A3
Introduction to Relational Databases and SQL ®

Version 4.0

Sela College

© 2013 Sela college All rights reserved.

All other trademarks are the property of their respective owners.

This course material has been prepared by:

Sela Software Labs Ltd.

14-18 Baruch Hirsch St. Bnei Brak 51202 Israel

Tel: 972-3- 6176666 Fax: 972-3- 6176667

Copyright: © Sela Software Labs Ltd.

All Materials contained in this book were prepared by Sela Software Labs Ltd. All rights of this book are reserved solely for Sela Software Labs Ltd. The book is intended for personal, noncommercial use. All materials published in this book are protected by copyright, and owned or controlled by Sela Software Labs Ltd, or the party credited as the provider of the Content. You may not modify, publish, transmit, participate in the transfer or sale of, reproduce, create new works from, distribute, perform, store on any magnetic device, display, or in any way exploit, any of the content in whole or in part. You may not alter or remove any trademark, copyright or other notice from copies of the content. You may not use the material in this book for the purpose of training of any kind, internal or for customers, without beforehand written approval of Sela Software Labs Ltd.

The Use of this book

The material in this book is designed to assist the student during the course. It does not include all of the information that will be referred to during the course and should not be regarded as a replacement for reference manuals.

Limits of Responsibility

Sela Software Labs Ltd invests significant effort in updating this book, however, Sela Software Labs Ltd is not responsible for any errors or material which may not meet specific requirements. The user alone is responsible for decisions based on the information contained in this book.

Protected Trademarks

In this book, protected trademarks appear that are under copyright. All rights to the trademarks in this material are reserved to the authors.

SELA wishes you success in the course!

Table of Contents

Module 01 - What is a database

Chapter Goals 3

Why are Databases Necessary? 4

Why are Databases Necessary? (contd.) 5

Why Are Databases Necessary ? (contd.) 6

The Purpose of a Database Management System 7

What is a Database Management? 8

The Database Management System Solution 9

Client/ Server Architecture 10

Log Files 11

The Logical Model and the Physical Model 12

The Logical Model 13

In-Class Exercise - Terminology 14

In the Following Chapters We Will: 15

Module 02 - The Relational Model

Chapter Goals 3
Set Theory 4
Domains 5
Relational Algebra 6
Operations in Relational Algebra 7
In-Class Exercise - Terminology 8
Projection (π) 9
Selection (σ) 10
Rename (ρ) 11
Class Exercise – Projection, Selection and Rename 12
Class Exercise – Projection, Selection and Rename (contd.) 13
Set Operations - Union 15
Set Operations - Intersection 16
Set Operations - Complement 17
Class Exercise – Set Operations 18
Cartesian Product 19
Condition Join 20
Equation Join – Equijoin 21
Natural Join 22
Class Exercise – Joins 23
Chapter Summary 24

Module 03 - Data Modeling

In This Chapter We Will: 6

The System Development Cycle 7

Step 1 - System Analysis 8

Step 1 - System Analysis (contd.) 9

Step 2- System Design 10

Step 3 - Data Modeling 11

Step 3 - Data Modeling (contd.) 12

Step 4 - Coding and Testing the System 13

Step 5 - Installation and Maintenance 14

Interim Summary 15

The "Universe of Discourse" - UOD 16

The Data Model - ENTITY RELATION Diagram (ERD) 17

What is an ENTITY? 18

ENTITY vs. Entity 19

Examples of ENTITIES and Their PROPERTIES: 20

Different Types of ENTITIES 21

Different Types of ENTITIES (contd.) 22

What is a PROPERTY? 25

What is a PROPERTY? (contd.) 26

More About PROPERTIES 27

Class Exercise - Find the ENTITY 28

RELATION Between ENTITIES 29

RELATION Between ENTITIES (contd.) 30

Class Exercise: Simple ENTITIES and RELATIONS 33

Class Exercise: Simple ENTITIES and RELATIONS (contd.) 34

RELATIONS 35

RELATION Level 36

RELATION Level (contd.) 37

Example 1- In the Theater 38

Example 2 - In the Restaurant *39*
Ternary RELATION, and Binary RELATIONS - Between 3 ENTITIES *40*
RELATION Level - Ternary *41*
Class Exercise - Is This a Ternary RELATION? *42*
Unary RELATION *43*
RELATION Functionality *44*
RELATION Functionality (contd.) *45*
1:N One to Many *47*
1:N or N:M *48*
N:M Many to Many *49*
N:M Many to Many (contd.) *50*
RELATION Functionality – Remarks *51*
Class Exercise: RELATION Functionality *52*
RELATION Cardinality *53*
RELATION Cardinality: Example *54*
*Level * Functionality * Cardinality* *55*
*Level * Cardinality, Examples* *56*
*Level * Cardinality, Examples (contd.)* *57*
*Class Exercise - Level * Cardinality* *58*
Some Important Dilemmas and Issues *59*
Some Important Dilemmas and Issues (contd.) *60*
From Universe of Discourse (UOD) to ENTITY RELATION Diagram (ERD) *62*
From Universe of Discourse (UOD) to ENTITY RELATION Diagram (ERD) (contd.) *63*
Check Your ERD - CODD Rules *64*
Check Your ERD - CODD Rules (contd.) *65*
E.R.D. – Class Examples *66*
Class Example 1: "See This" *67*
Solution - Class Example 1: "See This" *68*
Class Example 2: "Tell-vision" *69*
Solution - Class Example 2: "Tell -Vision" *70*
Class Example 3: DeepLawMass" *71*

Solution Class Example 3: "DeepLawMass" *72*

Class Example 4: "Joe's Movies" (Ternary connection between entities) *73*

Solution Class Example 4: "Joe's Movies" *74*

Class Example 5: "Banana Republic" *75*

Class Example 5: "Banana Republic" (contd.) *76*

Solution - Class Example 5: "Banana Republic" *77*

Lab: E.R.D. – Class Exercises *78*

Class Exercise: "Fishy Beauty" *79*

Class Exercise : "Fishy Beauty" (contd.) *80*

Class Exercise : The "EUROVISION" *81*

Class Exercise : The "EUROVISION" (contd.) *82*

Class Exercise : "Hotel California" (Time Period as an Entity) *83*

Class Exercise : "Dr. Do-Little" *84*

Class Exercise :"Talk To Me" *85*

Lab: E.R.D. – Class Exercise Solutions *86*

Class Exercise : "Fishy Beauty" Solution *87*

Class Exercise : "EUROVISION" Solution *88*

Class Exercises : "Hotel California" Solution *89*

Class Exercises : "Dr. Do-little" Solution *90*

Class Exercise : "Talk To Me" Solution *91*

Final Exercises – UOD, ERD – Exercise 1 ..... *92*

Final Exercises – UOD, ERD – Exercise 1 (contd.) *93*

Final Exercises – UOD, ERD – Exercise 1 (contd.) 1 ..... *00*

Final Exercises – UOD, ERD – Exercise 2 1 ..... *02*

Final Exercises – UOD, ERD – Exercise 2 (contd.) 1 ..... *03*

Module 04 - Converting the ERD to a Table Design

In This Chapter We Will 5
Translating the ERD to Tables and Columns 6
What is a Table? 7
The Process of Converting the ERD Into a Table Design 8
Converting ENTITIES Into Tables 9
Tables 10
Tables (contd.) 11
Class Exercise 01 12
Converting Relations 13
Binary 1:N 14
Binary 1:N (Implementation) 15
Class Exercise 02 16
Binary N:M 17
Binary N:M (contd.) 18
Class Exercise 03 22
Binary 1:1 23
Binary 1:1 (Implementation) Option A 24
Binary 1:1 (Implementation) Option B 25
Class Exercise 04 26
Ternary (or more) N:M 27
Ternary (or more) N:M (contd.) 28
Ternary (or more) N:M – cont'd 29
Class Exercise 05 - Ternary 30
Unary 1:N 31
Class Exercise 06 - Unary 1:N 32
Unary 1:1 33
Class Exercise 07 - Unary 1:1 34
From ERD to Tables - Summary 35
From ERD to Tables - Summary (contd.) 36

