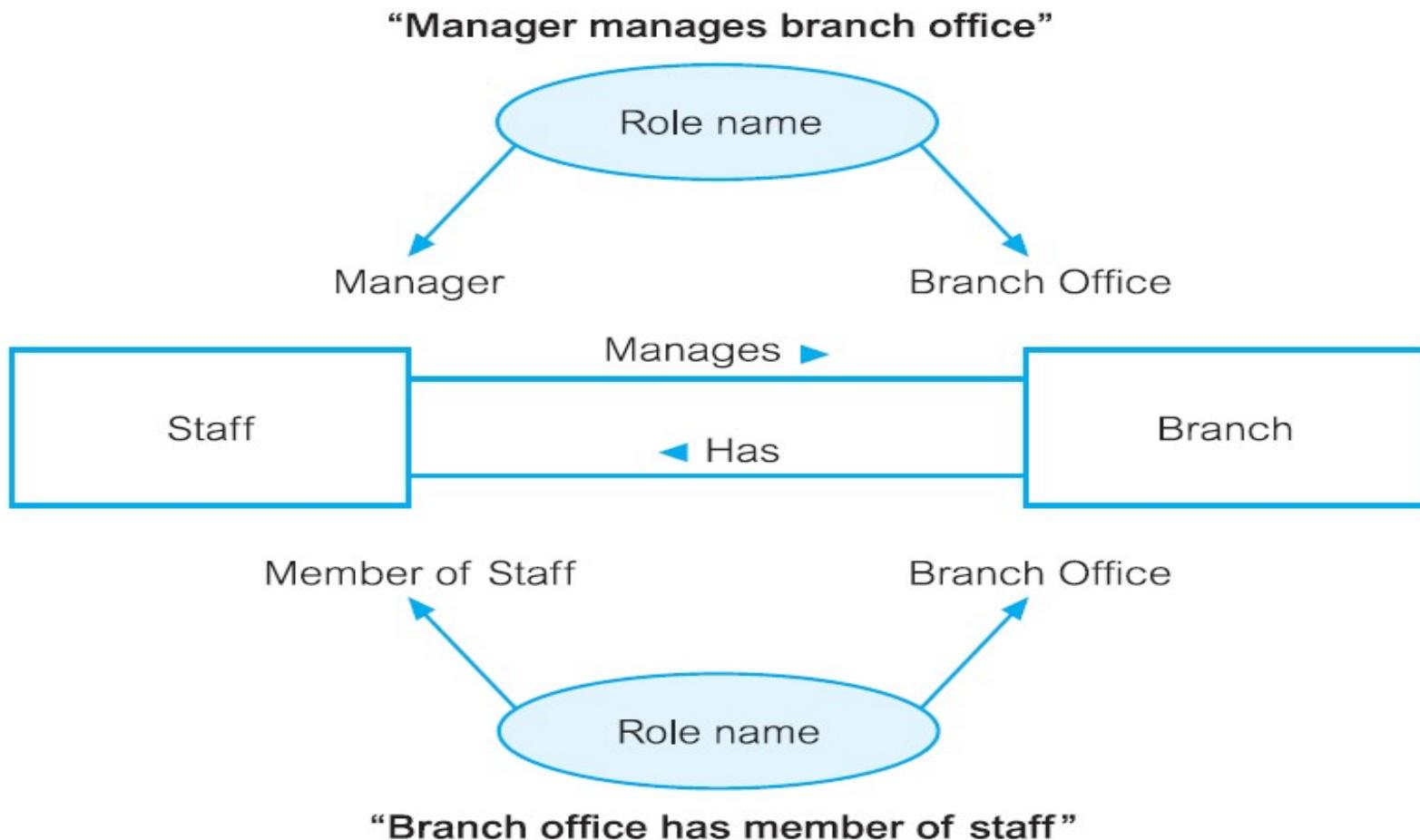
- relationship in which *same* entity type participates more than once in *different roles*
- role names important for recursive relationships to determine function of each participant; usually not required if function of participating entities is unambiguous

# example of recursive relationship *Supervises*

“Staff (Supervisor) supervises staff (Supervisee)”



# example of two relationships with role names



# Attribute

- property of an entity or relationship

# Attribute domain

- set of allowable values for one or more attributes

# Derived attributes

- attribute that represents value that is derivable from value of related attribute or set of attributes, not necessarily in same entity set

# Candidate Key

- minimal set of attributes that uniquely identifies each occurrence of an entity set

# Primary Key

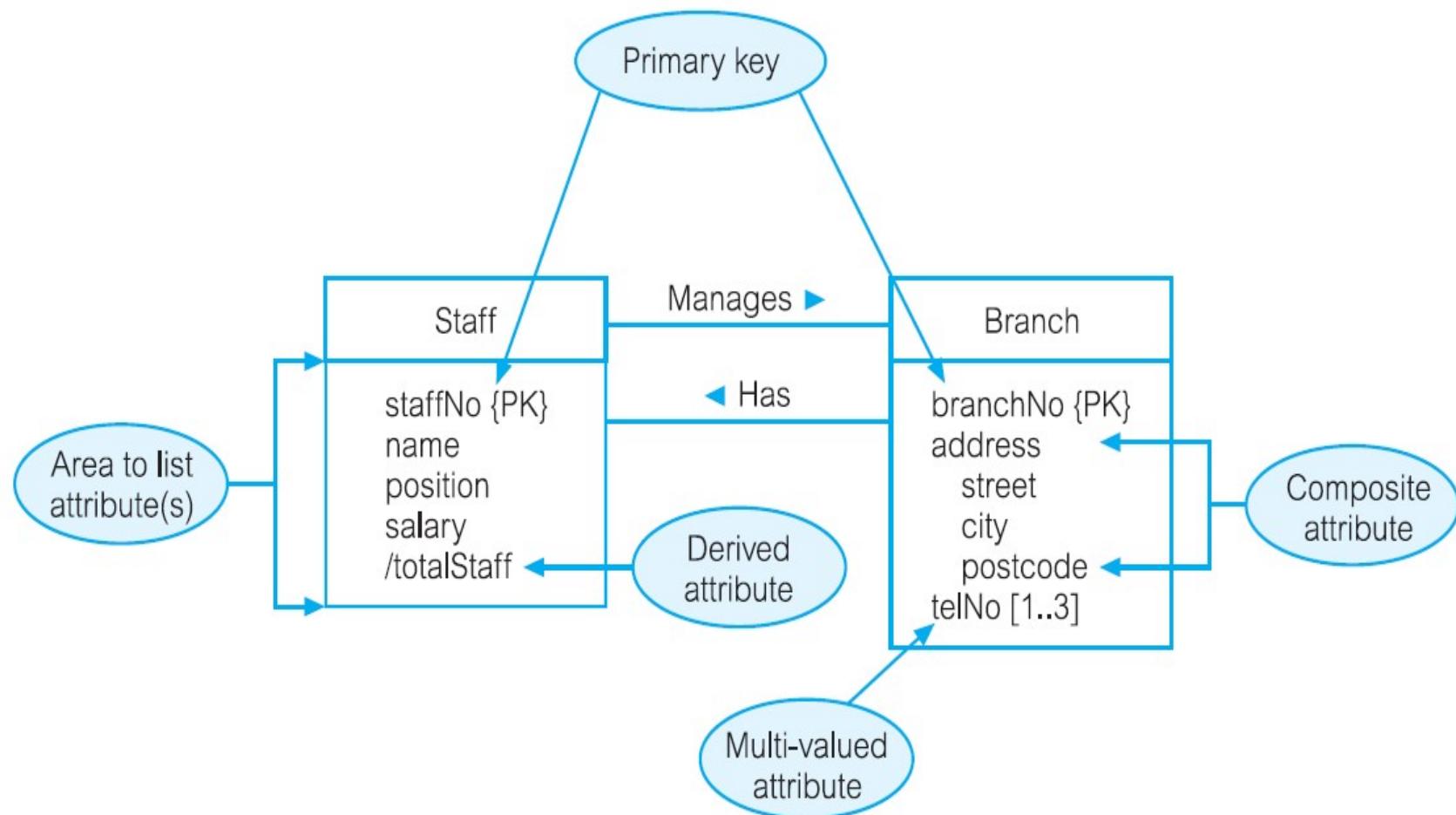
- candidate key that is selected to uniquely identify each occurrence of an entity set

# Alternate Key

- consider that member of staff has unique company-defined staff number (staffNo) and also unique National Insurance Number (NIN) used by government
- choice of primary key for entity is based on considerations of attribute length, minimal number of attributes required, and future certainty of uniqueness
- other referred to as **alternate key**

- divide rectangle representing entity
- upper part displays name of entity
- lower part lists names of attributes
- first attribute(s) to be listed is primary key labeled with tag {PK}
- additional tags that can be used include alternate key {AK}

# example shows ER diagram for Staff Branch entities and attributes



- can classify entity sets as being strong or weak

# Strong entity set

- entity set that is *not* existence-dependent on some other entity set

# Strong entity set

- if its existence does not depend upon existence of another entity type
- Staff, Branch, PropertyForRent, and Client
- each entity occurrence is uniquely identifiable using primary key attribute(s) of that entity set
- for example, we can uniquely identify each member of staff using *staffNo* attribute

# Weak entity set

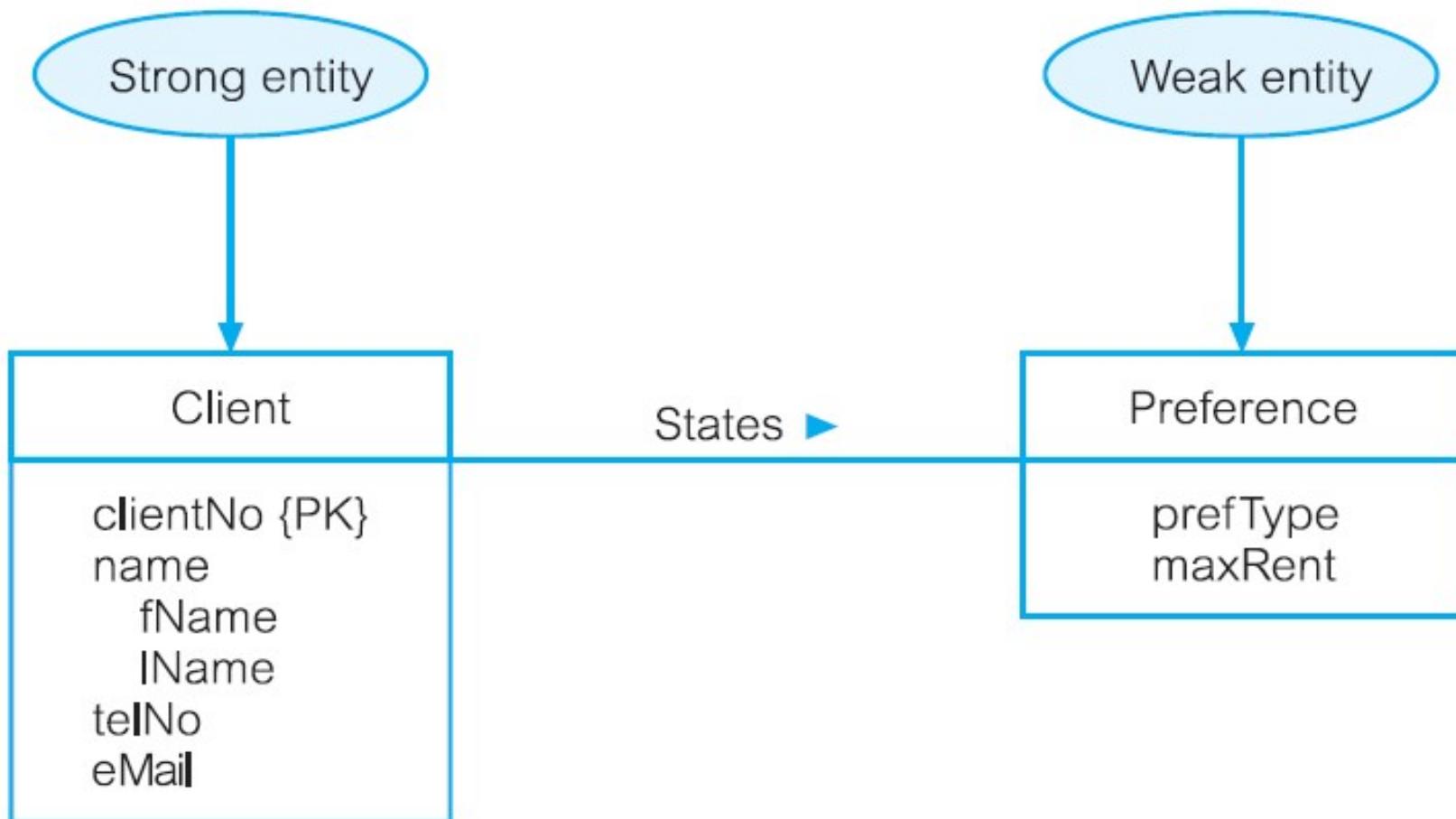
- entity set that is existence-dependent on some other entity set

# Weak entity set

- dependent on existence of another entity set
- each entity occurrence cannot be uniquely identified using only attributes associated with that entity set
- there is no primary key for *Preference* entity
- can uniquely identify each *Preference* only through relationship that preference has with client uniquely identifiable using primary key for *Client* entity set, namely *clientNo*

- *Preference* entity is described as having existence dependency for *Client* entity, which is referred to as being owner entity
- weak entity sets referred to as *child*, *dependent*, or *subordinate* entities
- strong entity types as *parent*, *owner*, or *dominant* entities

# strong entity set *Client* weak entity set *Preference*



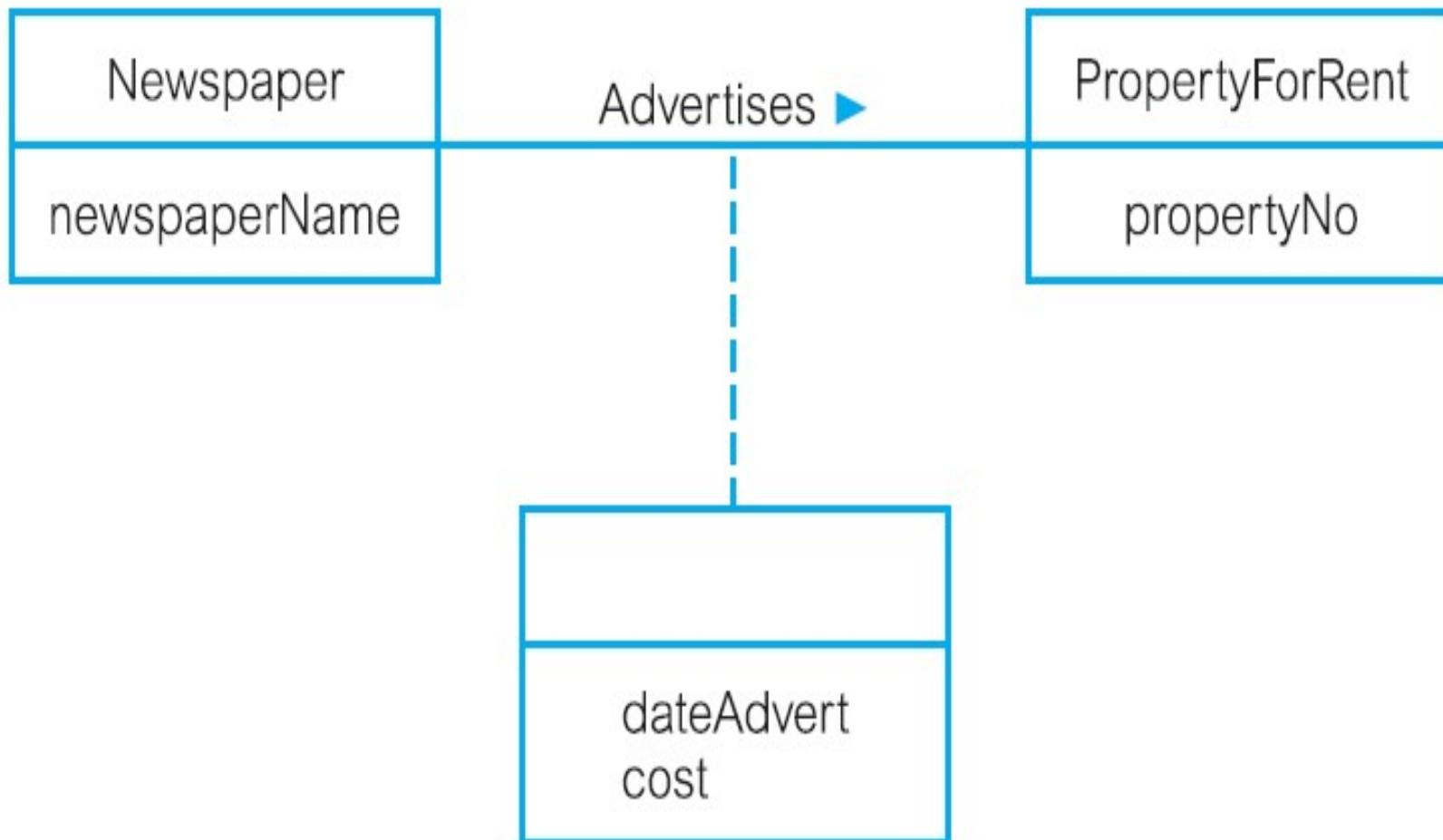
# Attributes on Relationships

- attributes can also be assigned to relationships
- for example, consider relationship *Advertises*, which associates *Newspaper* and *PropertyForRent* entity sets
- to record date property was advertised and cost, we associate this information with *Advertises* relationship as attributes

# Attributes on Relationships

- represent attributes associated with relationship using same symbol as an entity set; rectangle representing the attribute(s) is associated with relationship using dashed line

# example of relationship *Advertises* with attributes *dateAdvert* and *cost*



# Cardinality

- number (or range) of possible occurrences of an entity set that may relate to single occurrence of associated entity set through particular relationship
- describes maximum number of possible relationship occurrences for entity participating in given relationship
- main type of structural constraint on relationships is called ***cardinality***

# Cardinality

- constrains way that entities are related
- representation of business rules established by enterprise
- binary relationships are generally referred to as being one-to-one (1:1), one-to-many (1:m), or many-to-many (m:n)

# Cardinality

- (1:1) - a member of staff manages a (single) branch
- (1:m) - a member of staff oversees (many) properties for rent
- (m:n) – (many) newspapers advertise (many) properties for rent

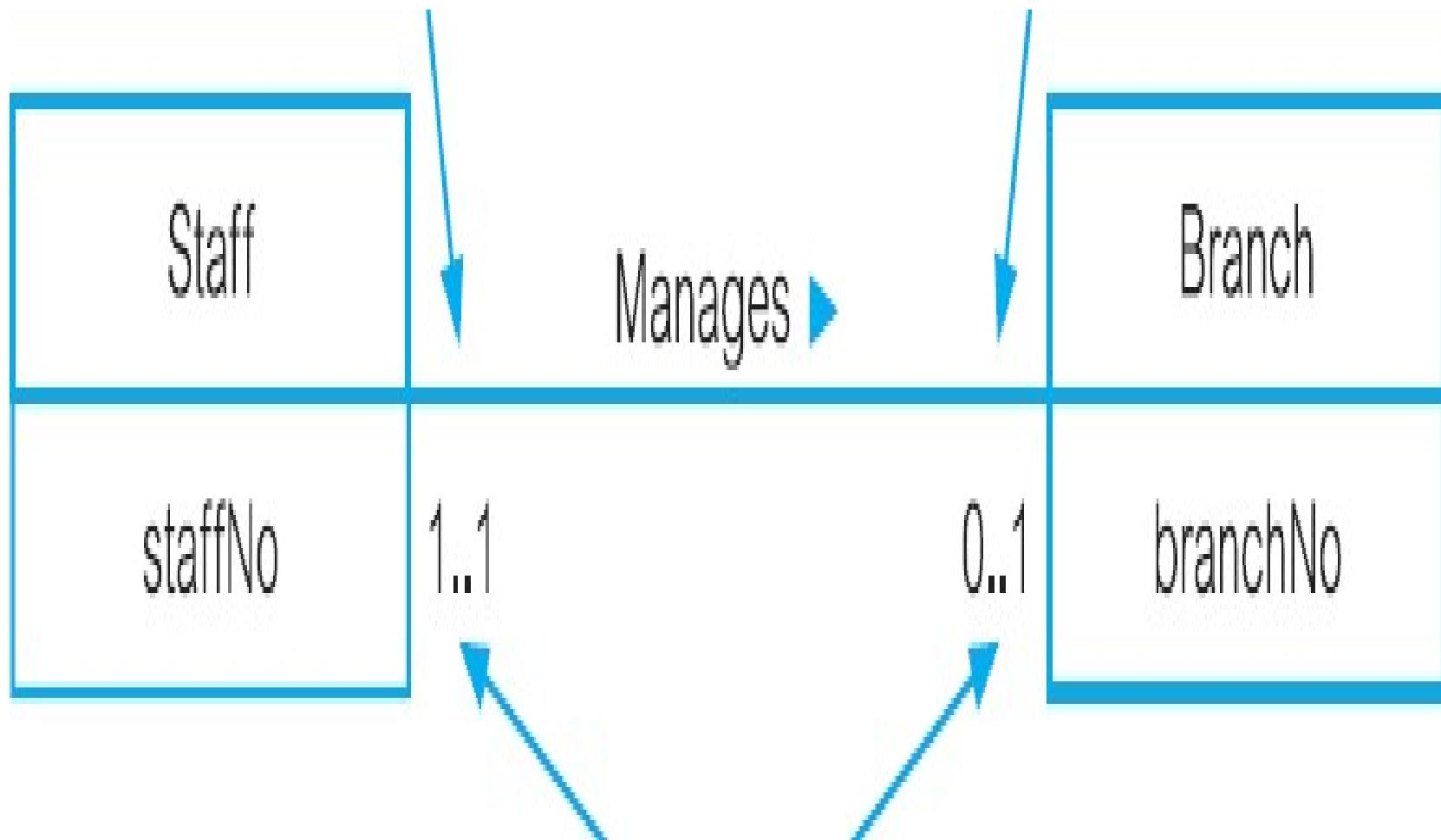
# One-to-One (1:1) Relationships

- relationship *Manages*, which relates *Staff* and *Branch*
- represent each entity occurrence using values for primary key attributes of *Staff* and *Branch* entities, namely *staffNo* and *branchNo*

# One-to-One (1:1) Relationships

- member of staff can manage *zero or one* branch and each branch is managed by *one* member of staff
- there is maximum of one branch for each member of staff involved in relationship and maximum of one member of staff for each branch

# Cardinality of Staff Manages Branch - one-to-one (1:1)



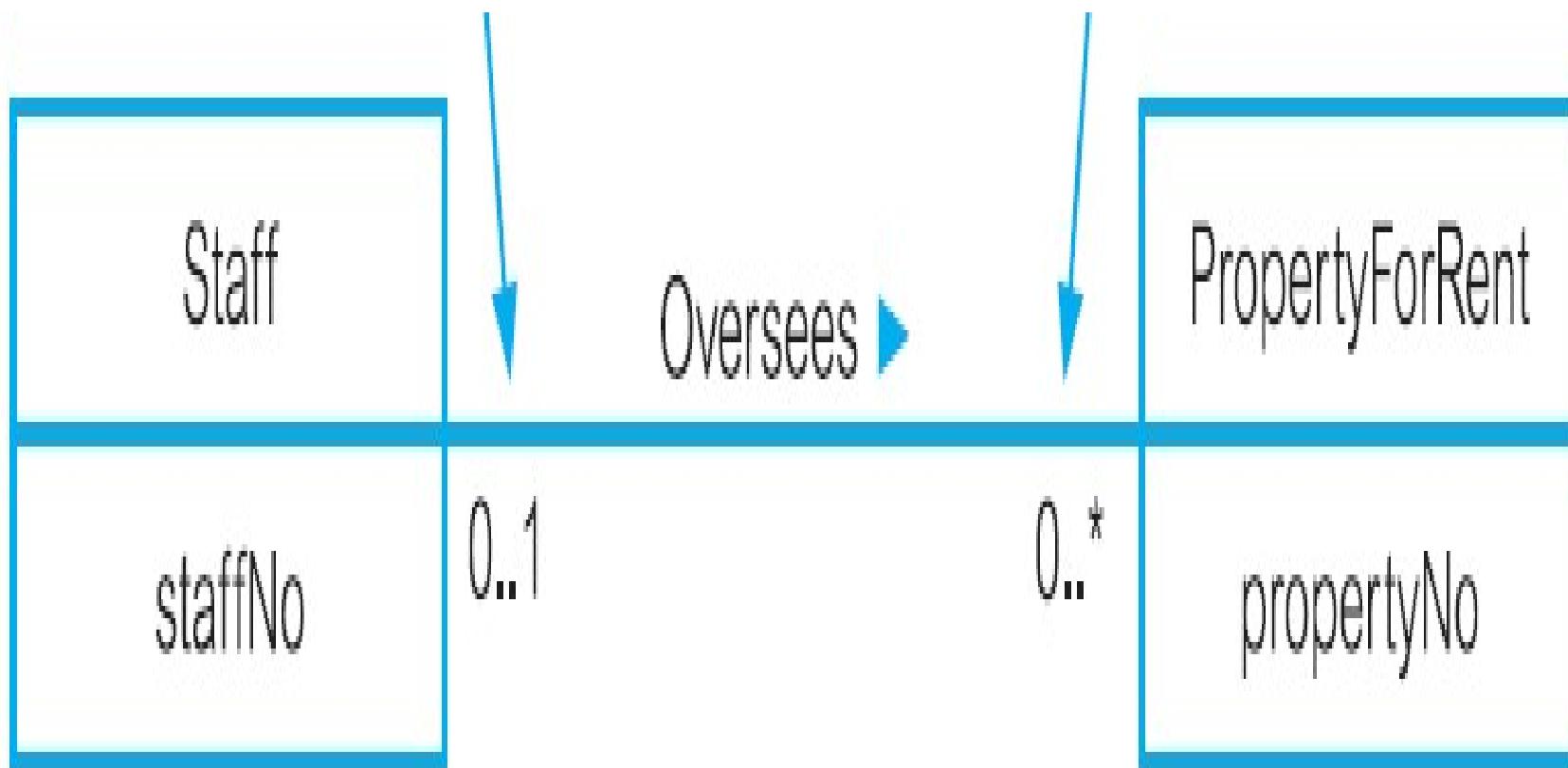
# One-to-Many (1:m) Relationships

- relationship *Oversees*, which relates *Staff* and *PropertyForRent* entity sets
- represent each entity occurrence using values for primary key attributes of *Staff* and *PropertyForRent* entities, namely *staffNo* and *propertyNo*

# One-to-Many (1:m) Relationships

- a member of staff can oversee *zero or more* properties for rent and property for rent is overseen by *zero or one* member of staff
- for members of staff participating in this relationship there are *many* properties for rent, and for properties participating in this relationship there is maximum of *one* member of staff

# One-to-Many (1:m) Relationships



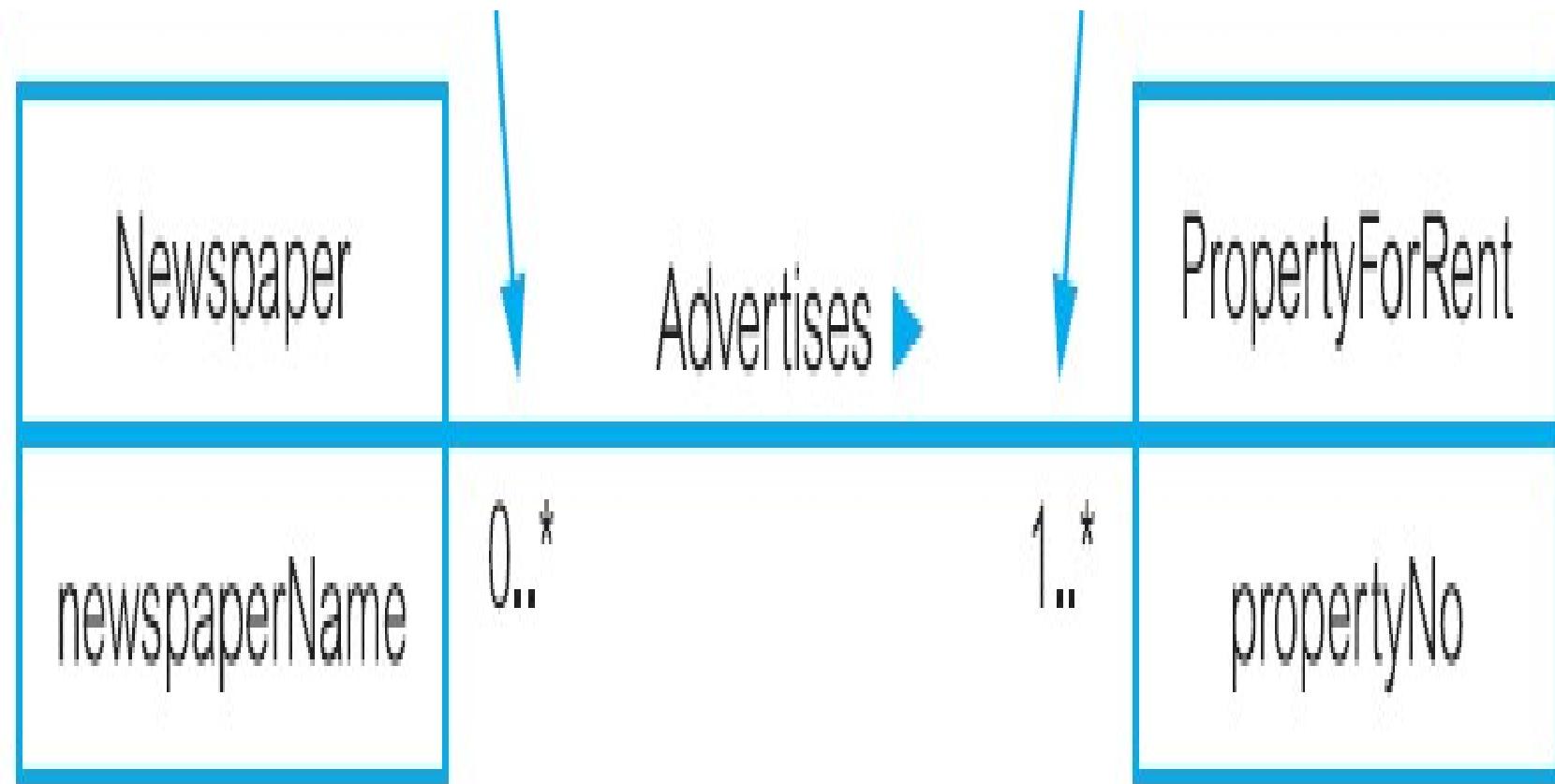
# Many-to-Many (m:n) Relationships

- relationship *Advertises* relates *Newspaper* and *PropertyForRent* entity sets
- represent each entity occurrence using values for primary key attributes of *Newspaper* and *PropertyForRent* entity sets, namely *newspaperName* and *propertyNo*

# Many-to-Many (m:n) Relationships

- one newspaper advertises *one or more* properties for rent and *one* property for rent is advertised in *zero or more* newspapers
- there are *many* properties for rent, and for each property for rent participating in this relationship there are *many* newspapers

# Many-to-Many (m:n) Relationships



# Participation

- determines whether all or only some entity occurrences participate in relationship
- participation constraint represents whether all entity occurrences are involved in particular relationship (referred to as ***mandatory*** participation) or only some (referred to as ***optional*** participation)

# Participation

- optional participation is represented as minimum value of 0
- mandatory participation is shown as minimum value of 1
- participation for given entity in relationship is represented by minimum value on *opposite side of relationship*

# Participation

- optional participation for *Staff* entity in *Manages* relationship is shown as minimum value of 0 for cardinality beside *Branch* entity
- mandatory participation for *Branch* entity in *Manages* relationship is shown as minimum value of 1 for cardinality beside *Staff* entity

# Problems: connection traps

- occur due to misinterpretation of meaning of certain relationships: *fan traps* and *chasm traps*
- to identify connection traps we must ensure that meaning of relationship is fully understood and clearly defined
- if we do not understand relationships we may create model that is not true representation of “real world”

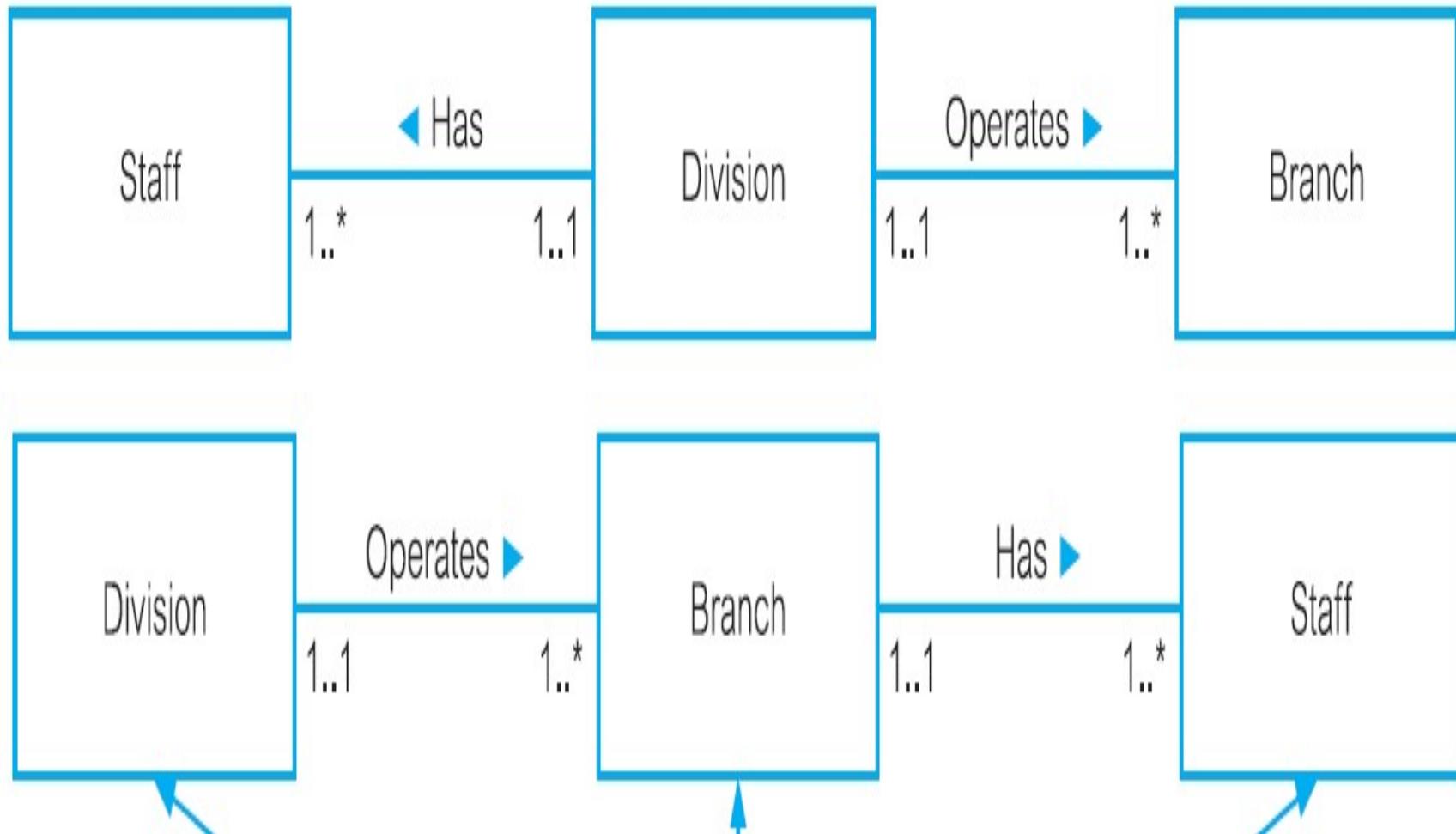
# Fan Trap

- model represents relationship between entity sets, but pathway between certain entity occurrences is ambiguous
- exist where two or more 1:m relationships fan out from same entity

# Fan Trap

- relationships (1:m) *Has* and *Operates* emanating from same entity *Division*
- a single division operates *one or more* branches and has *one or more* staff
- problem arises when we want to know which members of staff work at particular branch

# Example of Fan Trap



# Chasm Trap

- model suggests existence of relationship between entity sets, but pathway does not exist between certain entity occurrences

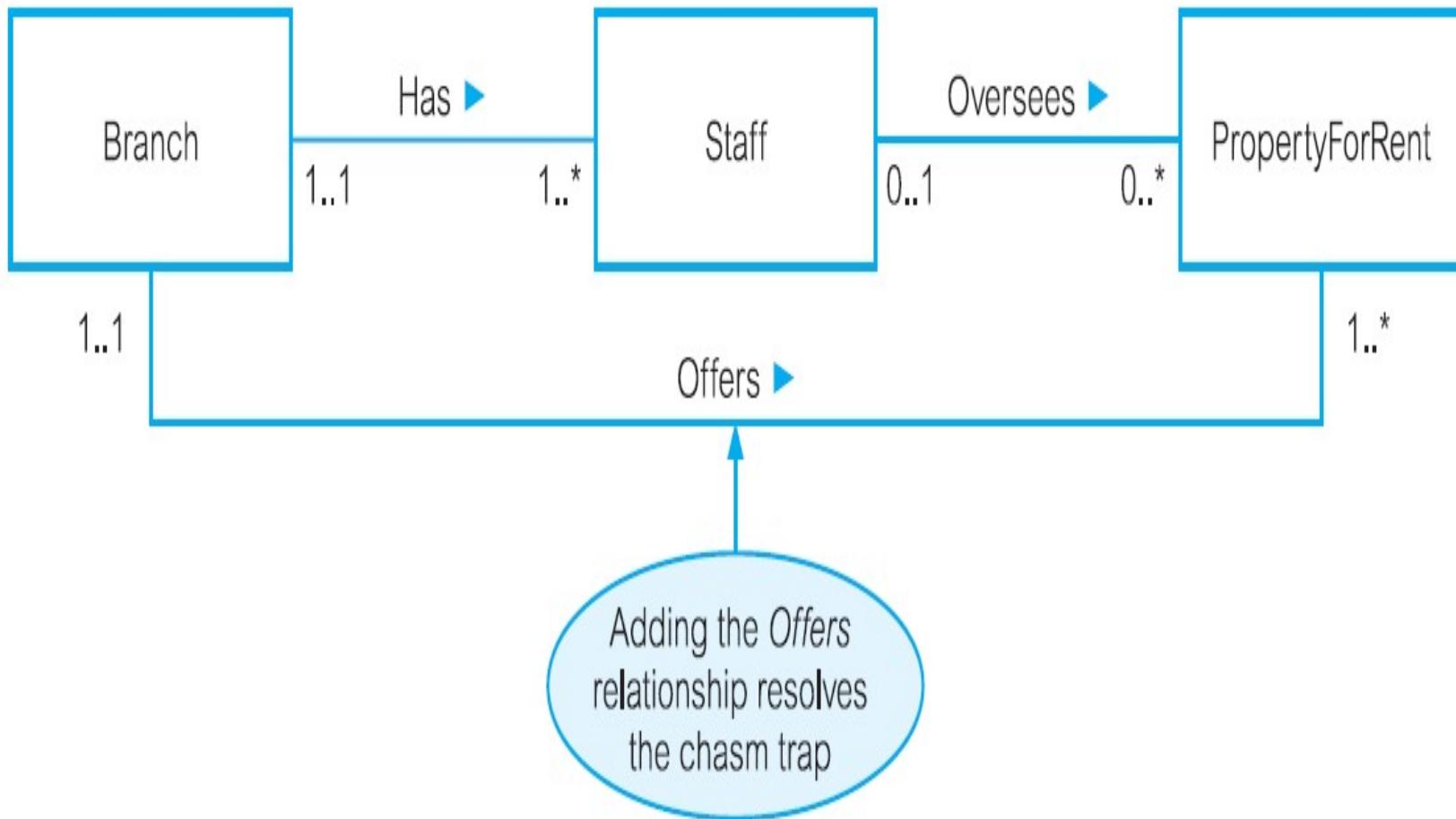
# Chasm Trap

- occur where there are one or more relationships with minimum multiplicity of zero (that is, optional participation) forming part of pathway between related entities

# Chasm Trap

- potential chasm trap illustrated in relationships between *Branch*, *Staff*, and *PropertyForRent* entities; model represents facts that single branch has *one or more* staff who oversee *zero or more* properties for rent; not all staff oversee property and not all properties are overseen
- problem arises when we want to know which properties are available at each branch

# Example of Chasm Trap



# **SPECIALIZATION / GENERALIZATION**

# Specialization/Generalization

- associated with special types of entities known as superclasses and subclasses, and process of attribute inheritance
- hierarchy of entities containing superclasses and subclasses

# Superclass

- entity set that includes one or more distinct subgroupings of its occurrences, which must be represented in data model

# Subclass

- distinct subgrouping of occurrences of entity set, which must be represented in data model

# for example

- entities that are members of *Staff* entity set may be classified as *Manager*, *SalesPersonnel*, and *Secretary*
- *Staff* entity is referred to as superclass of
- *Manager*, *SalesPersonnel*, and *Secretary* subclasses
- superclass/subclass relationship

# Superclass/Subclass Relationships

- each member of subclass is also member of superclass; but has distinct role
- relationship between superclass and subclass is one-to-one (1:1)

# Superclass/Subclass Relationships

- some superclasses may contain overlapping subclasses (illustrated by member of *Staff* who is both *Manager* and *Sales Personnel*)
- not every member of superclass is necessarily member of subclass (members of *Staff* without distinct job role such as *Manager* or *Sales Personnel*)

# Possible (but not indicated)

- All Up – use superclass without any subclasses
- All Down – use many subclasses without superclass
- use superclasses and subclasses to avoid describing different types of *Staff* with possibly different attributes within single entity

- avoids describing similar concepts more than once - saving time for designer
- adds more semantic information to design in form that is familiar to many
- “*Manager IS-A member of Staff*” - communicates significant semantic content in concise form

- for example, *Sales Personnel* may have special attributes such as *salesArea* and *carAllowance*
- if all are described by single *Staff* entity, this may result in a lot of nulls for job-specific attributes

- can also show relationships that are associated with only particular types of staff (subclasses) and not with staff, in general;
- for example, *Sales Personnel* may have distinct relationships that are not appropriate for all *Staff*, such as *SalesPersonnel Uses Car*

- *Sales Personnel* have common attributes with other *Staff*, such as *staffNo*, *name*, *position*, and *salary*

# relation AllStaff

The diagram illustrates the decomposition of the **AllStaff** relation into four smaller relations based on staff roles. The original relation has 9 columns, and each role is assigned specific attributes:

- Attributes appropriate for all staff:** staffNo, name, position, salary
- Attributes appropriate for branch Managers:** mgrStartDate, bonus
- Attributes appropriate for Sales Personnel:** salesArea, carAllowance
- Attribute appropriate for Secretarial staff:** typingSpeed

Arrows point from the column headers to their respective attribute groups.

| staffNo | name          | position       | salary | mgrStartDate | bonus | salesArea | carAllowance | typingSpeed |
|---------|---------------|----------------|--------|--------------|-------|-----------|--------------|-------------|
| SL21    | John White    | Manager        | 30000  | 01/02/95     | 2000  |           |              |             |
| SG37    | Ann Beech     | Assistant      | 12000  |              |       |           |              |             |
| SG66    | Mary Martinez | Sales Manager  | 27000  |              |       | SA1A      | 5000         |             |
| SA9     | Mary Howe     | Assistant      | 9000   |              |       |           |              |             |
| SL89    | Stuart Stern  | Secretary      | 8500   |              |       |           |              | 100         |
| SL31    | Robert Chin   | Snr Sales Asst | 17000  |              |       | SA2B      | 3700         |             |
| SG5     | Susan Brand   | Manager        | 24000  | 01/06/91     | 2350  |           |              |             |

# Attribute Inheritance

- subclass represents same “real world” object as in superclass, may possess subclass-specific attributes, as well as those associated with superclass
- *SalesPersonnel* subclass inherits all attributes of *Staff* superclass - *staffNo*, *name*, *position*, and *salary* together with those specifically associated with *SalesPersonnel* subclass - *salesArea* and *carAllowance*

# Hierarchy

- subclass is an entity in its own right and so it may also have one or more subclasses
- Specialization hierarchy (for example, *Manager* is a specialization of *Staff*)
- Generalization hierarchy (for example, *Staff* is a generalization of *Manager*)
- IS-A hierarchy (for example, *Manager* IS-A (member of) *Staff*)

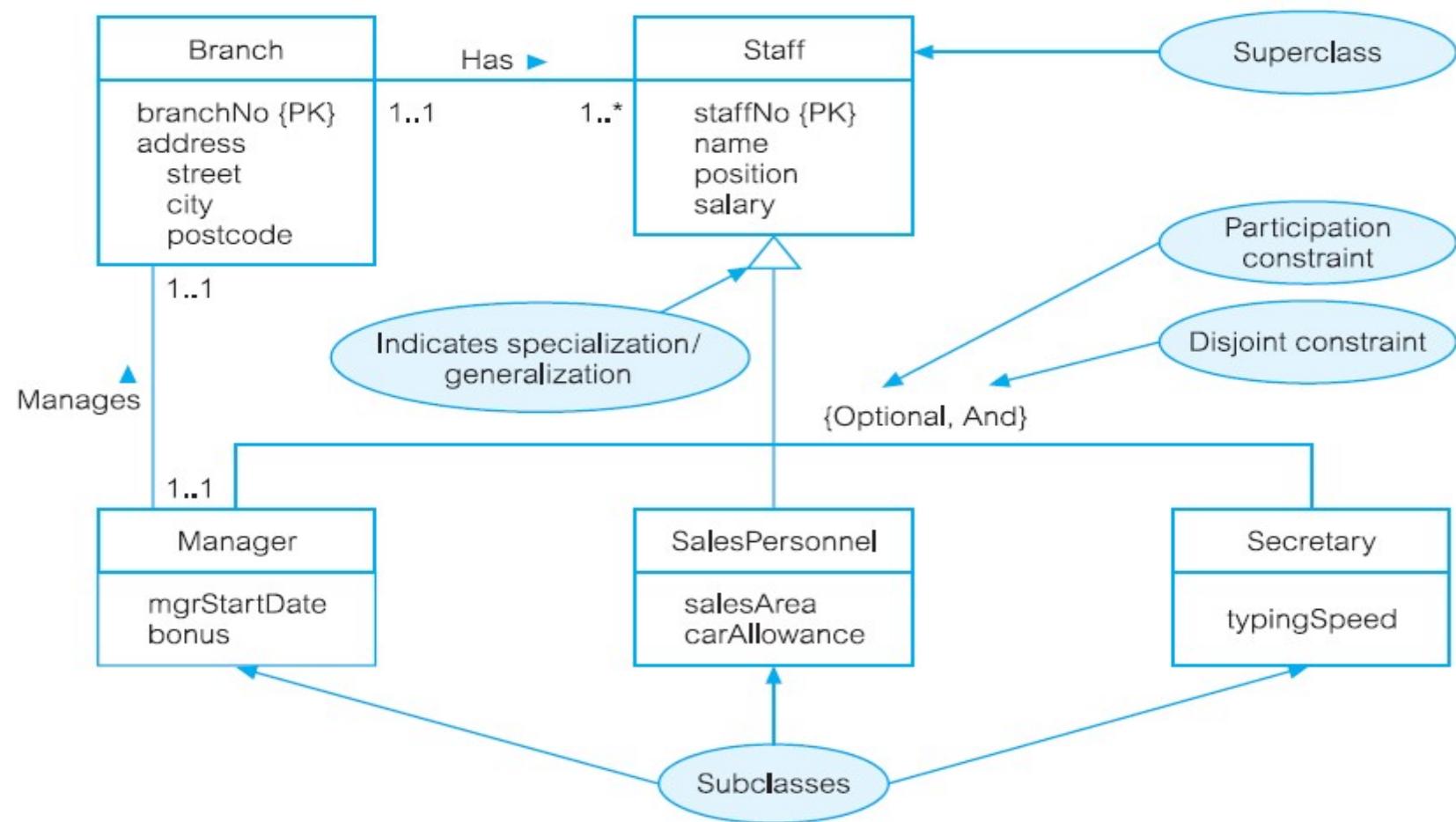
# Specialization process

- process of maximizing differences between members of an entity by identifying their distinguishing characteristics
- top-down approach to defining set of superclasses and their related subclasses

# Generalization process

- process of minimizing differences between entities by identifying their common characteristics
- bottom-up approach, that results in identification of generalized superclass from original entity types

# Specialization/generalization of Staff subclasses representing job roles

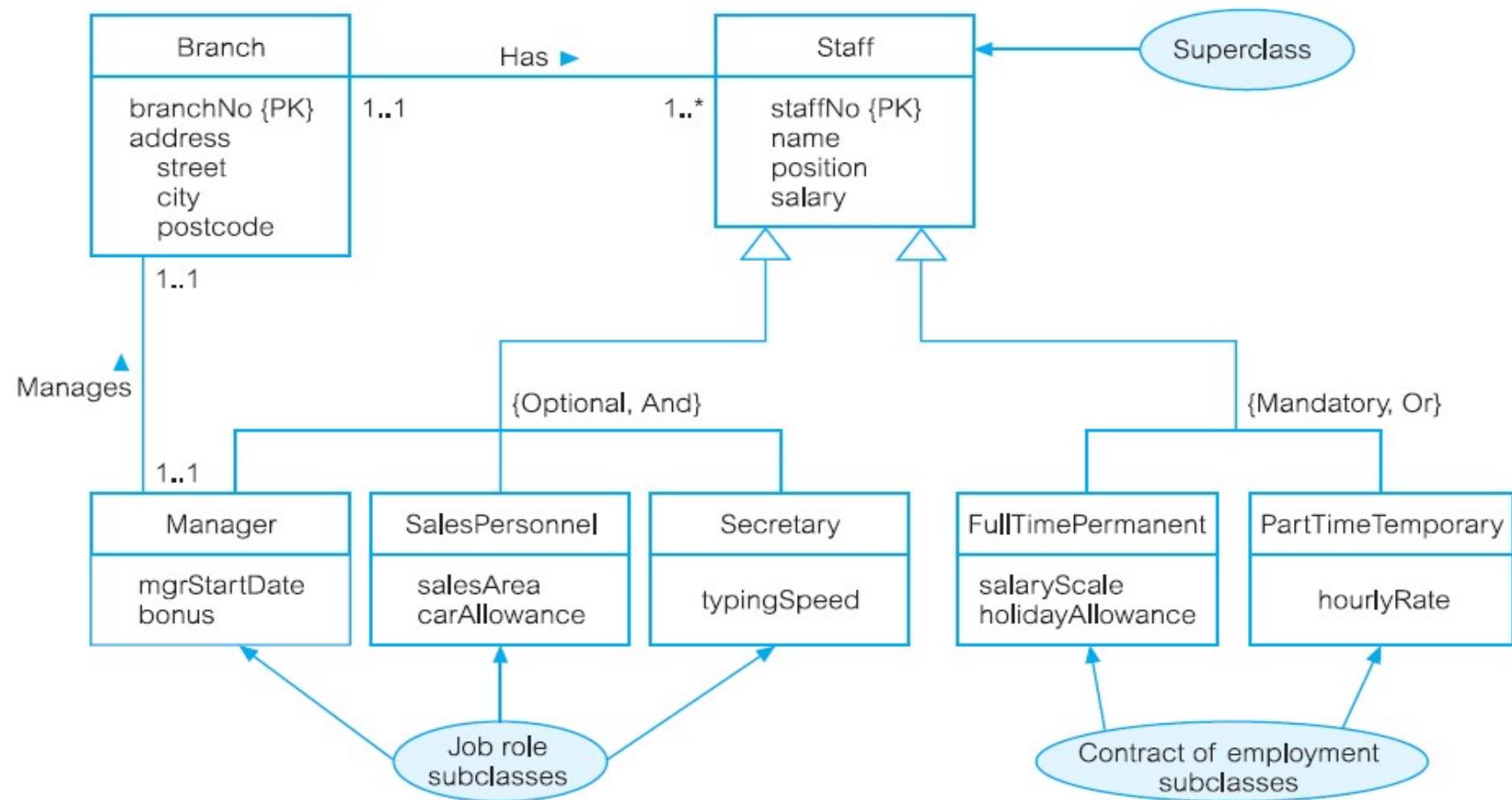


- may have several specializations of same entity based on different distinguishing characteristics

# for example

- another specialization of *Staff* entity may produce subclasses *FullTimePermanent* and *PartTimeTemporary*, which distinguishes between types of employment contract for members of staff
- attributes that are specific to *FullTimePermanent* (*salaryScale* and *holidayAllowance*) and *PartTimeTemporary* (*hourlyRate*)

# Staff entity: job roles & contracts of employment



# Constraints on Specialization/Generalization

- two constraints that may apply to specialization/generalization called **participation constraints** and **disjoint constraints**

# Participation constraints

- determines whether every member in superclass must participate as member of subclass
- may be mandatory or optional
- mandatory participation specifies that every member in superclass must also be member of a subclass

# Disjoint constraints

- describes relationship between members of subclasses and indicates whether it is possible for member of superclass to be member of one, or more than one, subclass - applies when superclass has more than one subclass
- if subclasses are disjoint, then an entity occurrence can be member of only one of subclasses

# Review Questions

- why is ER model considered top-down approach?
- describe four basic components of ER
- provide examples of unary, binary, ternary, relationships
- how basic ER components are represented in ER diagram
- describe what cardinality constraint represents for relationship
- describe how fan and chasm traps can occur

# Review Questions

- describe situations that would call for an specialization/generalization relationship
- describe and illustrate using an example process of attribute inheritance

# Database

# Movies

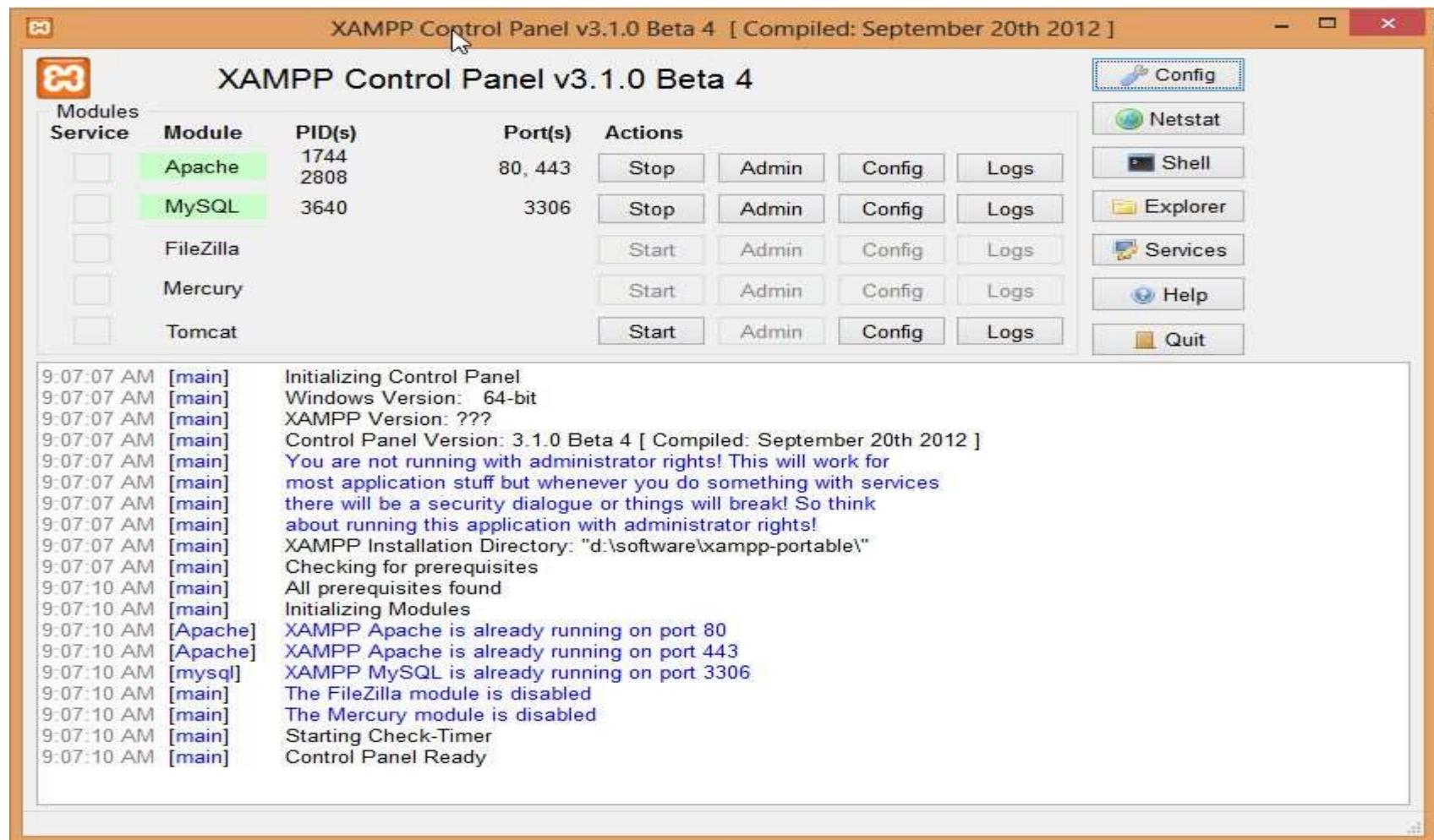
- [www.imdb.com](http://www.imdb.com)
- Internet Movie DataBase

# XAMPP

- free and open source cross-platform web server solution stack package, consisting mainly of the Apache HTTP Server (TomCat), MySQL database, and interpreters for scripts written in the PHP and Perl programming languages
- [www.apachefriends.org/en/xampp.html](http://www.apachefriends.org/en/xampp.html)
- Cross-platform (Linux, Windows, Solaris, Mac OS X)
- LAMP – Linux Apache MySQL Php

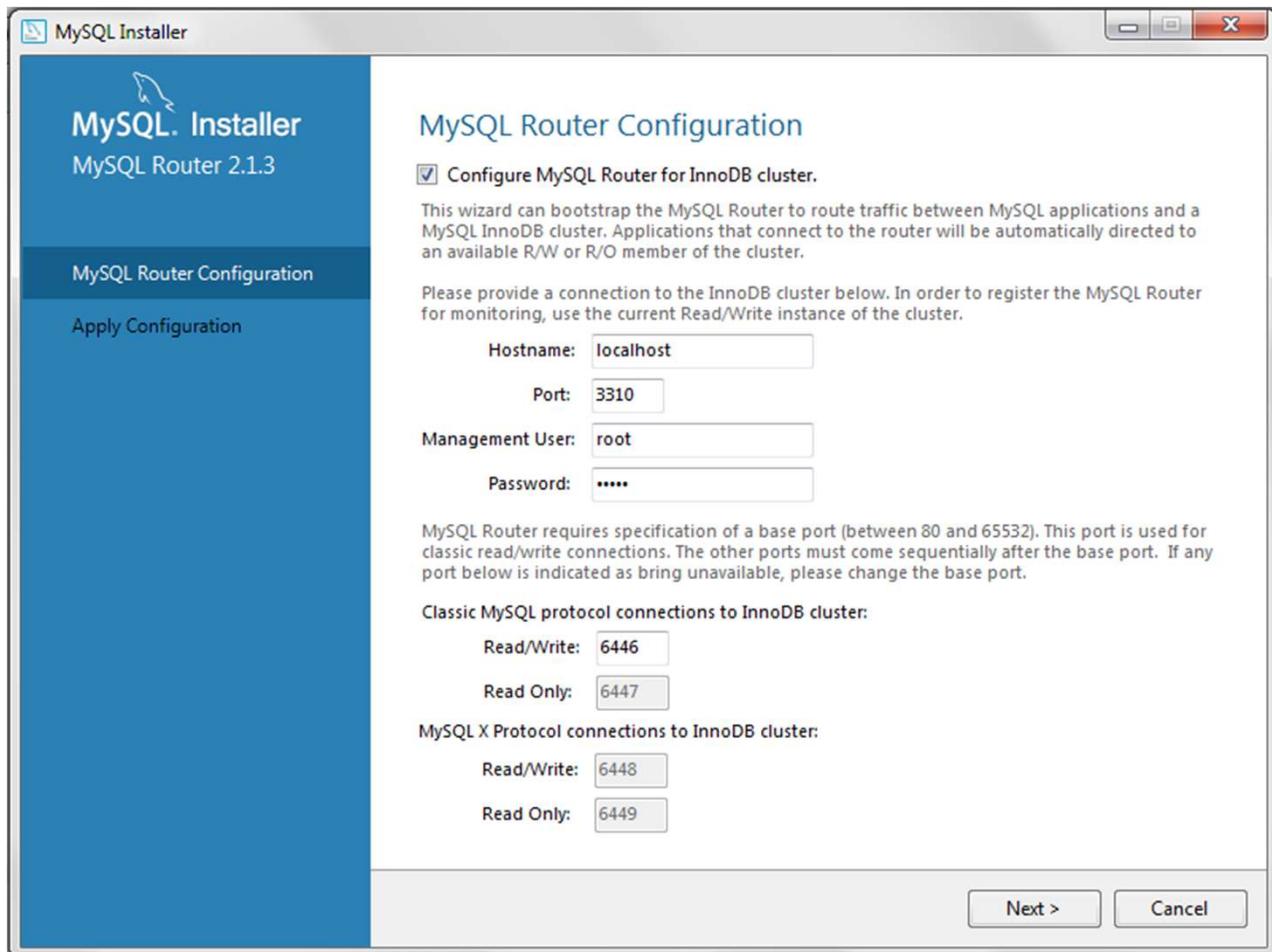
# Xampp Portable

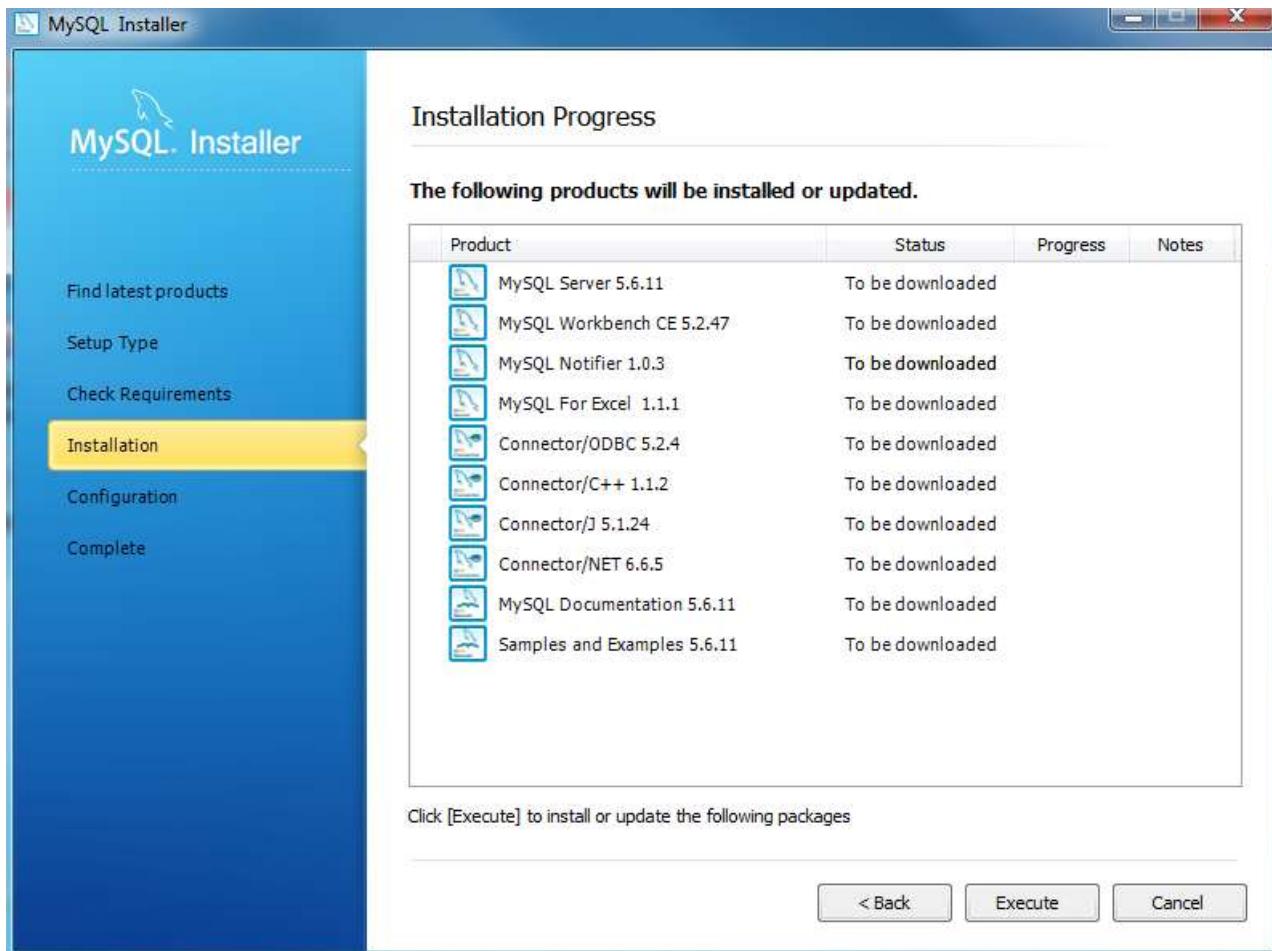
- Xampp\_start
- Xampp\_stop
- Xampp\_control

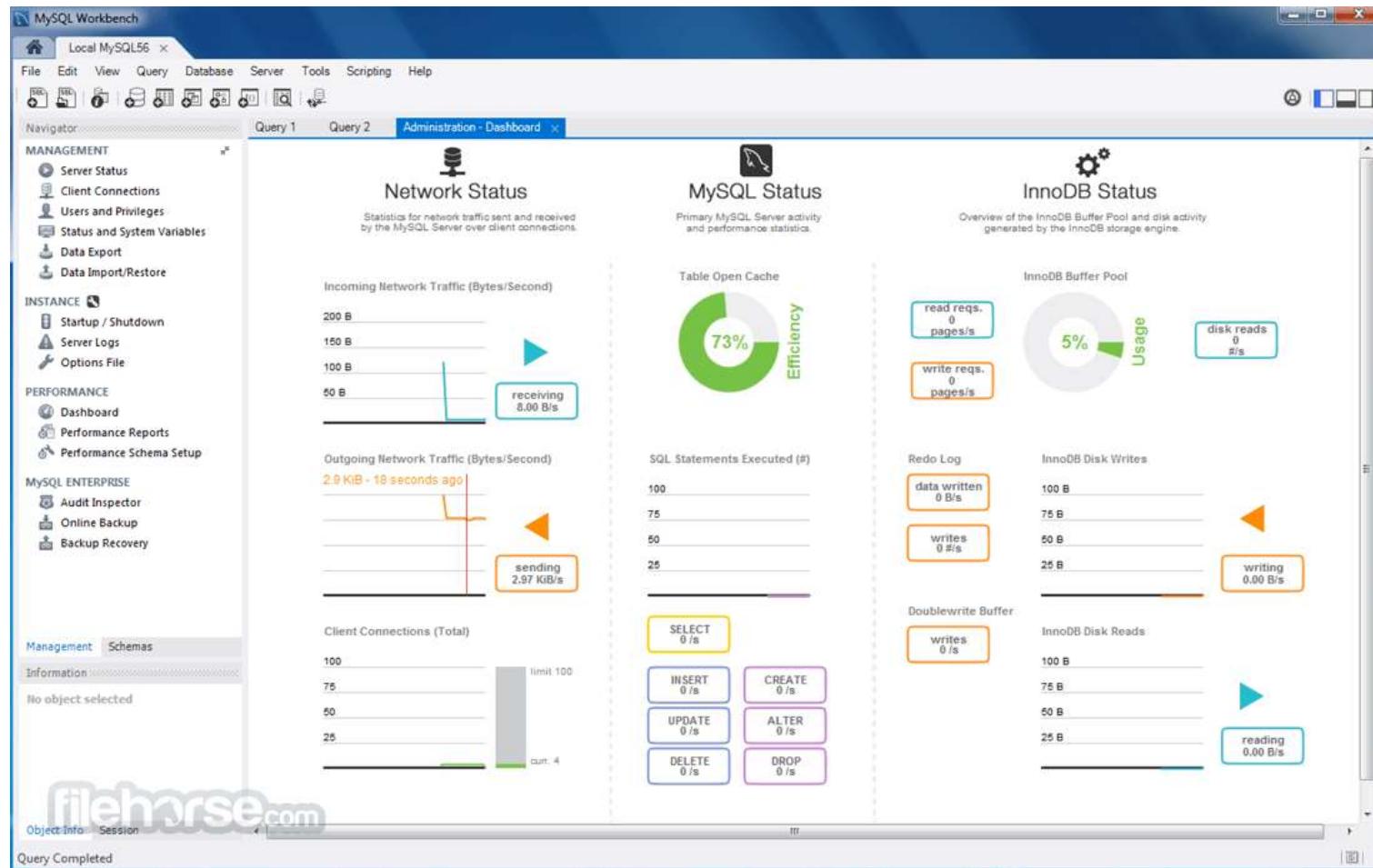


<https://dev.mysql.com/downloads/>

- MySQL Community Server (GPL)
- (Current Generally Available Release: **8.0.13**)
- MySQL Community Server is the world's most popular open source database.
- Prerequisites – Visual C Redistributable (2015)





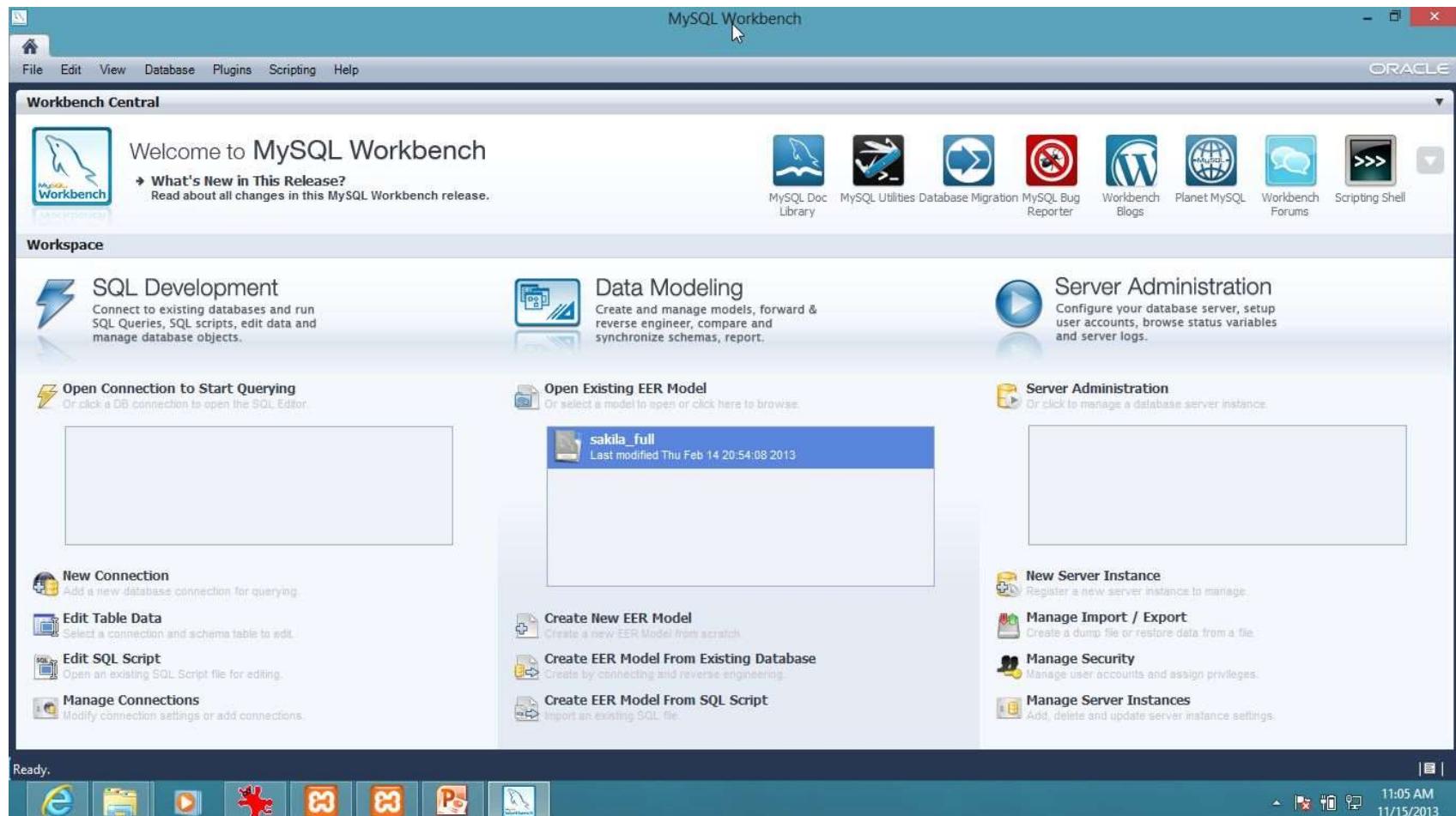


Lab Notes 07 – MySQL Database

# MySQL Workbench

- GNU General Public License or proprietary EULA
- visual database design tool that integrates SQL development, administration, database design, creation and maintenance into a single integrated development environment for the MySQL database system
- Oracle Corporation: [mysqlworkbench.org](http://mysqlworkbench.org)
- Cross-platform

# MySQL Workbench



- SQL Development
- Data Modeling
- DataBase Administration

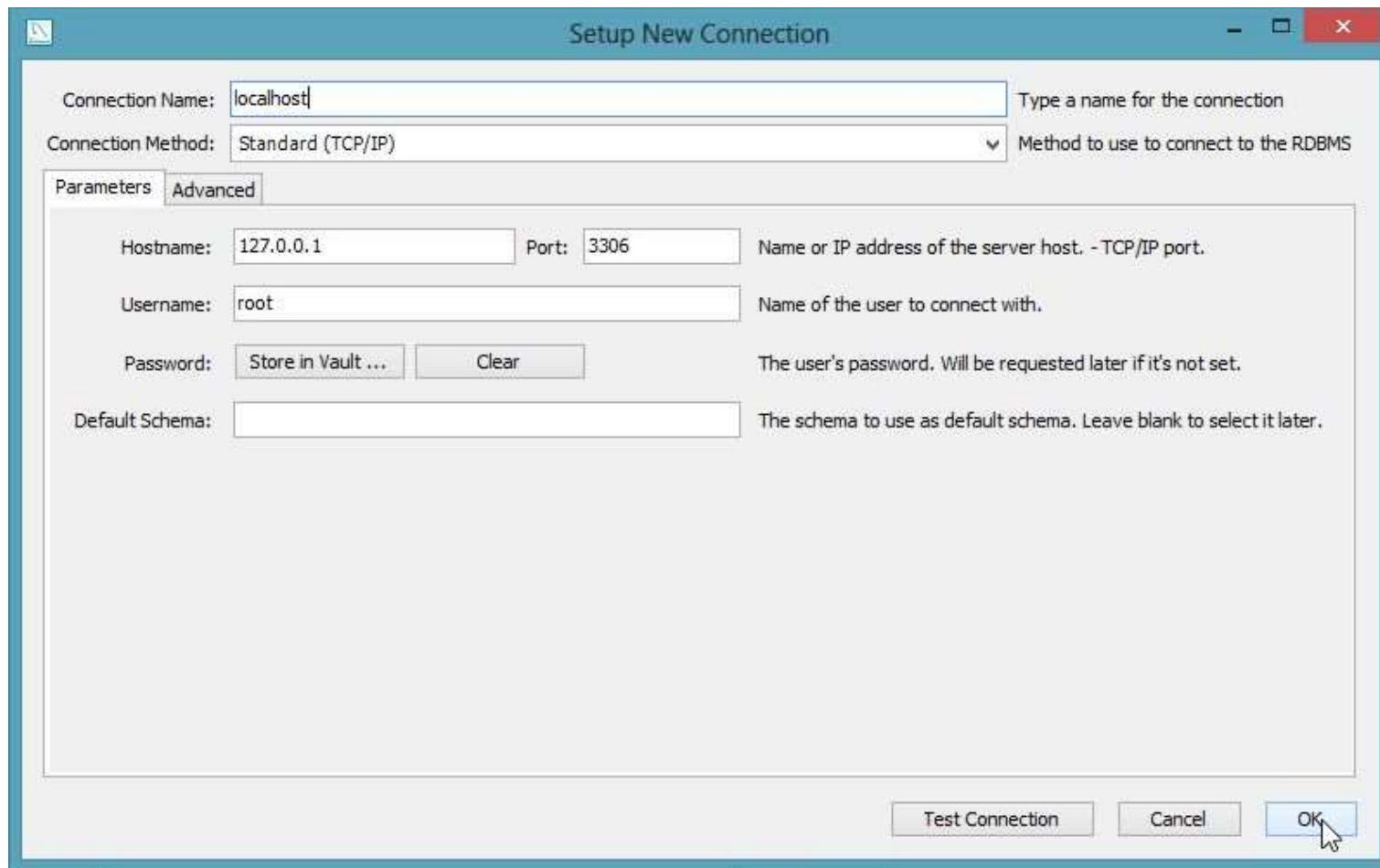
# SQL Development

# Create New Connection to database server

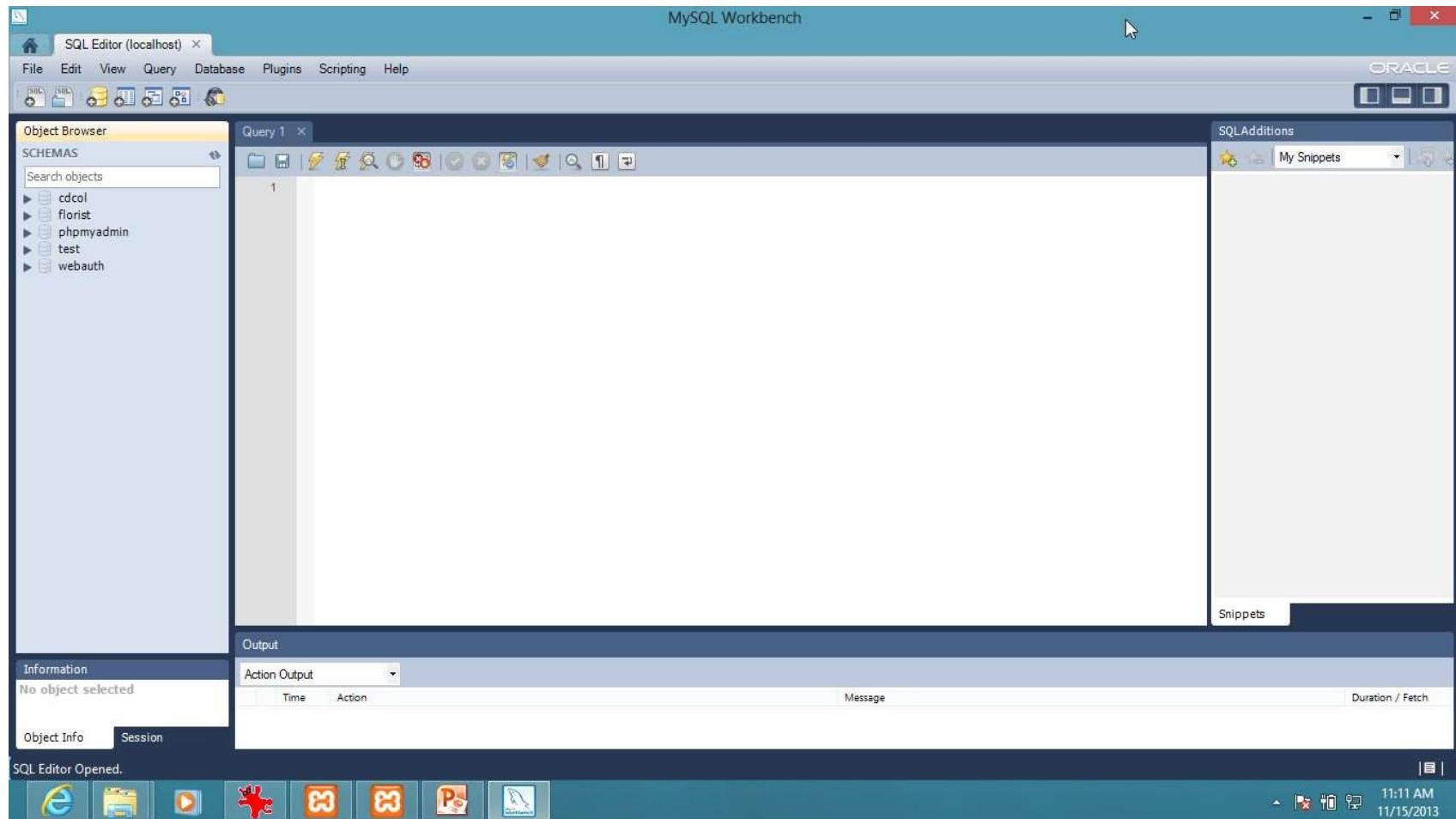


# Setup New Connection

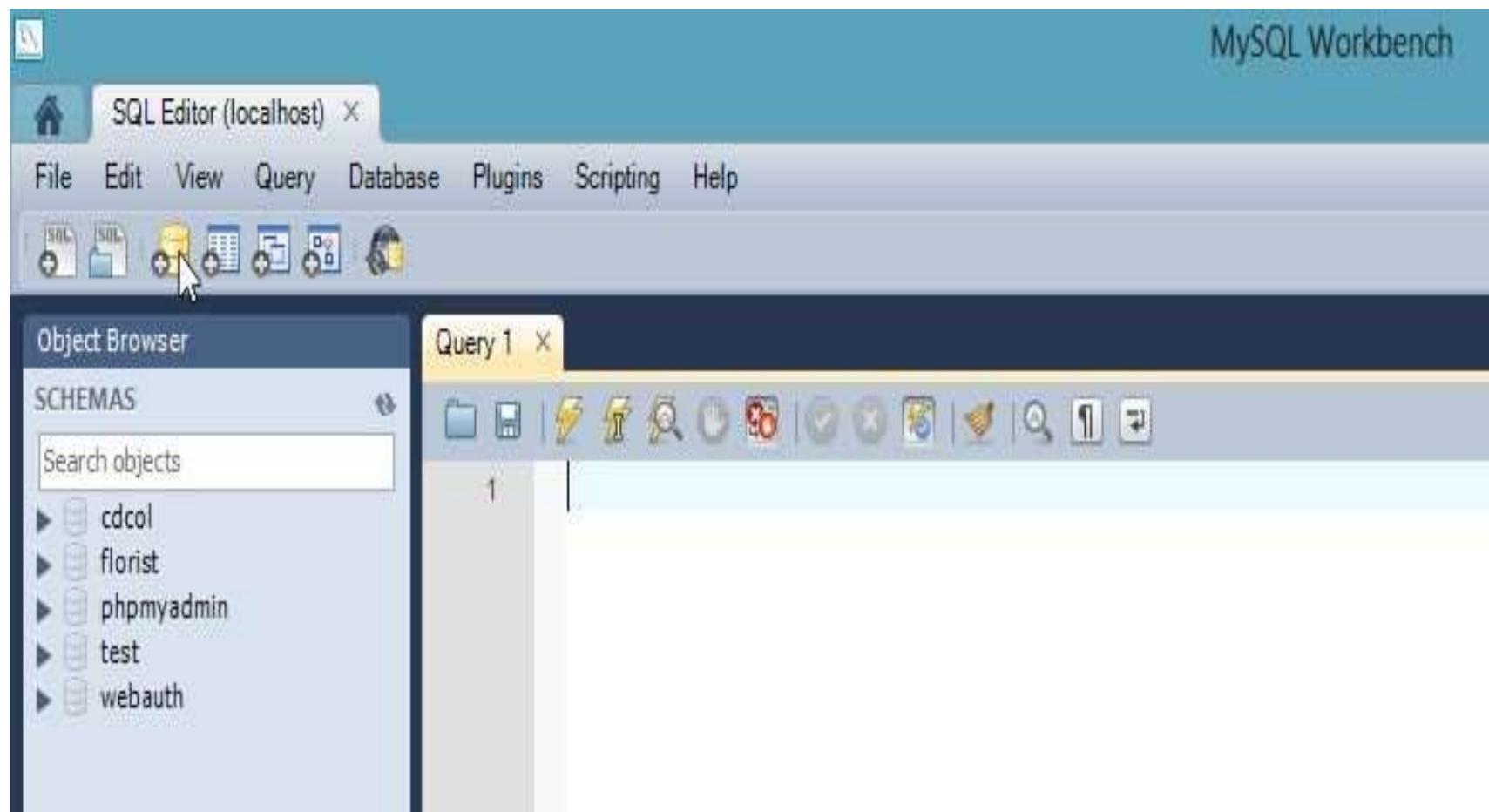
## localhost



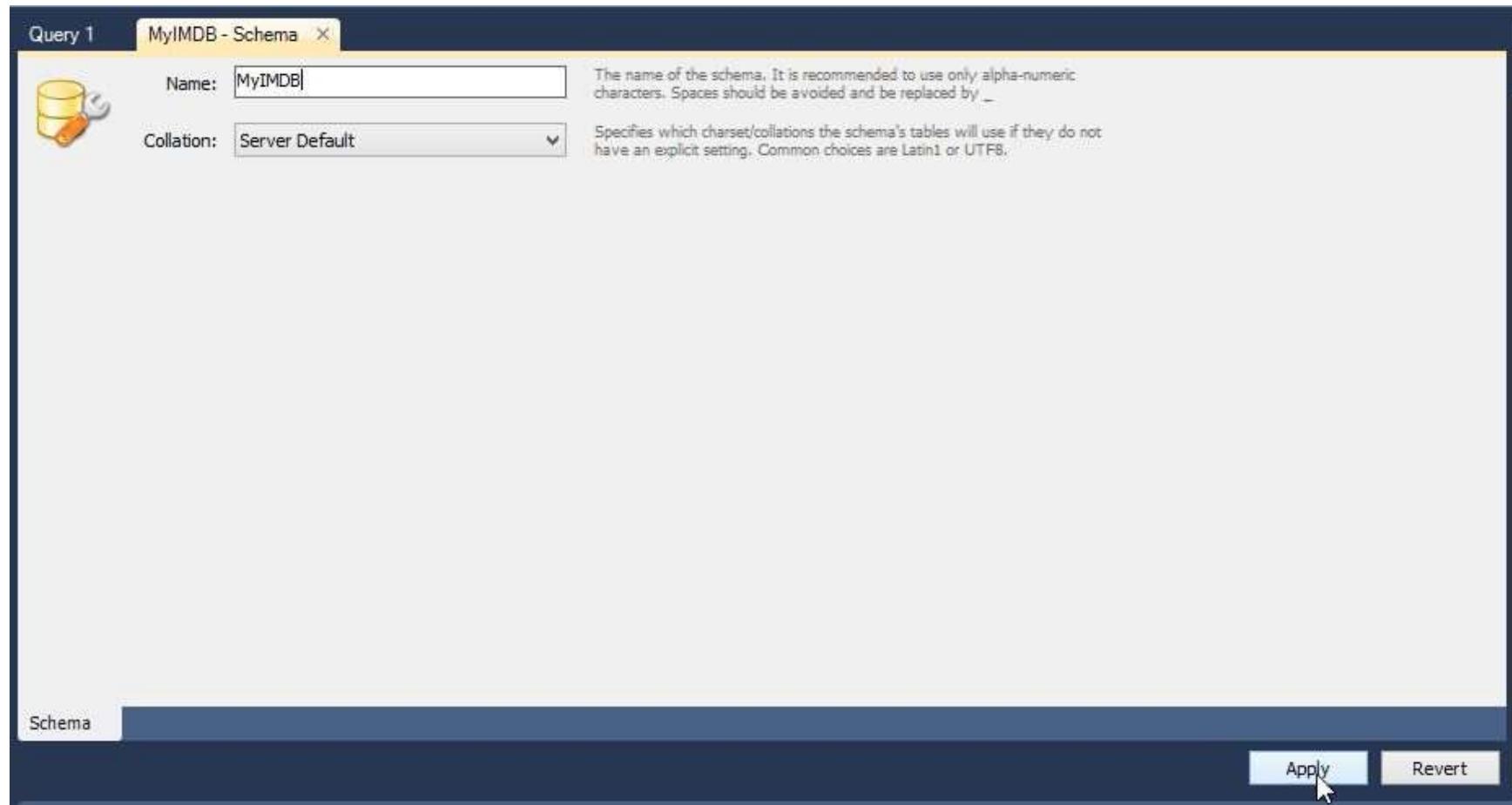
# Open Connection to Start Quering



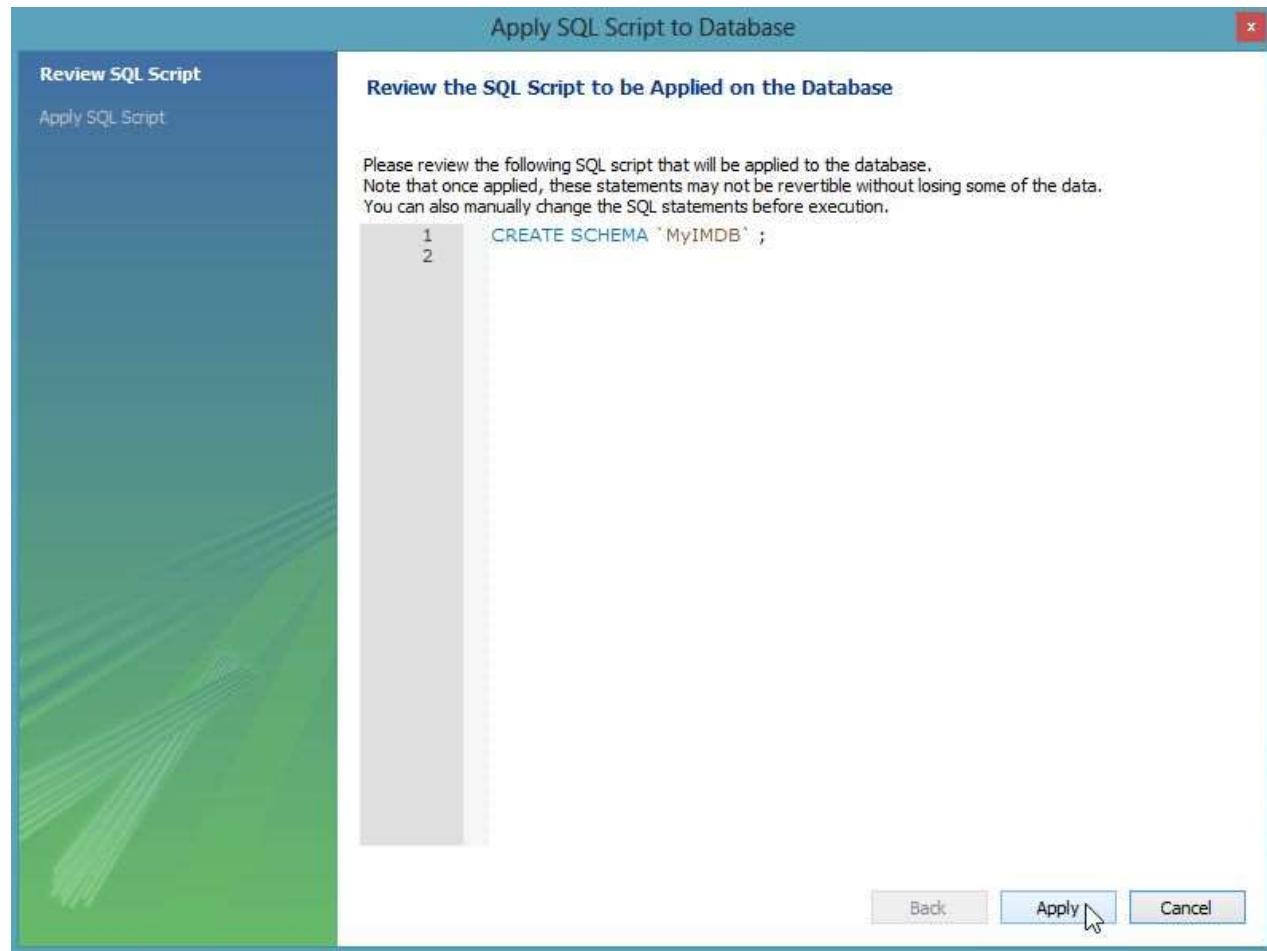
# Create New Database Schema



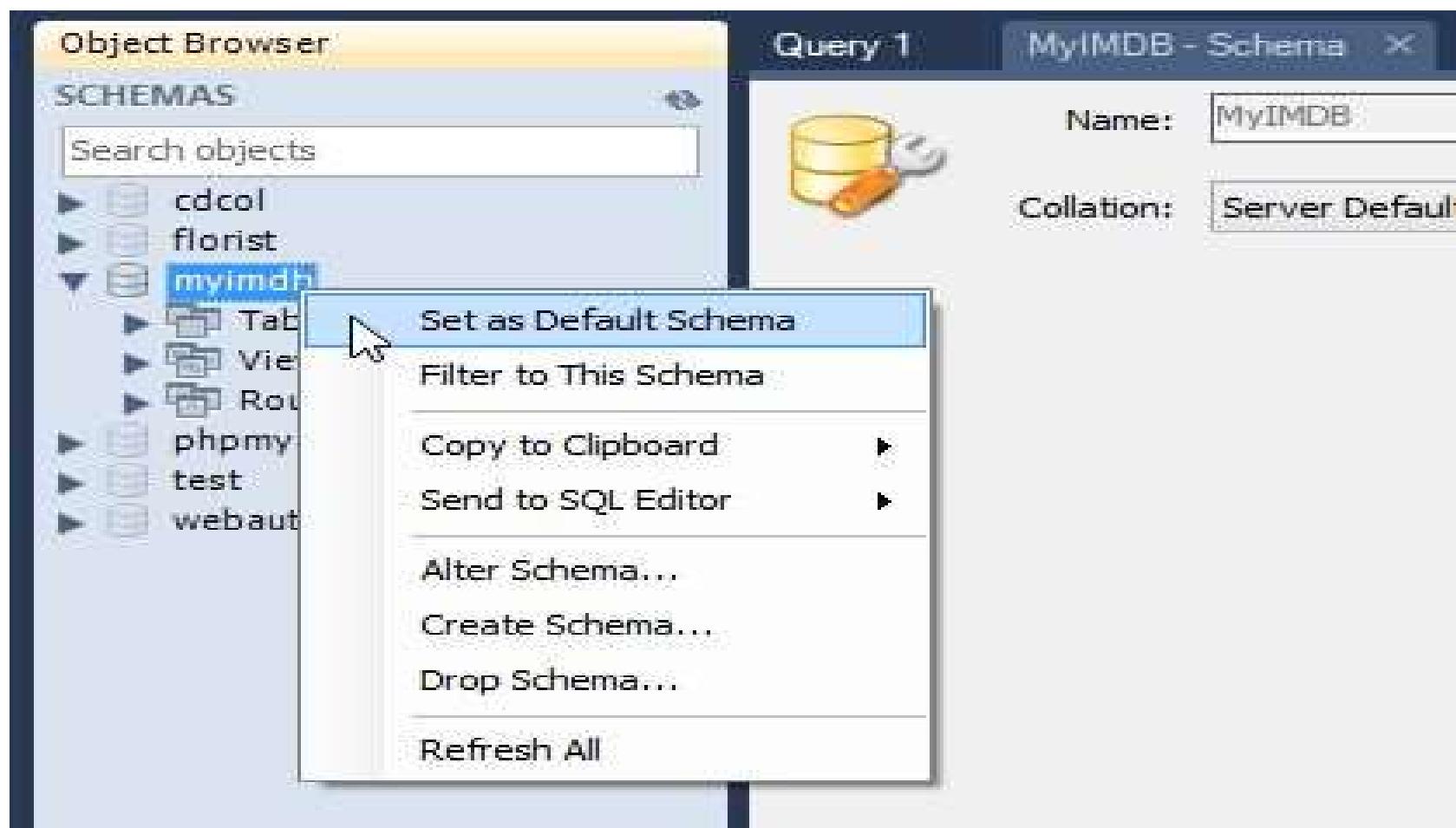
# Apply



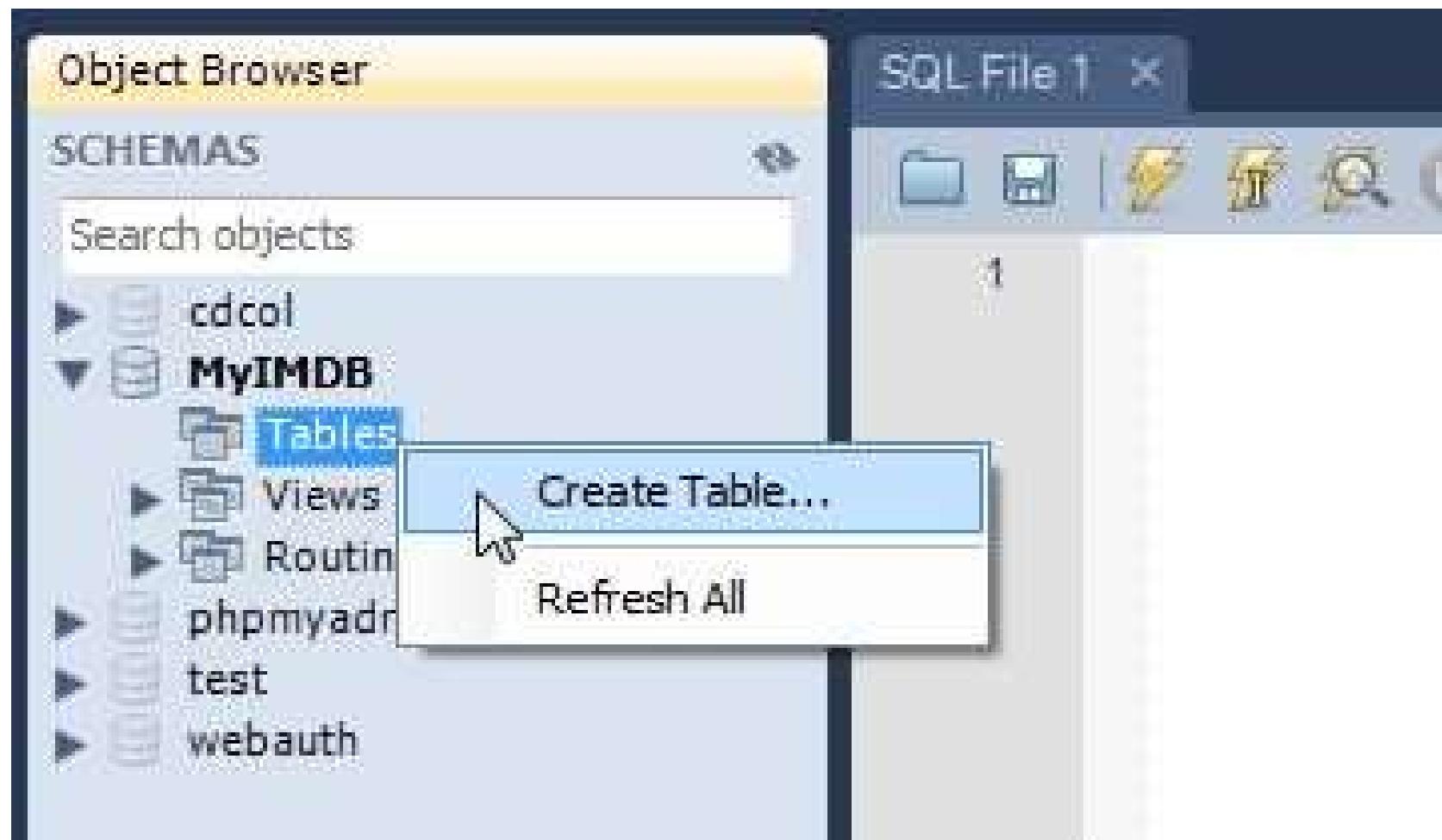
# SQL Script



# Set as Default Schema



# Create New Table



# Table *person*

SQL File 1    person - Table X

 Table Name:  Schema: MyIMDB

Collation: Schema Default Engine: InnoDB

Comments:

| Column Name | Datatype | PK                       | NN                       | UQ                       | BIN                      | UN                       | ZF                       | AI                       | Default |
|-------------|----------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|---------|
|             |          | <input type="checkbox"/> |         |