From ERD to Tables - Remarks *37*

From ERD to Tables - Remarks (contd.) *38*

Summery *40*

Transition to the Table Model Class Examples *41*

Class Example 1: "See This" *42*

Solution - Class Example 1: "See This" *43*

Transition to the Table Model 1: "See This" *44*

Class Example 2: "Tell-Vision" *45*

Class Example 2: "Tell-Vision" (contd.) *46*

E.R.D. - Class Example 2: "Tell-Vision" *47*

Transition to the Table Model 2: "Tell-Vision" *48*

Class Example 3: DeepLawMass" *49*

E.R.D. Class Example 3: DeepLawMass" *50*

Transition to the Table Model 3: "DeepLawMass" *51*

Class Example 4: "Joe's Movies" *52*

E.R.D. Class Example 4: "Joe's Movies" *53*

Transition to the Table Model 4: "Joe's Movies" *54*

Class Example 5 : "Banana Republic" *55*

Class Example 5 : "Banana Republic" (contd.) *56*

Transition to the Table Model 5: "Banana Republic" *58*

Lab: Transition to the Table Model Class Exercises *59*

Class Exercise 1: "Fishy Beauty" *60*

Class Exercise 1: "Fishy Beauty" (contd.) *61*

Class Exercise - "Fishy Beauty" ERD1 ..... *62*

Class Exercise 2: The "EUROVISION" *63*

Class Exercise - "EUROVISION" ERD 2 ..... *64*

Class Exercise 3: "Hotel California" *65*

Class Exercise – "Hotel California" ERD 3 ..... *66*

Class Exercise 4: "Dr. Do-little" *67*

Class Exercise - "Dr. Do-little" ERD 4 ..... *68*

Class Exercise 5: "Talk To Me" *69*

Lab: Transition to the Table Model: Class Exercise Solutions *71*

Class Exercise 1 "Fishy Beauty" Solution: *72*

Class Exercise 2 "EUROVISION" Solution: *73*

Class Exercise 3 "Hotel California" Solution: *74*

Class Exercise 4 "Dr. Do-little" Solution: *75*

Class Exercise 5 "Talk To Me" Solution: *76*

Module 05 - The Select Query

Chapter Goals 6

What is SQL? 7

SQL's Groups of Statements 8

SQL Language Elements 9

Database (DB) Management Environment 10

Class Example: Northwind - UOD 11

Class Example: Northwind – ERD 12

Class Example: NorthWind - Table Design (schema) 13

Select - The Logical Idea 14

Select - The Logical Idea (contd.) 15

The Select Command 16

The Select Statement 17

The Select Statement (contd.) 18

The Select Statement 19

Select - Caption 20

From 21

From (contd.) 22

Sql Server Functions 23

Common String Functions 24

Common Date & Time Functions 26

Common Mathematical Functions 29

The Where Clause 30

The Where Clause (contd.) 31

Where, And, Or 32

Order By 33

Class Example - Select Statement 34

Class Example - Select Statement (contd.) 35

Class Exercise - Select 36

Join Tables 37

Join Tables (contd.) *38*
Join Table - Examples *39*
How Do You Know Which Tables You Need? *40*
How Do You Know Which Tables You Need? (Example) *41*
Class Exercise - Join Tables *42*
Aggregation Query *43*
Aggregation Query (contd.) *44*
Aggregation Function Without Group By *46*
Group By <Column Name> *47*
Group By - Example *48*
Group By – Example (contd.) *49*
Aggregation - Having *50*
Aggregation - Having (contd.) *51*
Where and Having *52*
Aggregation - Remarks *53*
Class Examples - Aggregation *54*
More Class Examples - Aggregation *55*
More Class Examples – Aggregation (contd.) *56*
Class Exercise - Aggregation *58*
Outer Join *59*
An Outer Join *60*
Outer Join - Class Exercise *61*
A Sub Query *62*
A Sub Query (contd.) *63*
A Sub Query in Where – Option 1 ..... *65*
A Sub Query In Where – Option 2 ..... *66*
A Sub Query In Where – Option 2 (contd.) *67*
A Sub Query In a Select Statement *68*
A Sub Query In a Select Statement (contd.) *69*
Class Examples - Sub Query *70*
More Class Examples - Sub Query *71*

Exists *72*
Class Exercise - Sub Query *73*
Using Views *74*
Views in a Query - Syntax *75*
Views in a Query *76*
Class Example - View *77*
More Class Examples - View *78*
Class Exercise - Views *79*
Set Operators *80*
UNION and UNION ALL *81*
Except *83*
Intersect *84*
Class Exercise – Set Operators *85*
How to Resolve a Query - A Check List *86*
How to Resolve a Query - A Check List (contd.) *87*
Chapter Summary *88*
Class Exercise - Select Query *89*
Class Exercise - Select Query (contd.) *90*
Final Exercises – Select Query – Exercise 1 ..... *92*
Final Exercises – Select Query – Exercise 1 (contd.) *93*
Final Exercises – Select Query – Exercise 2 ..... *99*
Final Exercises – Select Query – Exercise 2 (contd.) 1 ..... *00*

Module 06 - Update, Insert, Delete

Chapter Goals *3*

Action Queries *4*

Insert New Rows *5*

The Insert Query - Remarks *6*

The Insert Query - Class Exercise *7*

The Update Query *8*

The Update Query - Class Exercise *9*

The Delete Query *10*

The Delete Query - Class Exercise *11*

Chapter Summary *12*

Module 07 - Data Definition Language

Chapter Goals 4

Data Definition Language (DDL) 5

Data Definition Language (DDL) (contd.) 6

Create Table <Table> 7

Data Types 8

Constraints 9

Primary Key (PK) 10

The Primary Key - Syntax 11

The Primary Key- Syntax (contd.) 12

The Foreign Key (FK) 13

The FK - Syntax 14

The FK – Syntax (contd.) 15

The Constraints - Not Null, Unique 16

The Constraints - Check 17

Create Table - Miscellaneous 18

Drop Table 19

Class Exercise - Create Table 20

Indexes 21

Indexes (contd.) 23

Index - Syntax 24

Create <view> 25

View 26

View - Remarks 27

View – Remarks (contd.) 28

Class Exercise - View 29

Alter <table> 30

Stored Procedures 31

Stored Procedures (contd.) 32

Alter <Stored Procedure> 33

Stored Procedures and Security *34*
Class Exercise – Stored Procedures *35*
User-defined Functions *36*
User-defined Functions (contd.) *37*
Class Exercise – User-defined Functions *38*
Difference Between SP and UDF *39*
Database Normalization Process *40*
Normal Forms *41*
Final Exercise – DDL and DML *42*
Final Exercise – DDL and DML (contd.) *43*
Final Exercise – UOD, ERD, Table Design, DDL, DML and Queries *45*
Final Exercise – UOD, ERD, Table Design, DDL, DML and Queries (contd.) *46*
Final Exercise *47*
Chapter Summary *48*

Module 01 - What is a database

Contents:

Chapter Goals 3
Why are Databases Necessary? 4
Why are Databases Necessary? (contd.) 5
Why Are Databases Necessary ? (contd.) 6
The Purpose of a Database Management System 7
What is a Database Management? 8
The Database Management System Solution 9
Client/ Server Architecture 10
Log Files 11
The Logical Model and the Physical Model 12
The Logical Model 13
In-Class Exercise - Terminology 14
In the Following Chapters We Will: 15

SELA

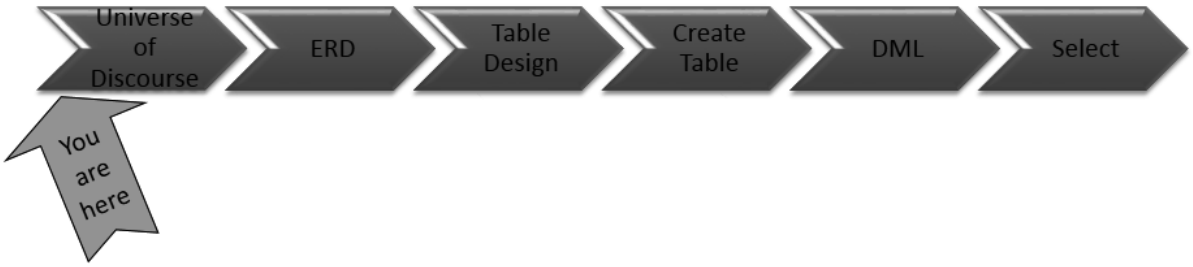
Introduction to Relational Databases and SQL

Module 01 - What is a Database?

© Copyright SELA software & Education Labs Ltd. | 14-18 Baruch Hirsch St Bnei Brak, 51202 Israel | www.selagroup.com

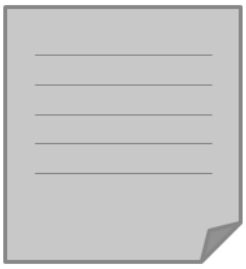
Chapter Goals

- Databases.
- The database engine.
- The user interface.
- The logical model and the physical model of the database.



Why are Databases Necessary?

- An organization requires software to manage its data.
- How is data managed?
  - Through the use of binary or ASCII files + a programming language.



If you ask a manager in an organization why do you need a database, probably you'll get different answers, but many know that database enables the management of data (it is implied from their name). In these days, databases are the backbone of almost every application. They enable the persistence of data during the application life cycle and to query that data later on. But to understand databases, we first need to understand the players.

What is an organization? Organizations include: banks; government institutions; schools; shops; even regular households.
The organizations are the main player in the database world. They need databases in order to hold and manage their precious business data.
Earlier, the data was stored in files which made the query process a very hard thing to do.
These days, databases are also file based, but they manage their data in a relational form, that enables a quick access to query data.
The databases are managing their data in binary or ASCII files. For example, C along with binary files, or Visual Basic and ASCII files.

Why are Databases Necessary? (contd.)

- What operations need to be performed on data?
  - Insert new data, update existing data.
  - Data needs to be displayed and queried.
    - Examples: sorting data or filtering data.
  - It is useful to print reports based on the data
- Managerial perspective
  - Data mining, ERP, CRM etc.

When you ask yourself what you can do with data a few things pop up. The first thing is to store it for later use. But there are other operations that we can do with data for example enter details for a new student, update product prices. These CUD (Create, Update, Delete) operations are data manipulation operations. You can also query the data and shape, of which the results you need for the organization's use.

For example: A list of student names sorted in alphabetical order or a list of all products whose stock levels are below 100 units.

Data is also useful for reports. For example:

- A report containing a customer list sorted alphabetically by customer name.
- A report containing a list of students' course, credit points and grades.

We also have the organization management perspective for the organization data. Managerial perspective is divided into strategic management which creates and changes the business (data mining), tactical management which manages the business (ERP) and operational management which runs the business (CRM).

Why Are Databases Necessary ? (contd.)

- What is the difference between data and information?
  - Data which is stored in a database is raw data.
    - Examples: date of birth, discount rate, course grade.
  - Information is, what is derived from the data.
    - Examples: a person's current age, the amount of a discount (in dollars), a student's grade point average.
  - Databases store data, software uses the data to calculate and derive useful information.

When we talk about data, we need to understand the difference between data and information.
Data – is raw data that nobody has yet processed. It can be a lot of things like employee name, number of car's doors or whatever you can think of, it's raw data.
Information – is processed data. We can understand a lot of different aspects or things from it. For example, we can calculate the average monthly salary of an employee. The monthly salary will be stored in a database and later we would be able to query that data and retrieve the average, which is the information that we need.

Databases are meant to hold data. Applications which use databases are the user interface that show the information (or processed data) to their users.

The Purpose of a Database Management System

- To handle a variety of file-related problems:
  - Over-handling of files.
  - Multi-user problems.
  - Each part of an organization creates its own "data files".
  - Performance problems:
    - Highly experienced programmers are required to create an efficient, large, multi-user file system.
- The solution - one large, centralized database engine.

After we understood why it is necessary to have databases, the next thing we need to understand is what is the purpose of a database management system or in short DBMS. Since in earlier times, the data was saved in files, let us understand the problems with files as a data source.

The first problem is over-handling of files. The problem of having a lot of files is that you find yourself managing files and not data.

Also, when you need a relevant data you will have to make a lot of effort to find it.

The second problem is multi-users. Multi-user problems occur when more than one user wants to update a file, a user deletes other user's file or other concurrent situations. What do you do in such cases? The answer to such problems can be very difficult.

The third problem is knowledge management inside the organization. In this problem, different teams manage the same data in different files which belong to each team. The problems that occur when each part of an organization creates its own "data files":

- Wasted programming
- Data duplication (each department has its own employee list).
- Data not updated (each department keeps a different version of the employee list, data is not shared amongst departments).

Having a very big file system is very hard to manage and will cause a very poor performance when we need to retrieve specific data or to manipulate data.

All these problems are answered by the database management system.

What is a Database Management?

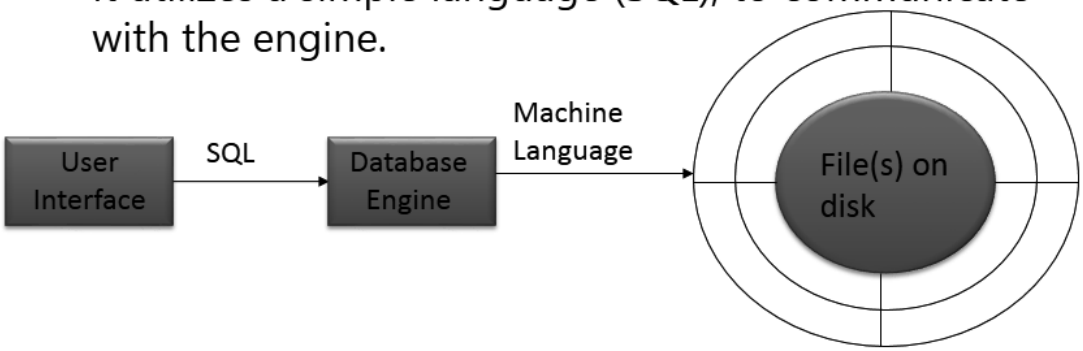
- Analyzing the context.
- Designing a data model.
- Data modeling – **Build the database**.
- **Administer the database** – Coding and testing the system.
- Facilitate the database
  - Retrieval services
  - Analysis services
  - Update services
  - Integration services

Database management is divided into five different parts:

- Analyzing the context – analyze the needs of the organization and the business of the organization in order to create a suitable database for it.
- Designing a data model – design the relational model based on the analysis of the first phase.
- Data modeling – the actual building of the database according to the data model, which includes transforming of the model into database tables and relations.
- Administer the database – Coding and testing the system.
- Facilitate the database – the actual part that enables the use of the database, which includes the insertion of data inside the database and the real management of that data (queries, analysis, data manipulation and system integration).

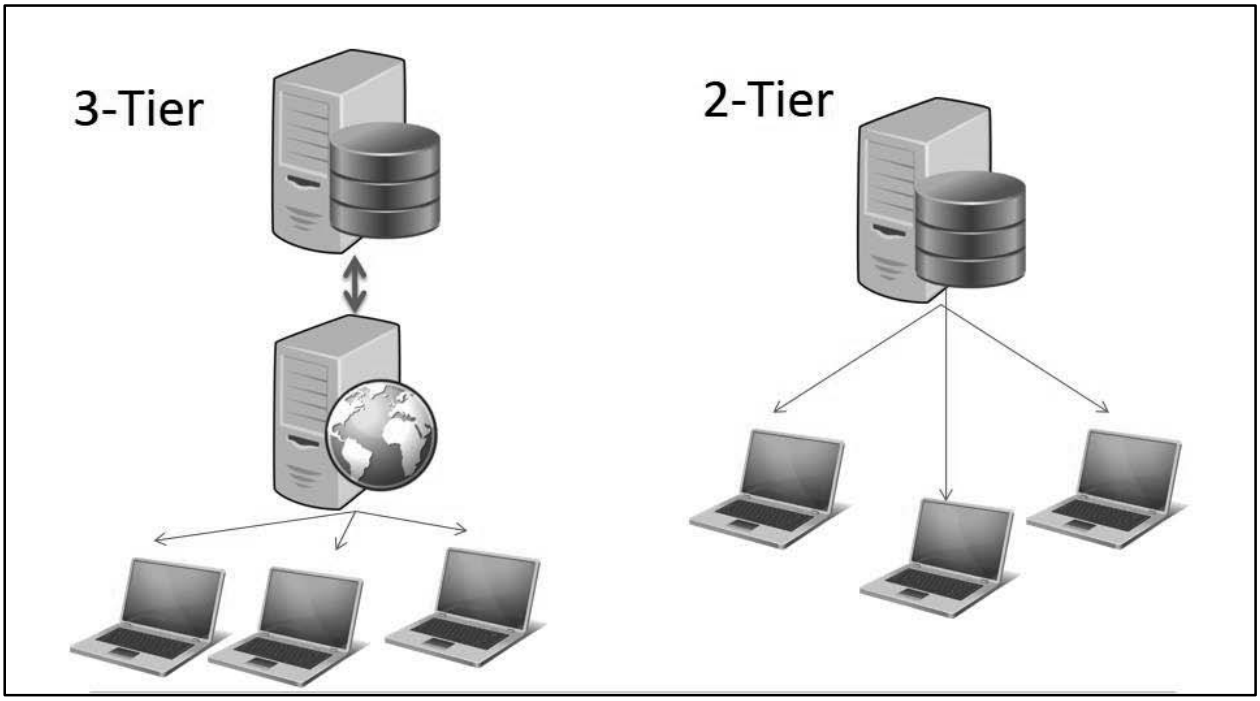
The Database Management System Solution

- The database engine:
  - The engine is a running application.
  - It performs file management for the organization (difficult work).
  - It utilizes a simple language (SQL), to communicate with the engine.



The database is an application that includes a database engine which is always running (unless a database crash happens). It uses the relational model for saving data which will be discussed later in the course. It uses a very simple but powerful SQL (Structured Query Language) language that enables the manipulation or querying of the stored data. There are a lot of vendors that sell databases. The famous databases are Oracle, SQL Server, DB2 and more.

Client/ Server Architecture



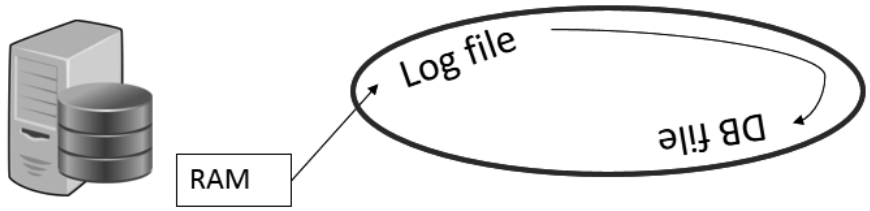
A database serves multiple clients:

- Multiple clients may update the data simultaneously.
- A data locking system prevents conflicts.

- The database can handle multiple clients.
  The clients connect to the database through a database connection which will be opened by the client.
  The open connection will enable the clients to perform data operations simultaneously.
- Data operations can be used to create, update and delete data.
  In order to prevent data conflicts and to keep data integrity, the database uses a data locking system when CUD operations are performed.
  Since there is a locking mechanism, deadlocks (a situation that two or more competing actions are waiting for the other to finish) can occur.
  Clients can query shaped data by their demand.
- Queried data is retrieved by the clients for further manipulation.

Log Files

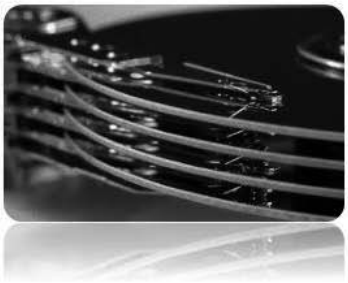
- When working with files:
  - The file is open in memory.
- When working with a database engine:
  - The engine passes data from memory to disk.
  - Updates are stored in the log file.
  - Allows for recovery in the case of a system crash.
  - Allows for control of the data transfer time.



When we work on a file, the file is open in memory and all the writing is made to memory until a forced data flush occurs (which flush the data to the disk) or the file is closed (when the writing end).
If the file is deleted somehow or an error occurs, the non flushed data is lost.
Databases are divided into two main files – a database file and a log file.
All the updates are stored in a log file and stored there until an explicit request is made to further store the data to the database file.
The log files enable system recovery plan when the database crashes.
You only need to go till the last backup and load it, and then use the log file to retrieve all the operations that occurred between the backup and the database crash.
You, as the database administrator, can control when the transfer time from the log file to the database file should occur.

The Logical Model and the Physical Model

- The Physical Model:
  - What is really happening on the disk?
    - Files, Addresses, Pointers....



- The Logical Model:
  - What the engine "shows" the user.
  - An abstract representation of the data.
  - In understandable language:
    - Tables, columns, etc.
  - How is this all applied? In the following chapters, we will explain.

Student
ID
Name
Address
Tel Num
Birth date

Example

When we talk about applications we talk about two aspects: the hardware (computer, wires, etc) and the software (applications, files, etc).
The database software is itself divided into two models. The physical model is what really happens on the disk. It includes the files of the database and all the stuff that is used to enable the running of the database server. The logical model of the database is how we represent the data inside the database, which is based on the relational model. That model is created from tables and relations between them and will be taught during the course.