Column Name:   
Collation:   
Comments:

Data Type:   
Default:   
 Primary Key     Not Null     Unique  
 Binary     Unsigned     Zero Fill  
 Auto Increment

Columns    Indexes    Foreign Keys    Triggers    Partitioning    Options

Apply Revert

# (database) Engine - InnoDB



# Idperson – INT - Primary Key, Auto Increment

| Column Name | Datatype | PK                                  | NN                                  | UQ                       | BIN                      | UN                       | ZF                       | AI                                  | Default |
|-------------|----------|-------------------------------------|-------------------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------------------------------------|---------|
| idperson    | INT      | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |         |

Column Name: idperson Data Type: INT  
Collation: Table Default Default:  
Comments:

Primary Key  Not Null  Unique  
 Binary  Unsigned  Zero Fill  
 Auto Increment

Columns Indexes Foreign Keys Triggers Partitioning Options Apply Revert

# Name – VarChar(45) – Not NULL, Unique (Alt Key)

| Column Name | Datatype    | PK                                  | NN                                  | UQ                                  | BIN                      | UN                       | ZF                       | AI                                  | Default |
|-------------|-------------|-------------------------------------|-------------------------------------|-------------------------------------|--------------------------|--------------------------|--------------------------|-------------------------------------|---------|
| idperson    | INT         | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> |         |
| name        | VARCHAR(45) | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/>            |         |
|             |             | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/>            |         |

Column Name: name Data Type: VARCHAR(45)  
Collation: Table Default Default:  
Comments:

Primary Key  Not Null  Unique  
 Binary  Unsigned  Zero Fill  
 Auto Increment

Columns Indexes Foreign Keys Triggers Partitioning Options Apply Revert

# Date of Birth, Height, Bio

| Column Name | Datatype    | PK                       | NN                                  | UQ                                  | BIN                      | UN                       | ZF                       | AI                       | Default                  |
|-------------|-------------|--------------------------|-------------------------------------|-------------------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| name        | VARCHAR(45) | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |                          |
| dob         | DATE        | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |                          |
| height      | FLOAT       | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |                          |
| bio         | TEXT        | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

Column Name: bio Data Type: TEXT

Collation: Table Default

Comments:

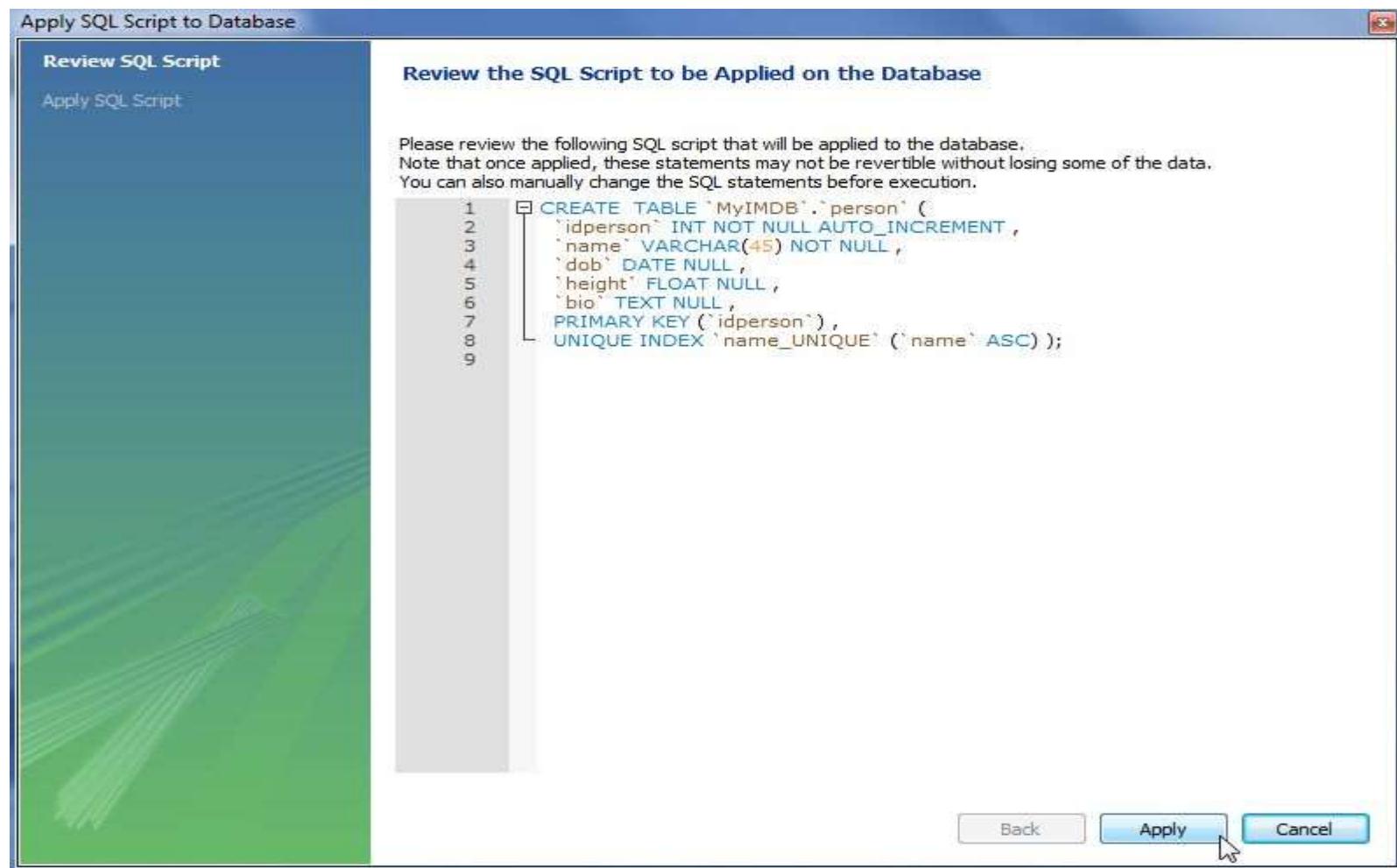
Default:

Primary Key    Not Null    Unique  
 Binary    Unsigned    Zero Fill  
 Auto Increment

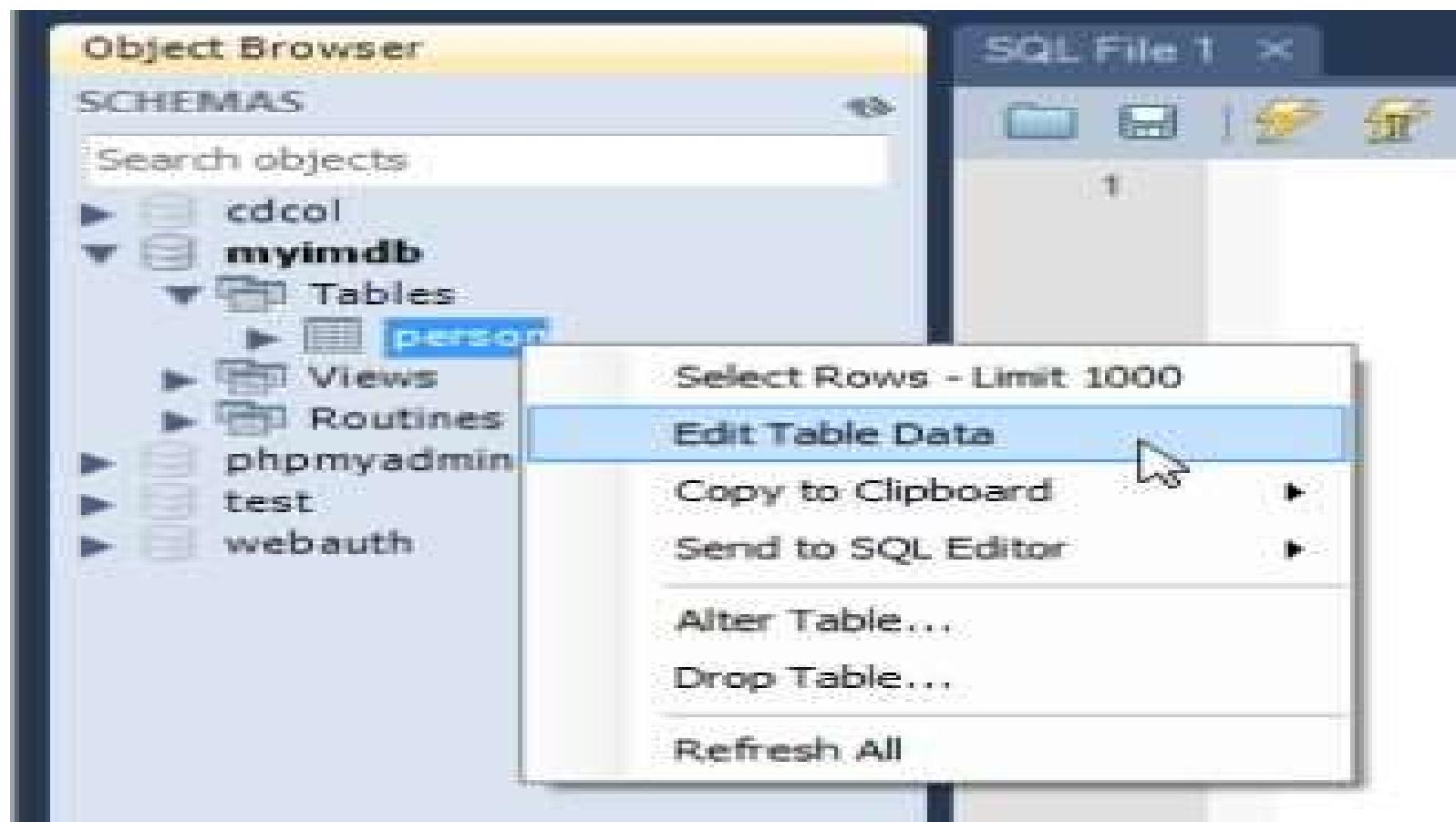
Columns   Indexes   Foreign Keys   Triggers   Partitioning   Options

Apply   Revert

# Apply SQL Script



# Edit Table Data



# Edit Table Data

The screenshot shows the MySQL Workbench interface with the 'person' table selected. The table has columns: idperson, name, dob, height, and bio. There are three rows of data:

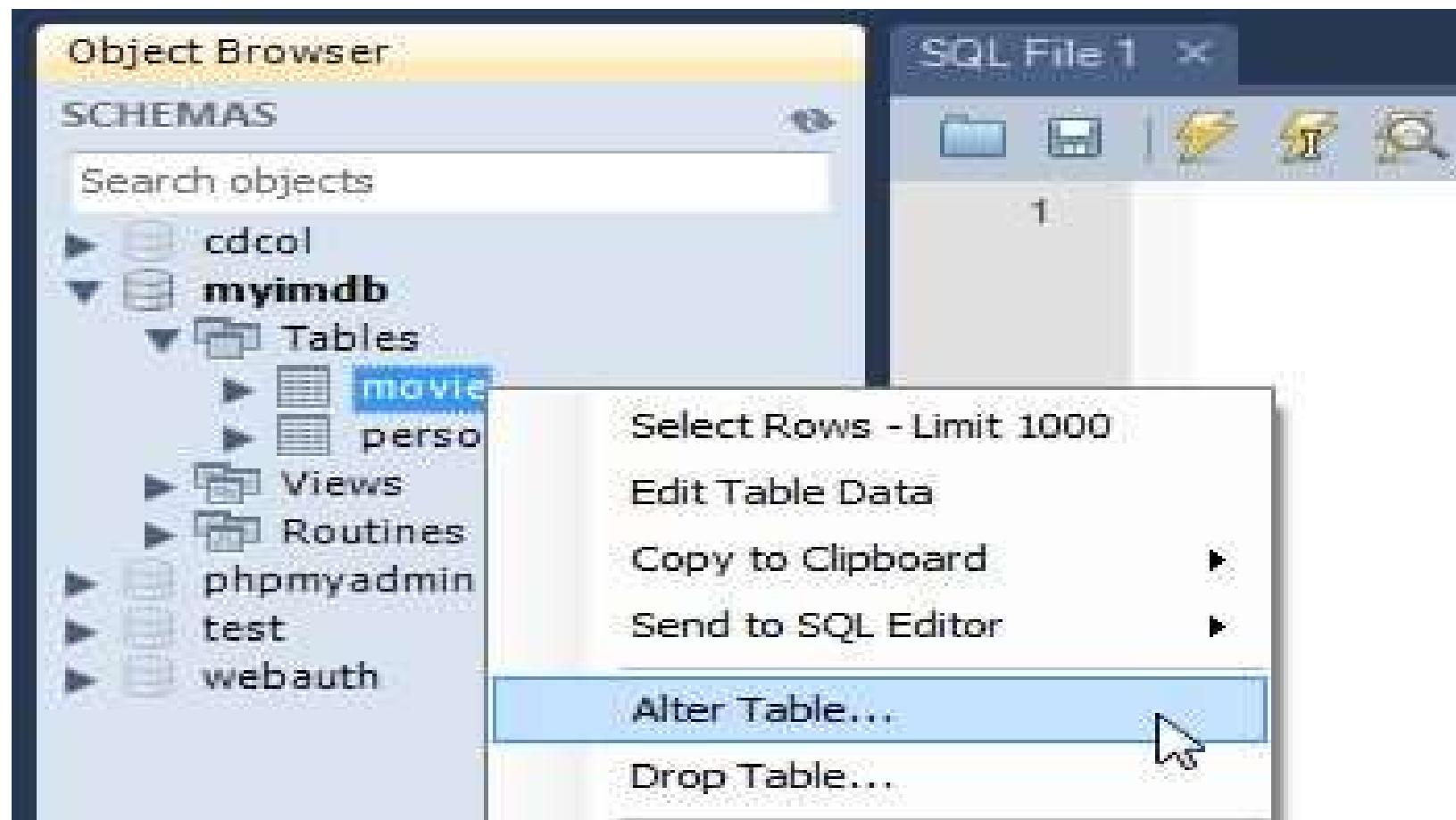
|   | idperson | name          | dob        | height | bio                                                                                                                                                                                                                     |
|---|----------|---------------|------------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | NULL     | Harrison Ford | 1942-07-13 | 1.85   | Harrison Ford was born on July 13, 1942 in Chicago, Illinois. His father had Irish and German ancestry, and his maternal grandparents were from Scotland.                                                               |
| 2 | NULL     | George Lucas  | 1944-05-14 | NULL   | George Lucas is a film director, producer, screenwriter, and entrepreneur. He is chairman of the board of The George Lucas Educational Foundation. In 1992, George Lucas was honored with the Irving G. Thalberg Award. |
| * | NULL     | NULL          | NULL       | NULL   | NULL                                                                                                                                                                                                                    |

The 'bio' column for Harrison Ford and George Lucas contains long descriptive text. The bottom right corner of the window shows 'Apply' and 'Cancel' buttons, with a cursor pointing at the 'Apply' button.

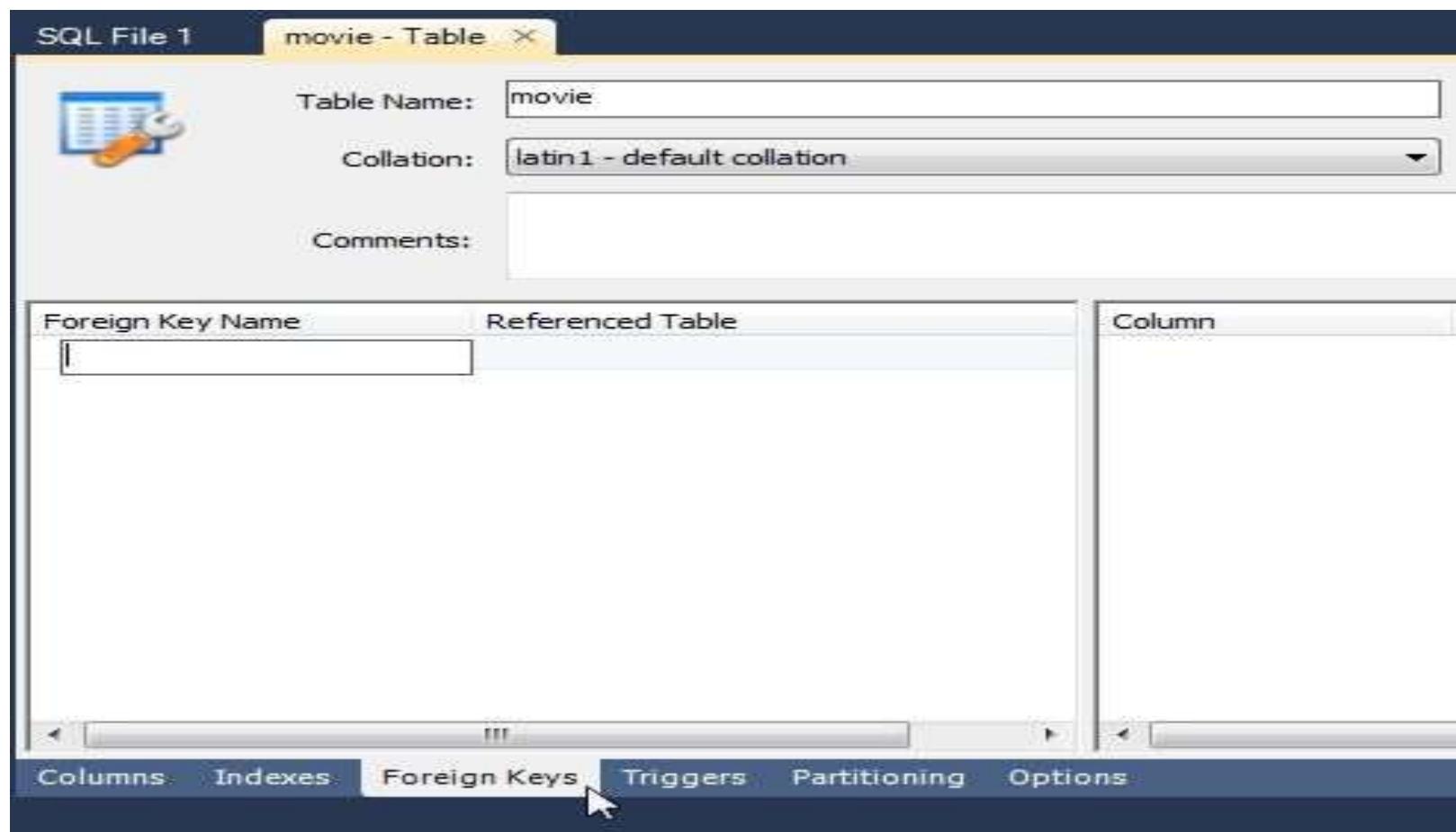
# TABLE *movie*

- CREATE TABLE `myimdb`.`movie` (
- `idmovie` INT NOT NULL AUTO\_INCREMENT ,
- `title` VARCHAR(150) NOT NULL ,
- `director` INT NOT NULL ,
- PRIMARY KEY (`idmovie`),
- UNIQUE INDEX `title\_UNIQUE` (`title` ASC);

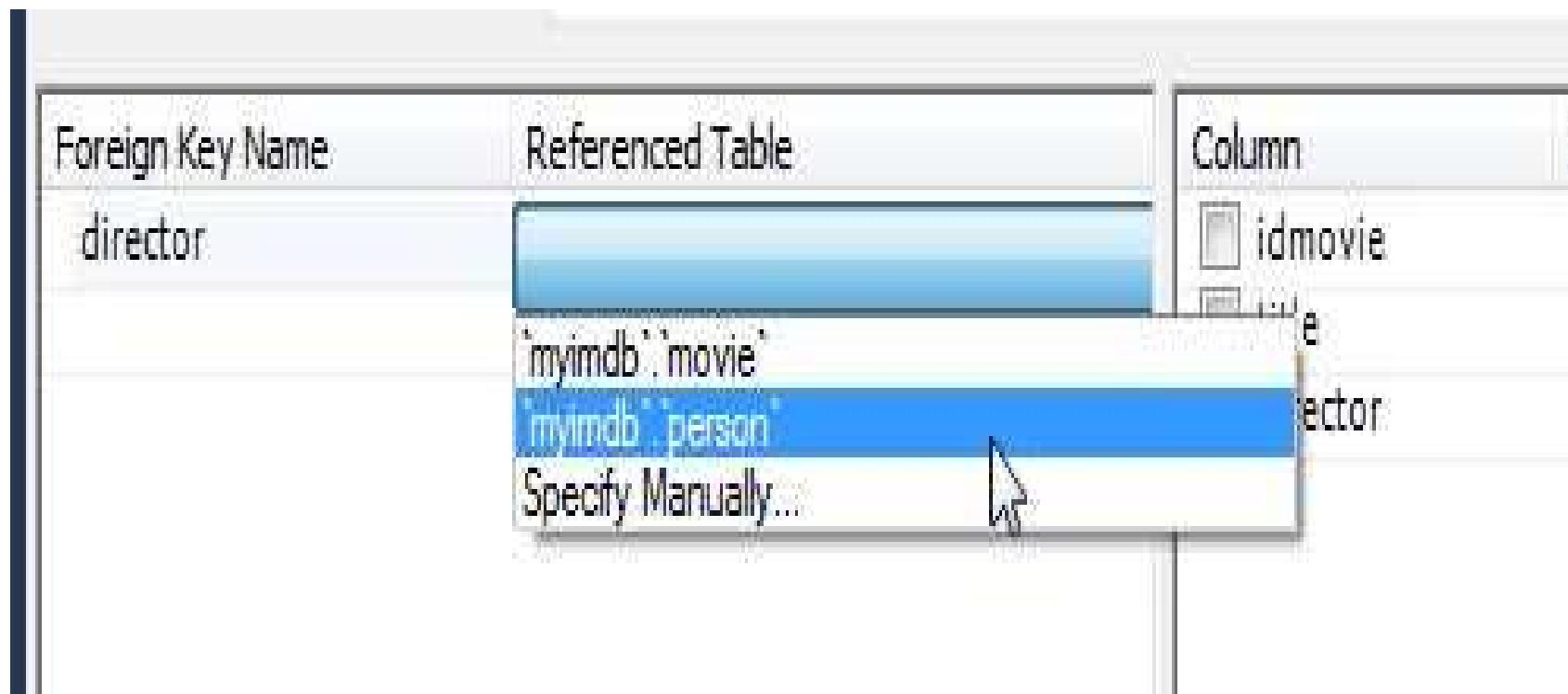
# Alter Table



# Foreign Key



# Referenced Table: Person



# Referenced Column: director

| Column                                       | Referenced Column                                                                 |
|----------------------------------------------|-----------------------------------------------------------------------------------|
| <input type="checkbox"/> idmovie             |                                                                                   |
| <input type="checkbox"/> title               |                                                                                   |
| <input checked="" type="checkbox"/> director | <input type="checkbox"/> idperson<br><input checked="" type="checkbox"/> idperson |

Foreign Key Options

On Update: NO ACTION ▾

On Delete: NO ACTION ▾

Skip in SQL generation

Foreign Key Comment



- Primary Table: person
- Primary Key: idperson
- Must match
- Foreign Table: movie
- Foreign Key: director
- What to do on Update, Delete

## Foreign Key Options

On Update: RESTRICT ▾

On Delete: RESTRICT ▾

- ALTER TABLE `myimdb`.`movie`
- ADD CONSTRAINT `director` FOREIGN KEY (`director`) REFERENCES `myimdb`.`person`(`idperson`)
- ON DELETE RESTRICT
- ON UPDATE RESTRICT,
- ADD INDEX `director\_idx`(`director` ASC) ;

# TABLE *roles*

(role could be reserved keyword)

- CREATE TABLE `myimdb`.`roles` (
- `idroles` INT NOT NULL AUTO\_INCREMENT ,
- `rolename` VARCHAR(45) NOT NULL ,
- PRIMARY KEY (`idroles`),
- UNIQUE INDEX `rolename\_UNIQUE`(`rolename` ASC);

# TABLE *casting*

- CREATE TABLE `myimdb`.`casting` (  
• `movie` INT NOT NULL , `person` INT NOT NULL , `RolePlay`  
INT NOT NULL , PRIMARY KEY (`movie`, `person`, `RolePlay`),  
• INDEX `castmovie\_idx` (`movie` ASC) , INDEX `castpers\_idx` (`person` ASC)  
, INDEX `castrole\_idx` (`RolePlay` ASC),  
• CONSTRAINT `castmovie` FOREIGN KEY (`movie`) REFERENCES  
`myimdb`.`movie`(`idmovie`) ON DELETE RESTRICT ON UPDATE  
RESTRICT,  
• CONSTRAINT `castpers` FOREIGN KEY (`person`) REFERENCES  
`myimdb`.`person`(`idperson`) ON DELETE RESTRICT ON UPDATE  
RESTRICT,  
• CONSTRAINT `castrole` FOREIGN KEY (`RolePlay`) REFERENCES  
`myimdb`.`roles`(`idroles`) ON DELETE RESTRICT ON UPDATE  
RESTRICT);

# TABLE *characterCast*

- CREATE TABLE `myimdb`.`characterCast` (
- `idmovie` INT NOT NULL ,
- `idperson` INT NOT NULL ,
- `characterName` VARCHAR(45) NULL ,
- `orderNo` INT NULL DEFAULT 1 ,
- PRIMARY KEY (`idmovie`, `idperson`));

# TABLE *characterCast*

- ALTER TABLE `myimdb`.`charactercast`
- ADD CONSTRAINT `movieCast` FOREIGN KEY (`idmovie`) REFERENCES `myimdb`.`movie`(`idmovie`) ON DELETE RESTRICT ON UPDATE RESTRICT,
- ADD CONSTRAINT `persCast` FOREIGN KEY (`idperson`) REFERENCES `myimdb`.`person`(`idperson`) ON DELETE RESTRICT ON UPDATE RESTRICT, ADD INDEX `movieCast\_idx`(`idmovie` ASC), ADD INDEX `persCast\_idx`(`idperson` ASC);

# Data Modeling

# Create Entity Relationship Model from Existing Database

The screenshot shows the MySQL Workbench interface. On the left, under 'Data Modeling', there is a section titled 'Open Existing EER Model' with a link to 'sakila\_full'. Below it are three buttons: 'Create New EER Model', 'Create EER Model From Existing Database' (which is highlighted with a mouse cursor), and 'Create EER Model From SQL Script'. On the right, under 'Server Administration', there is a section titled 'Server Administration' with a link to 'localhost'. Below it are four buttons: 'New Server Instance', 'Manage Import / Export', 'Manage Security', and 'Manage Server Instances'.

**Data Modeling**  
Create and manage models, forward & reverse engineer, compare and synchronize schemas, report.

**Server Administration**  
Configure your database server, setup user accounts, browse status variables and server logs.

**Open Existing EER Model**  
Or select a model to open or click here to browse.

**sakila\_full**  
Last modified Thu Feb 14 21:54:08 2013

**Create New EER Model**  
Create a new EER Model from scratch.

**Create EER Model From Existing Database**  
Creates by connecting and reverse engineering

**Create EER Model From SQL Script**  
Insert an existing SQL file.

**Server Administration**  
Or click to manage a database server instance.

**localhost**  
Local Type: Windows

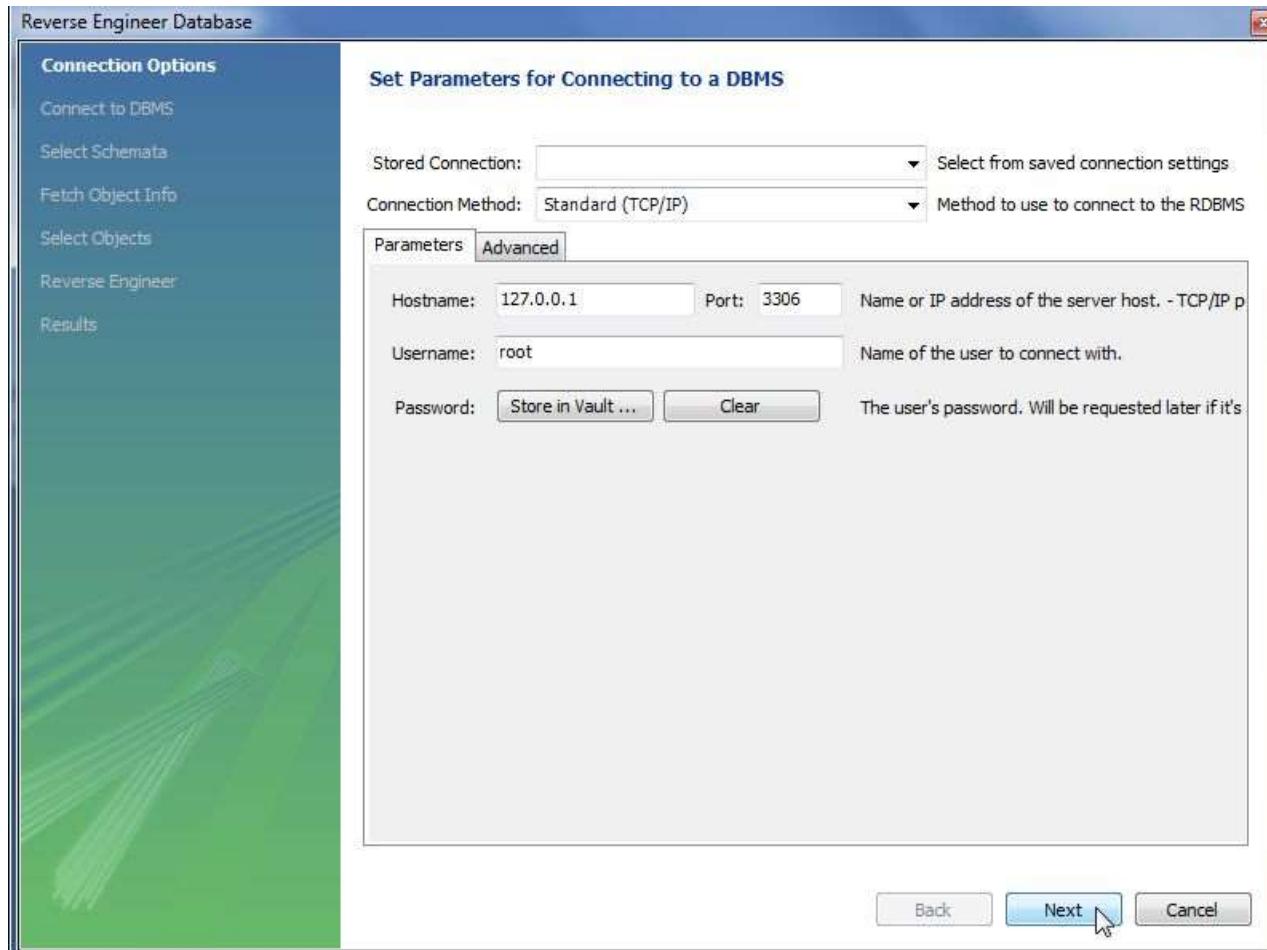
**New Server Instance**  
Register a new server instance to manage.

**Manage Import / Export**  
Create a dump file or restore data from a file.

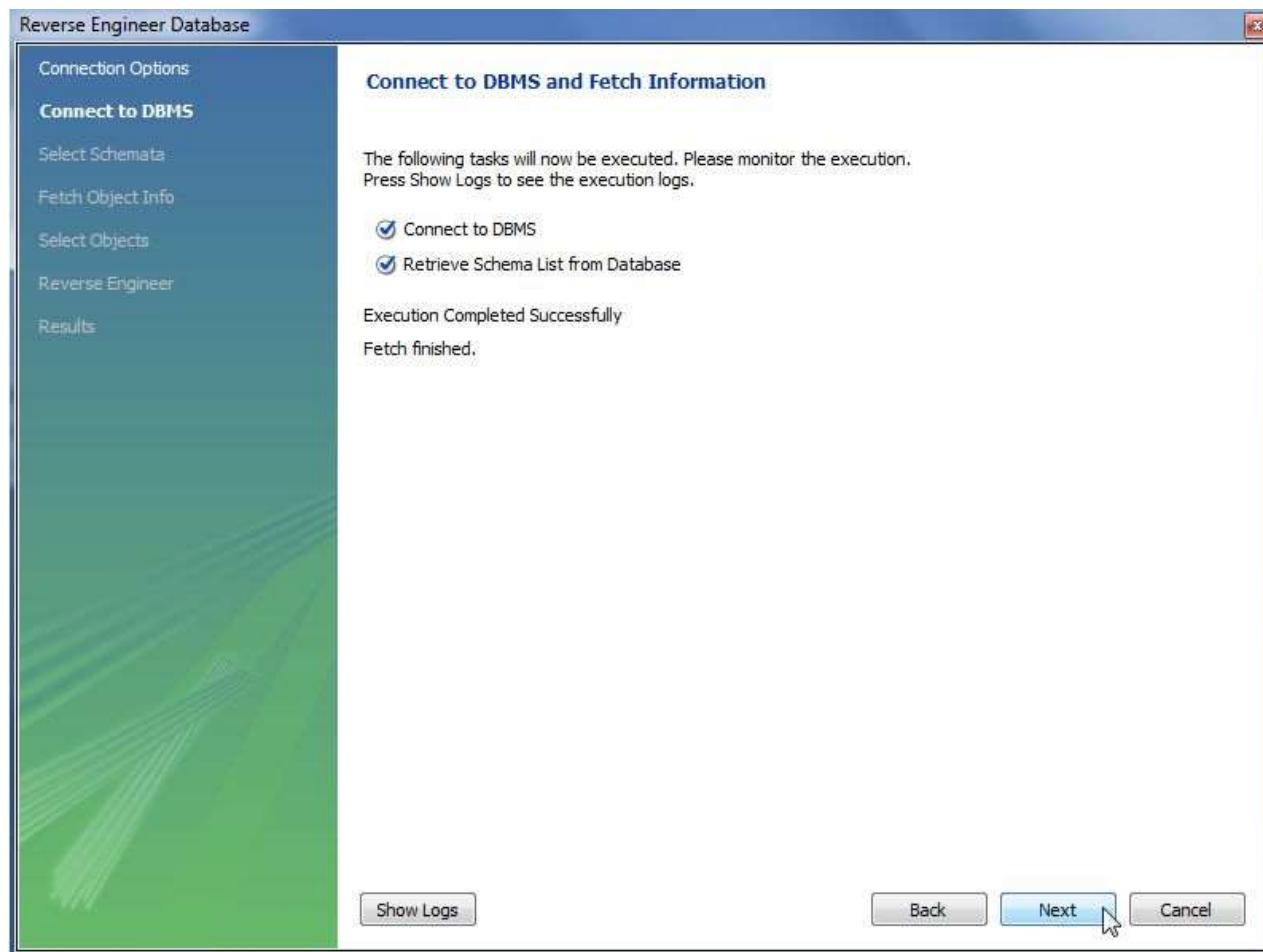
**Manage Security**  
Manage user accounts and assign privileges.

**Manage Server Instances**  
Add, delete and update server instance settings.

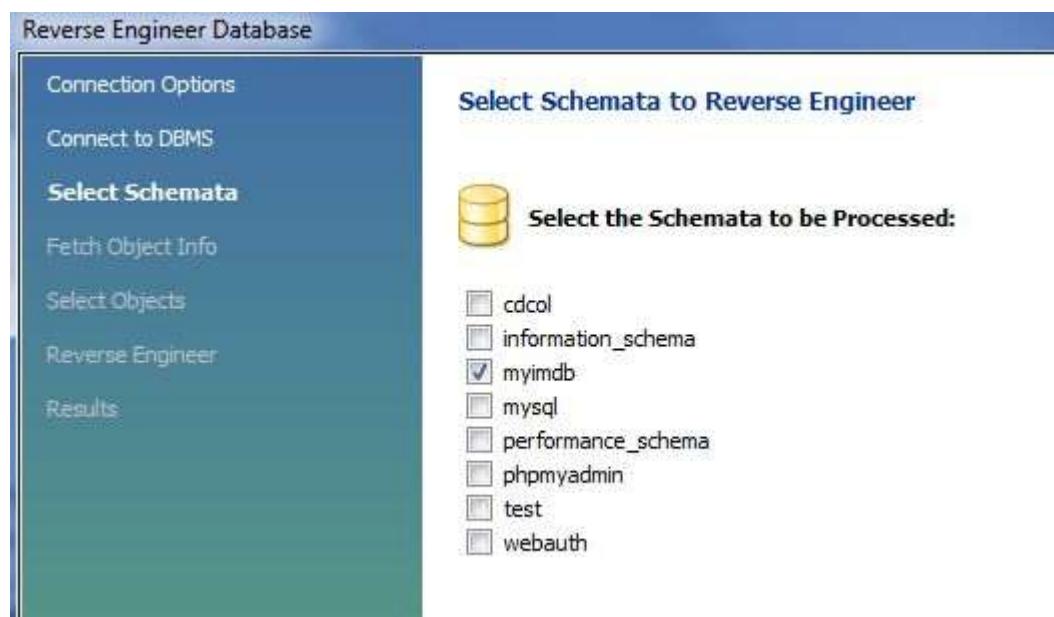
# Reverse Engineer Stored Connection: localhost



# ... Next ...

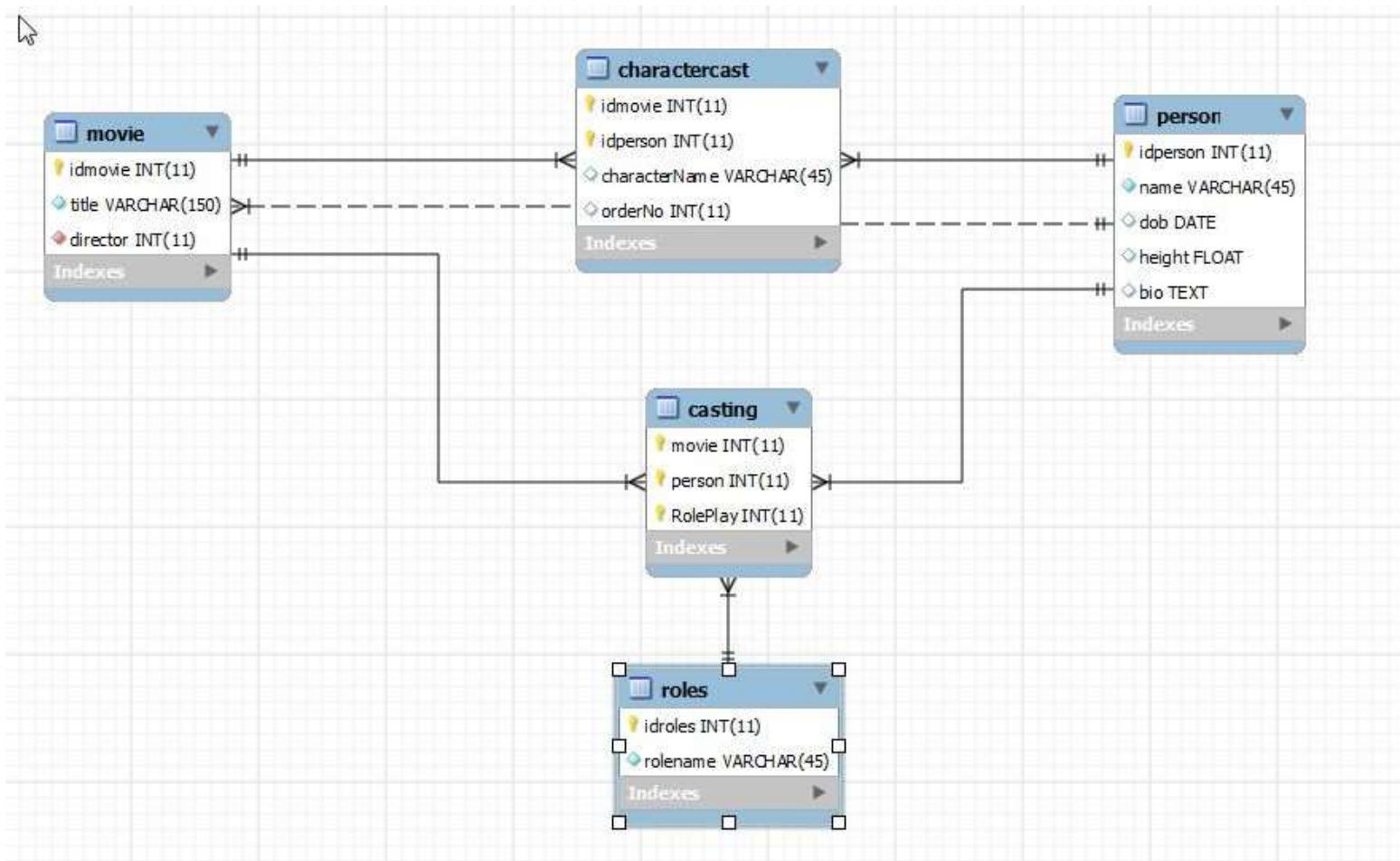


# Select Database Schema to Reverse Engineer



# DataBase Diagram

## Entity Relationship Diagram



# ER database diagram

- One to Many (weak) Relationship between Person and Movie
  - One Person direct many Movies
  - One Movie is directed by one single person

# ER database diagram

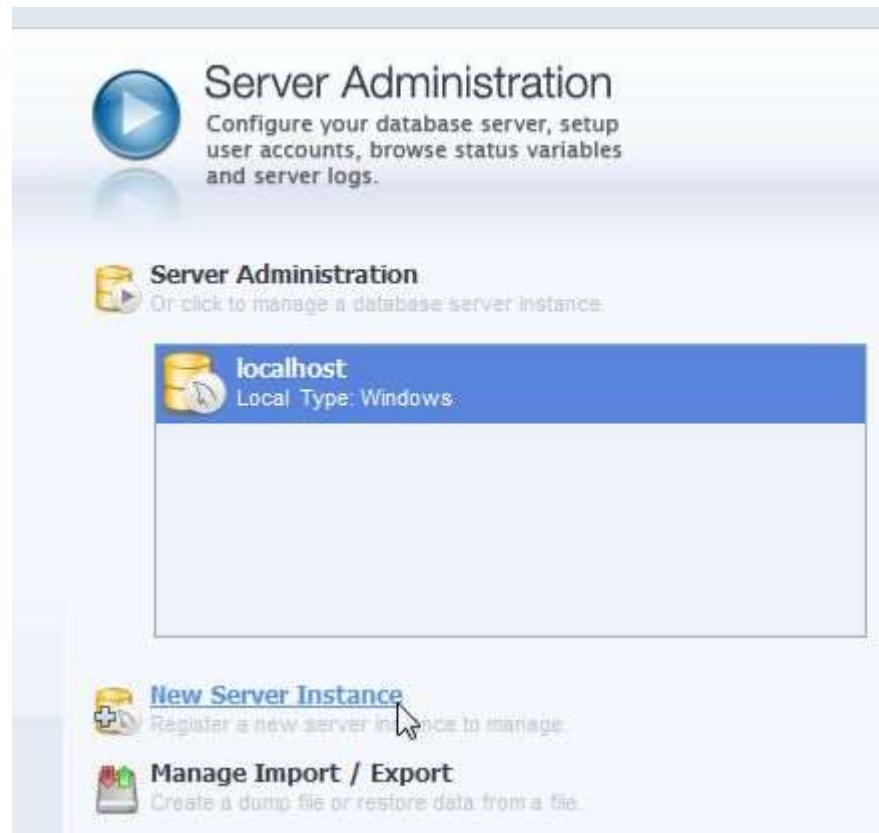
- Many to Many Relationship (Character Casting) between Movie and Person
  - Artistic roles
- Ternary Many to Many Relationship (Casting) between Movie and Person and Roles (Writer, ...)
  - Technical roles

# DataBase Administration

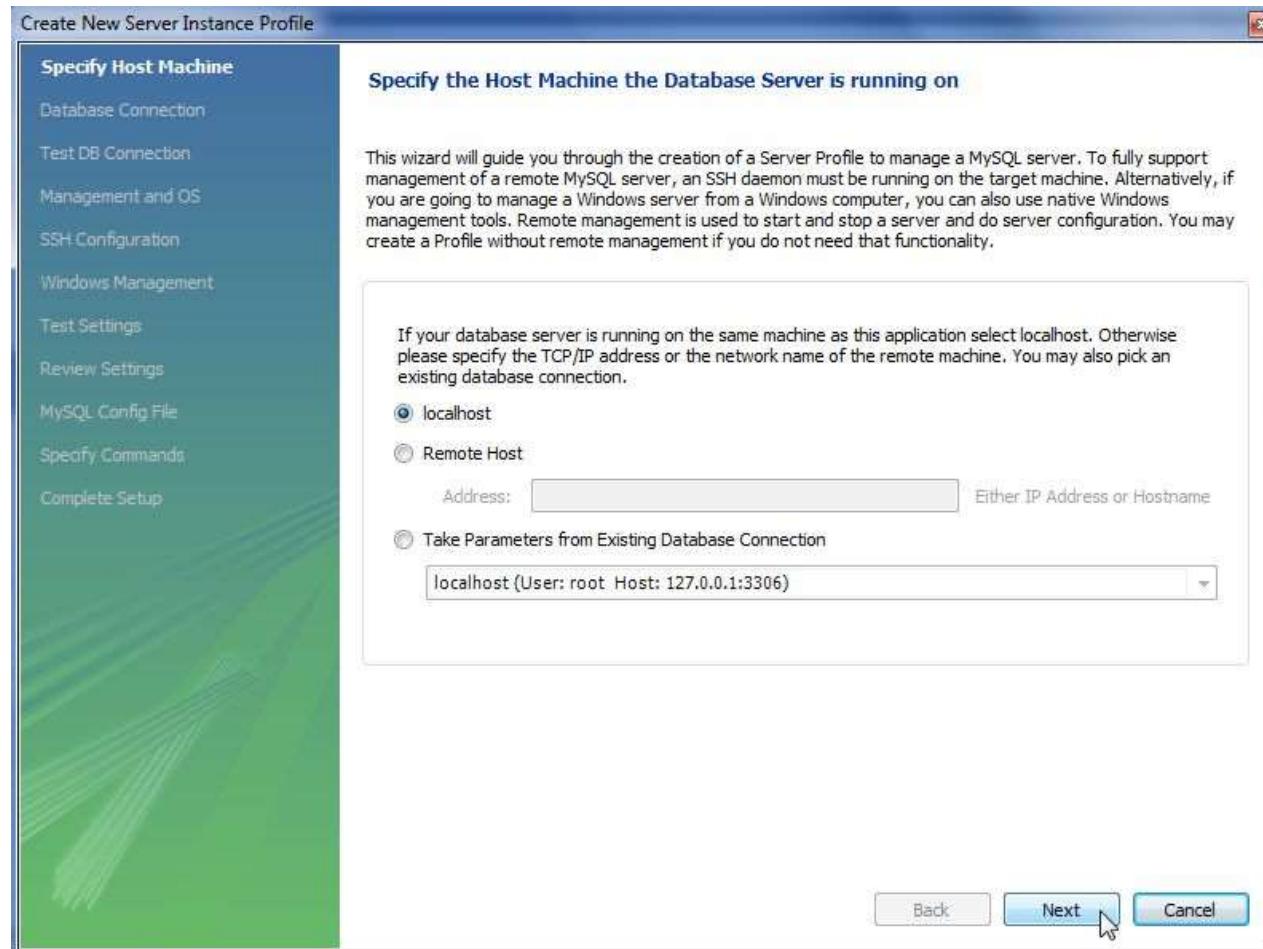
# Server Administration



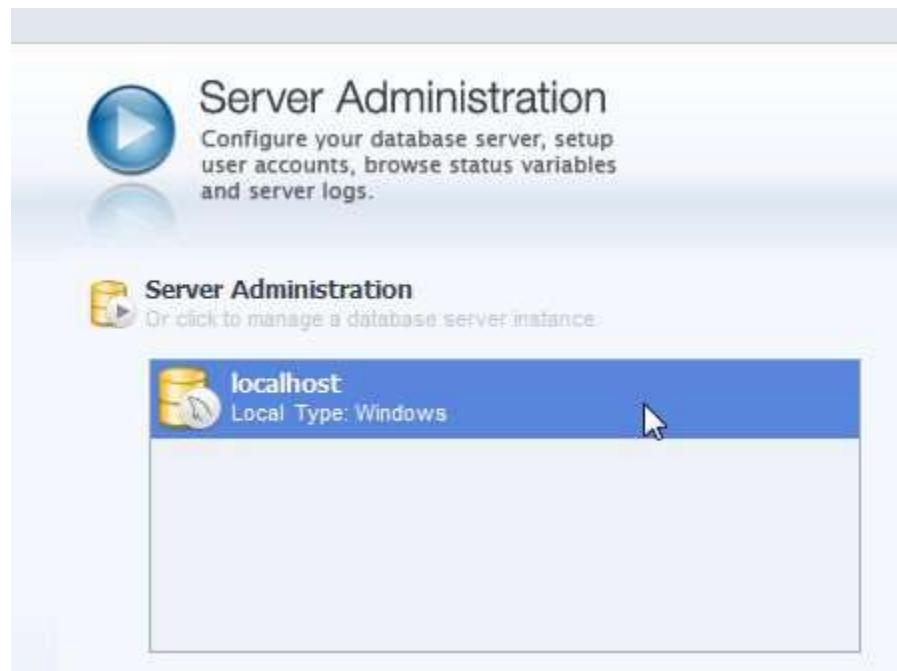
# Create New Server Instance



# localhost ... Next ...



# Administration of localhost server instance



MySQL Workbench

MySQL Model\* Admin (localhost)

File Edit View Database Plugins Scripting Help

Task and Object Browser

MANAGEMENT

- Server Status
- Startup / Shutdown
- Status and System Variables
- Server Logs

CONFIGURATION

- Options File

SECURITY

- Users and Privileges

DATA EXPORT / RESTORE

- Data Export
- Data Import/Restore

Server Status

INFO

|                        |                        |                       |                        |        |        |
|------------------------|------------------------|-----------------------|------------------------|--------|--------|
| Name: <b>localhost</b> | Host: <b>127.0.0.1</b> | Server: <b>5.5.27</b> | Status: <b>Running</b> | CPU: - | Mem: - |
|------------------------|------------------------|-----------------------|------------------------|--------|--------|

SYSTEM

SERVER HEALTH

|                     |            |                        |                   |
|---------------------|------------|------------------------|-------------------|
| Connection Usage: - | Traffic: - | Query Cache Hitrate: - | Key Efficiency: - |
|---------------------|------------|------------------------|-------------------|

CONNECTIONS

| ID | User | Host            | DB   | Command | Time | State | Info                  |
|----|------|-----------------|------|---------|------|-------|-----------------------|
| 31 | root | localhost:53644 | None | Query   | 0    | None  | SHOW FULL PROCESSLIST |
| 32 | root | localhost:53645 | None | Sleep   | 0    | None  |                       |

# Data Export

The screenshot shows the MySQL Workbench interface. On the left, the **Task and Object Browser** pane is open, displaying the following sections:

- MANAGEMENT**:
  - Server Status** (selected)
  - Startup / Shutdown
  - Status and System Variables
  - Server Logs
- CONFIGURATION**: Options File
- SECURITY**: Users and Privileges
- DATA EXPORT / RESTORE**:
  - Data Export** (mouse cursor is over this item)
  - Data Import/Restore

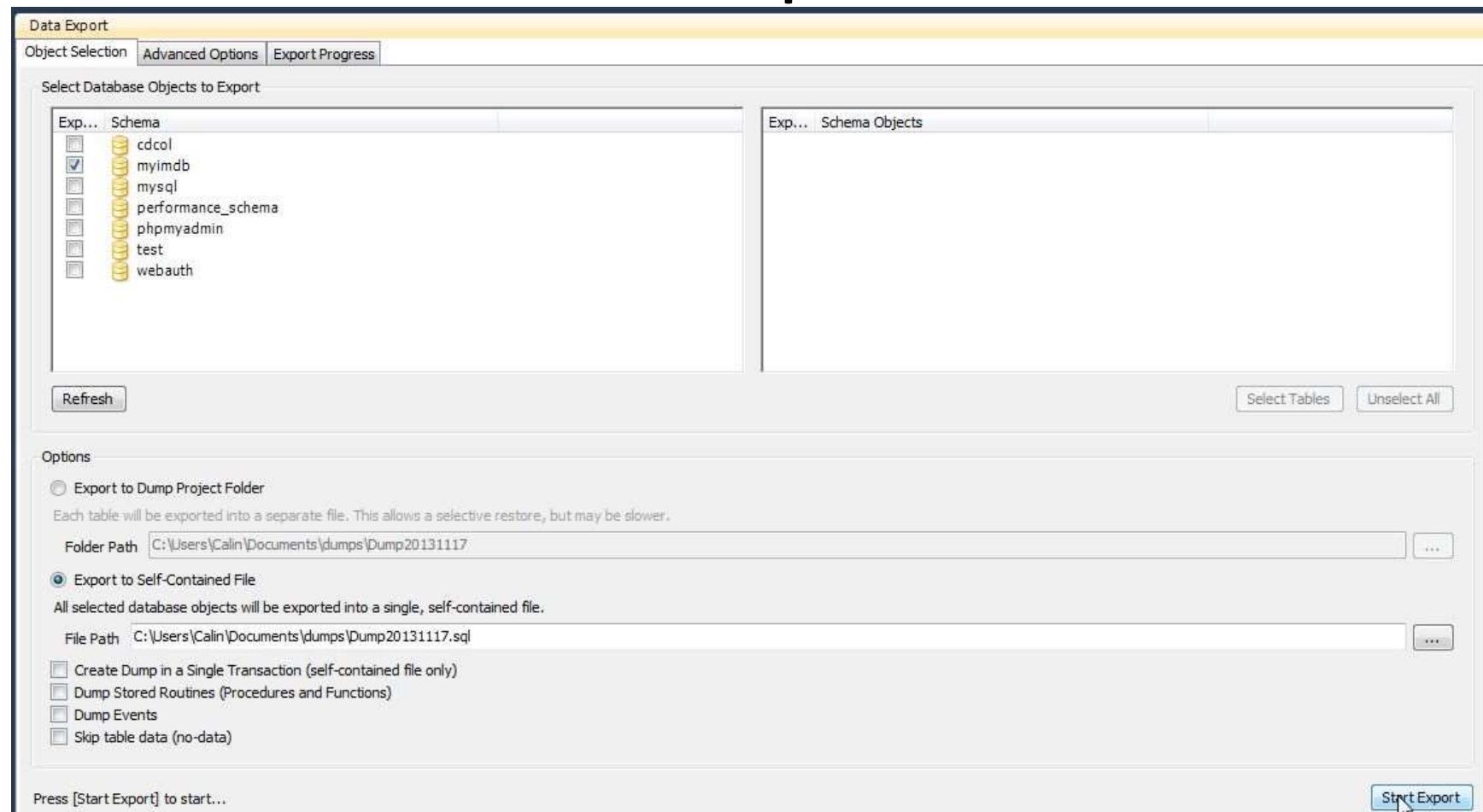
The right pane is titled **Server Status** and contains the following information:

| INFO        |                                                                                                     | SYSTEM          |           | SERVER HEALTH  |             |
|-------------|-----------------------------------------------------------------------------------------------------|-----------------|-----------|----------------|-------------|
|             | Name: <b>localhost</b><br>Host: <b>127.0.0.1</b><br>Server: <b>5.5.27</b><br>Status: <b>Running</b> | CPU: 0%         | Mem: 56%  |                | Traffic:    |
| CONNECTIONS |                                                                                                     |                 |           |                |             |
| <b>Id</b>   | <b>User</b>                                                                                         | <b>Host</b>     | <b>DB</b> | <b>Command</b> | <b>Time</b> |
| 31          | root                                                                                                | localhost:53644 | None      | Query          | 0 None      |
| 32          | root                                                                                                | localhost:53645 | None      | Sleep          | 0           |

# Select Database objects to Export

## Export to Self Contained File

### Start Export



# DataBases

## DataBase Design Normalization Process

# Objectives

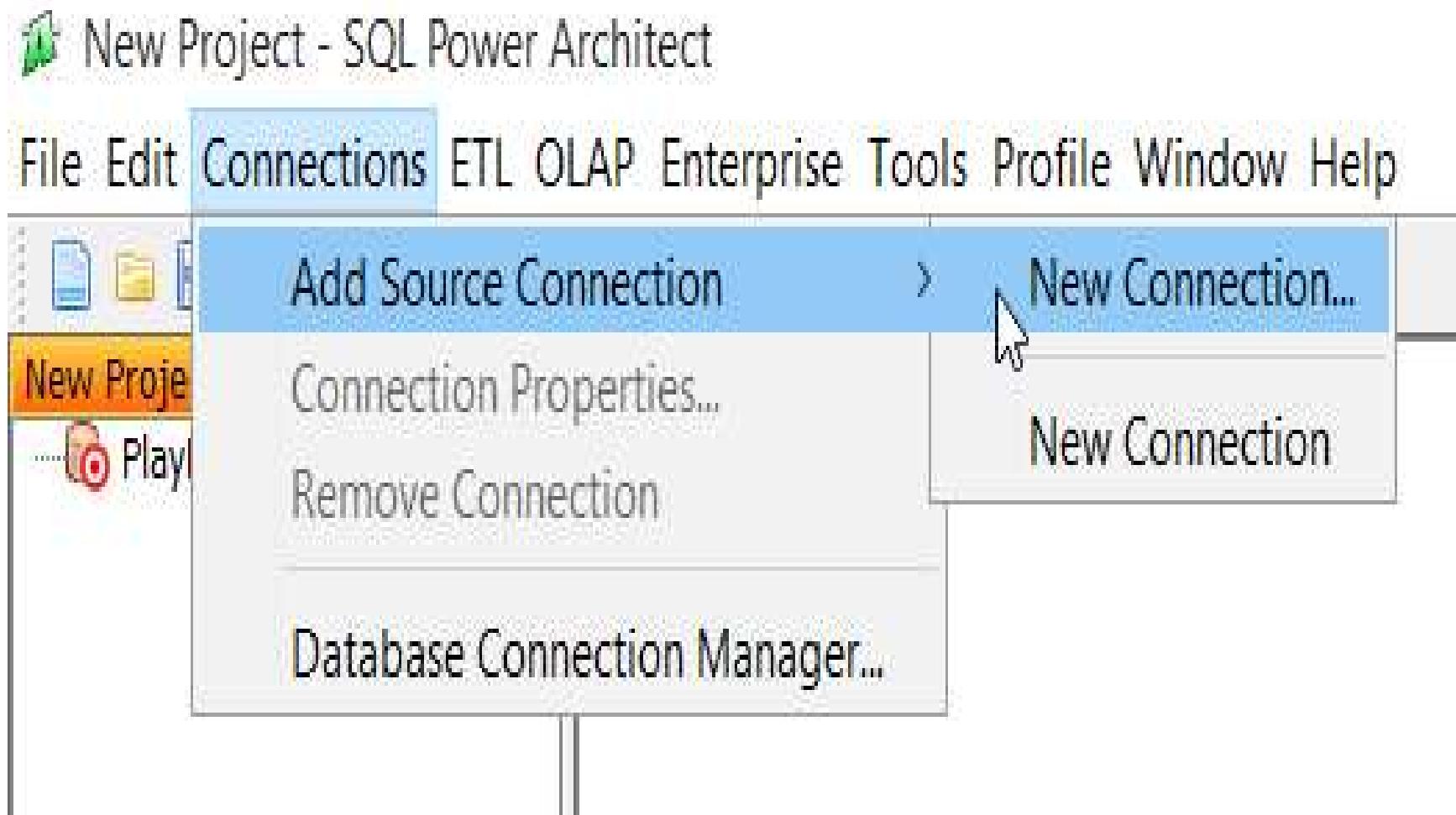
- Which database structures are considered Normal?
- Difference between snapshot and temporal databases
- Store Binary Large OBjects in databases

# **SQL POWER ARCHITECT DATA MODELING**

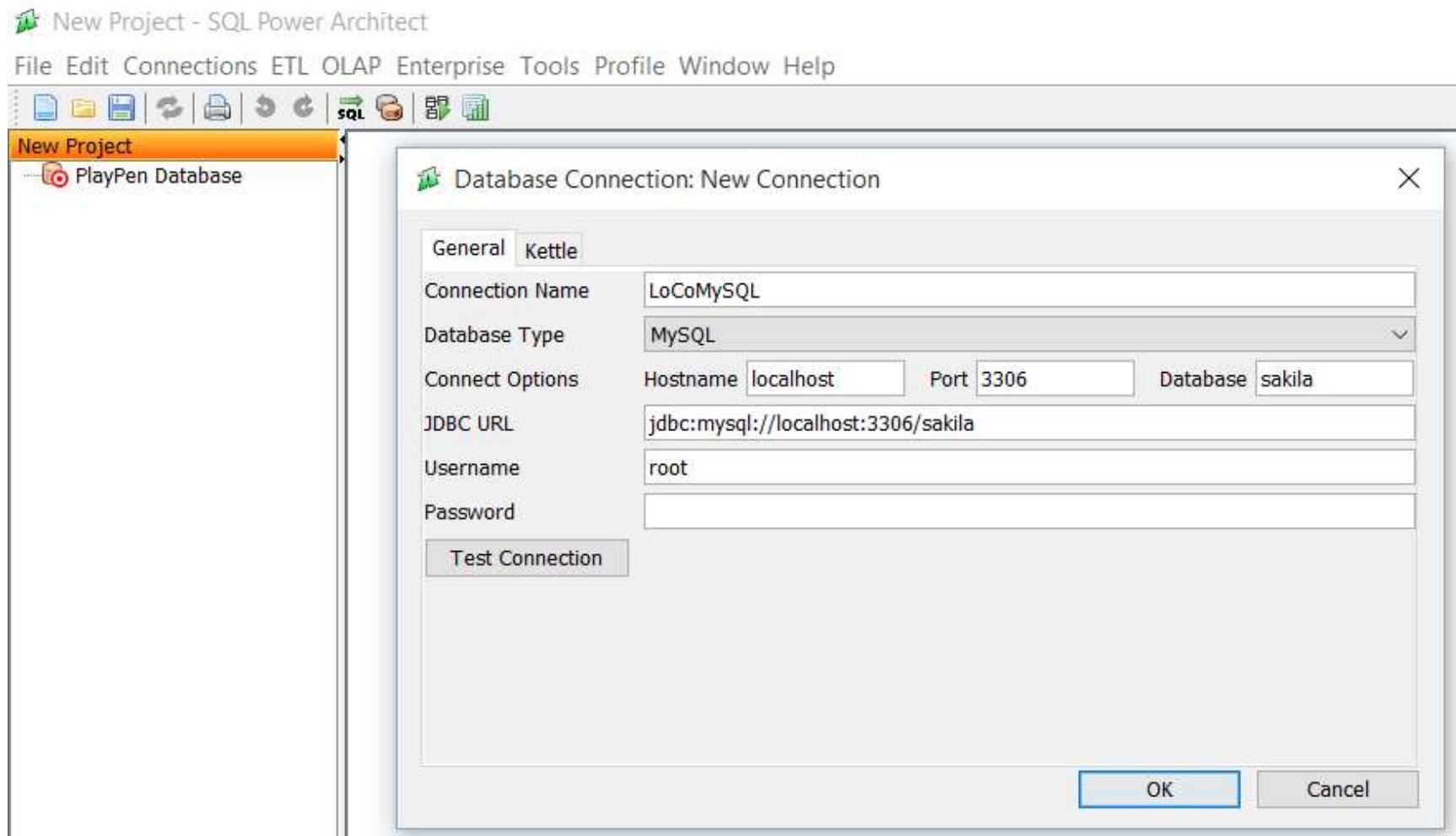
# SQL Power Architect – Data Modeling

- <http://www.sqlpower.ca/page/architect>
- free database tool to design data models; forward/reverse engineer to/from any database platform; works with any database using generic framework; runs on every platform: PC, Mac, Unix

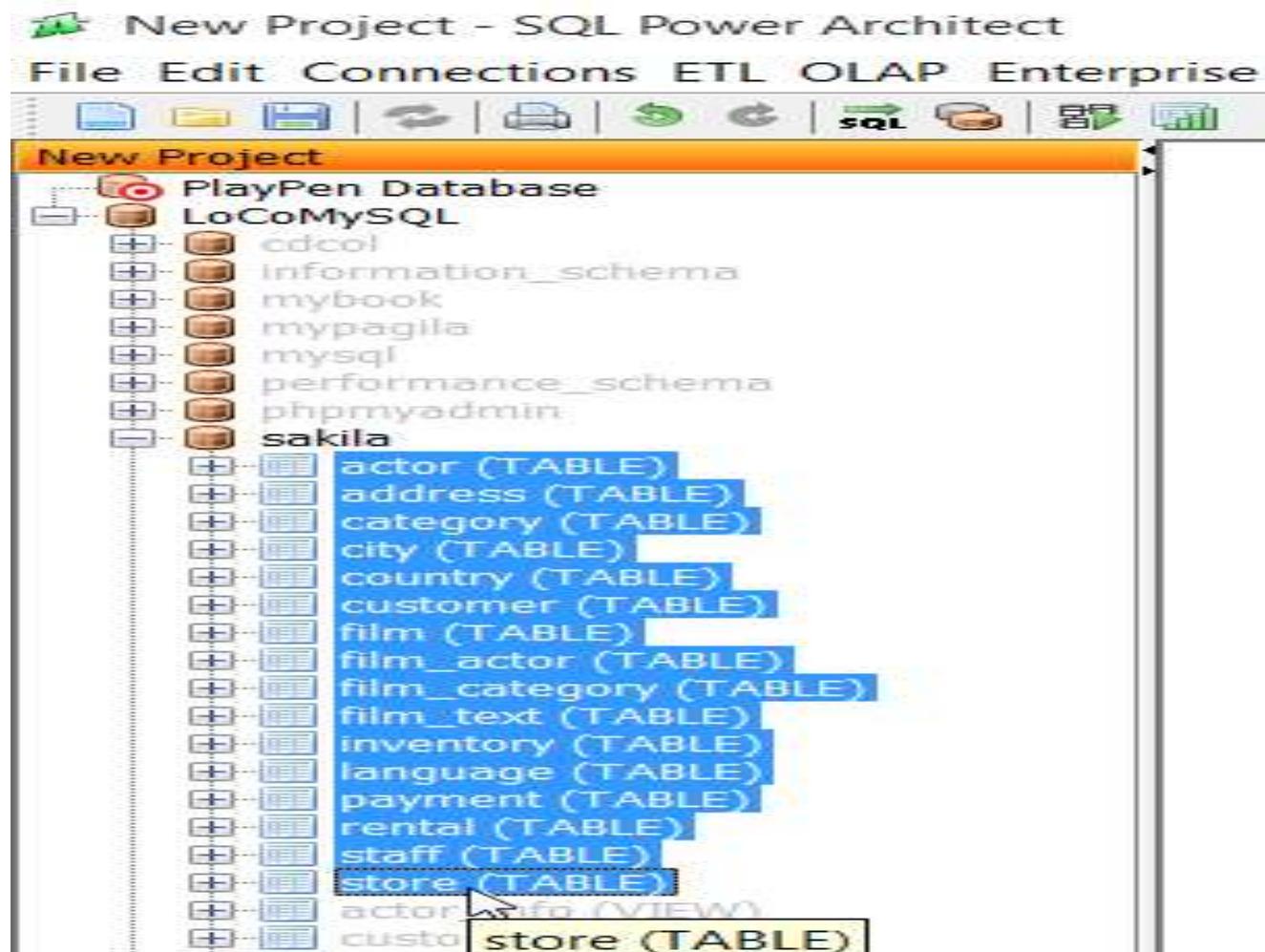
# SQL Power Architect – Data Modeling



# SQL Power Architect – Data Modeling

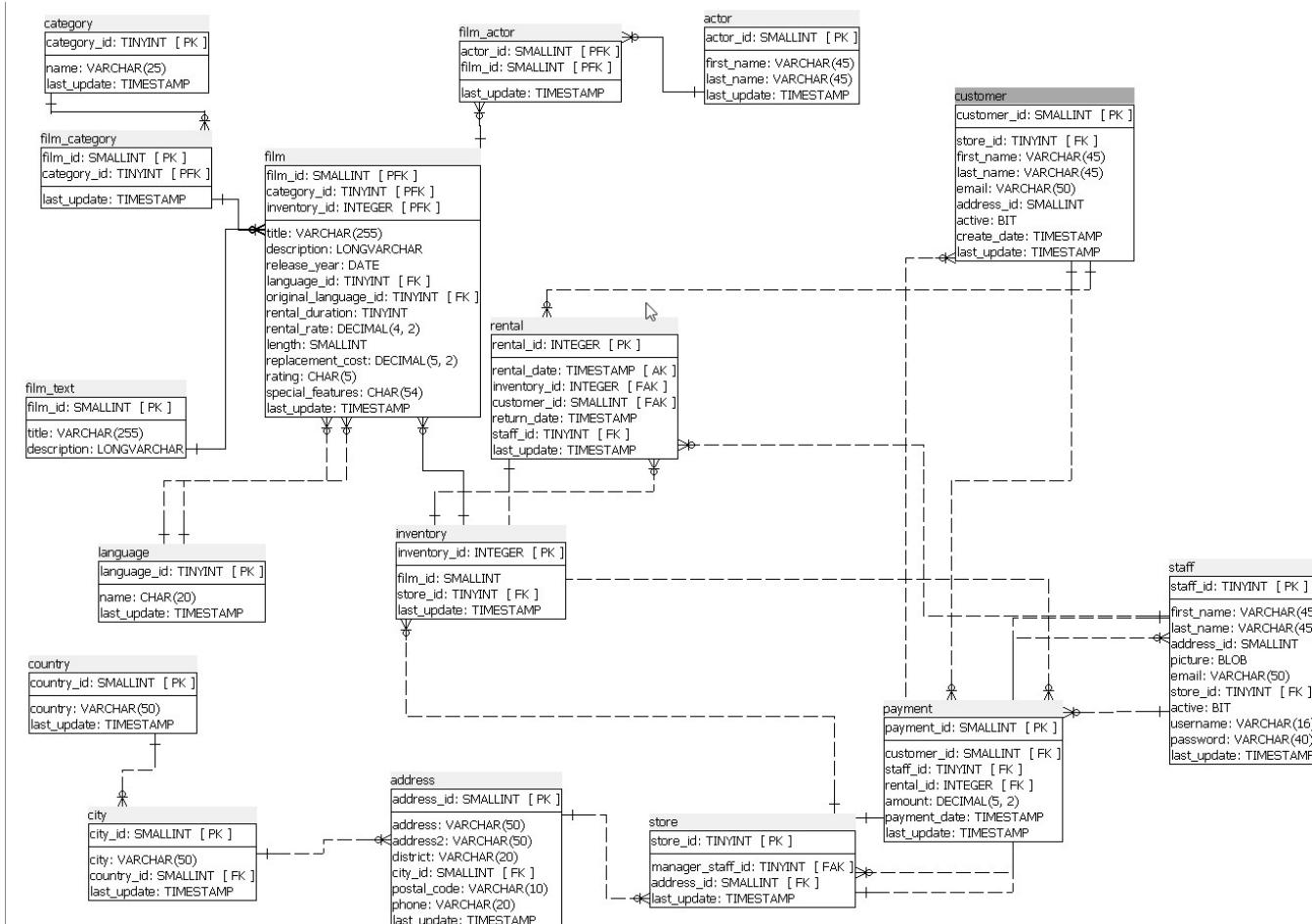


# Reverse Engineering



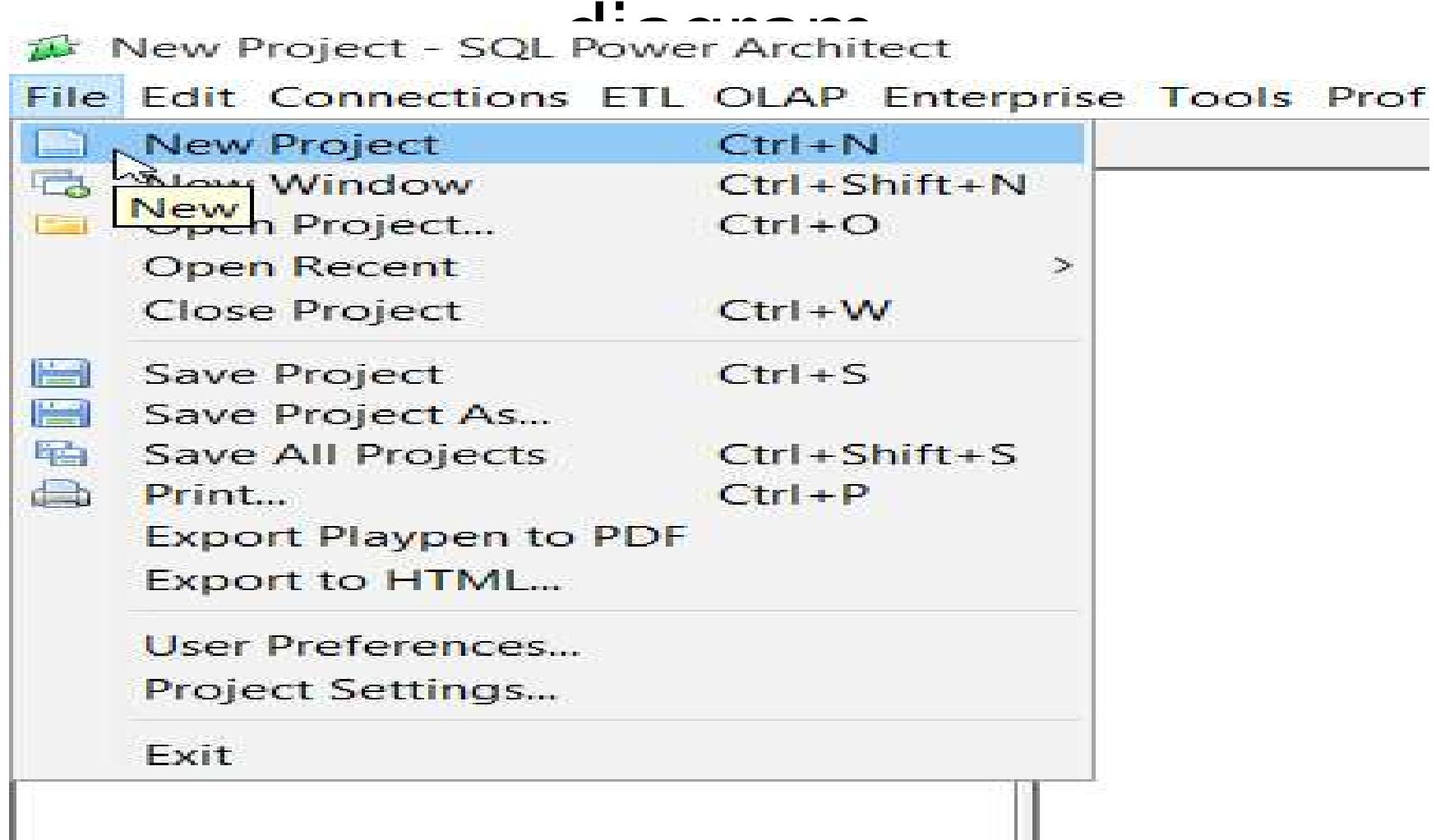
# Sample database

## Entity Relationship diagram



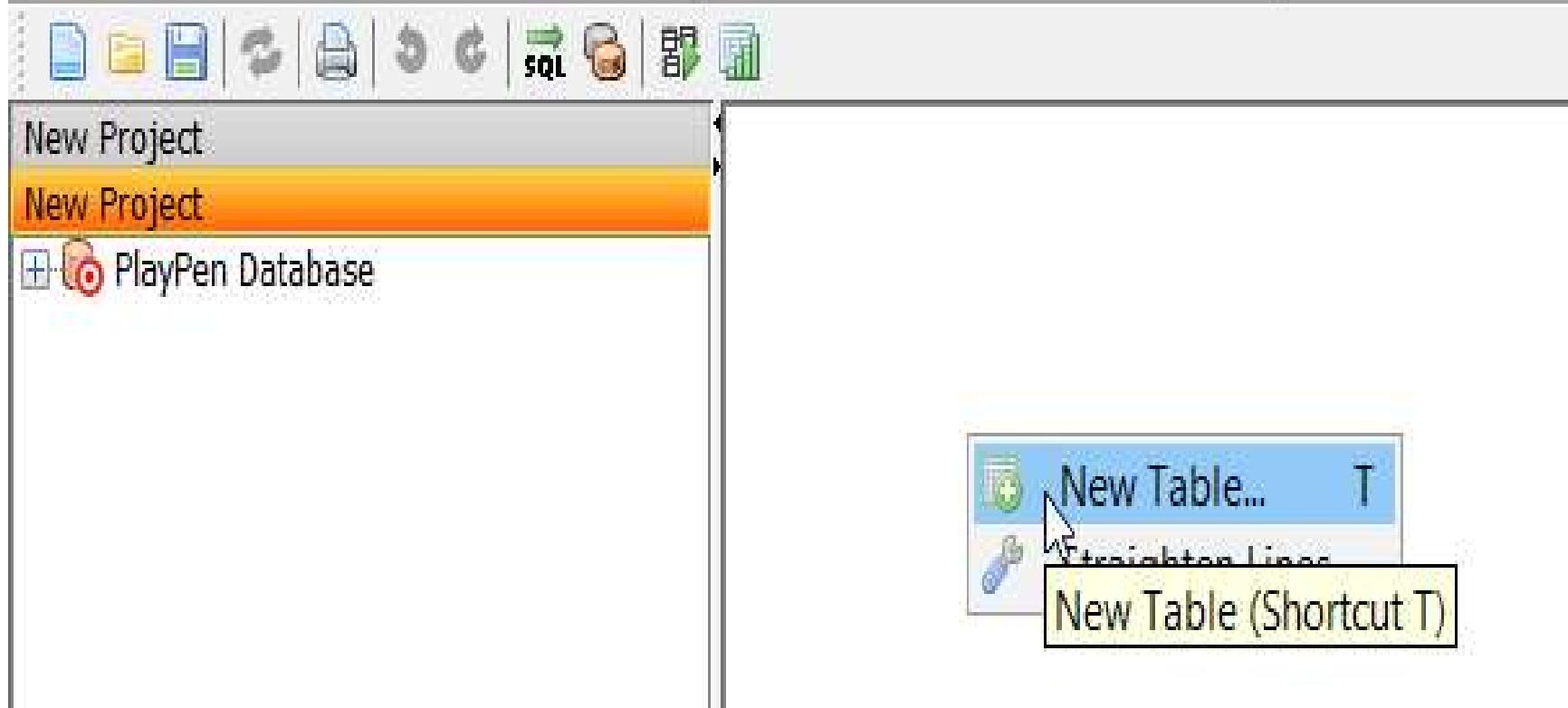
# Forward Engineering

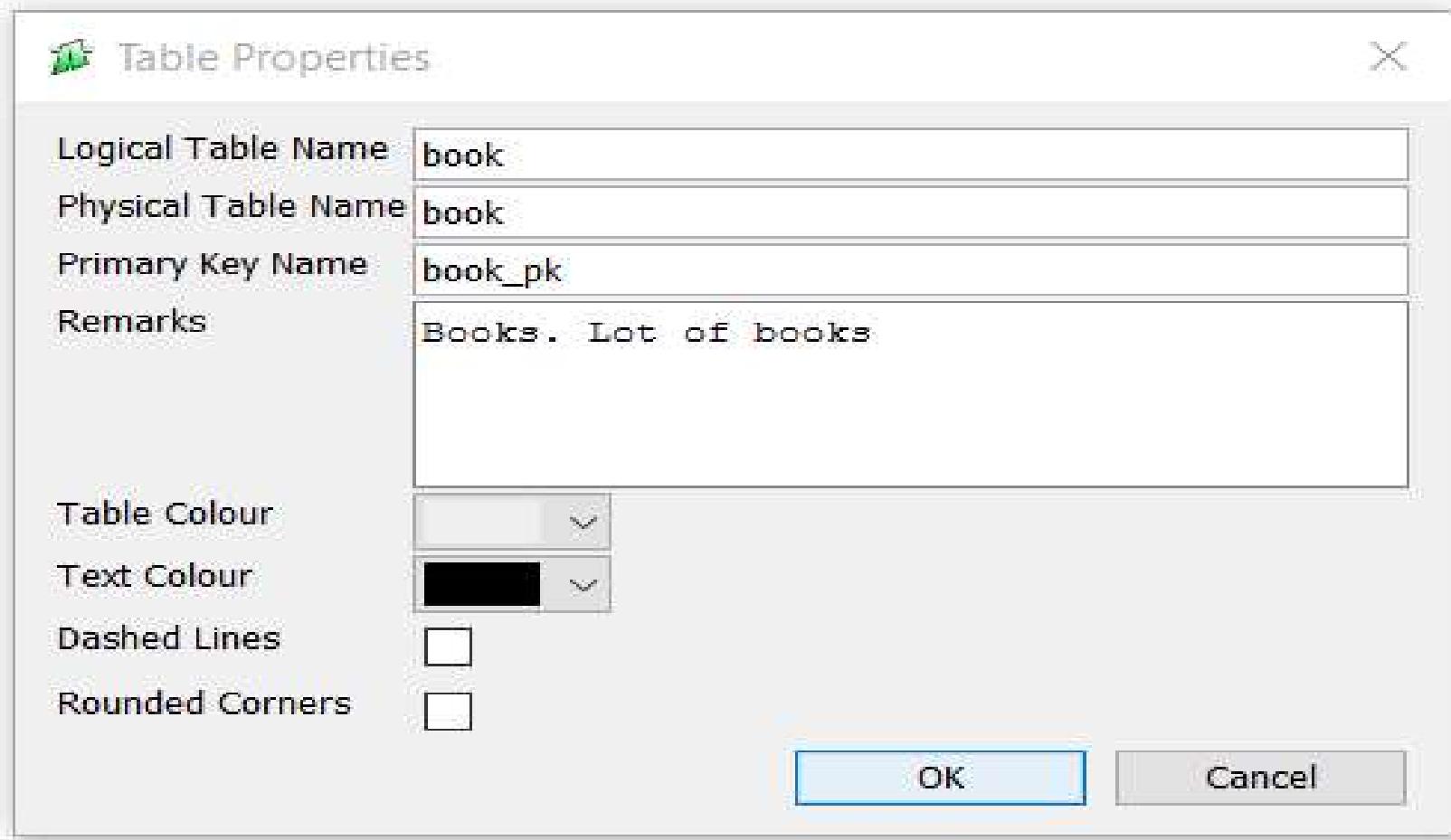
## Create Entity – Relationship



# New Project - SQL Power Architect

File Edit Connections ETL OLAP Enterprise Tools Profile Window Help







 Column Properties of New Column X

**Source for ETL Mapping**

**Logical Name**

**Physical Name**

In Primary Key

**Type**

**Precision**   **Scale**

**Allows Nulls**

**Auto Increment**

**Default Value**

**Sequence Name (Only applies to target platforms that use sequences)**

**Remarks**

OK Cancel

book

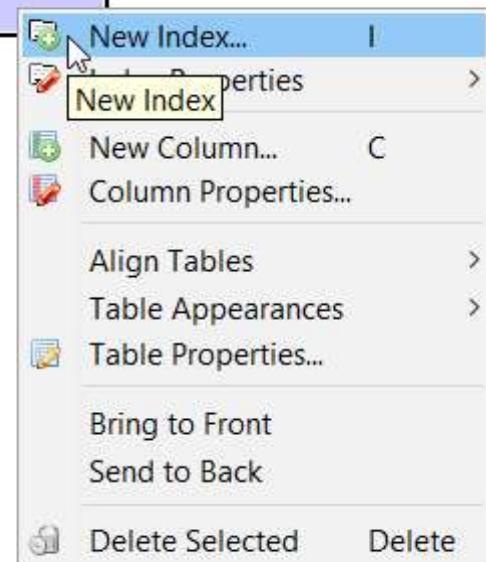
ISBN: CHAR(130) [ PK ]

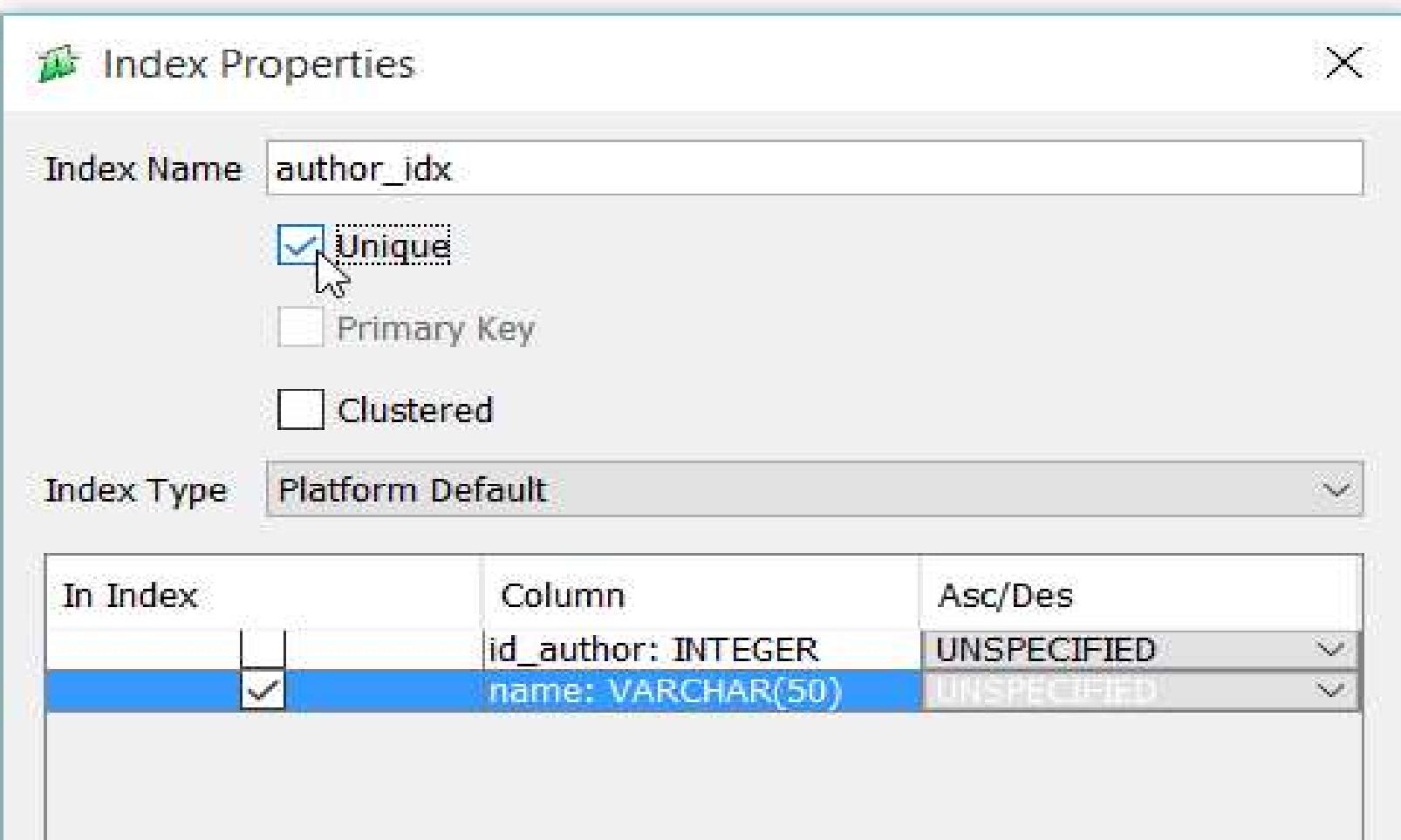
author\_id: INTEGER  
publisher\_id: INTEGER  
title: VARCHAR(80)  
price: FLOAT

author

id\_author: INTEGER [ PK ]

name: VARCHAR(50)



Index Properties

Index Name: author\_idx

Unique  
 Primary Key  
 Clustered

Index Type: Platform Default

| In Index                            | Column             | Asc/Des     |
|-------------------------------------|--------------------|-------------|
|                                     | id_author: INTEGER | UNSPECIFIED |
| <input checked="" type="checkbox"/> | name: VARCHAR(50)  | UNSPECIFIED |

# New Identifying Relationship

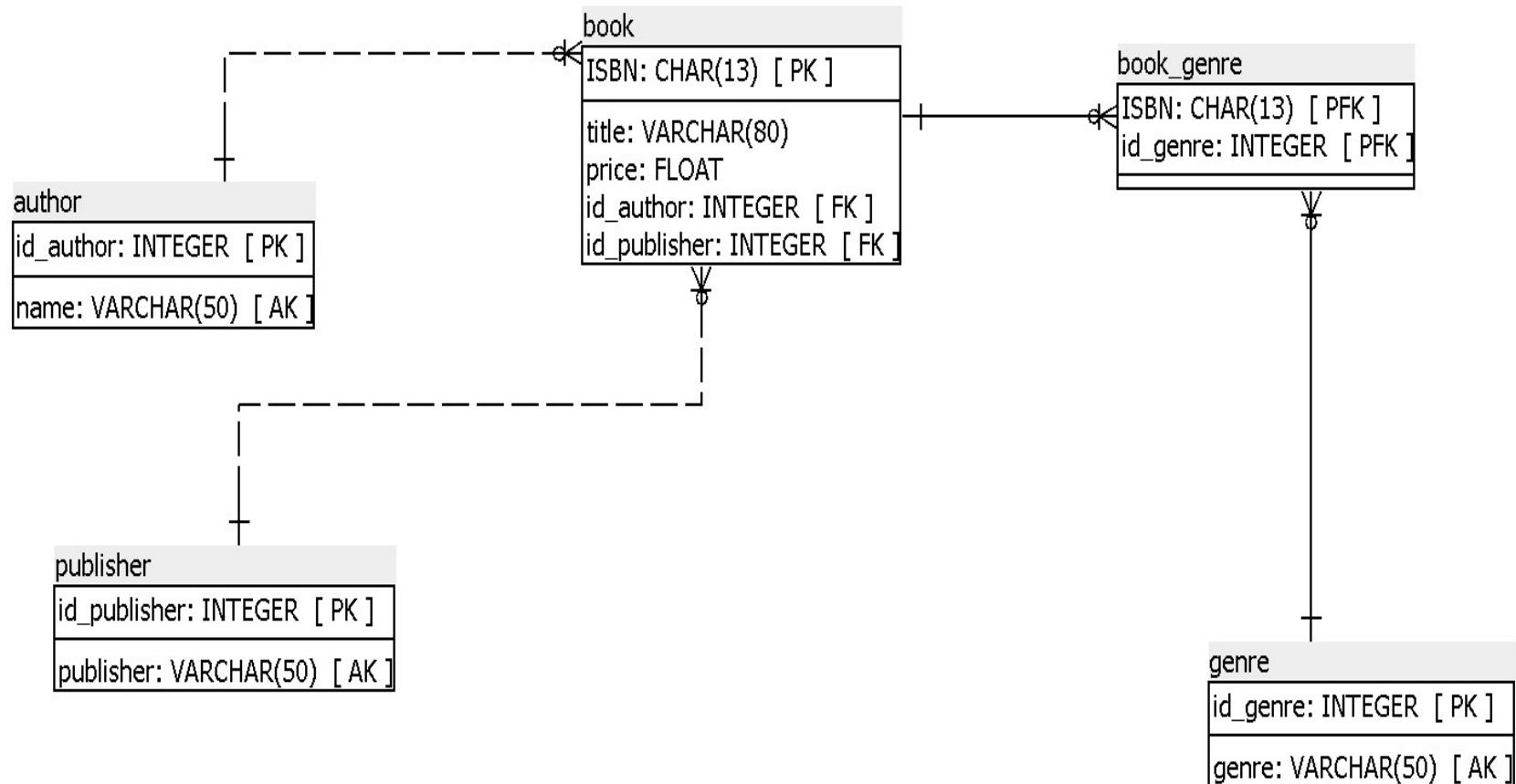
## book\_genre

|                           |
|---------------------------|
| ISBN: CHAR(13) [ PK ]     |
| genre_id: INTEGER [ PK ]  |
| id_genre: VARCHAR [ PFK ] |