The Logical Model

- There are several logical models (hierarchical, net, relational).
- Each model has its advantages and disadvantages (each is used for specific purposes).
- One of the most common - The relational model - will be taught in this course.

Who mentioned advantages?

The relational model is best known as, “the model with the tables”.

In-Class Exercise - Terminology

- In class, provide a brief explanation of the following terms:
  - Table
  - Engine
  - Client
  - Relational model
  - Physical vs. logical models
  - Columns
  - Server
  - Log file
  - SQL
  - Database action (insert, update...)

Explain in one sentence for etch term.

In the Following Chapters We Will:

- Learn about the database engine in more detail
- Examine the relational model
- Learn how to use the relational model to solve data management problems

Questions

Module 02 - The Relational Model

Contents:

Chapter Goals 3
Set Theory 4
Domains 5
Relational Algebra 6
Operations in Relational Algebra 7
In-Class Exercise - Terminology 8
Projection (π) 9
Selection (σ) 10
Rename (ρ) 11
Class Exercise – Projection, Selection and Rename 12
Class Exercise – Projection, Selection and Rename (contd.) 13
Class Exercise – Projection, Selection and Rename (contd.) 14
Set Operations - Union 15
Set Operations - Intersection 16
Set Operations - Complement 17
Class Exercise – Set Operations 18
Cartesian Product 19
Condition Join 20
Equation Join – Equijoin 21
Natural Join 22
Class Exercise – Joins 23
Chapter Summary 24

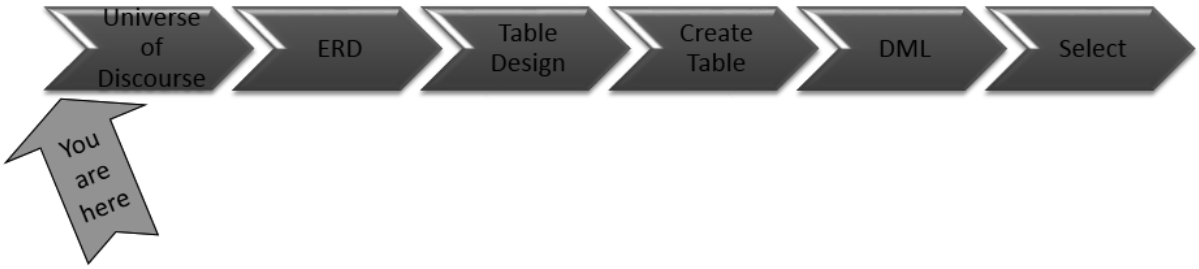
Introduction to Relational Databases and SQL

Module 02 - The Relational Model

© Copyright SELA software & Education Labs Ltd. | 14-18 Baruch Hirsch St Bnei Brak, 51202 Israel | www.selagroup.com

Chapter Goals

- In this chapter we will learn about:
  - Set theory
  - Relational Algebra



Set Theory

- Set theory is all about sets.
- There are two kind of sets:
  - Finite sets.
    Example:
    cars = { Volvo, Fiat, Mazda }
  - Infinite sets.
    Example:
    N = { 0, 1, 2, 3, ... }
- Sets are a collections of objects under two rules:
  - Duplication of set members is prohibited.
  - The order of set members is insignificant.

Set theory is a mathematical theory that deals with sets. The definition for a set is a collection of distinct objects which can be finite or infinite.
Finite sets are sets that end sometime. For example, software company set is a finite set or animal type set is finite.
Infinite sets are never ending sets like the natural number set or the real number set.
In order to be called a set in the set theory the collection of object need to answer two rules.
First rule, duplication can occur in the set which means that we have distinct objects included in one set.
The second rule is that, the order inside a set is insignificant, which means that we can mess with the set's members and that won't have an impact on the set itself.
Terminology:
Set
Finite set
Infinite set
Domain

Domains

- A domain is a set whose members:
  - Have the same data type
  - Have the same semantics
- Examples:
  months = { January, February, … December }
  computer games = { Doom, Dune, Warcraft … }

You should be familiar with the definition of what is a domain and nothing else in regard for it. Domains in set theory are sets under the following rules:

- The objects inside the set have the same data type.
- The objects inside the set have the same semantics.

There are a lot of domain example such as: numbers, flowers, companies and so on.

Relational Algebra

- The algebra of sets
- Deals with a set of operators that are designed to manipulate relational tables
- The operators can be joined to form a well defined expression

The relational algebra is the algebra of sets. It is a part of computer science and it started gaining popularity during 1970s. This algebra deals with sets and the operations that we can use to manipulate them. The sets are represented as relational tables which have relation between them and from that, comes the name of relational algebra. The operations that can be used can be composed and define expressions that can be used to manipulate the relational tables.

Operations in Relational Algebra

- Projection (π) – projects only the columns we need and eliminates the others.
- Selection (σ) – selects a subset of rows.
- Rename (ρ) – renames a column to a new name.
- Set operations.
  - Union (∪) – the set of all distinct elements in a collection
  - Intersection (∩) – the set of all the element that belongs in all the given collections
  - Complement (\) - unique values of one set in relation to another.
- Joins operations.

In relational algebra there are six fundamental operations which are:

1. Projection – projects only the columns we need in the given table and eliminates the others.
2. Selection – selects the subset of rows in the given table.
3. Rename – changes the column name to a new name.
4. Union – returns a new set that is composed from the two given sets, only with the distinct elements from the two given sets.
5. Intersection – returns a new set that is composed from the elements that are shared between the two given sets.
6. Join – creates a new table that is composed from two different tables. There are some kinds of join operations that we will discuss later in this chapter.

As you can see, the compliment isn't one of the six fundamental operations. Even then, we will learn about the operation and see how it operates.

In-Class Exercise - Terminology

- In class, provide a brief explanation of the following terms:
  - Relational algebra
  - Projection.
  - Selection.
  - Rename.
  - Union.
  - Intersection.
  - Complement.
  - Join Operations.

Projection (π)

- Projects only the columns we need and eliminates the others.
- Syntax:
  $\pi_{a_1,\ldots,a_n}(R)$
- Definition:
  Returns a table whose schema and content consist of a subset of R's columns
- Example:
  $\pi_{GameName}$(Games) on Games table

GameID	GameName
1	Doom
2	Dune

will project

GameName
Doom
Dune

By using projection we can select the columns that we want to return.
Explain the syntax: $\pi_{a_1,\ldots,a_n}(R)$

A1..An are the columns that we want to return from table R.

Selection (σ)

- Selects a subset of rows.
- Syntax:
  $\sigma_{\varphi}(R)$
- Definition:
  Returns a table containing the rows in R that satisfy ϕ
- Example:
  $\sigma_{GameID>1}(Games)$ on Games table

GameID	GameName
1	Doom
2	Dune

will selec

GameID	GameName
2	Dune

This selection selects all those records in R for which ϕ holds.

Rename (ρ)

- Renames a column to a new name.
- Syntax: $\rho_{a\,/\,b}(R)$
- Definition:
  Creates a copy of R but the B column is renamed to A
- Example:
  $\rho_{GameName\,/\,Name}(Games)$ on Games table

GameID	GameName
1	Doom
2	Dune

will create a new table

GameID	Name
1	Doom
2	Dune

Class Exercise – Projection, Selection and Rename

- Given the Products table:

ProductID	ProductName	Price
1	Table	3000
2	Chair	800
3	Bed	10000

- Write a projection statement that projects the product's name and price. Provide the result table.
- Write a selection that returns products with price higher than 1000. Provide the result table.
- Write a rename that returns a new table with ProductID renamed to ID and ProductName renamed to Name. Provide the result table.

Answers:

- $\Pi_{ProductName,\ Price}$ (Products)
- $\sigma_{price>1000}$ (Products)
- $\rho_{ProductName\ /Name}$ ($\rho_{ProductID\ /\ ID}$ (Products))

Class Exercise – Projection, Selection and Rename (contd.)

- Given the Students table:

StudentID	StudentName	StudentCountry	StudentYear
1	Lisa	USA	1
2	John	Germany	2
3	Mike	Germany	1

- Write a projection statement that projects a student's name and student year. Provide the result table.
- Write a selection that returns, students with student year bigger than 1, and country isn't Germany. Provide the result table.

Answers:

- $\Pi_{StudentName,\ StudentYear}$(Students)
- $\sigma_{studentYear>1\ \wedge\ StudentCountry<>"\ Germany"}$(Students)

Class Exercise – Projection, Selection and Rename (contd.)

- Write a rename that returns a new table with StudentID renamed to ID and StudentCountry renamed to Country. Provide the result table.

$\rho_{StudentID\ /ID}(\rho_{StudentCountry\ /\ Country}(\text{Products}))$

Set Operations - Union

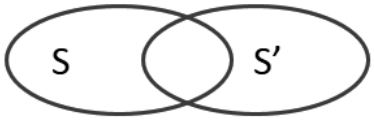
- Defined as the union of two sets into one set
- Syntax:
  S ∪ S'

- Definition:
  The elements in S or in S' or in both.
- Examples:
  { a } ∪ { b} = { a, b }
  { a, b, c, d } ∪ { a, e, f, c } = { a, b, c, d, e, f }

In the drawing all shapes are the union of S and S'.
All the elements in both sets.

Set Operations - Intersection

- Defined as the intersection of two sets into one set



- Syntax:
  S ∩ S'
- Definition:
  The elements in S that are also in S'.
- Examples:
  { a } ∩ { b} = { }
  { a, b, c, d } ∩ { a, e, f, c } = { a, c }

In the drawing, the dark part in S and S’ shows the intersection.
The elements that belong to s and s’.

Set Operations - Complement

- Defined as the union of two sets into one set
- Syntax:
  S \ S'

- Definition:
  The elements in S that are not in S'.
- Examples:
  { a } \ { b} = { a }
  { a, b, c, d } \ { a, e, f, c } = { b, d }

In the drawing the bright side of S is the complement.

Class Exercise – Set Operations

- What are the results of the following equations:
  - { a, b } ∪ { c, d }
  - { a, b, c } ∪ { d, e, f } ∪ { a, f }
  - { a, b } ∪ { b, c }
  - { a, b } ∩ { c, d }
  - { a, b } ∩ { a, c, d}
  - { a, b, c, f } ∩ { c, f, g, h } ∩ { a, c, h }
  - { a, b } \ { b, c }
  - { a, b } \ { a, b }
  - { a, b, c, d } \ { b, c, e, f }

Cartesian Product

- Cartesian Product, also called "cross join"
- Syntax:
  A X B where A and B are tables
- Definition:
  Contains all the pair combinations from A and B
- Example:

a
a1
a2

x	y
x1	y1
x2	y2

a	x	y
a1	x1	y1
a1	x2	y2
a2	x1	y1
a2	x2	y2

Combining each record in A with each record in B.

Condition Join

- Condition Join, also called "theta join"
- Syntax:
  A X_{φ} B where A and B are tables
- Definition:
  Contains all the pair combinations from A and B and eliminates the rows that don't satisfy ϕ
- Example:

a
6
2

$X_{a>x}$

x	y
3	5
5	6

=

a	x	y
6	3	5
6	5	6

Combining each record in A with each record in B which ϕ holds.

Equation Join – Equijoin

- A condition join with equality.
- Syntax:
  A $X_{A.a=B.a}$ B where A and B are tables
- Definition:
  Contains all the pair combinations from A and B and eliminates the rows that don't satisfy ϕ
- Example:

a
6
2

$X_{A.a=B.a}$

a	y
3	5
6	6

=

A.a	B.a	y
6	6	6

Natural Join

- Resembles Equijoin but the extra equal column is removed.
- Syntax:
  A X B where A and B are tables
- Definition:
  Equijoin but without extra equal column
- Example:

This is the most commonly used join in databases when we have foreign keys relations between tables.
- For using natural join we must have a column\s with the same name in both A and B tables.

Class Exercise – Joins

- Given the two tables:

a	b
6	3
5	4
8	5

x	y	a
4	5	6
9	4	5
5	3	3

- What is the cross join of the tables?
- What is the theta join of the table if the condition is a>y
- What is the Equijoin and natural join of the tables?

Chapter Summary

- We learnt about the parts that create the relational model:
  - Set theory
  - Relational Algebra
- These are the basics that will accompany us in the world of databases.

Questions

Module 03 - Data Modeling

Contents:

In This Chapter We Will: 6
The System Development Cycle 7
Step 1 - System Analysis 8
Step 1 - System Analysis (contd.) 9
Step 2- System Design 10
Step 3 - Data Modeling 11
Step 3 - Data Modeling (contd.) 12
Step 4 - Coding and Testing the System 13
Step 5 - Installation and Maintenance 14
Interim Summary 15
The "Universe of Discourse" - UOD 16
The Data Model - ENTITY RELATION Diagram (ERD) 17
What is an ENTITY? 18
ENTITY vs. Entity 19
Examples of ENTITIES and Their PROPERTIES: 20
Different Types of ENTITIES 21
Different Types of ENTITIES (contd.) 22
Different Types of ENTITIES (contd.) 23
Different Types of ENTITIES (contd.) 24
What is a PROPERTY? 25
What is a PROPERTY? (contd.) 26
More About PROPERTIES 27
Class Exercise - Find the ENTITY 28
RELATION Between ENTITIES 29
RELATION Between ENTITIES (contd.) 30

RELATION Between ENTITIES (contd.) 31
RELATION Between ENTITIES (contd.) 32
Class Exercise: Simple ENTITIES and RELATIONS 33
Class Exercise: Simple ENTITIES and RELATIONS (contd.) 34
RELATIONS 35
RELATION Level 36
RELATION Level (contd.) 37
Example 1- In the Theater 38
Example 2 - In the Restaurant 39
Ternary RELATION, and Binary RELATIONS - Between 3 ENTITIES 40
RELATION Level - Ternary 41
Class Exercise - Is This a Ternary RELATION? 42
Unary RELATION 43
RELATION Functionality 44
RELATION Functionality (contd.) 45
RELATION Functionality (contd.) 46
1:N One to Many 47
1:N or N:M 48
N:M Many to Many 49
N:M Many to Many (contd.) 50
RELATION Functionality – Remarks 51
Class Exercise: RELATION Functionality 52
RELATION Cardinality 53
RELATION Cardinality: Example 54
Level * Functionality * Cardinality 55
Level * Cardinality, Examples 56
Level * Cardinality, Examples (contd.) 57
Class Exercise - Level * Cardinality 58
Some Important Dilemmas and Issues 59
Some Important Dilemmas and Issues (contd.) 60
Some Important Dilemmas and Issues (contd.) 61

From Universe of Discourse (UOD) to ENTITY RELATION Diagram (ERD) 62
From Universe of Discourse (UOD) to ENTITY RELATION Diagram (ERD) (contd.) 63
Check Your ERD - CODD Rules 64
Check Your ERD - CODD Rules (contd.) 65
E.R.D. – Class Examples 66
Class Example 1: “See This” 67
Solution - Class Example 1: “See This” 68
Class Example 2: “Tell-vision” 69
Solution - Class Example 2: “Tell -Vision” 70
Class Example 3: DeepLawMass” 71
Solution Class Example 3: “DeepLawMass” 72
Class Example 4: “Joe’s Movies” (Ternary connection between entities) 73
Solution Class Example 4: “Joe’s Movies” 74
Class Example 5: “Banana Republic” 75
Class Example 5: “Banana Republic” (contd.) 76
Solution - Class Example 5: “Banana Republic” 77
Lab: E.R.D. – Class Exercises 78
Class Exercise: “Fishy Beauty” 79
Class Exercise : “Fishy Beauty” (contd.) 80
Class Exercise : The “EUROVISION” 81
Class Exercise : The “EUROVISION” (contd.) 82
Class Exercise : “Hotel California” (Time Period as an Entity) 83
Class Exercise : “Dr. Do-Little” 84
Class Exercise :“Talk To Me” 85
Lab: E.R.D. – Class Exercise Solutions 86
Class Exercise : “Fishy Beauty” Solution 87
Class Exercise : “EUROVISION” Solution 88
Class Exercises : “Hotel California” Solution 89
Class Exercises : “Dr. Do-little” Solution 90
Class Exercise : “Talk To Me” Solution 91
Final Exercises – UOD, ERD – Exercise 1 92

Final Exercises – UOD, ERD – Exercise 1 (contd.) 93
Final Exercises – UOD, ERD – Exercise 1 (contd.) 94
Final Exercises – UOD, ERD – Exercise 1 (contd.) 95
Final Exercises – UOD, ERD – Exercise 1 (contd.) 96
Final Exercises – UOD, ERD – Exercise 1 (contd.) 97
Final Exercises – UOD, ERD – Exercise 1 (contd.) 98
Final Exercises – UOD, ERD – Exercise 1 (contd.) 99
Final Exercises – UOD, ERD – Exercise 1 (contd.) 100
Final Exercises – UOD, ERD – Exercise 1 (contd.) 101
Final Exercises – UOD, ERD – Exercise 2 102
Final Exercises – UOD, ERD – Exercise 2 (contd.) 103
Final Exercises – UOD, ERD – Exercise 2 (contd.) 104
Final Exercises – UOD, ERD – Exercise 2 (contd.) 105
Final Exercises – UOD, ERD – Exercise 2 (contd.) 106
Final Exercises – UOD, ERD – Exercise 2 (contd.) 107
Final Exercises – UOD, ERD – Exercise 2 (contd.) 108
Final Exercises – UOD, ERD – Exercise 2 (contd.) 109
Final Exercises – UOD, ERD – Exercise 2 (contd.) 110
Final Exercises – UOD, ERD – Exercise 2 (contd.) 111

SELA

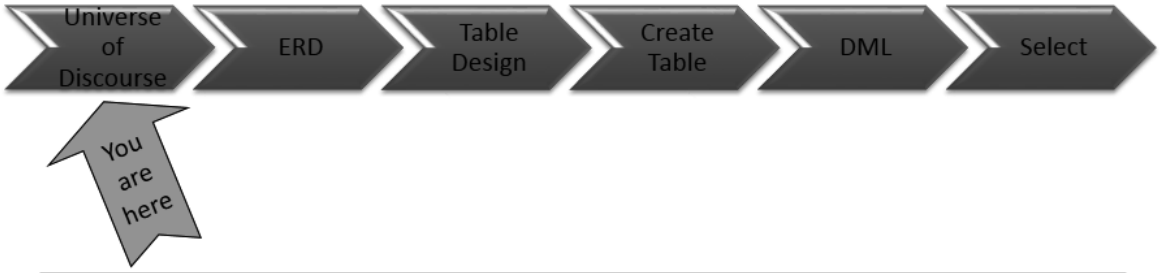
Introduction to Relational Databases and SQL

Module 03 - Data Modeling

© Copyright SELA software & Education Labs Ltd. | 14-18 Baruch Hirsch St Bnei Brak, 51202 Israel | www.selagroup.com

In This Chapter We Will:

- Study the system development cycle
- Study the organization's "universe of discourse".
- Learn to build an ENTITY RELATION DIAGRAM (ERD).
- Check the ERD based on normalization rules.
- Discuss the relational model (table model).



The System Development Cycle

- Step 1 - System analysis
- Step 2 - System design
- Step 3 - Data modeling
- Step 4 - Coding and testing the system
- Step 5 - Installation and maintenance.

In every system's creation, there are some cycles of development that can be divided into five steps:
analysis, design, modeling, coding, maintenance.
In the following slides we will deal with every step in more details.