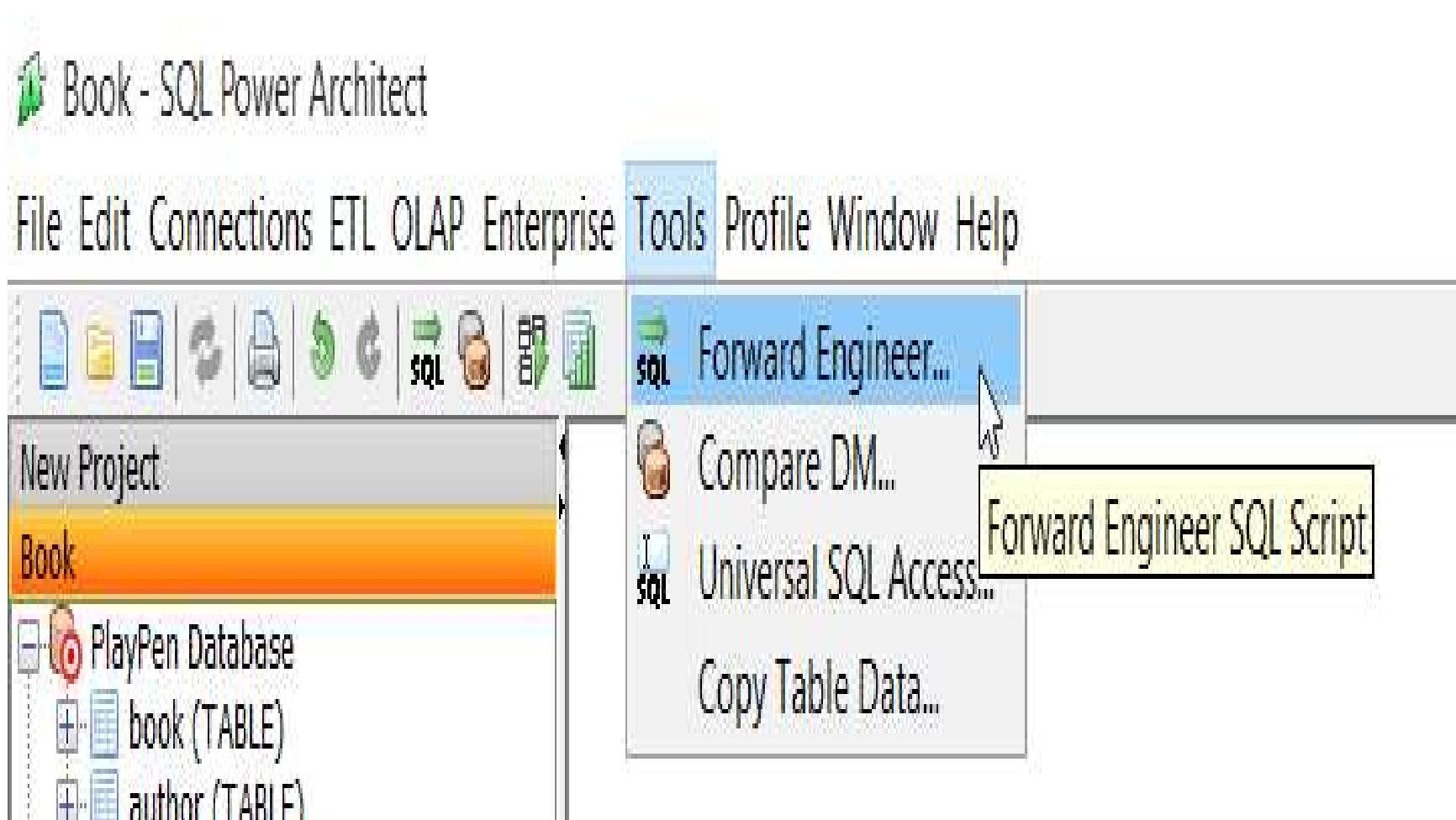


## genre

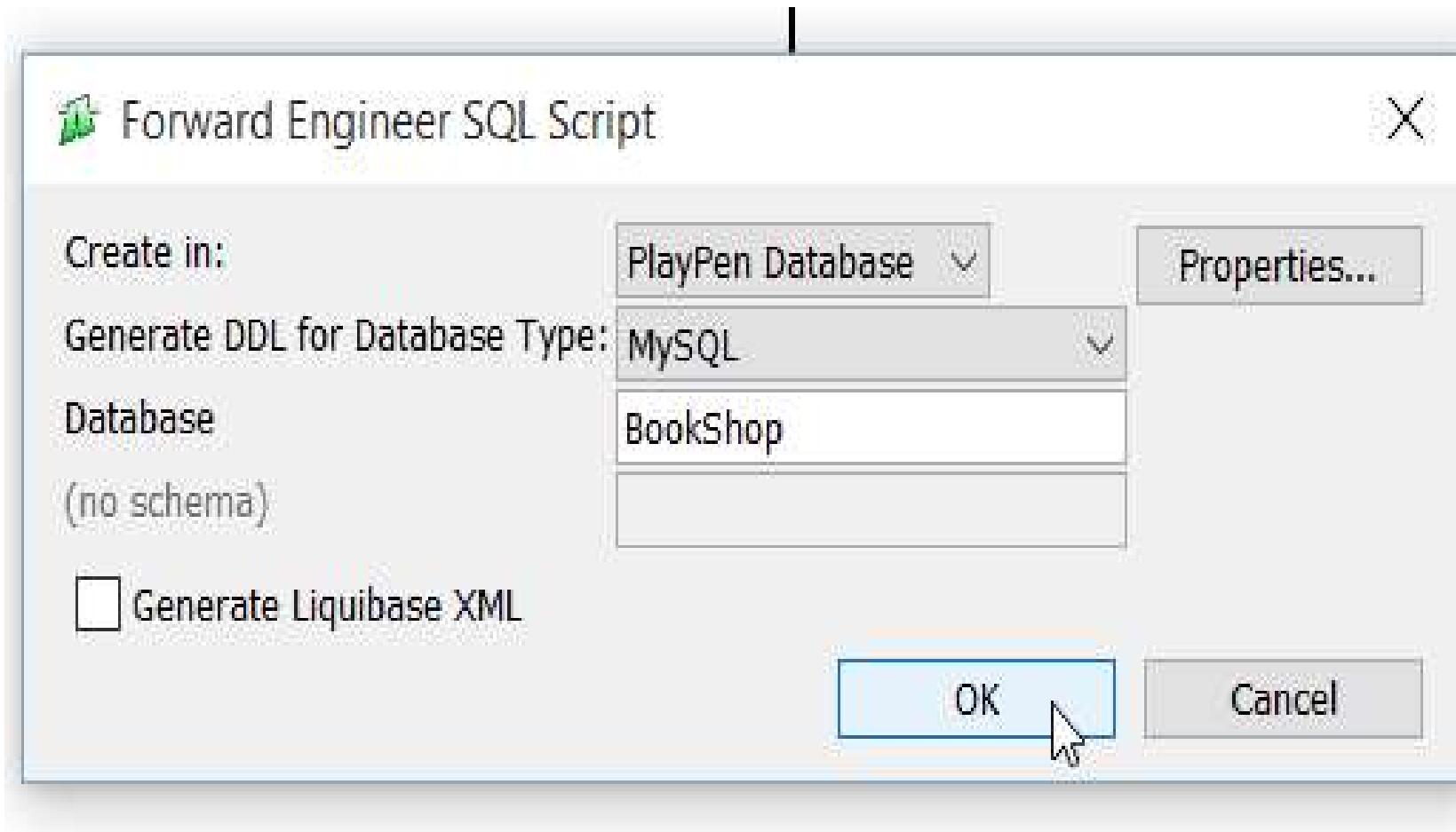
|                           |
|---------------------------|
| id_genre: VARCHAR [ PK ]  |
| genre: VARCHAR(50) [ AK ] |



# Forward Engineering



# Forward Engineering



# Forward Engineering

- Save SQL file
- Start DataBase Engine
  - MySQL / Microsoft SQL Server
- Start Front-End tool
  - MySQL WorkBench / Management Studio
- CREATE SCHEMA / DATABASE
- Run SQL file

- Generic SQL 92 standard
- Oracle
- PostgreSQL
- Microsoft SQL Server
- MySQL
- IBM DB2
- HSQLDB

# DataBases

## DataBase Design Normalization Process

**“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 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

# Objectives

- Describe data normalization process
- Perform data normalization process
- Test tables for irregularities using data normalization process

# Database Design

- Conceptual Data Modeling
- Normalization Process
- Implementing Base Table Structures

# **NORMALIZATION PROCESS**

# Normalization

- process of taking entities and attributes that have been discovered and making them suitable for the relational database
- process does this by removing redundancies and shaping data in manner that the relational engine desires

# Normalization

- based on a set of levels, each of which achieves level of correctness or adherence to a particular set of rules
- rules formally known as forms, normal forms
- First Normal Form(1NF)
  - which eliminates data redundancy
- and continue through to
- Fifth Normal Form (5NF)
  - which deals with decomposition of ternary relationships

# Normalization

- each level of normalization indicates an increasing degree of adherence to the recognized standards of database design
- as you increase degree of normalization of your data, you'll naturally tend to create an increasing number of tables of decreasing width (fewer columns)

# Why Normalize?

- eliminate data that's duplicated, chance it won't match when you need it
- avoid unnecessary coding needed to keep duplicated data in sync
- keep tables thin, increase number of values that will fit on a page (8K) – decrease number of reads that will be needed
- maximizing use of clustered indexes – allow for more optimum data access and joins
- lowering number of indexes per table - indexes are costly to maintain

# Eliminating duplicated data

- any piece of data that occurs more than once in database is an error waiting to happen

# Process of Normalization

- take entities that are complex and extract simpler entities from them
- continues until every table in database represents one thing and every column describes that thing

# 3 categories of normalization steps

- entity and attribute shape
- relationships between attributes
- multi-valued and join dependencies in entities

# Entity and attribute shape

## First Normal Form

- all attributes must be atomic, that is, only a single value represented in a single attribute in a single instance of an entity
- all instances of an entity must contain the same number of values
- all instances of an entity must be different

# First Normal Form

- violations often manifest themselves in implemented model with data handling being far less optimal, usually because of having to decode multiple values stored where a single one should be or because of having duplicated rows that cannot be distinguished from one another

- for example, consider group of data like 1, 2, 3, 5, 7
- likely represents five separate values
- atomicity is to consider whether you would ever need to deal with part of column without other parts of data in that same column
- 1, 2, 3, 5, 7 list always treated as single value, it might be acceptable to store value in single column
- if you might need to deal with value 3 individually, then the list is definitely not in First Normal Form
- if there is not plan to use list elements individually, you should consider whether it is still better to store each value individually to allow for future possible usage

# E-Mail Addresses

- name1@domain1.com
- AccountName: name1
- Domain: domain1.com

# E-Mail Addresses

- intend to access individual parts
- all you'll ever do is send e-mail, then single column is perfectly acceptable
- need to consider what domains you have e-mail addresses stored for, then it's a completely different matter

# Telephone Numbers

- AAA-EEE-DDDD (XXXX):
- AAA area code – indicates calling area located within a state
- EEE exchange - indicates a set of numbers within an area code
- DDDD number - used to make individual phone numbers unique
- XXXX extension - number that must be dialed after connecting

# Mailing Addresses

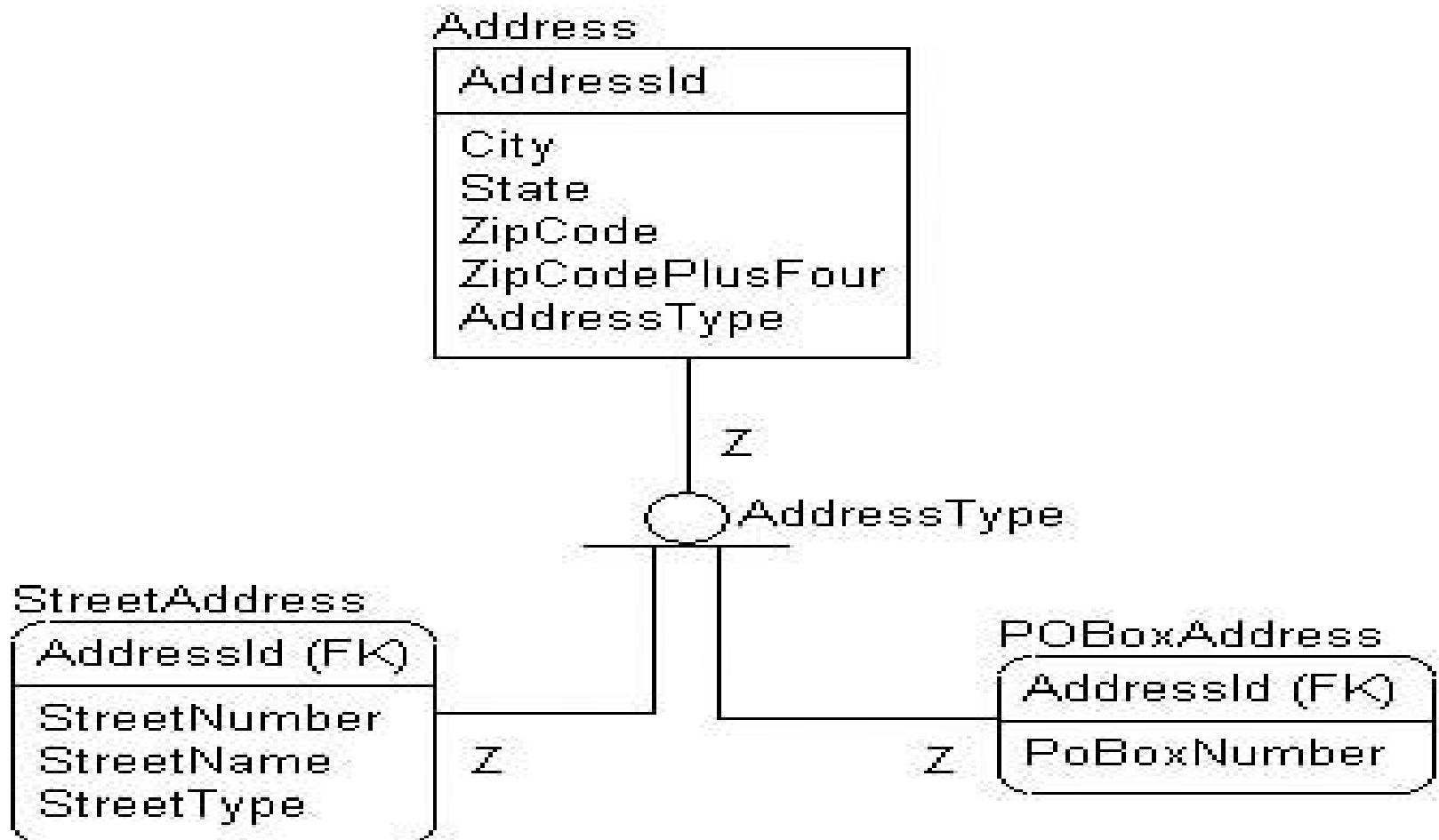
Address

|                 |
|-----------------|
| AddressId       |
| AddressLine1    |
| AddressLine2    |
| City            |
| State           |
| ZipCode         |
| ZipCodePlusFour |

Address

|                 |
|-----------------|
| AddressId       |
| StreetName      |
| StreetNumber    |
| StreetType      |
| City            |
| State           |
| ZipCode         |
| ZipCodePlusFour |

# Mailing Addresses



# All instances in entity contain same number of values

- entities have a fixed number of attributes
  - and tables have a fixed number of columns
- entities should be designed such that every attribute has a fixed number of values associated with it
- example of a violation of this rule in entities that have several attributes with same base name suffixed (or prefixed) with a number, such as Payment1, Payment2, and so on,

# Programming Anomalies avoided by First Normal Form

- modifying lists in single column
- modifying multipart values
- dealing with a variable number of facts in an instance

# Clues that design is not in First Normal Form

- string data that contains separator-type characters
- attribute names with numbers at the end
- tables with no or poorly defined keys

# Relationships Between Attributes

- Second Normal Form
  - relationships between non-key attributes and part of the primary key
- Third Normal Form
  - relationships between non-key attributes
- BCNF (Boyce Codd Normal Form)
  - relationships between non-key attributes and any key
- *Non-key attributes must provide a detail about the key, the whole key, and nothing but the key.*

# Second Normal Form

- entity must be in First Normal Form.
- each attribute must be a fact describing the entire key
- technically relevant only when a composite key (a key composed of two or more columns) exists in the entity

# Each non-key attribute must describe entire key

BookAuthor

AuthorSocialSecurityNumber

BookIsbnNumber

RoyaltyPercentage

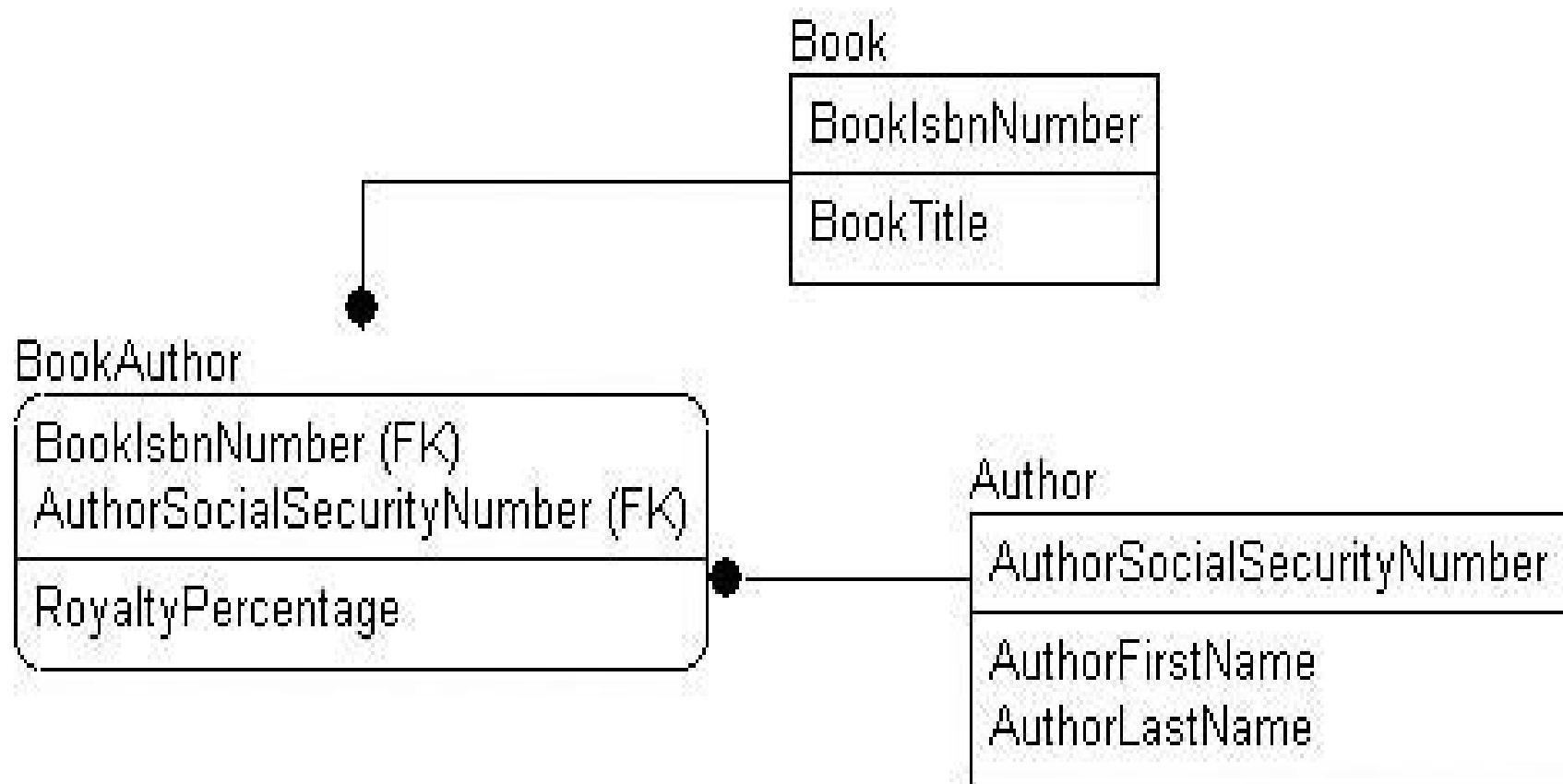
BookTitle

AuthorFirstName

AuthorLastName

- BookIsbnNumber attribute uniquely identifies book
- AuthorSocialSecurityNumber uniquely identifies author
- two columns create key that uniquely identifies an author for book
- BookTitle describes book
  - but doesn't describe author at all
- AuthorFirstName and AuthorLastName, describe author
  - but not book

- BookIsbnNumber → BookTitle
- AuthorSocialSecurityNumber → AuthorFirstName
- AuthorSocialSecurityNumber → AuthorLastName
- BookIsbnNumber,  
AuthorSocialSecurityNumber → RoyaltyPercentage



# Programming problems avoided

- all programming issues that arise with Second Normal Form (as well as Third and Boyce-Codd Normal Forms) deal with functional dependencies that can end up corrupting data

### Book Author Edit

Author Information

|            |             |
|------------|-------------|
| SSN        | 111-11-1111 |
| First Name | Fred        |
| Last Name  | Smith       |

OK

Cancel

Book Information

|       |                 |
|-------|-----------------|
| ISBN  | 1234567890      |
| Title | Database Design |

Royalty Percentage

|    |
|----|
| 15 |
|----|

- same author's information would have to be duplicated amongst all books
- cannot delete only book and keep author around
- cannot insert only author without book

# Anomalies

- UPDATE
  - duplicate date, have to update multiple rows
- INSERT
  - cannot insert data for an entity without relationship to any other entity
- DELETE
  - cannot delete data for an entity without risk of losing info. about related entity

# Clues that entity is not in Second Normal Form

- repeating key attribute name prefixes, indicating that values are probably describing some additional entity
- data in repeating groups, showing signs of functional dependencies between attributes
- composite keys without foreign key, which might be sign you have key values that identify multiple things in key

# Third Normal Form

- entity must be in Second Normal Form.
- non-key attributes cannot describe other non-key attributes

non-key attributes cannot describe  
other non-key attributes

Book

BookIsbnNumber

Title

Price

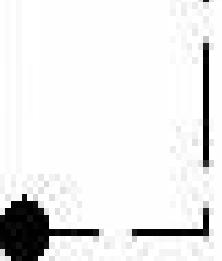
PublisherName

PublisherCity

- Title defines title for the book defined by BookIsbnNumber
- Price indicates price of the book
- PublisherName describes the book's publisher
- PublisherCity also sort of describes something about the book, in that it tells where the publisher was located
- doesn't make sense in this context, because location of publisher is directly dependent on what publisher is represented by PublisherName

Book

|                    |
|--------------------|
| BookIsbnNumber     |
| Title              |
| Price              |
| PublisherName (FK) |



Publisher

|               |
|---------------|
| PublisherName |
| City          |

- Publisher entity has data concerning only publisher
- Book entity has book information
  - now if we want to add information to our schema concerning the publisher, contact information or address, it's obvious where we add that information
- City attribute clearly identifying publisher
  - not the book

# Clues that entities are not in Third Normal Form

- multiple attributes with same prefix
  - much like Second Normal Form, only this time not in the key
- repeating groups of data
- summary data that refers to data in a different entity altogether
  - Price in Invoice as  $\text{SUM}(\text{Quantity} * \text{ProductCost})$  from LineItems

# Boyce-Codd Normal Form

- Ray Boyce, Edgar F. Codd
- entity is in First Normal Form.
- all attributes are fully dependent on a key
- every determinant is a key

# Entity in BCNF if every Determinant is key

- *Any attribute or combination of attributes on which any other attribute or combination of attributes is functionally dependent.*
- BCNF extends previous normal forms by saying that entity might have many keys, and all attributes must be dependent on one of these keys

- Third Normal Form table which does not have multiple overlapping candidate keys is guaranteed to be in BCNF
- Third Normal Form table with two or more overlapping candidate keys may or may not be in BCNF

# Court Bookings

| Court | Start Time | End Time | Rate Type |
|-------|------------|----------|-----------|
| 1     | 09:30      | 10:30    | SAVER     |
| 1     | 11:00      | 12:00    | SAVER     |
| 1     | 14:00      | 15:30    | STANDARD  |
| 2     | 10:00      | 11:30    | PREMIUM-B |
| 2     | 11:30      | 13:30    | PREMIUM-B |
| 2     | 15:00      | 16:30    | PREMIUM-A |

# Court Bookings

- hard court (Court1) and grass court (Court2)
- booking defined by Court and period for which the Court is reserved
- booking has Rate Type associated
  - SAVER for hard made by members
  - STANDARD for hard made by non-members
  - PREMIUM-A for grass made by members
  - PREMIUM-B for grass made by non-members

# Court Bookings - candidate keys

- {Court, Start Time}
- {Court, End Time}
- {Rate Type, Start Time}
- {Rate Type, End Time}

- table adheres to both 2NF and 3NF
- table does not adhere to BCNF
- because of dependency Rate Type → Court, in which the determining attribute (Rate Type) is neither a candidate key nor a superset of a candidate key

## Rate Types

| Rate Type | Court | Member Flag |
|-----------|-------|-------------|
| SAVER     | 1     | Yes         |
| STANDARD  | 1     | No          |
| PREMIUM-A | 2     | Yes         |
| PREMIUM-B | 2     | No          |

## Court Bookings

| Court | Start Time | End Time | Member Flag |
|-------|------------|----------|-------------|
| 1     | 09:30      | 10:30    | Yes         |
| 1     | 11:00      | 12:00    | Yes         |
| 1     | 14:00      | 15:30    | No          |
| 2     | 10:00      | 11:30    | No          |
| 2     | 11:30      | 13:30    | No          |
| 2     | 15:00      | 16:30    | Yes         |

- candidate keys for Rate Types table are {Rate Type} and {Court, Member Flag}
- candidate keys for Court Bookings table are {Court, Start Time} and {Court, End Time}
- both tables are in BCNF
- having one Rate Type associated with two different Courts is now impossible
- anomaly affecting original table has been eliminated

# Multivalued Dependencies

- Third Normal Form is generally considered pinnacle of proper database design
- serious problems might still remain in logical design

# Fourth Normal Form

- entity must be in BCNF
- there must not be more than one multivalued dependency between an attribute and the key of the entity

# Fourth Normal Form

- table is in 4NF if and only if, for every one of its non-trivial multivalued dependencies  $X \twoheadrightarrow Y$ ,  $X$  is a super key,  $X$  is either candidate key or superset thereof

# Fourth Normal Form violations

- ternary relationships
- lurking multivalued attributes
- temporal data/value history

| Restaurant       | Pizza Variety | Delivery Area |
|------------------|---------------|---------------|
| A1 Pizza         | Thick Crust   | Springfield   |
| A1 Pizza         | Thick Crust   | Shelbyville   |
| A1 Pizza         | Thick Crust   | Capital City  |
| A1 Pizza         | Stuffed Crust | Springfield   |
| A1 Pizza         | Stuffed Crust | Shelbyville   |
| A1 Pizza         | Stuffed Crust | Capital City  |
| Elite Pizza      | Thin Crust    | Capital City  |
| Elite Pizza      | Stuffed Crust | Capital City  |
| Vincenzo's Pizza | Thick Crust   | Springfield   |
| Vincenzo's Pizza | Thick Crust   | Shelbyville   |
| Vincenzo's Pizza | Thin Crust    | Springfield   |
| Vincenzo's Pizza | Thin Crust    | Shelbyville   |

- table has no non-key attributes
- meets all normal forms up to BCNF
- not in 4NF, non-trivial multivalued dependencies
- $\{\text{Restaurant}\} \rightarrow\!\!\!\rightarrow \{\text{Pizza Variety}\}$
- $\{\text{Restaurant}\} \rightarrow\!\!\!\rightarrow \{\text{Delivery Area}\}$
- eliminate possibility of anomalies

| <b>Restaurant</b> | <b>Pizza Variety</b> | <b>Restaurant</b> | <b>Delivery Area</b> |
|-------------------|----------------------|-------------------|----------------------|
| A1 Pizza          | Thick Crust          | A1 Pizza          | Springfield          |
| A1 Pizza          | Stuffed Crust        | A1 Pizza          | Shelbyville          |
| Elite Pizza       | Thin Crust           | A1 Pizza          | Capital City         |
| Elite Pizza       | Stuffed Crust        | Elite Pizza       | Capital City         |
| Vincenzo's Pizza  | Thick Crust          | Vincenzo's Pizza  | Springfield          |
| Vincenzo's Pizza  | Thin Crust           | Vincenzo's Pizza  | Shelb                |

- in contrast, if pizza varieties offered by restaurant sometimes did legitimately vary from one delivery area to another, the original three-column table would satisfy 4NF

# Fifth Normal Form

- not every ternary relationship can be broken down into two entities related to a third
- aim of 5NF is to ensure that any ternary relationships that still exist in 4NF that can be decomposed into entities are decomposed
- eliminates problems with update anomalies due to multivalued dependencies
  - much like in 4NF only these are trickier to find.

# Denormalization

- used primarily to improve performance in cases where overnormalized structures are causing overhead to query processor
- whether slightly slower (but 100 percent accurate) application is preferable to a faster application of lower accuracy
- during logical modeling, we should never step back from our normalized structures to performance-tune our applications proactively

# Key Terms

- Data normalization
- Database design / Data structures
- Logical database design
- Entity-relationship diagram conversion
- Functional dependencies
- Redundant data
- 1<sup>st</sup> Normal Form, 2<sup>nd</sup> NF, 3<sup>rd</sup> NF
- Boyce – Codd Normal Form

# Normalization of Database Tables

## Another Example

# 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:  $\text{PROJ\_NUM} \rightarrow \text{PROJ\_NAME}$ 
  - read as “ $\text{PROJ\_NUM}$  functionally determines  $\text{PROJ\_NAME}$ ”
  - attribute  $\text{PROJ\_NUM}$  is known as “determinant” attribute, and attribute  $\text{PROJ\_NAME}$  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

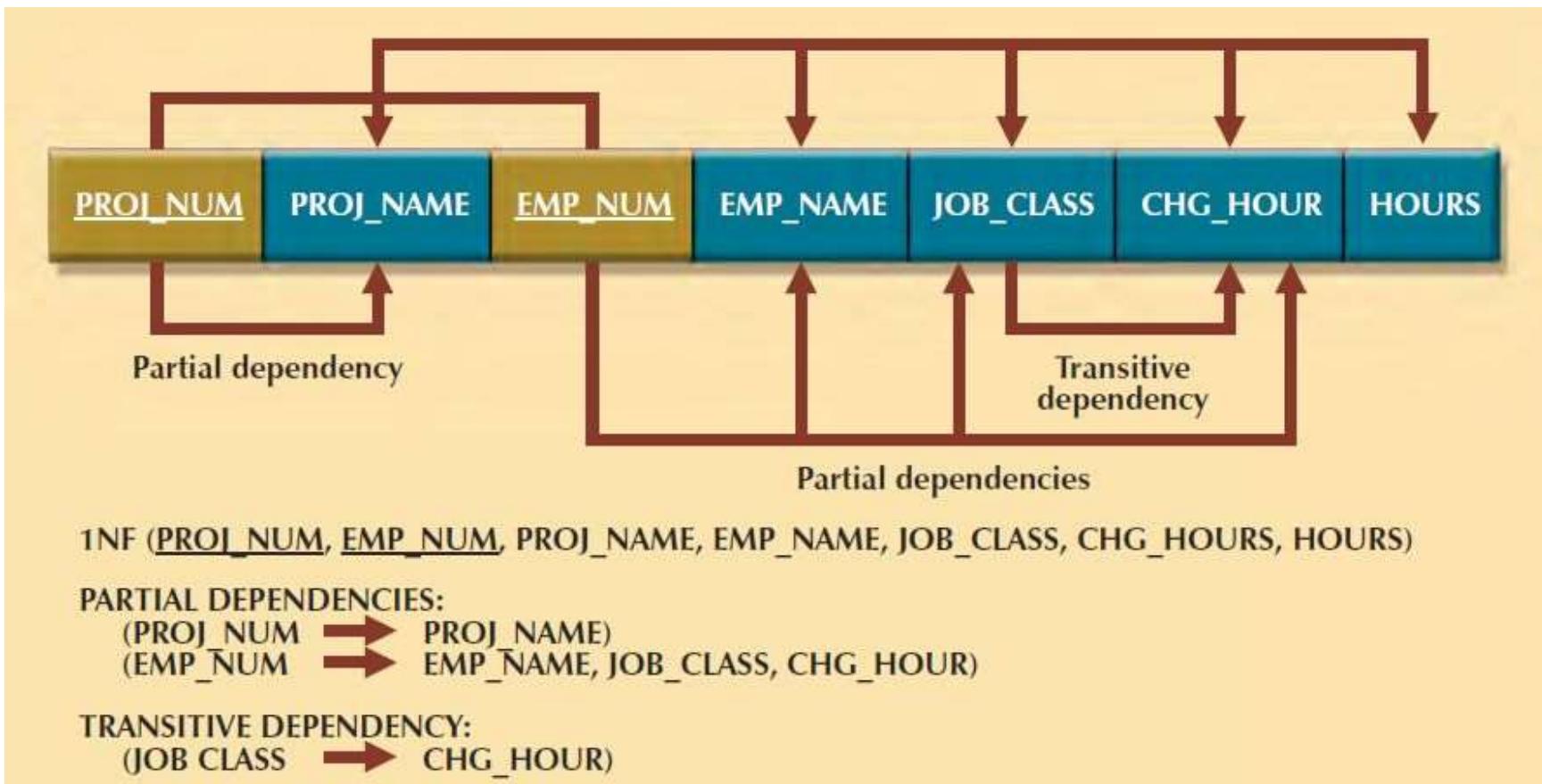
## Step 2: Identify the Primary Key

- 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

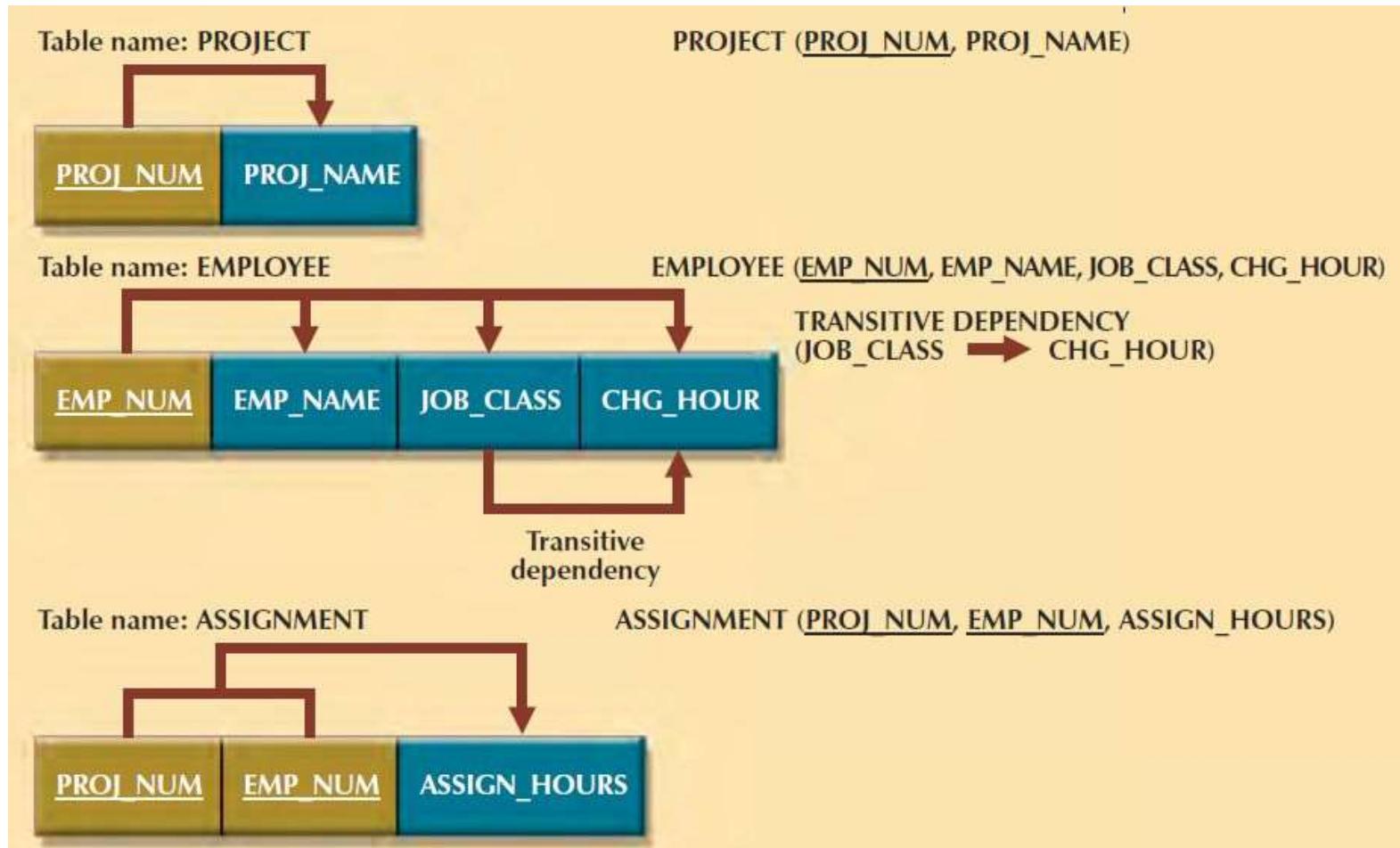
- PROJ\_NUM
- EMP\_NUM
- PROJ\_NUM EMP\_NUM
- each component will become key in new table
- original table is now divided into three tables (PROJECT, EMPLOYEE, and ASSIGNMENT)

# 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

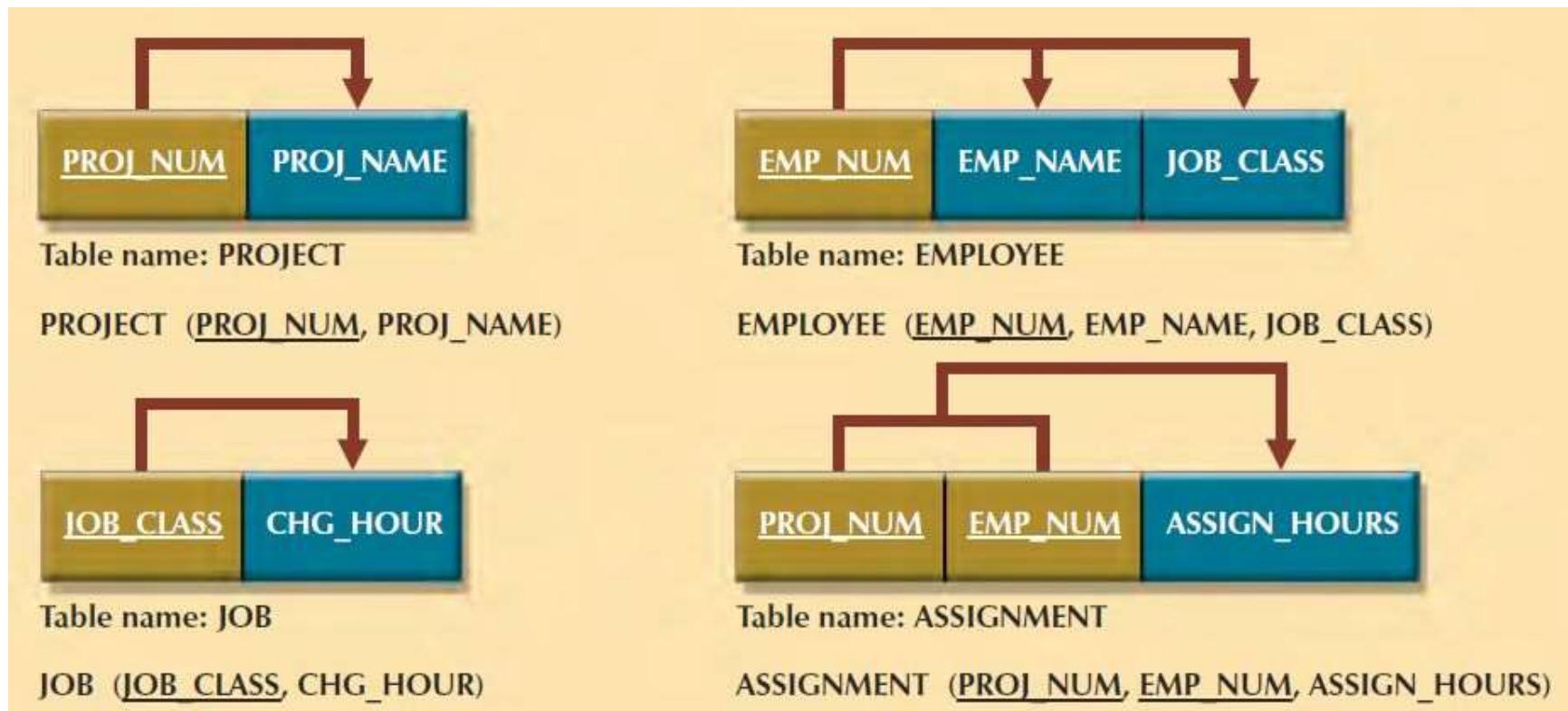
- for every transitive dependency, write a copy of its determinant as primary key for new table
- determinant is any attribute whose value determines other values within a row
- if you have three different transitive dependencies, you will have three different determinants
- determinant remain in original 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)



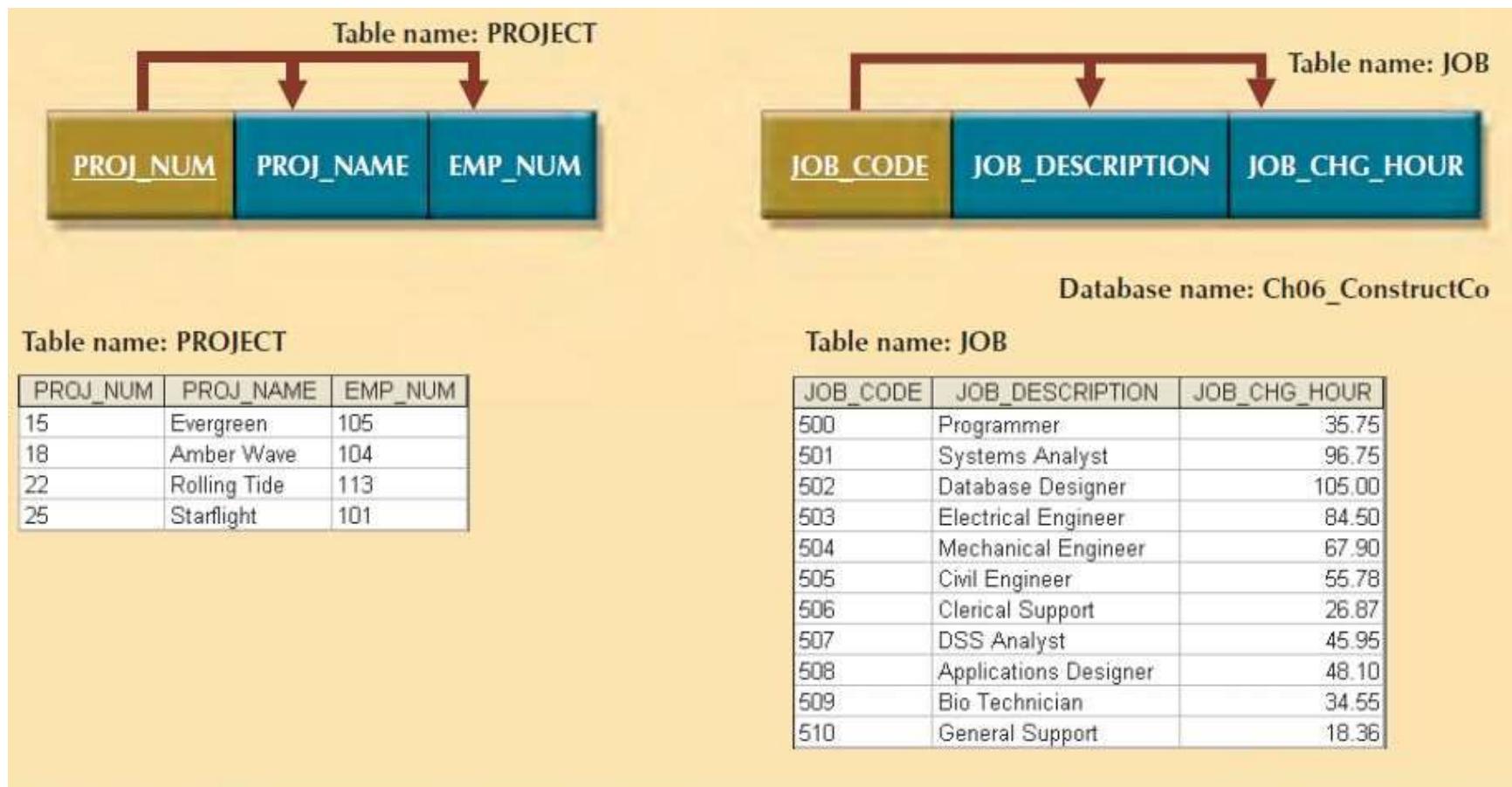
# third normal form (3NF)

- 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          | 18.38           | 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          | 46.95           | 55.14         |
| 1017       | 06-Mar-10   | 18       | 118     | 2.0          | 18.38           | 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

| 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)

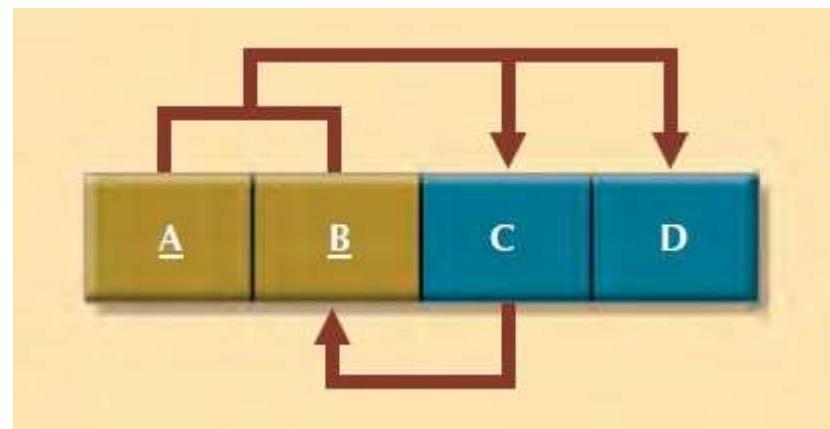
# Boyce-Codd normal form (BCNF)

- 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



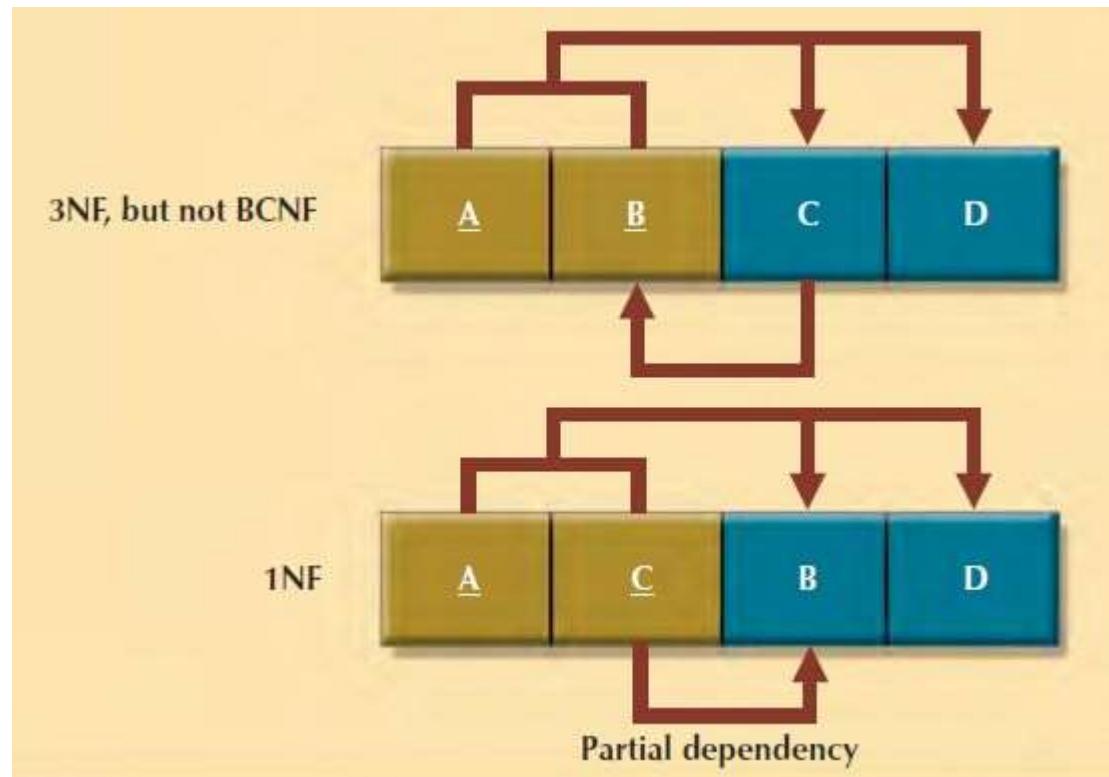
# functional dependencies

- $A + B \rightarrow C, D$
- $A + C \rightarrow B, D$
- $C \rightarrow B$
- 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

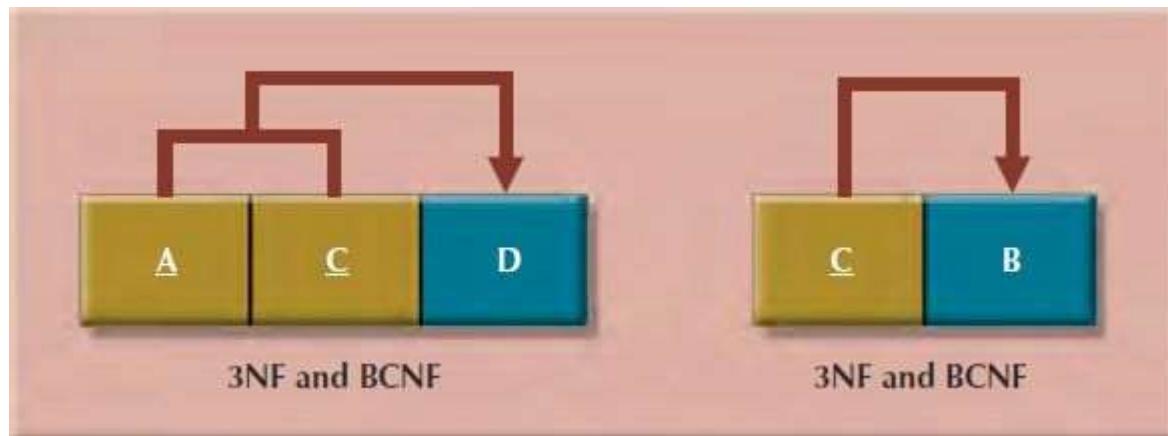
# convert into BCNF

- 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



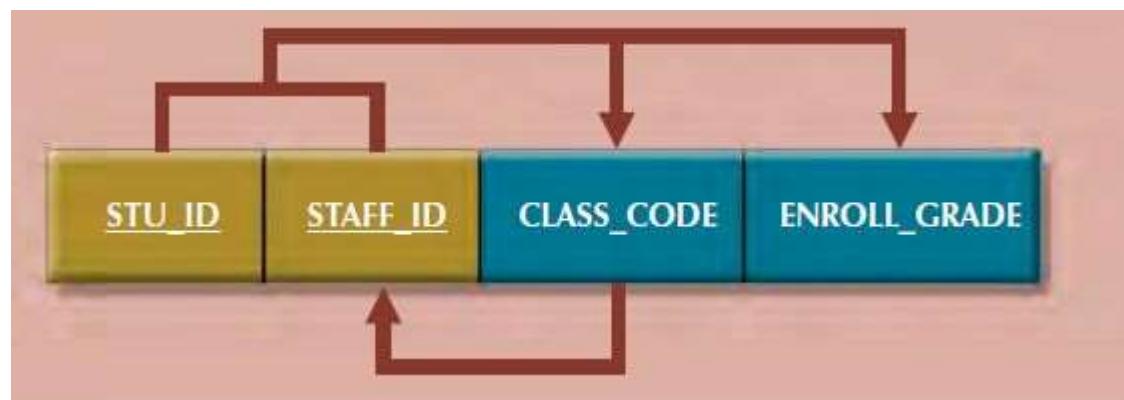
# Decomposition to BCNF



# 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



- $\text{STU\_ID} + \text{STAFF\_ID} \rightarrow \text{CLASS\_CODE}, \text{ENROLL\_GRADE}$
- $\text{CLASS\_CODE} \rightarrow \text{STAFF\_ID}$

- 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\_V3**

| EMP_NUM | ORG_CODE | ASSIGN_NUM |
|---------|----------|------------|
| 10123   | RC       | 1          |
| 10123   | RC       | 3          |
| 10123   | UW       | 4          |

**Table name: VOLUNTEER\_V2**

| EMP_NUM | ORG_CODE | ASSIGN_NUM |
|---------|----------|------------|
| 10123   | RC       |            |
| 10123   | UW       |            |
| 10123   |          | 1          |
| 10123   |          | 3          |
| 10123   |          | 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   |

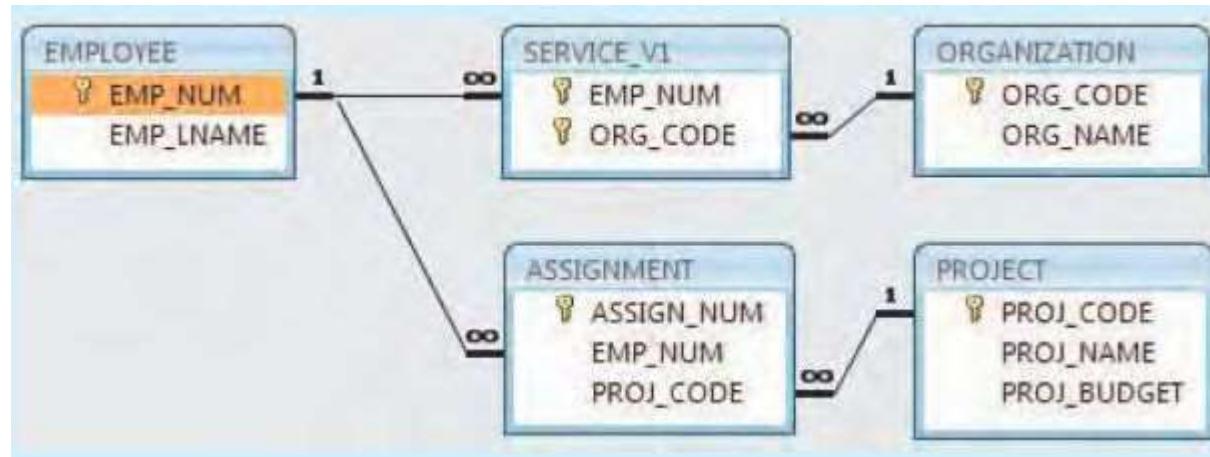
Table name: ORGANIZATION

| 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*

# Data-Modeling Checklist

# Business rules

- Properly document and verify all business rules with the end users.
- Ensure that all business rules are written precisely, clearly, and simply. The business rules must help identify entities, attributes, relationships, and constraints.
- Identify the source of all business rules, and ensure that each business rule is justified, dated, and signed off by an approving authority.

# Naming Conventions

- All 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 the model
  - For composite entities, may include a combination of abbreviated names of the entities linked through the composite entity

# Naming Conventions

- Attribute Names:
  - Should be unique within the entity
  - Should use the entity abbreviation as a prefix
  - Should be descriptive of the characteristic
  - Should use suffixes such as `_ID`, `_NUM`, or `_CODE` for the PK attribute
  - Should not be a reserved word
  - Should not contain spaces or special characters such as `@`, `!`, or `&`

# Naming Conventions

- Relationship Names:
  - Should be active or passive verbs that clearly indicate the nature of the relationship

# Entities

- Each entity should represent a single subject.
- Each entity should represent a set of distinguishable entity instances.
- All entities should be in 3NF or higher. Any entities below 3NF should be justified.
- The granularity of the entity instance should be clearly defined.
- The PK should be clearly defined and support the 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 a history
- Nonkey attributes must be fully dependent on the PK attribute

# Relationships

- Should clearly identify relationship participants
- Should clearly define participation, connectivity, and document cardinality

# ER Model

- Should be validated against expected processes: inserts, updates, and deletes
- Should evaluate where, when, and how to maintain a history
- Should not contain redundant relationships except as required (see attributes)
- Should minimize data redundancy to ensure single-place updates
- Should conform to the minimal data rule: “*All that is needed is there, and all that is there is needed.*”

# DataBase

## Database Design

# Information System

- Databases are a part of a larger picture called an information system
- database designers must recognize that database is a critical means to an end rather than an end in itself
  - managers want the database to serve their management needs
  - many databases seem to require that managers alter their routines to fit database requirements

# Systems Development Life Cycle (SDLC)

- creation and evolution of information systems follows an iterative pattern called the Systems Development Life Cycle (SDLC)
- continuous process of creation, maintenance, enhancement, and replacement of the information system
- similar cycle applies to databases

# Information System

- facilitates the transformation of data into information
- it allows for management of both data and information
- composed of people, hardware, software, database(s), application programs, and procedures
- Systems analysis
  - process that establishes need for and extent of an information system
- Systems development
  - process of creating an information system
- Purpose

# Generating information for decision making



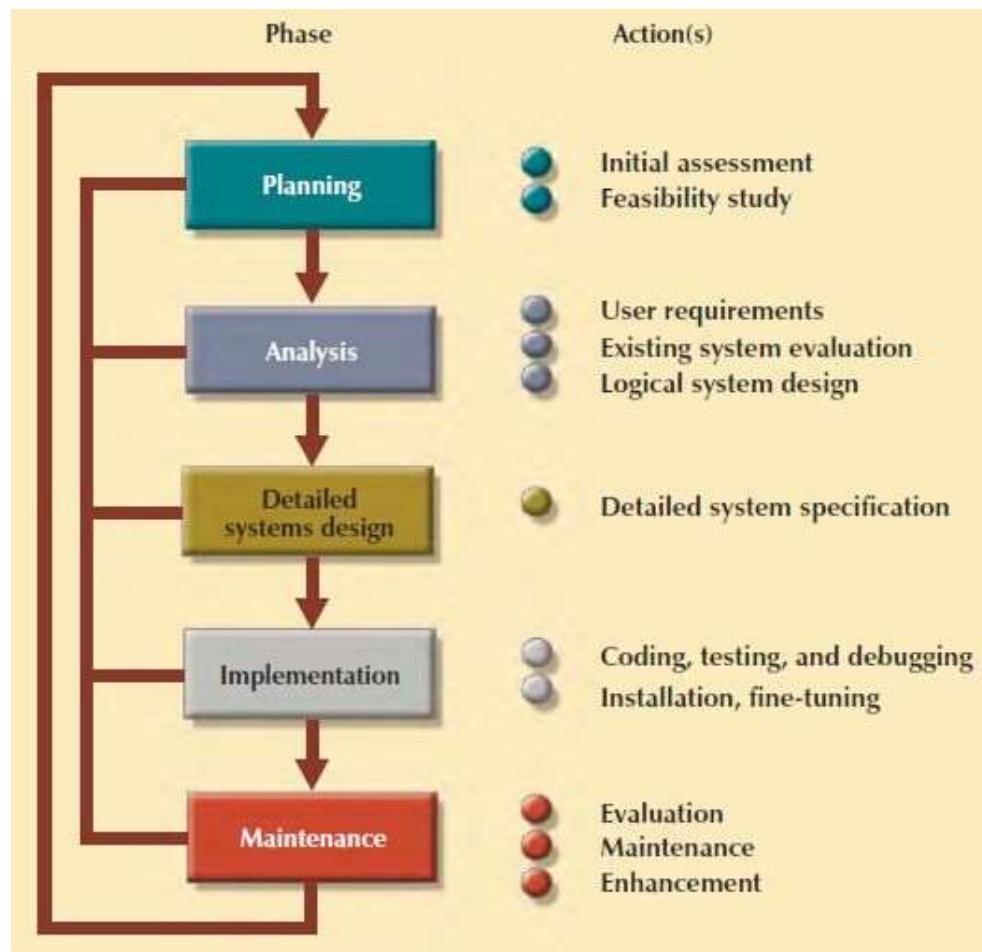
# Performance of information system

- Database design and implementation
- Application design and implementation
- Administrative procedures

# Systems Development Life Cycle

- general framework through which you can track and understand activities required to develop and maintain information systems
- there are alternative methodologies, such as:
- Unified Modeling Language (UML)
- Rapid Application Development (RAD)
- Agile Software Development

# Systems Development Life Cycle



# Systems Development Life Cycle

- traces the history (life cycle) of an information system
- traditional is divided into five phases
  - planning, analysis, detailed systems design, implementation, and maintenance
- iterative process, rather than sequential

# Planning

- yields a general overview of objectives
- Should the existing system be continued?
- Should the existing system be modified?
- Should the existing system be replaced?

# Feasibility study

- technical aspects of hardware and software requirements
- system cost
- operational cost
  - human, technical, and financial resources to keep system operational

# Analysis

- problems defined during planning phase are examined in greater detail of both individual needs and organizational needs
- What are the requirements of the current system's end users?
- Do those requirements fit into the overall information requirements?
- Is a thorough audit of user requirements

# Detailed Systems Design

- completes the design of the system's processes
- includes all necessary technical specifications for screens, menus, reports, and other
- steps are laid out for conversion from the old to the new system
- training principles and methodologies are also planned and must be submitted for management's approval

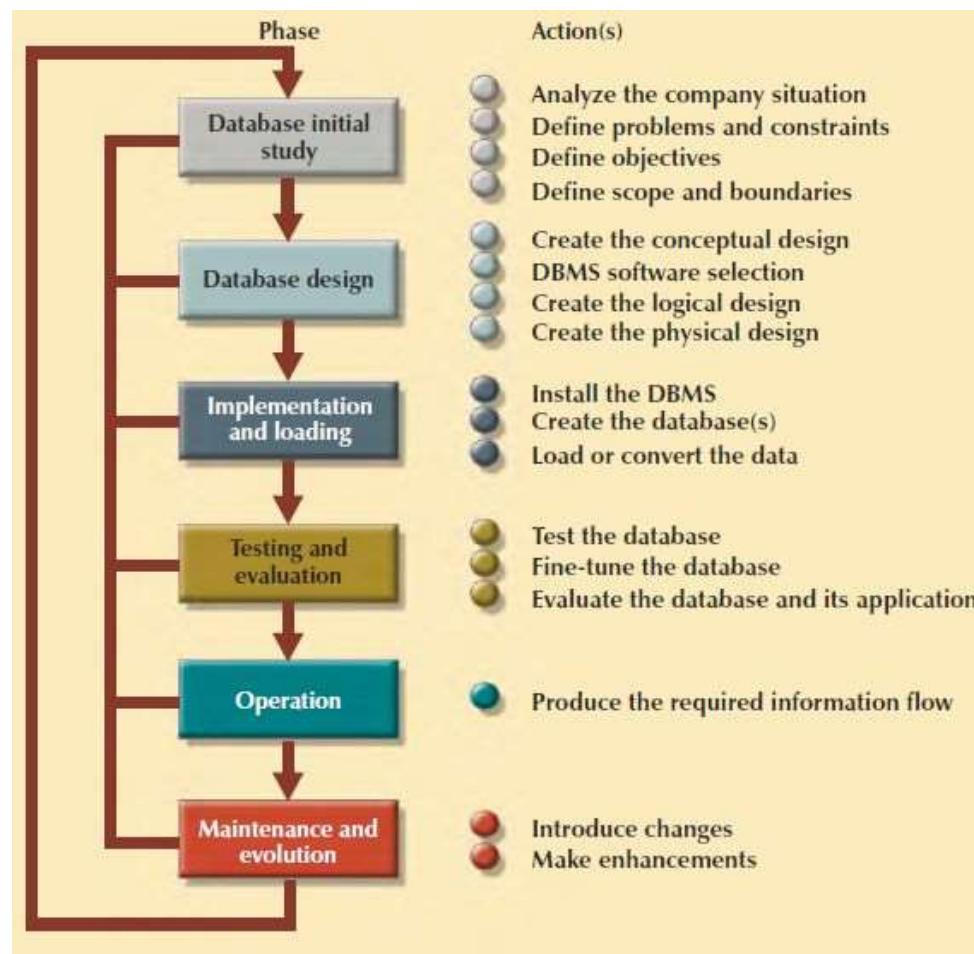
# Implementation

- hardware, software, DBMS and application programs are installed
- database design is implemented
- system enters into cycle of coding, testing, and debugging until it is ready to be delivered
- database contents loaded
  - customized user programs
  - database interface programs.
  - conversion programs
- system is subjected to exhaustive testing until it is ready for use
- final documentation is reviewed and printed
- end users are trained
- system is in full operation
- will be continuously evaluated and fine-tuned

# Maintenance

- almost as soon as system is operational, end users begin to request changes
- corrective maintenance in response to systems errors
- adaptive maintenance due to changes in the business environment
- perfective maintenance to enhance the system

# Database Life Cycle



# Database Life Cycle

- within larger information system, database is subject to a life cycle
- contains six phases:
  - database initial study, database design, implementation and loading, testing and evaluation, operation, and maintenance and evolution

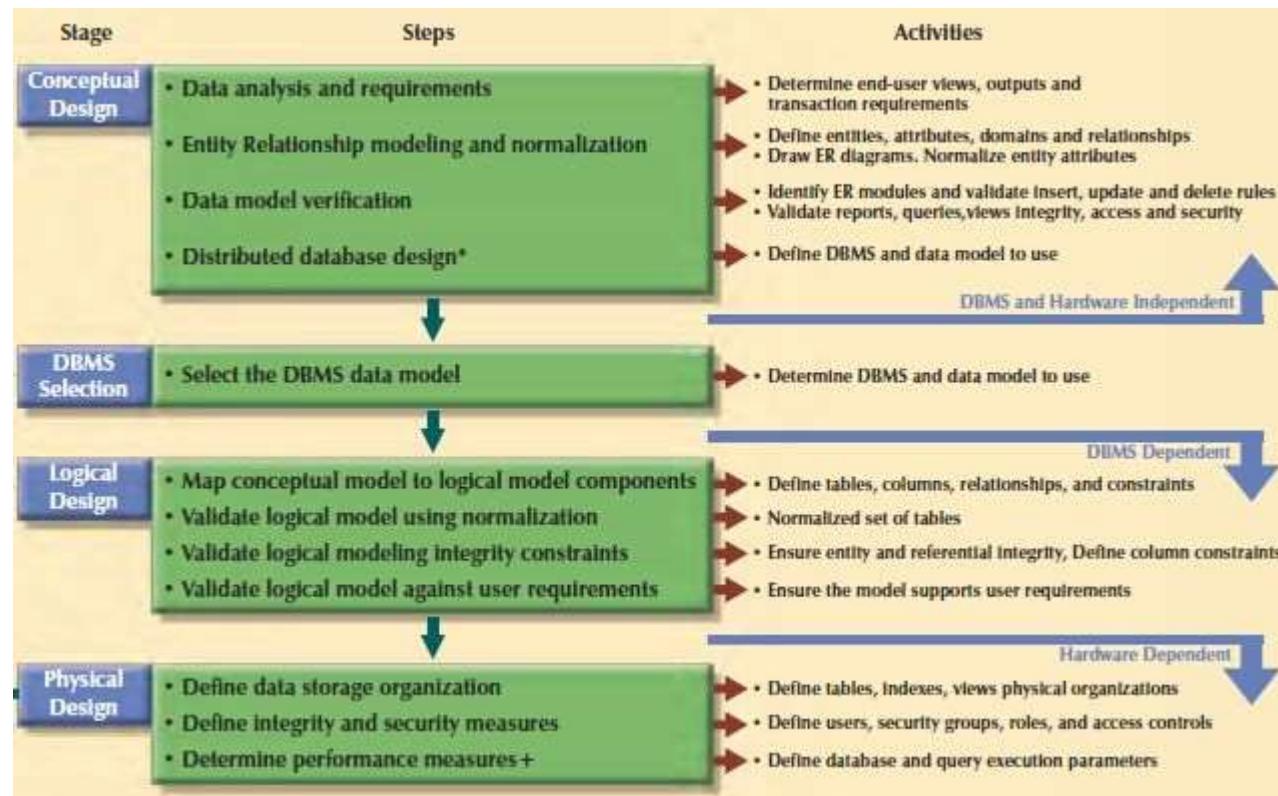
# Database Initial Study

- examine current system's operation
  - determine how and why current system fails
- database design is a technical business,
- it is also people-oriented
- overall purpose of database initial study is to:
  - analyze company situation
  - define problems and constraints
  - define objectives
  - define scope and boundaries

# Database Design

- arguably the most critical DBLC phase
  - making sure that final product meets user and system requirements
- business view of data as a source of information
- designer's view of data structure, its access, and activities required to transform data into information

# Database design process



# Implementation and Loading

- output of database design phase is a series of instructions detailing creation of tables, attributes, domains, views, indexes, security constraints, and storage and performance guidelines
- actually implement all these design specifications
- install DBMS
- create database(s)
- load or convert data
  - typically, data migrated from prior version of system

# Testing and Evaluation

- occurs in conjunction with applications programming
- ensure integrity, security, performance, and recoverability of the database
- evaluate the database and its application programs

# Testing and Evaluation

- physical security allows only authorized personnel physical access to specific areas
- password security allows assignment of access rights to specific authorized users
- access rights can be established through the use of database software
  - restrict operations (CREATE, UPDATE, DELETE, and so on) on predetermined objects such as databases, tables, views, queries, and reports
- audit trails are usually provided by DBMS to check for access violations
- data encryption can be used to render data useless to unauthorized users who might have violated some of database security layers

# Fine-Tune database

- different systems will place different performance requirements on database
- database performance tuning and query optimization

# Common sources of database failure

- Software
  - traceable to operating system, DBMS software, application programs, or viruses
- Hardware
  - may include memory chip errors, disk crashes, bad disk sectors, and “*disk full*” errors
- Programming exemptions
  - application programs or end users may roll back transactions when certain conditions are defined
  - caused by malicious or improperly tested code

# Common sources of database failure

- Transactions
  - system detects deadlocks and aborts one of transactions
- External factors
  - backups are especially important when a system suffers complete destruction from natural disaster

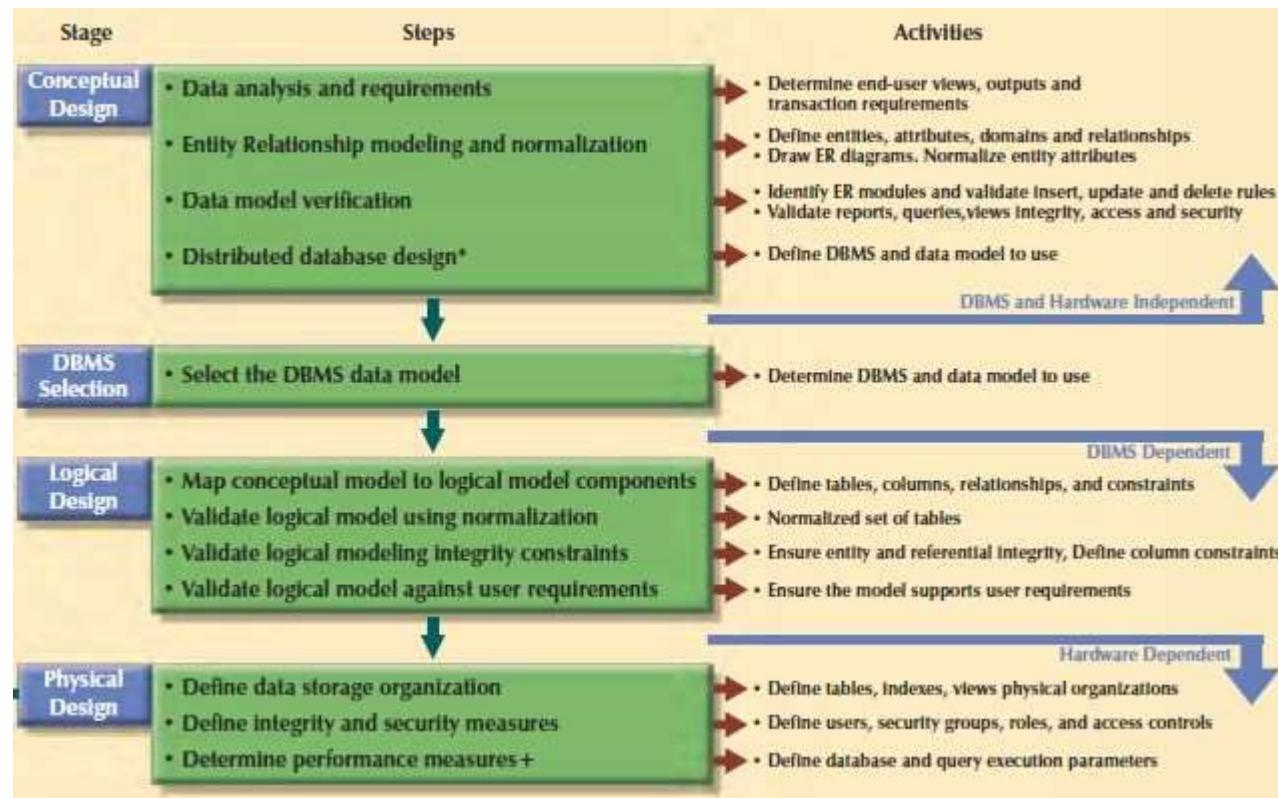
# Operation

- database passed evaluation stage, it is considered to be operational
- database, management, users, application programs constitute a complete information system

# Maintenance and Evolution

- DBA perform routine maintenance activities
- preventive maintenance (backup)
- corrective maintenance (recovery)
- adaptive maintenance
  - enhancing performance, add entities and attributes ...
- assignment of access permissions for new and old users
- generation of database access statistics to improve efficiency of system audits , to monitor system performance
- periodic security audits, system-usage summaries for internal billing or budgeting purposes

# Database design process



# Conceptual design

- design database that is independent of database software and physical details
- output of this process is a conceptual data model that describes the main data entities, attributes, relationships, and constraints of a given problem domain - descriptive and narrative in form
- generally composed of a graphical representation
- textual descriptions of main data elements, relationships, and constraints

# Conceptual design

- Data analysis and requirements
- Entity relationship modeling and normalization
- Data model verification
- Distributed database design

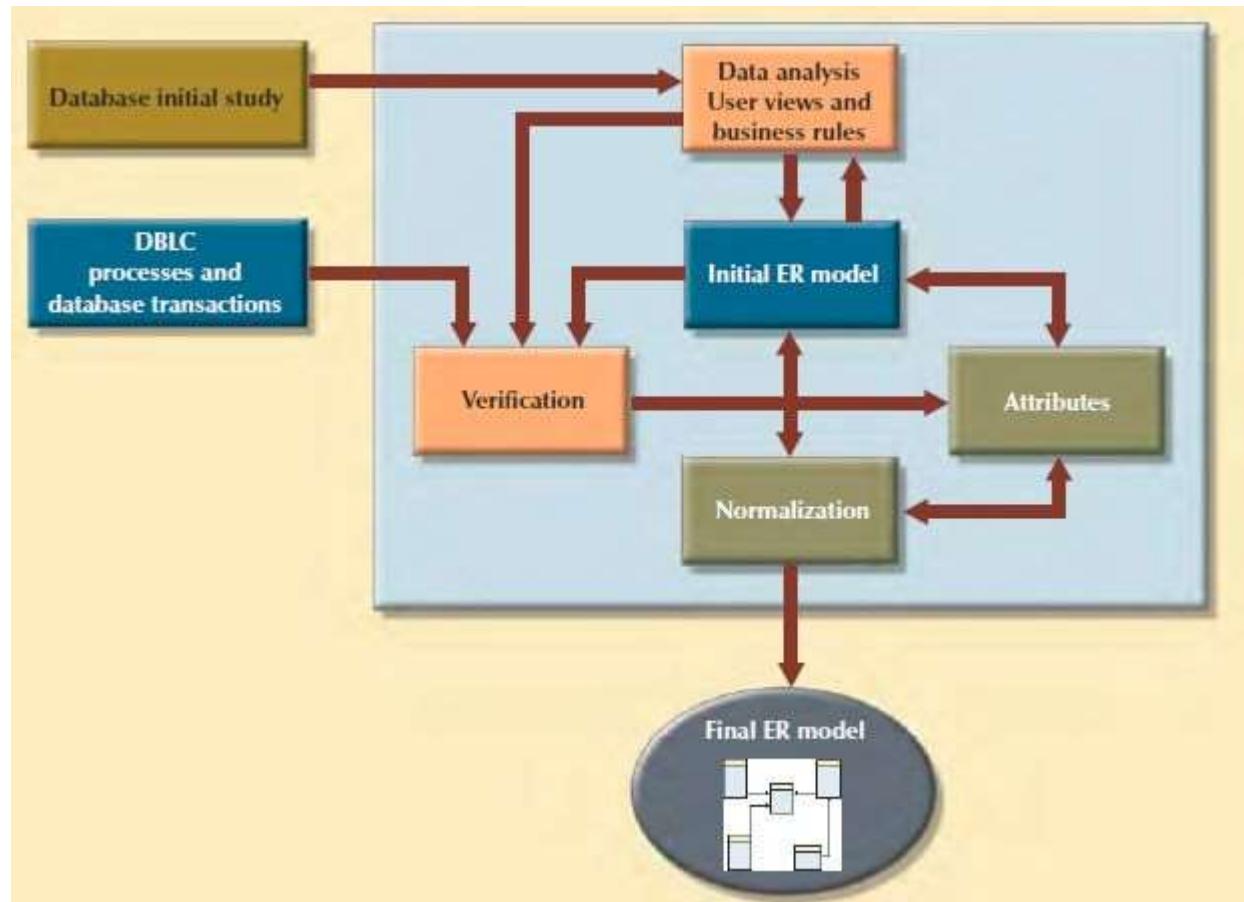
# Data analysis and requirements

- information needs
- information users
- information sources
- information constitution
  - data elements needed to produce information

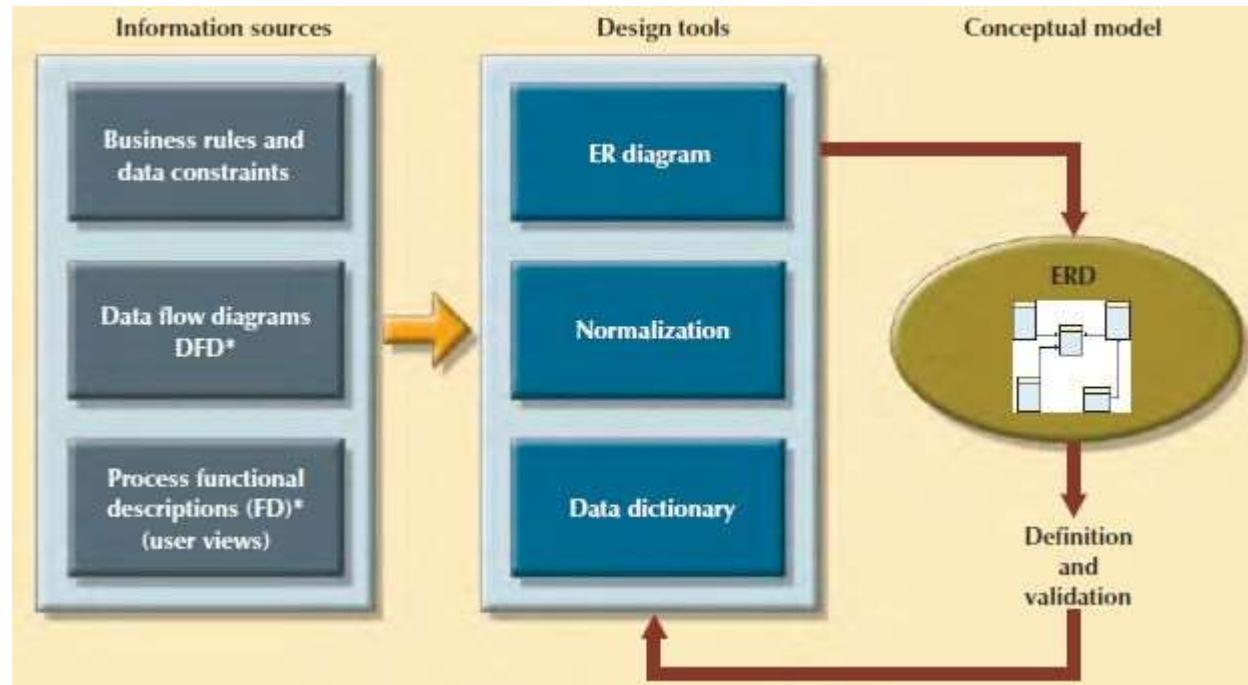
# Entity relationship modeling and normalization

1. identify, analyze, and refine business rules
2. identify main entities
3. define relationships among entities
4. define attributes, primary keys, and foreign keys for each of the entities
5. normalize the entities
6. complete initial ER diagram
7. validate ER model against end users' information and processing requirements
8. modify ER model

# iterative process based on many activities



# information sources



# Data model verification

1. identify the ER model's central entity.
2. identify each module and its components.
3. identify each module's transaction requirements:
  1. Internal: Updates/Inserts/Deletes/Queries/Reports
  2. External: Module interfaces
4. verify all processes against system requirements
5. make all necessary changes suggested
6. repeat Steps 2–5 for all modules

# Distributed database design

- although not a requirement for most databases
- sometimes database may need to be distributed among multiple geographically disperse locations
  - processes that access database may also vary from one location to another
- if database data and processes are to be distributed across system, portions of a database, known as database fragments, may reside in several physical locations
- database fragment
  - subset of database that is stored at given location
  - composed of subset of rows or columns from one or multiple tables

# DBMS Software Selection

- Common factors affecting purchasing decision:
  - cost
  - DBMS features and tools
  - underlying model
  - Portability
  - DBMS hardware requirements

# Logical design

- requires that all objects in conceptual model be mapped to specific constructs used by selected database model
- includes specifications for relations (tables), relationships, and constraints (i.e., domain definitions, data validations, and security views)

# Logical design

1. map conceptual model to logical model components
2. validate logical model using normalization
3. validate logical model integrity constraints
4. validate logical model against user requirements

# Mapping conceptual model to relational model

1. map strong entities
2. map supertype/subtype relationships
3. map weak entities
4. map binary relationships
5. map higher degree relationships

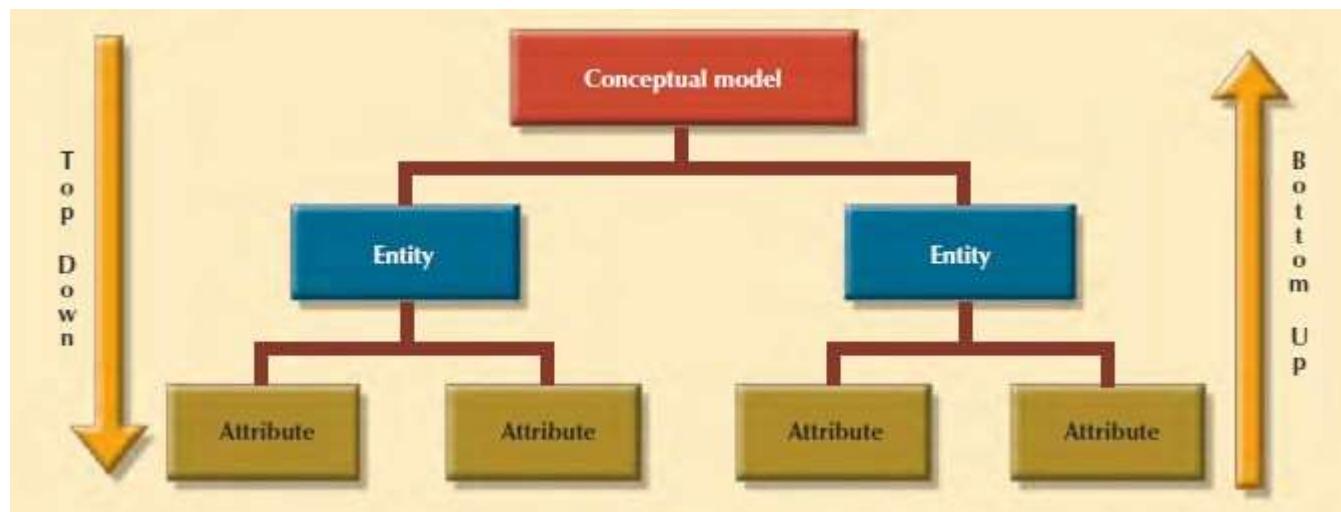
# Physical design

- process of determining data storage organization and data access characteristics
  -
- storage characteristics are function of types of devices supported by hardware, type of data access methods supported by system, and the DBMS
- relational databases are more insulated from physical details than older hierarchical and network models

# Database Design Strategies

- top-down design
  - starts by identifying data sets and then defines data elements for each of those sets
  - involves identification of different entity types and definition of each entity's attributes
- bottom-up design
  - first identifies data elements (items) and then groups them together in data sets
  - first defines attributes, and then groups them to form entities

# Top-down vs. bottom-up design sequencing



# Centralized design

- productive when data component is composed of a relatively small number of objects and procedures
- can be successfully done by single person or by small, informal design team
- operations and scope of problem are sufficiently limited

# Decentralized design

- might be used when data component of the system has a considerable number of entities and complex relations on which very complex operations are performed
- likely to be employed when problem itself is spread across several operational sites and each element is a subset of entire data set

# Aggregation problems

- synonyms and homonyms
  - various departments might know same object by different names (synonyms), or they might use same name to address different objects (homonyms) - object can be entity, attribute, or relationship
- entity and entity subtypes
  - entity subtype might be viewed as separate entity by one or more departments
  - must integrate such subtypes into higher-level entity
- conflicting object definitions
  - attributes can be recorded as different types (character, numeric), or different domains can be defined for same attribute
  - constraint definitions can vary
  - must remove such conflicts from model

# Problems: connection traps

- occur due to misinterpretation of meaning of certain relationships: *fan traps* and *chasm traps*
- to identify connection traps we must ensure that meaning of relationship is fully understood and clearly defined
- if we do not understand relationships we may create model that is not true representation of “real world”

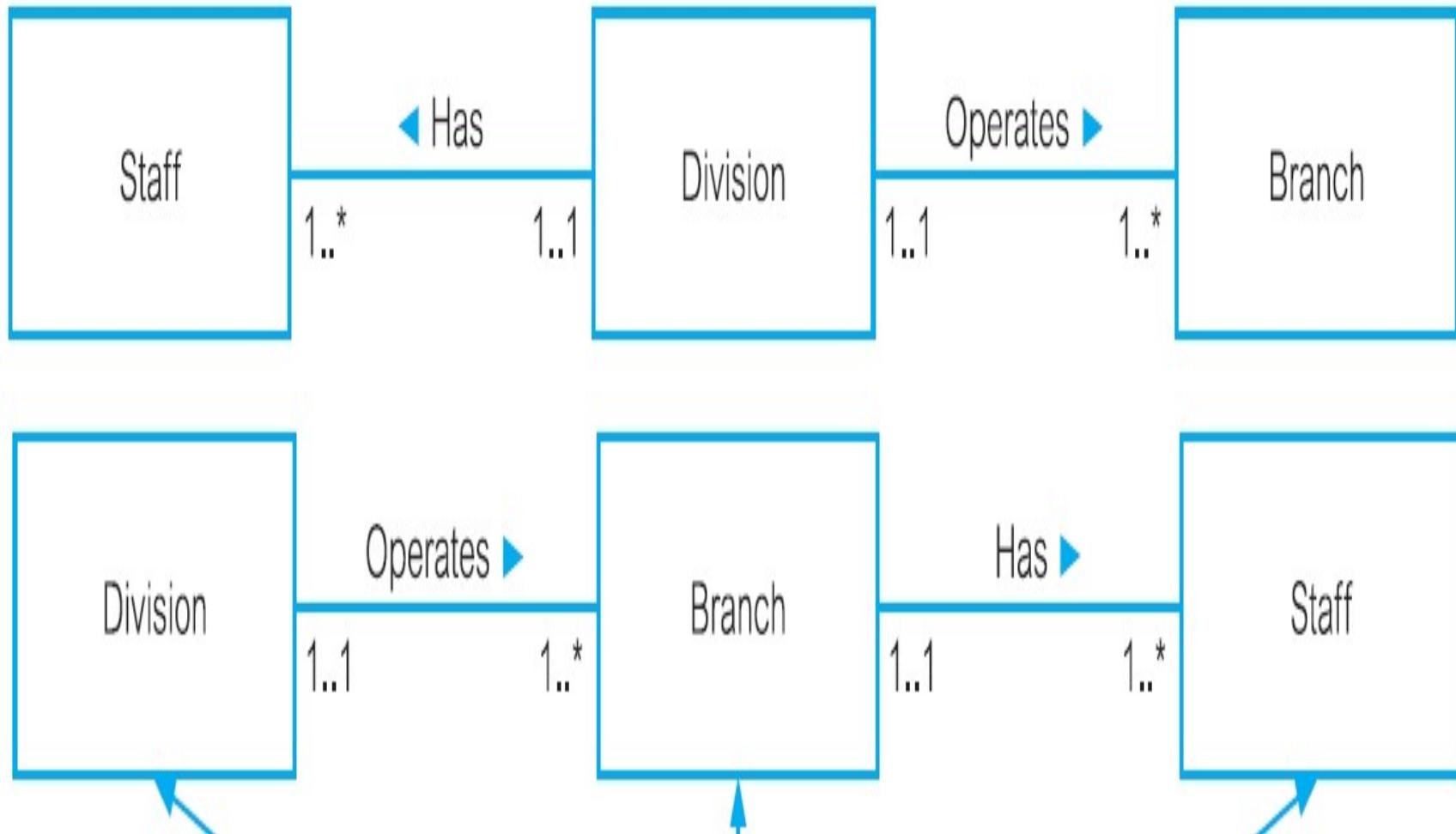
# Fan Trap

- model represents relationship between entity sets, but pathway between certain entity occurrences is ambiguous
- exist where two or more 1:m relationships fan out from same entity

# Fan Trap

- relationships (1:m) *Has* and *Operates* emanating from same entity *Division*
- a single division operates *one or more* branches and has *one or more* staff
- problem arises when we want to know which members of staff work at particular branch

# Example of Fan Trap



# Chasm Trap

- model suggests existence of relationship between entity sets, but pathway does not exist between certain entity occurrences

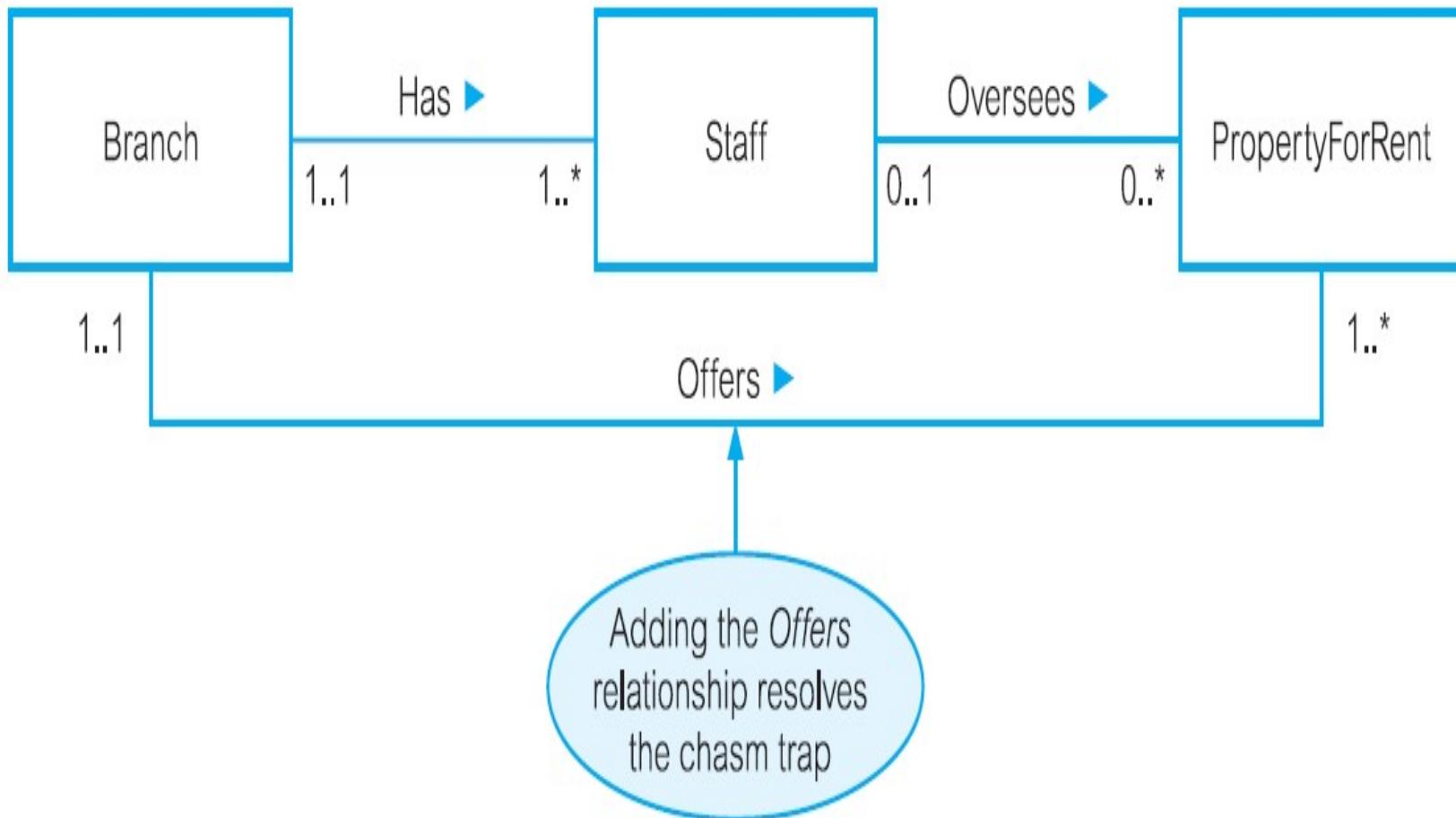
# Chasm Trap

- occur where there are one or more relationships with minimum multiplicity of zero (that is, optional participation) forming part of pathway between related entities

# Chasm Trap

- potential chasm trap illustrated in relationships between *Branch*, *Staff*, and *PropertyForRent* entities; model represents facts that single branch has *one or more* staff who oversee *zero or more* properties for rent; not all staff oversee property and not all properties are overseen
- problem arises when we want to know which properties are available at each branch

# Example of Chasm Trap



# **SPECIALIZATION / GENERALIZATION**

# Specialization/Generalization

- associated with special types of entities known as superclasses and subclasses, and process of attribute inheritance
- hierarchy of entities containing superclasses and subclasses

# Superclass

- entity set that includes one or more distinct subgroupings of its occurrences, which must be represented in data model

# Subclass

- distinct subgrouping of occurrences of entity set, which must be represented in data model

# for example

- entities that are members of *Staff* entity set may be classified as *Manager*, *SalesPersonnel*, and *Secretary*
- *Staff* entity is referred to as superclass of
- *Manager*, *SalesPersonnel*, and *Secretary* subclasses
- superclass/subclass relationship

# Superclass/Subclass Relationships

- each member of subclass is also member of superclass; but has distinct role
- relationship between superclass and subclass is one-to-one (1:1)

# Superclass/Subclass Relationships

- some superclasses may contain overlapping subclasses (illustrated by member of *Staff* who is both *Manager* and *Sales Personnel*)
- not every member of superclass is necessarily member of subclass (members of *Staff* without distinct job role such as *Manager* or *Sales Personnel*)

# Possible (but not indicated)

- All Up – use superclass without any subclasses
- All Down – use many subclasses without superclass
- use superclasses and subclasses to avoid describing different types of *Staff* with possibly different attributes within single entity

- avoids describing similar concepts more than once - saving time for designer
- adds more semantic information to design in form that is familiar to many
- “*Manager IS-A member of Staff*” - communicates significant semantic content in concise form

- for example, *Sales Personnel* may have special attributes such as *salesArea* and *carAllowance*
- if all are described by single *Staff* entity, this may result in a lot of nulls for job-specific attributes

- can also show relationships that are associated with only particular types of staff (subclasses) and not with staff, in general;
- for example, *Sales Personnel* may have distinct relationships that are not appropriate for all *Staff*, such as *SalesPersonnel Uses Car*

- *Sales Personnel* have common attributes with other *Staff*, such as *staffNo*, *name*, *position*, and *salary*

# relation AllStaff

The diagram illustrates the decomposition of the **AllStaff** relation into four smaller relations based on staff roles. The original relation has 9 columns, and each role-specific relation has 8 columns, sharing the first column with the original relation.

**Original Relation:**

| staffNo | name          | position       | salary | mgrStartDate | bonus | sales Area | car Allowance | typing Speed |
|---------|---------------|----------------|--------|--------------|-------|------------|---------------|--------------|
| SL21    | John White    | Manager        | 30000  | 01/02/95     | 2000  |            |               |              |
| SG37    | Ann Beech     | Assistant      | 12000  |              |       |            |               |              |
| SG66    | Mary Martinez | Sales Manager  | 27000  |              |       | SA1A       | 5000          |              |
| SA9     | Mary Howe     | Assistant      | 9000   |              |       |            |               |              |
| SL89    | Stuart Stern  | Secretary      | 8500   |              |       |            |               | 100          |
| SL31    | Robert Chin   | Snr Sales Asst | 17000  |              |       | SA2B       | 3700          |              |
| SG5     | Susan Brand   | Manager        | 24000  | 01/06/91     | 2350  |            |               |              |

**Decomposed Relations:**

- Attributes appropriate for all staff:** staffNo, name, position, salary
- Attributes appropriate for branch Managers:** staffNo, name, position, salary, mgrStartDate, bonus
- Attributes appropriate for Sales Personnel:** staffNo, name, position, salary, sales Area, car Allowance
- Attribute appropriate for Secretarial staff:** staffNo, name, position, salary, typing Speed

# Attribute Inheritance

- subclass represents same “real world” object as in superclass, may possess subclass-specific attributes, as well as those associated with superclass
- *SalesPersonnel* subclass inherits all attributes of *Staff* superclass - *staffNo*, *name*, *position*, and *salary* together with those specifically associated with *SalesPersonnel* subclass - *salesArea* and *carAllowance*

# Hierarchy

- subclass is an entity in its own right and so it may also have one or more subclasses
- Specialization hierarchy (for example, *Manager* is a specialization of *Staff*)
- Generalization hierarchy (for example, *Staff* is a generalization of *Manager*)
- IS-A hierarchy (for example, *Manager* IS-A (member of) *Staff*)

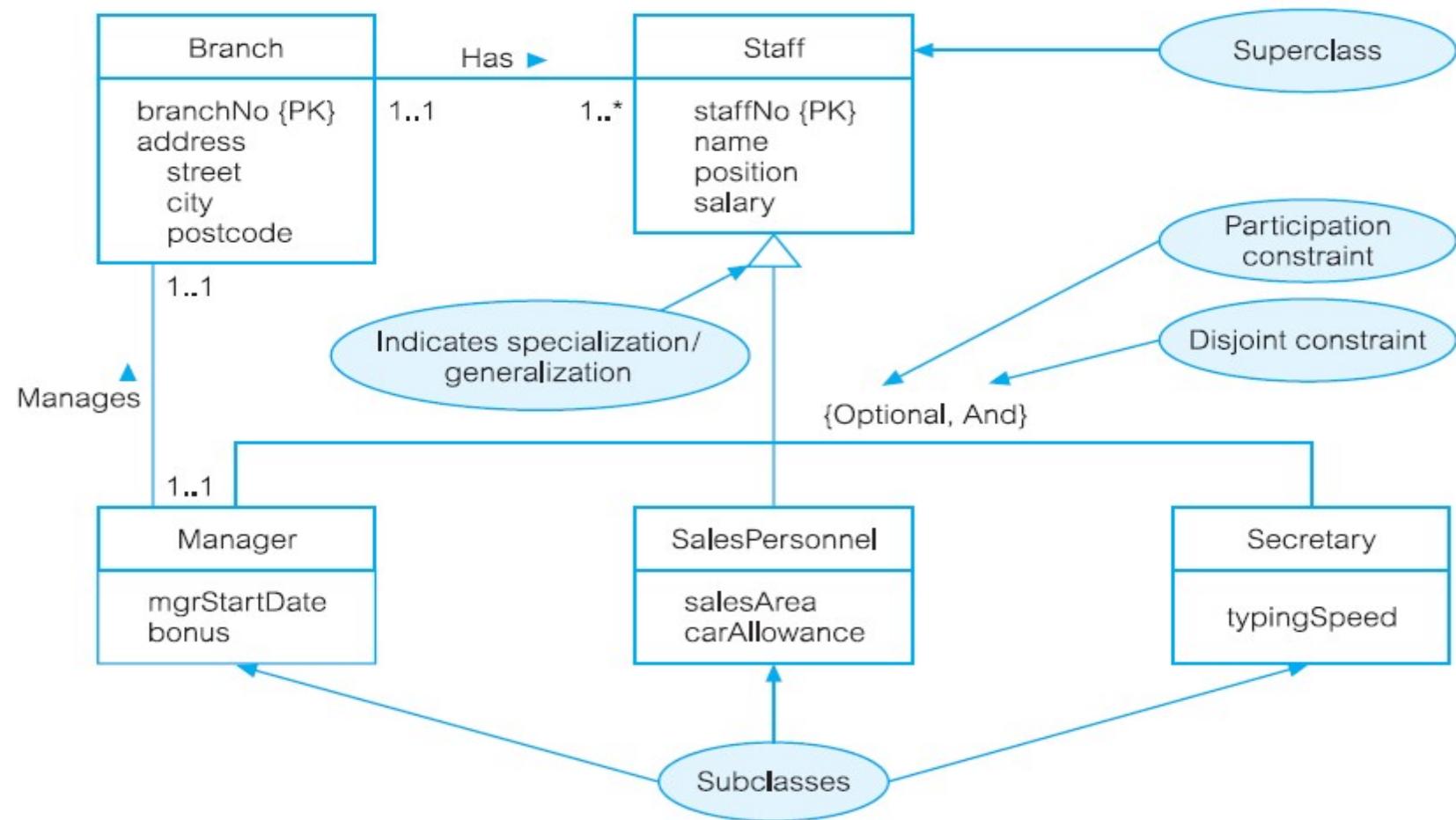
# Specialization process

- process of maximizing differences between members of an entity by identifying their distinguishing characteristics
- top-down approach to defining set of superclasses and their related subclasses

# Generalization process

- process of minimizing differences between entities by identifying their common characteristics
- bottom-up approach, that results in identification of generalized superclass from original entity types

# Specialization/generalization of Staff subclasses representing job roles

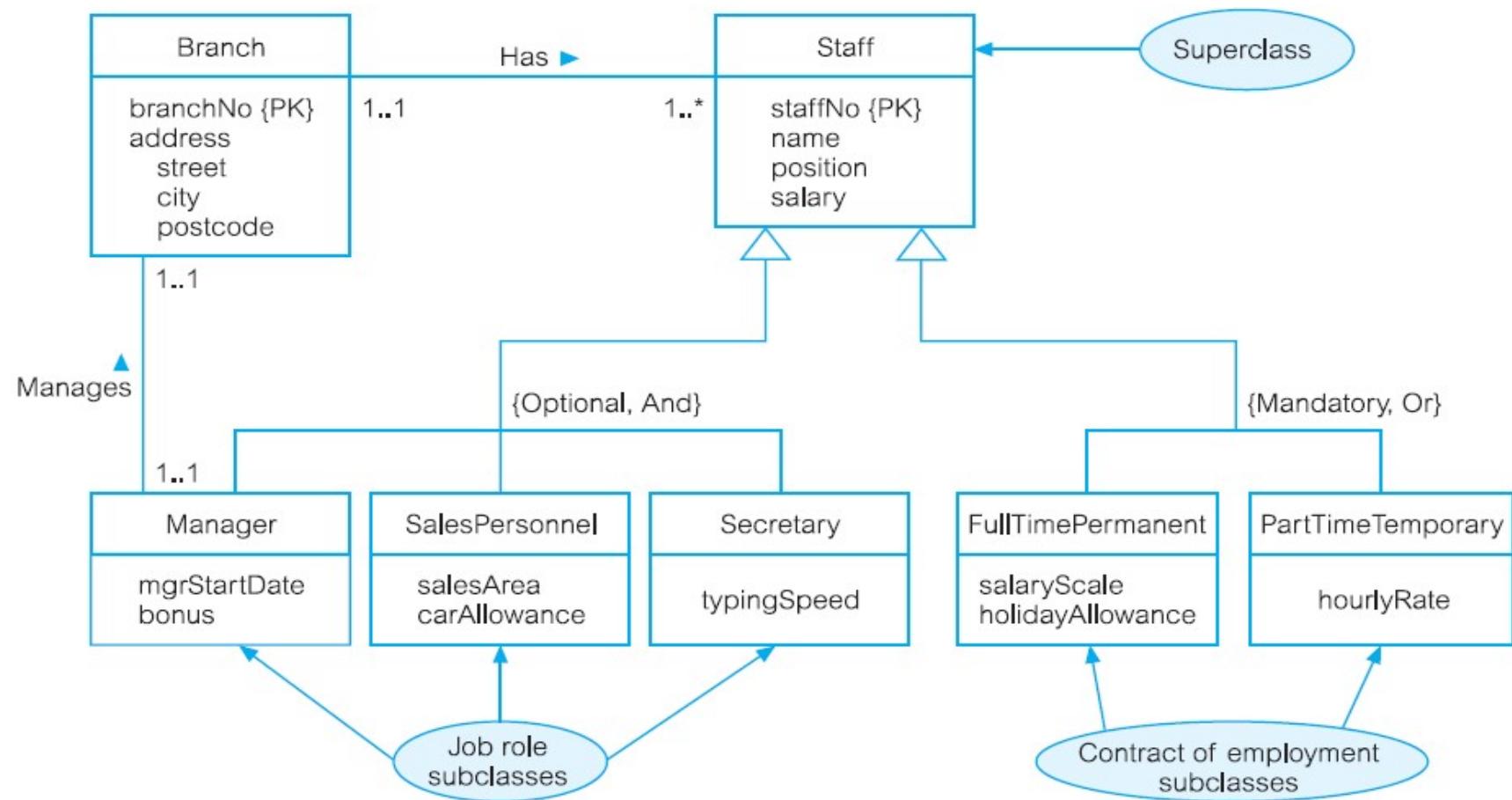


- may have several specializations of same entity based on different distinguishing characteristics

# for example

- another specialization of *Staff* entity may produce subclasses *FullTimePermanent* and *PartTimeTemporary*, which distinguishes between types of employment contract for members of staff
- attributes that are specific to *FullTimePermanent* (*salaryScale* and *holidayAllowance*) and *PartTimeTemporary* (*hourlyRate*)

# Staff entity: job roles & contracts of employment



# Constraints on Specialization/Generalization

- two constraints that may apply to specialization/generalization called **participation constraints** and **disjoint constraints**

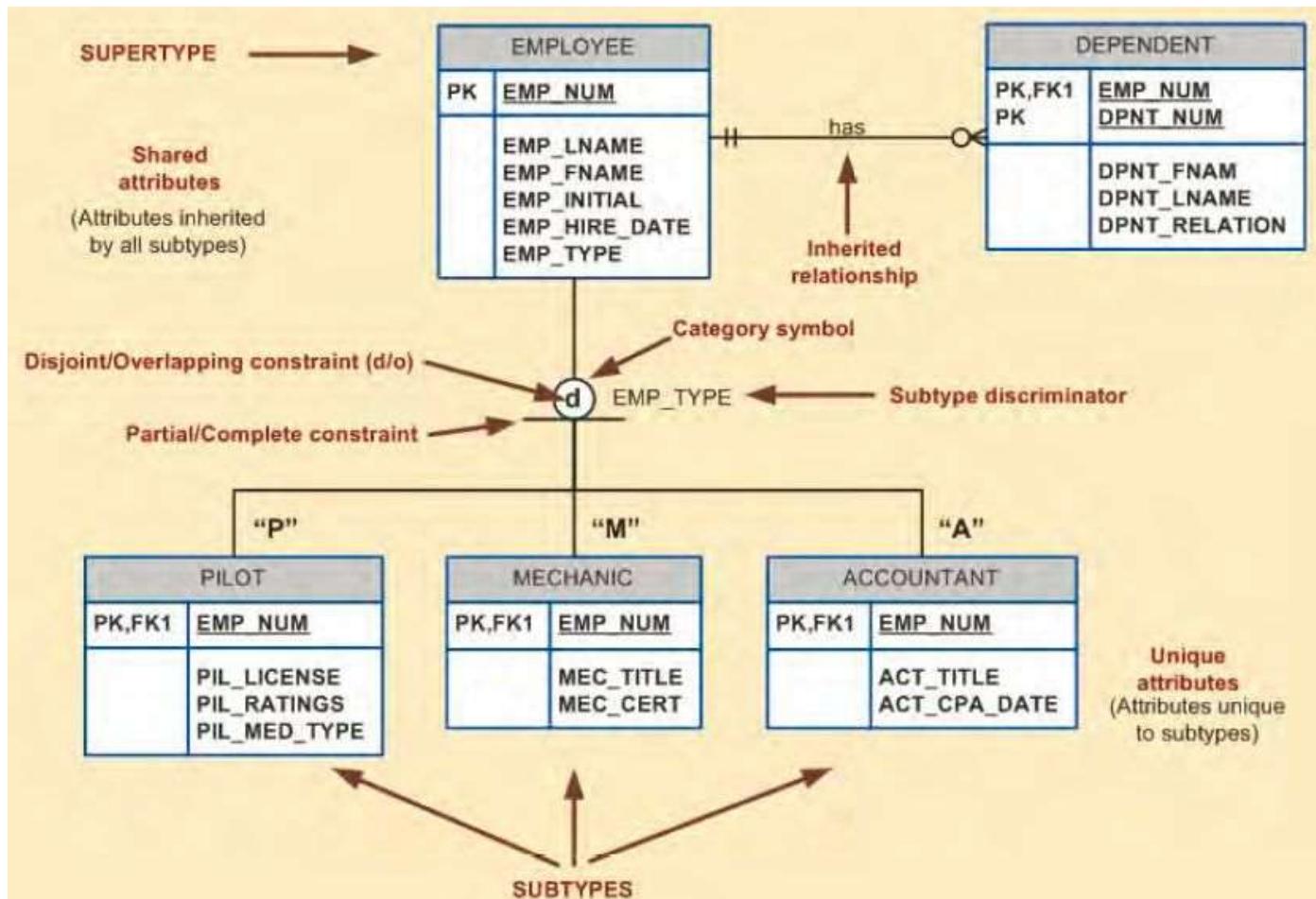
# Participation constraints

- determines whether every member in superclass must participate as member of subclass
- may be mandatory or optional
- mandatory participation specifies that every member in superclass must also be member of a subclass

# Disjoint constraints

- describes relationship between members of subclasses and indicates whether it is possible for member of superclass to be member of one, or more than one, subclass - applies when superclass has more than one subclass
- if subclasses are disjoint, then an entity occurrence can be member of only one of subclasses

# specialization hierarchy



# case of an aviation business

- employs pilots, mechanics, secretaries, accountants, database managers, and many others
- pilots share certain characteristics with other employees
  - such as last name (EMP\_LNAME) and hire date (EMP\_HIRE\_DATE)
- unlike other employees, pilots must meet special requirements such as flight hour restrictions, flight checks, and periodic training
- if all employee characteristics and special qualifications were stored in a single EMPLOYEE entity, you would have a lot of nulls

- based on hierarchy conclude that PILOT is subtype of EMPLOYEE, and that EMPLOYEE is supertype of PILOT
- entity supertype is generic (more general) entity type that is related to one or more specific (specialized) entity subtypes
- entity supertype contains common characteristics
- entity subtypes contain unique characteristics of each entity subtype
- There must be different, identifiable kinds or types of the entity in the user's environment.
- The different kinds or types of instances should have  
DataBase Course Notes 07 -  
Information 262  
one or more attributes that are unique to that kind or type

- pilots meet both criteria of being an identifiable kind of employee and having unique attributes that other employees do not possess, it is appropriate to create PILOT as a subtype of EMPLOYEE
- CLERK would not be an acceptable subtype of EMPLOYEE because it only satisfies one of the criteria—it is an identifiable kind of employee—but there are not any attributes that are unique to just clerks

# Specialization Hierarchy

- Entity supertypes and subtypes are organized in a *specialization hierarchy*
- specialization hierarchy formed by an EMPLOYEE supertype and three entity subtypes—PILOT, MECHANIC, and ACCOUNTANT
- 1:1 relationship between EMPLOYEE and its subtypes
  - for example PILOT subtype occurrence is related to one instance of the EMPLOYEE

- relationships depicted within specialization hierarchy are sometimes described in terms of “is-a” relationships
- for example pilot is an employee
- subtype can exist only within the context of a supertype
- subtype can have only one supertype to which it is directly related
- specialization hierarchy can have many levels of supertype/subtype relationships

# Inheritance

- enables entity subtype to inherit attributes and relationships of supertype
- pilots, mechanics, and accountants all inherit employee number, last name, first name, middle initial, hire date, and so on from EMPLOYEE entity
- pilots have attributes that are unique; same is true for mechanics and accountants
- all entity subtypes inherit their primary key attribute from their supertype
  - EMP\_NUM attribute is the primary key for each of the subtypes

# EMPLOYEE-PILOT

## supertype-subtype relationship

Table Name: EMPLOYEE

| EMP_NUM | EMP_LNAME  | EMP_FNAME | EMP_INITIAL | EMP_HIRE_DATE | EMP_TYPE |
|---------|------------|-----------|-------------|---------------|----------|
| 100     | Kosmycz    | Xavier    | T           | 15-Mar-88     |          |
| 101     | Lewis      | Marcos    |             | 25-Apr-89     | P        |
| 102     | Vandam     | Jean      |             | 20-Dec-93     | A        |
| 103     | Jones      | Victoria  | R           | 28-Aug-03     |          |
| 104     | Lange      | Edith     |             | 20-Oct-97     | P        |
| 105     | Williams   | Gabriel   | U           | 08-Nov-97     | P        |
| 106     | Duzak      | Mario     |             | 05-Jan-04     | P        |
| 107     | Drante     | Venite    | L           | 02-Jul-97     | M        |
| 108     | Wiesenbach | Joni      |             | 18-Nov-95     | M        |
| 109     | Travis     | Brett     | T           | 14-Apr-01     | P        |
| 110     | Oenlazi    | Stan      |             | 01-Dec-03     | A        |

Table Name: PILOT

| EMP_NUM | PIL_LICENSE | PIL_RATINGS          | PIL_MED_TYPE |
|---------|-------------|----------------------|--------------|
| 101     | ATP         | SEL/MEL/Inst/CFI     | 1            |
| 104     | ATP         | SEL/MEL/Inst         | 1            |
| 105     | COM         | SEL/MEL/Inst/CFI     | 2            |
| 106     | COM         | SEL/MEL/Inst         | 2            |
| 109     | COM         | SEL/MEL/SES/Inst/CFI | 1            |

# Subtype Discriminator

- attribute in supertype entity that determines to which subtype the supertype occurrence is related
- common practice to show subtype discriminator and its value for each subtype in the ER diagram

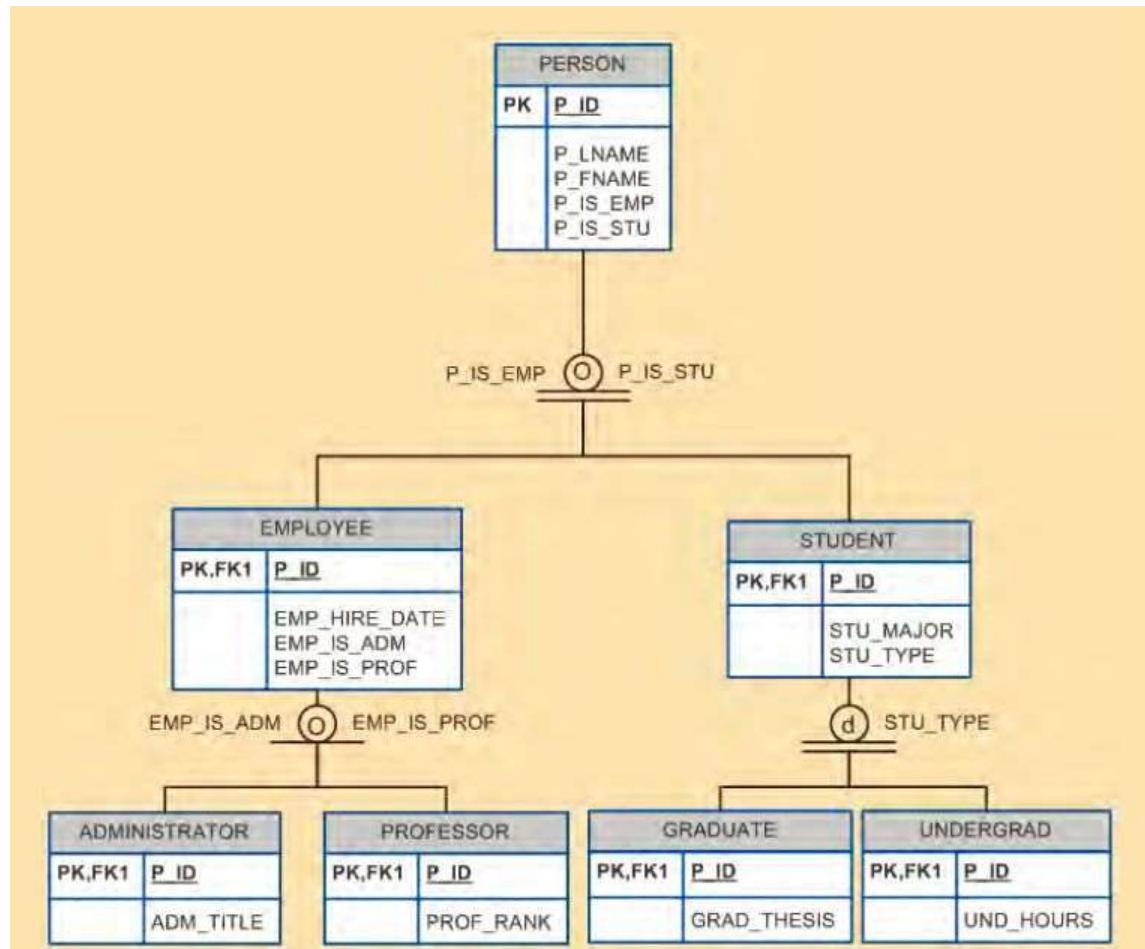
# Disjoint or overlapping entity subtypes

- Employee can be a pilot or a mechanic or an accountant
- business rules dictates that employee cannot belong to more than one subtype at a time
- Disjoint subtypes, also known as non-overlapping subtypes, are subtypes that contain unique subset of the supertype entity set

# Overlapping subtypes

- if business rule specifies that employees can have multiple classifications, EMPLOYEE supertype may contain overlapping job classification subtypes
- contain non-unique subsets of the supertype entity set
- for example, in university environment, person may be an employee or student or both

# Specialization hierarchy with overlapping subtypes



# Completeness constraint

- specifies whether each entity supertype occurrence must also be a member of at least one subtype
- partial completeness means that not every supertype occurrence is a member of a subtype
  - there may be some supertype occurrences that are not members of any subtype
- total completeness means that every supertype occurrence must be a member of at least one subtype

# Specialization Hierarchy Constraint Scenarios

| TYPE                                                                                         | DISJOINT CONSTRAINT                                                                                                                      | OVERLAPPING CONSTRAINT                                                                                                                        |
|----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Partial<br> | Supertype has optional subtypes.<br>Subtype discriminator can be null.<br>Subtype sets are unique.                                       | Supertype has optional subtypes.<br>Subtype discriminators can be null.<br>Subtype sets are not unique.                                       |
| Total<br> | Every supertype occurrence is a member of a (at least one) subtype.<br>Subtype discriminator cannot be null.<br>Subtype sets are unique. | Every supertype occurrence is a member of a (at least one) subtype.<br>Subtype discriminators cannot be null.<br>Subtype sets are not unique. |

# Specialization

- top-down process of identifying lower-level, more specific entity subtypes from a higher-level entity supertype
- based on grouping unique characteristics and relationships of subtypes

# Generalization

- bottom-up process of identifying higher-level, more generic entity supertype from lower-level entity subtypes
- based on grouping common characteristics and relationships of subtypes

# Entity Cluster

- is a “virtual” entity type used to represent multiple entities and relationships
- formed by combining multiple interrelated entities into a single abstract entity object with the purpose of simplifying ER Diagram and thus enhancing its readability

# Primary Key

- most important characteristic of entity is its primary key
- primary key's function is to guarantee entity integrity
- primary keys and foreign keys work together to implement relationships in relational model
- primary key has direct bearing on efficiency and effectiveness of database implementation

# Natural Key

- Natural Key or natural identifier is a real-world, generally accepted identifier used to distinguish — that is, uniquely identify — real-world objects
- familiar to end users and forms part of their day-to-day business vocabulary

- function of the primary key is to guarantee entity integrity, not to “describe” the entity

- primary keys and foreign keys are used to implement relationships among entities
- implementation of such relationships is done mostly behind the scenes, hidden from end users
- database applications should let end user choose among multiple descriptive narratives of different objects, while using primary key values behind the scenes

# Desirable Primary Key Characteristics

- **Unique values** - must uniquely identify each entity instance; it cannot contain nulls
- **Non-intelligent** - 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 an identifier
- for example, student ID of 650973 would be preferred over Doe, John C. as primary key identifier

# Desirable Primary Key Characteristics

- **No change over time** - if attribute has semantic meaning, it might be subject to updates; names do not make good primary keys
  - if you have Jane Doe as PK what happens if she changes her name when she gets married?
- PK should be permanent and unchangeable
- **Preferably single-attribute** - should have the minimum number of attributes; desirable but not required; simplify implementation of FK; multiple-attribute PK can cause PK of related entities to grow

# Desirable Primary Key Characteristics

- **Preferably numeric** - unique values can be better managed when they are numeric, because database can use internal routines to implement automatically increments values with the addition of each new row
- **Security-compliant** - selected primary key must not be composed of any attribute(s) that might be considered security risk or violation
  - using Social Security number as PK in EMPLOYEE table is not a good idea

# composite primary keys

- are particularly useful in two cases:
  1. as identifiers of composite entities, where each primary key combination is allowed only once in the M:N relationship
  2. as identifiers of weak entities, where the weak entity has a strong identifying relationship with the parent entity

# surrogate key

- instances when primary key doesn't exist in real world or when existing natural key might not be suitable
- standard practice to create surrogate key - primary key created by database designer to simplify identification of entity instances
- has no meaning in user's environment — it exists only to distinguish one entity instance from another
- it can be generated by the DBMS to ensure that unique values are always provided

# Database Design

- Data modeling and database design require skills that are acquired through experience
- experience is acquired through practice, regular and frequent repetition, applying the concepts learned to specific and different design problems
- presents four special design cases that highlight importance of flexible designs, proper identification of primary keys, and placement of foreign keys

- DESIGN CASES
- LEARNING FLEXIBLE DATABASE DESIGN
- Design Case #1: Implementing 1:1 Relationships
- Design Case #2: Maintaining History of Time-Variant Data
- Design Case #3: Fan Traps
- Design Case #4: Redundant Relationships

# Design Case #1: Implementing 1:1 Relationships

- Foreign Keys work with Primary Keys to implement relationships:
  - put PK of “one” side (parent entity)
  - put FK on “many” side (dependent entity)
- case of a 1:1 relationship between EMPLOYEE and DEPARTMENT based on business rule “one EMPLOYEE is the manager of one DEPARTMENT, and one DEPARTMENT is managed by one EMPLOYEE.”

- Place FK in both entities - is not recommended, as it would create duplicated work, and it could conflict with other existing relationships
- Place FK in one of entities (PK of one of two entities appears as FK in other entity)
  - preferred solution
  - but there is remaining question: which entity should be used

- One side is mandatory and other side is optional
  - Place PK of entity on mandatory side in entity on optional side as FK, and make FK mandatory
- Both sides are optional
  - Select FK that causes fewest nulls, or place FK in entity in which the (relationship) role is played
- Both sides are mandatory
  - Like previous
  - or consider revising your model to ensure that the two entities do not belong together in a single entity

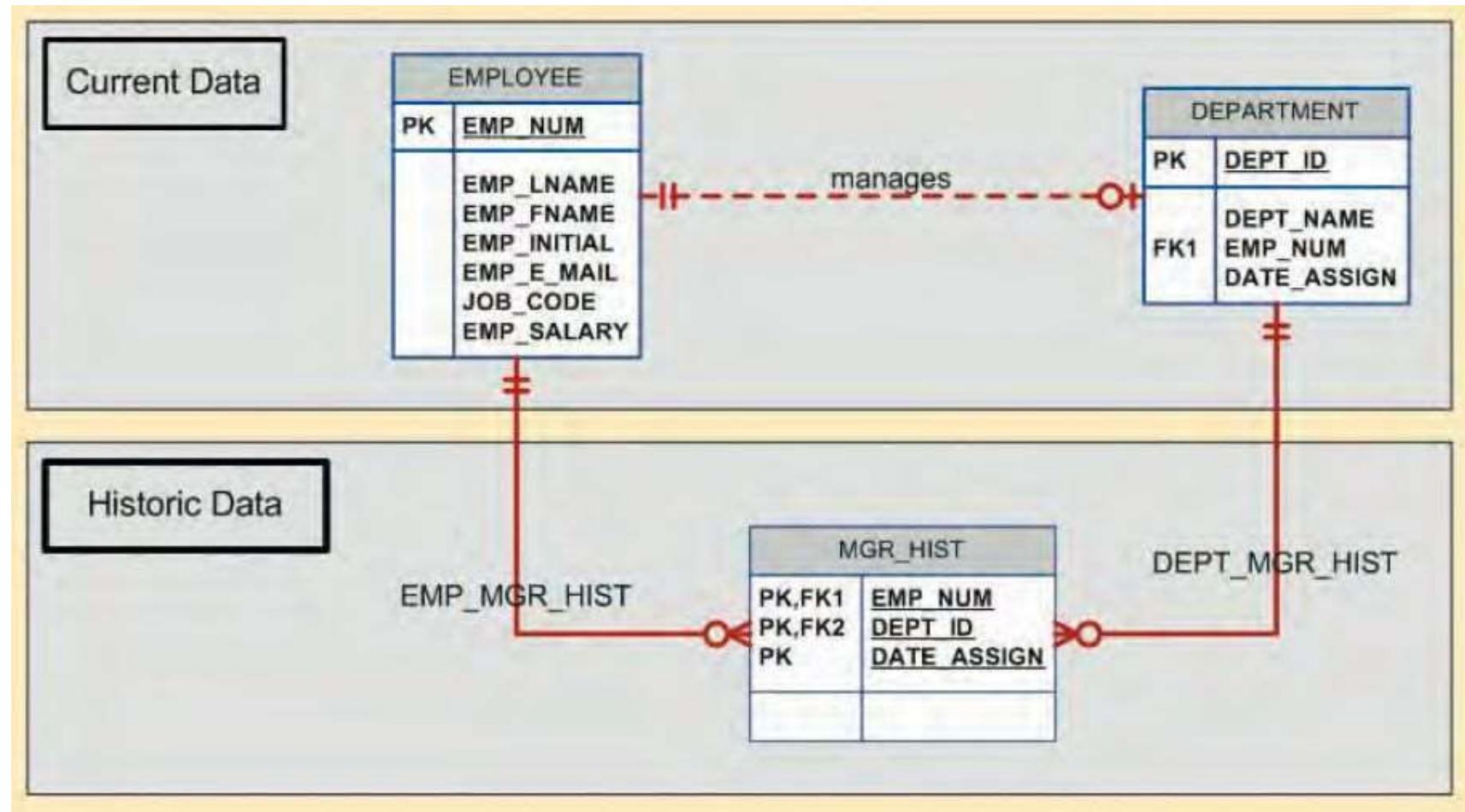
- As a designer, you must recognize that 1:1 relationships exist in the real world, and therefore, they should be supported in the data model.
- In fact, a 1:1 relationship is used to ensure that two entity sets are not placed in the same table.

# Design Case #2: Maintaining History of Time-Variant

- Company managers generally realize that good decision making is based on information that is generated through data stored in databases. Such data reflect current as well as past events.
- Normally, data changes are managed by replacing existing attribute value with new value
- However, there are situations in which history of values for a given attribute must be preserved.
- *time-variant data* refer to data whose values change over time and for which you must keep history of the data changes

- some attribute values, such as your date of birth or your Social Security number, are not time variant
- attributes such as your student GPA or your bank account balance are subject to change over time
- sometimes data changes are externally originated and event driven, such as product price change

- keeping history of time-variant data you must create a new entity in 1:M relationship with original entity; this new entity will contain new value, date of change
- for example, if you want to keep track of current manager as well as history of all department managers, you can create model



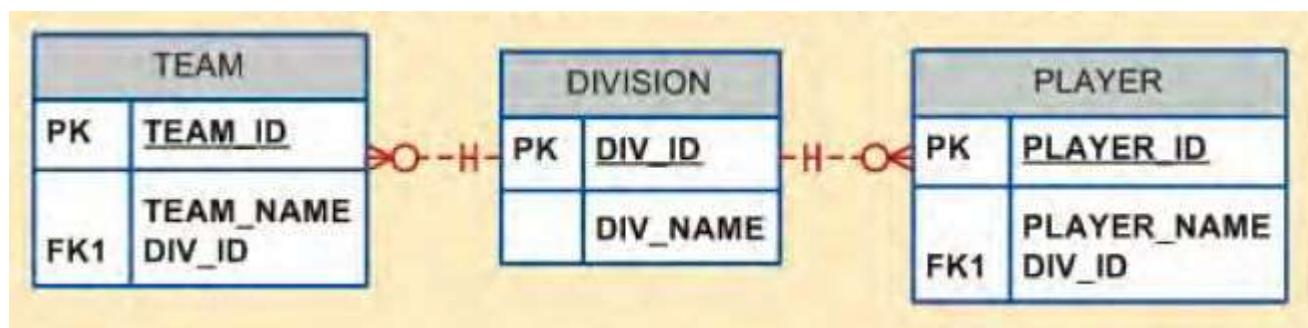
- note that “manages” relationship is optional in theory and redundant in practice
- at any time, you could find out who manager of department is by retrieving the most recent **DATE\_ASSIGN** date from **MGR\_HIST**
- differentiates between current data and historic data
- current manager relationship is implemented by “manages” relationship between **EMPLOYEE** and **DEPARTMENT**
- historic data are managed through **EMP\_MGR\_HIST** and **DEPT\_MGR\_HIST**

# Design Case #3: Fan Traps

- Creating data model requires proper identification of data relationships among entities. However, due to miscommunication or incomplete understanding of business rules or processes, it is not uncommon to misidentify relationships among entities
- design trap occurs when relationship is improperly or incompletely identified and is therefore represented in a way that is not consistent with real world
- most common design trap is known as fan trap

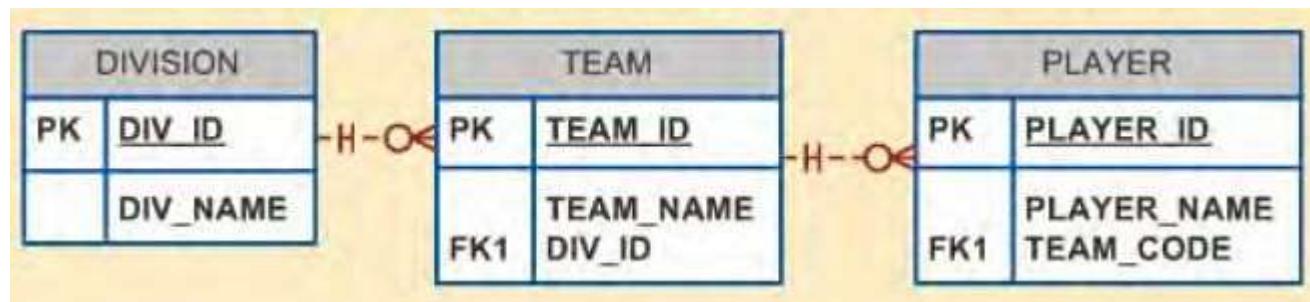
- *fan trap* occurs when you have one entity in two 1:M relationships to other entities, thus producing an association among the other entities that is not expressed in the model
- for example, assume the league has many divisions; each division has many players; each division has many teams
- might create

# Incorrect ER Diagram with fan trap problem



- DIVISION is in 1:M relationship with TEAM and in 1:M relationship with PLAYER
- representation is semantically correct, relationships are not properly identified
- there is no way to identify which players belong to which team
- note that the relationship lines for DIVISION instances fan out to TEAM and PLAYER entity instances — thus “fan trap” label

# Corrected ER Diagram after removal of the fan trap

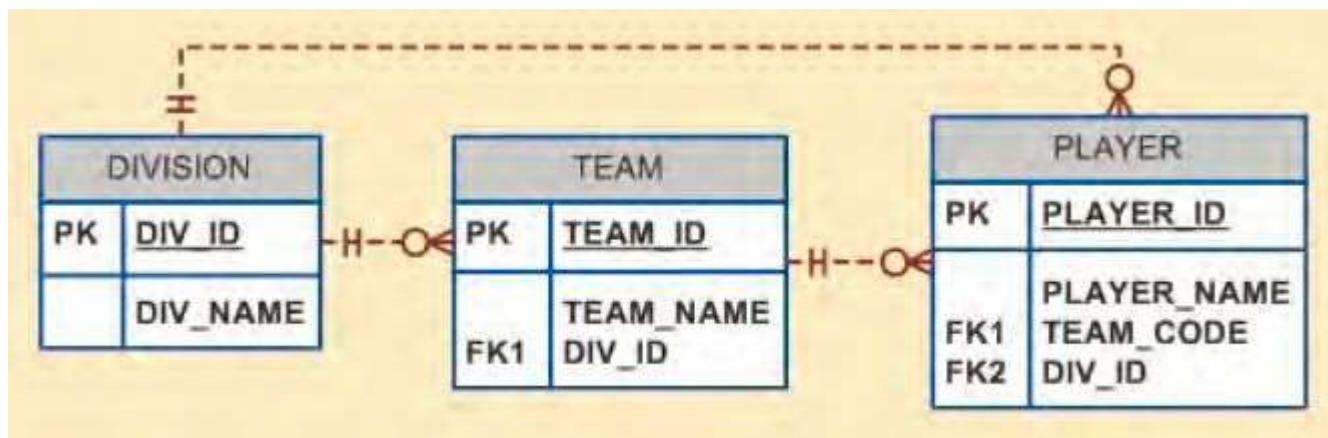


# Design Case #4: Redundant Relationships

- Although redundancy is often a good thing to have in computer environments (multiple backups in multiple places, for example), redundancy is seldom a good thing in the database environment
- Redundant relationships occur when there are multiple relationship paths between related entities
- main concern with redundant relationships is that they remain consistent across the model
- some designs use redundant relationships as a<sup>302</sup>  
way to simplify the design  
Database Course Notes 07-18  
Normalization

- an example of redundant relationships was introduced during discussion on maintaining a history of time-variant data
- use of redundant “manages” and “employs” relationships was justified by the fact that such relationships were dealing with current data rather than historic data

# Redundant Relationship



- note transitive 1:M relationship between DIVISION and PLAYER through TEAM entity
- relationship that connects DIVISION and PLAYER is redundant
- relationship could be safely deleted without losing any information-generation capabilities in model

# **DataBase Design Examples**

# Formula 1

- <https://www.formula1.com/>
- Races
- Drivers
- Teams

# Driver: Fernando ALONSO

- Team      McLaren
- Country    Spain
- Podiums   97
- Points     1830
- Grands Prix entered    272
- World Championships   2
- Highest race finish    1 (x32)
- Highest grid position   1
- Date of birth       29/07/1981
- Place of birth      Oviedo, Spain

- Who is famous for using the catchphrase "*allons-y* Alonso" ?
- **Time** traveler

# Driver: Fernando ALONSO

- Team      *McLaren*
- **Country Spain**
- *Podiums*    97
- *Points*     1830
- *Grands Prix entered*    272
- *World Championships*    2
- *Highest race finish*      1 (x32)
- *Highest grid position*    1
- **Date of birth**    **29/07/1981**
- **Place of birth**    **Oviedo, Spain**

# Entities

- Affiliation
- Result

# Affiliation

- ID of Fernando ALONSO
- ID of McLaren
- Begin Date
- End Date - could be NULL
  - NULL with meaning from Begin Date until today, tomorrow, end of time, ...

# Affiliation view

- as of today()
- JOIN FROM Driver, Affiliation, Team
- WHERE Begin Date <= today() AND
- (today () <= End Date OR End Date IS NULL)

- Begin End Date are time intervals
- Begin  $\leq$  End OR End IS NULL
- Time interval intersection
- $(T_{11}, T_{12}), T_{11} \leq T_{12}$
- $(T_{21}, T_{22}), T_{21} \leq T_{22}$

- Begin End Date are time intervals
- Begin  $\leq$  End OR End IS NULL
- Time interval intersection
- $(T_{11}, T_{12}), T_{21} \leq T_{22}$
- $(T_{21}, T_{22}), T_{21} \leq T_{22}$
- $T_{11} \leq T_{22} \text{ AND } T_{12} \geq T_{21}$

# Temporal Databases

- provide uniform and systematic way of dealing with historical data
- some data may be inherently historical, e.g., medical or judicial records
- regarded as enhancement of conventional (snapshot) database
- sets data into context of time, adds a time dimension

- over time, relational tables are updated, new rows are inserted, some rows are deleted and some attribute values might be modified - data in table changes over time
- if copy of table was taken each time before it is updated and if date of copy was added to all rows we could actually follow evolution of table

- in contrast, in many databases, one would only keep current copy of table - *current snapshot*
- *time cube* snapshot is time slice of such cube, conventional database always holds one slice whereas temporal database holds entire cube

- whenever an update occurs conventional database *physically* updates, throws old values away and stores new ones
- temporal database is updated *logically*, it marks old and new values with timestamps that indicate to which snapshot actually belong
- concept of *physical* vs. *logical deletion*

# Set ... Picture

- UPDATE AutoDriver
  - SET Picture =
- (SELECT BulkColumn FROM OPENROWSET (Bulk 'D:\Downloads\1458061391663.jpg', SINGLE\_BLOB) AS BLOB)
- WHERE id\_ad = 1;

# Executed Only Once in DataBase Server

- USE master
  - GO
- EXEC sp\_configure 'show advanced options', 1
  - GO
- RECONFIGURE WITH OVERRIDE
  - GO
- EXEC sp\_configure 'xp\_cmdshell', 1
  - GO
- RECONFIGURE WITH OVERRIDE
  - GO
- EXEC sp\_configure 'show advanced options', 0
  - GO

# Get ... Picture

- EXEC xp\_cmdshell
- 'BCP "SELECT [Picture] FROM [formula1].[dbo].[AutoDriver] WHERE [id\_ad] = 1" queryout "D:\Downloads\alonso.jpg" -T -n'
- ;

# Challenge

- build a “manele” database with pictures and sound

# manele

# MS SQL Server - configure

- USE master
  - GO
- EXEC sp\_configure 'show advanced options', 1
  - GO
- RECONFIGURE WITH OVERRIDE
  - GO
- EXEC sp\_configure 'xp\_cmdshell', 1
  - GO
- RECONFIGURE WITH OVERRIDE
  - GO
- EXEC sp\_configure 'show advanced options', 0
  - GO

# MS SQL Server – Data Type

- for binary - VARBINARY(max)

# MS SQL Server - SET

- UPDATE manea SET manea =
- (SELECT BulkColumn FROM OPENROWSET (Bulk 'D:\Downloads\Carmen Serban - Picioarele epilate sunt ca sarea in bucate.mp3', SINGLE\_BLOB) AS BLOB)
- WHERE id\_manea = 1;
- UPDATE manea SET manea =
- (SELECT BulkColumn FROM OPENROWSET (Bulk 'D:\Downloads\Carmen Serban-Viata bate filmul.mp3', SINGLE\_BLOB) AS BLOB)
- WHERE id\_manea = 2;

# MS SQL Server - GET

- EXEC xp\_cmdshell
- 'BCP "SELECT [manea] FROM [manele].[dbo].[manea] WHERE [id\_manea] = 1" queryout "D:\Downloads\FileOut1.mp3" -T -n'
- EXEC xp\_cmdshell
- 'BCP "SELECT [manea] FROM [manele].[dbo].[manea] WHERE [id\_manea] = 2" queryout "D:\Downloads\FileOut2.mp3" -T -n'

# MySQL - configure

- max\_allowed\_packet=100M
- in my.ini

# MySQL Server – Data Type

- LargeBLOB (Binary Large OBject)

# MySQL - SET

- UPDATE manea SET manea =
- LOAD\_FILE('D:/Downloads/Carmen Serban - Picioarele epilate sunt ca sarea in bucate.mp3')
- WHERE idmanea = 1;
- UPDATE manea SET manea =
- LOAD\_FILE('D:/Downloads/Carmen Serban- Viata bate filmul.mp3')
- WHERE idmanea = 2;

# MySQL - GET

- SELECT manea INTO DumpFile  
‘D:/Downloads/FileOut1.mp3’
- FROM manea WHERE idmanea = 1;
- SELECT manea INTO DumpFile  
‘D:/Downloads/FileOut2.mp3’
- FROM manea WHERE idmanea = 2;

Additional optional material

# **WISDOM**

# **Picioarele Epilate Sunt Ca Sarea in Bucate**

## **Epilate Legs Are Like Salt in Food**

# Viață bate filmul

- Mi-am tocit hainele în coate
- Să termin o facultate
- ....
- Sunt zilier cu facultate
- La un patron fără carte
- ....
  - Să am banii de chirie

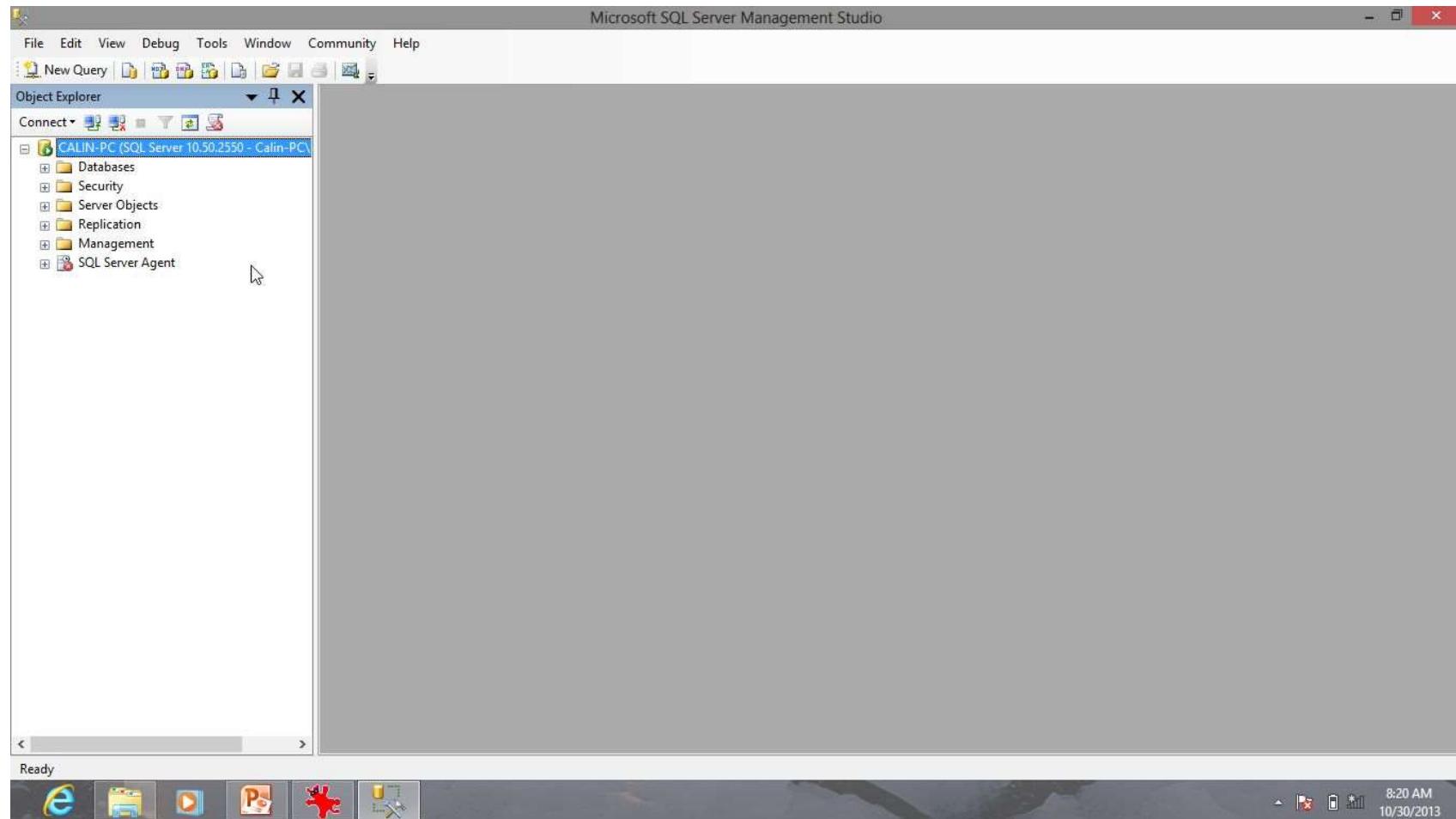
# Viață bate filmul

- ...
- Eu sunt fraierul cu carte
- Toți îmi spun intelectualul
- Dar nu am căscavalul
- Ieri eram inginerășul
- De azi să îmi spuneți nașul
- ...

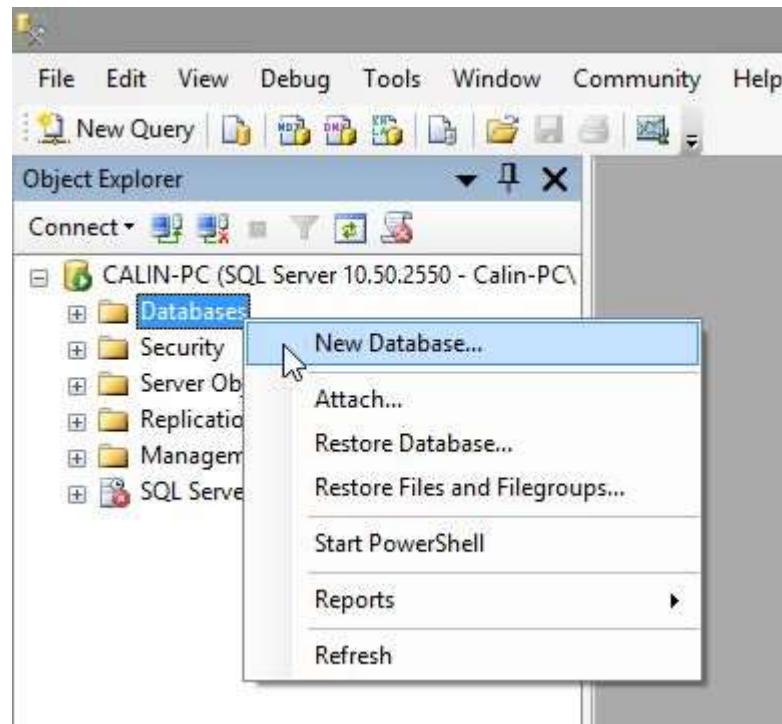
# Database

- In order to install MicroSoft SQL Server read slides:DBD Course07 DataBase Administration
- .pptx, ...pag. 60:
  - Preferably mixed mode authentication
  - Remember password for System Administrator
  - Make at least current user administrator
  - Make another user administrator

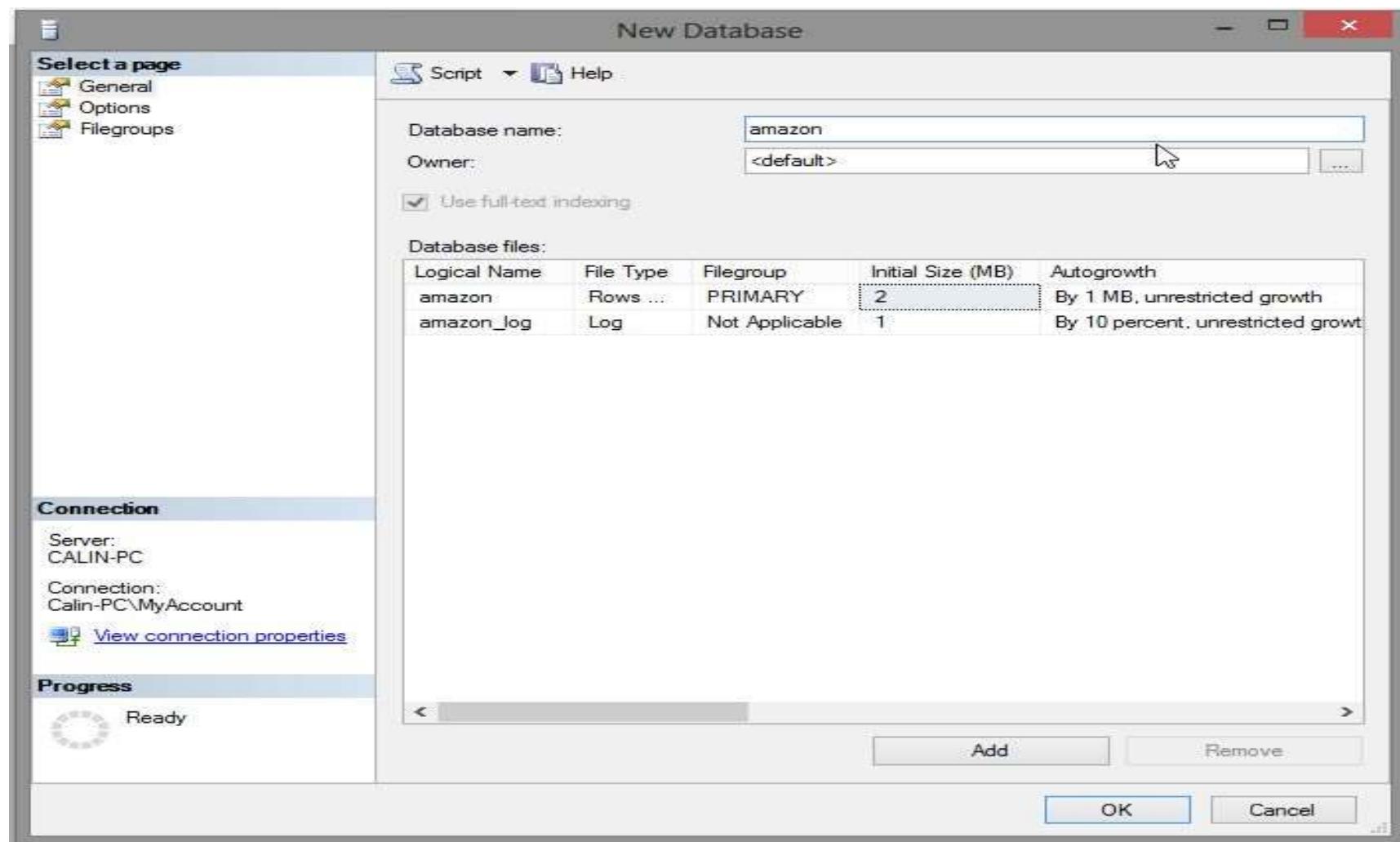
# MS SQL Server Management Studio



# Right click to Create New Database

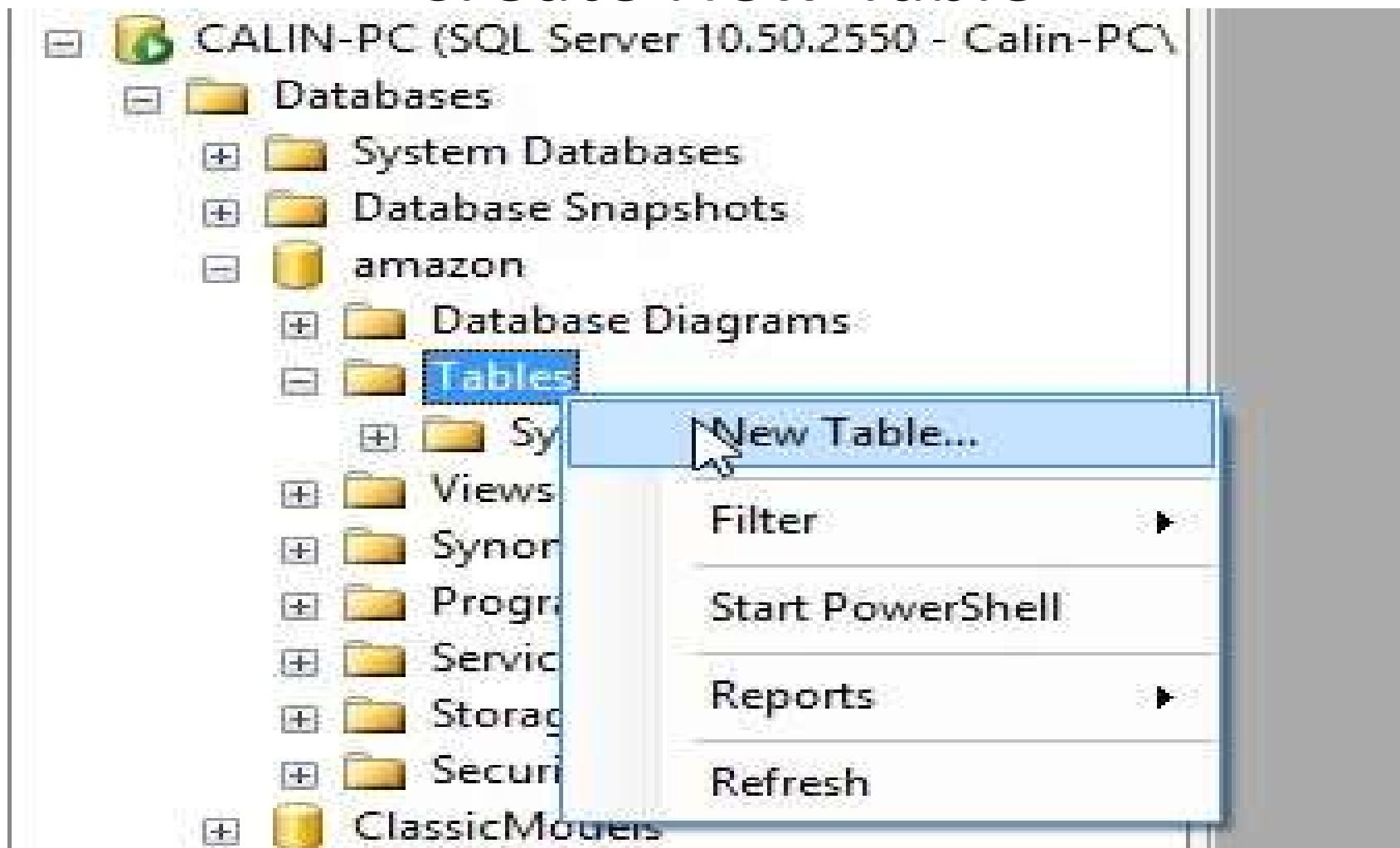


# Create Database amazon



# Database amazon

## Create New Table



ID\_Publisher – INTeger  
right click – Set Primary Key

CALIN-PC.amazon - dbo.Table\_1\*

| Column Name  | Data Type | Allow Nulls                         |
|--------------|-----------|-------------------------------------|
| ID_Publisher | int       | <input checked="" type="checkbox"/> |

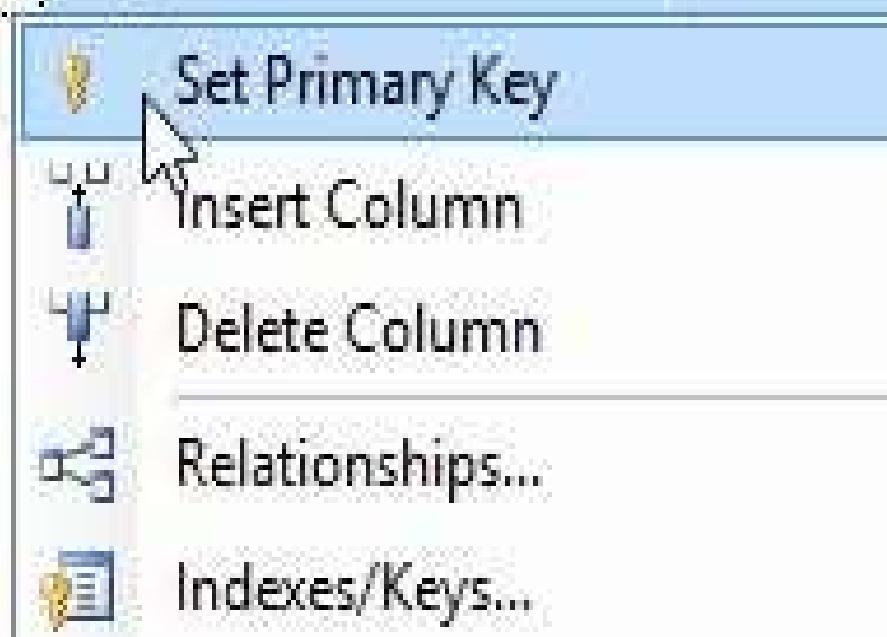
Set Primary Key

Insert Column

Delete Column

Relationships...

Indexes/Keys...



# ID\_Publisher – surrogate key Identity

Column Properties

| Full-text Specification       | No         |
|-------------------------------|------------|
| Has Non-SQL Server Subscriber | No         |
| Identity Specification        | Yes        |
| <b>(Is Identity)</b>          | <b>Yes</b> |
| Identity Increment            | 1          |
| Identity Seed                 | 1          |
| Indexable                     | Yes        |
| <b>(Is Identity)</b>          |            |

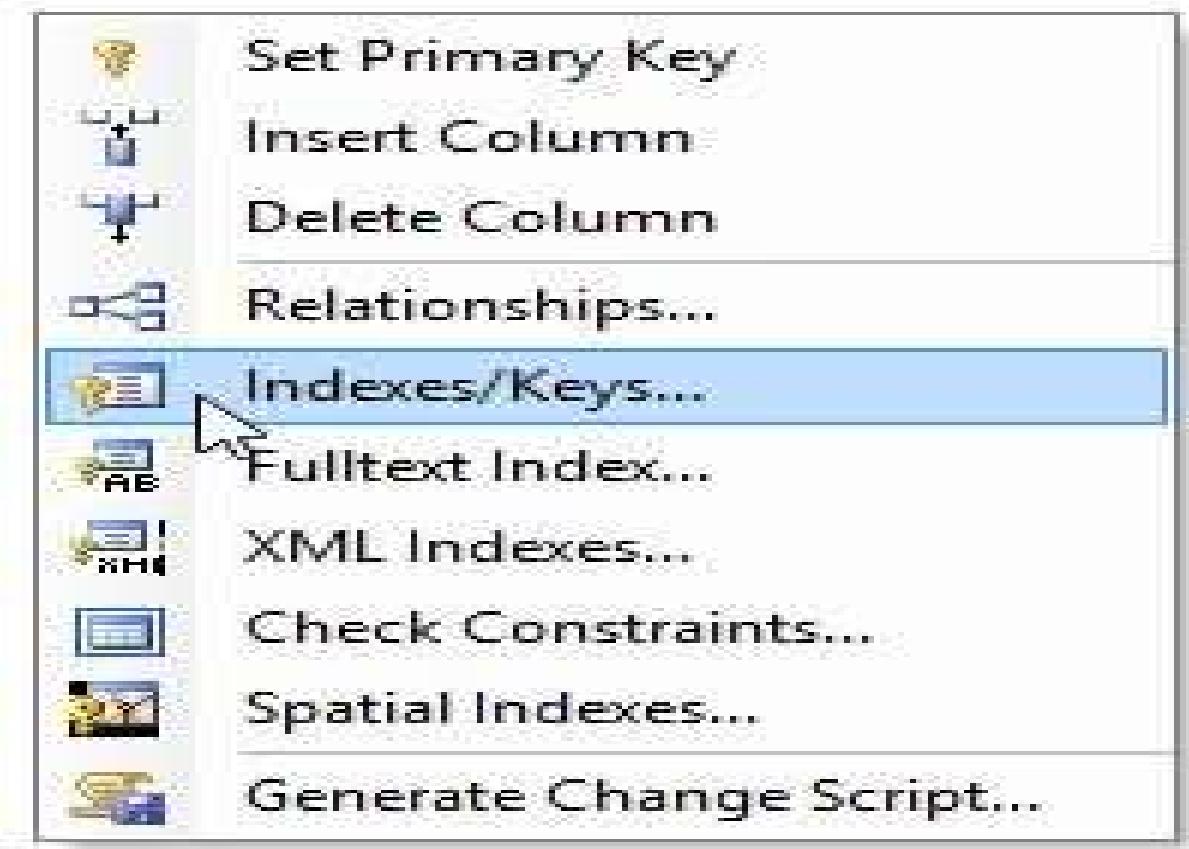
# Publisher – VarChar(50) – Not NULL

| CALIN-PC.amazon - dbo.Table_1* |              |              |                                     |
|--------------------------------|--------------|--------------|-------------------------------------|
|                                | Column Name  | Data Type    | Allow Nulls                         |
| PK                             | ID_Publisher | int          | <input type="checkbox"/>            |
|                                | Publisher    | varchar(50)  | <input type="checkbox"/>            |
| FK                             | Address      | varchar(150) | <input checked="" type="checkbox"/> |
|                                |              |              | <input type="checkbox"/>            |

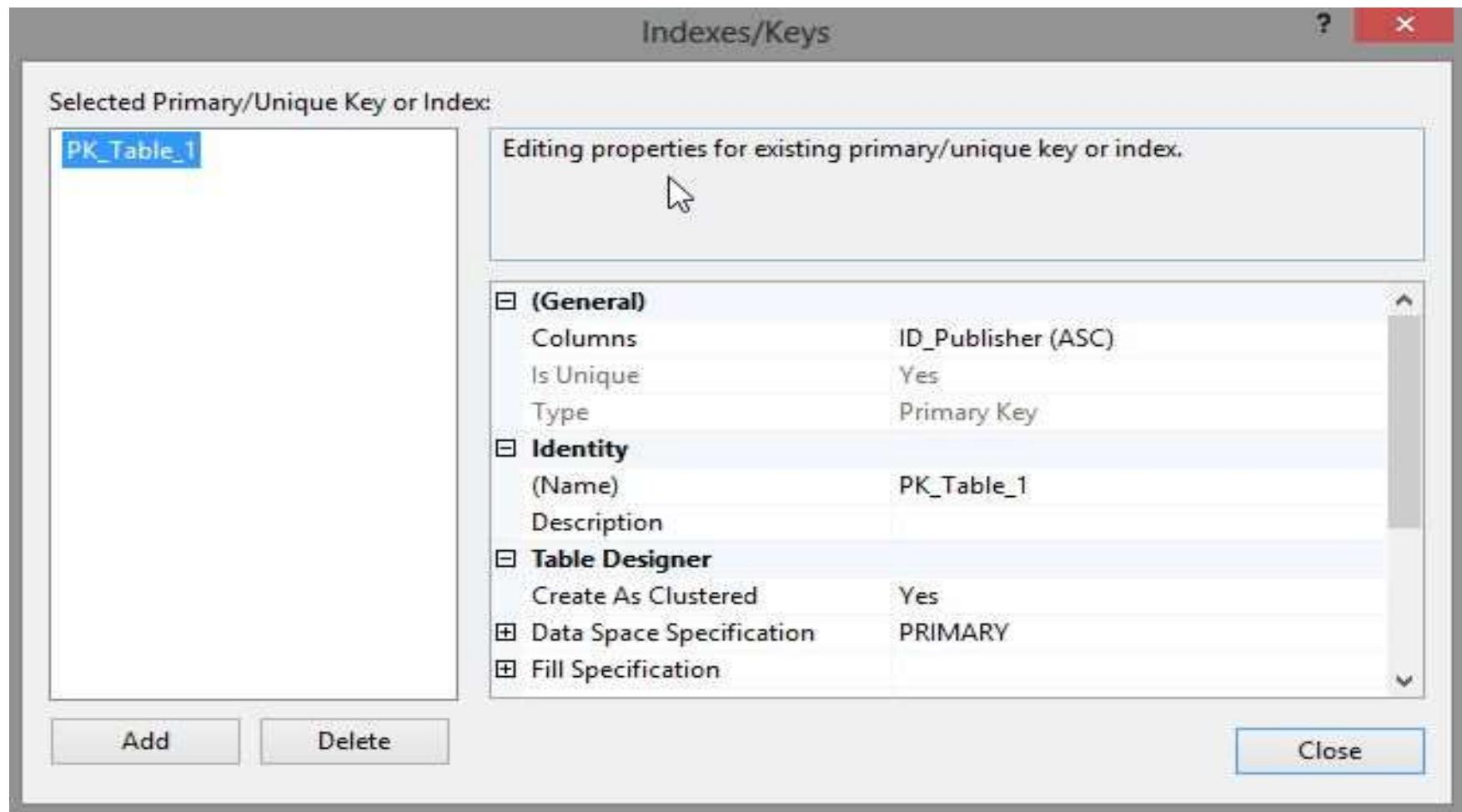
# Address – VarChar(150) –NULL allow

| CALIN-PC.amazon - dbo.Table_1* |              |              |                                     |
|--------------------------------|--------------|--------------|-------------------------------------|
|                                | Column Name  | Data Type    | Allow Nulls                         |
| PK                             | ID_Publisher | int          | <input type="checkbox"/>            |
|                                | Publisher    | varchar(50)  | <input type="checkbox"/>            |
| FK                             | Address      | varchar(150) | <input checked="" type="checkbox"/> |
|                                |              |              | <input type="checkbox"/>            |

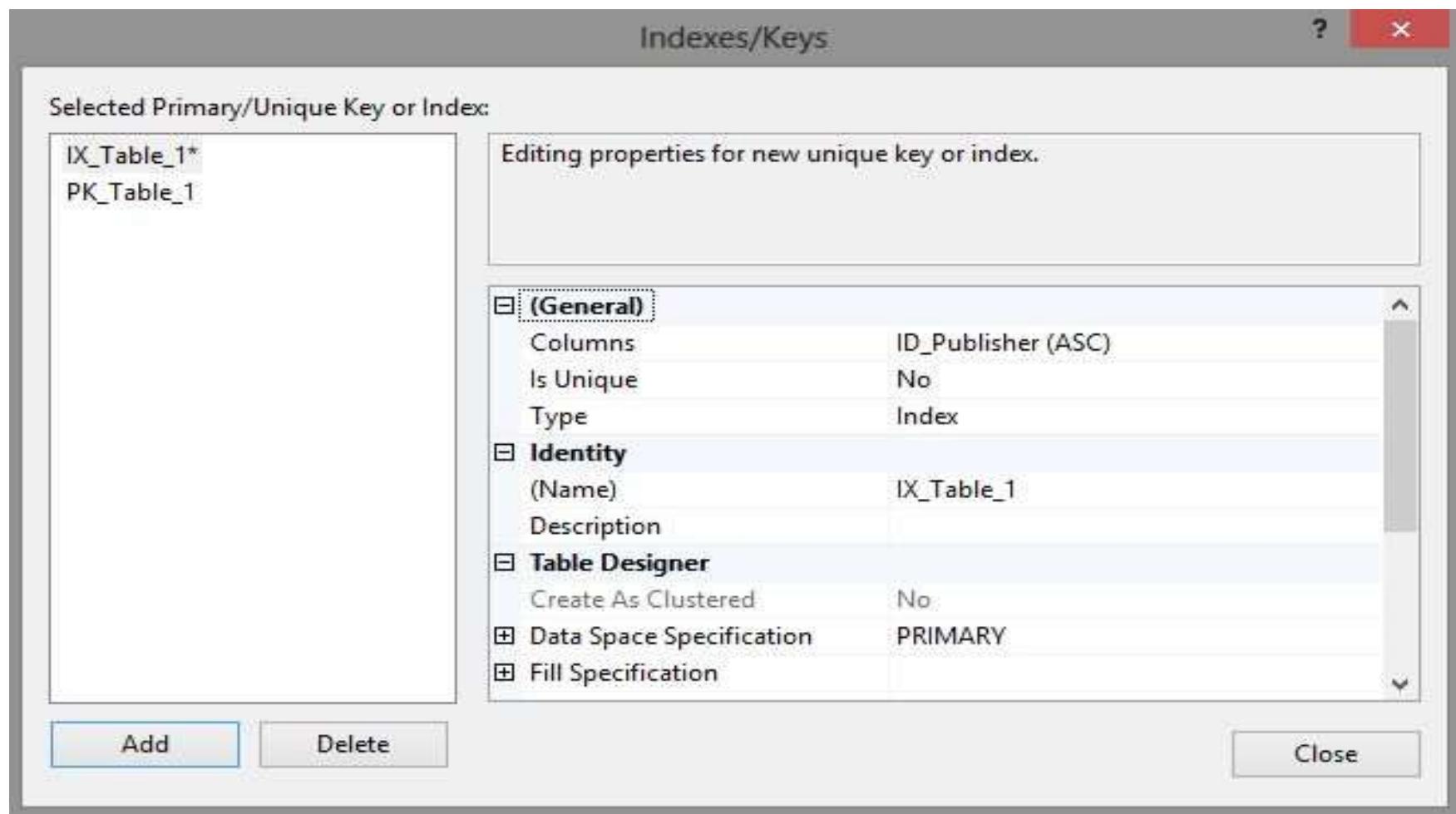
# Create Index / Keys



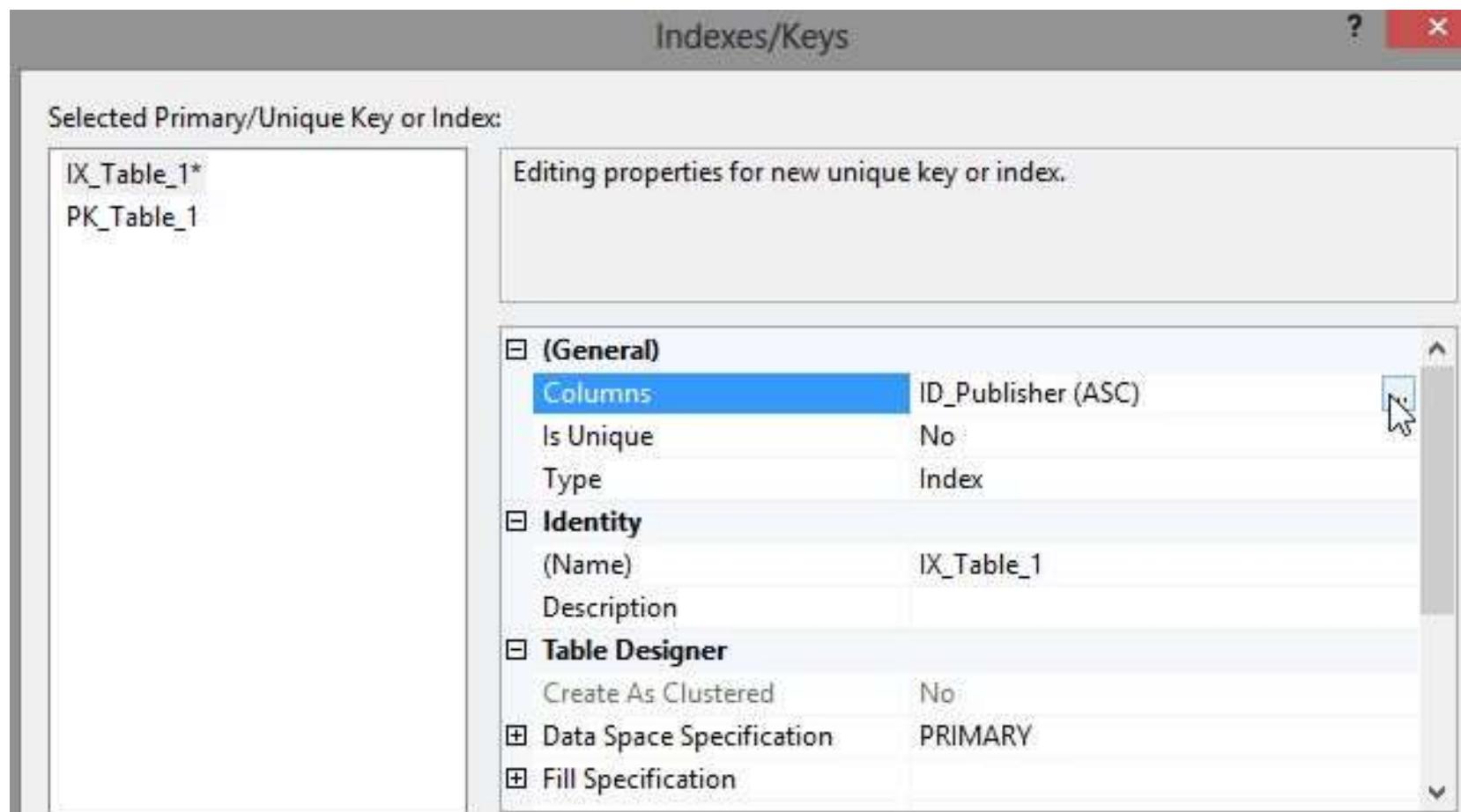
# Index for PK already created



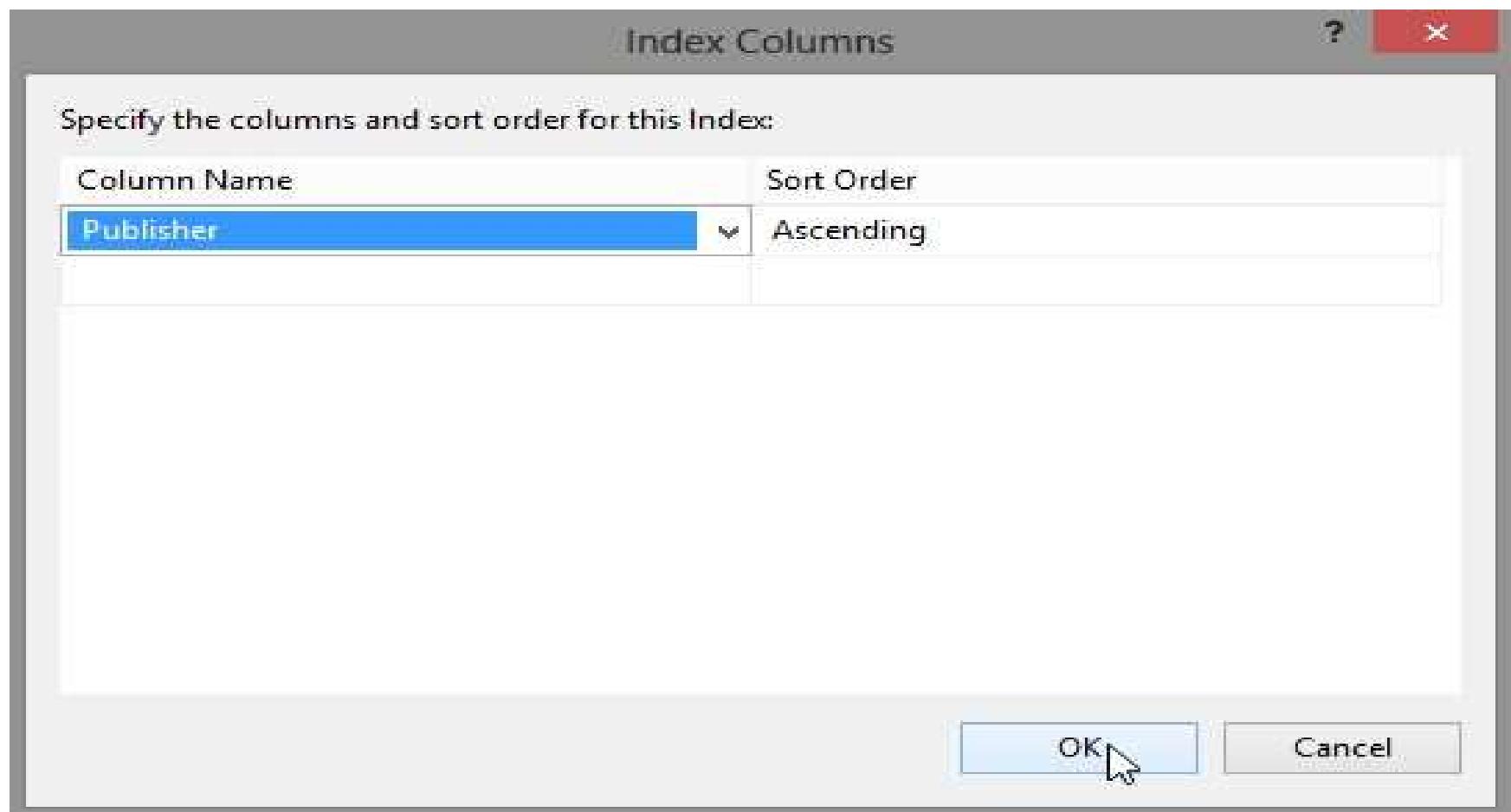
# Add – New Index



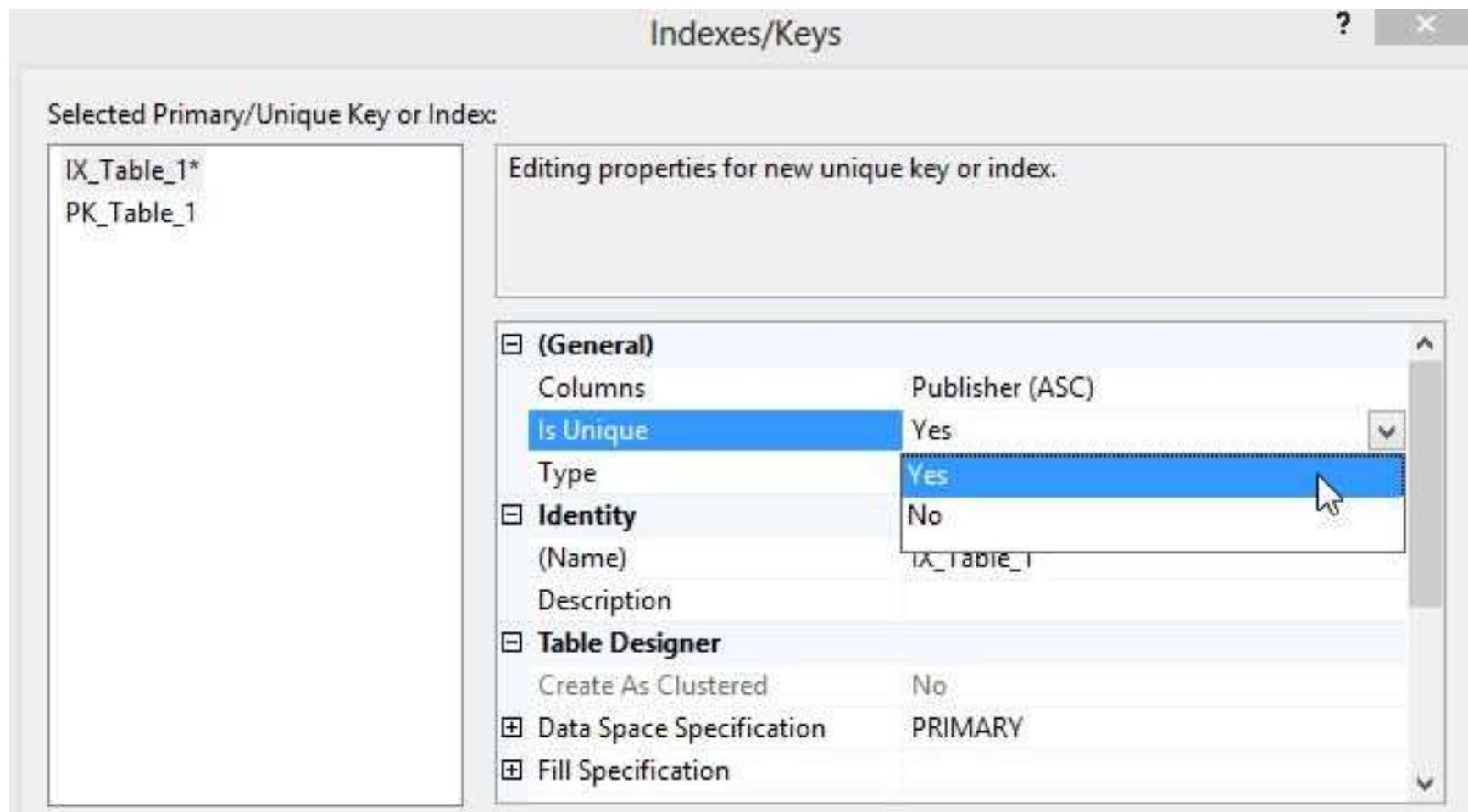
# Choose Columns



# Publisher



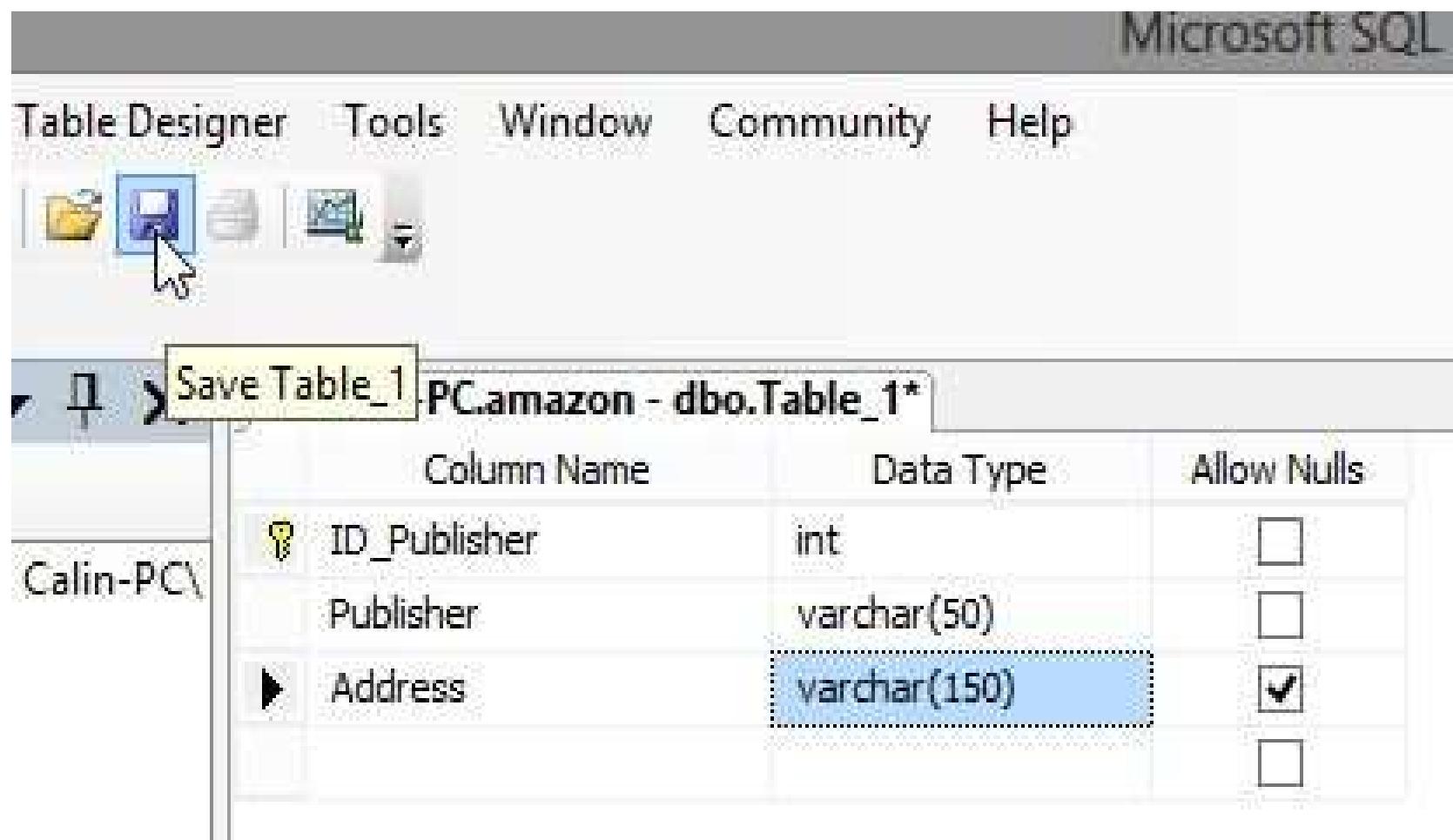
# Is Unique



# Publisher

- ID\_Publisher – INTeger – Identity – Primary Key (Not allow NULL)
- Publisher – VarChar(50) - Alternate Key (Is Unique, Not allow NULL)
- Address – VarChar(150) – allow NULL

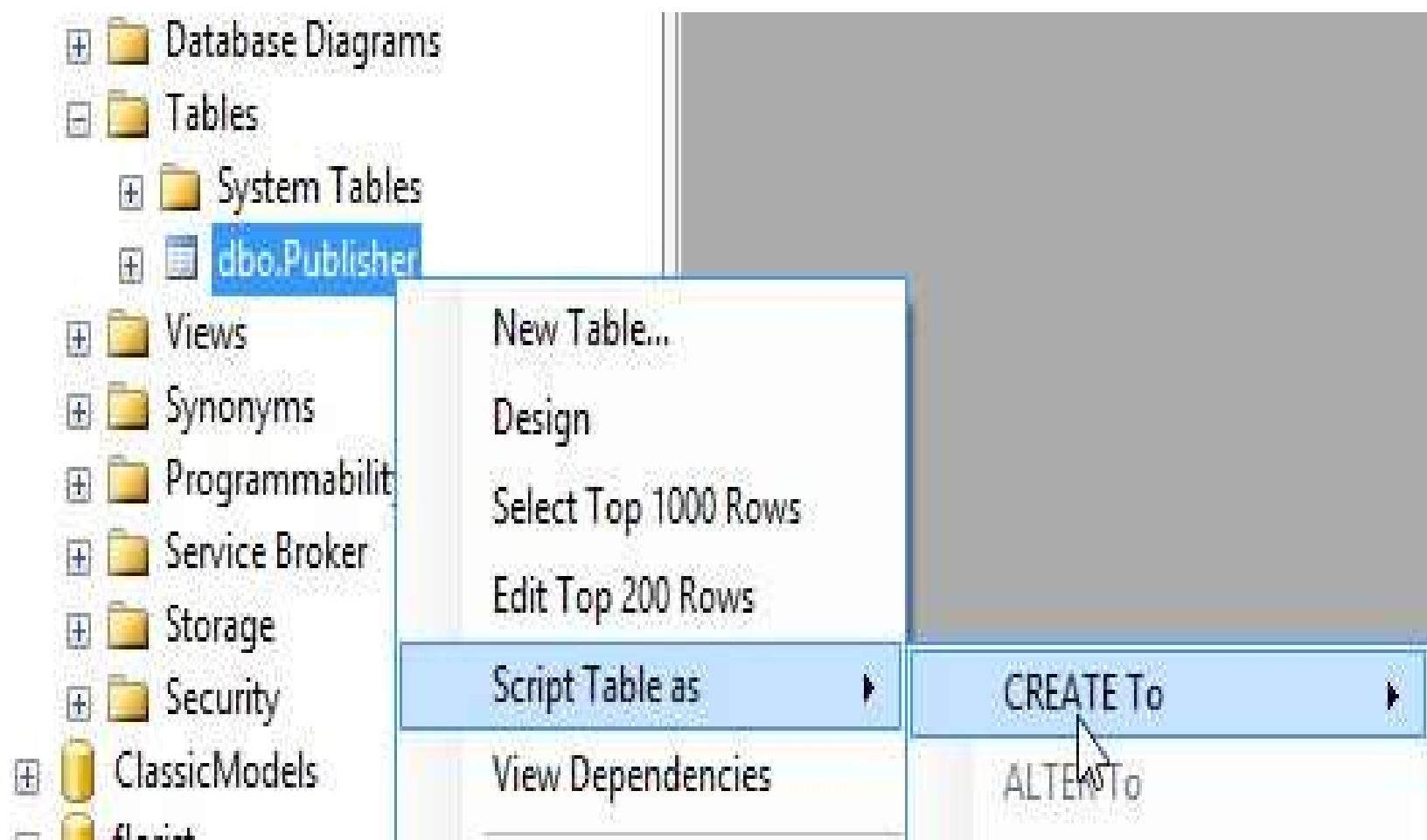
# Save Table



# Table Publisher



# Script Table



# CREATE TABLE [dbo].[Publisher](

- CREATE TABLE [dbo].[Publisher](
- [ID\_Publisher] [int] IDENTITY(1,1) NOT NULL,
- [Publisher] [varchar](50) NOT NULL,
- [Address] [varchar](150) NULL,
- CONSTRAINT [PK\_Publisher] PRIMARY KEY CLUSTERED
- (
- [ID\_Publisher] 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 Publisher

- CREATE TABLE [dbo].[Publisher](
- ID\_Publisher int IDENTITY(1,1) NOT NULL,
- Publisher varchar (50) NOT NULL,
- Address varchar (150) NULL,
- CONSTRAINT [PK\_Publisher] PRIMARY KEY (
- [ID\_Publisher] ASC
- ))

# CREATE TABLE Author

- CREATE TABLE [dbo].[Author](
- [ID\_Author] [int] IDENTITY(1,1) NOT NULL,
- [Author] [nvarchar](50) NOT NULL,
- CONSTRAINT [PK\_Author] PRIMARY KEY  
CLUSTERED
- (
- [ID\_Author] ASC
- ))

# CREATE TABLE Genre

- CREATE TABLE [dbo].[Genre](
- [ID\_Genre] [int] IDENTITY(1,1) NOT NULL,
- [Genre] [varchar](50) NOT NULL,
- CONSTRAINT [PK\_Genre] PRIMARY KEY  
CLUSTERED
- (
- [ID\_Genre] ASC
- ))

# CREATE TABLE Books

- CREATE TABLE [dbo].[Book](
  - [ISBN] [char](13) NOT NULL,
  - [Title] [varchar](50) NOT NULL,
  - [Publisher\_ID] [int] NOT NULL,
  - [Author\_ID] [int] NULL,
  - [Price] [money] NOT NULL,
  - [Pages] [int] NULL,
  - [PubDate] [date] NULL,
  - [Description] [varchar](max) NULL,
- CONSTRAINT [PK\_Book] PRIMARY KEY CLUSTERED
- ( [ISBN] ASC )

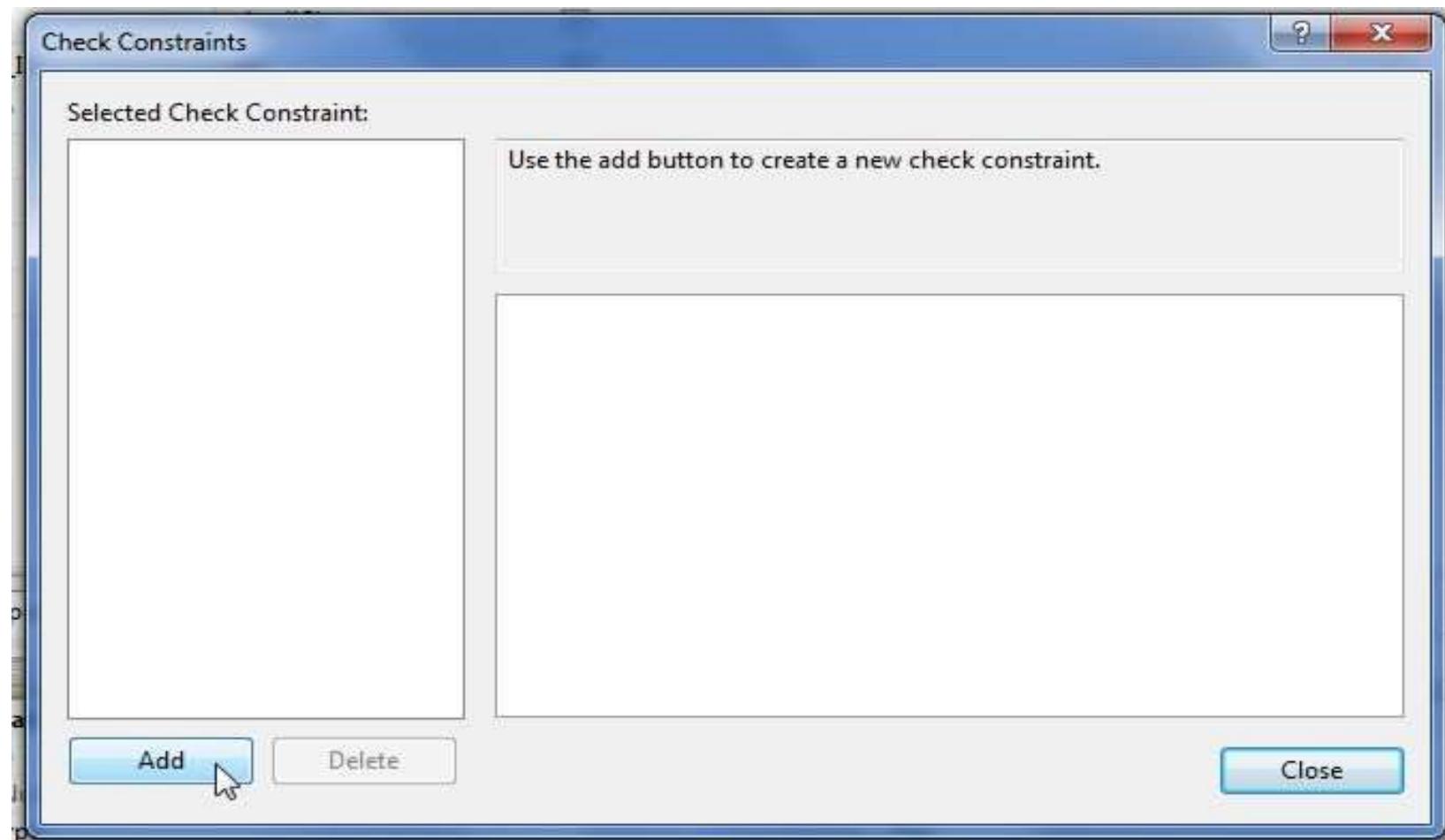
# Varchar(MAX)

- Description is VarChar(MAX) to replace deprecated TEXT, TLOB (Text Large OBject)
- VarBinary(MAX) replace deprecated IMAGE, BLOB (Binary Large OBject)