Step 1 - System Analysis

- Your client heads an organization.
- In order to learn how this organization works, you:
  - Meet the client, interview the client, write a report.
- What data does the organization store, and how is it stored?
  - Lists on paper - customer lists, cards - all customer orders, typed documents - salary slips).
- What procedures are performed within the organization?
  - New employee absorption, recording of orders, inventory counts.

System analysis deals with analysis of sets of interacting entities and is a very crucial part in the development cycle.
The details that you get from the clients will affect the way you design your solution.
If you won't analyze all the client needs, you will end up with a system that the client doesn't want or need.

This step includes gathering the client needs, learning how the client's organization works, understanding the client's data, learning the client's problems and more.
All the analysis will be conducted by meeting the client and gathering details from him/her or other persons from the organization as much as you can. After you gather the information you'll have to analyze it and from it you'll have your first hint about how to design the system (or in our case the database). Since no one can gather all the information and since organizations change in time the analysis should be done also during the other cycles in order to find "holes" that we didn't detected in the first place.

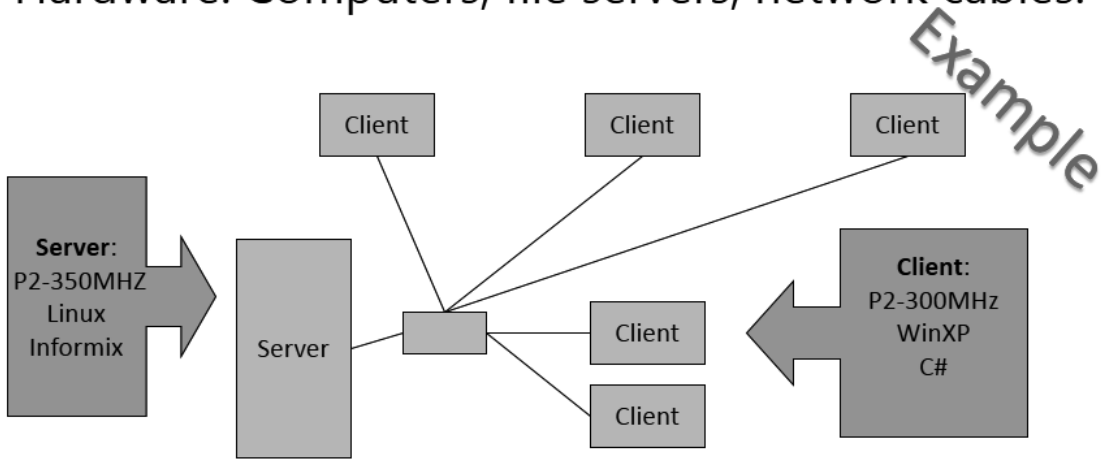
Step 1 - System Analysis (contd.)

- What reports does the organization require?
  - Lists of orders sorted by date, lists of departments and their employees.
- Learn about problems related to the current system.
  - It is impossible to maintain an updated list of customers. Hiring a new employee entails too much paper work.

Reporting is a very important module in most of the organization's systems, so we need to understand what kind of report the organization requires.
Understand the problems that the organization face today, for designing a better solution that solve those problems.

Step 2- System Design

- Hardware: Computers, file servers, network cables.



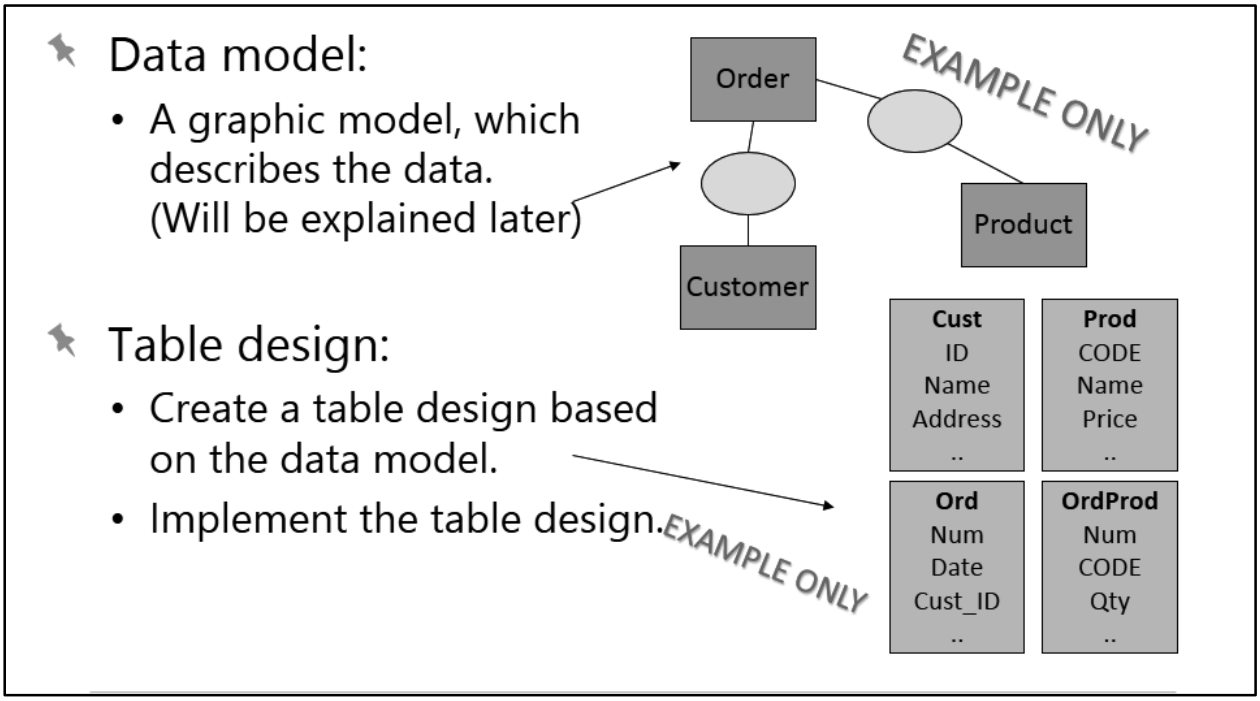
- Software: An operating system, network software, a database engine, user interface language.

System design is the process of shaping the system according to the requirements and the analysis that we have done in the first step in terms of architecture, components, modules, etc. The system design is made up of two parts:

- Hardware – the physical parts of the system which can be servers, network cables and more.
- Software – the components that construct the system with which the user interacts, such as operating system, database engines, GUI application and etc.

When we design the system in terms of hardware and software we should pay attention to the client's constraints and requirements. After finishing the design, a process of design review and also client review is necessary in order to get more opinions about the design and finding missing parts.

Step 3 - Data Modeling



Data modeling is used to define and analyze data requirements that a business process will need to support.

We first model the data in an ERD (Entity Relation Diagram) representation which is a graphical model that describes the data in terms of entities, properties and relations. Then, when we have an ERD, we transform its representation into tables and relations inside a database in a process that is called table design. After we have the table design in hand, we implement it inside the database engine that the company has, such as SQL Server, Oracle and other database engines.

Step 3 - Data Modeling (contd.)

- The data model is a representation.
- It represents the data, the organization requires, for its daily work.
- It allows better communication in regard of the organization's data.
- It is the “cornerstone” of all future development:
  - An organization is dynamic. Changes are frequent.
  - Databases are revised as well.
  - A strong (i.e., generic, simple, future-oriented, etc.) data model permits efficient responses to change.

The data model cannot be computerized since it is a representational form of data.
It allows the system analyzer to communicate better with the client and the programmer regarding the organization's data.
Building a good data model helps to create better applications that run in the long run and can change to meet the organization’s changing requirements. The data model is probably one of the most important parts in the creation of applications.

Step 4 - Coding and Testing the System

- Create the database using SQL.
- Build a client/server architecture.
- Create the graphic user interface which contains
  - User friendly forms.
  - Reports.
- Perform system tests.

In this course, we will talk in more details about the Structured Query Language (SQL) which is a standard for creating and manipulating databases. Using SQL, we can create the database from the data model and the table design that we made in the previous steps.
After the creation of the database, we can use it to query and manipulate data.
We use the database through GUI applications, that are built according to the organizational requirements. These applications are the gateway for manipulating data which can also occur by database management tools like SQL Server Management Studio (for SQL Server), Toad and SQL Developer (for oracle) and many more products.
Develop the application (GUI \ Servers \ Web Applications).
After we create the application, we need to perform system tests in order to detect bugs or unimplemented requirements.

Step 5 - Installation and Maintenance

- Install the system within the client's organization.
- Teach the organization's employees to work with the new system.
- Maintain the system and resolve problems that may occur.

This is a critical stage in the development cycle since maintenance is considered as the major cycle of creating a system (there are numbers that show that out of 100% of the developing time, 30% will be spent to create the system and 70% to maintain it).
The installation is a stage that deals with the assimilation of the system in the client's organization, and also includes the teaching of the system's functioning patterns to the employees. After we install the system and it is in production mode, we try to maintain the system and to solve bugs and other problems that can and will occur.

Interim Summary

- The development cycle contains 5 steps:
  - System analysis;
  - System design;
  - Data modeling;
  - Coding and system building;
  - Testing, installation and maintenance.
- Class discussion:
  - Your favorite restaurant wants to create a database system. What will be the 5 developing steps, for creating that system?
- This chapter will focus on the third step – data modeling.