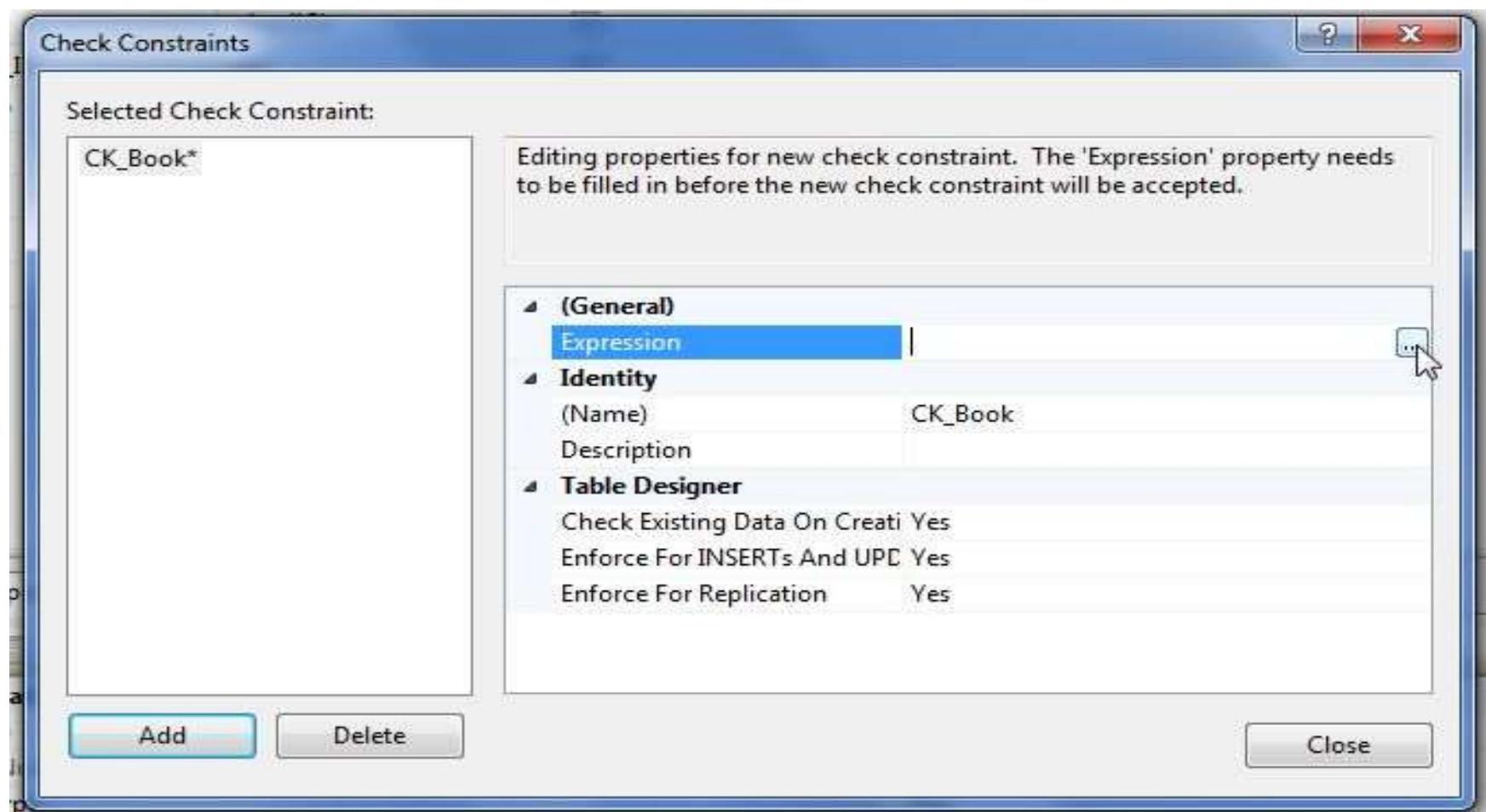
# Check Constraint



# Add Check Constraint



# Expression



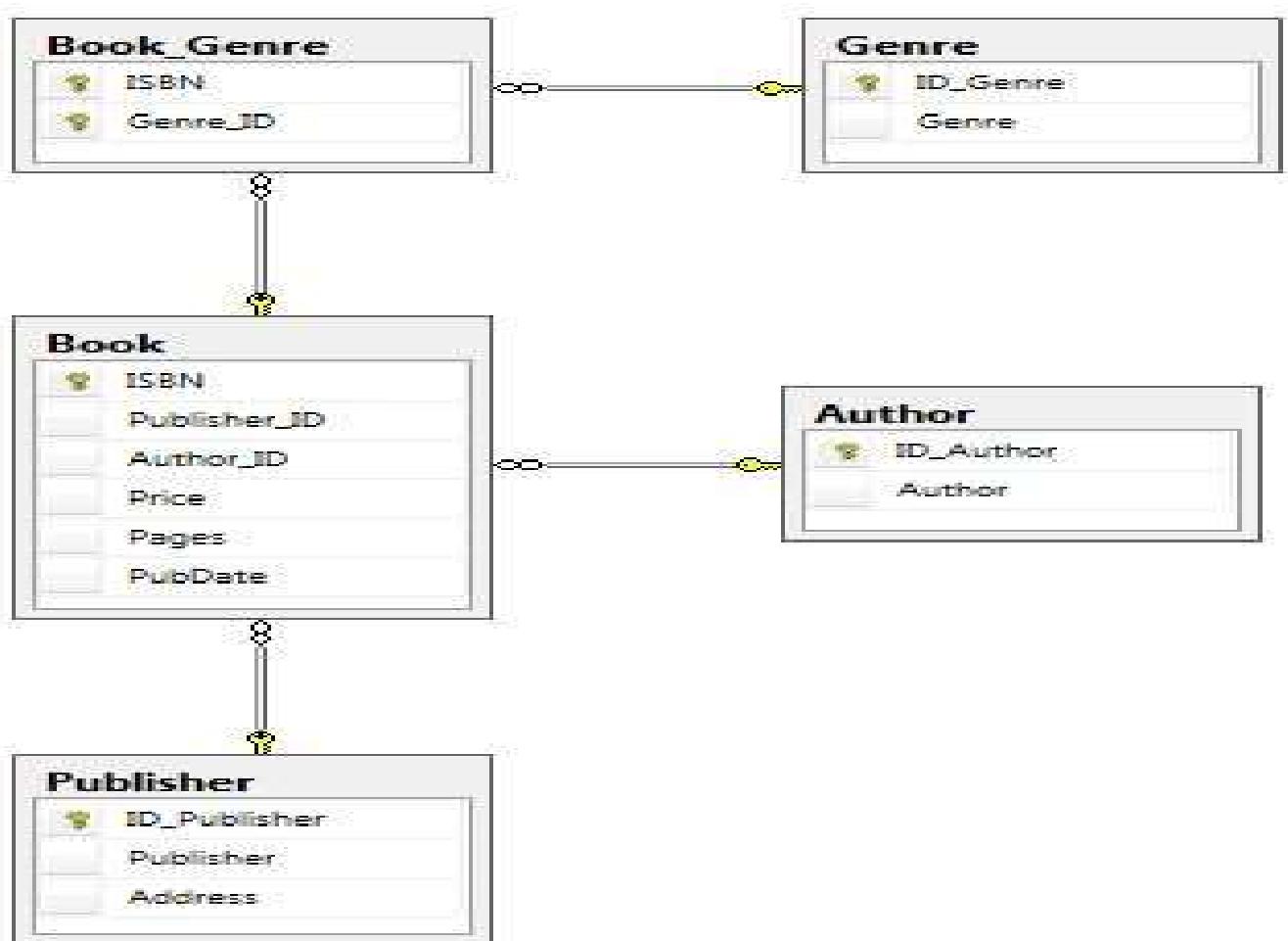
# Constraints

- Price > 0
- Pages > 0 OR Pages IS NULL
- ISBN LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'  
[0-9][0-9][0-9][0-9][0-9][0-9]

# CREATE TABLE Book\_Genre

- CREATE TABLE [dbo].[Book\_Genre](
- [ISBN] [char](13) NOT NULL,
- [Genre\_ID] [int] NOT NULL,
- CONSTRAINT [PK\_Book\_Genre] PRIMARY KEY CLUSTERED
- (
- [ISBN] ASC,
- [Genre\_ID] ASC
- ))

# DataBase Diagram



- Design is over ...
- Let's try to fill some data

# NVarChar ...

|    | ID_Author | Author             |
|----|-----------|--------------------|
|    | 1         | Raghu Ramakri...   |
|    | 2         | Isaac Asimov       |
|    | 3         | Mircea Cărtăres... |
| ▶* | NULL      | NULL               |

# View VBookGenre

- SELECT dbo.Book.ISBN, dbo.Book.Title,  
dbo.Genre.Genre
  - FROM dbo.Book INNER JOIN  
• dbo.Book\_Genre ON Book.ISBN  
= Book\_Genre.ISBN INNER JOIN  
• Genre ON Book\_Genre.Genre\_ID =  
dbo.Genre.ID\_Genre

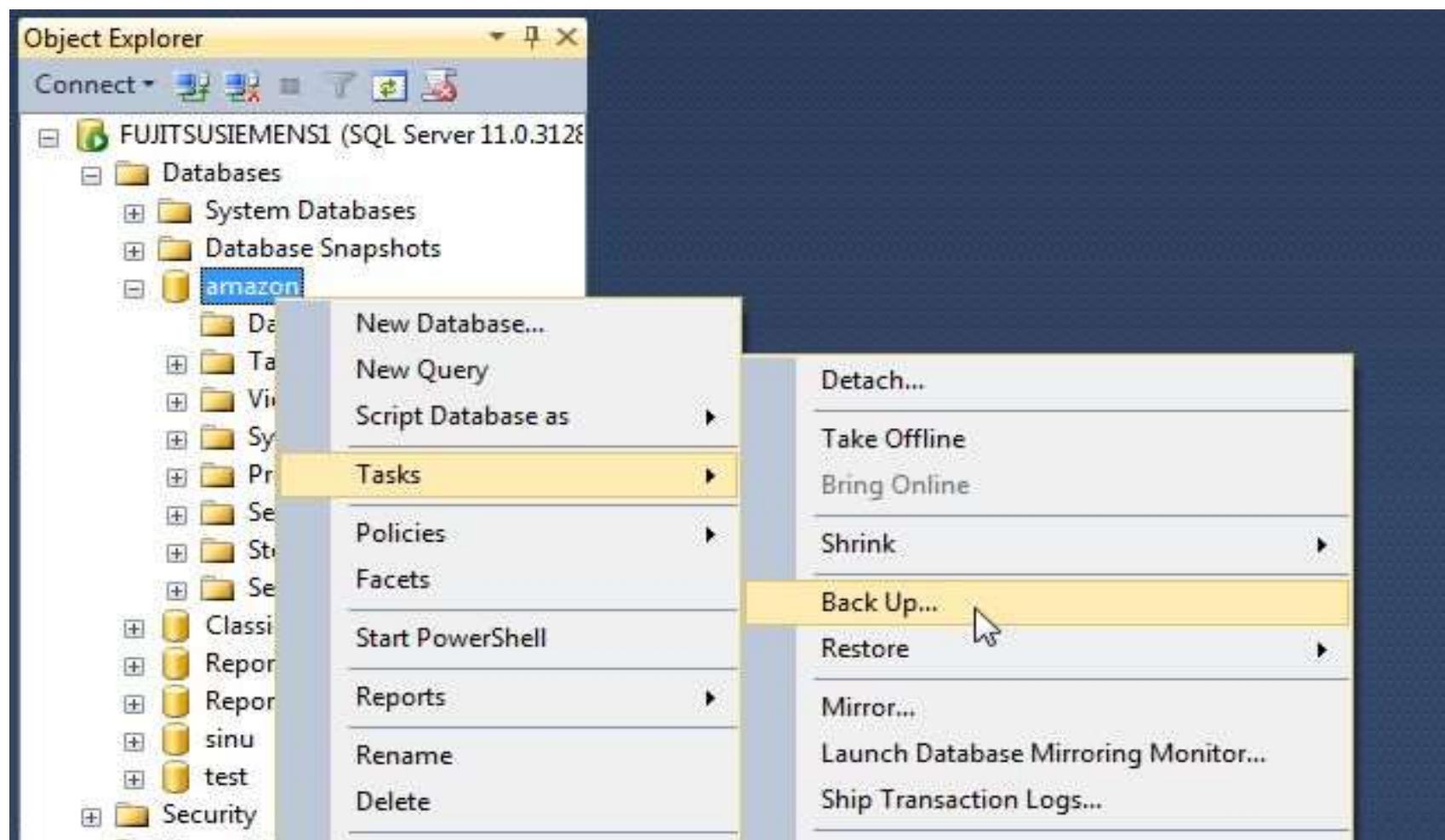
# View VBook

- SELECT dbo.Book.ISBN, dbo.Book.Title,  
dbo.Author.Author, dbo.Publisher.Publisher,  
dbo.Book.Price, dbo.Book.Pages,  
dbo.Book.PubDate, dbo.Book.Description
- FROM dbo.Author INNER JOIN  
• dbo.Book ON  
dbo.Author.ID\_Author = dbo.Book.Author\_ID  
INNER JOIN  
• dbo.Publisher ON  
dbo.Book.Publisher\_ID =  
dbo.Publisher.ID\_Publisher

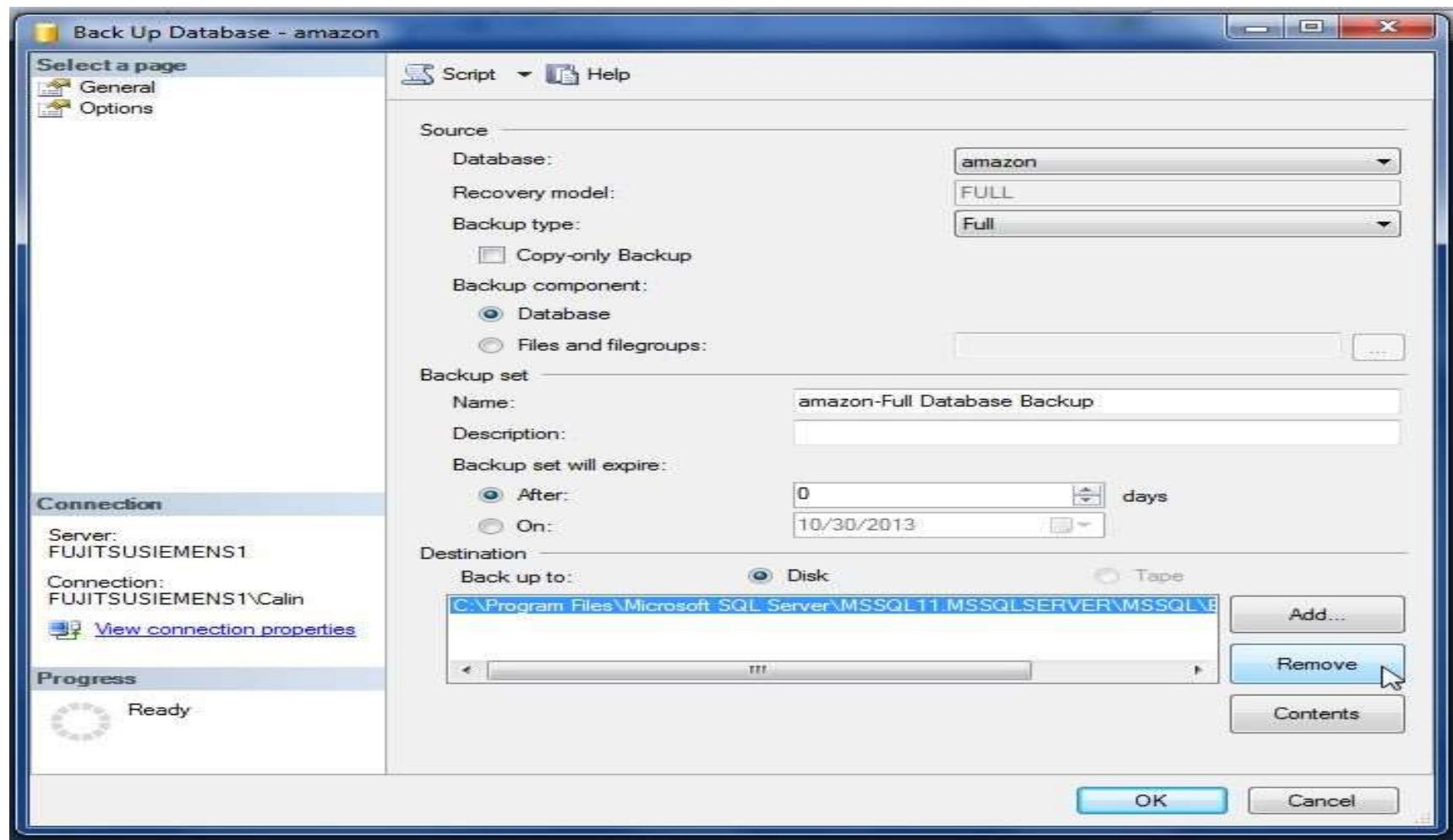
- DataBase Design – checked
  - Data – checked
  - External level – Views – checked
- 
- Spread the ... database

# BackUp & ReStore

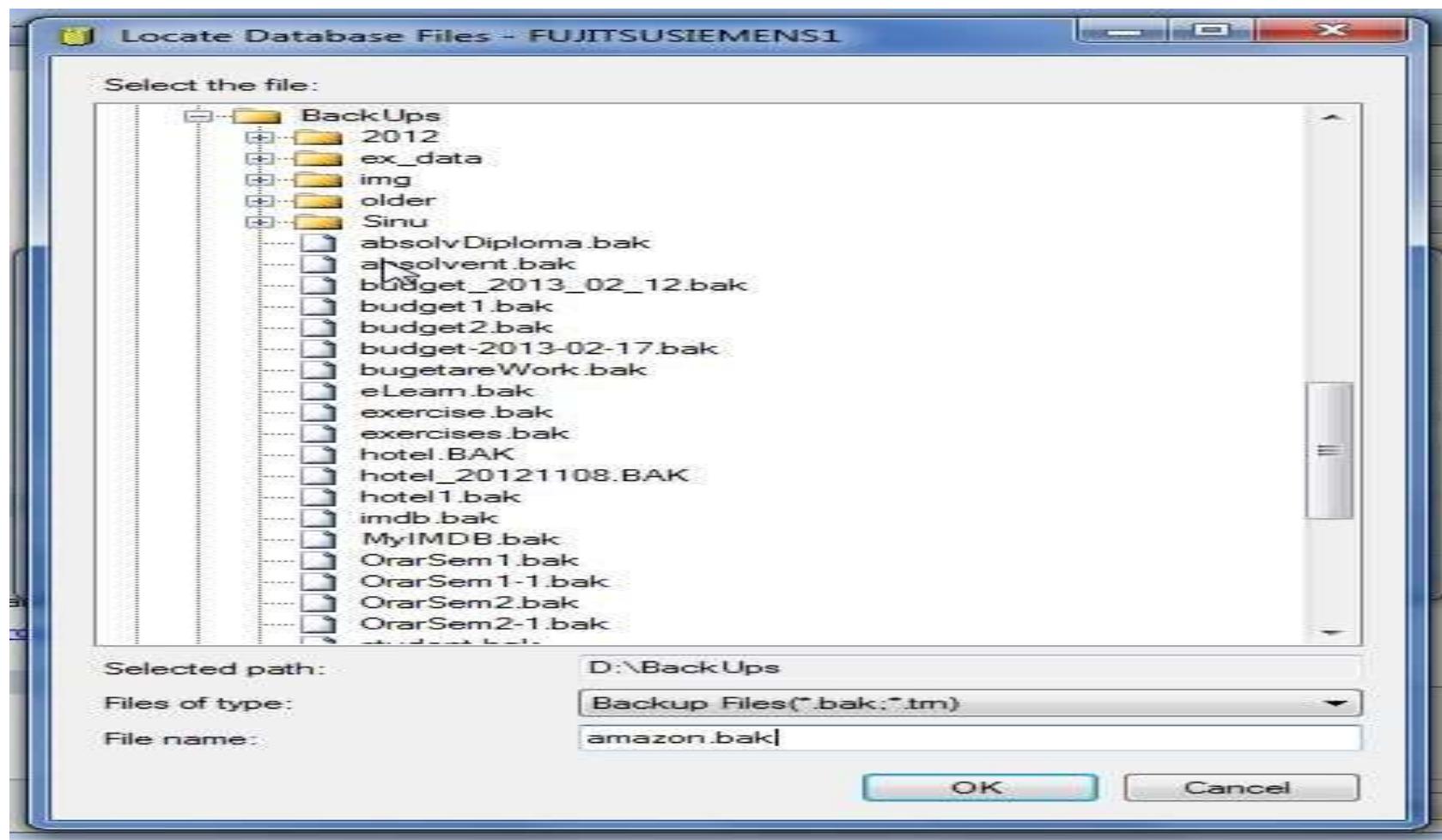
- Correct way
  - Backup & restore
  - 2008 backup could be restored in 2012
  - 2012 backup could not be restored in 2018
- Script database way
  - Works for small database
  - Works both ways between different versions
- Barbaric way
  - Attach, Detach



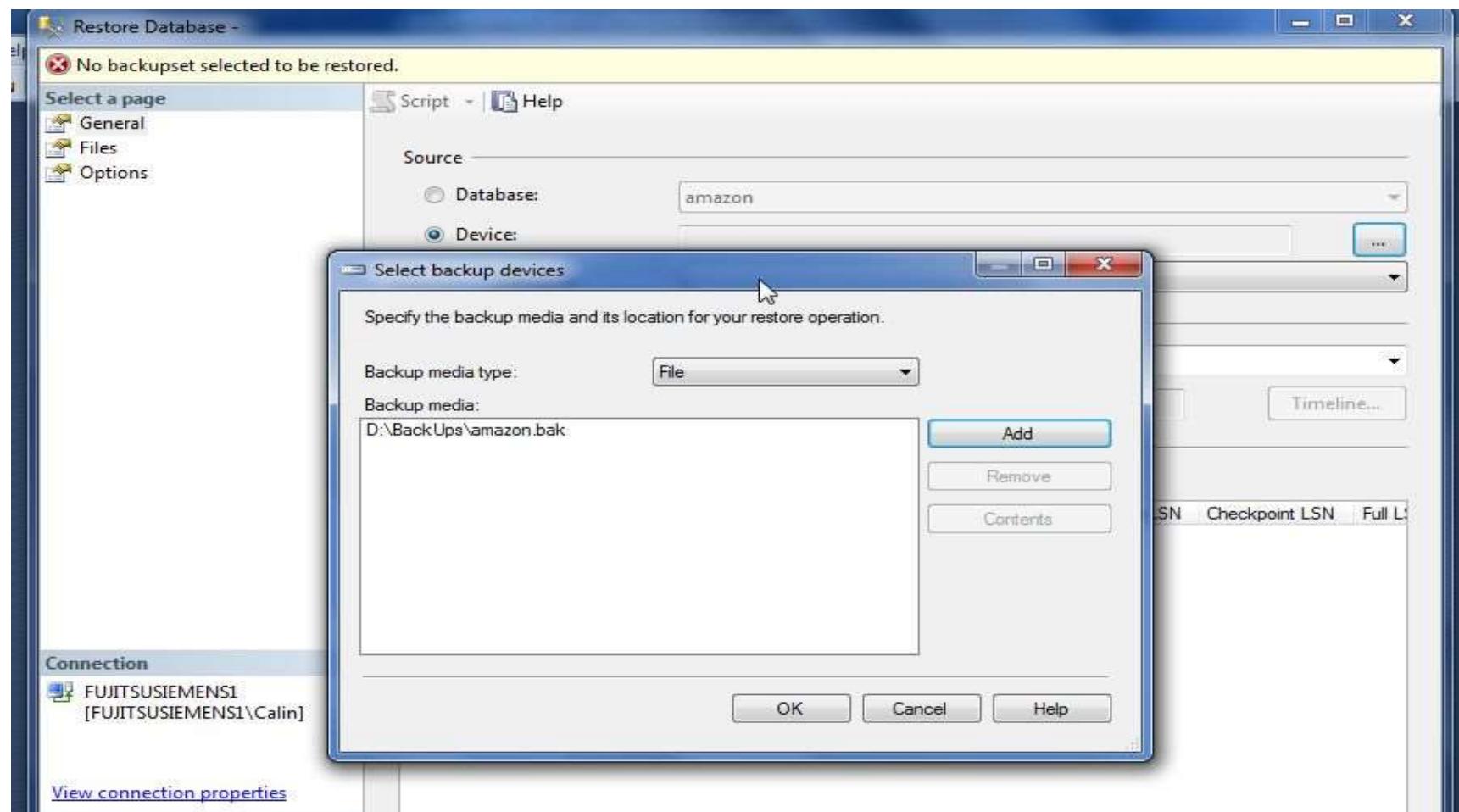
# Never forget to Remove



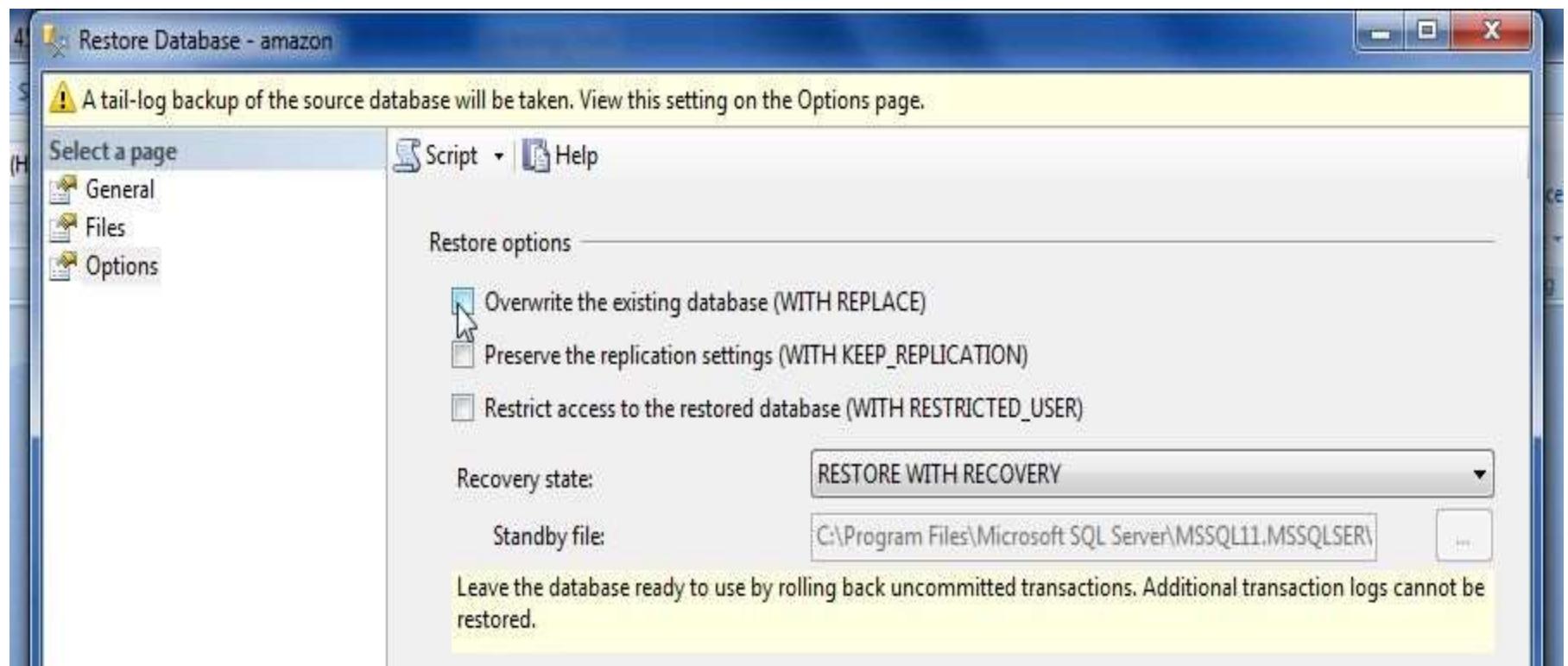
# Add



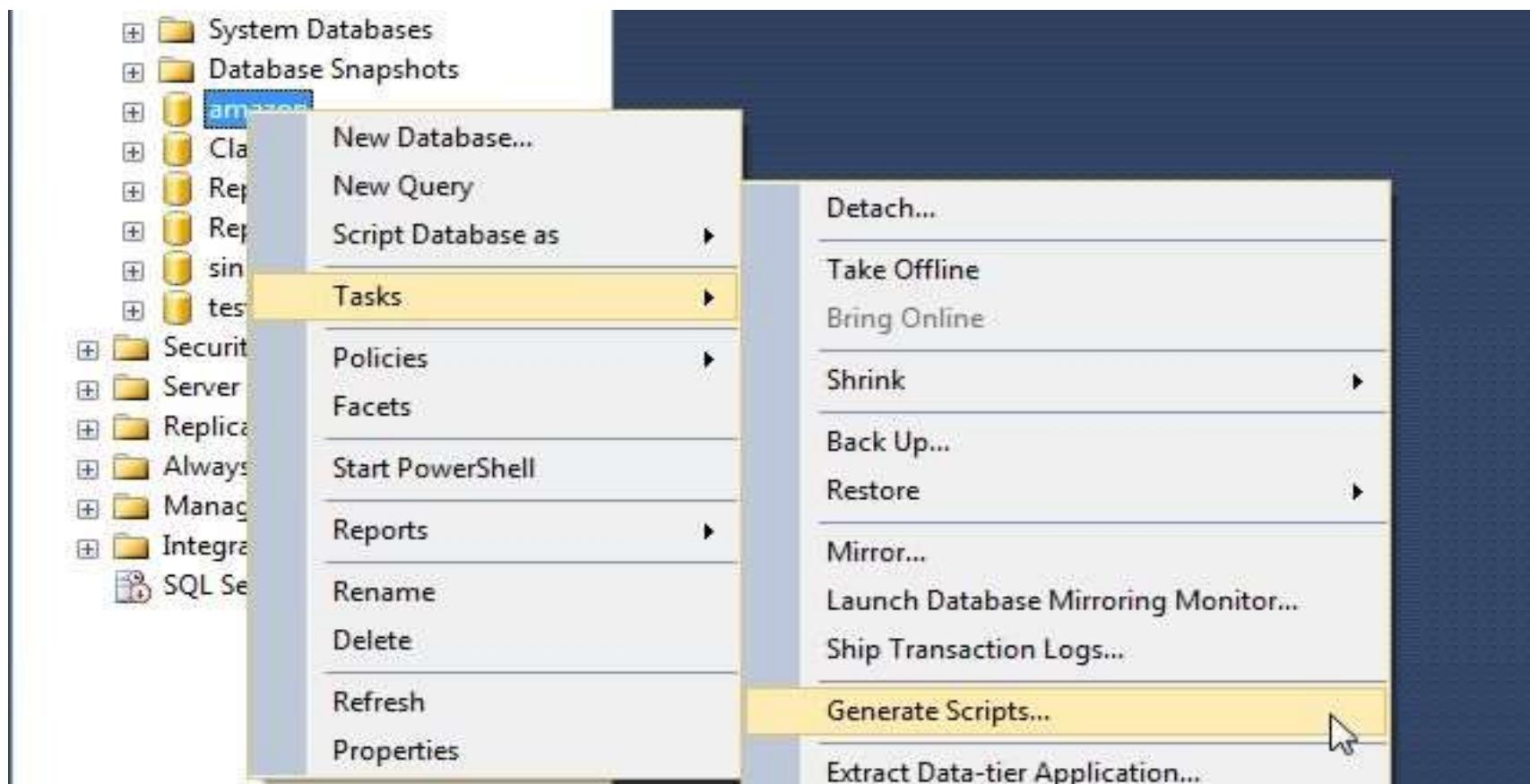
# Restore - device

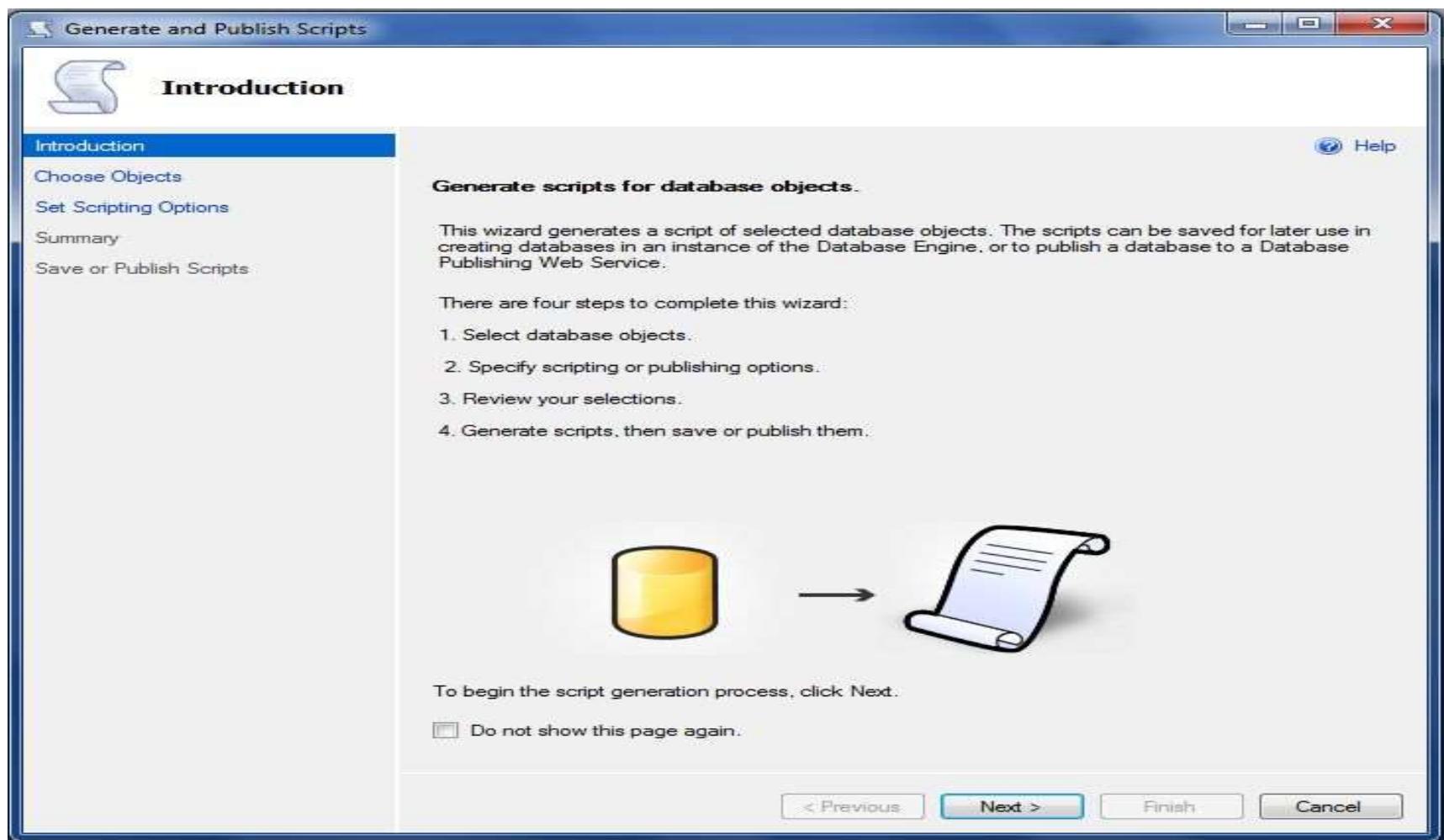


# OverWrite the existing database

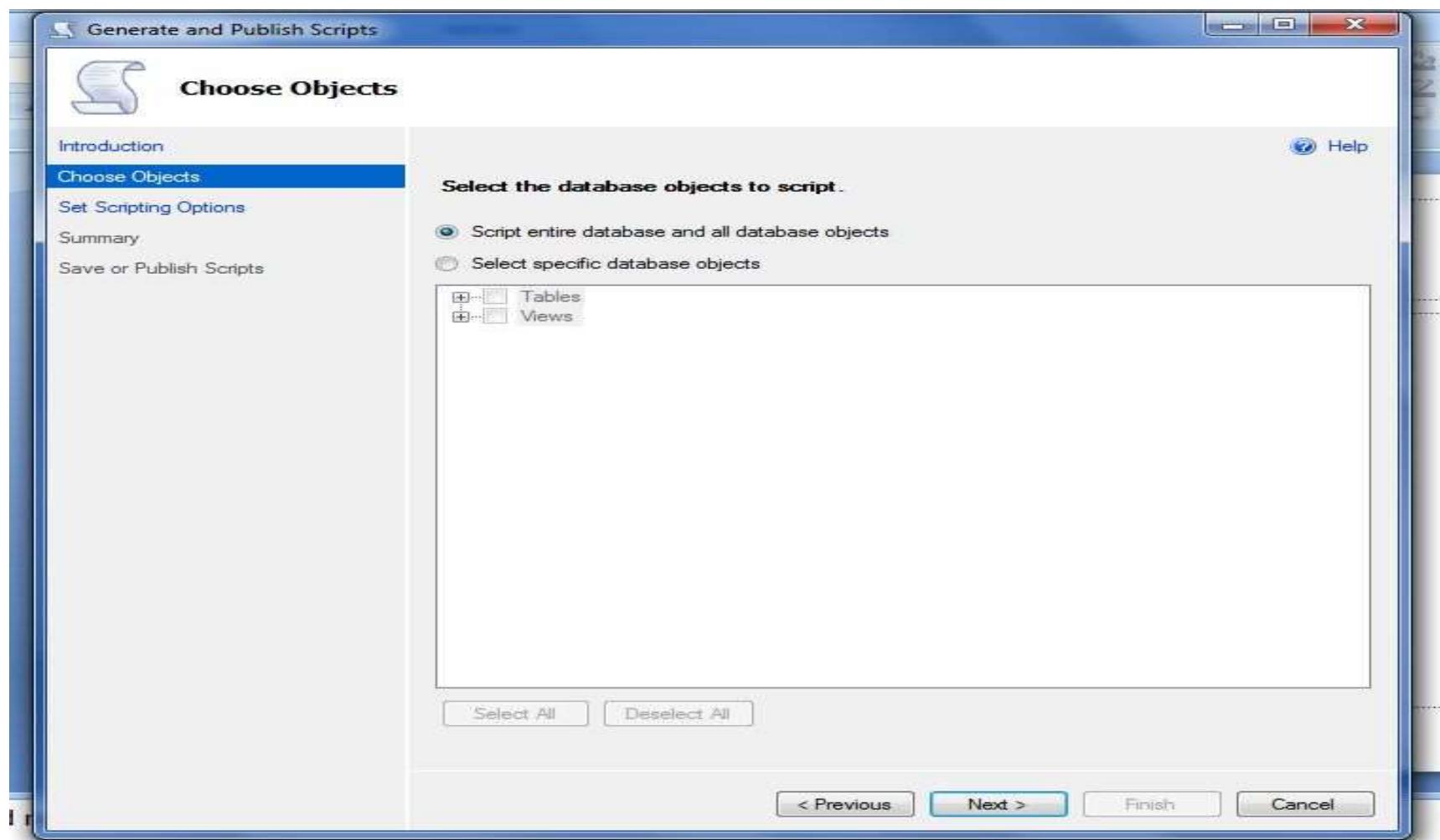


# Generate Scripts

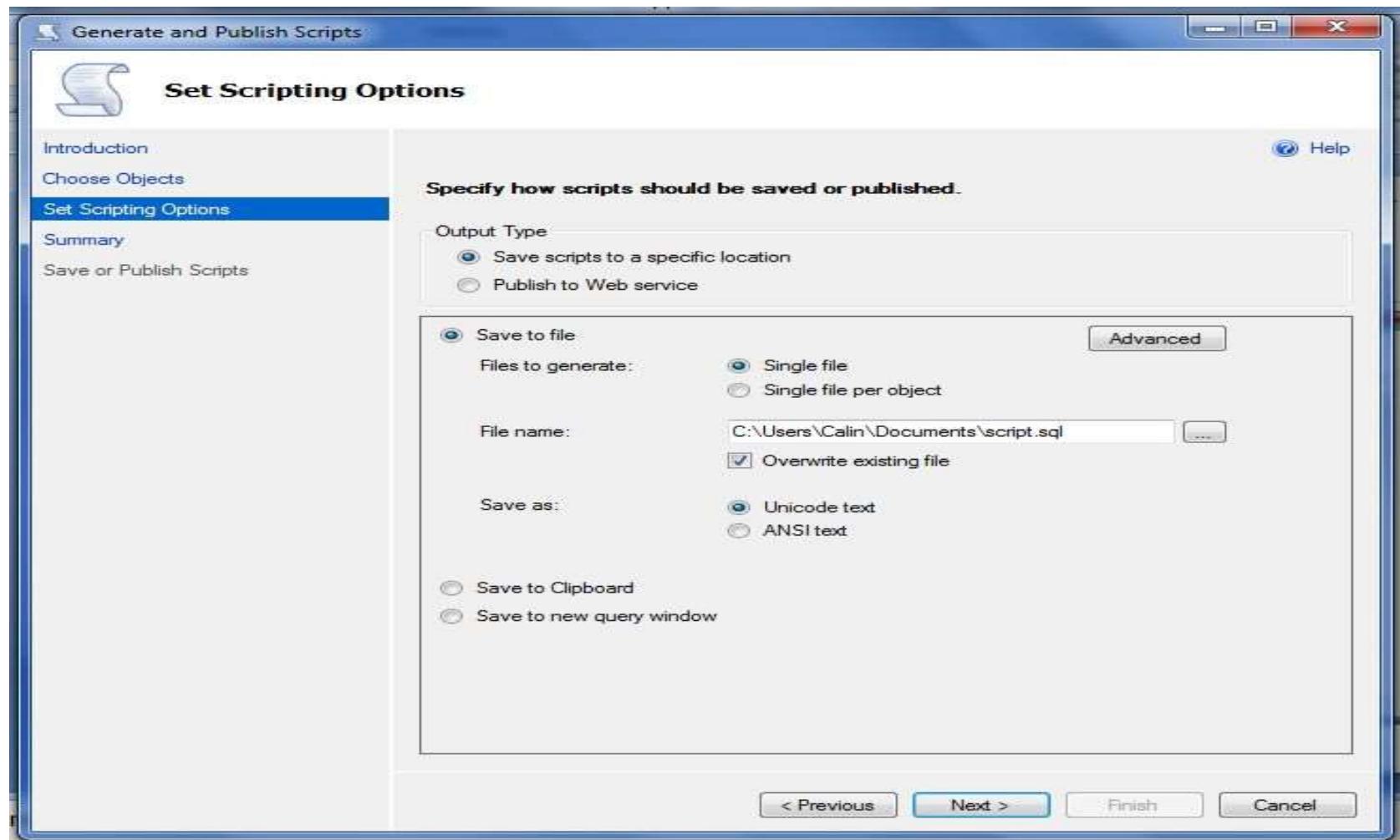




Lab Notes 09 MS SQL Server Database



Lab Notes 09 MS SQL Server Database



# Script entire database and all database objects

# DataBases

Relational Model -  
Relational Algebra @ Calculus

- Suppose you are given a relation  $R$  with four attributes  $ABCD$ . (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.
- 1.  $C \rightarrow D, C \rightarrow A, B \rightarrow C$
- 2.  $B \rightarrow C, D \rightarrow A$

# Objectives

- Explain why relational database model became practical in about 1980 and de facto standard today
- Define basic relational database terms such as relation and tuple
  - Describe major types of keys including primary, candidate, and foreign
  - Describe how one-to-one, one-to-many, and many-to-many relationships are implemented

# Objectives

- Describe how relational data retrieval is accomplished using the relational algebra select, project, and join operators
- Understand how the join operator facilitates data integration in relational database
- Describe also relational calculus

# Relational model

- Professionals in any discipline need to know the foundations of their field
- Database Professional - need to know theory of relations
- is not product-specific;
  - rather, it is concerned with principles

# Principles, not Products

- principle
  - source, root, origin; which is fundamental; essential nature; theoretical basis
- principles endure
- by contrast, products and technologies, change all the time

# Relation Model

- E. F. Codd, at the time researcher at IBM; late in 1968 that Codd, a mathematician, first realized that discipline of mathematics could be used to inject some solid principles and rigor into field of database

# original definition of relational model

- E. F. Codd
  - *Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks*, IBM Research Report RJ599, 1969.
  - *A Relational Model of Data for Large Shared Data Banks*, CACM 13, No. 6, 1970.
- C. J. Date
  - *An Introduction to Database Systems*, 8<sup>th</sup> Ed. Boston, Mass.: Addison-Wesley, 2004.
  - first edition of book was published in 1975

# Historical perspective

- Relational Database Management System (RDBMS) has become dominant data-processing software in use today, with an estimated total software revenue worldwide of US\$ 20-30 billion
- software represents second generation of DBMSs based on proposed *relational data model* in 1970, Edgar Codd, at IBM's San Jose Research Laboratory

# Historical perspective

- in relational model, all data is logically structured within relations (tables); each ***relation*** has name and is made up of named ***attributes*** (columns) of data; each ***tuple*** (row) contains one value per attribute
- great strength of relational model is this simple logical structure; behind this structure is sound theoretical foundation that is lacking in first generation of DBMSs (network & hierarchical)

# Mathematics

- always strive for elegance and orthogonality - dislike exceptions
- system is said to be designed in an *orthogonal* way if its set of components that together make up whole system capability are *non-overlapping* and *mutually independent*
  - each capability should be implemented by only one component
  - one component should only implement one capability of system

# Mathematics

- well-separated and independent components ensure that there are no side effects: using or even changing one component does not cause side effects in another area of system
- formal disciplines, most relevant ones in application of mathematics to field of databases
- Set theory

# Mathematics

- *Everything should be made as simple as possible, but not simpler.*
  - Albert Einstein (1879–1955)

# Codd relational model's objectives

- to allow high degree of data independence; application programs must not be affected by modifications to internal data representation, particularly by changes to file organizations, record orderings, or access paths
- to provide substantial grounds for dealing with data semantics, consistency, and redundancy problems; introduced concept of ***normalized*** relations that have no repeating groups
- to enable expansion of set-oriented data manipulation languages

# 1 - Prototype relational DBMS System R

- at IBM's San José Research Laboratory in California - prototype relational DBMS System R, which was developed during the late 1970s
- project was designed to prove practicality of relational model by providing an implementation of its data structures and operations
- proved to be excellent source of information about implementation concerns such as transaction management, concurrency control, recovery techniques, query optimization, data security and integrity, human factors, and user interfaces,

# led to two major developments

- development of a structured query language called SQL, which has since become formal International Organization for Standardization (ISO) and de facto standard language for RDBMS
- production of various commercial relational DBMS products during late 1970s and 1980s: IBM DB2 and Oracle

## 2 - INGRES

- development of relational model INGRES (Interactive Graphics Retrieval System) project at University of California at Berkeley
- involved development of prototype RDBMS, with research concentrating on same overall objectives as System R project - led to an academic version of INGRES, which contributed to general appreciation of relational concepts, and spawned commercial products INGRES

# Terminology and structural concepts of relational model

- relational model is based on mathematical concept of *relation*, which is physically represented as **table**
- we explain terminology and structural concepts of the relational model

# Relation

- is a table with columns and rows
- RDBMS requires only that database be perceived by user as tables
- perception applies only to logical structure of database: external and conceptual levels of ANSI-SPARC architecture
- does not apply to physical structure of database, which can be implemented using variety of storage structures

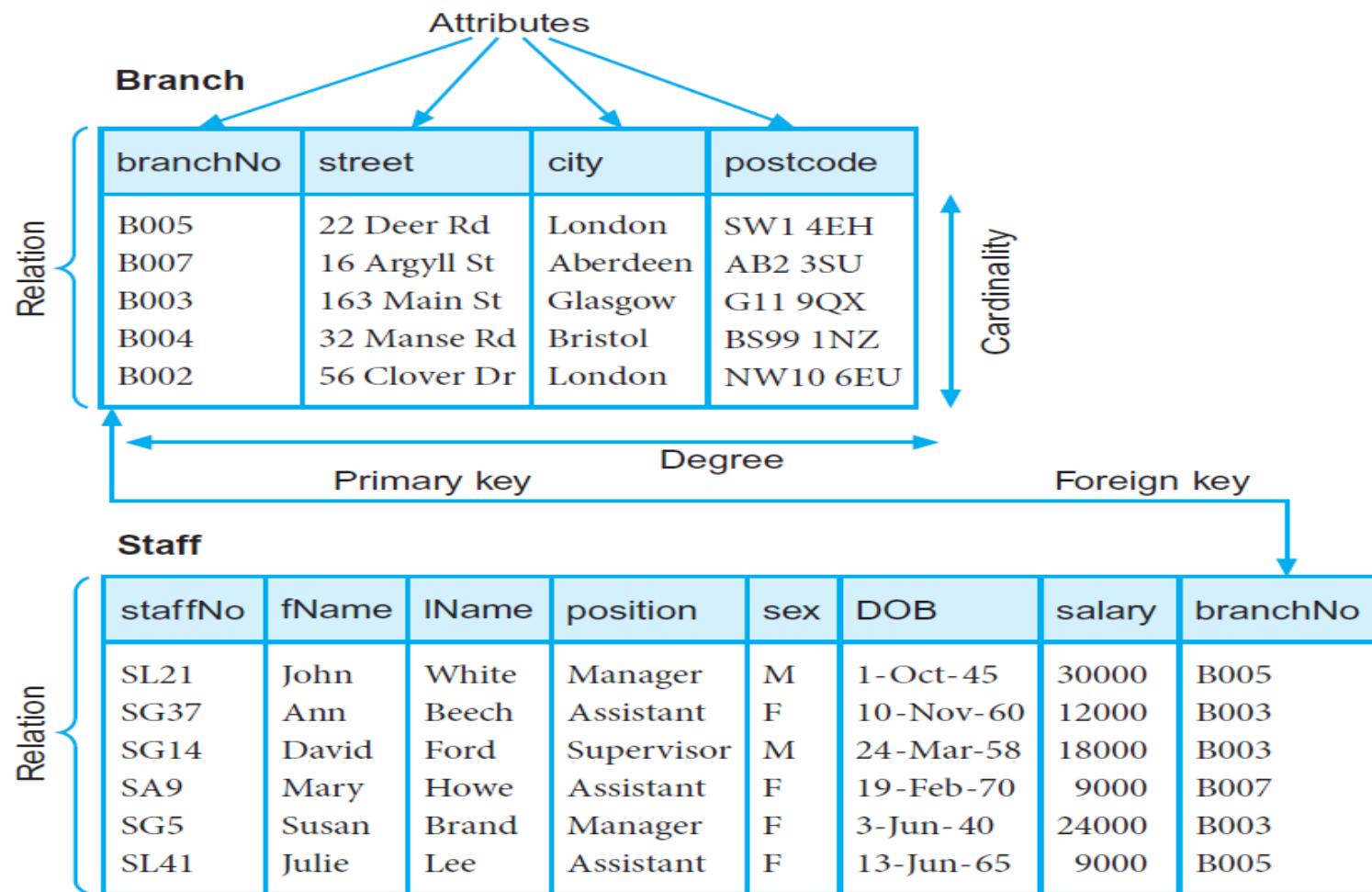
# Attribute

- is named column of relation
- relations used to hold information about objects to be represented in database as a two-dimensional table in which rows correspond to individual records and columns correspond to attributes
- attributes can appear in any order and relation will still be the same - therefore will convey same meaning

# Domain

- is set of allowable (possible) values for one or more attributes
- may be distinct for each attribute, or two or more attributes may be defined on same domain
- typically there will be values in domain that do not currently appear as values in corresponding attribute

# Instances of Branch & Staff relation



# Domains for some attributes

| Attribute | Domain Name   | Meaning                                | Domain Definition                              |
|-----------|---------------|----------------------------------------|------------------------------------------------|
| branchNo  | BranchNumbers | The set of all possible branch numbers | character: size 4, range B001-B999             |
| street    | StreetNames   | The set of all street names in Britain | character: size 25                             |
| city      | CityNames     | The set of all city names in Britain   | character: size 15                             |
| postcode  | Postcodes     | The set of all postcodes in Britain    | character: size 8                              |
| sex       | Sex           | The sex of a person                    | character: size 1, value M or F                |
| DOB       | DatesOfBirth  | Possible values of staff birth dates   | date, range from 1-Jan-20,<br>format dd-mmm-yy |
| salary    | Salaries      | Possible values of staff salaries      | monetary: 7 digits, range<br>6000.00-40000.00  |

# tuple

- is row of relation
- elements of relation are rows or tuples in table
- tuples can appear in any order and relation will still be the same - therefore convey same meaning
- structure of relation, together with specification of domains and any other restrictions on possible values, is called its intension; usually fixed - tuples are called extension (or state) of relation, which changes over time

# Degree

- is number of attributes it contains
- the Branch relation has four attributes or degree four – means that each row of table is a four-tuple, containing four values
- degree of a relation is property of *intension* of relation

# Cardinality

- is number of tuples it contains
- changes as tuples are added or deleted; is property of *extension* of relation and is determined from particular instance of relation at any given moment

# Relational database

- collection of normalized relations with distinct relation names
- consists of relations that are appropriately structured - refer to this as *normalization*

# Alternative terminology

- third set of terms is sometimes used;  
stems from fact that, physically, RDBMS  
may store each relation in a file

# Alternative terminology

| Formal term | Logical | Physical |
|-------------|---------|----------|
| Relation    | Table   | File     |
| Attribute   | Column  | Field    |
| Tuple       | Row     | Record   |

# MATHEMATICAL RELATIONS

# Sets and Elements

- set is a collection of objects, and those objects are called the elements of the set
- set is fully characterized by its distinct elements, and nothing else but these elements

# Sets and Elements

- elements of set don't have any ordering
- sets don't contain duplicate elements
- two sets A and B, are the same if each element of A is also an element of B and each element of B is also an element of A
- $\emptyset$  the empty set

# Sets and Elements

- set theory has a mathematical symbol  $\in$  that is used to state that some object is an element of some set
- $x \in S$
- $x \notin S \leftrightarrow \neg(x \in S)$

# Methods to specify sets

- enumerative method
  - $S1 := \{1, 2, 3, 5, 7, 11, 13, 17, 19, 23\}$
- predicative method
  - $S2 := \{x \in S \mid P(x)\}$ 
    - let  $P(x)$  be a given predicate with exactly one parameter named  $x$  of type  $S$
- substitutive method
  - $S3 := \{x^2 \mid x \in N\}$

# Cardinality

- sets can be finite and infinite
- when dealing with databases all your sets are finite by definition
- for every finite set  $S$ , cardinality is defined as number of elements of  $S$

# Subsets, Supersets

- A is a subset of B (B is a superset of A) if every element of A is an element of B
- $A \subseteq B$
- A is proper subset of B (B is proper superset of A) if A is subset of B and  $A \neq B$
- $A \subset B$

# Union, Intersection, Difference

- union  $A \cup B = \{x \mid x \in A \vee x \in B\}$
- intersection  $A \cap B = \{x \mid x \in A \wedge x \in B\}$
- difference  $A - B = \{x \mid x \in A \wedge x \notin B\}$

# (Binary) Relation

- two sets,  $D_1$  and  $D_2$ , where  $D_1 = \{2, 4\}$  and  $D_2 = \{1, 3, 5\}$ ; Cartesian product of these two sets, written  $D_1 \times D_2$ , is set of all ordered pairs such that first element is a member of  $D_1$  and second element is a member of  $D_2$  -  $D_1 \times D_2 = \{(2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5)\}$
- any subset of this Cartesian product is a relation

# Example of relation $R$

- $R = \{(2, 1), (4, 1)\}$
- $R = \{(x, y) \mid x \in D_1, y \in D_2, \text{ and } y = 1\}$
- easily extend notion to three sets; let  $D_1$ ,  $D_2$ , and  $D_3$  be sets; Cartesian product  $D_1 \times D_2 \times D_3$  of these three sets is set of all ordered triples such that first element is from  $D_1$ , second is from  $D_2$  and third element is from  $D_3$
- any subset of Cartesian product is relation

# N-ary Relation

- order of coordinates of ordered pair is important
- order of tuples it is not important – it is subset of Cartesian – subset of a set is a set – order of element in a set it is not important
- can extend three sets to a more general concept of n-ary relation

# Relation

- define general relation on  $n$  domains
- let  $D_1, D_2, \dots, D_n$  be  $n$  sets
- Cartesian product is defined as:
- $D_1 \times D_2 \times \dots \times D_n = \{(d_1, d_2, \dots, d_n) \mid d_1 \in D_1, d_2 \in D_2, \dots, d_n \in D_n\}$  and is usually written as  $\prod_{i=1}^n D_i$
- any set of  $n$ -tuples from this Cartesian product is relation on  $n$  sets

# DataBase Relations

- note that in defining these relations we have to specify sets, or domains, from which we choose values
- Relation schema: named relation defined by set of attribute and domain name pairs
- let  $A_1, A_2, \dots, A_n$  be attributes with domains  $D_1, D_2, \dots, D_n$
- then set  $\{A_1:D_1, A_2:D_2, \dots, A_n:D_n\}$  is relation schema

# DataBase Relations

- relation  $R$  defined by relation schema is set of mappings from attribute names to their corresponding domains
- relation  $R$  is set of  $n$ -tuples:
- $(A_1:d_1, A_2:d_2, \dots, A_n:d_n)$  such that  $d_1 \in D_1$ ,  $d_2 \in D_2, \dots, d_n \in D_n$
- each element in  $n$ -tuple consists of attribute and value for that attribute

# DataBase Relations

- normally we list attribute names as column headings and write out tuples as rows having form  $(d_1, d_2, \dots, d_n)$ , where each value is taken from appropriate domain
- Relation is any subset of Cartesian product of domains of attributes
- table is simply physical representation of such relation

# Branch and Staff relations

## Branch

| branchNo | street       | city     | postCode |
|----------|--------------|----------|----------|
| B005     | 22 Deer Rd   | London   | SW1 4EH  |
| B007     | 16 Argyll St | Aberdeen | AB2 3SU  |
| B003     | 163 Main St  | Glasgow  | G11 9QX  |
| B004     | 32 Manse Rd  | Bristol  | BS99 1NZ |
| B002     | 56 Clover Dr | London   | NW10 6EU |

## Staff

| staffNo | fName | IName | position   | sex | DOB       | salary | branchNo |
|---------|-------|-------|------------|-----|-----------|--------|----------|
| SL21    | John  | White | Manager    | M   | 1-Oct-45  | 30000  | B005     |
| SG37    | Ann   | Beech | Assistant  | F   | 10-Nov-60 | 12000  | B003     |
| SG14    | David | Ford  | Supervisor | M   | 24-Mar-58 | 18000  | B003     |
| SA9     | Mary  | Howe  | Assistant  | F   | 19-Feb-70 | 9000   | B007     |
| SG5     | Susan | Brand | Manager    | F   | 3-Jun-40  | 24000  | B003     |
| SL41    | Julie | Lee   | Assistant  | F   | 13-Jun-65 | 9000   | B005     |

- it shows for example that employee John White is manager with salary of 30,000, who works at branch (branchNo) B005, which, from first table, is at 22 Deer Rd in London
- there is relationship between Staff and Branch: branch office *has* staff
- there is no explicit link between these two tables; it is only by knowing that attribute branchNo in Staff have same value as branchNo of Branch

# Branch relation

- has attributes branchNo, street, city, and postcode, each with its domain
- is any subset of Cartesian product of domains, or any set of four-tuples in which first element is from domain BranchNumbers, second is from domain StreetNames, ...

# Branch relation instance

- one of tuples is: or more correctly
- $\{(B005, 22\text{ Deer Rd, London, SW1 4EH})\}$
- $\{(branchNo: B005, street: 22\text{ Deer Rd, city: London, postcode: SW1 4EH})\}$
- we refer to this as *relation instance*
- Branch table is convenient way of writing out all tuples that form relation at specific moment in time

# Relational database schema

- set of relation schemas, each with distinct name
- in same way that relation has schema, so too does relational database
- if  $R_1, R_2, \dots, R_n$  are set of relation schemas, then we can write *relational database schema*, or simply *relational schema*,  $R$ , as:  $R = \{R_1, R_2, \dots, R_n\}$

# Properties of Relations

- relation has name that is distinct from all other relation names in relational schema
- each cell of relation contains exactly one atomic (single) value; relations do not contain repeating groups
- relation that satisfies this property is said to be *normalized* in *first normal form*

# Properties of Relations

- each attribute has distinct name
- value of attribute are all from same domain
- each tuple is distinct; there are no duplicate tuples
- order of attributes has no significance
- order of tuples has no significance
- most of properties specified result from properties of mathematical relations

# Relational Keys

- ... there are no duplicate tuples ...
- therefore, we need to identify one or more attributes (*relational keys*) that uniquely identifies each tuple in relation

# Superkey

- attribute, or set of attributes, that uniquely identifies tuple within relation
- may contain additional attributes that are not necessary for unique identification
- interested in identifying superkeys that contain only minimum number of attributes necessary for unique identification

# Candidate Key

- superkey such that no proper subset is superkey within relation
- *Uniqueness* – in each tuple of  $R$ , values of  $K$  uniquely identify that tuple
- *Irreducibility* - no proper subset of  $K$  has uniqueness property
- when key consists of more than one attribute, we call it *composite key*
- there may be several candidate keys for relation

# Candidate Key

- consider Branch relation
- given value of city, we can determine several branch offices (London has two branch offices)
- attribute cannot be a candidate key
- company allocates each branch office unique branch number we can determine at most one tuple, so that *branchNo* is candidate key
- similarly, postcode is also candidate key

# Candidate Key

- consider relation *Viewing*, which contains information relating to properties viewed by clients
- relation comprises client number (*clientNo*), property number (*propertyNo*), date of viewing (*viewDate*) and, optionally, comment (*comment*)

# Candidate Key

- combination of *clientNo* and *propertyNo* identifies at most one tuple, so for *Viewing* together form (composite) candidate key
- if we take into account possibility that client may view property more than once, then we could add *viewDate* to composite key

# Candidate Key

- instance of relation cannot be used to prove that attribute or combination of attributes is CK
- fact that there are no duplicates for values that appear at particular moment in time does not guarantee that duplicates are not possible
- presence of duplicates in instance can be used to show that attribute combination is not CK
- identifying candidate key requires that we know the “real-world” meaning of attribute(s) involved; can decide whether duplicates are possible

# Primary Key

- candidate key that is selected to identify tuples uniquely within relation (by database designer)
- relation has no duplicate tuples; always possible to identify each row uniquely
- relation always has primary key
- in worst case, entire set of attributes could serve as primary key; usually some smaller subset is sufficient to distinguish tuples

# Alternate Key

- candidate keys that are not selected to be primary key
- for *Branch* relation, if we choose *branchNo* as primary key, *postcode* would then be an alternate key
- for *Viewing* relation, there is only one candidate key, comprising *clientNo* and *propertyNo*, so these attributes would automatically form primary key

# Foreign Key

- attribute, or set of attributes, within one relation that matches candidate key of some (possibly same) relation
- when attribute appears in more than one relation, its appearance usually represents relationship between tuples of two relations
- for example, inclusion of *branchNo* in both *Branch* and *Staff* relations is quite deliberate and links each branch to details of staff working at that branch

# Foreign Key

- in *Branch* relation, *branchNo* is primary key
- in *Staff* relation, *branchNo* attribute exists to match staff to branch office they work in
- in *Staff* relation, *branchNo* is *foreign key*
- we say that attribute *branchNo* in *Staff* relation *targets* primary key attribute *branchNo* in home relation, *Branch*

# Representing Relational Database Schemas

- *Branch (branchNo, street, city, postcode)*
- *Staff (staffNo, fName, lName, position, sex, DOB, salary, branchNo)*
- *PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)*
- *Client (clientNo, fName, lName, telNo, prefType, maxRent, eMail)*

# Representing Relational Database Schemas

- *PrivateOwner* (ownerNo, fName, lName, address, telNo, eMail, password)
- *Viewing* (clientNo, propertyNo, viewDate, comment)
- *Registration* (clientNo, branchNo, staffNo, dateJoined)

# Representing Relational Database Schemas

- common convention for representing relation schema is to give name of relation followed by attribute names in parentheses; primary key is underlined
- *conceptual model*, or *conceptual schema*, is set of all such schemas for database

# Instance of rental database

**Branch**

| branchNo | street       | city     | postcode |
|----------|--------------|----------|----------|
| B005     | 22 Deer Rd   | London   | SW1 4EH  |
| B007     | 16 Argyll St | Aberdeen | AB2 3SU  |
| B003     | 163 Main St  | Glasgow  | G11 9QX  |
| B004     | 32 Manse Rd  | Bristol  | BS99 1NZ |
| B002     | 56 Clover Dr | London   | NW10 6EU |

**Staff**

| staffNo | fName | IName | position   | sex | DOB       | salary | branchNo |
|---------|-------|-------|------------|-----|-----------|--------|----------|
| SL21    | John  | White | Manager    | M   | 1-Oct-45  | 30000  | B005     |
| SG37    | Ann   | Beech | Assistant  | F   | 10-Nov-60 | 12000  | B003     |
| SG14    | David | Ford  | Supervisor | M   | 24-Mar-58 | 18000  | B003     |
| SA9     | Mary  | Howe  | Assistant  | F   | 19-Feb-70 | 9000   | B007     |
| SG5     | Susan | Brand | Manager    | F   | 3-Jun-40  | 24000  | B003     |
| SL41    | Julie | Lee   | Assistant  | F   | 13-Jun-65 | 9000   | B005     |

**PropertyForRent**

| propertyNo | street        | city     | postcode | type  | rooms | rent | ownerNo | staffNo | branchNo |
|------------|---------------|----------|----------|-------|-------|------|---------|---------|----------|
| PA14       | 16 Holhead    | Aberdeen | AB7 5SU  | House | 6     | 650  | CO46    | SA9     | B007     |
| PL94       | 6 Argyll St   | London   | NW2      | Flat  | 4     | 400  | CO87    | SL41    | B005     |
| PG4        | 6 Lawrence St | Glasgow  | G11 9QX  | Flat  | 3     | 350  | CO40    |         | B003     |
| PG36       | 2 Manor Rd    | Glasgow  | G32 4QX  | Flat  | 3     | 375  | CO93    | SG37    | B003     |
| PG21       | 18 Dale Rd    | Glasgow  | G12      | House | 5     | 600  | CO87    | SG37    | B003     |
| PG16       | 5 Novar Dr    | Glasgow  | G12 9AX  | Flat  | 4     | 450  | CO93    | SG14    | B003     |

# Instance of rental database

**Client**

| clientNo | fName | IName   | telNo         | prefType | maxRent | eMail                  |
|----------|-------|---------|---------------|----------|---------|------------------------|
| CR76     | John  | Kay     | 0207-774-5632 | Flat     | 425     | john.kay@gmail.com     |
| CR56     | Aline | Stewart | 0141-848-1825 | Flat     | 350     | astewart@hotmail.com   |
| CR74     | Mike  | Ritchie | 01475-392178  | House    | 750     | mritchie01@yahoo.co.uk |
| CR62     | Mary  | Tregear | 01224-196720  | Flat     | 600     | maryt@hotmail.co.uk    |

**PrivateOwner**

| ownerNo | fName | IName  | address                       | telNo         | eMail             | password |
|---------|-------|--------|-------------------------------|---------------|-------------------|----------|
| CO46    | Joe   | Keogh  | 2 Fergus Dr, Aberdeen AB2 7SX | 01224-861212  | jkeogh@lhh.com    | *****    |
| CO87    | Carol | Farrel | 6 Achray St, Glasgow G32 9DX  | 0141-357-7419 | cfarrel@gmail.com | *****    |
| CO40    | Tina  | Murphy | 63 Well St, Glasgow G42       | 0141-943-1728 | tinam@hotmail.com | *****    |
| CO93    | Tony  | Shaw   | 12 Park Pl, Glasgow G4 0QR    | 0141-225-7025 | tony.shaw@ark.com | *****    |

**Viewing**

| clientNo | propertyNo | viewDate  | comment        |
|----------|------------|-----------|----------------|
| CR56     | PA14       | 24-May-13 | too small      |
| CR76     | PG4        | 20-Apr-13 | too remote     |
| CR56     | PG4        | 26-May-13 |                |
| CR62     | PA14       | 14-May-13 | no dining room |
| CR56     | PG36       | 28-Apr-13 |                |

**Registration**

| clientNo | branchNo | staffNo | dateJoined |
|----------|----------|---------|------------|
| CR76     | B005     | SL41    | 2-Jan-13   |
| CR56     | B003     | SG37    | 11-Apr-12  |
| CR74     | B003     | SG37    | 16-Nov-11  |
| CR62     | B007     | SA9     | 7-Mar-12   |

# Null

- represents value for attribute that is currently unknown or is not applicable for this tuple
- mean logical value “unknown”, value is not applicable to particular tuple, or that no value has yet been supplied
- way to deal with incomplete or exceptional data
- not same as zero or empty text string
- represents absence of value

# Null

- for example, in Viewing relation *comment* attribute may be undefined until potential renter has visited property and returned comment to agency
- without nulls, it becomes necessary to introduce false data to represent this state

# Null

- can cause implementation problems, arising from fact that relational model is based on first-order predicate calculus, which is two-valued or Boolean logic; allowing nulls means that we have to work with higher-valued logic, such as three-valued logic (true, false, null)
- Codd regarded nulls as an integral part of relational model

# Integrity Constraints

- relational integrity constraints ensure that data is accurate
- because every attribute has an associated domain, there are ***domain constraints*** that form restrictions on set of values allowed for attributes of relations
- ***integrity rules*** - constraints that apply to all instances of database: ***entity integrity*** and ***referential integrity***

# Integrity Constraints

- because every attribute has an associated domain, there are ***domain constraints*** that form restrictions on set of values allowed for attributes of relations
- ***integrity rules*** - constraints that apply to all instances of database: ***entity integrity*** and ***referential integrity***

# Entity integrity

- in base relation, no attribute of primary key can be null
- PK is minimal identifier used to identify tuples uniquely; if we allow null for any part of PK we are implying that not all attributes are needed
- for example, *branchNo* is PK of *Branch*, we should not be able to insert tuple with null
- Viewing (clientNo, propertyNo, ...) we should not be able to insert tuple into Viewing with null for clientNo or null for propertyNo or nulls for both

# Entity integrity

- using data of *Viewing* relation consider query  
“List all comments from viewings”
- produce unary relation consisting of attribute comment
- this attribute must be PK, but it contains nulls
- this relation is not base relation (derived relation), model allows primary key to contain nulls

# Referential integrity

- if foreign key exists in relation, either foreign key value must match a candidate key value of some tuple in its home relation or foreign key value must be wholly null

# Referential integrity

- for example, *branchNo* in *Staff* is FK targeting *branchNo* attribute in home relation, *Branch*
- possible to create *Staff* record with *branchNo* B025, if there is already *branchNo* B025 in *Branch* relation
- we should be able to create *Staff* record with null *branchNo* to allow for situation where member of staff has joined company but has not yet been assigned to particular branch office

# General constraints

- additional rules specified by users of database that define or constrain some aspect of organization
- for example, if an upper limit of 20 has been placed upon number of staff that may work at branch office
- level of support for general constraints varies from system to system

# General constraints

- *declarative integrity constraint* is statement about database that is always true
- *trigger* is procedural code that is automatically executed in response to certain events on particular table or view in database; mostly used for maintaining integrity of information

# Views

- in relational model has slightly different meaning than view from architecture
- rather than being entire external model of user's view, view is *virtual* or *derived relation* that does not necessarily exist in its own right, but may be dynamically derived from one or more base relations
- external model can consist of both base (conceptual-level) relations and views derived from base relations

# Views

- relation that appears to user to exist, can be manipulated as if it were base relation, but does not necessarily exist in storage (although its definition is stored in system catalog)
- contents of view are defined as query on one or more base relations
- operations on view are automatically translated into operations on relations from which it is derived

# Views

- dynamic – changes made to base relations that affect view are immediately reflected in view
- when users make permitted changes to (updating) view, these changes are made to underlying relations

# Views

- provides powerful and flexible security mechanism by hiding parts of database from certain users; users are not aware of existence of any attributes or tuples that are missing
- it permits users to access data in way that is customized to their needs, same data can be seen by different users in different ways
- it can simplify complex operations on base relations

# Views

- for example, if view defined as combination (join) of two relations users may now perform more simple operations on view, which will be translated by DBMS into equivalent operations
- some members of staff should see *Staff* tuples without salary attribute
- attributes may be renamed or order changed; user accustomed to calling *branchNo* attribute of branches by full name

# Views

- provides *logical data independence*;  
supports reorganization of conceptual schema
- for example, if new attribute is added to relation,  
existing users can be unaware of its existence if  
their views are defined to exclude it; if existing  
relation is rearranged or split up, view may be  
defined so that users can continue to see their  
original views

# Codd's rules

- Codd came up with rules database must obey in order to be regarded as relational
- hardly any commercial product follows all

# Rule 0: Foundation

- system must qualify as relational, as a database, and as a management system
- system must use its relational facilities (exclusively) to manage database
- foundation rule, which acts as base for all other 12 rules derive from this

# Rule 1: Information

- all information in database is to be represented in one and only one way, namely by values in column positions within rows of tables
- data stored in database, may it be user data or metadata, must be value of some table cell; everything in database must be stored in a table format

# Rule 2: Guaranteed Access

- all data must be accessible; essentially restatement of fundamental requirement for primary keys; every individual scalar value in database must be logically addressable by specifying name of table, name of column and primary key value
- every single data element (value) is guaranteed to be accessible logically with combination of table-name, primary-key (row), and attribute-name (column); no other means, such as pointers, can be used to access data

# Rule 3: Systematic Treatment of NULL Values

- DBMS must allow each field to remain null (or empty); must support representation of "missing information and inapplicable information" that is systematic, distinct from all regular values (0, "", ...), and independent of data type; manipulated by DBMS in systematic way
- NULL values in database must be given systematic and uniform treatment; interpreted as
  - data is missing, data is not known, or data is not applicable

# Rule 4: Active Online Catalog

- system must support online relational catalog that is accessible to authorized users by means of their regular query language; users must be able to access database's structure (catalog) using same query language that they use to access database's data
- structure description of entire database must be stored in online catalog, known as *data dictionary*, which can be accessed by authorized users; users can use same query language to access catalog which they use to access database itself

# Rule 5: Comprehensive Data Sub-Language

- system must support at least one relational language that
  - has linear syntax
  - can be used both interactively and within application programs
  - supports data definition operations (including view definitions), data manipulation operations (update and retrieval), security and integrity constraints, transaction management operations (begin, commit, and rollback)

# Rule 5: Comprehensive Data Sub-Language

- database can only be accessed using language having linear syntax that supports data definition, data manipulation, and transaction management operations; language can be used directly or by means of some application
- if database allows access to data without any help of this language, then it is considered violation

# Rule 6: View Updating

- all views those can be updated theoretically, must be updated by system.
- all views of database, which can theoretically be updated, must also be updatable by system

# Rule 7: High-Level Insert, Update, and Delete

- system must support set-at-a-time insert, update, and delete operators; data can be retrieved from relational database in sets constructed of data from multiple rows and/or multiple tables; insert, update, and delete operations should be supported for any retrievable set rather than just for single row in single table

# Rule 7: High-Level Insert, Update, and Delete

- database must support high-level insertion, updation, and deletion; this must not be limited to single row, that is, it must also support union, intersection and minus operations to yield sets of data records

# Rule 8: Physical Data Independence

- changes to physical level (how data is stored, whether in arrays or linked lists etc.) must not require change to application based on structure
- data stored in database must be independent of applications that access database; any change in physical structure of database must not have any impact on how data is being accessed by external applications

# Rule 9: Logical Data Independence

- changes to logical level (tables, columns, rows, ...) must not require change to application based on structure; more difficult to achieve
- logical data in database must be independent of its user's view (application); any change in logical data must not affect applications using it

# Rule 10: Integrity Independence

- integrity constraints must be specified separately from application programs and stored in catalog; it must be possible to change such constraints as and when appropriate without unnecessarily affecting existing applications
- database must be independent of application that uses it; all its integrity constraints can be independently modified without need of any change in application; makes database independent of front-end application & interface

# Rule 11: Distribution Independence

- distribution of portions of database to various locations should be invisible to users; existing applications should continue to operate
  - when distributed version is first introduced
  - when existing distributed data are redistributed
- end-user must not be able to see that data is distributed over various locations; users should always get impression that data is located at one site only;
- regarded as foundation of distributed database

# Rule 12: Non-Subversion Rule

- if system provides low-level (record-at-a-time) interface, then that interface cannot be used to subvert system; bypassing relational security or integrity constraint
- if system has an interface that provides access to low-level records, then interface must not be able to subvert system and bypass security and integrity constraints

# Relational model

## three major components

- structure
- integrity
- manipulation

# Relation

- are defined over domains, (data types)
- domain set of values from which actual attributes in actual relations take their actual values
- n-ary relations can be pictured as table with  $n$  columns
  - columns correspond to attributes of relation
  - rows correspond to tuples

# Departments and Employees database sample values

DEPT

| DNO | DNAME       | BUDGET |
|-----|-------------|--------|
| D1  | Marketing   | 10M    |
| D2  | Development | 12M    |
| D3  | Research    | 5M     |

EMP

| ENO | ENAME | DNO | SALARY |
|-----|-------|-----|--------|
| E1  | Lopez | D1  | 40K    |
| E2  | Cheng | D1  | 42K    |
| E3  | Finzi | D2  | 30K    |
| E4  | Saito | D2  | 35K    |



DEPT.DNO referenced by EMP.DNO

# Keys

- every relation has at least one *candidate key* - unique identifier
- combination of attributes such that every tuple in relation has unique value for combination in question
- *primary key* is candidate key that's been singled out for special treatment

# Keys

- candidate keys, not primary keys, are significant from relational point of view
- *foreign key* is set of attributes in one relation whose values are required to match values of some candidate key in some other relation (or possibly in same relation)

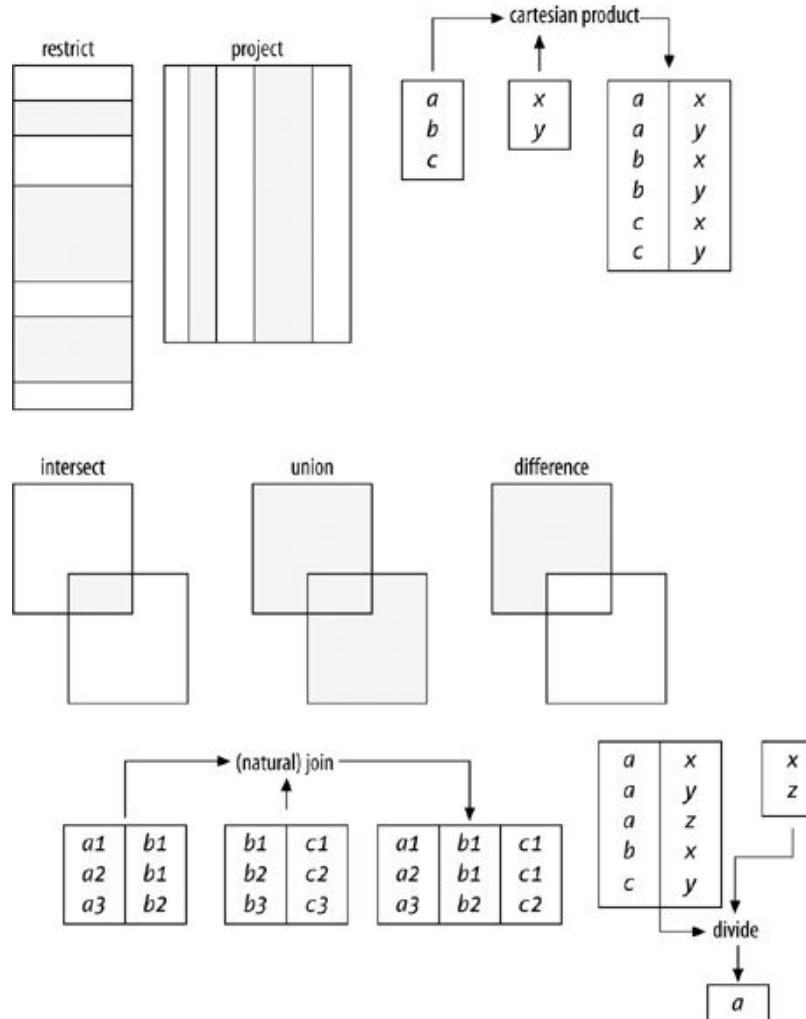
# Integrity Constraint

- basically just boolean expression that must evaluate to TRUE
  - in case of departments and employees, for example, we might have constraint to effect that SALARY values must be greater than 0
- Entity integrity
  - primary key attributes don't permit nulls.
- Referential integrity
  - there mustn't be any unmatched foreign key values
- null is "marker" that means value unknown

# Manipulation

- set of relational operators, collectively called relational algebra
- relational assignment operator

# Relational Algebra



# Properties of Relations

- every relation has *heading* and *body*
- heading is set of attributes (name, type pair)
- body is set of tuples that conform to that heading
- relation doesn't really contain tuples, it contains body, and that body in turn contains tuples

# Properties of Relations

- number of attributes in heading is the *degree (arity)*, and number of tuples in body is the *cardinality*
- tuples of relation are unordered; follows because body is set, and sets in mathematics have no ordering to their elements
- attributes of relation are also unordered because heading too is mathematical set

# Properties of Relations

- relations never contain duplicate tuples;  
follows because body is set of tuples, and  
sets in mathematics do not contain  
duplicate elements
- relational operations always produce result  
without duplicate tuples
- SQL results are allowed to contain  
duplicate rows

# Properties of Relations

- relations are always normalized (1NF); at every row-and-column intersection we always see just single value

# Tuple

- Let  $T_1, T_2, \dots, T_n$ , ( $n > 0$ ) be type names, not necessarily all distinct.
- Associate with each  $T_i$  a distinct attribute name,  $A_i$ ; each of the  $n$  attribute-name : type-name combinations that results is an attribute.
- Associate with each attribute a value  $v_i$  of type  $T_i$ ; each of the  $n$  attribute : value combinations that results is component.
- set of all  $n$  components thus defined,  $t$  say, is tuple value (or just tuple for short) over the attributes  $A_1, A_2, \dots, A_n$
- value  $n$  is the degree of  $t$ ;
- set of all  $n$  attributes is the heading of  $t$

# Relation

- Let  $\{H\}$  be a tuple heading and let  $t_1, t_2, \dots, t_n$ , ( $n > 0$ ) be distinct tuples with heading  $\{H\}$ .
- The combination,  $r$  say, of  $\{H\}$  and the set of tuples  $\{t_1, t_2, \dots, t_n\}$  is a relation value (or just relation for short) over attributes  $A_1, A_2, \dots, A_n$ , where  $A_1, A_2, \dots, A_n$  are the attributes in  $\{H\}$ .
- heading of  $r$  is  $\{H\}$
- $r$  has same attributes that heading does
- body of  $r$  is set of tuples  $\{t_1, t_2, \dots, t_n\}$

- theory and practice of relational database
- treatment of the theory
- rigorous and mathematical
- presents two formal query languages associated with relational model
- query languages are specialized languages for asking questions that involve data in a database

- convinced that familiarity with areas of mathematics that will be presented and on which the relational model of data is based, is a strong prerequisite for anybody who aims to be professionally involved with databases

# Mathematics

- always strive for elegance and orthogonality - dislike exceptions
- system is said to be designed in an *orthogonal* way if its set of components that together make up whole system capability are *non-overlapping* and *mutually independent*
  - each capability should be implemented by only one component
  - one component should only implement one capability of system

# Mathematics

- well-separated and independent components ensure that there are no side effects: using or even changing one component does not cause side effects in another area of system
- two formal disciplines, most relevant ones in application of mathematics to field of databases
- Logic
- Set theory

# Mathematics

- *Everything should be made as simple as possible, but not simpler.*
  - Albert Einstein (1879–1955)

# **RELATIONS**

# Relation

- common misconception that word relational in “relational model of data, relational database” refers to relationships” (many-to-one, many-to-many, and so on) that can exist between different entity types
- nothing to do with R of ERD
- Entity Relationship Diagrams

# Relation

- word refers to mathematical concept of a (n-ary) relation, mathematical, set-theory concept
- **Binary Relations**

# Binary Relations

- from set A to set B is defined as subset of Cartesian product  $A \times B$
- called binary relation because it deals with two sets (A and B)
- *order of coordinates of ordered pair is important*
- example based on  $A := \{X, Y, Z\}$ ,  $B := \{1, 2\}$
- $R1 := \{(X; 1), (X; 2), (Y; 1), (Z; 2)\}$
- $(\forall p \in R1: p \in A \times B)$

# Binary Relations

- $R$  is a binary relation from set  $A$  to set  $B \leftrightarrow R \in \wp(A \times B)$
- **Functions**

# Functions

- binary relation that doesn't have two elements that have same first coordinate
- in function two distinct ordered pairs always have different first coordinates
- $F \in \wp(A \times B) \wedge$
- $(\forall p_1, p_2 \in F: p_1 \neq p_2 \rightarrow \pi_1(p_1) \neq \pi_1(p_2)) \wedge$
- $\{\pi_1(p_1) \mid p \in F\} = A$

# Functions

- $\text{dom}(F) = A$
- $\text{rng}(F) \subseteq B$

# Identity Function

- over  $A$  if  $A$  is domain of  $f$ , and for all elements  $x$  in set  $A$ ,  $f$  of  $x$  equals  $x$
- $\text{id}(A)$  to represent identity function over  $A$

# Subset of Function

- every proper subset of function F is again a function

# N-ary Relation

- more general concept of n-ary relation  
(that is, not just a binary one)
- concept introduced by database field

# Operations on Functions

- functions are sets, can apply set operators union, intersection, and difference

# Compatible (Joinable) Functions

- generic property that two functions F and G should have for union of those two functions to result in a function
- function F is compatible with function G  $\leftrightarrow$
- $(\forall c \in (\text{dom}(F) \cap \text{dom}(G)) : F(c)=G(c))$

# Function Composition

- let  $A$ ,  $B$ , and  $C$  be sets
- let  $f$  be a function over  $A$  into  $B$
- let  $g$  be a function over  $\text{rng}(f)$  into  $C$
- composition of functions  $f$  and  $g$ , notation  $g \circ f$ , is defined as
- $g \circ f := \{(a; g(f(a)) \mid a \in \text{dom}(f) \wedge f(a) \in \text{dom}(g)\}$

# Limitation of Function

- subset of given function, want to consider those ordered pairs whose first coordinate can be chosen from some given subset of the domain of the function
- $F \downarrow A := \{ p \mid p \in F \wedge \pi_1(p) \in A\}$

# Set Functions

- function in which every ordered pair holds a set as its second coordinate
- $H := \{ (\text{team}; \{\text{'Cowboys'}, \text{'Vikings'}, \text{'Saints'}, \text{'Cardinals'}\}), \text{location}; \{\text{'Dallas'}, \text{'Minnesota'}, \text{'New Orleans'}, \text{'Arizona'}\}) \}$

- can consider set function  $H$  to enumerate, through first coordinates of its ordered pairs, two aspects of team: name of team (coordinate team) and location of team (coordinate location)
- such first coordinates are referred to as *attributes*
- attached to these attributes are the value sets as second coordinates of ordered pairs that could be considered to represent set of possible values for these attributes

# Characterizations

- can consider set function  $H$  to characterize a player
- set function called characterization when you use it to describe something in real world by listing relevant attributes in combination with set of admissible values for each attribute
- relevant **attributes** are first coordinates of ordered pairs
- sets of admissible values are their respective second coordinates **attribute value sets**

# External Predicates

- characterizations can be considered to characterize predicate in real world
- characterization introduces (names of) parameters of predicate and corresponding value sets from which these parameters can take their values

# Characterizations

- $H := \{$
- $(empno; [1..99]),$
- $(ename; varchar(10)),$
- $(born; date),$
- $(job; \{'CLERK', 'SALESMAN', 'TRAINER', 'MANAGER', 'PRESIDENT'\}),$
- $(salary; [1000..4999])$
- $\}$

# External Predicates

- previous characterization characterizes following predicate:
- employee ENAME is assigned employee number EMPNO, is born at date BORN, holds position JOB, and has a monthly salary of SALARY dollars

- characterization will be basis of table design
- predicate characterized by characterization of table design represents user-understood meaning of that table design
- such predicate is referred to as external predicate of that table

# Generalized Product of Set Function

- let  $F$  be set function
- generalized product of  $F$ , notation  $\Pi(F)$
- $\Pi(F) = \{f \mid f \text{ is a function} \wedge \text{dom}(f) = \text{dom}(F) \wedge (\forall c \in \text{dom}(f): f(c) \in F(c))\}$

# Generalized Product of Set Function

- generates set with all possible functions that have same domain as  $F$ , and for which the ordered pairs (in these functions) have second coordinate that is chosen from (that is, is an element of) the relevant set that was attached to same first coordinate in  $F$

- $\Pi(\{(a; \{1,2,3\}), (b; \{4,5\})\}) = \{$
- $\{(a; 1), (b; 4)\}, \{(a; 1), (b; 5)\},$
- $\{(a; 2), (b; 4)\}, \{(a; 2), (b; 5)\},$
- $\{(a; 3), (b; 4)\}, \{(a; 3), (b; 5)\} \}$

- $H := \{ (\text{team}; \{\text{'Cowboys'}, \text{'Vikings'}, \text{'Saints'}, \text{'Cardinals'}\}), \text{location}; \{\text{'Dallas'}, \text{'Minnesota'}, \text{'New Orleans'}, \text{'Arizona'}\}) \}$
- $\Pi(H) =$

- $\Pi(\mathsf{H}) = \{$
- $\{(team; 'Cowboys'), (location; 'Dallas')\}, \{(team; 'Cowboys'), (location; 'Minnesota')\},$
- $\{(team; 'Cowboys'), (location; 'New Orleans')\}, \{(team; 'Cowboys'), (location; 'Arizona')\},$
- $\{(team; 'Vikings'), (location; 'Dallas')\}, \{(team; 'Vikings'), (location; 'Minnesota')\},$
- $\{(team; 'Vikings'), (location; 'New Orleans')\}, \{(team; 'Vikings'), (location; 'Arizona')\},$
- $\{(team; 'Saints'), (location; 'Dallas')\}, \{(team; 'Saints'), (location; 'Minnesota')\},$
- $\{(team; 'Saints'), (location; 'New Orleans')\}, \{(team; 'Saints'), (location; 'Arizona')\},$
- $\{(team; 'Cardinals'), (location; 'Dallas')\}, \{(team; 'Cardinals'), (location; 'Minnesota')\},$
- $\{(team; 'Cardinals'), (location; 'New Orleans')\}, \{(team; 'Cardinals'), (location; 'Arizona')\} \}$

- refer to functions in result set of generalized product of set function as ***tuples***
- such set functions will typically signify a characterization
- elements of tuple, ordered pairs, are called attribute-value pairs of tuple
- in relational database design, tuple represents proposition in real world

- tuple {(empno; 100), (ename;'Smith'),  
(born; '19-jan-1964'), (job; 'MANAGER'),  
(salary; 5000) }
- denotes proposition “Employee Smith is assigned employee number 100, is born at date 19-jan-1964, holds position MANAGER, and has a monthly salary of 5000 dollars”

# Closed world assumption

- if given tuple appears in table design, then we assume that proposition denoted by that tuple is currently true proposition in real world

- other way around, if a given tuple that could appear in table design, but currently does not appear, then we assume that proposition denoted by that tuple is false proposition in real world
- principle is referred to as "Closed World Assumption" tuples held in database design describe all, and only, currently true propositions in real world

# **APPLICATION**

- mathematical foundation laid down
- how this can be applied to specify database and database designs

# Key Terms

- Entity, Entity identifier, Keys
- Attribute, Column, Domain of values
- Tuple, Row
- Equijoin
- Relation, Relational model, Relational database
- Logic

# **DataBase**

## **Relational Queries**

# Query languages

- specialized languages for asking questions, or *queries*, that involve the data in a database
- **relational algebra** - queries are specified *operational*
  - set of operators
  - each query describes a step-by-step procedure for computing the desired answer
- **relational calculus** - queries are specified nonprocedural *declarative*
- expressive power of algebra and calculus ?
- these formal query languages have greatly influenced commercial query languages such as SQL, which we will discuss later

# Influence on the design of commercial query languages

- Structured Query language (SQL)
- Query-by-Example (QBE)

- inputs and outputs of a query are relations
- present a number of sample queries using following schema:
- Sailors(*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
- Boats(*bid*: integer, *bname*: string, *color*: string)
- Reserves(*sid*: integer, *bid*: integer, *day*: date)

| <i>sid</i> | <i>sname</i> | <i>rating</i> | <i>age</i> |
|------------|--------------|---------------|------------|
| 22         | Dustin       | 7             | 45.0       |
| 31         | Lubber       | 8             | 55.5       |
| 58         | Rusty        | 10            | 35.0       |

Figure 4.1 Instance  $S1$  of Sailors

| <i>sid</i> | <i>sname</i> | <i>rating</i> | <i>age</i> |
|------------|--------------|---------------|------------|
| 28         | yuppy        | 9             | 35.0       |
| 31         | Lubber       | 8             | 55.5       |
| 44         | guppy        | 5             | 35.0       |
| 58         | Rusty        | 10            | 35.0       |

Figure 4.2 Instance  $S2$  of Sailors

| <i>sid</i> | <i>bid</i> | <i>day</i> |
|------------|------------|------------|
| 22         | 101        | 10/10/96   |
| 58         | 103        | 11/12/96   |

Figure 4.3 Instance  $R1$  of Reserves

# **Relational Algebra**

# Relational algebra

- queries composed using a collection of operators
- every operator in algebra accepts (one or two) relation instances as arguments and returns a relation instance as the result
- compose operators to form a complex query
- *expression* is recursively defined to be
  - Relation
  - unary algebra operator applied to a single expression
  - binary algebra operator applied to two expressions

# Relational query

- step-by-step procedure for computing the desired answer
- procedural nature of the algebra
  - algebra expression is a recipe, or a plan, for evaluating a query
- relational systems in fact use algebra expressions to represent query evaluation plans

# Selection and Projection

- operators to
  - *select* rows from a relation ( $\sigma$ )
  - *project* columns ( $\pi$ )
- operations allow us to manipulate data in a single relation

# Instance of Sailors $S_2$

| <i>sid</i> | <i>sname</i> | <i>rating</i> | <i>age</i> |
|------------|--------------|---------------|------------|
| 28         | yuppy        | 9             | 35.0       |
| 31         | Lubber       | 8             | 55.5       |
| 44         | guppy        | 5             | 35.0       |
| 58         | Rusty        | 10            | 35.0       |

$$\sigma_{\text{rating} > 8} (S_2)$$

| <i>sid</i> | <i>sname</i> | <i>rating</i> | <i>age</i> |
|------------|--------------|---------------|------------|
| 28         | yuppy        | 9             | 35.0       |
| 58         | Rusty        | 10            | 35.0       |

$$\pi_{\text{name}, \text{rating}}(S_2)$$

| <i>sname</i> | <i>rating</i> |
|--------------|---------------|
| yuppy        | 9             |
| Lubber       | 8             |
| guppy        | 5             |
| Rusty        | 10            |

# Selection operator

$$\sigma_{selection\_condition}$$

- specifies the tuples to retain through a *selection condition*
  - boolean expression using logical connectives  $\vee$  and  $\wedge$  of *terms* that have the form *attribute*<sub>1</sub>, op *attribute*<sub>2</sub>, where op is one of comparison operators  $<$ ;  $\leq$ ;  $=$ ;  $\geq$ ;  $>$
- reference to an attribute can be by position or by name
- schema of result is schema of input relation instance

# Projection operator

$$\pi_{\text{column\_list}}$$

- projection operator allows us to extract columns from a relation
- schema of result of projection is determined by fields that are projected in the *column list*
- result is a relation

$$\pi_{age}(S_2)$$

| <i>sid</i> | <i>sname</i> | <i>rating</i> | <i>age</i> |
|------------|--------------|---------------|------------|
| 28         | yuppy        | 9             | 35.0       |
| 31         | Lubber       | 8             | 55.5       |
| 44         | guppy        | 5             | 35.0       |
| 58         | Rusty        | 10            | 35.0       |

$$\pi_{age}(S_2)$$

| <i>sid</i> | <i>sname</i> | <i>rating</i> | <i>age</i> |
|------------|--------------|---------------|------------|
| 28         | yuppy        | 9             | 35.0       |
| 31         | Lubber       | 8             | 55.5       |
| 44         | guppy        | 5             | 35.0       |
| 58         | Rusty        | 10            | 35.0       |

|            |
|------------|
| <i>age</i> |
| 35.0       |
| 55.5       |

# Projection $\pi$ result is a relation

- follows from the definition of a relation as a *set* of tuples
- in practice, real systems often omit the expensive step of eliminating *duplicate tuples*, leading to relations that are multisets

# Composition of relational algebra operators

- since result of a relational algebra expression is always a relation, we can substitute an expression wherever a relation is expected
- for example, we can compute the names and ratings of highly rated sailors by combining two of the preceding queries
- $\pi_{sname, rating} (\sigma_{rating > 8} (S_2))$

# Set Operations

- following standard operations on sets are also available in relational algebra
  - *union* ( $\cup$ )
  - *intersection* ( $\cap$ )
  - *set-difference* ( $-$ )
  - *cross-product* ( $\times$ )

# Union ( $\cup$ )

- $R \cup S$  returns a relation instance containing all tuples that occur in *either* relation instance  $R$  or relation instance  $S$  (or both)

# Union-compatible

- $R$  and  $S$  must be *union-compatible*, and the schema of the result is defined to be identical to the schema of  $R$  (or  $S$ )
  - they have the same number of fields
  - corresponding fields, taken in order from left to right, have the same *domains*
- for convenience, we will assume that the fields of  $R \cup S$  inherit names from  $R$

# Intersection ( $\cap$ )

- $R \cap S$  returns a relation instance containing all tuples that occur in *both*  $R$  and  $S$
- relations  $R$  and  $S$  must be union-compatible, and the schema of the result is defined to be identical to the schema of  $R$

# Set-difference ( $-$ )

- $R-S$  returns a relation instance containing all tuples that occur in  $R$  but not in  $S$
- relations  $R$  and  $S$  must be union-compatible, and the schema of the result is defined to be identical to the schema of  $R$

# Cross-product ( $\times$ )

- $R \times S$  returns a relation instance whose schema contains all the fields of  $R$  (in the same order as they appear in  $R$ ) followed by all the fields of  $S$  (in the same order as they appear in  $S$ )
- result of  $R \times S$  contains one tuple  $\langle r, s \rangle$  (the concatenation of tuples  $r$  and  $s$ ) for each pair of tuples  $r \in R; s \in S$
- sometimes called ***Cartesian product***

# Cross-product naming conflict

- convention that fields of  $R \times S$  inherit names from the corresponding fields of  $R$  and  $S$
- possible for both  $R$  and  $S$  to contain one or more fields having the same name
  - corresponding fields in  $R \times S$  are unnamed and are referred to solely by position

# Renaming operator $\rho$

- $\rho (R(F), E)$
- takes an arbitrary relational algebra expression  $E$  and returns an instance of a (new) relation called  $R$
- $R$  contains same tuples as result of  $E$ , and has same schema as  $E$ , but some fields are renamed
- field names in relation  $R$  are same as in  $E$ , except for fields renamed in *renaming list*  $F$

# Renaming list

- terms having the form  $oldname \rightarrow newname$  or  $position \rightarrow newname$
- to be well-defined, references to fields may be unambiguous, and no two fields in result must have the same name

# Additional operators

- customary to include some additional operators in algebra, but they can all be defined in terms of already defined operators
- renaming operator needed for syntactic convenience
- can consider even  $\cap$  operator as redundant,  $R \cap S$  can be defined as  $R - (R - S)$

# Join operation

- one of the most useful operations in relational algebra
- most commonly used way to combine information from two or more relations

# Join operation

- can be defined as a cross-product followed by selections and projections, joins arise much more frequently in practice than plain cross-products
- important to recognize joins and implement them without materializing the underlying cross-product
- variants of join operation

# Condition Joins

- most general version of the join operation accepts a *join condition*  $c$  and a pair of relation instances as arguments, and returns a relation instance
- *join condition* is identical to a *selection condition* in form

# Condition Joins $|><|_c$

- $R |><|_c S = \sigma_c (R \times S)$
- defined to be a cross-product followed by a selection
- condition  $c$  can (and typically does) refer to attributes of both  $R$  and  $S$
- reference to an attribute of a relation, say  $R$ , can be by position (of form  $R.i$ ) or by name (of form  $R.name$ )

# Equijoin $|><|_c$

- special case of join operation  $R |><|S$  is when *join condition* consists solely of equalities (connected by  $\wedge$ ) of form  $R.name_1 = S.name_2$ , that is, equalities between two fields in  $R$  and  $S$
- obviously, there is some redundancy in retaining both attributes in result
- join operation is refined by doing an additional projection in which  $S.name_2$  is dropped

$$S_1 |><| R \text{.sid} = S \text{.sid} \quad R_1$$

| <i>sid</i> | <i>sname</i> | <i>rating</i> | <i>age</i> | <i>bid</i> | <i>day</i> |
|------------|--------------|---------------|------------|------------|------------|
| 22         | Dustin       | 7             | 45.0       | 101        | 10/10/96   |
| 58         | Rusty        | 10            | 35.0       | 103        | 11/12/96   |

# Natural Join $|><|$

- further special case of the join operation  $R |><| S$  is an equijoin in which equalities are specified on *all* fields having the same name in  $R$  and  $S$
- we can simply omit the join condition
- default is that join condition is a collection of equalities on all common fields
- result is guaranteed not to have two fields with the same name

- if the two relations have no attributes in common,  $R |><| S$  is simply the cross-product
- several variants of joins that are not discussed
- *outer joins*

# **Relational Calculus**

# Relational Calculus

- nonprocedural, or *declarative*,
- describe set of answers without being explicit about how they should be computed

# Relational Calculus

## 2 variants

- tuple relational calculus (TRC)
  - variables in TRC take on tuples as values
  - influence on SQL
- domain relational calculus (DRC)
  - variables range over field values
  - influenced QBE

# **Tuple Relational Calculus**

# Tuple Relational Calculus

- *tuple variable* is a variable that takes on tuples of a particular relation schema as values
  - every value assigned to a given tuple variable has the same number and type of fields

# Tuple Relational Calculus

- tuple relational calculus query has the form  $\{T \mid p(T)\}$ , where  $T$  is a tuple variable and  $p(T)$  denotes a *formula* that describes  $T$ 
  - where  $T$  is the only free variable in formula  $p$
- result of this query is the set of all tuples  $t$  for which the formula  $p(T)$  evaluates to true with  $T = t$
- language for writing formulas  $p(T)$  is thus at the heart of TRC and is essentially a simple subset of *first-order logic*

# Tuple Relational Calculus

- As a simple example, consider the following query.
  - *Find all sailors with a rating above 7?*

# Tuple Relational Calculus

- $\{S \mid S \in \text{Sailors} \wedge S.\text{rating} > 7\}$
- answer contains those instances of  $S$  that pass this test

# Syntax of TRC Queries

- $Rel$  - relation name
- $R$  and  $S$  - tuple variables
  - $a$  - an attribute of  $R$
  - $b$  - an attribute of  $S$
- op - an operator in the set  $\{<, \leq, \neq, \geq, >\}$

# Atomic formula in TRC

- $R \in Rel$
- $R.a$  op  $S.b$
- $R.a$  op *constant*
- *constant* op  $R.a$

# TRC Formula

- recursively defined to be one of the following, where  $p$  and  $q$  are themselves formulas and  $p(R)$  denotes a formula in which variable  $R$  appears
- any atomic formula
- $\neg p$ ,  $p \vee q$ ,  $p \wedge q$ ,  $p \Rightarrow q$
- $\exists R( p(R))$ , where  $R$  is a tuple variable
- $\forall R( p(R))$ , where  $R$  is a tuple variable

- In the last two clauses *quantiers*  $\forall$  and  $\exists$  are said to *bind* the variable  $R$
- variable is said to be *free* in a formula or *subformula* (a formula contained in a larger formula) if the (sub)formula does not contain an occurrence of a quantier that binds it

- observe that every variable in a TRC formula appears in a subformula that is atomic, and every relation schema specifies a domain for each field
- ensures that each variable in TRC formula has a well-defined domain from which values for the variable are drawn
  - each variable has a well-defined *type*

# Examples of TRC Queries

- *Find the sailor name, boat id, and reservation date for each reservation?*
- $\{P \mid \exists R \in \text{Reserves} \wedge \exists S \in \text{Sailors}$
- $(R.sid = S.sid \wedge P.bid = R.bid \wedge P.day = R.day \wedge P.sname = S.sname) \}$

# Examples of TRC Queries

- for each Reserves tuple, we look for a tuple in Sailors with the same *sid*
- given a pair of such tuples, we construct an answer tuple  $P$  with fields *sname*, *bid*, and *day* by copying corresponding fields from these two tuples

# **Domain relational calculus**

# Domain relational calculus

- ***domain variable*** ranges over values in domain of some attribute

# Domain relational calculus

- domain relational calculus query has the form  $\{<x_1, x_2, \dots, x_n> \mid p(<x_1, x_2, \dots, x_n>)\}$
- where each  $x_i$  is either a *domain variable* or a constant and  $p(<x_1, x_2, \dots, x_n>)$  denotes a *DRC formula* whose only free variables are the variables among the  $x_i$ ,  $1 \leq i \leq n$
- result of this query is the set of all tuples  $\{<x_1, x_2, \dots, x_n>$  for which the formula evaluates to true

- A DRC formula is defined in a manner that is very similar to the definition of a TRC formula.
- difference is that the variables are now domain variables

# Syntax of DRC Queries

- op - an operator in the set { $<$ ,  $\leq$ ,  $\neq$ ,  $\geq$ ,  $>$ }
- $X$  and  $Y$  be domain variables

# Atomic formula in DRC

- $\{x_1, x_2, \dots, x_n\} \in Rel$ 
  - where  $Rel$  is a relation with  $n$  attributes
  - each  $x_i$ ,  $1 \leq i \leq n$  is either variable or constant
- $X \text{ op } Y$
- $X \text{ op } \text{constant}$
- $\text{constant} \text{ op } X$

# DRC Formula

- recursively defined to be one of the following, where  $p$  and  $q$  are themselves formulas and  $p(X)$  denotes a formula in which variable  $X$  appears
- any atomic formula
- $\neg p, p \vee q, p \wedge q, p \Rightarrow q$
- $\exists R( p(R))$ , where  $R$  is a domain variable
- $\forall R( p(R))$ , where  $R$  is a domain variable

# Examples of DRC Queries

- *Find all sailors with a rating above 7?*
- $\{<I, N, T, A> \mid \{<I, N, T, A> \in \text{Sailors} \wedge T > 7\}$

# **Expressive power of algebra and calculus**

- every query that can be expressed in relational algebra also can be expressed in safe relational calculus

# Unsafe queries

- consider the query  $\{S \mid \neg(S \in \text{Sailors})\}$ 
  - syntactically correct
  - set of such  $S$  tuples is obviously infinite, in the context of infinite domains
- restrict relational calculus to disallow unsafe queries

- if query language can express all queries that we can express in relational algebra, it is said to be *relationally complete*
- practical query language is expected to be relationally complete
- in addition, commercial query languages typically support features that allow us to express some queries that cannot be expressed in relational algebra

# Structured Query Language SQL

- is the most widely used commercial relational database language
- originally developed at IBM in System-R projects
- other vendors introduced DBMS products based on SQL, and it is now a de facto standard
- current ANSI/ISO standard for SQL-92

# Key Terms

- Relational algebra
- Relational calculus
- Data retrieval
- Select operator
- Join operator
- Natural join

# **Thank you for your kindly attention!**

# DataBases

Relational Model -  
Relational Algebra @ Calculus

- Suppose you are given a relation  $R$  with four attributes  $ABCD$ . (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.
- 1.  $C \rightarrow D, C \rightarrow A, B \rightarrow C$
- 2.  $B \rightarrow C, D \rightarrow A$

# Objectives

- Explain why relational database model became practical in about 1980 and de facto standard today
- Define basic relational database terms such as relation and tuple
  - Describe major types of keys including primary, candidate, and foreign
  - Describe how one-to-one, one-to-many, and many-to-many relationships are implemented

# Objectives

- Describe how relational data retrieval is accomplished using the relational algebra select, project, and join operators
- Understand how the join operator facilitates data integration in relational database
- Describe also relational calculus

# Relational model

- Professionals in any discipline need to know the foundations of their field
- Database Professional - need to know theory of relations
- is not product-specific;
  - rather, it is concerned with principles

# Principles, not Products

- principle
  - source, root, origin; which is fundamental; essential nature; theoretical basis
- principles endure
- by contrast, products and technologies, change all the time

# Relation Model

- E. F. Codd, at the time researcher at IBM; late in 1968 that Codd, a mathematician, first realized that discipline of mathematics could be used to inject some solid principles and rigor into field of database

# original definition of relational model

- E. F. Codd
  - *Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks*, IBM Research Report RJ599, 1969.
  - *A Relational Model of Data for Large Shared Data Banks*, CACM 13, No. 6, 1970.
- C. J. Date
  - *An Introduction to Database Systems*, 8<sup>th</sup> Ed. Boston, Mass.: Addison-Wesley, 2004.
  - first edition of book was published in 1975

# Historical perspective

- Relational Database Management System (RDBMS) has become dominant data-processing software in use today, with an estimated total software revenue worldwide of US\$ 20-30 billion
- software represents second generation of DBMSs based on proposed *relational data model* in 1970, Edgar Codd, at IBM's San Jose Research Laboratory

# Historical perspective

- in relational model, all data is logically structured within relations (tables); each ***relation*** has name and is made up of named ***attributes*** (columns) of data; each ***tuple*** (row) contains one value per attribute
- great strength of relational model is this simple logical structure; behind this structure is sound theoretical foundation that is lacking in first generation of DBMSs (network & hierarchical)

# Mathematics

- always strive for elegance and orthogonality - dislike exceptions
- system is said to be designed in an *orthogonal* way if its set of components that together make up whole system capability are *non-overlapping* and *mutually independent*
  - each capability should be implemented by only one component
  - one component should only implement one capability of system

# Mathematics

- well-separated and independent components ensure that there are no side effects: using or even changing one component does not cause side effects in another area of system
- formal disciplines, most relevant ones in application of mathematics to field of databases
- Set theory

# Mathematics

- *Everything should be made as simple as possible, but not simpler.*
  - Albert Einstein (1879–1955)

# Codd relational model's objectives

- to allow high degree of data independence; application programs must not be affected by modifications to internal data representation, particularly by changes to file organizations, record orderings, or access paths
- to provide substantial grounds for dealing with data semantics, consistency, and redundancy problems; introduced concept of ***normalized*** relations that have no repeating groups
- to enable expansion of set-oriented data manipulation languages

# 1 - Prototype relational DBMS System R

- at IBM's San José Research Laboratory in California - prototype relational DBMS System R, which was developed during the late 1970s
- project was designed to prove practicality of relational model by providing an implementation of its data structures and operations
- proved to be excellent source of information about implementation concerns such as transaction management, concurrency control, recovery techniques, query optimization, data security and integrity, human factors, and user interfaces,

# led to two major developments

- development of a structured query language called SQL, which has since become formal International Organization for Standardization (ISO) and de facto standard language for RDBMS
- production of various commercial relational DBMS products during late 1970s and 1980s: IBM DB2 and Oracle

## 2 - INGRES

- development of relational model INGRES (Interactive Graphics Retrieval System) project at University of California at Berkeley
- involved development of prototype RDBMS, with research concentrating on same overall objectives as System R project - led to an academic version of INGRES, which contributed to general appreciation of relational concepts, and spawned commercial products INGRES

# Terminology and structural concepts of relational model

- relational model is based on mathematical concept of *relation*, which is physically represented as **table**
- we explain terminology and structural concepts of the relational model

# Relation

- is a table with columns and rows
- RDBMS requires only that database be perceived by user as tables
- perception applies only to logical structure of database: external and conceptual levels of ANSI-SPARC architecture
- does not apply to physical structure of database, which can be implemented using variety of storage structures

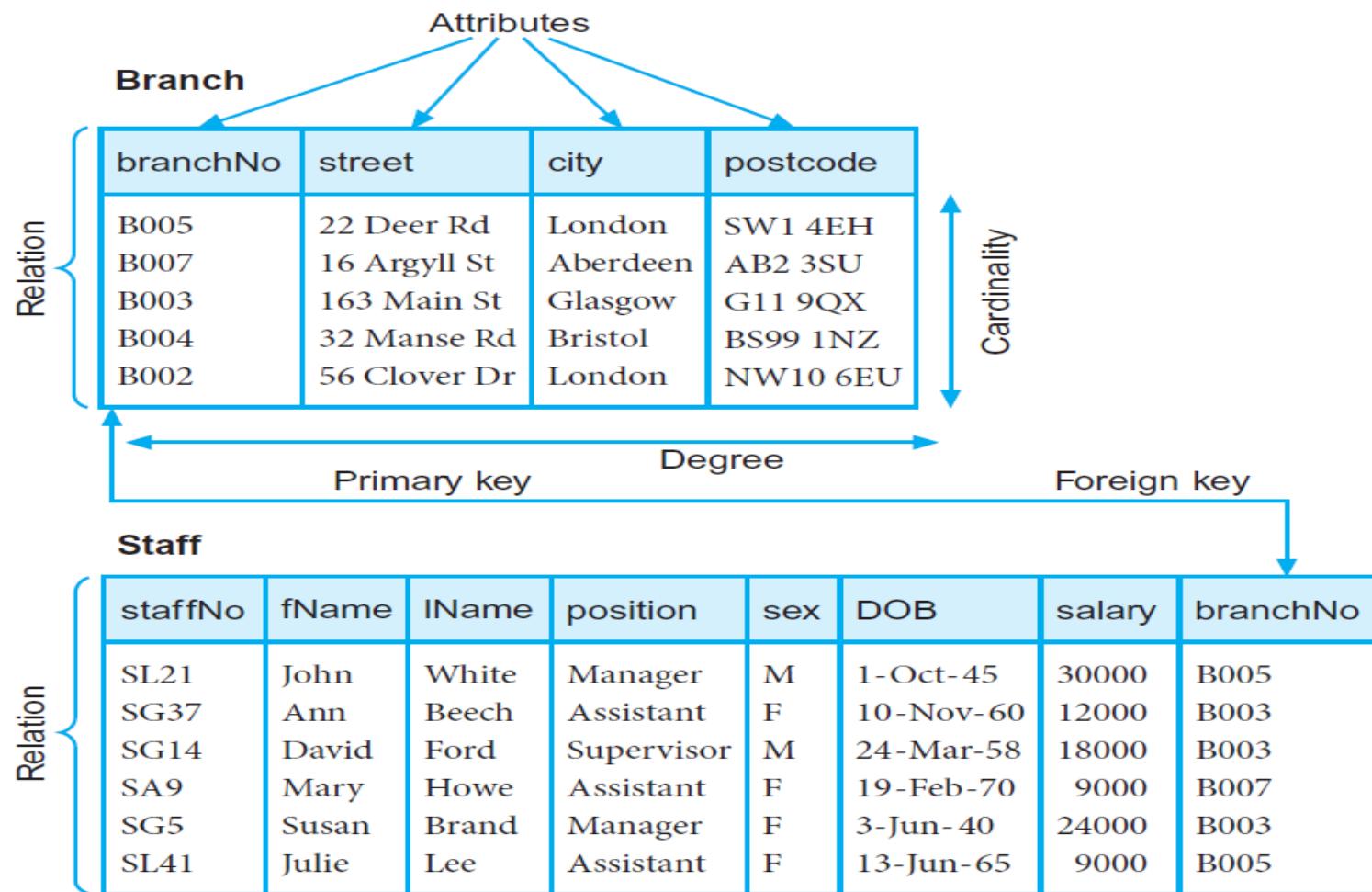
# Attribute

- is named column of relation
- relations used to hold information about objects to be represented in database as a two-dimensional table in which rows correspond to individual records and columns correspond to attributes
- attributes can appear in any order and relation will still be the same - therefore will convey same meaning

# Domain

- is set of allowable (possible) values for one or more attributes
- may be distinct for each attribute, or two or more attributes may be defined on same domain
- typically there will be values in domain that do not currently appear as values in corresponding attribute

# Instances of Branch & Staff relation



# Domains for some attributes

| Attribute | Domain Name   | Meaning                                | Domain Definition                              |
|-----------|---------------|----------------------------------------|------------------------------------------------|
| branchNo  | BranchNumbers | The set of all possible branch numbers | character: size 4, range B001-B999             |
| street    | StreetNames   | The set of all street names in Britain | character: size 25                             |
| city      | CityNames     | The set of all city names in Britain   | character: size 15                             |
| postcode  | Postcodes     | The set of all postcodes in Britain    | character: size 8                              |
| sex       | Sex           | The sex of a person                    | character: size 1, value M or F                |
| DOB       | DatesOfBirth  | Possible values of staff birth dates   | date, range from 1-Jan-20,<br>format dd-mmm-yy |
| salary    | Salaries      | Possible values of staff salaries      | monetary: 7 digits, range<br>6000.00-40000.00  |

# tuple

- is row of relation
- elements of relation are rows or tuples in table
- tuples can appear in any order and relation will still be the same - therefore convey same meaning
- structure of relation, together with specification of domains and any other restrictions on possible values, is called its intension; usually fixed - tuples are called extension (or state) of relation, which changes over time

# Degree

- is number of attributes it contains
- the Branch relation has four attributes or degree four – means that each row of table is a four-tuple, containing four values
- degree of a relation is property of *intension* of relation

# Cardinality

- is number of tuples it contains
- changes as tuples are added or deleted; is property of *extension* of relation and is determined from particular instance of relation at any given moment

# Relational database

- collection of normalized relations with distinct relation names
- consists of relations that are appropriately structured - refer to this as *normalization*

# Alternative terminology

- third set of terms is sometimes used;  
stems from fact that, physically, RDBMS  
may store each relation in a file

# Alternative terminology

| Formal term | Logical | Physical |
|-------------|---------|----------|
| Relation    | Table   | File     |
| Attribute   | Column  | Field    |
| Tuple       | Row     | Record   |

# MATHEMATICAL RELATIONS

# Sets and Elements

- set is a collection of objects, and those objects are called the elements of the set
- set is fully characterized by its distinct elements, and nothing else but these elements

# Sets and Elements

- elements of set don't have any ordering
- sets don't contain duplicate elements
- two sets A and B, are the same if each element of A is also an element of B and each element of B is also an element of A
- $\emptyset$  the empty set

# Sets and Elements

- set theory has a mathematical symbol  $\in$  that is used to state that some object is an element of some set
- $x \in S$
- $x \notin S \leftrightarrow \neg(x \in S)$

# Methods to specify sets

- enumerative method
  - $S1 := \{1, 2, 3, 5, 7, 11, 13, 17, 19, 23\}$
- predicative method
  - $S2 := \{x \in S \mid P(x)\}$ 
    - let  $P(x)$  be a given predicate with exactly one parameter named  $x$  of type  $S$
- substitutive method
  - $S3 := \{x^2 \mid x \in N\}$

# Cardinality

- sets can be finite and infinite
- when dealing with databases all your sets are finite by definition
- for every finite set  $S$ , cardinality is defined as number of elements of  $S$

# Subsets, Supersets

- A is a subset of B (B is a superset of A) if every element of A is an element of B
- $A \subseteq B$
- A is proper subset of B (B is proper superset of A) if A is subset of B and  $A \neq B$
- $A \subset B$

# Union, Intersection, Difference

- union  $A \cup B = \{x \mid x \in A \vee x \in B\}$
- intersection  $A \cap B = \{x \mid x \in A \wedge x \in B\}$
- difference  $A - B = \{x \mid x \in A \wedge x \notin B\}$

# (Binary) Relation

- two sets,  $D_1$  and  $D_2$ , where  $D_1 = \{2, 4\}$  and  $D_2 = \{1, 3, 5\}$ ; Cartesian product of these two sets, written  $D_1 \times D_2$ , is set of all ordered pairs such that first element is a member of  $D_1$  and second element is a member of  $D_2$  -  $D_1 \times D_2 = \{(2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5)\}$
- any subset of this Cartesian product is a relation

# Example of relation $R$

- $R = \{(2, 1), (4, 1)\}$
- $R = \{(x, y) \mid x \in D_1, y \in D_2, \text{ and } y = 1\}$
- easily extend notion to three sets; let  $D_1$ ,  $D_2$ , and  $D_3$  be sets; Cartesian product  $D_1 \times D_2 \times D_3$  of these three sets is set of all ordered triples such that first element is from  $D_1$ , second is from  $D_2$  and third element is from  $D_3$
- any subset of Cartesian product is relation

# N-ary Relation

- order of coordinates of ordered pair is important
- order of tuples it is not important – it is subset of Cartesian – subset of a set is a set – order of element in a set it is not important
- can extend three sets to a more general concept of n-ary relation

# Relation

- define general relation on  $n$  domains
- let  $D_1, D_2, \dots, D_n$  be  $n$  sets
- Cartesian product is defined as:
- $D_1 \times D_2 \times \dots \times D_n = \{(d_1, d_2, \dots, d_n) \mid d_1 \in D_1, d_2 \in D_2, \dots, d_n \in D_n\}$  and is usually written as  $\prod_{i=1}^n D_i$
- any set of  $n$ -tuples from this Cartesian product is relation on  $n$  sets

# DataBase Relations

- note that in defining these relations we have to specify sets, or domains, from which we choose values
- Relation schema: named relation defined by set of attribute and domain name pairs
- let  $A_1, A_2, \dots, A_n$  be attributes with domains  $D_1, D_2, \dots, D_n$
- then set  $\{A_1:D_1, A_2:D_2, \dots, A_n:D_n\}$  is relation schema

# DataBase Relations

- relation  $R$  defined by relation schema is set of mappings from attribute names to their corresponding domains
- relation  $R$  is set of  $n$ -tuples:
- $(A_1:d_1, A_2:d_2, \dots, A_n:d_n)$  such that  $d_1 \in D_1$ ,  $d_2 \in D_2, \dots, d_n \in D_n$
- each element in  $n$ -tuple consists of attribute and value for that attribute

# DataBase Relations

- normally we list attribute names as column headings and write out tuples as rows having form  $(d_1, d_2, \dots, d_n)$ , where each value is taken from appropriate domain
- Relation is any subset of Cartesian product of domains of attributes
- table is simply physical representation of such relation

# Branch and Staff relations

## Branch

| branchNo | street       | city     | postCode |
|----------|--------------|----------|----------|
| B005     | 22 Deer Rd   | London   | SW1 4EH  |
| B007     | 16 Argyll St | Aberdeen | AB2 3SU  |
| B003     | 163 Main St  | Glasgow  | G11 9QX  |
| B004     | 32 Manse Rd  | Bristol  | BS99 1NZ |
| B002     | 56 Clover Dr | London   | NW10 6EU |

## Staff

| staffNo | fName | IName | position   | sex | DOB       | salary | branchNo |
|---------|-------|-------|------------|-----|-----------|--------|----------|
| SL21    | John  | White | Manager    | M   | 1-Oct-45  | 30000  | B005     |
| SG37    | Ann   | Beech | Assistant  | F   | 10-Nov-60 | 12000  | B003     |
| SG14    | David | Ford  | Supervisor | M   | 24-Mar-58 | 18000  | B003     |
| SA9     | Mary  | Howe  | Assistant  | F   | 19-Feb-70 | 9000   | B007     |
| SG5     | Susan | Brand | Manager    | F   | 3-Jun-40  | 24000  | B003     |
| SL41    | Julie | Lee   | Assistant  | F   | 13-Jun-65 | 9000   | B005     |

- it shows for example that employee John White is manager with salary of 30,000, who works at branch (branchNo) B005, which, from first table, is at 22 Deer Rd in London
- there is relationship between Staff and Branch: branch office *has* staff
- there is no explicit link between these two tables; it is only by knowing that attribute branchNo in Staff have same value as branchNo of Branch

# Branch relation

- has attributes branchNo, street, city, and postcode, each with its domain
- is any subset of Cartesian product of domains, or any set of four-tuples in which first element is from domain BranchNumbers, second is from domain StreetNames, ...

# Branch relation instance

- one of tuples is: or more correctly
- $\{(B005, 22\text{ Deer Rd, London, SW1 4EH})\}$
- $\{(branchNo: B005, street: 22\text{ Deer Rd, city: London, postcode: SW1 4EH})\}$
- we refer to this as *relation instance*
- Branch table is convenient way of writing out all tuples that form relation at specific moment in time

# Relational database schema

- set of relation schemas, each with distinct name
- in same way that relation has schema, so too does relational database
- if  $R_1, R_2, \dots, R_n$  are set of relation schemas, then we can write *relational database schema*, or simply *relational schema*,  $R$ , as:  $R = \{R_1, R_2, \dots, R_n\}$

# Properties of Relations

- relation has name that is distinct from all other relation names in relational schema
- each cell of relation contains exactly one atomic (single) value; relations do not contain repeating groups
- relation that satisfies this property is said to be *normalized* in *first normal form*

# Properties of Relations

- each attribute has distinct name
- value of attribute are all from same domain
- each tuple is distinct; there are no duplicate tuples
- order of attributes has no significance
- order of tuples has no significance
- most of properties specified result from properties of mathematical relations

# Relational Keys

- ... there are no duplicate tuples ...
- therefore, we need to identify one or more attributes (*relational keys*) that uniquely identifies each tuple in relation

# Superkey

- attribute, or set of attributes, that uniquely identifies tuple within relation
- may contain additional attributes that are not necessary for unique identification
- interested in identifying superkeys that contain only minimum number of attributes necessary for unique identification

# Candidate Key

- superkey such that no proper subset is superkey within relation
- *Uniqueness* – in each tuple of  $R$ , values of  $K$  uniquely identify that tuple
- *Irreducibility* - no proper subset of  $K$  has uniqueness property
- when key consists of more than one attribute, we call it *composite key*
- there may be several candidate keys for relation

# Candidate Key

- consider Branch relation
- given value of city, we can determine several branch offices (London has two branch offices)
- attribute cannot be a candidate key
- company allocates each branch office unique branch number we can determine at most one tuple, so that *branchNo* is candidate key
- similarly, postcode is also candidate key

# Candidate Key

- consider relation *Viewing*, which contains information relating to properties viewed by clients
- relation comprises client number (*clientNo*), property number (*propertyNo*), date of viewing (*viewDate*) and, optionally, comment (*comment*)

# Candidate Key

- combination of *clientNo* and *propertyNo* identifies at most one tuple, so for *Viewing* together form (composite) candidate key
- if we take into account possibility that client may view property more than once, then we could add *viewDate* to composite key

# Candidate Key

- instance of relation cannot be used to prove that attribute or combination of attributes is CK
- fact that there are no duplicates for values that appear at particular moment in time does not guarantee that duplicates are not possible
- presence of duplicates in instance can be used to show that attribute combination is not CK
- identifying candidate key requires that we know the “real-world” meaning of attribute(s) involved; can decide whether duplicates are possible

# Primary Key

- candidate key that is selected to identify tuples uniquely within relation (by database designer)
- relation has no duplicate tuples; always possible to identify each row uniquely
- relation always has primary key
- in worst case, entire set of attributes could serve as primary key; usually some smaller subset is sufficient to distinguish tuples

# Alternate Key

- candidate keys that are not selected to be primary key
- for *Branch* relation, if we choose *branchNo* as primary key, *postcode* would then be an alternate key
- for *Viewing* relation, there is only one candidate key, comprising *clientNo* and *propertyNo*, so these attributes would automatically form primary key

# Foreign Key

- attribute, or set of attributes, within one relation that matches candidate key of some (possibly same) relation
- when attribute appears in more than one relation, its appearance usually represents relationship between tuples of two relations
- for example, inclusion of *branchNo* in both *Branch* and *Staff* relations is quite deliberate and links each branch to details of staff working at that branch

# Foreign Key

- in *Branch* relation, *branchNo* is primary key
- in *Staff* relation, *branchNo* attribute exists to match staff to branch office they work in
- in *Staff* relation, *branchNo* is *foreign key*
- we say that attribute *branchNo* in *Staff* relation *targets* primary key attribute *branchNo* in home relation, *Branch*

# Representing Relational Database Schemas

- *Branch (branchNo, street, city, postcode)*
- *Staff (staffNo, fName, lName, position, sex, DOB, salary, branchNo)*
- *PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)*
- *Client (clientNo, fName, lName, telNo, prefType, maxRent, eMail)*

# Representing Relational Database Schemas

- *PrivateOwner (ownerNo, fName, lName, address, telNo, eMail, password)*
- *Viewing (clientNo, propertyNo, viewDate, comment)*
- *Registration (clientNo, branchNo, staffNo, dateJoined)*

# Representing Relational Database Schemas

- common convention for representing relation schema is to give name of relation followed by attribute names in parentheses; primary key is underlined
- *conceptual model*, or *conceptual schema*, is set of all such schemas for database

# Instance of rental database

**Branch**

| branchNo | street       | city     | postcode |
|----------|--------------|----------|----------|
| B005     | 22 Deer Rd   | London   | SW1 4EH  |
| B007     | 16 Argyll St | Aberdeen | AB2 3SU  |
| B003     | 163 Main St  | Glasgow  | G11 9QX  |
| B004     | 32 Manse Rd  | Bristol  | BS99 1NZ |
| B002     | 56 Clover Dr | London   | NW10 6EU |

**Staff**

| staffNo | fName | IName | position   | sex | DOB       | salary | branchNo |
|---------|-------|-------|------------|-----|-----------|--------|----------|
| SL21    | John  | White | Manager    | M   | 1-Oct-45  | 30000  | B005     |
| SG37    | Ann   | Beech | Assistant  | F   | 10-Nov-60 | 12000  | B003     |
| SG14    | David | Ford  | Supervisor | M   | 24-Mar-58 | 18000  | B003     |
| SA9     | Mary  | Howe  | Assistant  | F   | 19-Feb-70 | 9000   | B007     |
| SG5     | Susan | Brand | Manager    | F   | 3-Jun-40  | 24000  | B003     |
| SL41    | Julie | Lee   | Assistant  | F   | 13-Jun-65 | 9000   | B005     |

**PropertyForRent**

| propertyNo | street        | city     | postcode | type  | rooms | rent | ownerNo | staffNo | branchNo |
|------------|---------------|----------|----------|-------|-------|------|---------|---------|----------|
| PA14       | 16 Holhead    | Aberdeen | AB7 5SU  | House | 6     | 650  | CO46    | SA9     | B007     |
| PL94       | 6 Argyll St   | London   | NW2      | Flat  | 4     | 400  | CO87    | SL41    | B005     |
| PG4        | 6 Lawrence St | Glasgow  | G11 9QX  | Flat  | 3     | 350  | CO40    |         | B003     |
| PG36       | 2 Manor Rd    | Glasgow  | G32 4QX  | Flat  | 3     | 375  | CO93    | SG37    | B003     |
| PG21       | 18 Dale Rd    | Glasgow  | G12      | House | 5     | 600  | CO87    | SG37    | B003     |
| PG16       | 5 Novar Dr    | Glasgow  | G12 9AX  | Flat  | 4     | 450  | CO93    | SG14    | B003     |

# Instance of rental database

**Client**

| clientNo | fName | IName   | telNo         | prefType | maxRent | eMail                  |
|----------|-------|---------|---------------|----------|---------|------------------------|
| CR76     | John  | Kay     | 0207-774-5632 | Flat     | 425     | john.kay@gmail.com     |
| CR56     | Aline | Stewart | 0141-848-1825 | Flat     | 350     | astewart@hotmail.com   |
| CR74     | Mike  | Ritchie | 01475-392178  | House    | 750     | mritchie01@yahoo.co.uk |
| CR62     | Mary  | Tregear | 01224-196720  | Flat     | 600     | maryt@hotmail.co.uk    |

**PrivateOwner**

| ownerNo | fName | IName  | address                       | telNo         | eMail             | password |
|---------|-------|--------|-------------------------------|---------------|-------------------|----------|
| CO46    | Joe   | Keogh  | 2 Fergus Dr, Aberdeen AB2 7SX | 01224-861212  | jkeogh@lhh.com    | *****    |
| CO87    | Carol | Farrel | 6 Achray St, Glasgow G32 9DX  | 0141-357-7419 | cfarrel@gmail.com | *****    |
| CO40    | Tina  | Murphy | 63 Well St, Glasgow G42       | 0141-943-1728 | tinam@hotmail.com | *****    |
| CO93    | Tony  | Shaw   | 12 Park Pl, Glasgow G4 0QR    | 0141-225-7025 | tony.shaw@ark.com | *****    |

**Viewing**

| clientNo | propertyNo | viewDate  | comment        |
|----------|------------|-----------|----------------|
| CR56     | PA14       | 24-May-13 | too small      |
| CR76     | PG4        | 20-Apr-13 | too remote     |
| CR56     | PG4        | 26-May-13 |                |
| CR62     | PA14       | 14-May-13 | no dining room |
| CR56     | PG36       | 28-Apr-13 |                |

**Registration**

| clientNo | branchNo | staffNo | dateJoined |
|----------|----------|---------|------------|
| CR76     | B005     | SL41    | 2-Jan-13   |
| CR56     | B003     | SG37    | 11-Apr-12  |
| CR74     | B003     | SG37    | 16-Nov-11  |
| CR62     | B007     | SA9     | 7-Mar-12   |

# Null

- represents value for attribute that is currently unknown or is not applicable for this tuple
- mean logical value “unknown”, value is not applicable to particular tuple, or that no value has yet been supplied
- way to deal with incomplete or exceptional data
- not same as zero or empty text string
- represents absence of value

# Null

- for example, in Viewing relation *comment* attribute may be undefined until potential renter has visited property and returned comment to agency
- without nulls, it becomes necessary to introduce false data to represent this state

# Null

- can cause implementation problems, arising from fact that relational model is based on first-order predicate calculus, which is two-valued or Boolean logic; allowing nulls means that we have to work with higher-valued logic, such as three-valued logic (true, false, null)
- Codd regarded nulls as an integral part of relational model

# Integrity Constraints

- relational integrity constraints ensure that data is accurate
- because every attribute has an associated domain, there are ***domain constraints*** that form restrictions on set of values allowed for attributes of relations
- ***integrity rules*** - constraints that apply to all instances of database: ***entity integrity*** and ***referential integrity***

# Integrity Constraints

- because every attribute has an associated domain, there are ***domain constraints*** that form restrictions on set of values allowed for attributes of relations
- ***integrity rules*** - constraints that apply to all instances of database: ***entity integrity*** and ***referential integrity***

# Entity integrity

- in base relation, no attribute of primary key can be null
- PK is minimal identifier used to identify tuples uniquely; if we allow null for any part of PK we are implying that not all attributes are needed
- for example, *branchNo* is PK of *Branch*, we should not be able to insert tuple with null
- Viewing (clientNo, propertyNo, ...) we should not be able to insert tuple into Viewing with null for clientNo or null for propertyNo or nulls for both

# Entity integrity

- using data of *Viewing* relation consider query  
“List all comments from viewings”
- produce unary relation consisting of attribute comment
- this attribute must be PK, but it contains nulls
- this relation is not base relation (derived relation), model allows primary key to contain nulls

# Referential integrity

- if foreign key exists in relation, either foreign key value must match a candidate key value of some tuple in its home relation or foreign key value must be wholly null

# Referential integrity

- for example, *branchNo* in *Staff* is FK targeting *branchNo* attribute in home relation, *Branch*
- possible to create *Staff* record with *branchNo* B025, if there is already *branchNo* B025 in *Branch* relation
- we should be able to create *Staff* record with null *branchNo* to allow for situation where member of staff has joined company but has not yet been assigned to particular branch office

# General constraints

- additional rules specified by users of database that define or constrain some aspect of organization
- for example, if an upper limit of 20 has been placed upon number of staff that may work at branch office
- level of support for general constraints varies from system to system

# General constraints

- *declarative integrity constraint* is statement about database that is always true
- *trigger* is procedural code that is automatically executed in response to certain events on particular table or view in database; mostly used for maintaining integrity of information

# Views

- in relational model has slightly different meaning than view from architecture
- rather than being entire external model of user's view, view is *virtual* or *derived relation* that does not necessarily exist in its own right, but may be dynamically derived from one or more base relations
- external model can consist of both base (conceptual-level) relations and views derived from base relations

# Views

- relation that appears to user to exist, can be manipulated as if it were base relation, but does not necessarily exist in storage (although its definition is stored in system catalog)
- contents of view are defined as query on one or more base relations
- operations on view are automatically translated into operations on relations from which it is derived

# Views

- dynamic – changes made to base relations that affect view are immediately reflected in view
- when users make permitted changes to (updating) view, these changes are made to underlying relations

# Views

- provides powerful and flexible security mechanism by hiding parts of database from certain users; users are not aware of existence of any attributes or tuples that are missing
- it permits users to access data in way that is customized to their needs, same data can be seen by different users in different ways
- it can simplify complex operations on base relations

# Views

- for example, if view defined as combination (join) of two relations users may now perform more simple operations on view, which will be translated by DBMS into equivalent operations
- some members of staff should see *Staff* tuples without salary attribute
- attributes may be renamed or order changed; user accustomed to calling *branchNo* attribute of branches by full name

# Views

- provides *logical data independence*;  
supports reorganization of conceptual schema
- for example, if new attribute is added to relation,  
existing users can be unaware of its existence if  
their views are defined to exclude it; if existing  
relation is rearranged or split up, view may be  
defined so that users can continue to see their  
original views

# Codd's rules

- Codd came up with rules database must obey in order to be regarded as relational
- hardly any commercial product follows all

# Rule 0: Foundation

- system must qualify as relational, as a database, and as a management system
- system must use its relational facilities (exclusively) to manage database
- foundation rule, which acts as base for all other 12 rules derive from this

# Rule 1: Information

- all information in database is to be represented in one and only one way, namely by values in column positions within rows of tables
- data stored in database, may it be user data or metadata, must be value of some table cell; everything in database must be stored in a table format

# Rule 2: Guaranteed Access

- all data must be accessible; essentially restatement of fundamental requirement for primary keys; every individual scalar value in database must be logically addressable by specifying name of table, name of column and primary key value
- every single data element (value) is guaranteed to be accessible logically with combination of table-name, primary-key (row), and attribute-name (column); no other means, such as pointers, can be used to access data

# Rule 3: Systematic Treatment of NULL Values

- DBMS must allow each field to remain null (or empty); must support representation of "missing information and inapplicable information" that is systematic, distinct from all regular values (0, "", ...), and independent of data type; manipulated by DBMS in systematic way
- NULL values in database must be given systematic and uniform treatment; interpreted as
  - data is missing, data is not known, or data is not applicable

# Rule 4: Active Online Catalog

- system must support online relational catalog that is accessible to authorized users by means of their regular query language; users must be able to access database's structure (catalog) using same query language that they use to access database's data
- structure description of entire database must be stored in online catalog, known as *data dictionary*, which can be accessed by authorized users; users can use same query language to access catalog which they use to access database itself

# Rule 5: Comprehensive Data Sub-Language

- system must support at least one relational language that
  - has linear syntax
  - can be used both interactively and within application programs
  - supports data definition operations (including view definitions), data manipulation operations (update and retrieval), security and integrity constraints, transaction management operations (begin, commit, and rollback)

# Rule 5: Comprehensive Data Sub-Language

- database can only be accessed using language having linear syntax that supports data definition, data manipulation, and transaction management operations; language can be used directly or by means of some application
- if database allows access to data without any help of this language, then it is considered violation

# Rule 6: View Updating

- all views those can be updated theoretically, must be updated by system.
- all views of database, which can theoretically be updated, must also be updatable by system

# Rule 7: High-Level Insert, Update, and Delete

- system must support set-at-a-time insert, update, and delete operators; data can be retrieved from relational database in sets constructed of data from multiple rows and/or multiple tables; insert, update, and delete operations should be supported for any retrievable set rather than just for single row in single table

# Rule 7: High-Level Insert, Update, and Delete

- database must support high-level insertion, updation, and deletion; this must not be limited to single row, that is, it must also support union, intersection and minus operations to yield sets of data records

# Rule 8: Physical Data Independence

- changes to physical level (how data is stored, whether in arrays or linked lists etc.) must not require change to application based on structure
- data stored in database must be independent of applications that access database; any change in physical structure of database must not have any impact on how data is being accessed by external applications

# Rule 9: Logical Data Independence

- changes to logical level (tables, columns, rows, ...) must not require change to application based on structure; more difficult to achieve
- logical data in database must be independent of its user's view (application); any change in logical data must not affect applications using it

# Rule 10: Integrity Independence

- integrity constraints must be specified separately from application programs and stored in catalog; it must be possible to change such constraints as and when appropriate without unnecessarily affecting existing applications
- database must be independent of application that uses it; all its integrity constraints can be independently modified without need of any change in application; makes database independent of front-end application & interface

# Rule 11: Distribution Independence

- distribution of portions of database to various locations should be invisible to users; existing applications should continue to operate
  - when distributed version is first introduced
  - when existing distributed data are redistributed
- end-user must not be able to see that data is distributed over various locations; users should always get impression that data is located at one site only;
- regarded as foundation of distributed database

# Rule 12: Non-Subversion Rule

- if system provides low-level (record-at-a-time) interface, then that interface cannot be used to subvert system; bypassing relational security or integrity constraint
- if system has an interface that provides access to low-level records, then interface must not be able to subvert system and bypass security and integrity constraints

# Relational model

## three major components

- structure
- integrity
- manipulation

# Relation

- are defined over domains, (data types)
- domain set of values from which actual attributes in actual relations take their actual values
- n-ary relations can be pictured as table with  $n$  columns
  - columns correspond to attributes of relation
  - rows correspond to tuples

# Departments and Employees database sample values

DEPT

| DNO | DNAME       | BUDGET |
|-----|-------------|--------|
| D1  | Marketing   | 10M    |
| D2  | Development | 12M    |
| D3  | Research    | 5M     |

EMP

| ENO | ENAME | DNO | SALARY |
|-----|-------|-----|--------|
| E1  | Lopez | D1  | 40K    |
| E2  | Cheng | D1  | 42K    |
| E3  | Finzi | D2  | 30K    |
| E4  | Saito | D2  | 35K    |



DEPT.DNO referenced by EMP.DNO

# Keys

- every relation has at least one *candidate key* - unique identifier
- combination of attributes such that every tuple in relation has unique value for combination in question
- *primary key* is candidate key that's been singled out for special treatment

# Keys

- candidate keys, not primary keys, are significant from relational point of view
- *foreign key* is set of attributes in one relation whose values are required to match values of some candidate key in some other relation (or possibly in same relation)

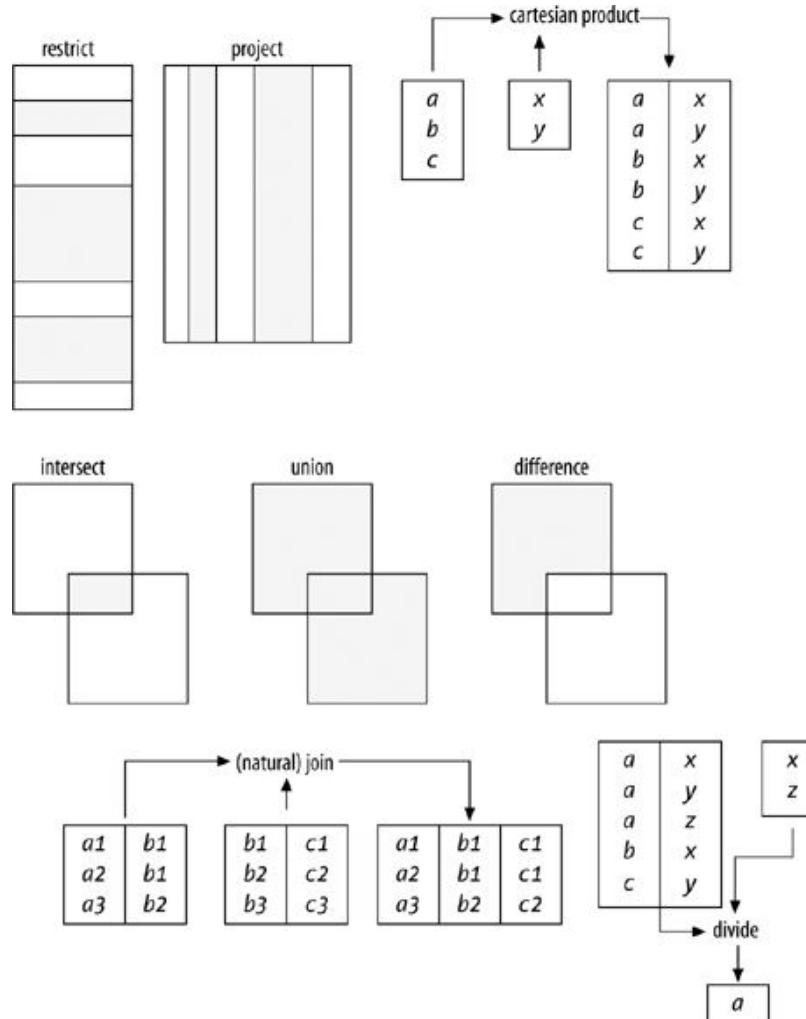
# Integrity Constraint

- basically just boolean expression that must evaluate to TRUE
  - in case of departments and employees, for example, we might have constraint to effect that SALARY values must be greater than 0
- Entity integrity
  - primary key attributes don't permit nulls.
- Referential integrity
  - there mustn't be any unmatched foreign key values
- null is "marker" that means value unknown

# Manipulation

- set of relational operators, collectively called relational algebra
- relational assignment operator

# Relational Algebra



# Properties of Relations

- every relation has *heading* and *body*
- heading is set of attributes (name, type pair)
- body is set of tuples that conform to that heading
- relation doesn't really contain tuples, it contains body, and that body in turn contains tuples

# Properties of Relations

- number of attributes in heading is the *degree (arity)*, and number of tuples in body is the *cardinality*
- tuples of relation are unordered; follows because body is set, and sets in mathematics have no ordering to their elements
- attributes of relation are also unordered because heading too is mathematical set

# Properties of Relations

- relations never contain duplicate tuples;  
follows because body is set of tuples, and  
sets in mathematics do not contain  
duplicate elements
- relational operations always produce result  
without duplicate tuples
- SQL results are allowed to contain  
duplicate rows

# Properties of Relations

- relations are always normalized (1NF); at every row-and-column intersection we always see just single value

# Tuple

- Let  $T_1, T_2, \dots, T_n$ , ( $n > 0$ ) be type names, not necessarily all distinct.
- Associate with each  $T_i$  a distinct attribute name,  $A_i$ ; each of the  $n$  attribute-name : type-name combinations that results is an attribute.
- Associate with each attribute a value  $v_i$  of type  $T_i$ ; each of the  $n$  attribute : value combinations that results is component.
- set of all  $n$  components thus defined,  $t$  say, is tuple value (or just tuple for short) over the attributes  $A_1, A_2, \dots, A_n$
- value  $n$  is the degree of  $t$ ;
- set of all  $n$  attributes is the heading of  $t$

# Relation

- Let  $\{H\}$  be a tuple heading and let  $t_1, t_2, \dots, t_n$ , ( $n > 0$ ) be distinct tuples with heading  $\{H\}$ .
- The combination,  $r$  say, of  $\{H\}$  and the set of tuples  $\{t_1, t_2, \dots, t_n\}$  is a relation value (or just relation for short) over attributes  $A_1, A_2, \dots, A_n$ , where  $A_1, A_2, \dots, A_n$  are the attributes in  $\{H\}$ .
- heading of  $r$  is  $\{H\}$
- $r$  has same attributes that heading does
- body of  $r$  is set of tuples  $\{t_1, t_2, \dots, t_n\}$

- theory and practice of relational database
- treatment of the theory
- rigorous and mathematical
- presents two formal query languages associated with relational model
- query languages are specialized languages for asking questions that involve data in a database

- convinced that familiarity with areas of mathematics that will be presented and on which the relational model of data is based, is a strong prerequisite for anybody who aims to be professionally involved with databases

# Mathematics

- always strive for elegance and orthogonality - dislike exceptions
- system is said to be designed in an *orthogonal* way if its set of components that together make up whole system capability are *non-overlapping* and *mutually independent*
  - each capability should be implemented by only one component
  - one component should only implement one capability of system

# Mathematics

- well-separated and independent components ensure that there are no side effects: using or even changing one component does not cause side effects in another area of system
- two formal disciplines, most relevant ones in application of mathematics to field of databases
- Logic
- Set theory

# Mathematics

- *Everything should be made as simple as possible, but not simpler.*
  - Albert Einstein (1879–1955)

# **RELATIONS**

# Relation

- common misconception that word relational in “relational model of data, relational database” refers to relationships” (many-to-one, many-to-many, and so on) that can exist between different entity types
- nothing to do with R of ERD
- Entity Relationship Diagrams

# Relation

- word refers to mathematical concept of a (n-ary) relation, mathematical, set-theory concept
- **Binary Relations**

# Binary Relations

- from set A to set B is defined as subset of Cartesian product  $A \times B$
- called binary relation because it deals with two sets (A and B)
- *order of coordinates of ordered pair is important*
- example based on  $A := \{X, Y, Z\}$ ,  $B := \{1, 2\}$
- $R1 := \{(X; 1), (X; 2), (Y; 1), (Z; 2)\}$
- $(\forall p \in R1: p \in A \times B)$

# Binary Relations

- $R$  is a binary relation from set  $A$  to set  $B \leftrightarrow R \in \wp(A \times B)$
- **Functions**

# Functions

- binary relation that doesn't have two elements that have same first coordinate
- in function two distinct ordered pairs always have different first coordinates
- $F \in \wp(A \times B) \wedge$
- $(\forall p_1, p_2 \in F: p_1 \neq p_2 \rightarrow \pi_1(p_1) \neq \pi_1(p_2)) \wedge$
- $\{\pi_1(p_1) \mid p \in F\} = A$

# Functions

- $\text{dom}(F) = A$
- $\text{rng}(F) \subseteq B$

# Identity Function

- over  $A$  if  $A$  is domain of  $f$ , and for all elements  $x$  in set  $A$ ,  $f$  of  $x$  equals  $x$
- $\text{id}(A)$  to represent identity function over  $A$

# Subset of Function

- every proper subset of function F is again a function

# N-ary Relation

- more general concept of n-ary relation  
(that is, not just a binary one)
- concept introduced by database field

# Operations on Functions

- functions are sets, can apply set operators union, intersection, and difference

# Compatible (Joinable) Functions

- generic property that two functions F and G should have for union of those two functions to result in a function
- function F is compatible with function G  $\leftrightarrow$
- $(\forall c \in (\text{dom}(F) \cap \text{dom}(G)) : F(c)=G(c))$

# Function Composition

- let  $A$ ,  $B$ , and  $C$  be sets
- let  $f$  be a function over  $A$  into  $B$
- let  $g$  be a function over  $\text{rng}(f)$  into  $C$
- composition of functions  $f$  and  $g$ , notation  $g \circ f$ , is defined as
- $g \circ f := \{(a; g(f(a)) \mid a \in \text{dom}(f) \wedge f(a) \in \text{dom}(g)\}$

# Limitation of Function

- subset of given function, want to consider those ordered pairs whose first coordinate can be chosen from some given subset of the domain of the function
- $F \downarrow A := \{ p \mid p \in F \wedge \pi_1(p) \in A\}$

# Set Functions

- function in which every ordered pair holds a set as its second coordinate
- $H := \{ (\text{team}; \{\text{'Cowboys'}, \text{'Vikings'}, \text{'Saints'}, \text{'Cardinals'}\}), \text{location}; \{\text{'Dallas'}, \text{'Minnesota'}, \text{'New Orleans'}, \text{'Arizona'}\}) \}$

- can consider set function  $H$  to enumerate, through first coordinates of its ordered pairs, two aspects of team: name of team (coordinate team) and location of team (coordinate location)
- such first coordinates are referred to as *attributes*
- attached to these attributes are the value sets as second coordinates of ordered pairs that could be considered to represent set of possible values for these attributes

# Characterizations

- can consider set function  $H$  to characterize a player
- set function called characterization when you use it to describe something in real world by listing relevant attributes in combination with set of admissible values for each attribute
- relevant **attributes** are first coordinates of ordered pairs
- sets of admissible values are their respective second coordinates **attribute value sets**

# External Predicates

- characterizations can be considered to characterize predicate in real world
- characterization introduces (names of) parameters of predicate and corresponding value sets from which these parameters can take their values

# Characterizations

- $H := \{$
- $(empno; [1..99]),$
- $(ename; varchar(10)),$
- $(born; date),$
- $(job; \{\text{'CLERK'}, \text{'SALESMAN'}, \text{'TRAINER'}, \text{'MANAGER'}, \text{'PRESIDENT'}\}),$
- $(salary; [1000..4999])$
- $\}$

# External Predicates

- previous characterization characterizes following predicate:
- employee ENAME is assigned employee number EMPNO, is born at date BORN, holds position JOB, and has a monthly salary of SALARY dollars

- characterization will be basis of table design
- predicate characterized by characterization of table design represents user-understood meaning of that table design
- such predicate is referred to as external predicate of that table

# Generalized Product of Set Function

- let  $F$  be set function
- generalized product of  $F$ , notation  $\Pi(F)$
- $\Pi(F) = \{f \mid f \text{ is a function} \wedge \text{dom}(f) = \text{dom}(F) \wedge (\forall c \in \text{dom}(f): f(c) \in F(c))\}$

# Generalized Product of Set Function

- generates set with all possible functions that have same domain as  $F$ , and for which the ordered pairs (in these functions) have second coordinate that is chosen from (that is, is an element of) the relevant set that was attached to same first coordinate in  $F$

- $\Pi(\{(a; \{1,2,3\}), (b; \{4,5\})\}) = \{$
- $\{(a; 1), (b; 4)\}, \{(a; 1), (b; 5)\},$
- $\{(a; 2), (b; 4)\}, \{(a; 2), (b; 5)\},$
- $\{(a; 3), (b; 4)\}, \{(a; 3), (b; 5)\} \}$

- $H := \{ (\text{team}; \{\text{'Cowboys'}, \text{'Vikings'}, \text{'Saints'}, \text{'Cardinals'}\}), \text{location}; \{\text{'Dallas'}, \text{'Minnesota'}, \text{'New Orleans'}, \text{'Arizona'}\}) \}$
- $\Pi(H) =$

- $\Pi(\mathsf{H}) = \{$
- $\{(team; 'Cowboys'), (location; 'Dallas')\}, \{(team; 'Cowboys'), (location; 'Minnesota')\},$
- $\{(team; 'Cowboys'), (location; 'New Orleans')\}, \{(team; 'Cowboys'), (location; 'Arizona')\},$
- $\{(team; 'Vikings'), (location; 'Dallas')\}, \{(team; 'Vikings'), (location; 'Minnesota')\},$
- $\{(team; 'Vikings'), (location; 'New Orleans')\}, \{(team; 'Vikings'), (location; 'Arizona')\},$
- $\{(team; 'Saints'), (location; 'Dallas')\}, \{(team; 'Saints'), (location; 'Minnesota')\},$
- $\{(team; 'Saints'), (location; 'New Orleans')\}, \{(team; 'Saints'), (location; 'Arizona')\},$
- $\{(team; 'Cardinals'), (location; 'Dallas')\}, \{(team; 'Cardinals'), (location; 'Minnesota')\},$
- $\{(team; 'Cardinals'), (location; 'New Orleans')\}, \{(team; 'Cardinals'), (location; 'Arizona')\} \}$

- refer to functions in result set of generalized product of set function as ***tuples***
- such set functions will typically signify a characterization
- elements of tuple, ordered pairs, are called attribute-value pairs of tuple
- in relational database design, tuple represents proposition in real world

- tuple {(empno; 100), (ename;'Smith'),  
(born; '19-jan-1964'), (job; 'MANAGER'),  
(salary; 5000) }
- denotes proposition “Employee Smith is assigned employee number 100, is born at date 19-jan-1964, holds position MANAGER, and has a monthly salary of 5000 dollars”

# Closed world assumption

- if given tuple appears in table design, then we assume that proposition denoted by that tuple is currently true proposition in real world

- other way around, if a given tuple that could appear in table design, but currently does not appear, then we assume that proposition denoted by that tuple is false proposition in real world
- principle is referred to as "Closed World Assumption" tuples held in database design describe all, and only, currently true propositions in real world

# **APPLICATION**

- mathematical foundation laid down
- how this can be applied to specify database and database designs

# Key Terms

- Entity, Entity identifier, Keys
- Attribute, Column, Domain of values
- Tuple, Row
- Equijoin
- Relation, Relational model, Relational database
- Logic

# **DataBase**

## **Relational Queries**

# Query languages

- specialized languages for asking questions, or *queries*, that involve the data in a database
- **relational algebra** - queries are specified *operational*
  - set of operators
  - each query describes a step-by-step procedure for computing the desired answer
- **relational calculus** - queries are specified nonprocedural *declarative*
- expressive power of algebra and calculus ?
- these formal query languages have greatly influenced commercial query languages such as SQL, which we will discuss later

# Influence on the design of commercial query languages

- Structured Query language (SQL)
- Query-by-Example (QBE)

- inputs and outputs of a query are relations
- present a number of sample queries using following schema:
- Sailors(*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
- Boats(*bid*: integer, *bname*: string, *color*: string)
- Reserves(*sid*: integer, *bid*: integer, *day*: date)

| <i>sid</i> | <i>sname</i> | <i>rating</i> | <i>age</i> |
|------------|--------------|---------------|------------|
| 22         | Dustin       | 7             | 45.0       |
| 31         | Lubber       | 8             | 55.5       |
| 58         | Rusty        | 10            | 35.0       |

Figure 4.1 Instance  $S1$  of Sailors

| <i>sid</i> | <i>sname</i> | <i>rating</i> | <i>age</i> |
|------------|--------------|---------------|------------|
| 28         | yuppy        | 9             | 35.0       |
| 31         | Lubber       | 8             | 55.5       |
| 44         | guppy        | 5             | 35.0       |
| 58         | Rusty        | 10            | 35.0       |

Figure 4.2 Instance  $S2$  of Sailors

| <i>sid</i> | <i>bid</i> | <i>day</i> |
|------------|------------|------------|
| 22         | 101        | 10/10/96   |
| 58         | 103        | 11/12/96   |

Figure 4.3 Instance  $R1$  of Reserves

# **Relational Algebra**

# Relational algebra

- queries composed using a collection of operators
- every operator in algebra accepts (one or two) relation instances as arguments and returns a relation instance as the result
- compose operators to form a complex query
- *expression* is recursively defined to be
  - Relation
  - unary algebra operator applied to a single expression
  - binary algebra operator applied to two expressions

# Relational query

- step-by-step procedure for computing the desired answer
- procedural nature of the algebra
  - algebra expression is a recipe, or a plan, for evaluating a query
- relational systems in fact use algebra expressions to represent query evaluation plans

# Selection and Projection

- operators to
  - *select* rows from a relation ( $\sigma$ )
  - *project* columns ( $\pi$ )
- operations allow us to manipulate data in a single relation

# Instance of Sailors $S_2$

| <i>sid</i> | <i>sname</i> | <i>rating</i> | <i>age</i> |
|------------|--------------|---------------|------------|
| 28         | yuppy        | 9             | 35.0       |
| 31         | Lubber       | 8             | 55.5       |
| 44         | guppy        | 5             | 35.0       |
| 58         | Rusty        | 10            | 35.0       |

$$\sigma_{\text{rating} > 8} (S_2)$$

| <i>sid</i> | <i>sname</i> | <i>rating</i> | <i>age</i> |
|------------|--------------|---------------|------------|
| 28         | yuppy        | 9             | 35.0       |
| 58         | Rusty        | 10            | 35.0       |

$$\pi_{\text{name}, \text{rating}}(S_2)$$

| <i>sname</i> | <i>rating</i> |
|--------------|---------------|
| yuppy        | 9             |
| Lubber       | 8             |
| guppy        | 5             |
| Rusty        | 10            |

# Selection operator

$\sigma_{selection\_condition}$

- specifies the tuples to retain through a *selection condition*
  - boolean expression using logical connectives  $\vee$  and  $\wedge$  of *terms* that have the form *attribute*<sub>1</sub>, op *attribute*<sub>2</sub>, where op is one of comparison operators  $<$ ;  $\leq$ ;  $=$ ;  $\geq$ ;  $>$
- reference to an attribute can be by position or by name
- schema of result is schema of input relation instance

# Projection operator

$$\pi \text{ } column\_list$$

- projection operator allows us to extract columns from a relation
- schema of result of projection is determined by fields that are projected in the *column list*
- result is a relation

$$\pi_{age}(S_2)$$

| <i>sid</i> | <i>sname</i> | <i>rating</i> | <i>age</i> |
|------------|--------------|---------------|------------|
| 28         | yuppy        | 9             | 35.0       |
| 31         | Lubber       | 8             | 55.5       |
| 44         | guppy        | 5             | 35.0       |
| 58         | Rusty        | 10            | 35.0       |

$$\pi_{age}(S_2)$$

| <i>sid</i> | <i>sname</i> | <i>rating</i> | <i>age</i> |
|------------|--------------|---------------|------------|
| 28         | yuppy        | 9             | 35.0       |
| 31         | Lubber       | 8             | 55.5       |
| 44         | guppy        | 5             | 35.0       |
| 58         | Rusty        | 10            | 35.0       |

|            |
|------------|
| <i>age</i> |
| 35.0       |
| 55.5       |

# Projection $\pi$ result is a relation

- follows from the definition of a relation as a *set* of tuples
- in practice, real systems often omit the expensive step of eliminating *duplicate tuples*, leading to relations that are multisets

# Composition of relational algebra operators

- since result of a relational algebra expression is always a relation, we can substitute an expression wherever a relation is expected
- for example, we can compute the names and ratings of highly rated sailors by combining two of the preceding queries
- $\pi_{sname, rating} (\sigma_{rating > 8} (S_2))$

# Set Operations

- following standard operations on sets are also available in relational algebra
  - *union* ( $\cup$ )
  - *intersection* ( $\cap$ )
  - *set-difference* ( $-$ )
  - *cross-product* ( $\times$ )

# Union ( $\cup$ )

- $R \cup S$  returns a relation instance containing all tuples that occur in *either* relation instance  $R$  or relation instance  $S$  (or both)

# Union-compatible

- $R$  and  $S$  must be *union-compatible*, and the schema of the result is defined to be identical to the schema of  $R$  (or  $S$ )
  - they have the same number of fields
  - corresponding fields, taken in order from left to right, have the same *domains*
- for convenience, we will assume that the fields of  $R \cup S$  inherit names from  $R$

# Intersection ( $\cap$ )

- $R \cap S$  returns a relation instance containing all tuples that occur in *both*  $R$  and  $S$
- relations  $R$  and  $S$  must be union-compatible, and the schema of the result is defined to be identical to the schema of  $R$

# Set-difference ( $-$ )

- $R-S$  returns a relation instance containing all tuples that occur in  $R$  but not in  $S$
- relations  $R$  and  $S$  must be union-compatible, and the schema of the result is defined to be identical to the schema of  $R$

# Cross-product ( $\times$ )

- $R \times S$  returns a relation instance whose schema contains all the fields of  $R$  (in the same order as they appear in  $R$ ) followed by all the fields of  $S$  (in the same order as they appear in  $S$ )
- result of  $R \times S$  contains one tuple  $\langle r, s \rangle$  (the concatenation of tuples  $r$  and  $s$ ) for each pair of tuples  $r \in R; s \in S$
- sometimes called ***Cartesian product***

# Cross-product naming conflict

- convention that fields of  $R \times S$  inherit names from the corresponding fields of  $R$  and  $S$
- possible for both  $R$  and  $S$  to contain one or more fields having the same name
  - corresponding fields in  $R \times S$  are unnamed and are referred to solely by position

# Renaming operator $\rho$

- $\rho (R(F), E)$
- takes an arbitrary relational algebra expression  $E$  and returns an instance of a (new) relation called  $R$
- $R$  contains same tuples as result of  $E$ , and has same schema as  $E$ , but some fields are renamed
- field names in relation  $R$  are same as in  $E$ , except for fields renamed in *renaming list*  $F$

# Renaming list

- terms having the form  $oldname \rightarrow newname$  or  $position \rightarrow newname$
- to be well-defined, references to fields may be unambiguous, and no two fields in result must have the same name

# Additional operators

- customary to include some additional operators in algebra, but they can all be defined in terms of already defined operators
- renaming operator needed for syntactic convenience
- can consider even  $\cap$  operator as redundant,  $R \cap S$  can be defined as  $R - (R - S)$

# Join operation

- one of the most useful operations in relational algebra
- most commonly used way to combine information from two or more relations

# Join operation

- can be defined as a cross-product followed by selections and projections, joins arise much more frequently in practice than plain cross-products
- important to recognize joins and implement them without materializing the underlying cross-product
- variants of join operation

# Condition Joins

- most general version of the join operation accepts a *join condition*  $c$  and a pair of relation instances as arguments, and returns a relation instance
- *join condition* is identical to a *selection condition* in form

# Condition Joins $|><|_c$

- $R |><|_c S = \sigma_c (R \times S)$
- defined to be a cross-product followed by a selection
- condition  $c$  can (and typically does) refer to attributes of both  $R$  and  $S$
- reference to an attribute of a relation, say  $R$ , can be by position (of form  $R.i$ ) or by name (of form  $R.name$ )

# Equijoin $|><|_c$

- special case of join operation  $R |><|S$  is when *join condition* consists solely of equalities (connected by  $\wedge$ ) of form  $R.name_1 = S.name_2$ , that is, equalities between two fields in  $R$  and  $S$
- obviously, there is some redundancy in retaining both attributes in result
- join operation is refined by doing an additional projection in which  $S.name_2$  is dropped

$$S_1 |><| R.\text{sid}=\text{S}.\text{sid} \quad R_1$$

| <i>sid</i> | <i>sname</i> | <i>rating</i> | <i>age</i> | <i>bid</i> | <i>day</i> |
|------------|--------------|---------------|------------|------------|------------|
| 22         | Dustin       | 7             | 45.0       | 101        | 10/10/96   |
| 58         | Rusty        | 10            | 35.0       | 103        | 11/12/96   |

# Natural Join $|><|$

- further special case of the join operation  $R |><| S$  is an equijoin in which equalities are specified on *all* fields having the same name in  $R$  and  $S$
- we can simply omit the join condition
- default is that join condition is a collection of equalities on all common fields
- result is guaranteed not to have two fields with the same name

- if the two relations have no attributes in common,  $R |><| S$  is simply the cross-product
- several variants of joins that are not discussed
- *outer joins*

# **Relational Calculus**

# Relational Calculus

- nonprocedural, or *declarative*,
- describe set of answers without being explicit about how they should be computed

# Relational Calculus

## 2 variants

- tuple relational calculus (TRC)
  - variables in TRC take on tuples as values
  - influence on SQL
- domain relational calculus (DRC)
  - variables range over field values
  - influenced QBE

# **Tuple Relational Calculus**

# Tuple Relational Calculus

- *tuple variable* is a variable that takes on tuples of a particular relation schema as values
  - every value assigned to a given tuple variable has the same number and type of fields

# Tuple Relational Calculus

- tuple relational calculus query has the form  $\{T \mid p(T)\}$ , where  $T$  is a tuple variable and  $p(T)$  denotes a *formula* that describes  $T$ 
  - where  $T$  is the only free variable in formula  $p$
- result of this query is the set of all tuples  $t$  for which the formula  $p(T)$  evaluates to true with  $T = t$
- language for writing formulas  $p(T)$  is thus at the heart of TRC and is essentially a simple subset of *first-order logic*

# Tuple Relational Calculus

- As a simple example, consider the following query.
  - *Find all sailors with a rating above 7?*

# Tuple Relational Calculus

- $\{S \mid S \in \text{Sailors} \wedge S.\text{rating} > 7\}$
- answer contains those instances of  $S$  that pass this test

# Syntax of TRC Queries

- $Rel$  - relation name
- $R$  and  $S$  - tuple variables
  - $a$  - an attribute of  $R$
  - $b$  - an attribute of  $S$
- op - an operator in the set  $\{<, \leq, \neq, \geq, >\}$

# Atomic formula in TRC

- $R \in Rel$
- $R.a$  op  $S.b$
- $R.a$  op *constant*
- *constant* op  $R.a$

# TRC Formula

- recursively defined to be one of the following, where  $p$  and  $q$  are themselves formulas and  $p(R)$  denotes a formula in which variable  $R$  appears
- any atomic formula
- $\neg p$ ,  $p \vee q$ ,  $p \wedge q$ ,  $p \Rightarrow q$
- $\exists R( p(R))$ , where  $R$  is a tuple variable
- $\forall R( p(R))$ , where  $R$  is a tuple variable

- In the last two clauses *quantiers*  $\forall$  and  $\exists$  are said to *bind* the variable  $R$
- variable is said to be *free* in a formula or *subformula* (a formula contained in a larger formula) if the (sub)formula does not contain an occurrence of a quantier that binds it

- observe that every variable in a TRC formula appears in a subformula that is atomic, and every relation schema specifies a domain for each field
- ensures that each variable in TRC formula has a well-defined domain from which values for the variable are drawn
  - each variable has a well-defined *type*

# Examples of TRC Queries

- *Find the sailor name, boat id, and reservation date for each reservation?*
- $\{P \mid \exists R \in \text{Reserves} \wedge \exists S \in \text{Sailors}$
- $(R.sid = S.sid \wedge P.bid = R.bid \wedge P.day = R.day \wedge P.sname = S.sname) \}$

# Examples of TRC Queries

- for each Reserves tuple, we look for a tuple in Sailors with the same *sid*
- given a pair of such tuples, we construct an answer tuple  $P$  with fields *sname*, *bid*, and *day* by copying corresponding fields from these two tuples

# **Domain relational calculus**

# Domain relational calculus

- ***domain variable*** ranges over values in domain of some attribute

# Domain relational calculus

- domain relational calculus query has the form  $\{<x_1, x_2, \dots, x_n> \mid p(<x_1, x_2, \dots, x_n>)\}$
- where each  $x_i$  is either a *domain variable* or a constant and  $p(<x_1, x_2, \dots, x_n>)$  denotes a *DRC formula* whose only free variables are the variables among the  $x_i$ ,  $1 \leq i \leq n$
- result of this query is the set of all tuples  $\{<x_1, x_2, \dots, x_n>$  for which the formula evaluates to true

- A DRC formula is defined in a manner that is very similar to the definition of a TRC formula.
- difference is that the variables are now domain variables

# Syntax of DRC Queries

- op - an operator in the set { $<$ ,  $\leq$ ,  $\neq$ ,  $\geq$ ,  $>$ }
- $X$  and  $Y$  be domain variables

# Atomic formula in DRC

- $\{x_1, x_2, \dots, x_n\} \in Rel$ 
  - where  $Rel$  is a relation with  $n$  attributes
  - each  $x_i$ ,  $1 \leq i \leq n$  is either variable or constant
- $X \text{ op } Y$
- $X \text{ op } \text{constant}$
- $\text{constant} \text{ op } X$

# DRC Formula

- recursively defined to be one of the following, where  $p$  and  $q$  are themselves formulas and  $p(X)$  denotes a formula in which variable  $X$  appears
- any atomic formula
- $\neg p, p \vee q, p \wedge q, p \Rightarrow q$
- $\exists R( p(R))$ , where  $R$  is a domain variable
- $\forall R( p(R))$ , where  $R$  is a domain variable

# Examples of DRC Queries

- *Find all sailors with a rating above 7?*
- $\{<I, N, T, A> \mid \{<I, N, T, A> \in \text{Sailors} \wedge T > 7\}$

# **Expressive power of algebra and calculus**

- every query that can be expressed in relational algebra also can be expressed in safe relational calculus

# Unsafe queries

- consider the query  $\{S \mid \neg(S \in \text{Sailors})\}$ 
  - syntactically correct
  - set of such  $S$  tuples is obviously infinite, in the context of infinite domains
- restrict relational calculus to disallow unsafe queries

- if query language can express all queries that we can express in relational algebra, it is said to be *relationally complete*
- practical query language is expected to be relationally complete
- in addition, commercial query languages typically support features that allow us to express some queries that cannot be expressed in relational algebra

# Structured Query Language SQL

- is the most widely used commercial relational database language
- originally developed at IBM in System-R projects
- other vendors introduced DBMS products based on SQL, and it is now a de facto standard
- current ANSI/ISO standard for SQL-92

# Key Terms

- Relational algebra
- Relational calculus
- Data retrieval
- Select operator
- Join operator
- Natural join

# **Thank you for your kindly attention!**

# DataBase

# **Reports**

# OverView

- Reports
- Access
- Access Reports
- BIRT

# Practical Alternatives

- BIRT – Business Intelligence Reporting Tool (based on Java)
- .NET Reporting Services
  - SQL Server Reporting Services – SSRS
  - Support in .NET
- MS Access

# **Access**

# Access

- tool for managing databases
- build Database Application
  - build full user interface for database
- build Web Database Application
- DataBase Management System (DBMS), part of Microsoft Office that combines relational DataBase engine with graphical user interface and software-development tools

# Access

- stores data in its own format Jet Database Engine
  - can also import or link directly to data stored in other applications and databases
- supported by Visual Basic for Applications (VBA), object-based programming language that can reference variety of objects including DAO (Data Access Objects), ActiveX Data Objects, ....
- visual objects used in forms and reports expose their methods and properties in VBA program environment, VBA code modules may declare and call Windows operating system operations

# Access

- Jet Database format (MDB or ACCDB 2007) can contain application and data in one file
- makes it very convenient to distribute entire application to another user, who can run it in disconnected environments
- does not implement database triggers, stored procedures, or transaction logging
- Access 2010 includes table-level triggers and stored procedures built into data engine

# Access

- users can create tables, queries, forms and reports, connect with macros; use VBA to write rich solutions with advanced data manipulation and user control
- import and export of data to many formats
- often used by people downloading data from enterprise level databases for manipulation, analysis, and reporting

# Access

- from programmer's perspective is its relative compatibility with SQL (Structured Query Language)
- parameterized queries; parameterized stored procedures
- ASP.NET web forms can query Access database, retrieve records and display them on browser

- other personal / office database tools:
- Filemaker
- Kexi

# FileMaker

- cross-platform relational database application from a subsidiary of Apple Inc
- it integrates database engine with graphical user interface (GUI), allowing users to modify database by dragging elements into layouts, screens, or forms
- developed primarily for Apple Macintosh
- available for Microsoft Windows
- desktop, server, iOS and web-delivery

# Kexi

- Kexi (Access for Linux) visual database applications creator tool by KDE, designed to fill gap between spreadsheets and database solutions requiring more sophisticated development
- can be used for designing and implementing databases, data inserting and processing, and performing queries
- lack of applications having features of Microsoft Access, FoxPro, Oracle Forms or FileMaker

# Kexi

- member of Calligra Suite
- works under Linux/Unix and Solaris operating systems
- application and libraries available under LGPL

# Access objects

- Tables
- Queries – Data Sets
- Forms
- Reports

# Tables

- Set Up Connection to SQL Server
- External Data → Import & Link → ODBC Database
- New Machine Data Source
- Pick latest driver, which will have name like SQL Server Native Client 11.0
  - Change default database to ...
- External Data → Import & Link → Linked Table Manager
- Access adds each linked table to your database

# Queries

- Manipulate Data with Queries
- like Data Sets
- external user view
- Structured Query Language
- Query Wizard, Query Designer
  - SQL view (for users from Comp. Sci. UTCN)
  - Design view (for all the rest ...)

# Reports

# Access Report

- experienced report writers build report from scratch using Layout view or Design view
- newbies can create simple one-click report with single click
- Create → Report section of toolbar

```
SELECT * FROM Categories
```

# Create → Report

The screenshot shows the Microsoft Access ribbon with the 'Create' tab selected. In the 'Reports' section, the 'Report' icon is highlighted with a mouse cursor. To the left, the 'Tables' section shows the 'dbo\_Categories' table selected. The table data grid displays four rows of category information:

| CategoryID | CategoryName | Description                                         | Picture    |
|------------|--------------|-----------------------------------------------------|------------|
| 1          | Beverages    | Soft drinks, coffee and tea                         | OLE object |
| 2          | Condiments   | Sweet and savory sauces, relishes, spreads and oils | OLE object |
| 3          | Confections  | Desserts, candies and jellies                       | OLE object |

dbo\_Categories

The screenshot shows a Microsoft Access database table titled 'dbo\_Categories'. The table has columns: CategoryID, CategoryName, Description, and Picture. The data is as follows:

| CategoryID | CategoryName   | Description                                                | Picture |
|------------|----------------|------------------------------------------------------------|---------|
| 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 | 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                                           |         |

Page 1 of 1

# Report Layout

- create report on top of query
- Create
- choose Create → Report → Blank Report
- click field in Field List, drag it over to report in appropriate position

Field List

Show only fields in the current record source

Fields available for this view:

- dbo\_Categories Edit Table
  - CategoryID
  - CategoryName
  - Description
  - Picture
- dbo\_Products Edit Table
  - ProductID
  - ProductName
  - SupplierID
  - CategoryID
  - QuantityPerUnit
  - UnitPrice
  - UnitsInStock
  - UnitsOnOrder
  - ReorderLevel
  - Discontinued

Report1 Report1Filter1

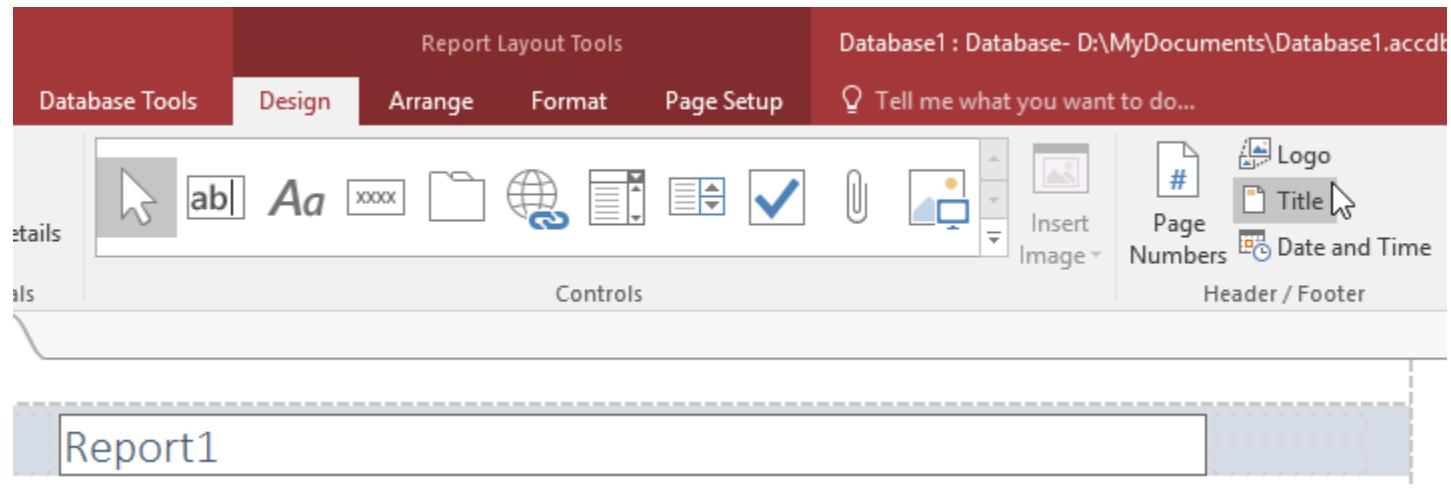
| ID of Product | Product Name                    | Unit Price | Category Name  |
|---------------|---------------------------------|------------|----------------|
| 1             | Chai                            | \$18.00    | Beverages      |
| 2             | Chang                           | \$19.00    | Beverages      |
| 3             | Aniseed Syrup                   | \$10.00    | Condiments     |
| 4             | Chef Anton's Cajun Seasoning    | \$22.00    | Condiments     |
| 5             | Chef Anton's Gumbo Mix          | \$21.35    | Condiments     |
| 6             | Grandma's Boysenberry Spread    | \$25.00    | Condiments     |
| 7             | Uncle Bob's Organic Dried Pears | \$30.00    | Produce        |
| 8             | Northwoods Cranberry Sauce      | \$40.00    | Condiments     |
| 9             | Mishi Kobe Niku                 | \$97.00    | Meat/Poultry   |
| 10            | Ikura                           | \$31.00    | Seafood        |
| 11            | Queso Cabrales                  | \$21.00    | Dairy Products |
| 12            | Queso Manchego La Pastora       | \$38.00    | Dairy Products |
| 13            | Konbu                           | \$6.00     | Seafood        |
| 14            | Tofu                            | \$23.25    | Produce        |
| 15            | Genen Shouyu                    | \$15.50    | Condiments     |
| 16            | Pavlova                         | \$17.45    | Confections    |

# Add other elements

- Report Layout Tools → Design → Header/Footer
- **title** is large-text caption that appears at top of first report page
- **logo** is tiny picture that usually sits in top-left corner of report, next to title

# Add other elements

- **page numbers** indicate current page (and, optionally, total number of pages)
- in Layout view, treats report as though all data occupies one page, so you need to scroll to end to see this element
- **date information** indicates day and time that report was run; ordinarily, appears in header section, to right of title



- tweak formatting by changing fonts, colors, and borders
- optionally, choose File → Print → Print to print report
- save your report to use later

# Conditional Formatting

- for example to spot products over \$100
- with conditional formatting emphasize these information with different formatting
- define condition that, if true, apply additional formatting to value in column
- select value in column where you want to apply conditional formatting and then choose Report Layout Tools | Format→Control Format→Conditional Formatting

- Click New Rule to create new conditional formatting rule
- default rule type, "Check values in the current record or use an expression"  
"Compare to other records," used to create data bars

Report1

|                            |          |                |
|----------------------------|----------|----------------|
| Genen Shōyu                | \$15.50  | Condiments     |
| Pavlova                    | \$17.45  | Confections    |
| Alice Mutton               | \$39.00  | Meat/Poultry   |
| Carnarvon Tigers           | \$62.50  | Seafood        |
| Teatime Chocolate Biscuits | \$9.20   | Confections    |
| Sir Rodney's Marmalade     | \$81.00  | Confections    |
| Sir Rodney's Scones        | \$10.00  | Confections    |
| Gustaf's Knäckebrot        | \$21.00  | Grains/Cereals |
| Tunnbröd                   | \$9.00   | Grains/Cereals |
| Guaraná Fantástica         | \$4.50   | Beverages      |
| NuNuCa Nuß-Nougat-Creme    | \$14.00  | Confections    |
| Gumbär Gummibärchen        | \$31.23  | Confections    |
| Schoggi Schokolade         | \$43.90  | Confections    |
| Rössle Sauerkraut          | \$45.60  | Produce        |
| Thüringer Rostbratwurst    | \$123.79 | Meat/Poultry   |
| Nord-Ost Matjeshering      | \$25.89  | Seafood        |
| Gorgonzola Telino          | \$12.50  | Dairy Products |
| Mascarpone Fabioli         | \$32.00  | Dairy Products |
| Geitost                    | \$2.50   | Dairy Products |
| Sasquatch Ale              | \$14.00  | Beverages      |
| Spiced Rum                 | \$10.00  |                |

# Views of Report

- **Layout View** - shows what report will look like when printed, complete with real data
  - use this view to format and rearrange basic building blocks of report
- **Report View** - almost same as Layout, but doesn't let you make changes
  - use Report view to copy portions of report to Clipboard, so you can paste them into other programs

# Views of Report

- **Print Preview** - shows live preview of report; preview is paginated; can also change print settings and export
- **Design View** - shows template view where you can define different sections of report; give complete, unrestrained flexibility to customize your report
- often begin creating report in Layout view and then add more exotic effects in Design

# Export report Formats

- PDF
- Word (RTF), Excel
- HTML Document
  - drawback is that formatting, layout, and pagination won't be preserved exactly, disadvantage if someone wants to print exported report
- Text
- XML

# Filter Report

- right-click any value in column, and then look for filter submenu - menu depends on data type
- right-click CategoryName field, you see submenu named Text Filters
- right-click Price field, you see submenu named Number Filters
- can apply filters to multiple columns
- to remove filter, right-click column, and choose Clear Filter

# Filter Report

Report1

Title of Report1 Saturday, Oct

| ProductName                     | UnitPrice | CategoryName   |
|---------------------------------|-----------|----------------|
| Chai                            | \$18.00   | Beverages      |
| Chang                           | \$19.00   | Beverages      |
| Aniseed Syrup                   | \$10.00   | Condiments     |
| Chef Anton's Cajun Seasoning    | \$22.00   | Condiments     |
| Chef Anton's Gumbo Mix          | \$21.35   | Condiments     |
| Grandma's Boysenberry Spread    | \$25.00   | Condiments     |
| Uncle Bob's Organic Dried Pears | \$30.00   | Produce        |
| Northwoods Cranberry Sauce      | \$40.00   | Condiments     |
| Mishi Kobe Niku                 | \$97.00   | Meat/Poultry   |
| Ikura                           | \$31.00   | Seafood        |
| Queso Cabrales                  | \$21.00   | Dairy Products |
| Queso Manchego La Pastora       | \$38.00   | Dairy Products |
| Konbu                           | \$6.00    | Seafood        |
| Tofu                            | \$23.25   | Produce        |
| Genen Shouyu                    | \$15.50   | Condiments     |

Copy Paste Paste Formatting Insert Merge/Split Layout Select Entire Row Select Entire Column Group On CategoryName Total CategoryName Sort A to Z Sort Z to A Clear filter from CategoryName Text Filters Equals "Seafood" Does Not Equal "Seafood" Contains "Seafood" Does Not Contain "Seafood" Delete Delete Row Delete Column Change To Position Gridlines

# Sort Report

- report has same order as underlying data source - if report built on query, filter and order is determined
- can apply directly in your report, right-click appropriate column header, and then look for sorting options
- if you want more complex sort that uses more than one column need to build query for your report

# Sort Report

The screenshot shows a report titled "Report1" with the main title "Title of Report". The report contains a table with four columns: ProductName, UnitPrice, and two unnamed columns represented by blue squares. A context menu is open over the "UnitPrice" column, listing options like Paste, Paste Formatting, Insert, Merge/Split, Layout, and several sorting and grouping options. The "Sort A to Z" option is highlighted with a mouse cursor.

| ProductName                  | UnitPrice |           |  |
|------------------------------|-----------|-----------|--|
| Chai                         | \$18.00   | Beverage  |  |
| Chang                        | \$19.00   | Beverage  |  |
| Aniseed Syrup                | \$10.00   | Condiment |  |
| Chef Anton's Cajun Seasoning | \$22.00   | Condiment |  |

# Design advance report in Design View

- Property - Record Source – Query
- includes placeholders where Access inserts necessary information

The screenshot shows the Microsoft Access Design View for a report named "Report1". The report layout consists of several sections: Report Header, Page Header, Detail, Page Footer, and Report Footer. The Detail section contains fields for ProductName, UnitPrice, and CategoryName. The Report Footer section contains a placeholder for page numbers: ="Page "& [Page] & " of " & [Pages]. To the right of the report, a "Property Sheet" window is open, showing the following properties for the report:

|                  |        |
|------------------|--------|
| Record Source    | Query1 |
| Filter           |        |
| Filter On Load   | No     |
| Order By         |        |
| Order By On Load | Yes    |
| Allow Filters    | Yes    |

# Design View – 5 sections

- **1 - Report Header** - appears once at beginning of report, on first page
- add titles, logos, ...
- **5 - Report Footer** - appears once at very end of report
- print summary information, copyright statements, date of printing, ...