We learnt the five steps of the development cycle.
In the last few years, there is another development process which has gathered more attention than the others; which is called Agile. Agile will not be discussed in this course, but if you like to read about what it is, check the following links:
http://en.wikipedia.org/wiki/Agile_software_development
http://agilemanifesto.org/

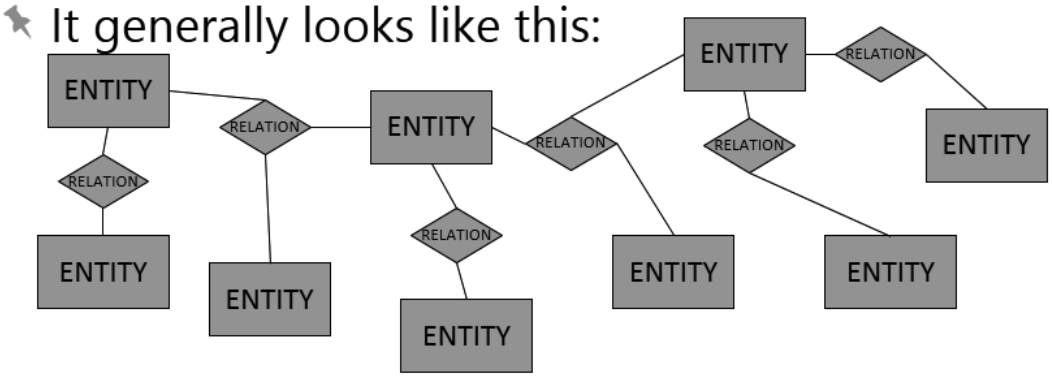
The "Universe of Discourse" - UOD

- Is a component of system design.
- Is a set of statements, which describe the organization's data.
  - The UOD for the restaurant "looks" like this:
    - A new order is opened for each customer.
    - An order contains dishes. A dish is from a dish type.
    - Each dish type contains many ingredients.
    - Dish type is from a specific category (salad, meat, dessert, etc.).
    - And more....
- Is translated into a data model.

The "Universe of Discourse" (UOD) is also called the "domain of discourse".
It is a component that holds the system design inside it, in terms of statements. We build a set of statements according to the organization's data that describes the data, and from it we can understand the domain problem of the organization.
From these statements we will be able to create an ERD (Entity Relation Diagram), which is the data model that will provide a graphical implementation of the "Universe of Discourse". The ERD will be taught later on. We talked about databases and we learnt that database represents a model for some aspects of an organization. These aspects are the "universe of discourse" or "domain of discourse".

The Data Model -
ENTITY RELATION Diagram (ERD)

- Is comprised (surprisingly), of ENTITIES and RELATIONS.
- PROPERTIES are not displayed in the ERD.
- Constraints (will be discussed later).
- It generally looks like this:



ERD (ENTITY RELATION Diagram) is a diagram that is built from entities and their relations. From ERD's, we can create a database, which will be explained in the next chapter. The ERD is constructed from ENTITIES and RELATIONS.

We use the upper case notation because we describe the blueprint of a single Entity or Relation through ENTITY and RELATION. Every ENTITY has PROPERTIES which is a characteristic of some piece of data. The PROPERTIES aren't displayed in the ERD, but you should always remember that they exist. Also, ERD holds data constraints which aren't part of the general look that we see in the slide, but will be discussed early on.

What is an ENTITY?

- An ENTITY is:
  - An autonomous object within the organization.
  - An entity has properties.
  - Data, which must be stored by the organization.
  - An ENTITY (as well as other components), will become a part of a table in the relational model.

ENTITIES are blueprint of data structures that hold data inside their PROPERTIES. They don't have behavior which in software is sometimes called functionality. From an ENTITY, we would be able to create a database table in the future (in the next chapter). Every ENTITY is an autonomous object within the organization. We use the UOD to understand how to create an ENTITY, and then later on we create a database table from that representation. In terms of object oriented development, an ENTITY could be compared to a class which holds the sketch of how an object will look and behave.

ENTITY vs. Entity

- ENTITY == Entity Type:
  - "Client" is an ENTITY (== an Entity type).
  - "Mr. Brown" is an Entity (an instance) of the ENTITY, “Client”.
  - Additional examples of ENTITIES: Flight, Product, Airplane, Car, Country, Person, Student.

ENTITY is a blueprint of an Entity. It defines how an instance of ENTITY will look like. When we create an instance of an ENTITY it will be referred as Entity. In programming, the same correlation exists between classes (ENTITIES) and objects (Entities). Every Entity has a type which is the ENTITY. Other examples for the example in the slide:

- “Ford” is an Entity of the ENTITY “Car”
- “F16” is an Entity of the ENTITY “FighterJet”
- “Doom” is an Entity of the ENTITY “ComputerGame”
- And so on.

Examples of ENTITIES and Their PROPERTIES:

- A client is an ENTITY. A client's PROPERTIES include: first and last name, address, telephone number, etc.
- A product is an ENTITY. A product's PROPERTIES include: name, code, weight, price, etc.
- An order is an ENTITY. An order's PROPERTIES include: order number, order date, delivery date, total price, etc.
- A course is an ENTITY. A course's PROPERTIES include: start date, duration, credit points, recommended books, pre-requisites.

Every ENTITY has PROPERTIES that describes what data can it hold. The PROPERTIES must have a data type which in terms of ERD aren't relevant, since its part of the model implementation. Every ENTITY must have one (not zero) or more PROPERTIES. An ENTITY without PROPERTIES can't be called an ENTITY because it means that it doesn't hold data inside and therefore doesn't imply to one of the ENTITY definitions. There are many more examples of PROPERTIES for an ENTITY other then what is written in the slide:

- A house is an ENTITY. A house's PROPERTIES include: number of doors, number of windows, number of walls, etc.
- A computer game is an ENTITY. A computer game's PROPERTIES include: number of stages, game name, game cost, game type, etc.
- And more.

Different Types of ENTITIES

- There are several types of ENTITIES. Here is a method for distinguishing between them:
  - A Physical (Real) ENTITY - Represents something real, something that exists in the world, a concrete object, such as:
    - A customer, a product, an airplane, a student, a copy of a book.
    - Class exercise: Name 10 physical ENTITIES and their PROPERTIES.
  - An Event ENTITY - an ENTITY can be an event. An event has a time and/or a date:
    - A flight, a party, an order, a meeting, a lesson.
    - Class Exercise: Name 10 event ENTITIES and their PROPERTIES.

There are five types of ENTITIES:

- A Physical ENTITY – which describes something real that we can really understand.
- An Event ENTITY - which describes an event that has meaning in terms of time or date.
- An abstract ENTITY – which describes an abstract object in the world. For example, a person is an abstract ENTITY. From it we can built representations like students and professors (in regard to school), customers and employees (in terms of a company) and more.
- A categorical ENTITY – which describes an ENTITY that is above the physical ENTITY, which describes (and stores), some of the PROPERTIES of the physical ENTITY. This is an ENTITY that expands the details of another ENTITY.
- An ENTITY that was 'born' in the normalization process – normalization process will be discussed later.

Different Types of ENTITIES (contd.)

- An abstract ENTITY - An abstract object in the world.
  - Example1: A Bus line - this ENTITY will store data regarding the average trip duration, bus stops along the line, schedules, etc. (The bus is the physical ENTITY. Each trip (on that line) is an event ENTITY.)
  - Example2: A Course - "Introduction to biology", stores duration, recommended books, credit points.

Remember, an ENTITY will be a table, an Entity will be a row in that table

Different Types of ENTITIES (contd.)

- (An event ENTITY would be a course cycle (it has specific dates and participants)
- Class Exercise: Name 10 abstract ENTITIES and their PROPERTIES.

Different Types of ENTITIES (contd.)

- A categorical ENTITY -
  is an ENTITY above the physical ENTITY which describes (and stores), some of the PROPERTIES of the physical ENTITY.

  Somewhat like an abstract ENTITY

  - Example: A book - the physical ENTITY is a copy of the book, the book's ENTITY stores the name and the price. The copy's ENTITY stores data regarding lending and damage to the specific copy.
  - Class Exercise: Name 10 categorical ENTITIES and their PROPERTIES.
- An ENTITY that was 'born' in the normalization process (will be discussed later).

Another Example for a categorical ENTITY:
Product type - the physical ENTITY is a product. The type stores general data regarding the product such as its actual price, whereas the product stores data regarding manufacture date, serial number.

What is a PROPERTY?

- There are several types of PROPERTIES:
  - Atomic properties (which cannot be separated):
    - Data items, which cannot be parsed or calculated.
    - Example: ID number, last name, passport number, product price.
  - Compound properties (which can be separated):
    - Each part of the PROPERTY will still have meaning, but...
    - The whole PROPERTY has more meaning than its part .
    - Examples: Date of birth, Flight number (LY444), address.
  - Calculated or Derived Properties (generally not stored in a database).
    - Example: Total price, age, student's grade point average.

PROPERTIES define a data holder inside an ENTITY. Every Property will have a type which will determine the type of data it holds. There are three PROPERTY types which are:

- Atomic properties – properties that we can't calculate or parse since calculation won't mean a thing. Try to calculate a first name… what does it mean?
- Compound properties – a property that gets its meaning from its ingredients. Every ingredient has a meaning but when we compound them, we get a more meaningful representation of the data. For example, an address is compound from street, house number, zip code and more.
- Calculated or derived properties – properties that are created from calculation process or other means. They are usually a part of the business logic and not a part of the database. Even then we need to understand that when we query a database for data, we can get the ingredients that we need in order to make the calculation. In the next lessons, we will see aggregate functions that will help us to do the calculations on the stored data.

What is a PROPERTY? (contd.)

- PROPERTY == Property Type.
  - "Last Name" is a property type or PROPERTY.
  - "Pitt" is a Property or an instance of the PROPERTY.

More About PROPERTIES

- A Unique PROPERTY - a PROPERTY that identifies each Entity (instance) of the ENTITY.
  - Examples: passport number, order number, employee ID, serial number, barcode, license plate number, country code.
- Optional vs. Mandatory PROPERTY:
  - Mandatory (must be recorded for each instance):
    - Examples: employee name, car chasse number, product serial number, user ID.
  - Optional (need not be recorded for every instance):
    - Examples: product description, employee's maiden name, a customer's mobile telephone number.
- Class Exercise: name 10 unique, mandatory and optional PROPERTIES.