# Design View – 5 sections

- **2 - Page Header** - appears just under report header on first page, and at top of each subsequent page
- page numbers, column headers
- **3- Detail** - repeated once for each record in report
- **4 - Page Footer** - appears at bottom of each page
- if you don't use page header for page numbers, this section provides your other option

- **label control** holds fixed text
- column headers are labels
- **text box control** holds dynamic expression, text that can change
- **image control** holds picture

- **line control** lets you draw vertical, horizontal, and even diagonal lines
- to separate content graphically
- **rectangle control** lets you draw formatted rectangles around other controls to help content stand out
- **page break control** lets you split Detail section into separate pages exactly where you want
- when printing forms that wind up on separate pages (like invoices for different customers)

# Fine-Tuning Reports with Properties

- **Format** contains options you'll change most often, including font, color, borders, and margins
- **Data** identifies where control gets its information – in Detail section identifies linked field's name
- **Event** lets you attach Visual Basic code that springs into action when something specific happens

# Expressions

- let text boxes and other controls show dynamic values – for example:
- =FirstName & " " & LastName
- Format:
- Text – Full justification, Left, Center
- Number – Right justification

# Group on

- chooses field used for grouping
- with Header section / with Footer section
  - keep group together on one page
- With ... totaled, SubTotals, ...
- can add as many levels of sorting and grouping as you want, to slice and dice your data into smaller, more tightly focused subgroups

# DashBoards

- Dashboards often provide at-a-glance views of KPIs (Key Performance Indicators) relevant to particular objective or business process
- another name for "report"
- Report (more static) / Dashboard (more dynamic)
- Charts - graphical representation of data

# **Business Intelligence**

# 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
- Eclipse sample database
- <https://www.eclipse.org/birt/documentation/sample-database.php>

# 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

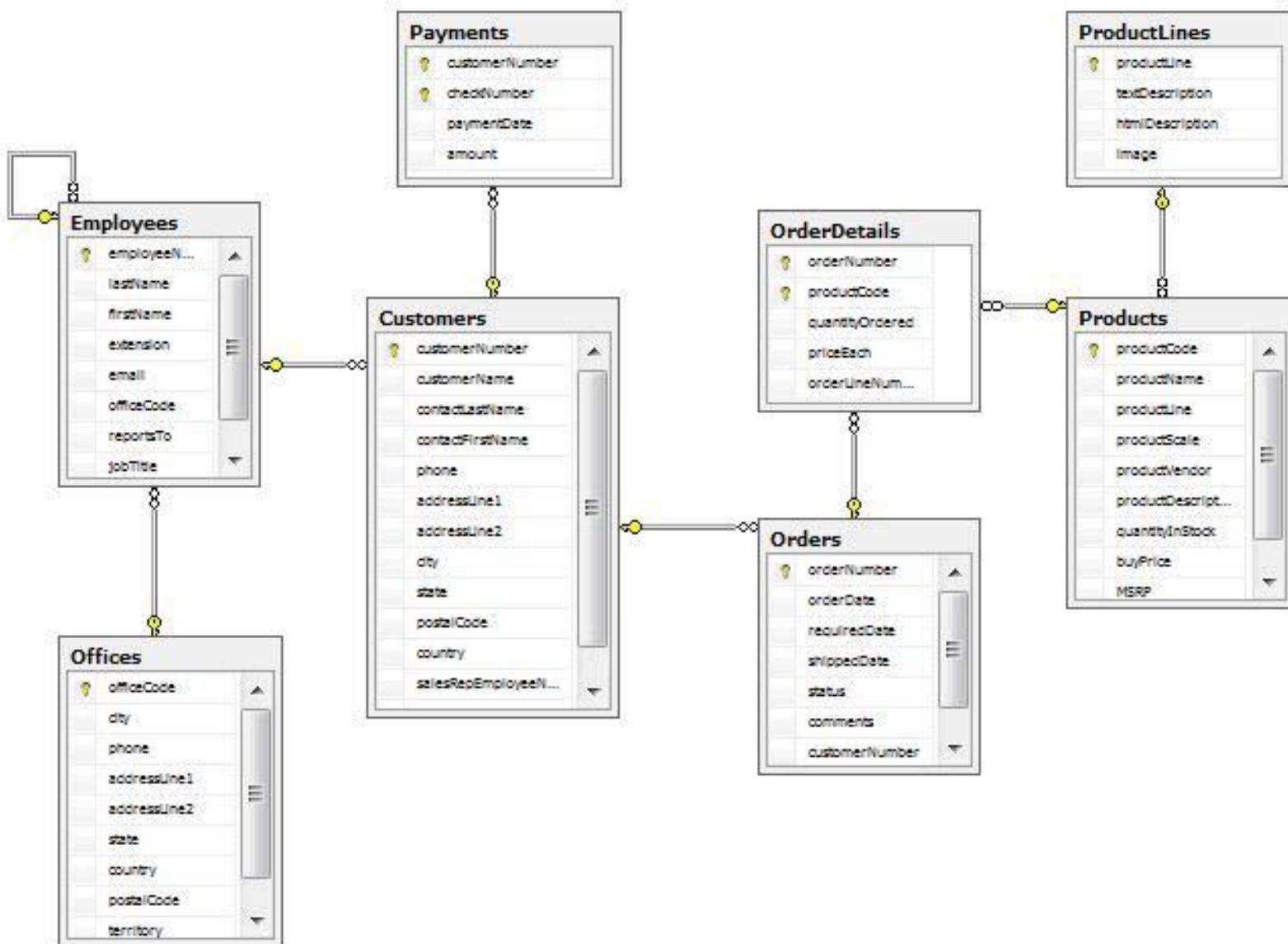
- models are classified as 7 distinct product lines
  - Classic Cars, Vintage Cars, Motorcycles, Trucks and Buses, Planes, Ships, Trains

# 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

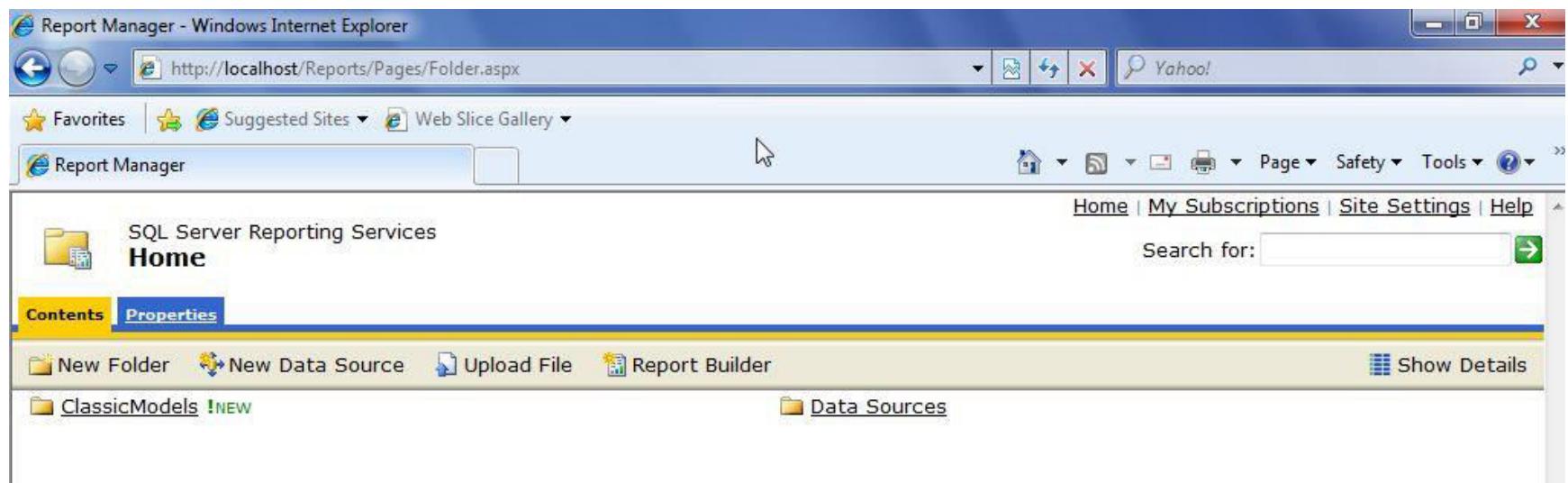
- each order contains an average of 9 unique products (order line items) with an average quantity of 35 per product
- reflects negotiated price per product
  - based on corresponding product's MSRP as well as quantity per product and maybe customer or other elements



# 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". The address bar shows the URL "http://localhost/Reports/Pages/Report.aspx?ItemPath=%2fClassicModels%2fSalesEmployee". The main content area 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 organized by Territory (APAC, EMEA) and Office (4 Paris). The report includes several rows of data for different employees and their corresponding sales details.

| 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     | 1,163    |
|           |         |                  |                            | 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 Access report window titled "Geographically". The report displays sales data in a grid format with the following columns: Territory, Country, Customer, Order Date, Buy Price, Price / Item, Quantity, Discount, and Value. The data is grouped by Territory (APAC, EMEA, Japan, NA) and Country (Australia, New Zealand, Singapore). The report includes a header bar with navigation buttons (Back, Forward, Home, Find, Next), a zoom control (100%), and export options (Select a format, Export).

| 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    |       |
|           |             | Australian Collectors, Co.   |             | 3,149.21  | 5,159.19     | 1,926    | 564.63    |       |
|           |             | Australian Gift Network, Co  |             | 900.36    | 1,528.57     | 545      | 152.3     |       |
|           |             | Souveniers And Things Co.    |             | 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*

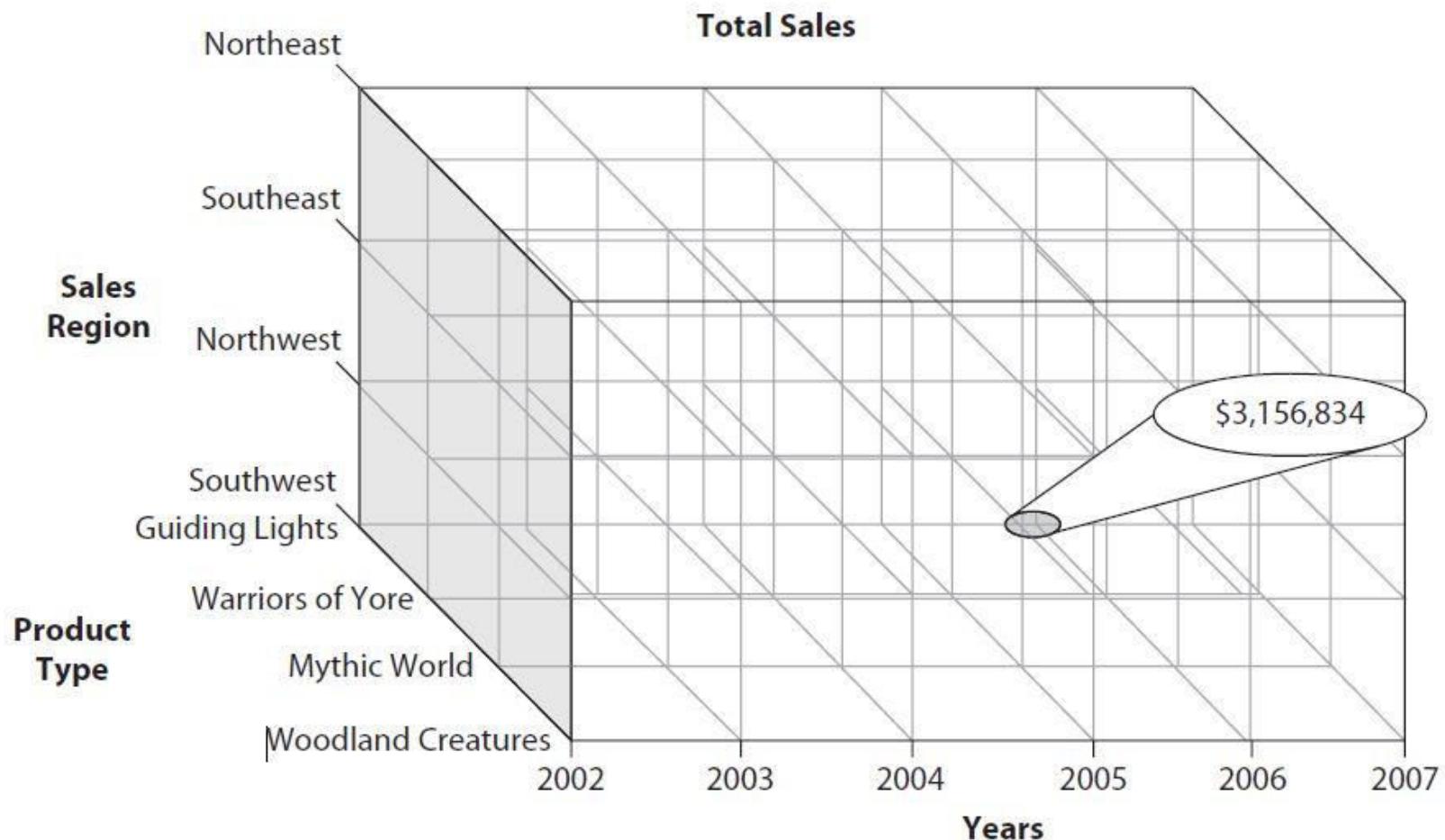
# Dimension

- 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

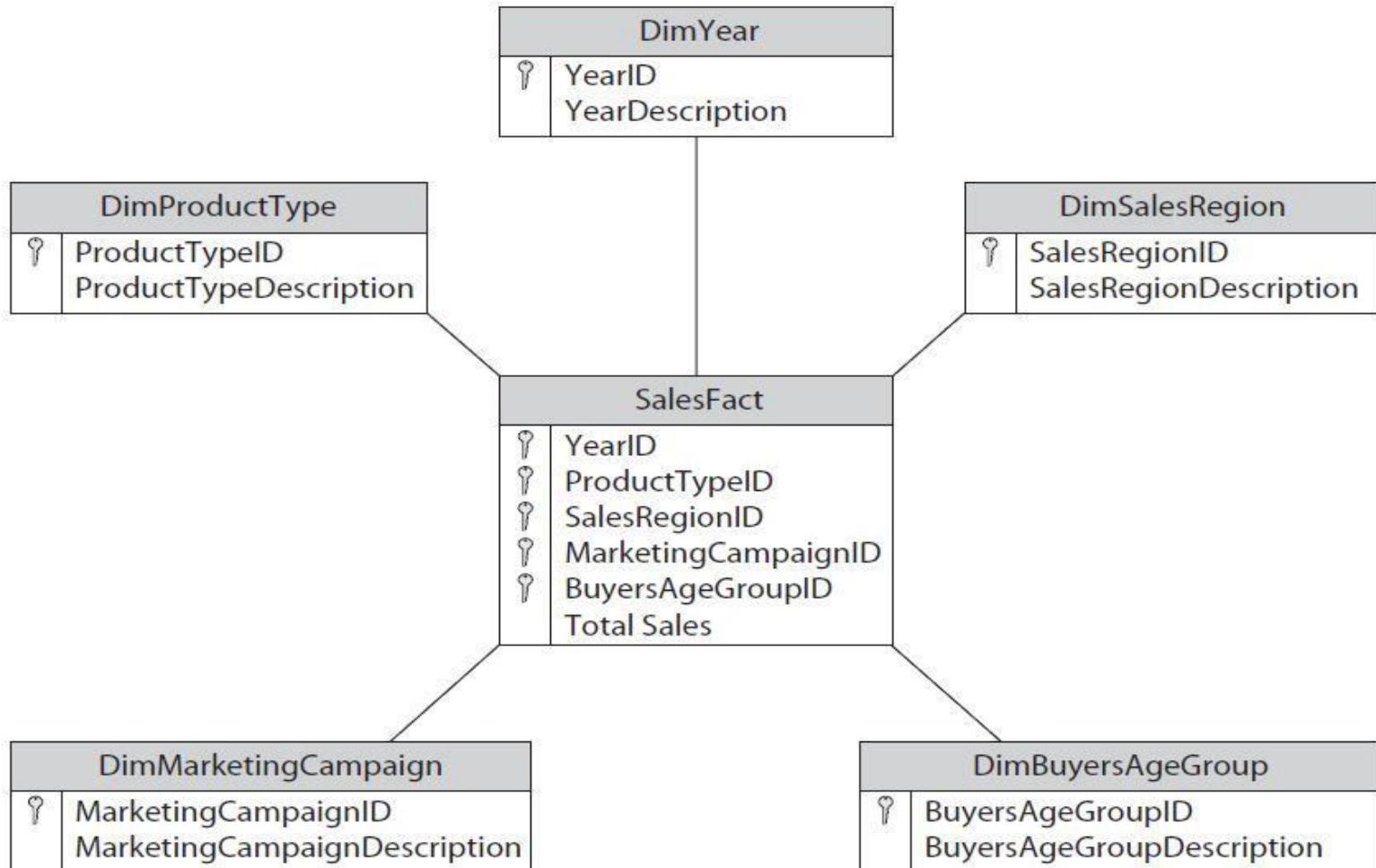
- can continue to spread out measure using additional dimensions, such as marketing campaign dimension and buyer's age
- becomes difficult to represent in illustration
- refer to these structures with four, five, or even more dimensions as cubes
- typical dimension: time, space, products

- 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

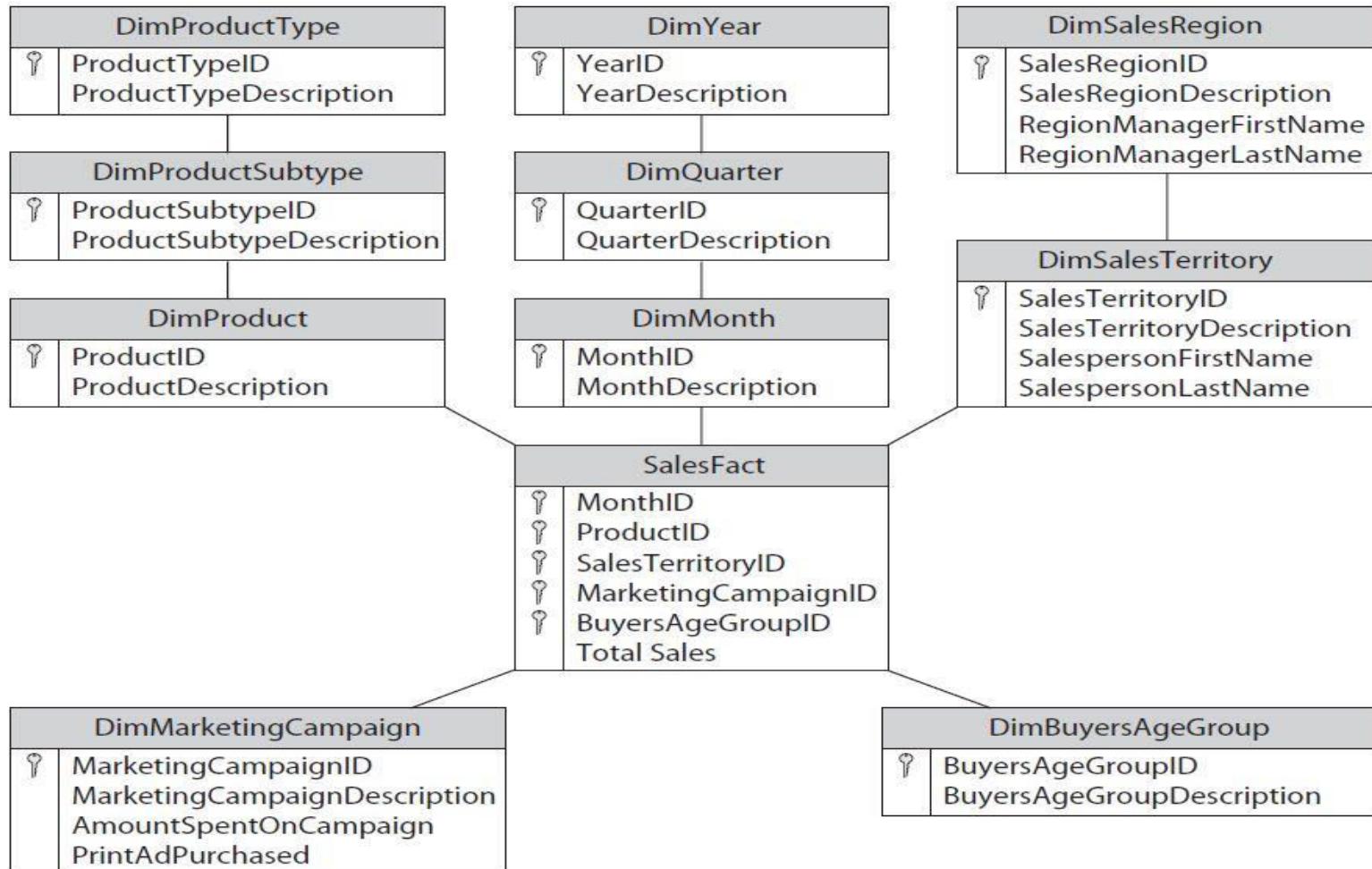
# Hierarchies

- 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

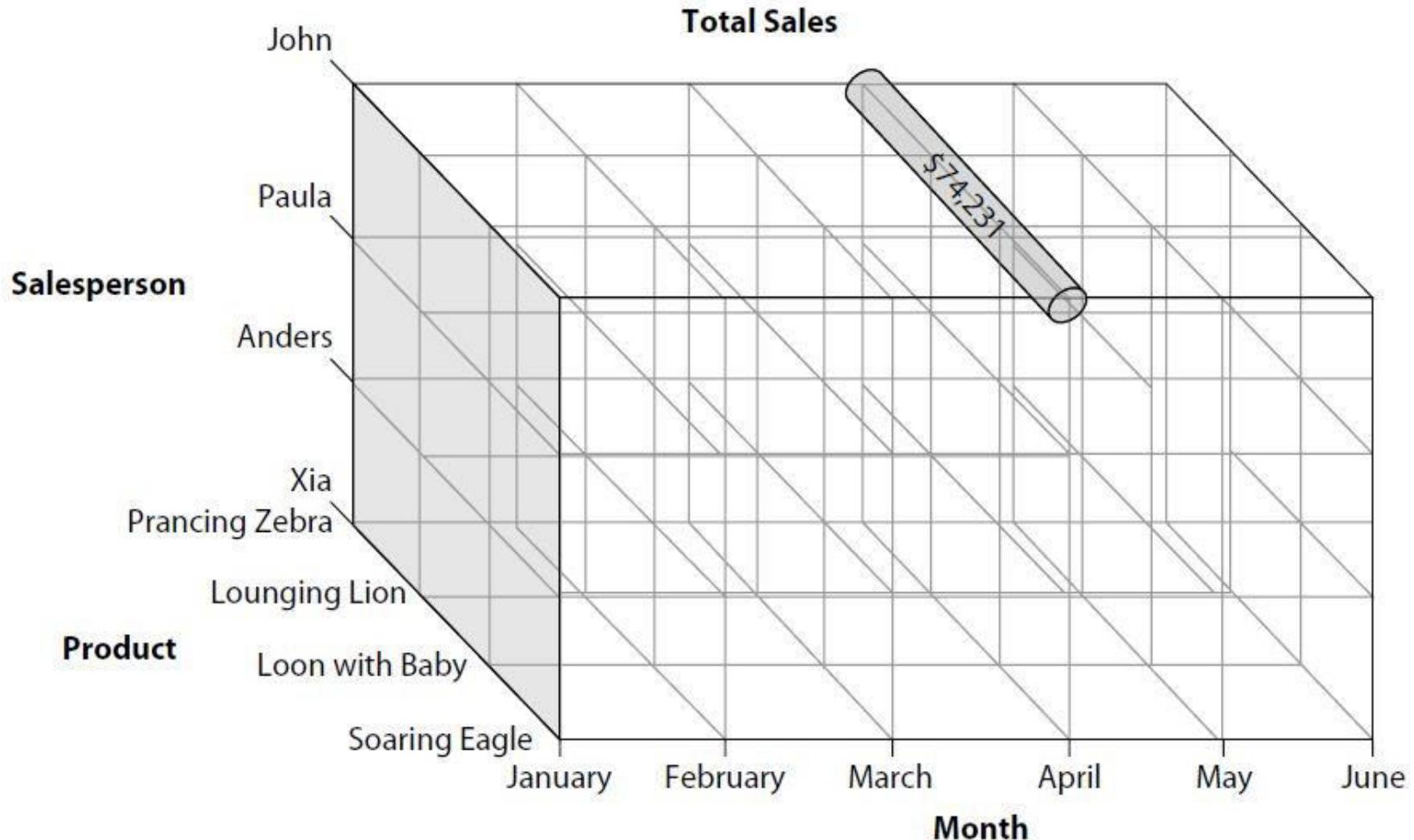
# Cube

- 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*

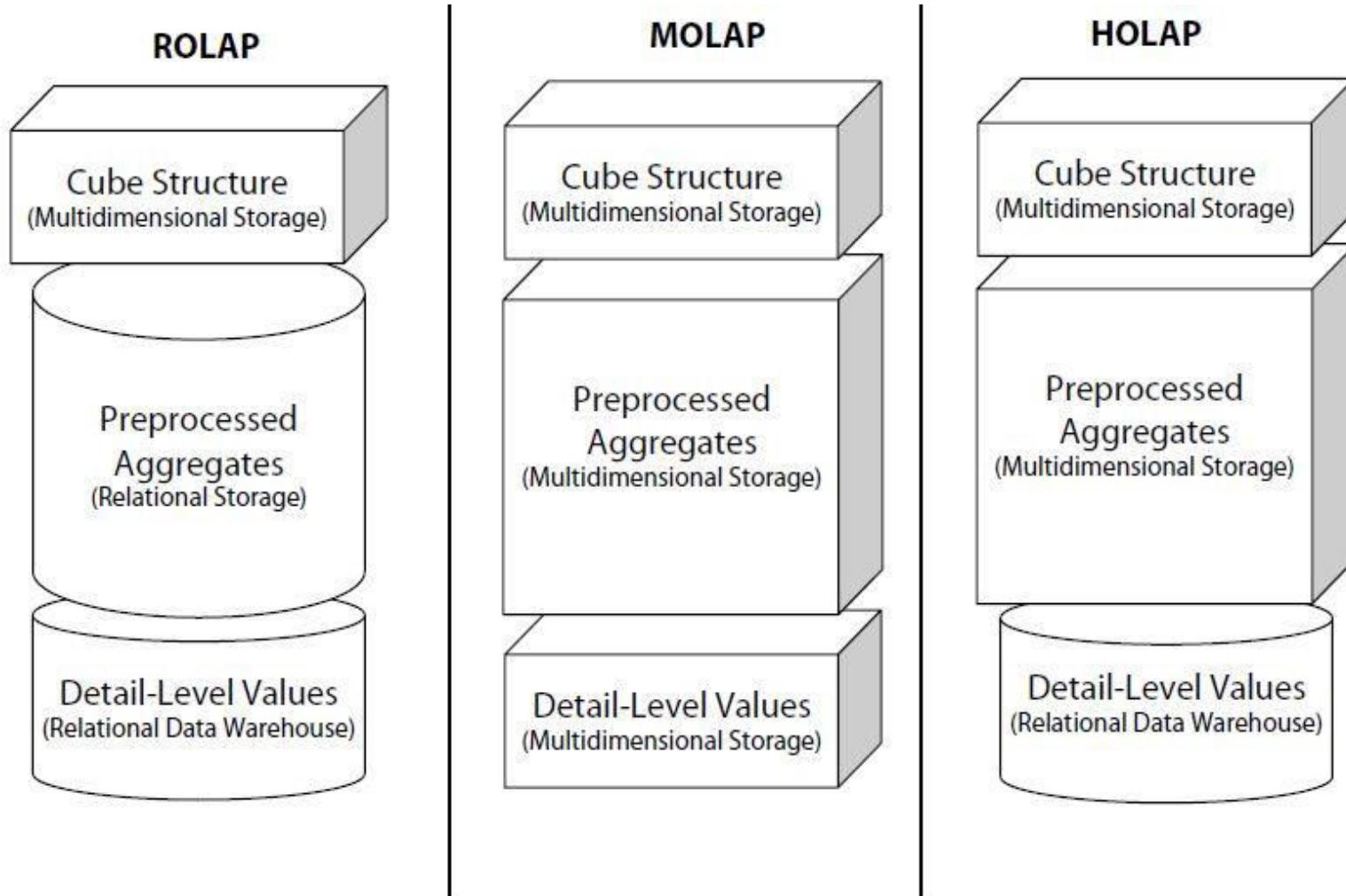
# Multi-Dimensional Database

- 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

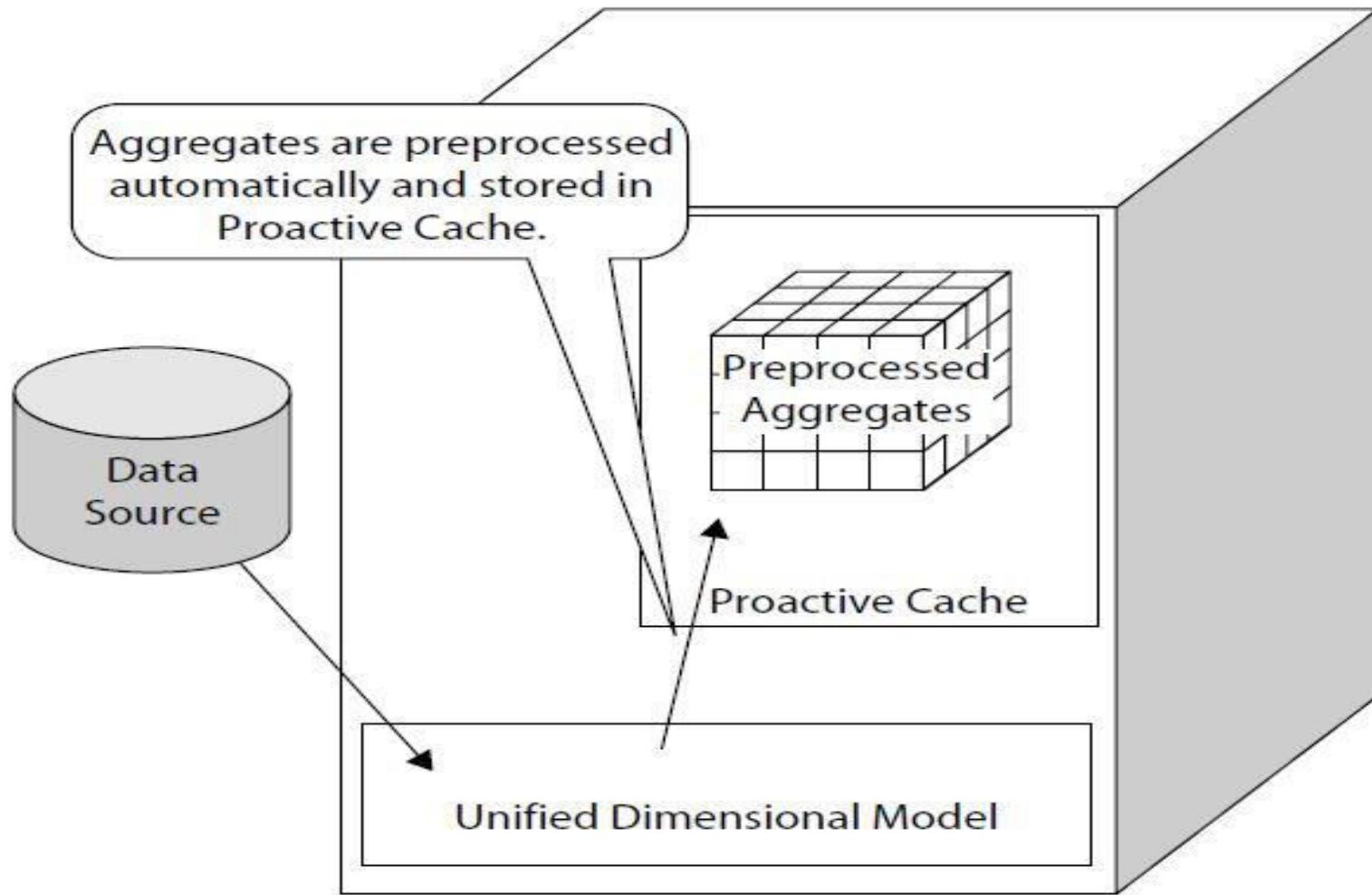
# Data Views

- 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*

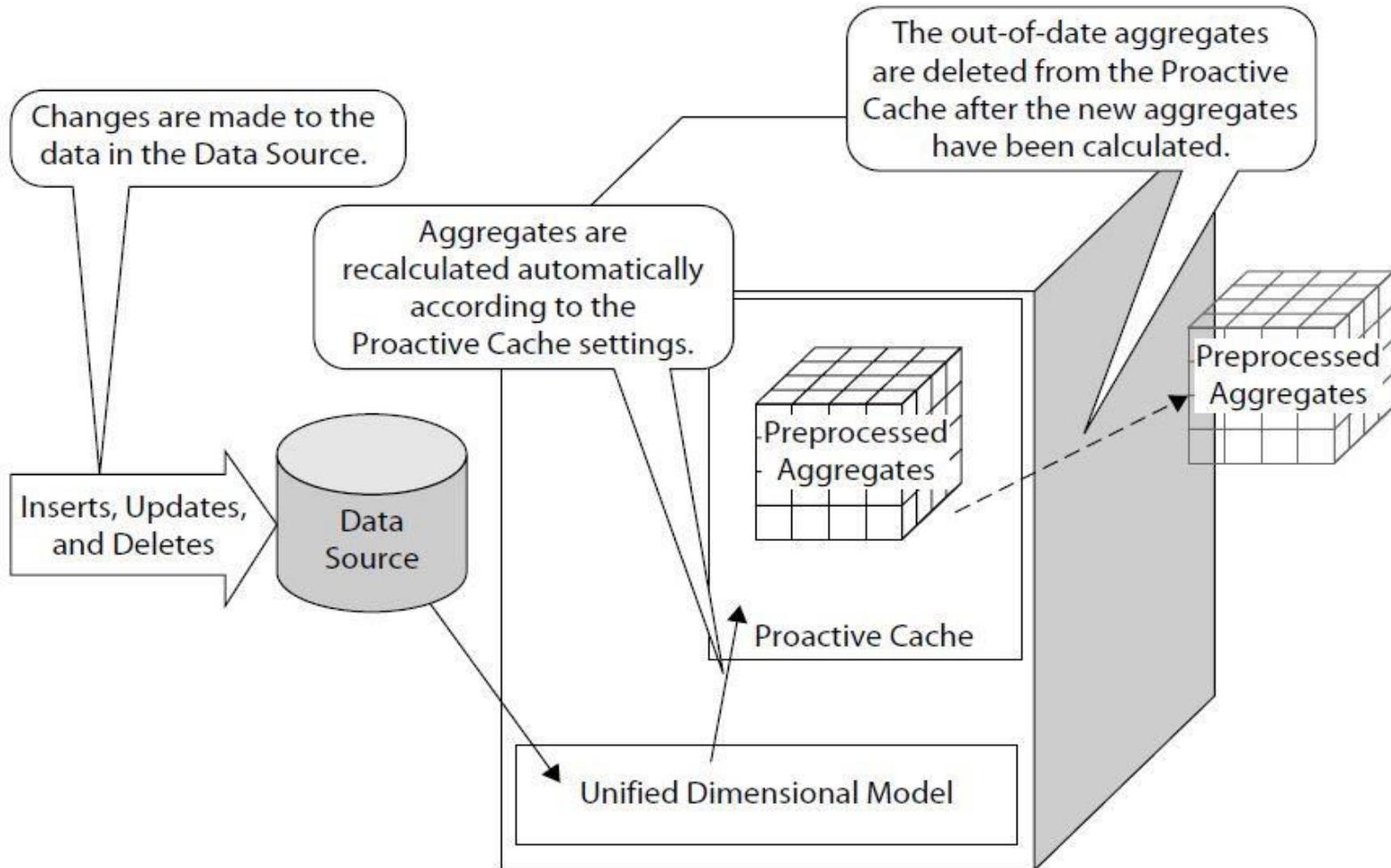
# Proactive Caching

- 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)

# **DataBase Administrator**

## **SQL Server 2008**

# **DBA RESPONSIBILITIES**

Course Notes 12 - 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**

Course Notes 12 - 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

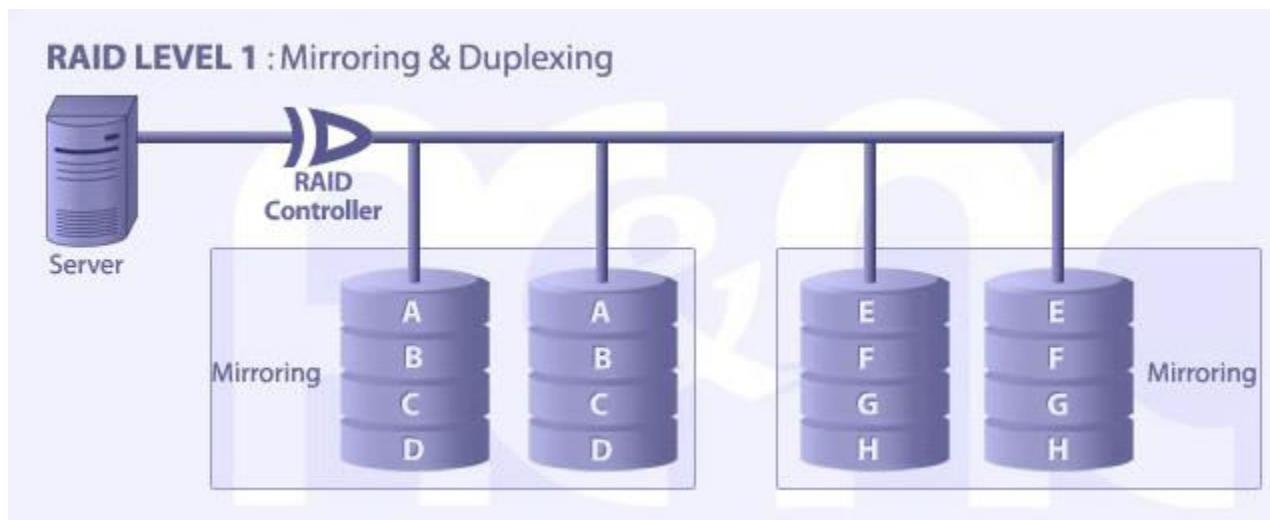
# RAID 0



# 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

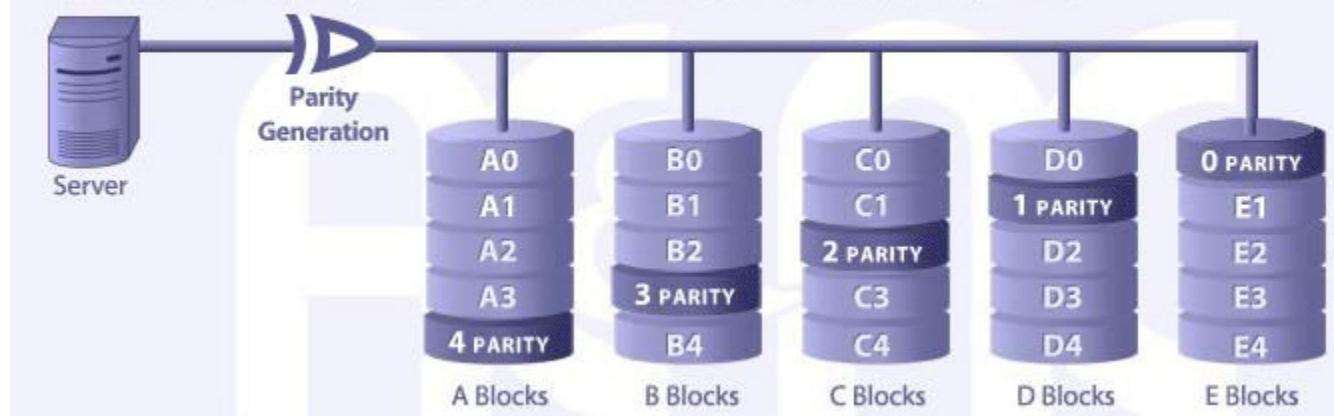
- requires at least three disks
- using parity to provide redundancy
- when disk failure occurs data stored on that disk is dynamically recovered using parity information on remaining disks

# 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**

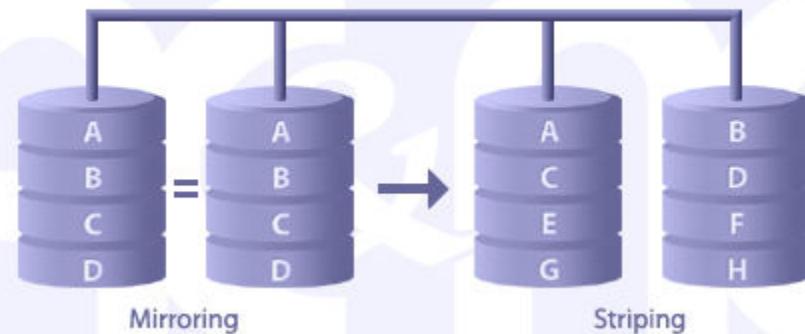


# RAID 10

- 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

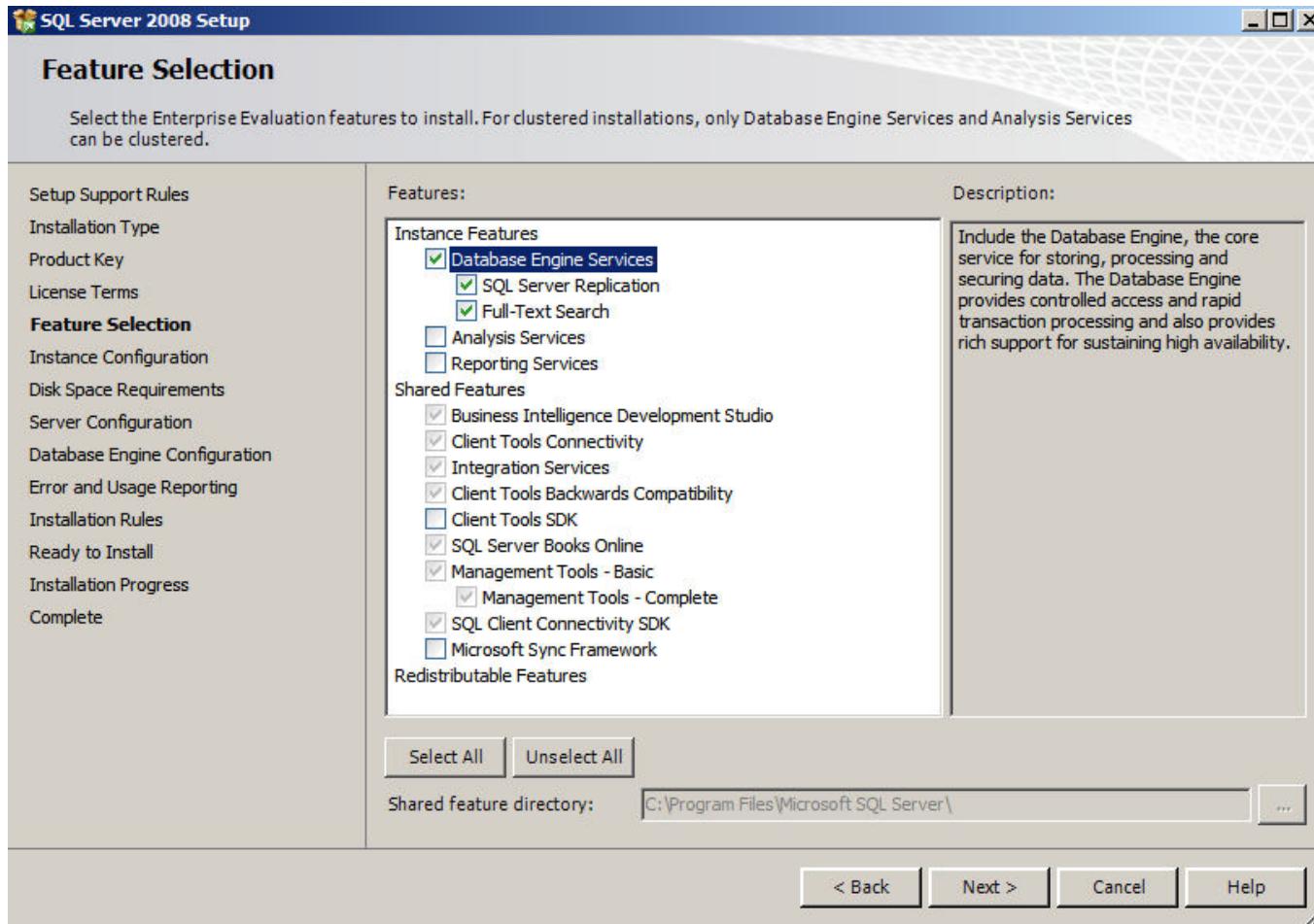
# Start installation; run setup.exe from SQL Server DVD

- 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

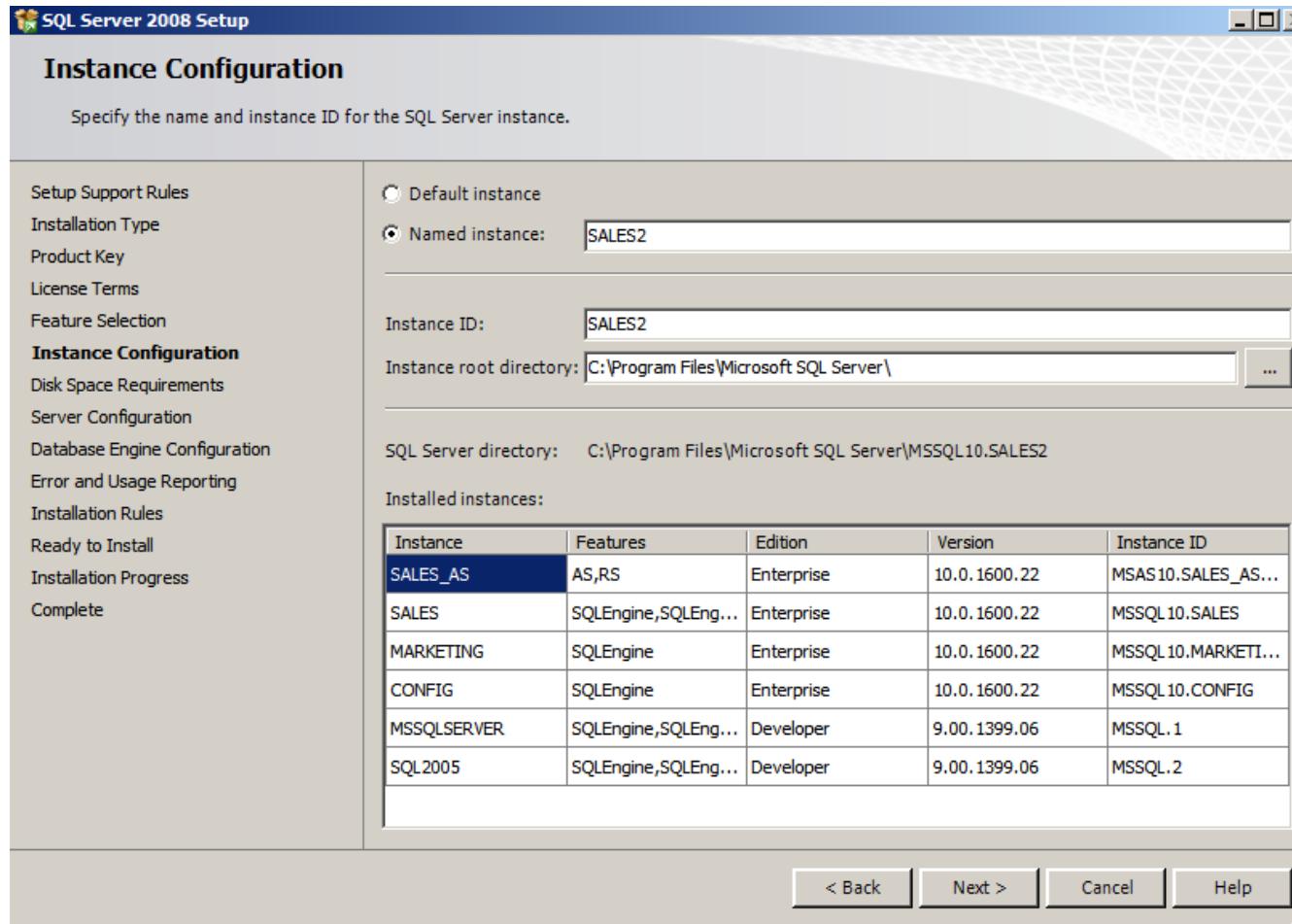
# SQL Server Installation Center



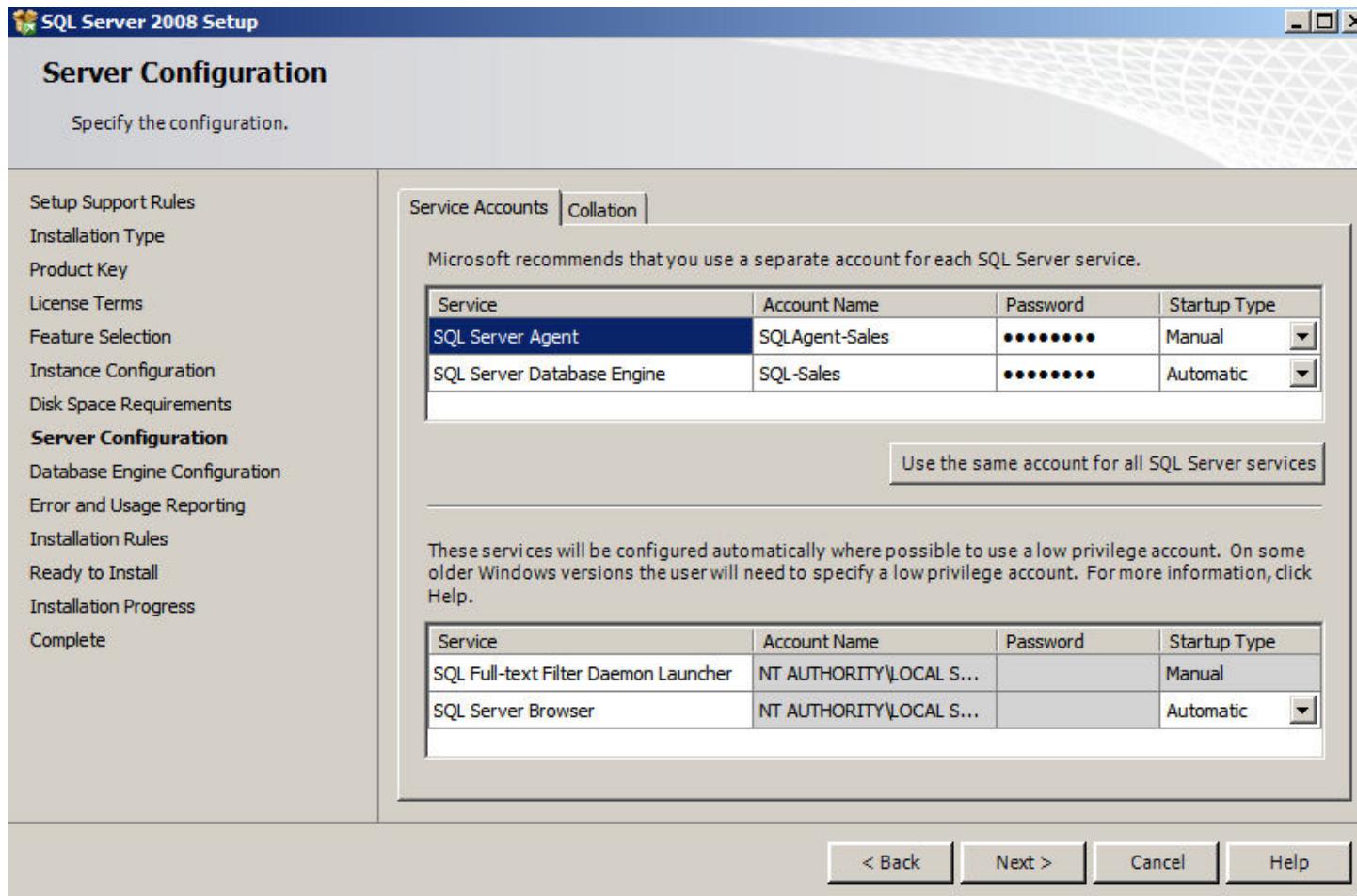
# Features screen



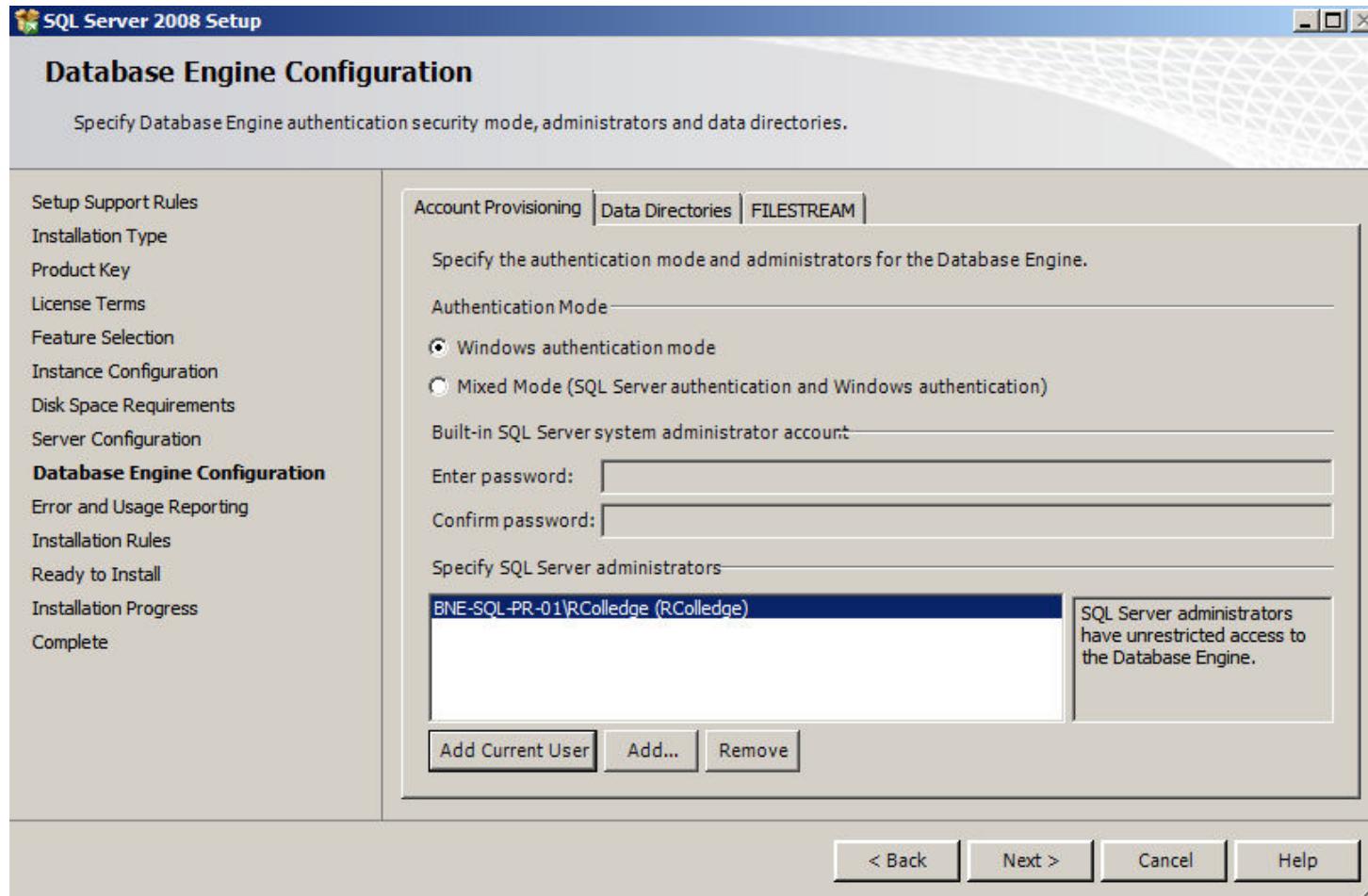
# Select between default and named instance



# Select service account and startup type for each service

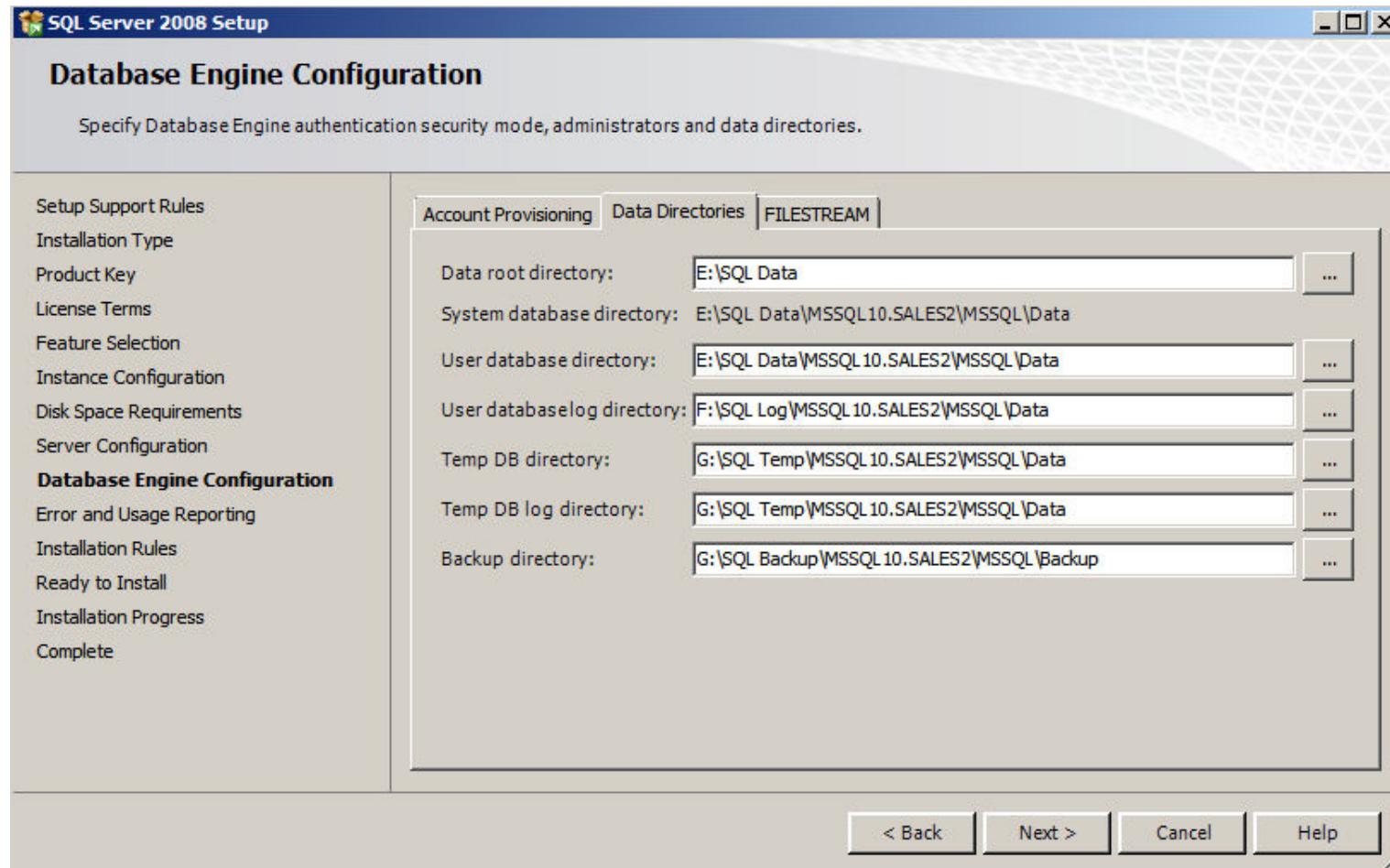


# Select authentication mode



- 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



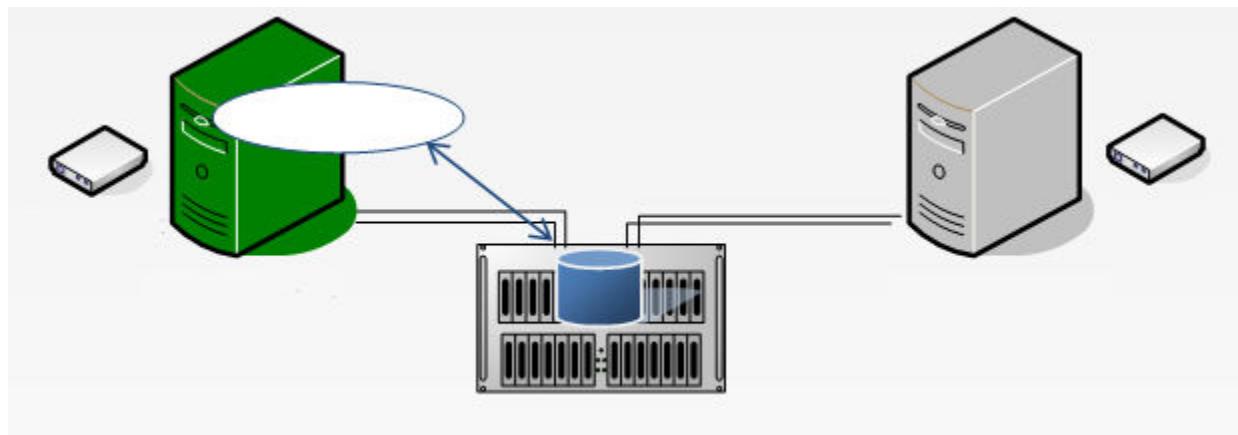
# Upgrading to SQL Server 2008

- 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**

Course Notes 12 - SQL Server  
Administration

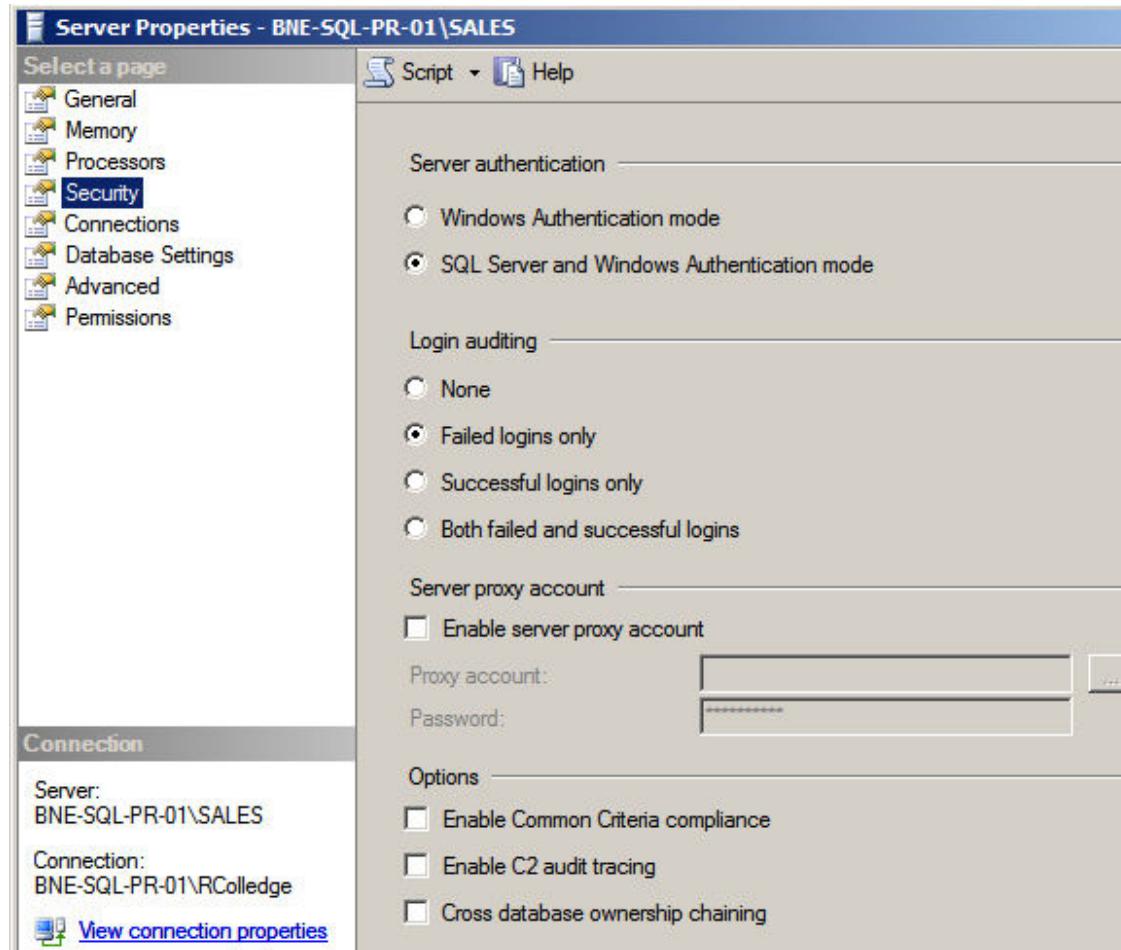
# Configuration

- **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!

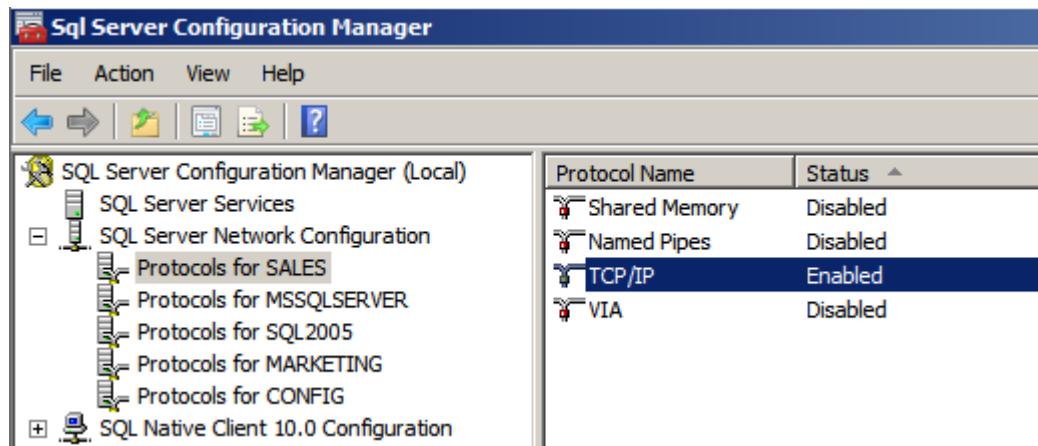
# Networking Protocols

- 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 listenent 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

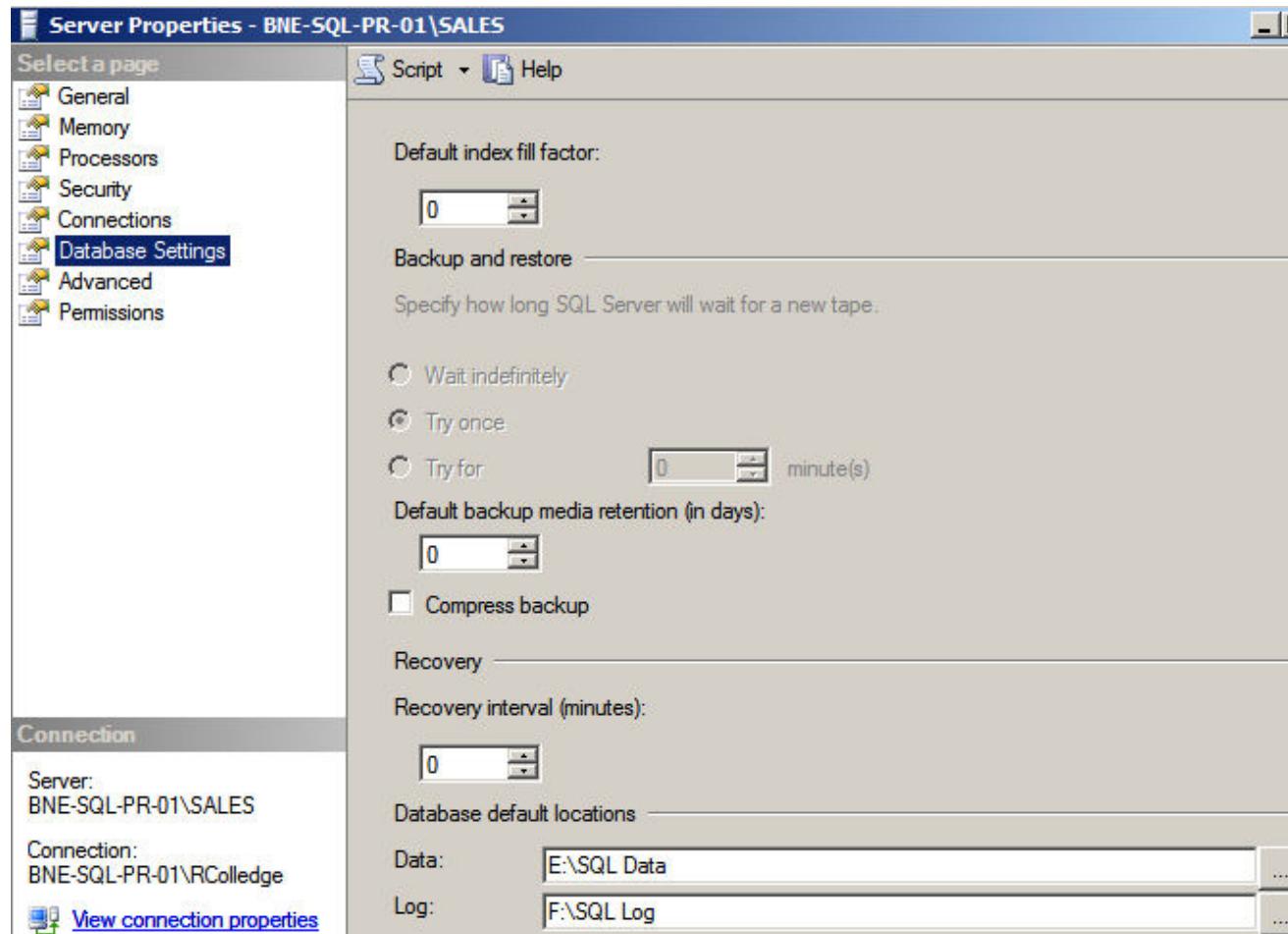
# Server configuration - Fill factor

- 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 Script ▾ Help

General  
 Memory  
 Processors  
 Security  
 Connections  
 Database Settings  
 Advanced  
 Permissions

**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

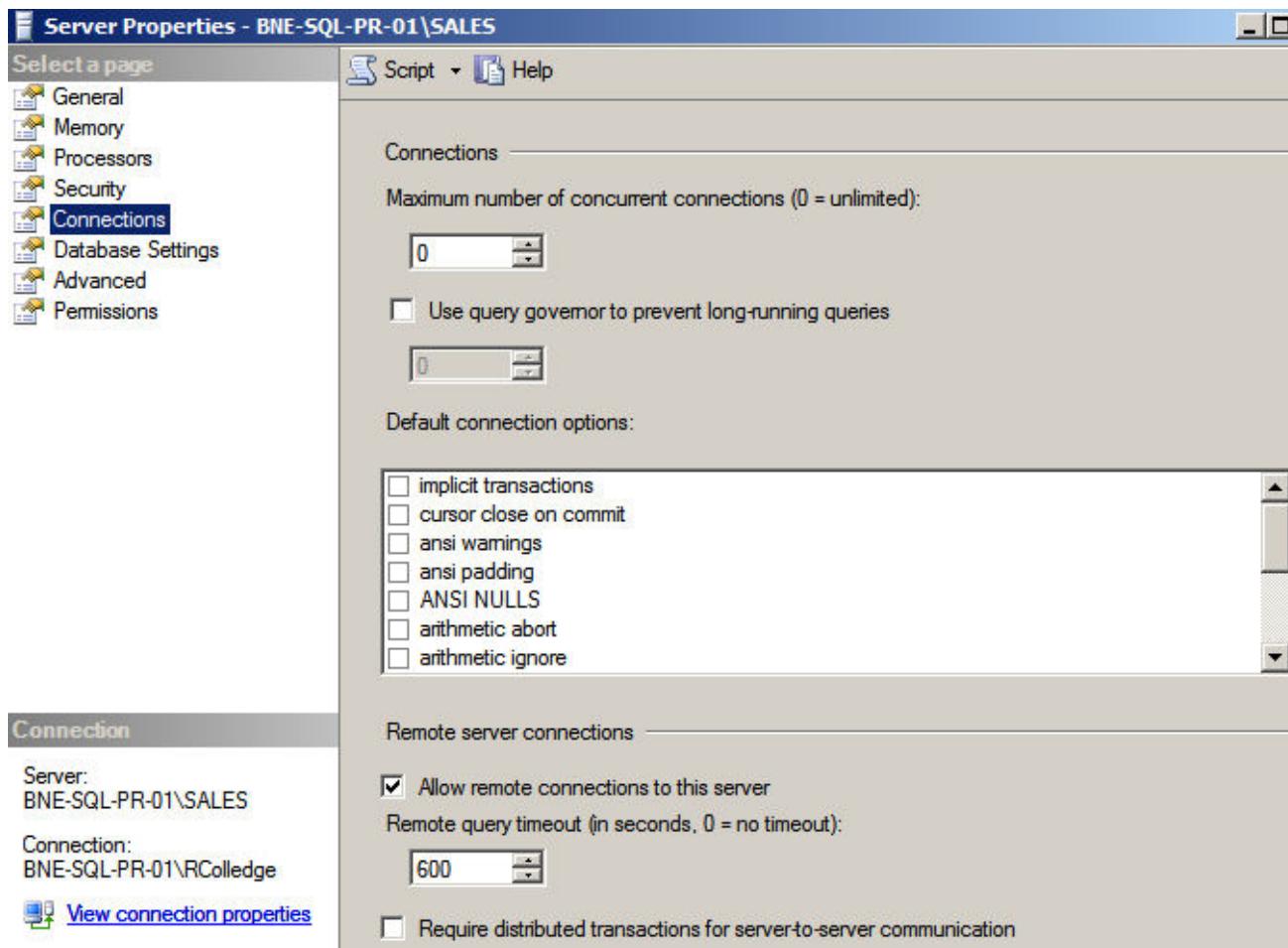
[View connection properties](#)

**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

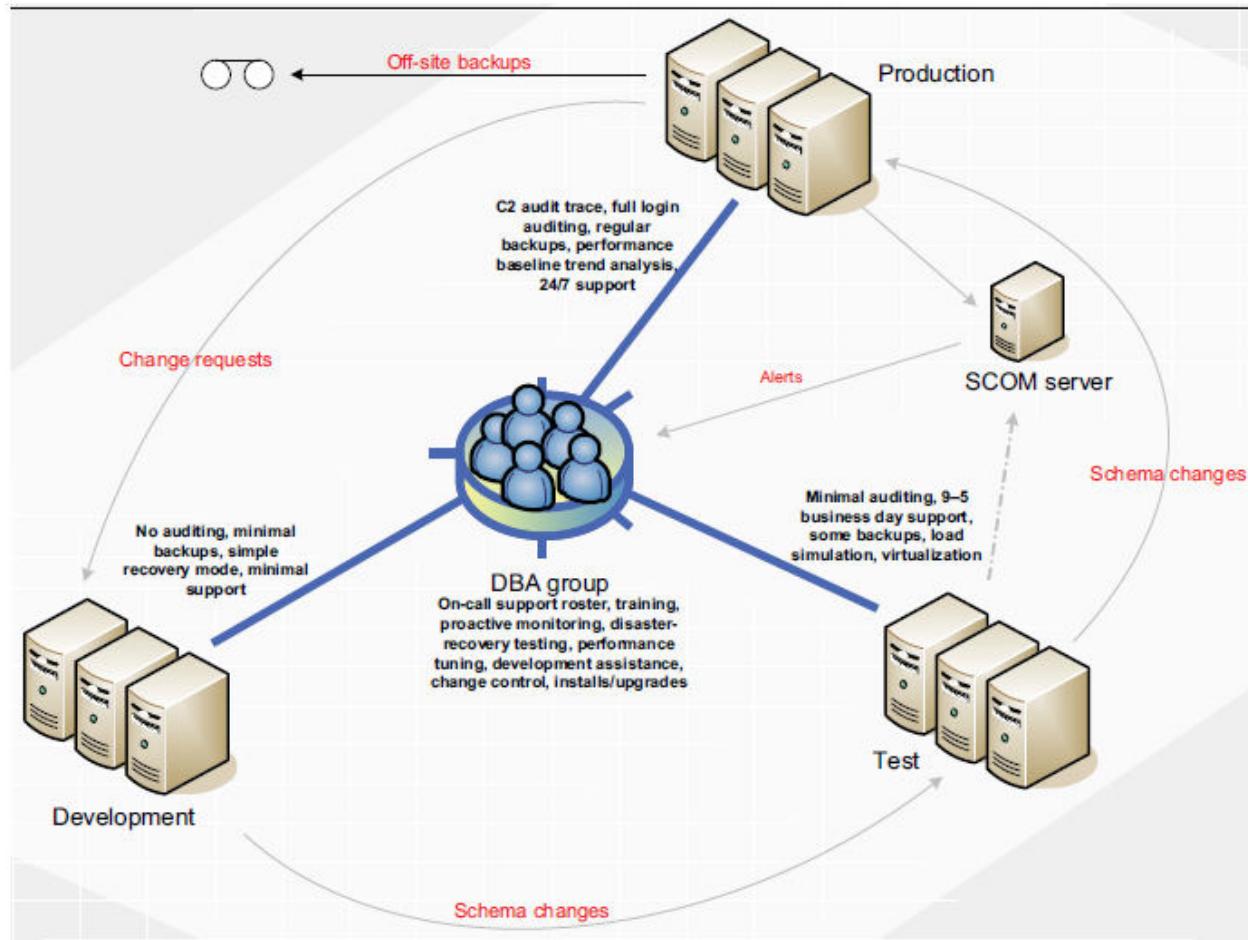
- accessed via Control Panel, advanced options of System Properties let you choose between adjusting processor resources to favor foreground applications or background services
- as background service, SQL Server obviously benefits from this option being set to Background Services

# 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

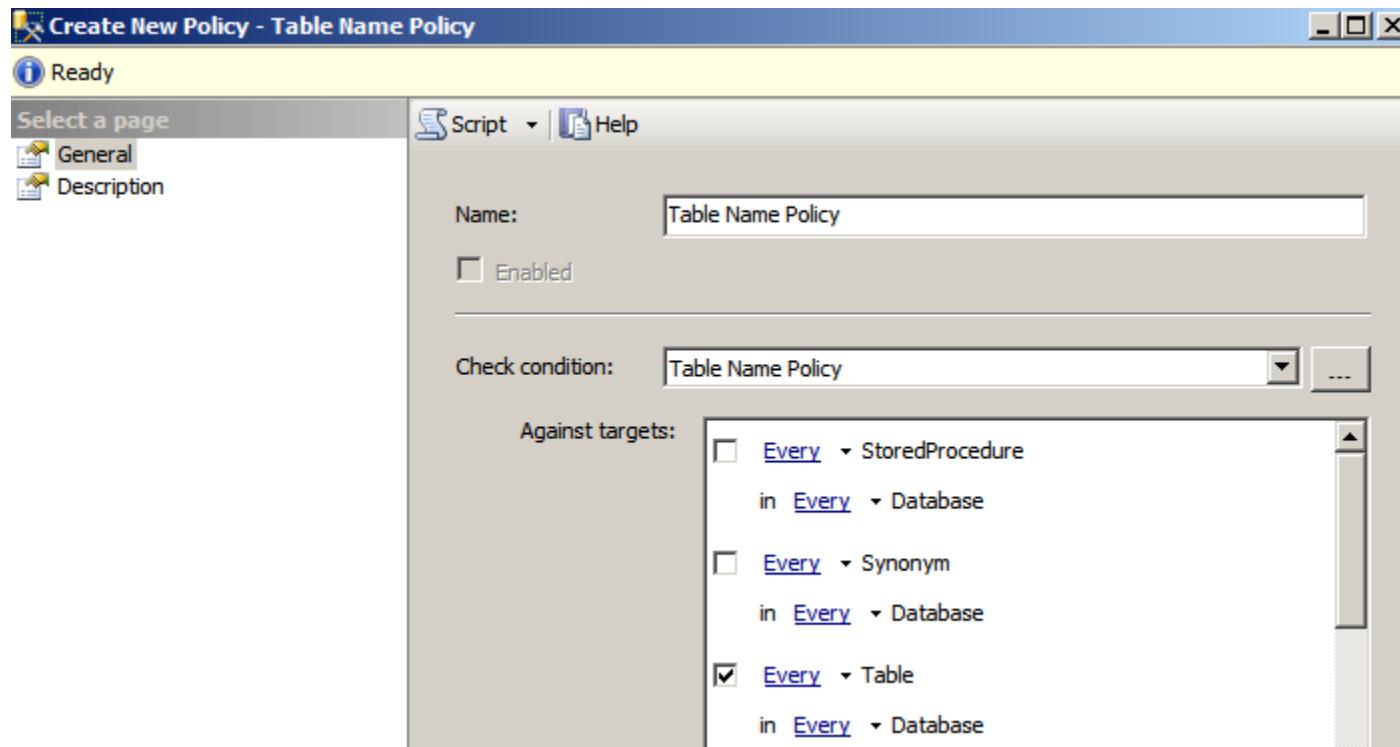
# Enterprise DBA challenges

- 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: DNE-SQL-PR-01\SALES [DNE-SQL-PR-01\RColledge]

[View connection properties](#)

Progress: Ready

XPCmdShellEnabled  
xp\_cmdshell creates a Windows process that has same security rights as the SQL Server service.

# 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 |
|-------|------------------------------|----------|-------|
| AND   | @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 |

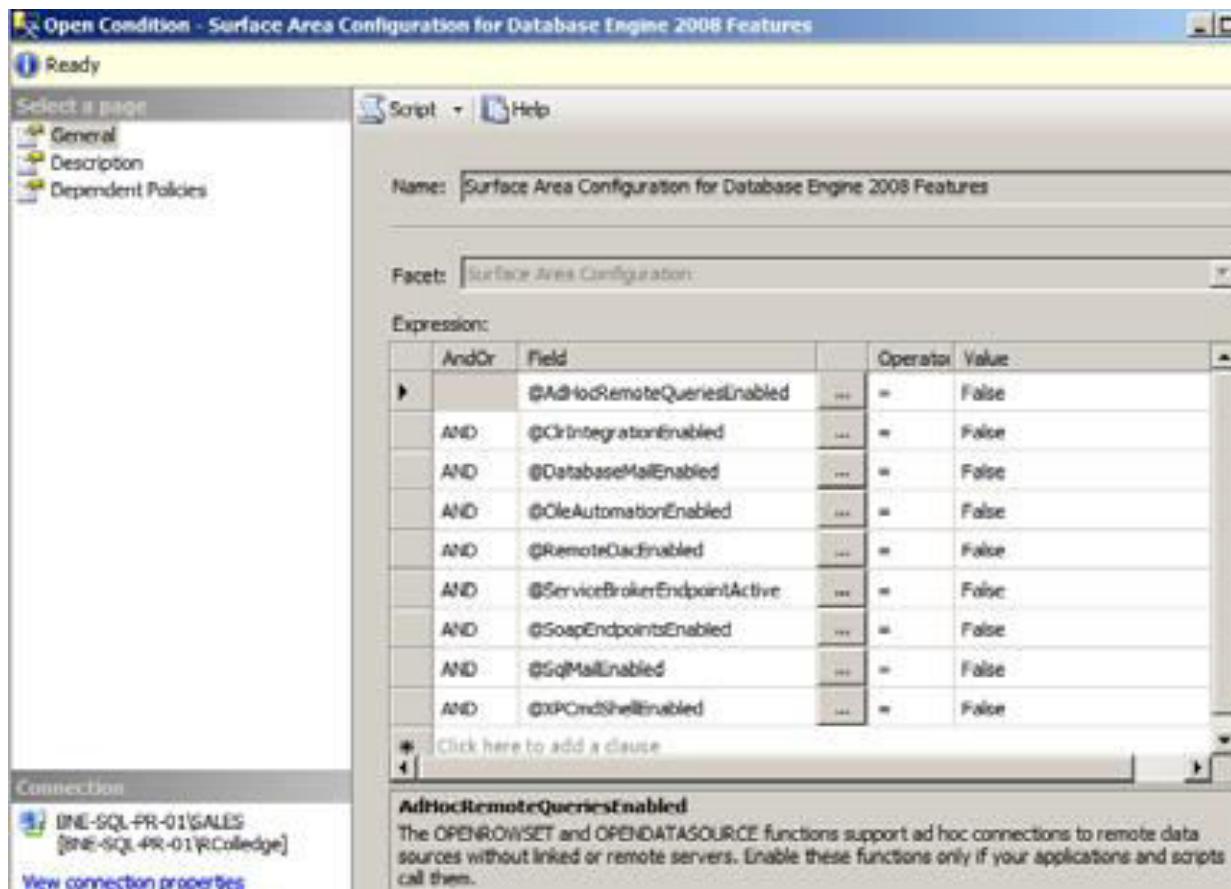
Click here to add a clause

Connection

LNE-SQL-PR-01\GALES  
[LNE-SQL-PR-01]\Colledge]

[View connection properties](#)

**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.



## Course Notes 12 - SQL Server Administration

# Configuration

- Security
- Configuring SQL Server
- Policy-based management
- **Data management**

# Database file

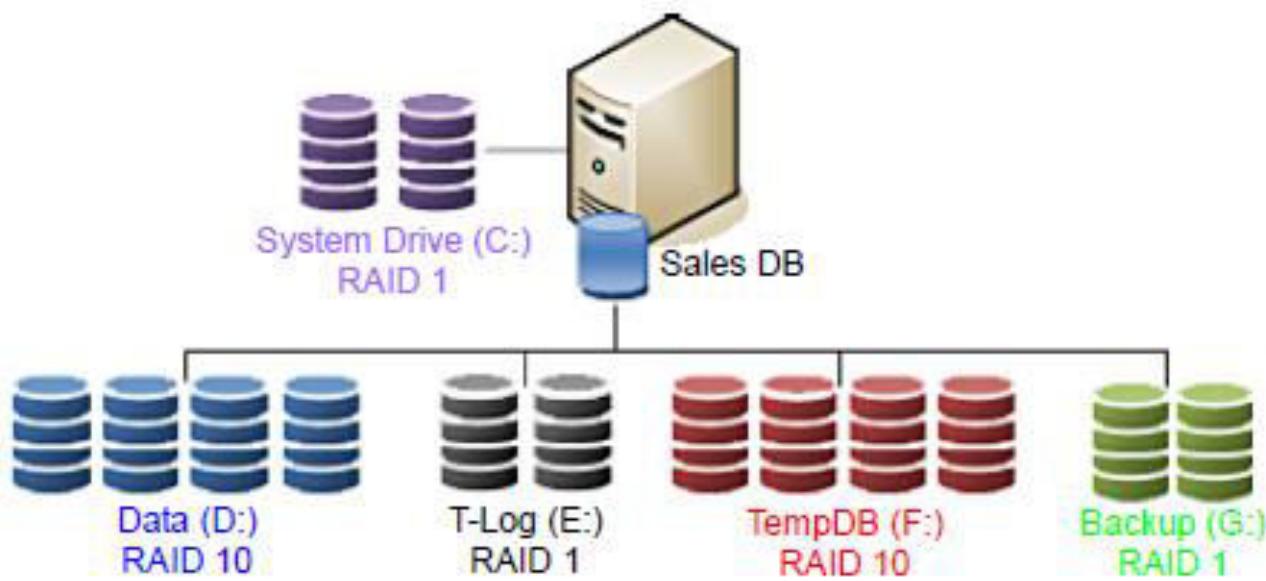
- Primary data file
  - by default only data file, contains system tables and information on all files within database
  - by default file has an .mdf extension
- Secondary data file
  - usually have an .ndf extension
  - optional files that can be added to database for performance and/or administrative 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

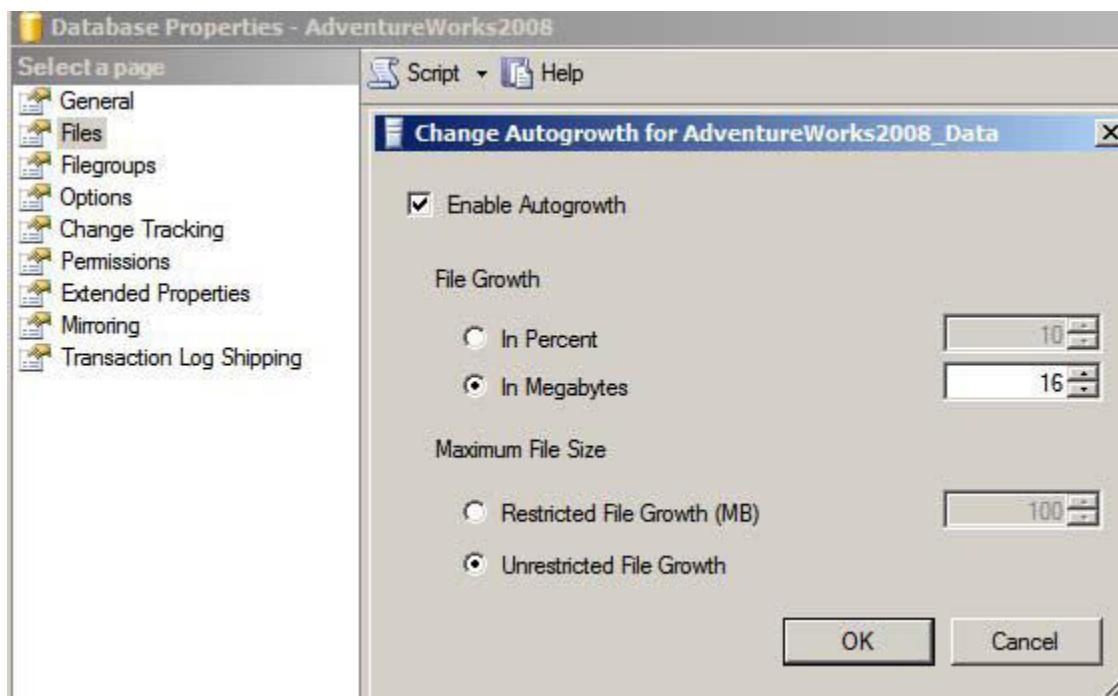
# Performance

- 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

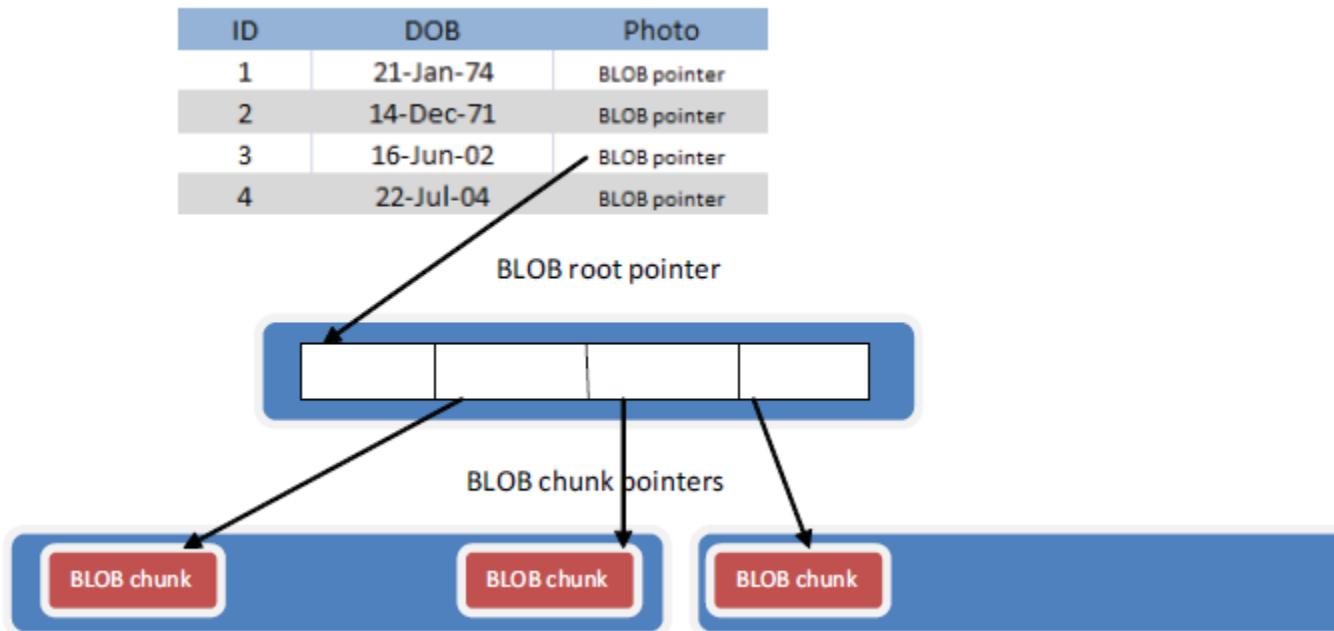
# BLOB storage with FileStream

- 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

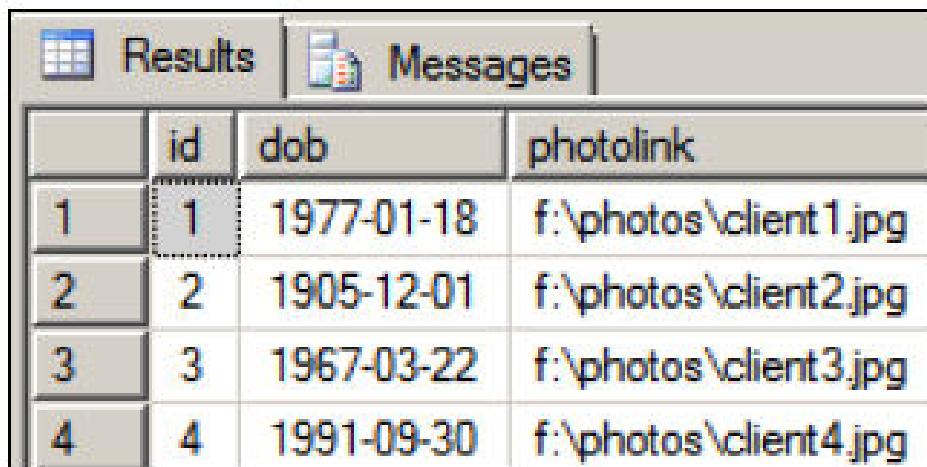
# BLOBS in database



# 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



|   | <a href="#">id</a> | <a href="#">dob</a> | <a href="#">photolink</a> |
|---|--------------------|---------------------|---------------------------|
| 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

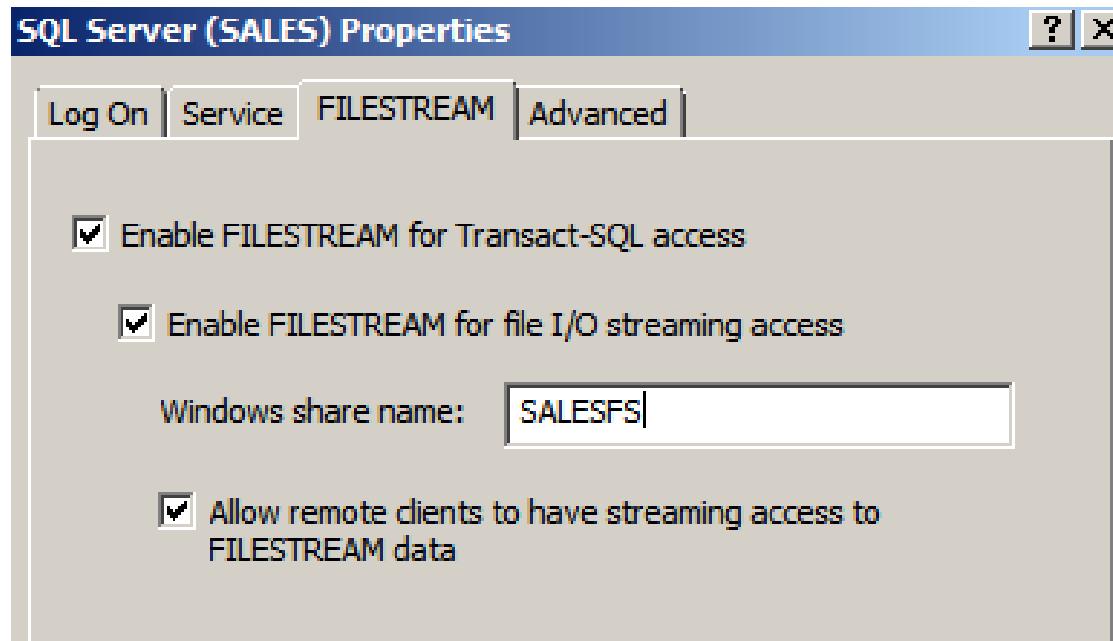
- BLOBs can be stored in file system
  - size of each BLOB is limited only by NTFS volume size limitations
    - overcomes 2GB limit
- Full transactional consistency exists between BLOB and database
- BLOBs are included in backup and restore operations

# 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**

Course Notes 12 - 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***

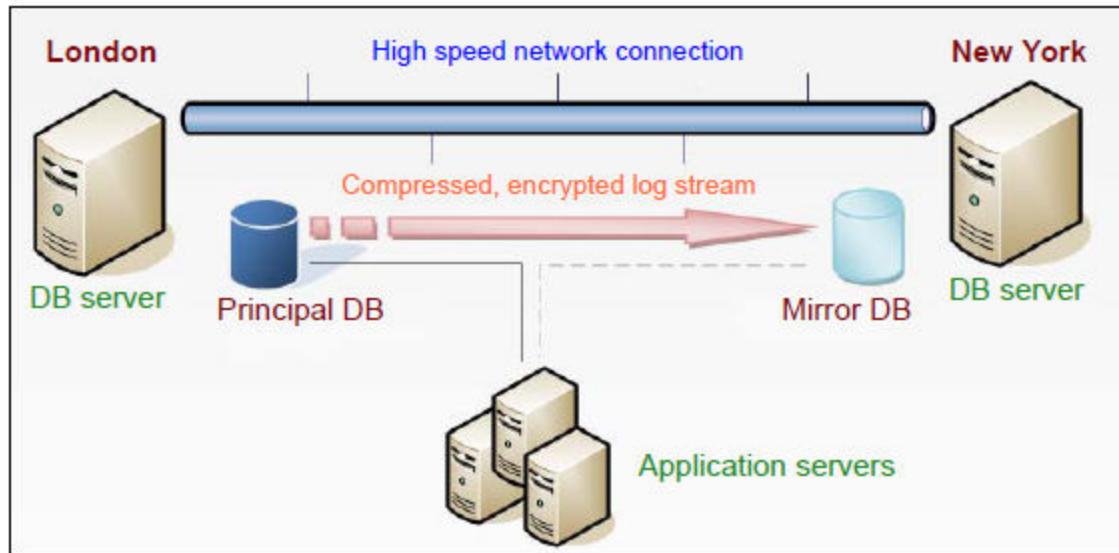
**DBA OPERATIONS**

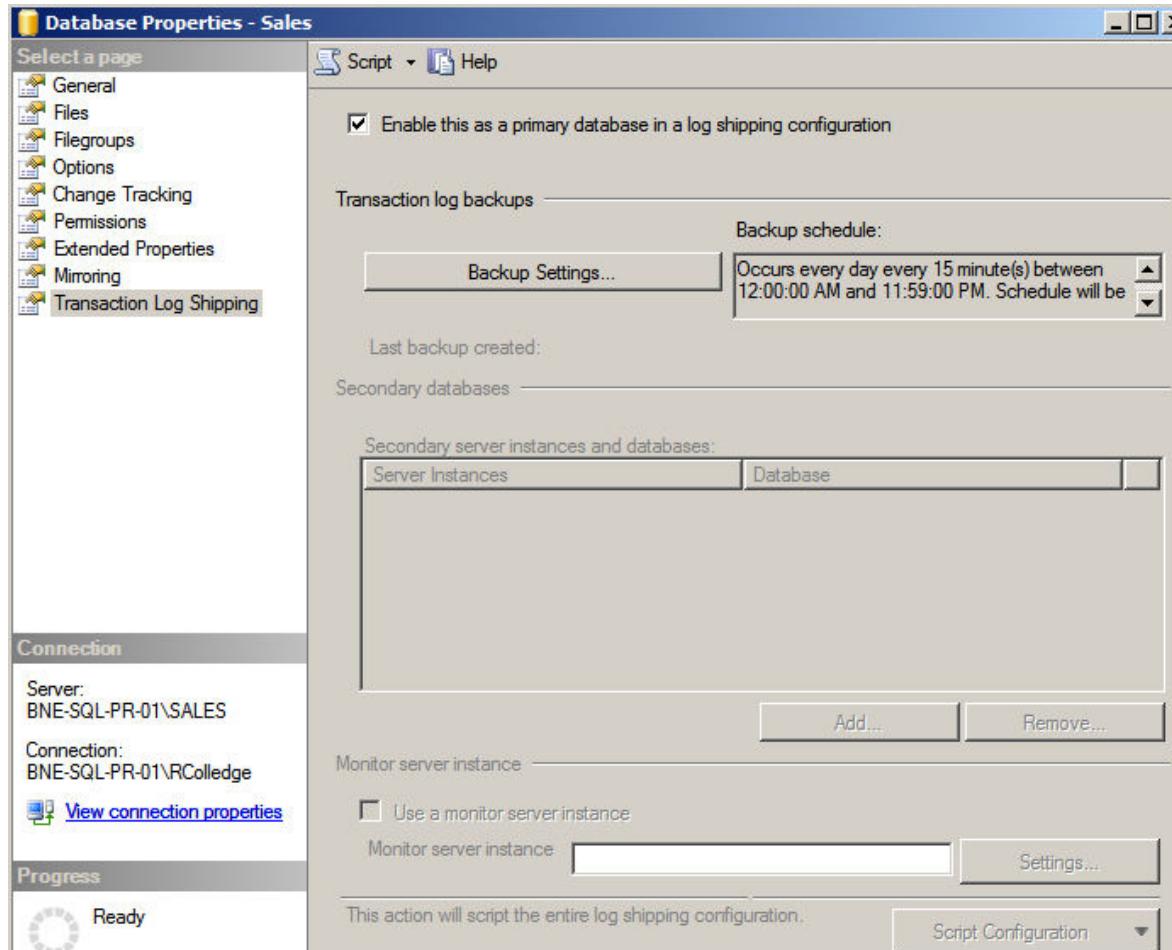
# High-availability

- 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

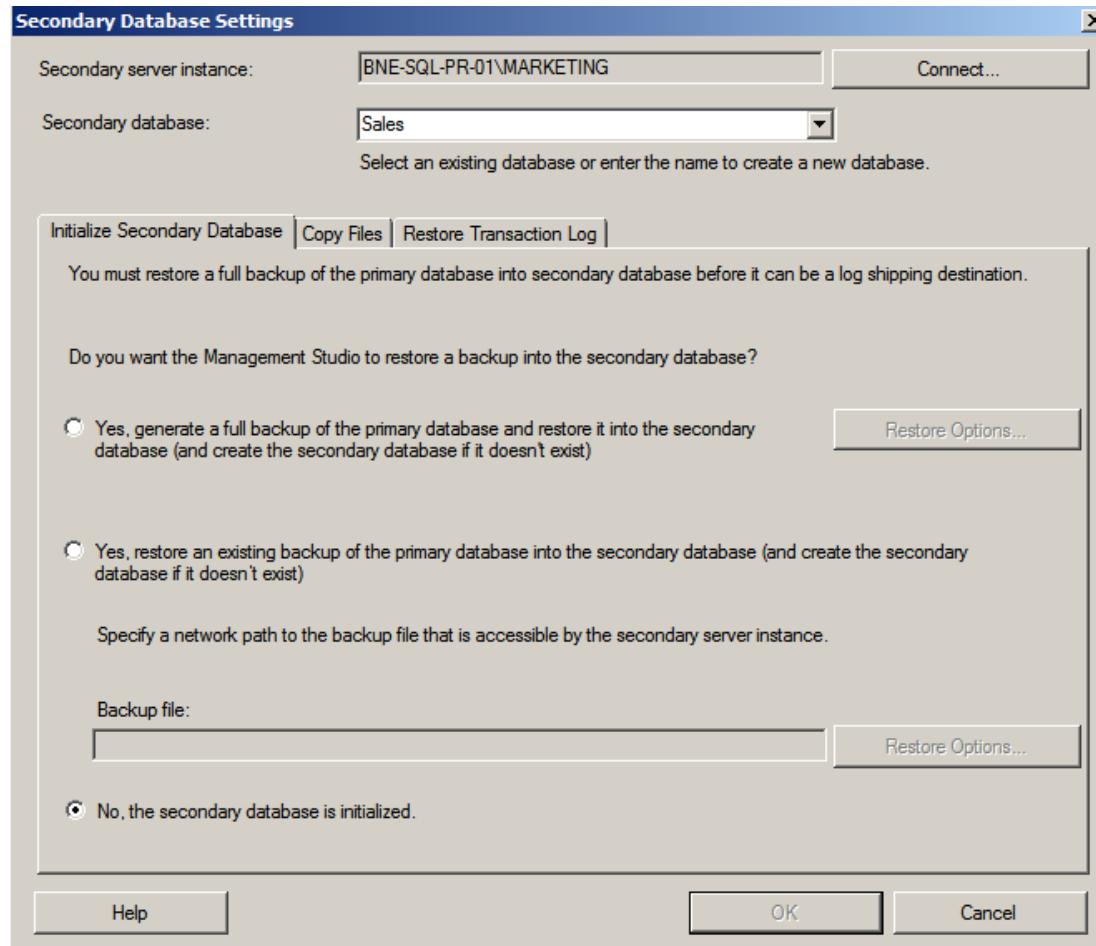
# Database mirroring

- 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





Course Notes 12 - SQL Server  
Administration

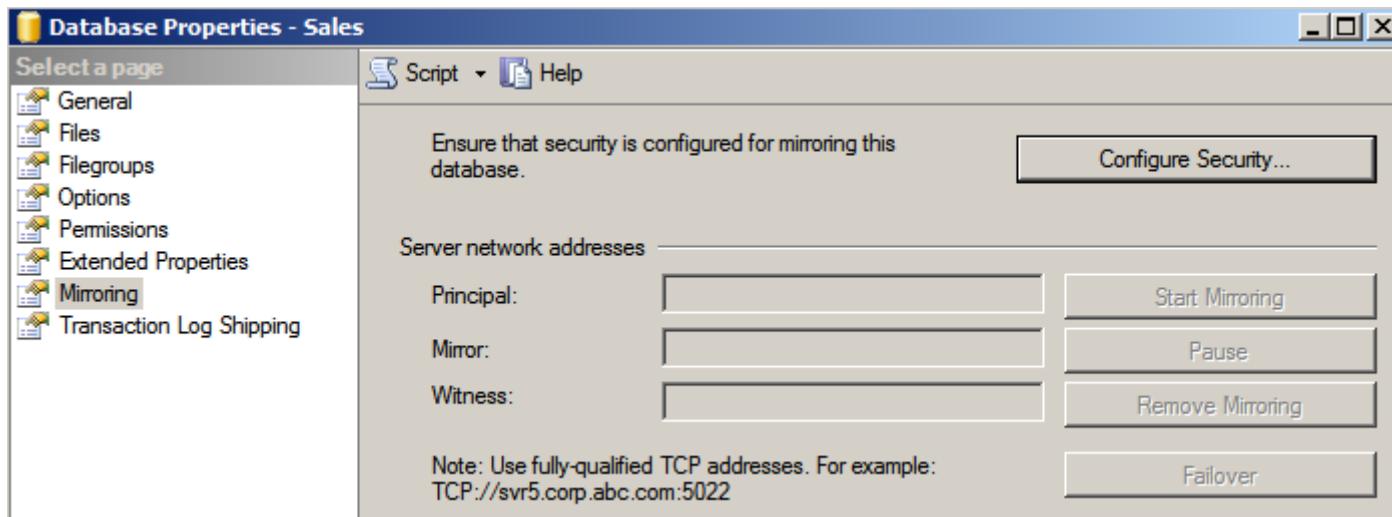


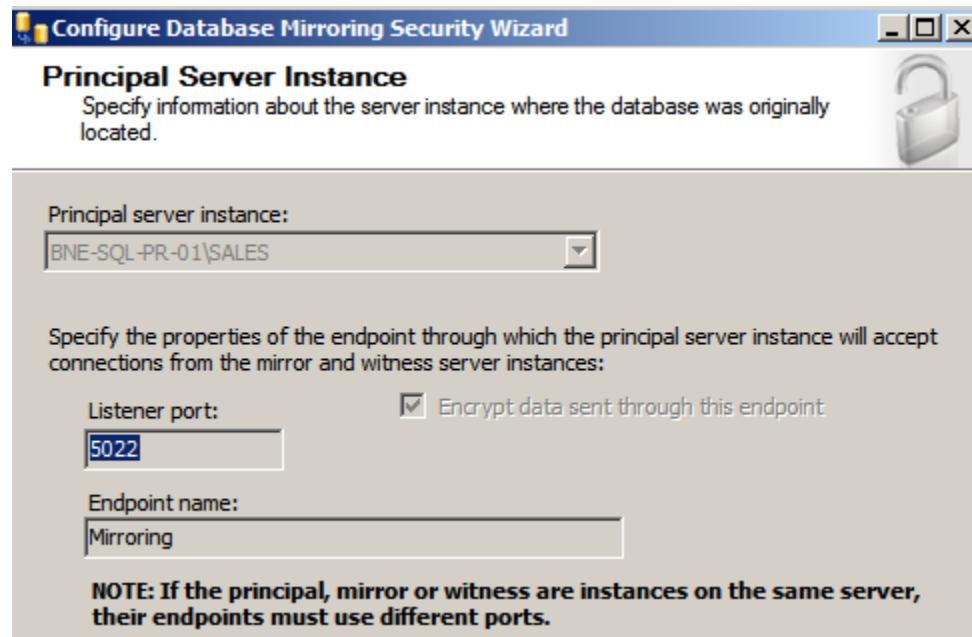
# Failover and role reversal

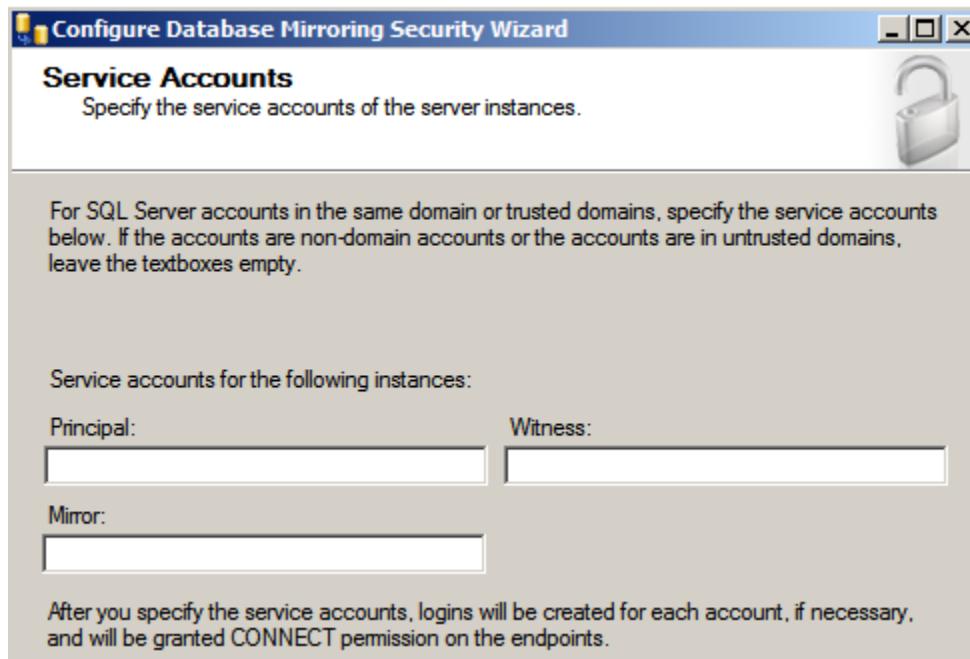
- if primary instance in log shipping solution fails (or a manual failover is required for planned maintenance), failover process is largely a manual effort, requiring you to back up, copy, and restore outstanding logs to secondary instance

# 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 |

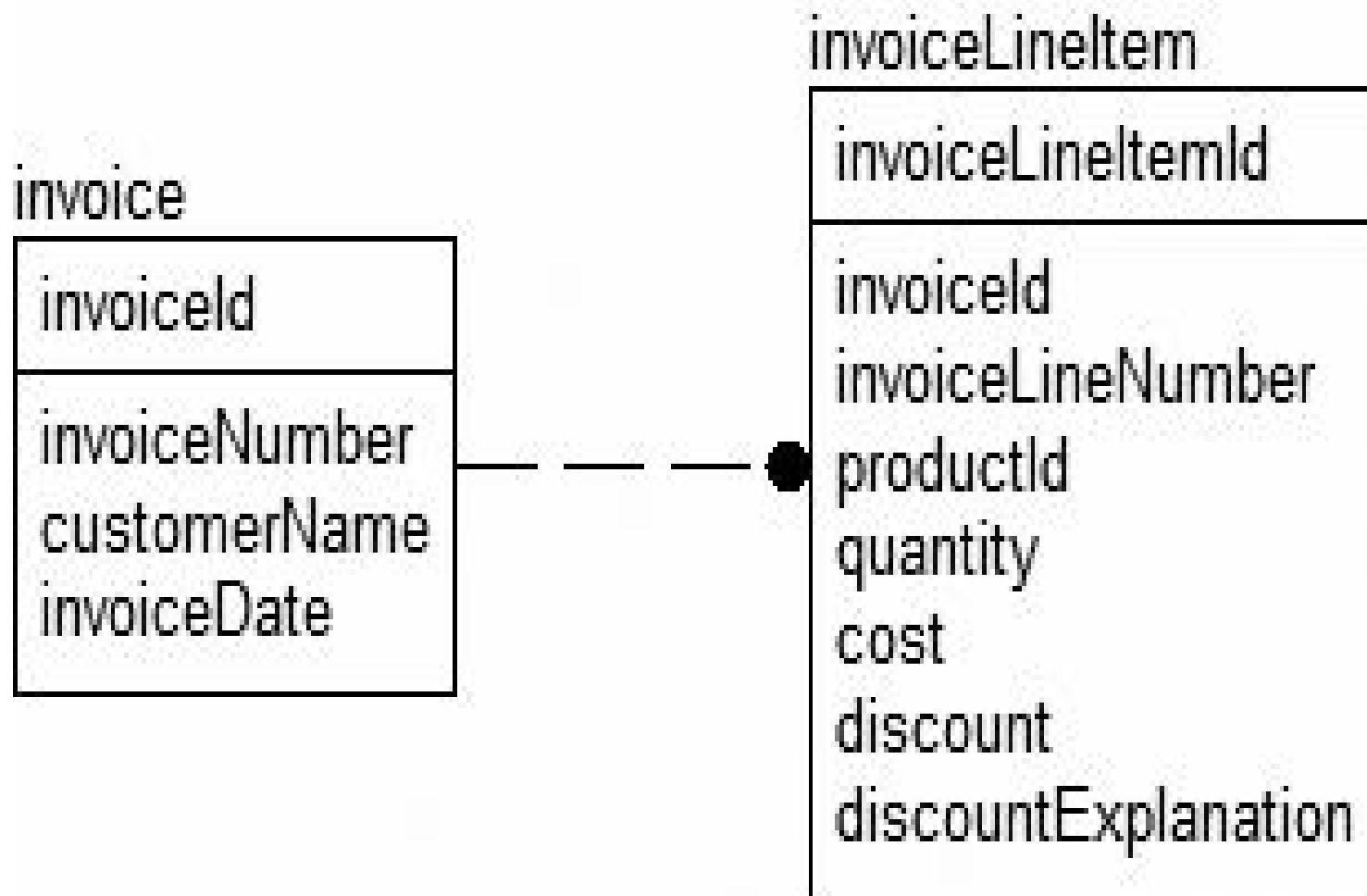
# Domain tables

- 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.invoicelId column

# Ownership Relationships



# Many-to-Many Relationship

- needs to be an index on the two migrated keys from the two parent tables

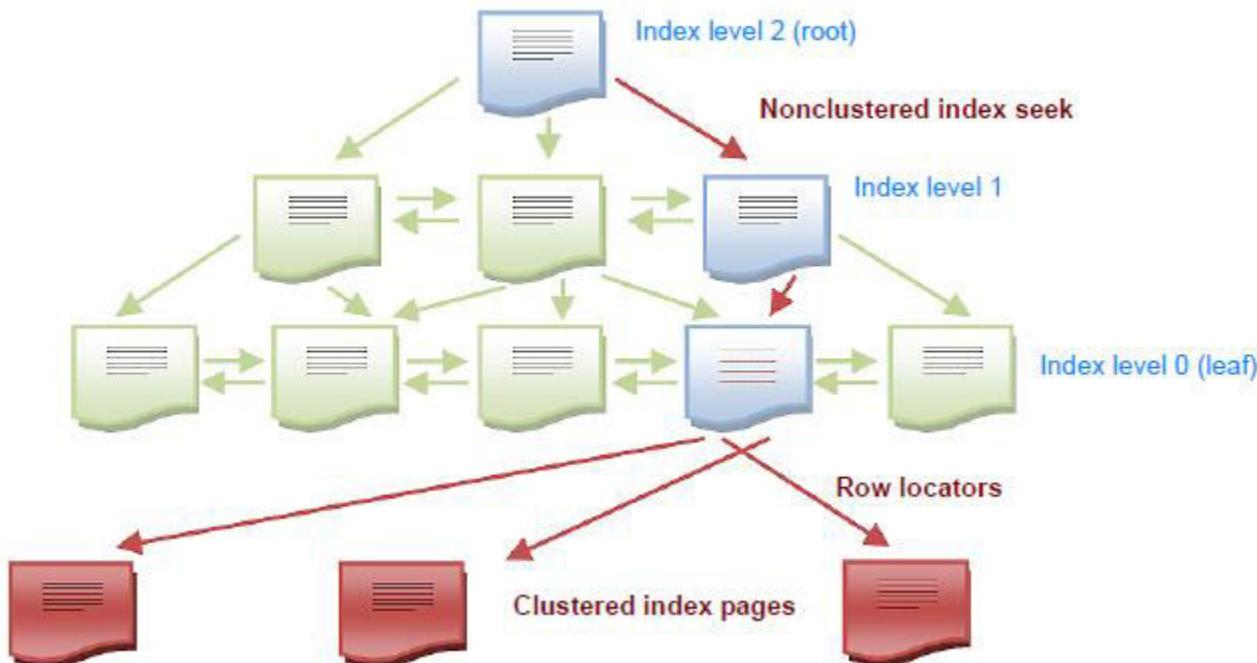
# Best Practices

- 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.

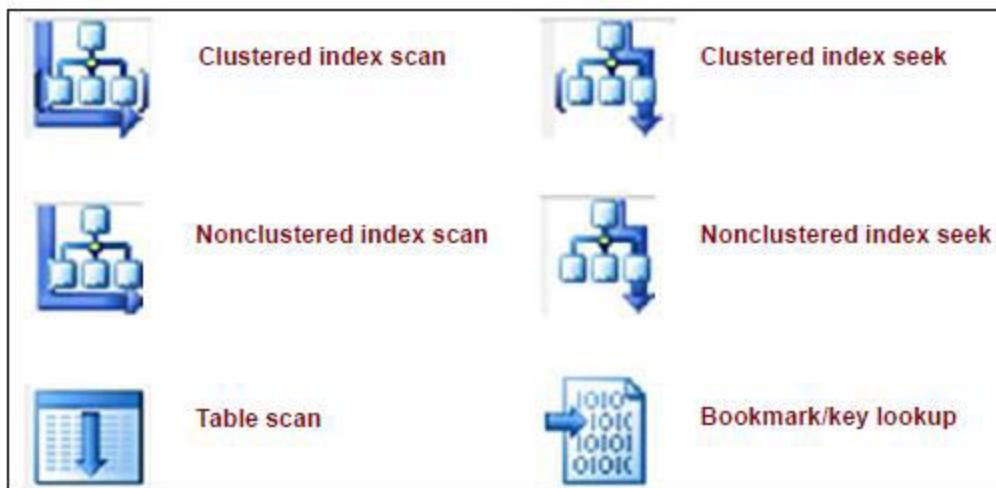
# clustered index

- 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

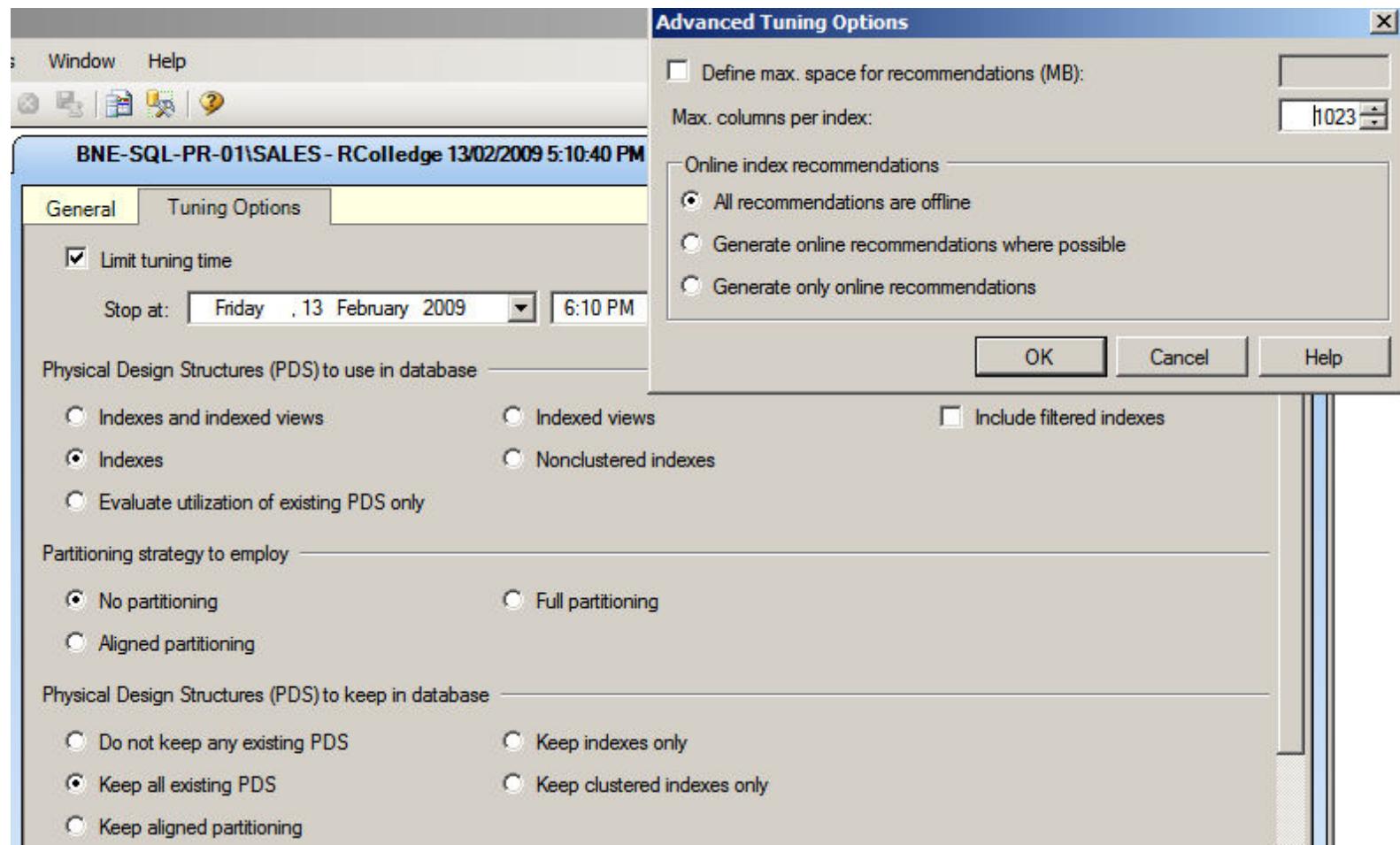
- returns information on how indexes are being used
- for each index, counts are kept on number of times index has been scanned, updated, used for lookup or seek purposes

# `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



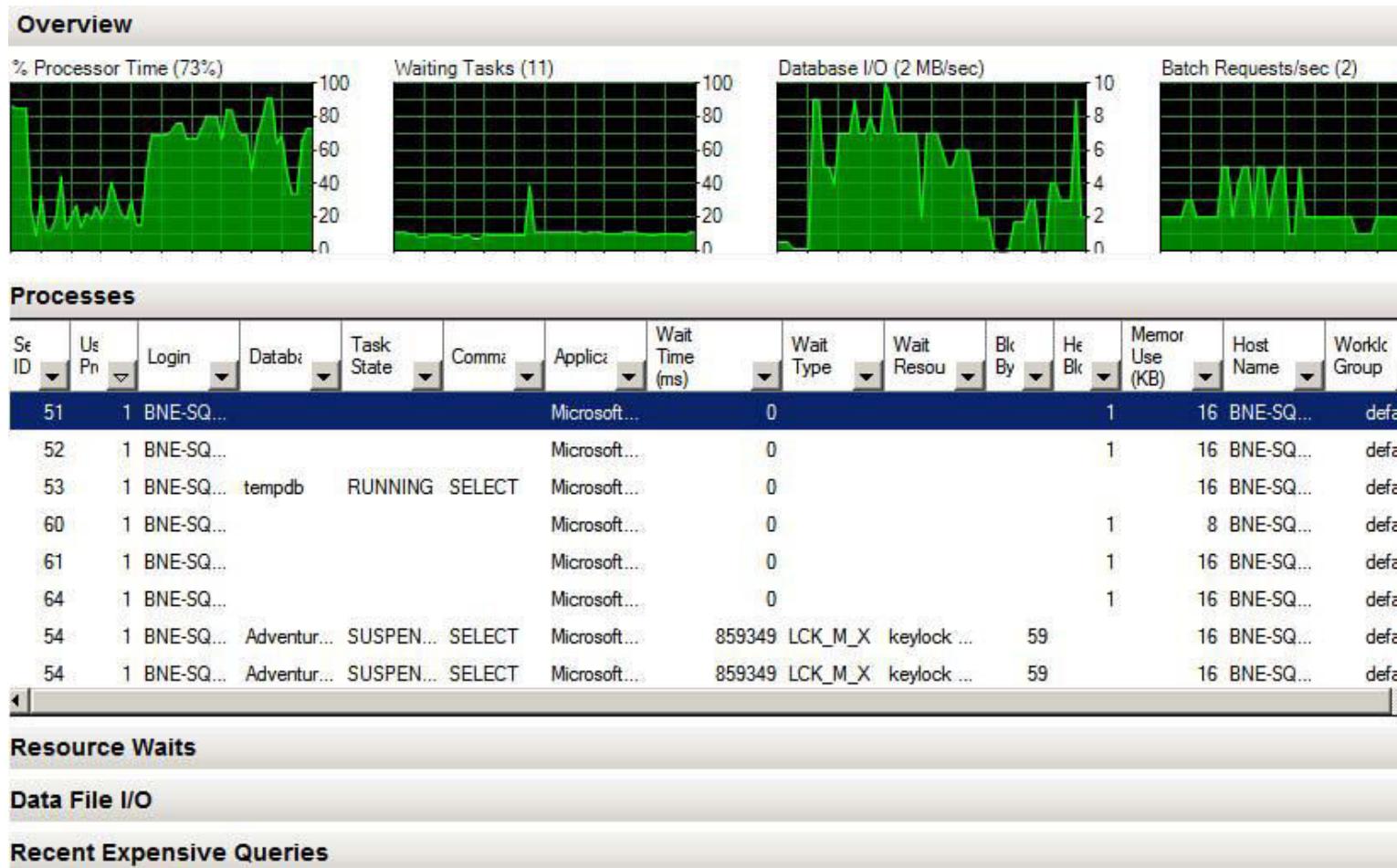
# `sp_updatestats`

- 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**

Course Notes 12 - 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) |
| <a href="#">AdventureWorks2008</a> | 180.06          |       | 500.06            | 320                     | 2.00            |       | 354.00            | 352                     |
| <a href="#">master</a>             | 4.00            |       | 4.00              | 0                       | 1.25            |       | 1.25              | 0                       |
| <a href="#">MDW</a>                | 100.00          |       | 100.00            | 0                       | 10.00           |       | 10.00             | 0                       |
| <a href="#">model</a>              | 2.25            |       | 2.25              | 0                       | 0.75            |       | 0.75              | 0                       |
| <a href="#">msdb</a>               | 17.06           |       | 18.81             | 1.75                    | 2.75            |       | 2.75              | 0                       |
| <a href="#">tempdb</a>             | 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       |

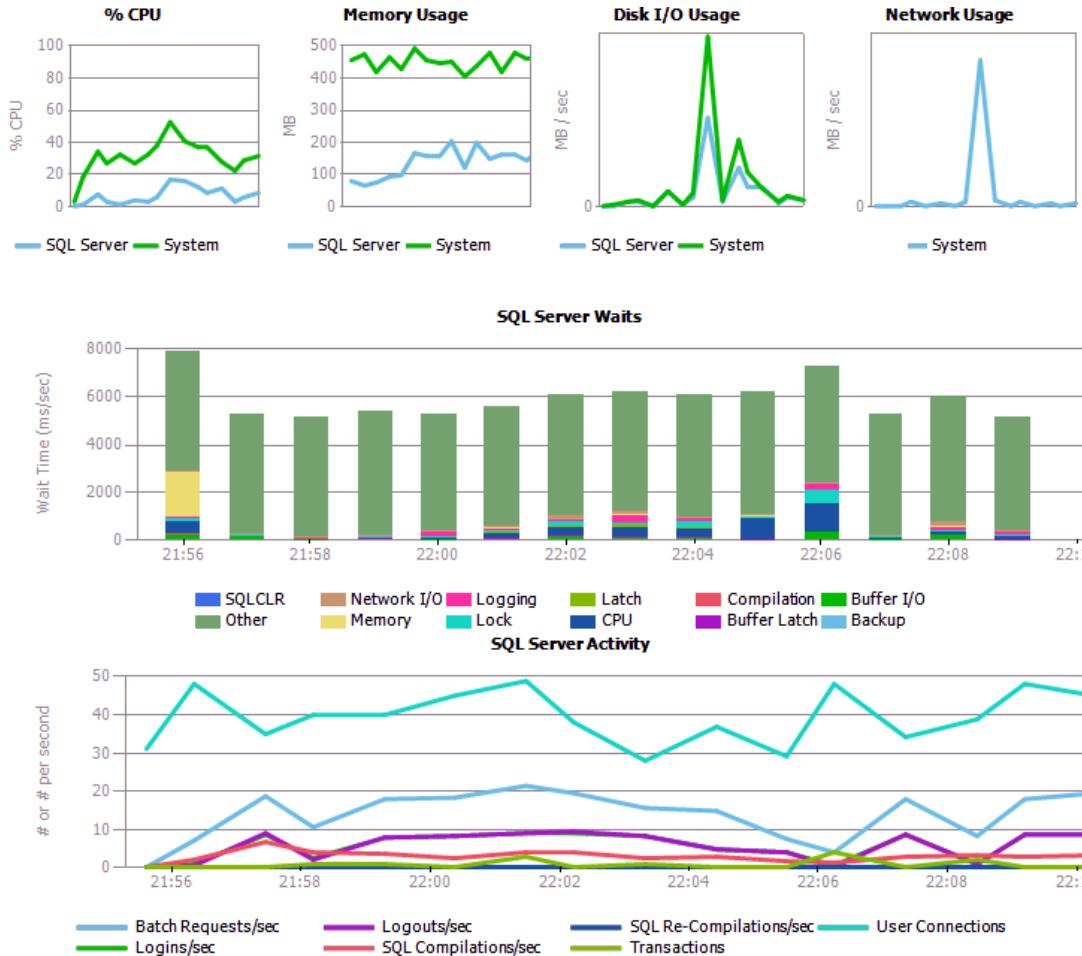
|                       |     |
|-----------------------|-----|
| Total CPU (sec):      | 1.0 |
| Total Duration (sec): | 8.0 |
| Total Physical Reads: | 0   |
| Total Logical Writes: | 0   |
| Total Executions:     | 2   |

|                                 |   |
|---------------------------------|---|
| 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 %<br>SOS_SCHEDULER_YIELD<br>CXPACKET           | % Processor Time<br>% User Time<br>% Privileged Time<br>Interrupts/sec<br>Processor Queue Length<br>Context Switches/sec                                                                                                           | sys.dm_os_wait_stats<br>sys.dm_exec_query_stats<br>sys.dm_exec_sql_text<br>sys.dm_exec_cached_plans<br>sys.dm_osSchedulers | Signal waits < 25%.<br>High % privileged time may indicate hardware/driver issue.<br>If Context Switches/sec > 20,000, consider fibre mode.<br>CXPACKET waits on OLTP systems < 5%. Consider MAXDOP 1 and index analysis.                                                                                              |
| Memory      | PAGEIOLATCH_*                                              | Page Life Expectancy<br>Buffer Cache Hit Ratio<br>Pages/sec<br>Page Faults/sec<br>Memory Grants Pending<br>CheckPoint Pages/sec<br>Lazy Writes/sec<br>Readahead Pages/sec                                                          | sys.dm_os_wait_stats<br>sys.dm_os_memory_clerks                                                                            | Page Life Expectancy > 500.<br>Sudden page life drops may indicate poor indexing.                                                                                                                                                                                                                                      |
| Disk IO     | PAGEIOLATCH_*<br>ASYNC_I/O_COMPLETION<br>WRITLOG<br>LOGMGR | Disk Seconds/Read and Write<br>Disk Reads and Writes/sec<br>% Disk Time<br>Current and Avg. Disk Queue Length<br>Log Flush Wait Time<br>Log Flush Waits/sec<br>Average Latch Wait Time<br>Latch Waits/sec<br>Total Latch Wait Time | sys.dm_os_wait_stats<br>sys.dm_exec_query_stats<br>sys.dm_exec_sql_text<br>sys.dm_io_virtual_file_stats                    | Ensure storage tested with SQLIO/SIM before production implementation.<br>tempdb separation.<br>Disk Layout best practices including autogrow/shrink and t-log separation.<br>Disk Sec/read and write <20 ms.<br>Transaction log disk < 10ms.<br>Disk queue length < 2 per disk in volume.                             |
| Network     | NET_WAITFOR_PACKET                                         | Bytes Received/sec<br>Bytes Sent/sec<br>Output Q length<br>Dropped/Discarded Packets                                                                                                                                               | sys.dm_os_wait_stats                                                                                                       | Consider number of application round trips.<br>Switched gigabit network connections.                                                                                                                                                                                                                                   |
| Blocking    | LOCK_*                                                     | Lock Waits/sec<br>Lock Wait Time                                                                                                                                                                                                   | sys.dm_os_wait_stats<br>sys.dm_db_index_operational_stats                                                                  | Consider SQL Profiler's blocked process report and deadlock graphs.<br>Check transaction length, isolation levels and optimistic locking.                                                                                                                                                                              |
| Indexing    | Refer disk and memory waits                                | Forwarded Records/sec<br>Full Scans/sec<br>Index Searches/sec<br>Pages Splits/sec                                                                                                                                                  | sys.dm_os_wait_stats<br>sys.dm_db_index_operational_stats                                                                  | Page Life Expectancy > 500.<br>Sudden page life drops may indicate poor indexing.                                                                                                                                                                                                                                      |
| Compilation | RESOURCE_SEMAPHORE_<br>QUERY_COMPILE                       | SQL Compilations/sec<br>SQL Recompilations/sec<br>Batch Requests/sec<br>Auto Param Attempts/sec<br>Failed Auto Param Attempts/sec<br>Cache Hit Ratio (Plan Cache)                                                                  | sys.dm_os_wait_stats<br>sys.dm_exec_cached_plans<br>sys.dm_exec_sql_text                                                   | Parameterized queries with sp_executesql.<br>Consider forced parameterization (after acknowledging possible downsides).<br>Consider "optimize for ad hoc workloads" (after acknowledging possible downsides).<br>Pay attention to large plan cache on 64-bit systems.<br>Plan reuse on an OLTP system should be > 90%. |

## Course Notes 12 - SQL Server Administration

# 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 database interface with two tabs: 'Results' and 'Messages'. The 'Results' tab is active, displaying a table with three columns: 'ID' (containing values 1, 2, 3, 2, 2, 2, 2, 3, 3), 'Value' (containing values 50.30, 123.30, 132.90, 50.30, 123.30, 132.90, 88.90, 50.30, 123.30), and a primary key column (containing values 1, 2, 3, 4, 5, 6, 7, 8, 9). The row with ID 1 is selected, indicated by a dashed border around its entire row.

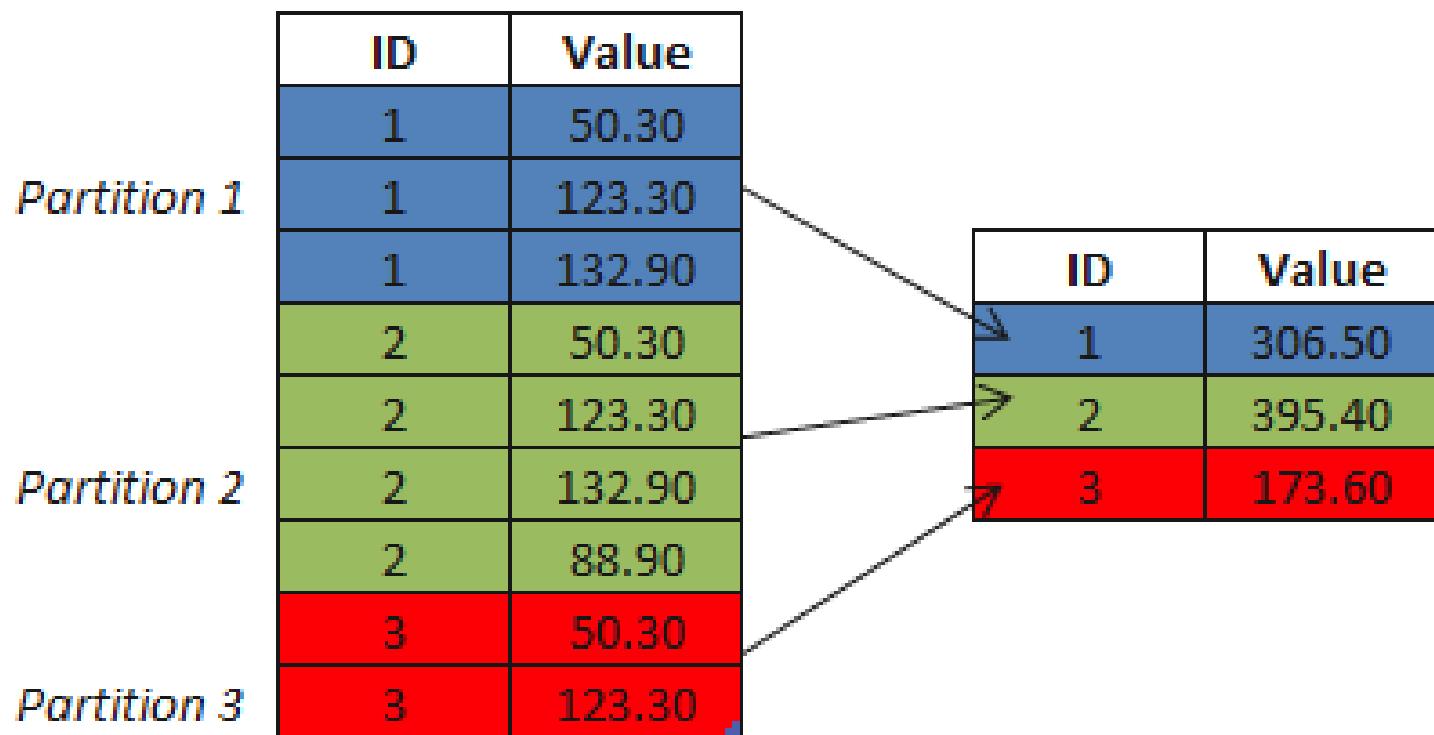
|   | ID | Value  |
|---|----|--------|
| 1 | 1  | 50.30  |
| 2 | 1  | 123.30 |
| 3 | 1  | 132.90 |
| 4 | 2  | 50.30  |
| 5 | 2  | 123.30 |
| 6 | 2  | 132.90 |
| 7 | 2  | 88.90  |
| 8 | 3  | 50.30  |
| 9 | 3  | 123.30 |

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

The screenshot shows a database interface with two tabs: 'Results' and 'Messages'. The 'Results' tab is active, displaying a table with three columns: 'ID' (containing values 1, 2, 3), 'Value' (containing values 306.50, 395.40, 173.60), and a column labeled '(No column name)' which contains the same values as the 'Value' column. The row with ID 1 is selected, indicated by a dashed border around its entire row.

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

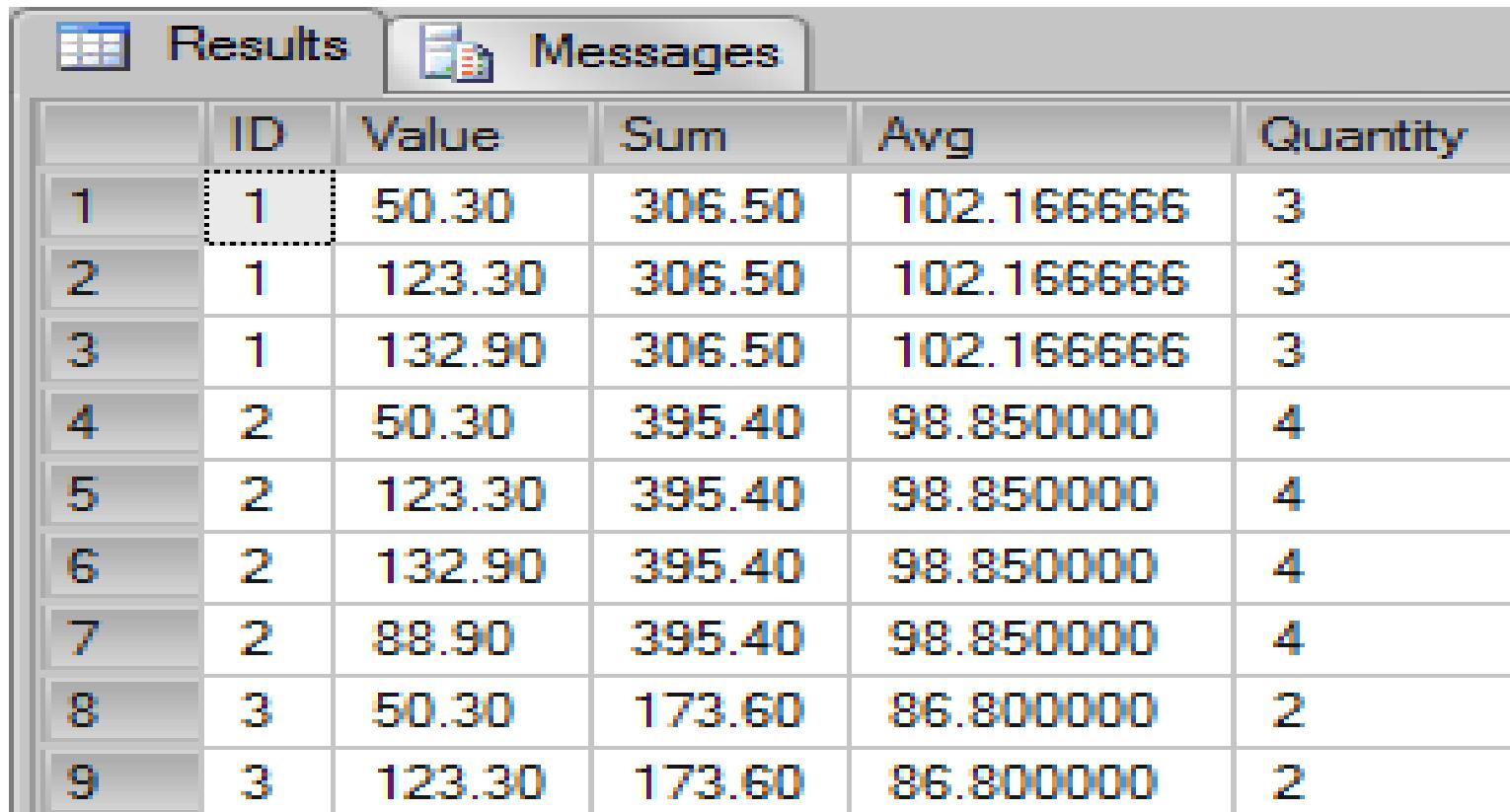
# Partitions



```

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

```



The screenshot shows the SQL Server Management Studio interface with the 'Results' tab selected. The results display a table with six columns: ID, Value, Sum, Avg, and Quantity. The data is grouped by ID, showing multiple rows for each ID with different values. The 'Messages' tab is also visible above the results.

|   | 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) ...

The diagram illustrates the effect of the `OVER(PARTITION BY ID)` clause. It shows a source table on the left and a target table on the right.

**Source Table:**

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

**Target Table:**

| ID | Value  | Sum    | Avg         | Quantity |
|----|--------|--------|-------------|----------|
| 1  | 50.30  | 306.50 | 102.166.666 | 3        |
| 1  | 123.30 | 306.50 | 102.166.666 | 3        |
| 1  | 132.90 | 306.50 | 102.166.666 | 3        |
| 2  | 50.30  | 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  | 88.90  | 395.40 | 98.850.000  | 4        |
| 3  | 50.30  | 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 $n^{\text{th}}$ 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

- $\ln(dp \text{ or numeric})$  - (same as input): natural logarithm
- $\log(dp \text{ or numeric})$  - (same as input): base 10 logarithm
- $\log(b \text{ numeric}, x \text{ numeric})$  - numeric: logarithm to base b
- $\mod(y, x)$  - (same as argument types): remainder of y/x
- $\pi()$  – dp: "π" constant  $\pi() = 3.14159265358979$
- $\text{pow}(a \text{ dp}, b \text{ dp})$  – dp: a raised to power of b
- $\text{pow}(a \text{ numeric}, b \text{ numeric})$  – numeric: a raised to power of b
- $\text{random}()$  -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 1<sup>st</sup> 0001 up to December 31<sup>st</sup> 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
- *cast(x.list + ', ' + e.ename as varchar(100)),*
- and WHERE clause from that same query
- *where e.deptno = x.deptno 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

- $x.len+1$
- increments LEN (which starts at 1) every time an employee has been evaluated; when incremented value equals number of employees in department
- $where len = cnt$
- 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

| Anchor UNION ALL 1. Recursive UNION ALL 2. Recursive UNION ALL ... 4. Recursive UNION ALL |     |                     |       |     |
|-------------------------------------------------------------------------------------------|-----|---------------------|-------|-----|
| deptno                                                                                    | cnt | list                | empno | len |
| 10                                                                                        | 3   | CLARK               | 7782  | 1   |
| 10                                                                                        | 3   | KING                | 7839  | 1   |
| 10                                                                                        | 3   | MILLER              | 7934  | 1   |
| 20                                                                                        | 5   | SMITH               | 7369  | 1   |
| 20                                                                                        | 5   | JONES               | 7566  | 1   |
| 20                                                                                        | 5   | SCOTT               | 7788  | 1   |
| 20                                                                                        | 5   | ADAMS               | 7876  | 1   |
| 20                                                                                        | 5   | FORD                | 7902  | 1   |
| 30                                                                                        | 6   | ALLEN               | 7499  | 1   |
| 30                                                                                        | 6   | WARD                | 7521  | 1   |
| 30                                                                                        | 6   | MARTIN              | 7654  | 1   |
| 30                                                                                        | 6   | BLAKE               | 7698  | 1   |
| 30                                                                                        | 6   | TURNER              | 7844  | 1   |
| 30                                                                                        | 6   | JAMES               | 7900  | 1   |
| 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   |
| 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   |

# 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

# **PIVOT**

# **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!**

# DataBases

no, php & html

# Objectives

- Present HTML (Hyper Text Markup Language)
  - in order to embed database programming in Web application it is necessary to understand first layer – HTML – where we will embed database related stuff
- Present Web architecture, Server Side Scripting / Client Side Scripting
- Web server (apache, MS Internet Information Server), database server, application layer / server (php, ASP, Java Server Pages)

# Objectives

- Present php language

# Dynamic web page

- Classical hypertext navigation occurs among "static" documents
- client-side scripting to change interface behaviors within a specific web page, in response to mouse or keyboard actions or at specified timing events. In this case the dynamic behavior occurs within the presentation.
- server-side scripting to change the supplied page source between pages, adjusting the sequence or reload of the web pages or web content supplied to the browser.  
Server responses may be determined by such conditions as data in a posted HTML form, parameters in the URL, the type of browser being used, the passage of time, or a database or server state.

# Client-side

- generated on the client's computer
- web server retrieves the page and sends it as is
- web browser then processes the code embedded in the page (normally JavaScript) and displays the page
- browsers dependent
- Ajax is a newer web development technique for creating client-side dynamic Web pages
- Google Maps is an example of a web application that uses Ajax techniques

# Ajax

- web development technique used for creating interactive web applications
- increase web page's interactivity, speed, functionality, and usability, make web pages feel more responsive by exchanging small amounts of data with server behind the scenes, so that the entire web page does not have to be reloaded each time the user requests a change
- JavaScript is the programming language in which Ajax function calls are made
- data retrieved is commonly formatted using XML
- asynchronous in that XML data loading does not interfere with normal HTML and JavaScript page loading

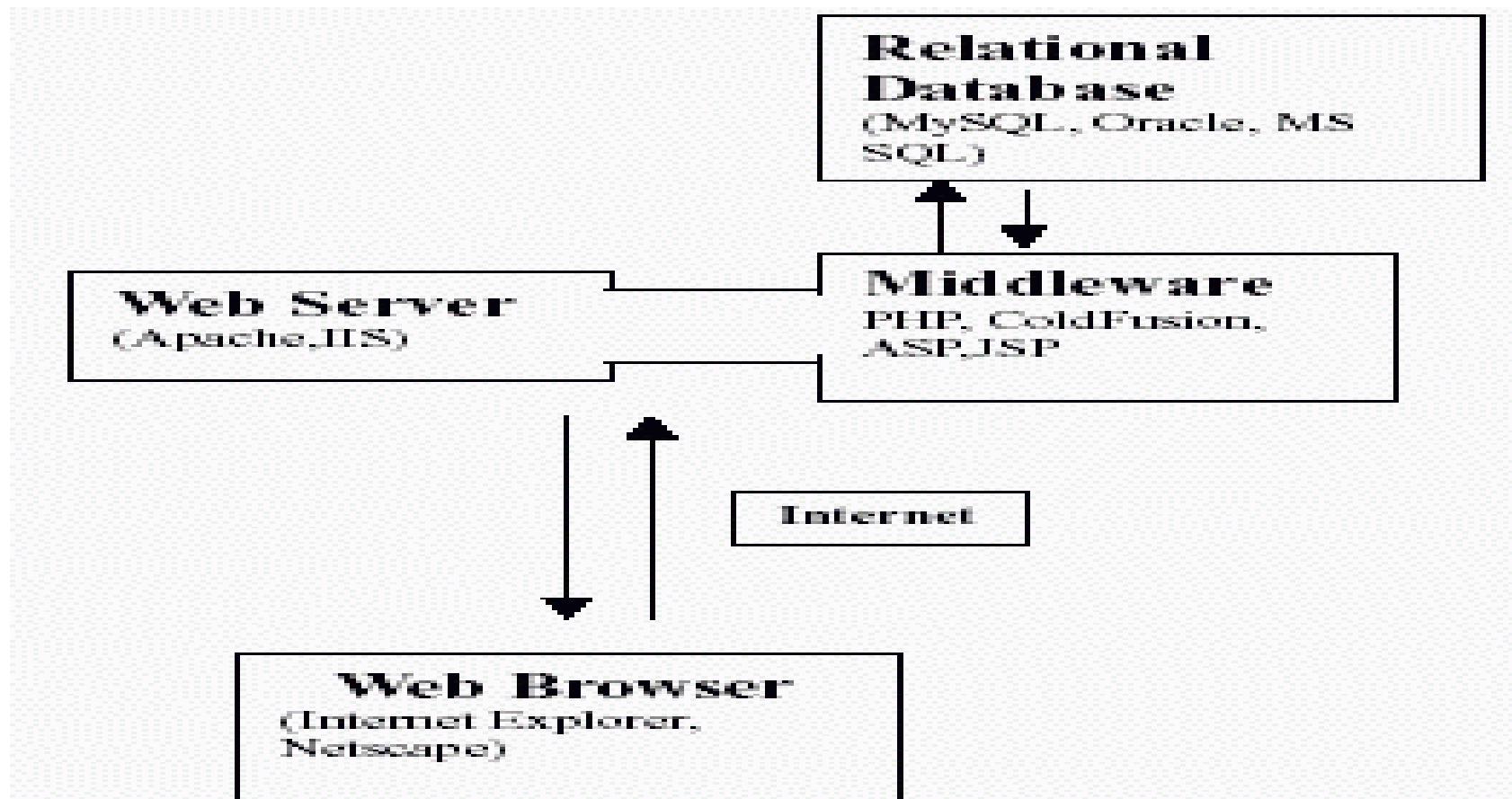
# Server-side

- browser sends an HTTP request.
- server retrieves the requested script or program
- server executes the script or program which typically outputs an HTML web page
  - program usually obtains input from the query string or standard input which may have been obtained from a submitted web form
- server sends the HTML output to the client's browser

# Server-side

- many possibilities for dynamic content
- can be a strain on low-end, high-traffic machines
- if not properly secured, server-side scripts could be exploited to gain access to a machine

# Database Web application



# Server-side script

- web server technology in which a user's request is fulfilled by running a script directly on the web server to generate dynamic HTML pages
- provide interactive web sites that interface to databases
- different from client-side scripting where scripts are run by the viewing web browser, usually in JavaScript or an Java applet
- primary advantage - ability to highly customize the response based on the user's requirements, access rights, or queries into data stores.

# Middleware Server-side script technologies

- Active Server Pages (ASP)
  - Microsoft designed solution allowing various languages (generally VBscript is used) inside a HTML-like outer page
  - Primarily a Windows technology, but with limited support on other platforms
- ASP.NET
  - Set of web application development technologies, part of Microsoft's .NET platform and is the successor to ASP
  - Programmers can use it to build dynamic web sites, web applications and XML web services

# Middleware Server-side script technologies

- ColdFusion
  - Cross platform tag-based commercial server side scripting system
- Java Server Pages (JSP)
  - A Java-based system for embedding Java-related code in HTML pages
- PHP
  - Common open source solution based on including code in its own language into an HTML page

# PHP

- recursive acronym for "PHP: Hypertext Preprocessor"
- reflective programming language originally designed for producing dynamic web pages
- used mainly in server-side scripting

# PHP

- main implementation produced The PHP Group
  - implementation serves to define de facto standard
    - there is no formal specification
- released under the PHP License
  - considered to be free software
- currently, two major versions of PHP are being actively developed: 5.x and 4.4.x

# PHP

- Paradigm: imperative, object-oriented
- Appeared in: 1995
- Latest release: 5.2.4/ 30 Aug 2007
- Influenced by: C, Perl, Java, C++, Python
- OS: Cross-platform
- Website: *<http://php.net/>*

# SQL

- Paradigm: Declarative (contrast: Imperative)
- Appeared in: 1974
- Latest release: 2008
- Influenced by: Datalog, a subset of Prolog
- C.J. Date, E.F. Codd, R.F. Boyce, ... IBM

# PHP

- generally runs on a web server
  - taking PHP code as its input and
  - creating Web pages as output
- acts as a filter
  - takes input from a file or stream containing text and special PHP instructions and
  - outputs another stream of data for display

# PHP

- can be deployed on most web servers and on almost every operating system and platform
- can be used with a large number of relational database management systems
- millions Internet domains are currently hosted on servers with PHP installed

# LAMP architecture

- popular in the Web industry as a way of deploying inexpensive, reliable, scalable, secure web applications
- PHP is commonly used as the P in this bundle alongside Linux, Apache and MySQL, although the P can also refer to Python or Perl

# LAMP architecture

- solution stack of software
- usually free software / open-source software
- used to run dynamic Web sites or servers
- the original expansion is as follows:
  - Linux, referring to the operating system;
  - Apache, the Web server;
  - MySQL, the database management system
  - PHP, the programming language

# LAMP / WAMP architecture

- Linux, GNU/Linux a Unix-like computer operating system / Microsoft Windows families of software operating systems
- Apache HTTP Server (<http://httpd.apache.org>) - free software/open source web server, the most popular in use; serves as the de facto reference platform against which other web servers are designed and judged
- MySQL ([mysql.com](http://mysql.com)) - multithreaded, multi-user, SQL Database Management System (DBMS) with more than ten million installations

# LAMP / WAMP architecture

- A Microsoft Windows-based variant of a LAMP (software bundle)
  - WAMP5
    - *www.en.wampserver.com*
  - **WOS Portable**
    - ***www.chsoftware.net/en/useware/wos/wos.htm***

# HyperText Markup Language

- predominant markup language for web pages
- provides a means to describe the structure of text-based information in a document — by denoting certain text as headings, paragraphs, lists, and so on — and to supplement that text with interactive forms, embedded images, and other objects
- written in the form of labels (known as tags), surrounded by angle brackets
- can also describe, to some degree, the appearance and semantics of a document
- can include embedded scripting language code which can affect the behavior of web browsers and other HTML processors

- You can, of course, use a WYSIWYG (What You See Is What You Get) HTML editor to make websites but they have 3 main disadvantages:
  - They sometimes use excess code to create a look on a page which slows down loading times
  - They do not always create fully compatible code
  - Some WYSIWYG editors change any HTML code you enter by hand

# HTML tags

- <tag>
- There are two types of tag. Opening and closing tags. Closing tags are only different as they have a / before them:
- </tag>
- Nearly all tags have a closing tag but a few do not.

# HTML tags

```
<html>
 <head>
 <title>Untitled</title>
 </head>

 <body>
 ...
 </body>
</html>
```

# HTML tags

- <html>
  - beginning of an HTML document
- <head>
  - beginning of the header section - contains configuration options for the page
- <title>Untitled</title>
  - tells the browser what to display as title of page
  - appears in the title bar at the top of the browser
- </head>
  - end of the header section

# HTML tags

- <body>
  - Everything between these is in the body of the page
  - This is where all text, images etc. are.
  - This is the most important part of the page.
- </body>
- </html>
  - end of the HTML document

# <font> tag

- display the text in a standard font
- tag attributes
- <font face="Arial" size="7" color="#FF0000">text</font>

# <p> tag

- stands for Paragraph; used to break up text into paragraphs
- <p> tag has attribute which can be added to it - align option
- <p align="right / left / center ">Text</p>
- can use the <center> tag or the <p align="center">
- hardly ever necessary to use the align="left" attribute as nearly all browsers automatically align text to the left

# <br> tag

- want to leave a space after your paragraphs
- should use the <Br> (break) tag
- there is no end tag
- <br> = <BR> = <Br> = <bR>
- Tags are case insensitive

<!-- Your comment -->

# <img> tag

- Images are added to pages
- must use the src attribute to choose the image to insert
- can either be a relative reference or a direct reference
- can resize images inside the browser using two other image attributes: width and height
- Alt tells the browser what the alternative text for an image should be if the browser has images turned off

# <a> tag

- used when creating hyperlinks and bookmarks; stands for anchor
- need to use the href attribute
- to make a piece of text or an image into a hyperlink you contain it in:

```
 Link
```

# Hyperlinks

- Web Page or Site <a href="http://www.webaddress.com/folder/page">
- Local Page <a href="pagename.html">
- Local Page In A Folder Level Below <a href="foldername/pagename.html">
- Local Page In A Folder Level Above <a href="../pagename.html">
- Open E-mail Program With E-mail Addressed <a href="mailto:yourname@yourname.com">
- Bookmarked Section <a href="#bookmarkname">
- Bookmarked Section In Another Page <a href="pagelocation.htm#bookmarkname">

# Bookmarks

- instead of changing the href variable use the name variable. For example:
- <a name="top"></a>
- will create a bookmark called top in the text
- can then link to this using a standard hyperlink:
- <a href="#top">Back To Top</a>

# PHP syntax

- parses code within its delimiters
  - <?php ... ?>
- allow statements to be embedded within HTML documents
- print & echo functions
  - practically identical

```
<html>
 <head> <title></title>
 </head>
 <body>
 <?php
echo 'Hello, World!';
 ?>
 </body>
</html>
```

# PHP syntax

- variables are prefixed with a dollar symbol \$
  - type does not need to be specified in advance
- unlike function and class names, variable names are case sensitive
- treats new lines as whitespace
- statements are terminated by a semicolon
- three types of comment syntax:
  - /\* \*/ which serves as block comments
  - // as well as # which is used for inline comments

# Data types

- Integer
- Double
- Boolean
- String
- Object
- Array
- Null
- Resource

# Data types

- numbers in a platform-dependent range - range typically 32-bit signed integers
- Integer variables can be assigned using decimal (positive and negative), octal and hexadecimal notations
- Real numbers can be specified using floating point notation, or two forms of scientific notation
- native Boolean type
  - Using Boolean type conversion rules, non-zero values can be interpreted as true and zero as false

# Data types

- Null data type represents a variable that has no value
  - only value in the Null data type is NULL
- Arrays support both numeric and string indices, and are heterogeneous
- Arrays can contain elements of any type that PHP can handle, including resources, objects, and even other arrays

# Source code encoders, optimizers and accelerators

- PHP scripts are normally kept as human-readable source code, even on production web servers
  - While this allows flexibility, it can raise issues with security and performance
- Encoders hinder source code reverse engineering
  - hide source code
  - compile code into "opcode"
  - downside of this latter approach is that a special extension has to be installed on the server in order to run encoded scripts,
  - however the approach of encoding compiled code and use of an extension offers typically the best performance, security and opportunity for additional features that may be useful for developers

# Source code encoders, optimizers and accelerators

- Code optimizers improve the quality of the compiled code by reducing its size and making changes that can reduce the execution time and improve performance. The nature of the PHP compiler is such that there are often many opportunities for code optimization.
- Accelerators offer performance gains by caching the compiled form of a PHP script in shared memory to avoid the overhead of parsing and compiling the code every time the script runs. They may also perform code optimization to provide increased execution performance
- most commonly used packages for source code protection are from Zend Technologies

# Testing PHP script

- write file
- upload it to your Web Server
- using your browser, go to the URL of the script
- *C:\...path from root of drive...\script.php*
- *http://localhost\...path from root of WebServer...\script.php*

# Outputting Variables

```
<?php
$welcome_text = "Hello and welcome to my
website.;"
print($welcome_text);
?>
```

# Formatting Text

```
print("<font face=\"FontFace\"
size=\"FontSize\" color=\"FontColor\">Hello
and welcome to my website.");
```

# IF Structure

```
IF (something == something else)
{
 THEN Statement
} else {
 ELSE Statement
}
```

# WHILE, DO WHILE

while (expr)  
    statement

do statement  
while (expr)

# FOR

```
for (expr1; expr2; expr3)
statement
```

# Arrays

- `$array[key] = value;`
- should always use quotes around a string literal array index
  - For example, use `$foo['bar']` and not `$foo[bar]`
- do not quote keys which are constants or variables, as this will prevent PHP from interpreting them
- It works because PHP automatically converts a bare string (unquoted string which does not correspond to any known symbol) into string
- PHP may in future define constants which, unfortunately for your code, have the same name

# Arrays

```
exArr[1] = "one";
exArr[2] = 2;
exArr[1] = "trei";
```

```
band["Brian"] = "May";
band["Roger"] = "Taylor";
band["John"] = "Deacon";
```

```
echo "The fourth member of
the band is $band[3];"
```

# User-defined functions

```
function name($arg1, $arg2, ..., $argn)
{
 echo "Example function.\n";
 return $retval;
}
```

# Functions

- All functions and classes in PHP have the global scope - they can be called outside a function even if they were defined inside and vice versa
- PHP does not support function overloading
- Function and classes names are case-insensitive
- support variable numbers of arguments to functions, default arguments
- It is possible to call recursive functions
  - avoid recursive function/method calls with over 100-200 recursion levels as it can smash the stack and cause a termination of the current script

# File include

- When a file is included, the code it contains inherits the variable scope of the line on which the include occurs. Any variables available at that line in the calling file will be available within the called file, from that point forward. However, all functions and classes defined in the included file have the global scope.

# Basic include() example

- GeneralFunctions.php

```
function DataBaseConnect ($arg1, $arg2, ...,
 $argn)
{
}
}
```

- test.php

```
include 'GeneralFunctions.php';
DataBaseConnect ();
```

# Setting Up Forms

- A form is an area that can contain form elements.
- Form elements are elements that allow the user to enter information in a form
- A form is defined with the `<form> </form>` tags

```
<form action="process.php" method="post">
```

# Text Fields

- when you want the user to type letters, numbers, etc. in a form

```
<form>
```

First name:

```
<input type="text" name="firstname">
```

```


```

Last name:

```
<input type="text" name="lastname">
```

```
</form>
```

- in most browsers, the width of the text field is 20 characters by default

# Radio Buttons

- used when you want the user to select one of a limited number of choices

```
<form>
```

```
 <input type="radio" name="sex" value="male">
 Male
```

```


```

```
 <input type="radio" name="sex" value="female">
 Female
```

```
</form>
```

- Note that only one option can be chosen

# Checkboxes

- used when you want the user to select one or more options of a limited number of choices

```
<form>
```

I have a bike:

```
<input type="checkbox" name="vehicle" value="Bike" />
```

```


```

I have a car:

```
<input type="checkbox" name="vehicle" value="Car" />
```

```


```

I have an airplane:

```
<input type="checkbox" name="vehicle" value="Airplane" />
```

```
</form>
```

# Drop Down list

```
<form>
<select name="cars">
<option value="volvo">Volvo</option>
<option value="saab">Saab</option>
<option value="fiat"
 selected="selected">Fiat</option>
<option value="audi">Audi</option>
</select>
</form>
```

# Textarea

- multi-line text input control; user can write text (an unlimited number of characters) in the text-area

```
<textarea rows="10" cols="30">
```

Initial text to be displayed in text-area.

```
</textarea>
```

# Submit button

```
<input type="submit" value="Submit">
<input type="button" value="Hello world!">
<input type="reset" value="Reset">
<input type="hidden" value="Hidden">
```

# Form

```
<form action="process.php" method="post">
...
<input name = ...>
<select name = ...>
<input type="submit" value = ...">
<input type="button" value = ...>
...
</form>
```

In process.php?param1=val1&

```
$variablename=$_POST['variable'];
```

```
$variablename=$_GET['variable'];
```

- basically takes the variable from the POST or GET method (the name of a form field) and assigns it to the variable

```
$variablename
```

```
$variablename=$_GET['param1'];
```

# Predefined Variables

- Server variables: `$_SERVER`
- Environment variables: `$_ENV`
- HTTP Cookies: `$_COOKIE`
- HTTP File upload variables: `$_FILES`
- Request variables: `$_REQUEST`
- Session variables: `$_SESSION`

# Predefined Functions

- IBM DB2 Universal Database
- MS SQL Server database
- MySQL database
- Oracle database
- PostgreSQL
- ODBC access several databases that have borrowed the semantics of the ODBC API
  - unified into a single set of ODBC functions

# Setting Up The Database

- See Database course ...

# Inserting Information

- And also Updating, Deleting ...
- See Database course ...
  - SQL part

# Displaying Data

```
bool mssql_connect(
 [string $servername
 [, string $username
 [, string $password
 [, bool $new_link]]])

bool mssql_select_db (string
 $database_name
 [, resource $link_identifier])
```

# Connecting Database

```
$conn = mssql_connect('MYSQLSERVER', 'sa',
 'password');
mssql_select_db('[my data-base]', $conn);
```

# Displaying Data from contacts

- Select all the records in the contacts table
- There will be output from this command it must be executed with the results being assigned to a variable:

```
$query="SELECT * FROM contacts";
$result=mssql_query($query);
```

- In this case the whole contents of the database's table is now contained in a special array with the name \$result.
- Before you can output this data you must change each piece into a separate variable

# Query database

```
mixed mssql_query (string $query
[, resource $link_identifier
[, int $batch_size]])
```

# Counting Rows

```
$num=mssql_numrows($result);
```

```
int mssql_num_rows (resource $result)
```

- will set the value of \$num to be the number of rows stored in \$result (the output you got from the database)
  - can then be used in a loop to get all the data and output

# Setting Up The Loop

- \$i is the number of times the loop has run and is used to make sure the loop stops at the end of the results so there are no errors

```
$i=0;
while ($i < $num) {
```

CODE

```
$i++;
}
```

- For can be used also

# Assigning Data To Variables

```
string mssql_result (resource $result, int
$row, mixed $field)
```

- field can be the field's offset, the field's name or the field's table dot field's name (tablename.fieldname)

```
$variable=mysql_result($result,$i,"fieldname
");
```

- Recommended high-performance alternatives:
  - `mssql_fetch_row()`
  - `mssql_fetch_array()`
  - `mssql_fetch_object()`

# Displaying Data from contacts

```
$first=mssql_result($result,$i,"first");
$last=mssql_result($result,$i,"last");
$phone=mssql_result($result,$i,"phone");
$mobile=mssql_result($result,$i,"mobile");
$fax=mssql_result($result,$i,"fax");
$email=mssql_result($result,$i,"email");
$web=mssql_result($result,$i,"web");
```

# Displaying Data from contacts

```
mssql_connect('localhost','username','password');
@mssql_select_db($database) or die("Unable to select
database");

$query="SELECT * FROM contacts";
$result=mysql_query($query);

$num=mysql_numrows($result);

mysql_close();
```

# Combining The Script

```
echo "<center>Database
Output</center>

";
```

```
$i=0;
while ($i < $num) {

$first=mysql_result($result,$i,"first");
...
$web=mysql_result($result,$i,"web");
```

# Format the output

```
echo "$first $last
Phone: $phone
Mobile:
$mobile
Fax: $fax
E-mail: $email
Web:
$web
<hr>
";
```

```
$i++;
}
```

# MYSQL DATABASES

# Data Types

- many data types in programming languages, probably will only need a cursory examination
- numeric
- character
- date
- other types

# Numeric Types

- used to store numbers
- specific functions, arithmetic, that you can perform on numbers and on most numeric data types

# Integers

- exact whole number
- *columnname INT*
- TINYINT
  - -128 ... +127, 1 byte
- SMALLINT
  - -32 thousand ... +32 thousand, 2 bytes
- MEDIUMINT
  - -8.3 million ... +8.3 million, 3 bytes
- INT
  - -2 billion (1012) ... +2 billion, 4 bytes
- BIGINT
  - -9 trillion (1018) ... +9 trillion, 8 bytes

# UNSIGNED

- will only store numbers as positive integers
- range then goes from zero through to double the values
- *columnname INT UNSIGNED*
- fine for using integers as unique IDs for rows
- but if you start doing arithmetic with signed and unsigned data types it can lead to problems

# BOOL

- one that can have two states, true or false, on or off, 0 or 1
- actually is the same as using TINYINT(1) data type
- it is best just to use TINYINT, and then within your code determine what number you need to store for on or off, 0 or 1
- if you are transferring your MySQL tables to another system that implements BOOL (other data types) in a different way, the scripts that you need to write to export the data will need to create the actions necessary to reflect that difference

# DECIMAL

- used for storing numbers that are not whole numbers, where the numbers after the decimal point are important, especially currency
- *columnname DECIMAL(precision, decimals)*
- precision
  - number of digits that are needed to store the complete number
  - if you are storing negative numbers the minus sign is included in these digits
- decimals
  - number of digits to store after the decimal point
- decimal number stored can be different from inserted

# Floating Point Numbers

- data types that have previously been mentioned are for storing exact values
- floating point numbers enable you to store a very wide range of numbers but with some levels of inaccuracy
  - the bigger the number gets, the less detail is stored about it

# FLOAT

- *columnname FLOAT(precision)*
- precision
  - number of digits that are to be stored
- *columnname FLOAT(magnitude, decimals)*
- magnitude
  - number of digits used to store the number
- decimals
  - number of decimal digits to be stored
- forced to be of the single precision type

# DOUBLE

- works in exactly the same way as the FLOAT declaration
- *columnname DOUBLE (magnitude, decimals)*
- number stored will obviously be of the double precision type

# CHAR

- *columnname CHAR(length)*
- length
  - width of the string in characters
  - can be between 0 and 253
- once you declare a column with CHAR value of fixed length, all of the strings that are stored in that column become that length

# VARCHAR

- *columnname VARCHAR(length)*
- Length
  - maximum width of the string in characters
  - can be between 0 and 253
- once you declare a column with CHAR value of fixed length, all of the strings that are stored in that column become that length

# Fixed or Variable Length?

- variable length data types make it easier for storing and searching
- if you are searching for the word “Hello!” in a CHAR(10) column, nothing would be found, instead you have to search for “Hello!....”
- fixed length types have the advantage that they are faster to search
- Phone number (characters no arithmetic performed) – CHAR – they are fixed length
- Names – VARCHAR – they are variable length

# TEXT

- storing things like comments, reviews, or other large blocks of text
- *columnname TEXT*
- *TINYTEXT*
  - *maximum length of string in characters is 255*
- *TEXT*
  - *maximum length of string in characters is 65,535*
- *MEDIUMTEXT*
  - *maximum length of string in characters is 16,777,215*
- *LONGTEXT*
  - *maximum length of string in characters is 4,294,967,295*

# BLOB

- Binary Large OBject used for storing data
  - could store jpeg image within a BLOB column
- conform to same size constraints as TEXT columns mentioned previously but are called TINYBLOB, BLOB, MEDIUMBLOB and LONGBLOB
- when you are searching or ordering it BLOB will be case sensitive and TEXT will not be
- both BLOB and TEXT objects are really pointers to areas of storage on server, and so are not physically stored within table
- explains why operations on this type of column are slower than when searching normal column types

# DATETIME

- allows you to store the date and the time in one column
- *columnname DATETIME*
- details how to insert values into this column type ...?
- when you retrieve DATETIME value, MySQL returns string of format “YYYYMM-DD HH:MM:SS”
- if you only insert date part the HH:MM:SS are set by default to 00:00:00
- if you insert date that is invalid stores this date as “0000-00-00 00:00:00” to show that there has been an error
- can store values for all dates and times between the year 1000 and the year 9999

# TIMESTAMP

- works in a similar way to DATETIME
- *columnname TIMESTAMP(length)*
- has added functionality that it will automatically update itself under certain conditions
  - whenever you create new row, or change contents of row, and do not explicitly change contents of the first TIMESTAMP column in that row, this column will be automatically updated with the date and time of the change
  - if, however, you explicitly set the value of the column when you are altering or creating row, the TIMESTAMP column will not auto-update because you have specified what you want to go in it manually

- DATE
- store date but do not need to store time
- *columnname DATE*
- TIME
- store time value or period
- *columnname TIME*

# Other Types

- SET
- ENUM

- most difficult data type: date @ time
- different database - different functions implemented to deal with date & time
- other database – can have no SET or ENUM data types

# Data Types

- standard – numbers, characters, text, blob
  - pretty much the same in all databases
- date & time
  - exist in all databases
  - different representations, different functions to deal with – difficult to export - import
- non standard
  - difficult or impossible to export - import

# POPULATING THE DATABASE

# INSERT

- main method of putting data into table
- *INSERT INTO tablename (columnname, columnname, ...)*
- *VALUES (value, value, ...);*
- adds new record to table, putting date given into fields specified
- must ensure that all column names specified have corresponding entries in VALUES section
- values must be same data types as column they are going to be inserted into

# INSERT

- any columns that are not specified are left at their default value if you specified it, or set to NULL
- for example, if you have a column that is auto-generating (AUTO\_INCREMENT)

# INSERT

- *INSERT INTO webpage (Title)*
- *VALUES (“Home”)*
- click onto Execute Query button to run
- to view contents
- *SELECT \* FROM webpage*

# INSERT All Columns

- *INSERT INTO tablename*
- *VALUES (value, value, ...);*
- *INSERT INTO webpage VALUES (6, “,”How to Contact Me”)*
- auto-generating field can lead to errors, run above query a second time has the error message:
- Duplicate entry ‘6’ for key 1
- ID column is primary key and so wouldn’t allow you to insert key with that value a second time

# INSERT Columns from Query

- *INSERT INTO tablename*
- *SELECT rows*
- *FROM anothertablename*
- *WHERE condition;*

# Inserting Data with Scripts

- run a SQL script
- collection of SQL queries separated by colons and stored in text file

# Inserting Data with Scripts

- *USE weblogdatabase;*
- *CREATE TABLE Cookies (*
- *CookieID MEDIUMINT NOT NULL AUTO\_INCREMENT PRIMARY KEY,*
- *DateCreated DATETIME);*
- *INSERT INTO Cookies (DateCreated) VALUES (“2002-01-01”);*
- *INSERT INTO Cookies (DateCreated) VALUES (“2002-01-02”);*
- *INSERT INTO Cookies (DateCreated) VALUES (“2002-01-04”);*
- *INSERT INTO Cookies (DateCreated) VALUES (“2002-01-07”);*
- *INSERT INTO Cookies (DateCreated) VALUES (“2002-01-13”);*
- *INSERT INTO Cookies (DateCreated) VALUES (“2002-01-22”);*
- *(phpMyAdmin – Export / Import – SQL)*

# Directly Inserting Data

- *LOAD DATA INFILE ‘filename’*
- *INTO TABLE tablename (Column1, Column2, ...)*

# Directly Inserting Data

- file has commas separating columns, standard of representing data in text files, called Comma Separated Variable file
- by default LOAD DATA command looks for tab to separate columns
- have to tell the separators are commas
- done by adding following to the query
- FIELDS TERMINATED BY ‘,’

# Directly Inserting Data

- first line is list of column names in the order of the file
- useful information to us but not to LOAD DATA command
- we could delete that first row which may be needed by another program
- we need to add the line:
- *IGNORE 1 LINES*

# UPDATE Statement

- used to update existing records in a table
- `UPDATE table_name  
SET column1=value1, column2=value2, ...  
WHERE some_column=some_value;`

# DELETE Statement

- used to delete rows in a table
- `DELETE FROM table_name  
WHERE some_column=some_value;`

# **Thank you for your kindly attention!**

# DataBases

no, php & html

# Objectives

- Present HTML (Hyper Text Markup Language)
  - in order to embed database programming in Web application it is necessary to understand first layer – HTML – where we will embed database related stuff
- Present Web architecture, Server Side Scripting / Client Side Scripting
- Web server (apache, MS Internet Information Server), database server, application layer / server (php, ASP, Java Server Pages)

# Objectives

- Present php language

# Dynamic web page

- Classical hypertext navigation occurs among "static" documents
- client-side scripting to change interface behaviors within a specific web page, in response to mouse or keyboard actions or at specified timing events. In this case the dynamic behavior occurs within the presentation.
- server-side scripting to change the supplied page source between pages, adjusting the sequence or reload of the web pages or web content supplied to the browser.  
Server responses may be determined by such conditions as data in a posted HTML form, parameters in the URL, the type of browser being used, the passage of time, or a database or server state.

# Client-side

- generated on the client's computer
- web server retrieves the page and sends it as is
- web browser then processes the code embedded in the page (normally JavaScript) and displays the page
- browsers dependent
- Ajax is a newer web development technique for creating client-side dynamic Web pages
- Google Maps is an example of a web application that uses Ajax techniques

# Ajax

- web development technique used for creating interactive web applications
- increase web page's interactivity, speed, functionality, and usability, make web pages feel more responsive by exchanging small amounts of data with server behind the scenes, so that the entire web page does not have to be reloaded each time the user requests a change
- JavaScript is the programming language in which Ajax function calls are made
- data retrieved is commonly formatted using XML
- asynchronous in that XML data loading does not interfere with normal HTML and JavaScript page loading

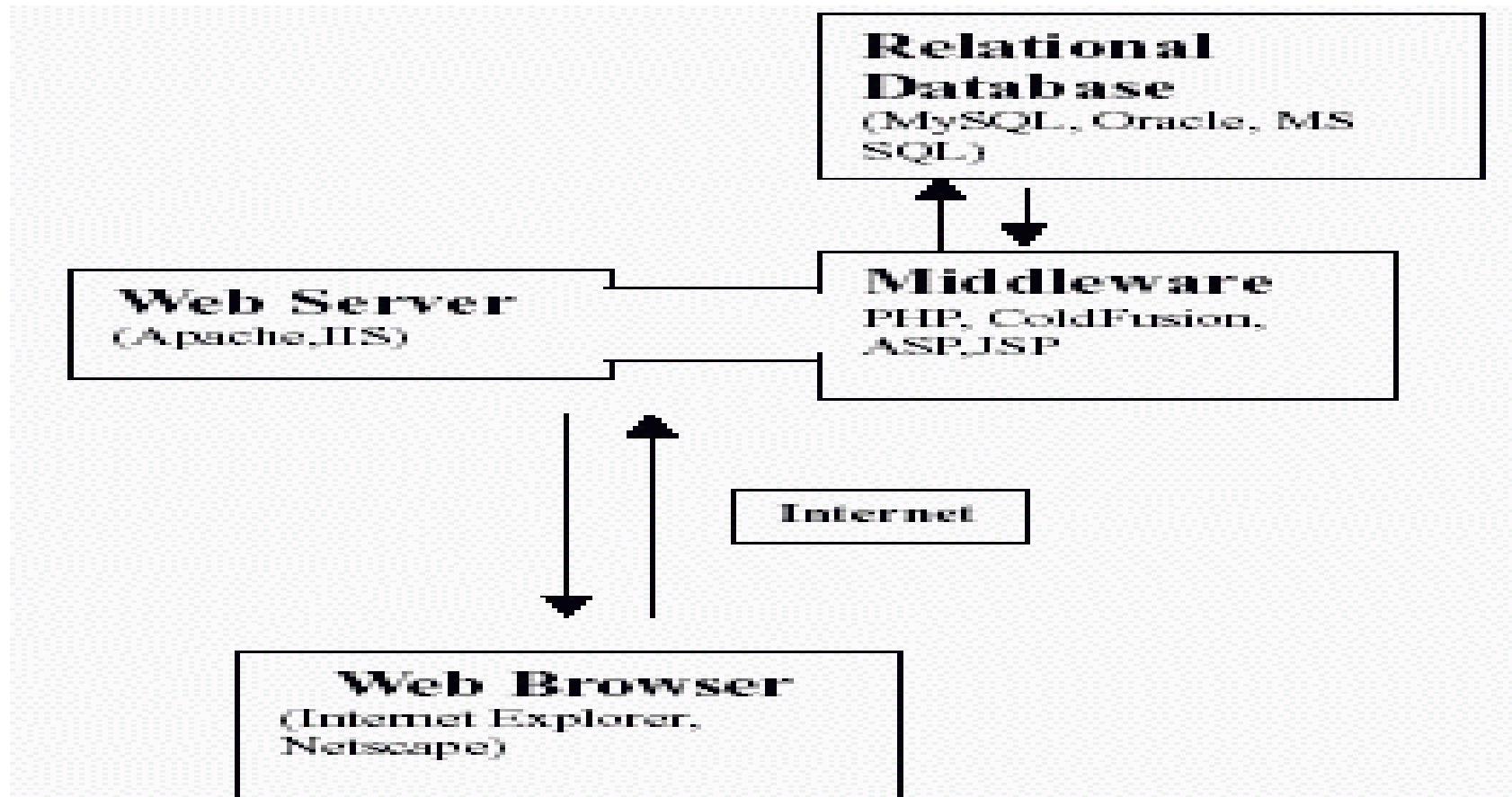
# Server-side

- browser sends an HTTP request.
- server retrieves the requested script or program
- server executes the script or program which typically outputs an HTML web page
  - program usually obtains input from the query string or standard input which may have been obtained from a submitted web form
- server sends the HTML output to the client's browser

# Server-side

- many possibilities for dynamic content
- can be a strain on low-end, high-traffic machines
- if not properly secured, server-side scripts could be exploited to gain access to a machine

# Database Web application



# Server-side script

- web server technology in which a user's request is fulfilled by running a script directly on the web server to generate dynamic HTML pages
- provide interactive web sites that interface to databases
- different from client-side scripting where scripts are run by the viewing web browser, usually in JavaScript or an Java applet
- primary advantage - ability to highly customize the response based on the user's requirements, access rights, or queries into data stores.

# Middleware Server-side script technologies

- Active Server Pages (ASP)
  - Microsoft designed solution allowing various languages (generally VBscript is used) inside a HTML-like outer page
  - Primarily a Windows technology, but with limited support on other platforms
- ASP.NET
  - Set of web application development technologies, part of Microsoft's .NET platform and is the successor to ASP
  - Programmers can use it to build dynamic web sites, web applications and XML web services

# Middleware Server-side script technologies

- ColdFusion
  - Cross platform tag-based commercial server side scripting system
- Java Server Pages (JSP)
  - A Java-based system for embedding Java-related code in HTML pages
- PHP
  - Common open source solution based on including code in its own language into an HTML page

# PHP

- recursive acronym for "PHP: Hypertext Preprocessor"
- reflective programming language originally designed for producing dynamic web pages
- used mainly in server-side scripting

# PHP

- main implementation produced The PHP Group
  - implementation serves to define de facto standard
    - there is no formal specification
- released under the PHP License
  - considered to be free software
- currently, two major versions of PHP are being actively developed: 5.x and 4.4.x

# PHP

- Paradigm: imperative, object-oriented
- Appeared in: 1995
- Latest release: 5.2.4/ 30 Aug 2007
- Influenced by: C, Perl, Java, C++, Python
- OS: Cross-platform
- Website: *<http://php.net/>*

# SQL

- Paradigm: Declarative (contrast: Imperative)
- Appeared in: 1974
- Latest release: 2008
- Influenced by: Datalog, a subset of Prolog
- C.J. Date, E.F. Codd, R.F. Boyce, ... IBM

# PHP

- generally runs on a web server
  - taking PHP code as its input and
  - creating Web pages as output
- acts as a filter
  - takes input from a file or stream containing text and special PHP instructions and
  - outputs another stream of data for display

# PHP

- can be deployed on most web servers and on almost every operating system and platform
- can be used with a large number of relational database management systems
- millions Internet domains are currently hosted on servers with PHP installed

# LAMP architecture

- popular in the Web industry as a way of deploying inexpensive, reliable, scalable, secure web applications
- PHP is commonly used as the P in this bundle alongside Linux, Apache and MySQL, although the P can also refer to Python or Perl

# LAMP architecture

- solution stack of software
- usually free software / open-source software
- used to run dynamic Web sites or servers
- the original expansion is as follows:
  - Linux, referring to the operating system;
  - Apache, the Web server;
  - MySQL, the database management system
  - PHP, the programming language

# LAMP / WAMP architecture

- Linux, GNU/Linux a Unix-like computer operating system / Microsoft Windows families of software operating systems
- Apache HTTP Server (<http://httpd.apache.org>) - free software/open source web server, the most popular in use; serves as the de facto reference platform against which other web servers are designed and judged
- MySQL ([mysql.com](http://mysql.com)) - multithreaded, multi-user, SQL Database Management System (DBMS) with more than ten million installations

# LAMP / WAMP architecture

- A Microsoft Windows-based variant of a LAMP (software bundle)
  - WAMP5
    - *www.en.wampserver.com*
  - **WOS Portable**
    - ***www.chsoftware.net/en/useware/wos/wos.htm***

# HyperText Markup Language

- predominant markup language for web pages
- provides a means to describe the structure of text-based information in a document — by denoting certain text as headings, paragraphs, lists, and so on — and to supplement that text with interactive forms, embedded images, and other objects
- written in the form of labels (known as tags), surrounded by angle brackets
- can also describe, to some degree, the appearance and semantics of a document
- can include embedded scripting language code which can affect the behavior of web browsers and other HTML processors

- You can, of course, use a WYSIWYG (What You See Is What You Get) HTML editor to make websites but they have 3 main disadvantages:
  - They sometimes use excess code to create a look on a page which slows down loading times
  - They do not always create fully compatible code
  - Some WYSIWYG editors change any HTML code you enter by hand

# HTML tags

- <tag>
- There are two types of tag. Opening and closing tags. Closing tags are only different as they have a / before them:
- </tag>
- Nearly all tags have a closing tag but a few do not.

# HTML tags

```
<html>
 <head>
 <title>Untitled</title>
 </head>

 <body>
 ...
 </body>
</html>
```

# HTML tags

- <html>
  - beginning of an HTML document
- <head>
  - beginning of the header section - contains configuration options for the page
- <title>Untitled</title>
  - tells the browser what to display as title of page
  - appears in the title bar at the top of the browser
- </head>
  - end of the header section

# HTML tags

- <body>
  - Everything between these is in the body of the page
  - This is where all text, images etc. are.
  - This is the most important part of the page.
- </body>
- </html>
  - end of the HTML document

# <font> tag

- display the text in a standard font
- tag attributes
- <font face="Arial" size="7" color="#FF0000">text</font>

# <p> tag

- stands for Paragraph; used to break up text into paragraphs
- <p> tag has attribute which can be added to it - align option
- <p align="right / left / center ">Text</p>
- can use the <center> tag or the <p align="center">
- hardly ever necessary to use the align="left" attribute as nearly all browsers automatically align text to the left

# <br> tag

- want to leave a space after your paragraphs
- should use the <Br> (break) tag
- there is no end tag
- <br> = <BR> = <Br> = <bR>
- Tags are case insensitive

<!-- Your comment -->

# <img> tag

- Images are added to pages
- must use the src attribute to choose the image to insert
- can either be a relative reference or a direct reference
- can resize images inside the browser using two other image attributes: width and height
- Alt tells the browser what the alternative text for an image should be if the browser has images turned off

# <a> tag

- used when creating hyperlinks and bookmarks; stands for anchor
- need to use the href attribute
- to make a piece of text or an image into a hyperlink you contain it in:

```
 Link
```

# Hyperlinks

- Web Page or Site <a href="http://www.webaddress.com/folder/page">
- Local Page <a href="pagename.html">
- Local Page In A Folder Level Below <a href="foldername/pagename.html">
- Local Page In A Folder Level Above <a href="../pagename.html">
- Open E-mail Program With E-mail Addressed <a href="mailto:yourname@yourname.com">
- Bookmarked Section <a href="#bookmarkname">
- Bookmarked Section In Another Page <a href="pagelocation.htm#bookmarkname">

# Bookmarks

- instead of changing the href variable use the name variable. For example:
- <a name="top"></a>
- will create a bookmark called top in the text
- can then link to this using a standard hyperlink:
- <a href="#top">Back To Top</a>

# PHP syntax

- parses code within its delimiters
  - <?php ... ?>
- allow statements to be embedded within HTML documents
- print & echo functions
  - practically identical

```
<html>
 <head> <title></title>
 </head>
 <body>
 <?php
echo 'Hello, World!';
 ?>
 </body>
</html>
```

# PHP syntax

- variables are prefixed with a dollar symbol \$
  - type does not need to be specified in advance
- unlike function and class names, variable names are case sensitive
- treats new lines as whitespace
- statements are terminated by a semicolon
- three types of comment syntax:
  - /\* \*/ which serves as block comments
  - // as well as # which is used for inline comments

# Data types

- Integer
- Double
- Boolean
- String
- Object
- Array
- Null
- Resource

# Data types

- numbers in a platform-dependent range - range typically 32-bit signed integers
- Integer variables can be assigned using decimal (positive and negative), octal and hexadecimal notations
- Real numbers can be specified using floating point notation, or two forms of scientific notation
- native Boolean type
  - Using Boolean type conversion rules, non-zero values can be interpreted as true and zero as false

# Data types

- Null data type represents a variable that has no value
  - only value in the Null data type is NULL
- Arrays support both numeric and string indices, and are heterogeneous
- Arrays can contain elements of any type that PHP can handle, including resources, objects, and even other arrays

# Source code encoders, optimizers and accelerators

- PHP scripts are normally kept as human-readable source code, even on production web servers
  - While this allows flexibility, it can raise issues with security and performance
- Encoders hinder source code reverse engineering
  - hide source code
  - compile code into "opcode"
  - downside of this latter approach is that a special extension has to be installed on the server in order to run encoded scripts,
  - however the approach of encoding compiled code and use of an extension offers typically the best performance, security and opportunity for additional features that may be useful for developers

# Source code encoders, optimizers and accelerators

- Code optimizers improve the quality of the compiled code by reducing its size and making changes that can reduce the execution time and improve performance. The nature of the PHP compiler is such that there are often many opportunities for code optimization.
- Accelerators offer performance gains by caching the compiled form of a PHP script in shared memory to avoid the overhead of parsing and compiling the code every time the script runs. They may also perform code optimization to provide increased execution performance
- most commonly used packages for source code protection are from Zend Technologies

# Testing PHP script

- write file
- upload it to your Web Server
- using your browser, go to the URL of the script
- *C:\...path from root of drive...\script.php*
- *http://localhost\...path from root of WebServer...\script.php*

# Outputting Variables

```
<?php
$welcome_text = "Hello and welcome to my
website.;"
print($welcome_text);
?>
```

# Formatting Text

```
print("<font face=\"FontFace\"
size=\"FontSize\" color=\"FontColor\">Hello
and welcome to my website.");
```

# IF Structure

```
IF (something == something else)
{
 THEN Statement
} else {
 ELSE Statement
}
```

# WHILE, DO WHILE

while (expr)  
    statement

do statement  
while (expr)

# FOR

for (expr1; expr2; expr3)  
statement

# Arrays

- `$array[key] = value;`
- should always use quotes around a string literal array index
  - For example, use `$foo['bar']` and not `$foo[bar]`
- do not quote keys which are constants or variables, as this will prevent PHP from interpreting them
- It works because PHP automatically converts a bare string (unquoted string which does not correspond to any known symbol) into string
- PHP may in future define constants which, unfortunately for your code, have the same name

# Arrays

```
exArr[1] = "one";
exArr[2] = 2;
exArr[1] = "trei";
```

```
band["Brian"] = "May";
band["Roger"] = "Taylor";
band["John"] = "Deacon";
```

```
echo "The fourth member of
the band is $band[3];"
```

# User-defined functions

```
function name($arg1, $arg2, ..., $argn)
{
 echo "Example function.\n";
 return $retval;
}
```

# Functions

- All functions and classes in PHP have the global scope - they can be called outside a function even if they were defined inside and vice versa
- PHP does not support function overloading
- Function and classes names are case-insensitive
- support variable numbers of arguments to functions, default arguments
- It is possible to call recursive functions
  - avoid recursive function/method calls with over 100-200 recursion levels as it can smash the stack and cause a termination of the current script

# File include

- When a file is included, the code it contains inherits the variable scope of the line on which the include occurs. Any variables available at that line in the calling file will be available within the called file, from that point forward. However, all functions and classes defined in the included file have the global scope.

# Basic include() example

- GeneralFunctions.php

```
function DataBaseConnect ($arg1, $arg2, ...,
 $argn)
{
}
}
```

- test.php

```
include 'GeneralFunctions.php';
DataBaseConnect ();
```

# Setting Up Forms

- A form is an area that can contain form elements.
- Form elements are elements that allow the user to enter information in a form
- A form is defined with the `<form> </form>` tags

```
<form action="process.php" method="post">
```

# Text Fields

- when you want the user to type letters, numbers, etc. in a form

```
<form>
```

First name:

```
<input type="text" name="firstname">
```

```


```

Last name:

```
<input type="text" name="lastname">
```

```
</form>
```

- in most browsers, the width of the text field is 20 characters by default

# Radio Buttons

- used when you want the user to select one of a limited number of choices

```
<form>
```

```
 <input type="radio" name="sex" value="male">
 Male
```

```


```

```
 <input type="radio" name="sex" value="female">
 Female
```

```
</form>
```

- Note that only one option can be chosen

# Checkboxes

- used when you want the user to select one or more options of a limited number of choices

```
<form>
```

I have a bike:

```
<input type="checkbox" name="vehicle" value="Bike" />
```

```


```

I have a car:

```
<input type="checkbox" name="vehicle" value="Car" />
```

```


```

I have an airplane:

```
<input type="checkbox" name="vehicle" value="Airplane" />
```

```
</form>
```

# Drop Down list

```
<form>
<select name="cars">
<option value="volvo">Volvo</option>
<option value="saab">Saab</option>
<option value="fiat"
 selected="selected">Fiat</option>
<option value="audi">Audi</option>
</select>
</form>
```

# Textarea

- multi-line text input control; user can write text (an unlimited number of characters) in the text-area

```
<textarea rows="10" cols="30">
```

Initial text to be displayed in text-area.

```
</textarea>
```

# Submit button

```
<input type="submit" value="Submit">
<input type="button" value="Hello world!">
<input type="reset" value="Reset">
<input type="hidden" value="Hidden">
```

# Form

```
<form action="process.php" method="post">
...
<input name = ...>
<select name = ...>
<input type="submit" value = ...">
<input type="button" value = ...>
...
</form>
```

In process.php?param1=val1&

```
$variablename=$_POST['variable'];
```

```
$variablename=$_GET['variable'];
```

- basically takes the variable from the POST or GET method (the name of a form field) and assigns it to the variable

```
$variablename
```

```
$variablename=$_GET['param1'];
```

# Predefined Variables

- Server variables: `$_SERVER`
- Environment variables: `$_ENV`
- HTTP Cookies: `$_COOKIE`
- HTTP File upload variables: `$_FILES`
- Request variables: `$_REQUEST`
- Session variables: `$_SESSION`

# Predefined Functions

- IBM DB2 Universal Database
- MS SQL Server database
- MySQL database
- Oracle database
- PostgreSQL
- ODBC access several databases that have borrowed the semantics of the ODBC API
  - unified into a single set of ODBC functions

# Setting Up The Database

- See Database course ...

# Inserting Information

- And also Updating, Deleting ...
- See Database course ...
  - SQL part

# Displaying Data

```
bool mssql_connect(
 [string $servername
 [, string $username
 [, string $password
 [, bool $new_link]]])

bool mssql_select_db (string
 $database_name
 [, resource $link_identifier])
```

# Connecting Database

```
$conn = mssql_connect('MYSQLSERVER', 'sa',
 'password');
mssql_select_db('[my data-base]', $conn);
```

# Displaying Data from contacts

- Select all the records in the contacts table
- There will be output from this command it must be executed with the results being assigned to a variable:

```
$query="SELECT * FROM contacts";
$result=mssql_query($query);
```

- In this case the whole contents of the database's table is now contained in a special array with the name \$result.
- Before you can output this data you must change each piece into a separate variable

# Query database

```
mixed mssql_query (string $query
[, resource $link_identifier
[, int $batch_size]])
```

# Counting Rows

```
$num=mssql_numrows($result);
```

```
int mssql_num_rows (resource $result)
```

- will set the value of \$num to be the number of rows stored in \$result (the output you got from the database)
  - can then be used in a loop to get all the data and output

# Setting Up The Loop

- \$i is the number of times the loop has run and is used to make sure the loop stops at the end of the results so there are no errors

```
$i=0;
while ($i < $num) {
```

CODE

```
$i++;
}
```

- For can be used also

# Assigning Data To Variables

```
string mssql_result (resource $result, int
$row, mixed $field)
```

- field can be the field's offset, the field's name or the field's table dot field's name (tablename.fieldname)

```
$variable=mysql_result($result,$i,"fieldname
");
```

- Recommended high-performance alternatives:
  - `mssql_fetch_row()`
  - `mssql_fetch_array()`
  - `mssql_fetch_object()`

# Displaying Data from contacts

```
$first=mssql_result($result,$i,"first");
$last=mssql_result($result,$i,"last");
$phone=mssql_result($result,$i,"phone");
$mobile=mssql_result($result,$i,"mobile");
$fax=mssql_result($result,$i,"fax");
$email=mssql_result($result,$i,"email");
$web=mssql_result($result,$i,"web");
```

# Displaying Data from contacts

```
mssql_connect('localhost','username','password');
@mssql_select_db($database) or die("Unable to select
database");

$query="SELECT * FROM contacts";
$result=mysql_query($query);

$num=mysql_numrows($result);

mysql_close();
```

# Combining The Script

```
echo "<center>Database
Output</center>

";
```

```
$i=0;
while ($i < $num) {

$first=mysql_result($result,$i,"first");
...
$web=mysql_result($result,$i,"web");
```

# Format the output

```
echo "$first $last
Phone: $phone
Mobile:
$mobile
Fax: $fax
E-mail: $email
Web:
$web
<hr>
";
```

```
$i++;
```

```
}
```

# **Thank you for your kindly attention!**