There are more things about PROPERTIES then just data holders. We can describe PROPERTIES behavior, as unique PROPERTY and optional/mandatory PROPERTIES. These kind of PROPERTIES help us to shape the relevant data inside the ENTITIES. A unique PROPERTY is a property that identifies an Entity. Optional PROPERTY is a property that isn't relevant for the ENTITY but it might be needed sometimes. Mandatory PROPERTY is the opposite of optional PROPERTY and you must have the piece of data that it holds which is meaningful for the ENTITY.

Class Exercise - Find the ENTITY

- Find the ENTITY in each of the following sentences:
- What type of ENTITY is it?
- For each ENTITY, write its PROPERTIES.
- For each PROPERTY, what type of PROPERTY is it?
  - An employee works on several projects.
  - A flight departs at a specific date and hour.
  - The company produces several product types.
  - A student takes several courses.
  - The doctor receives many patients.
  - Each event takes place in a different court.
  - Each diagnosis can be treated with several medications.
  - Cooks specialize in several types of dishes.
  - A waiter serves a dish on a table at the restaurant.

RELATION Between ENTITIES

- A RELATION describes the relation between one or more ENTITIES.
  - Employee, to work in a department - John works in the sales department, Jane for finance.
  - Car, to be of Car Type - The car with license plate 67-309-01 is a Porsche, that red car, with the broken window, is a Subaru.
  - A student, to enroll in a course - George enrolled in a biology class, Hanna enrolled in math.

As was mention earlier, ERD is composed of ENTITIES and RELATIONS. Since we talked about ENTITIES now the RELATIONS time has arrived.
RELATION describe the way ENTITIES are connected to each other. The RELATION between ENTITIES shows how they relate. There can be zero to many RELATION between ENTITIES which will compose the ecosystem of the ERD. RELATION describes the actual type of relation.

RELATION Between ENTITIES (contd.)

- RELATION==Relation type.
  - "to enroll" is the RELATION.
  - The relation (instance) of this case is the fact that George enrolled in Biology (and failed the exam...☹).

RELATION Between ENTITIES (contd.)

- A RELATION is drawn in the ERD in the following manner:

- Information is derived from ENTITIES and RELATIONS.
  - Examples:
    - We can obtain information about employees and the departments in which they work.
    - We can check the number of airplanes the company has of each airplane type.
    - We can obtain information regarding the courses a student enrolled in and the grades he received.

RELATIONS in ERD are described as a circle which hold the RELATION phase and lines that connect the RELATION to the relevant ENTITIES. We can use the RELATION to understand the connection between ENTITIES and of course, they relate to each other. This added information is very useful to understand the ecosystem inside the ERD. A RELATION will be transformed into foreign keys in the database.

RELATION Between ENTITIES (contd.)

- A RELATION can have PROPERTIES that emerge from the RELATION:
  - Entities: Student, Course; RELATION: to enroll; RELATION PROPERTIES: year she enrolled in the course and grades she received.
  - Entities: Dish, Ingredient; RELATION: to be made of; RELATION PROPERTIES: quantity.
  - Entities: Actor, Movie; RELATION: to act in; RELATION PROPERTIES: salary he received for his participation.
- A RELATION can be between more than 2 ENTITIES (to be discussed later).
- A RELATION can be between an ENTITY and itself (to be discussed later).

Like ENTITIES, a RELATION can also have PROPERTIES. These PROPERTIES will add more information about the RELATION with regard to the ENTITIES that the RELATION is connected to. A RELATION can be between more than 2 ENTITIES and also between an ENTITY and itself.

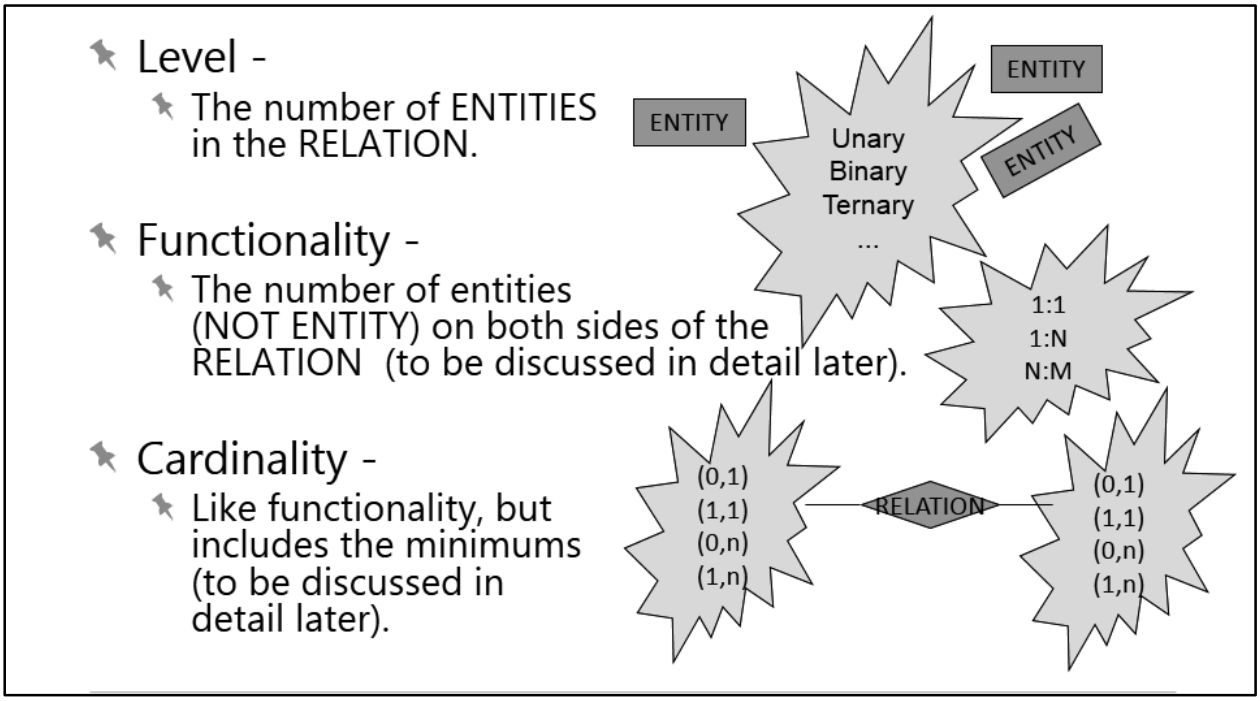
Class Exercise: Simple ENTITIES and RELATIONS

- Find the ENTITIES in the following sentences.
- What type of ENTITY is it?
- Find the RELATIONS in the sentences.
- For each ENTITY and RELATION write its PROPERTIES.
  - In each country there are many cities.
  - A client places several orders each year.
  - A car type has many part types.
  - The naval ships have a home port.
  - A meeting is between many employees.

Class Exercise: Simple ENTITIES and RELATIONS (contd.)

- Many authors wrote the book.
- A book has many copies.
- "Pictures in the Exhibition".

RELATIONS



RELATIONS have some definitions that they must hold in order to understand them. Such definitions are the level, functionality and cardinality of a RELATION. The level is the number of ENTITIES that relate through a given RELATION. It can be unary (for one ENTITY), binary (for two ENTITIES), ternary (for three ENTITIES) and even more. The functionality is the number of entities that can exist on both sides of the RELATION. It can be 1:1 (one to one), 1:N (one to many) and N:M (many to many). The cardinality resembles the functionality but adds the details with regard to the minimum of entities in both sides of the RELATION. It can be (0,1) which means zero or one, (1,1) must have one entity in this side, (0,n) zero to many and (1,n) must have one entity in the side or more than one.

RELATION Level

- A Binary RELATION (the simplest relation level):
  - A RELATION between 2 ENTITIES.
    - Examples:

    A mechanic serviced a car.

    A song is composed by an artist.

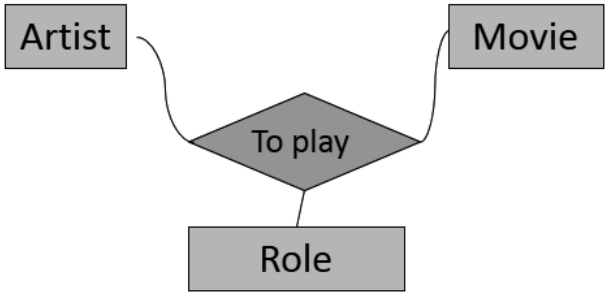
    A copy of a book.

    An employee works in a department.

As mentioned in the previous slide, a RELATION level shows how many ENTITIES are related in the RELATION. The most common RELATION level is the binary RELATION, which means that two ENTITIES have a relationship between them.

RELATION Level (contd.)

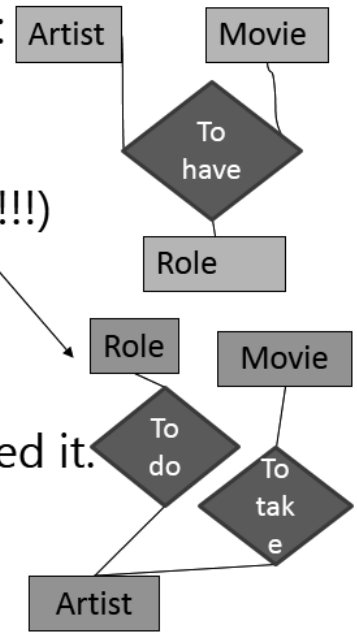
- Ternary RELATION:
  - A RELATION between 3 ENTITIES.
  - It is necessary to store data related to all 3 ENTITIES.
  - A Ternary RELATION is drawn as follows:



A ternary RELATION is a relation between three ENTTIES. Sometimes it is needed to create a meaningful RELATION between more than two ENTITIES. Without the ternary relation, we will lose the information that we want to hold with regard to some ENTITY.

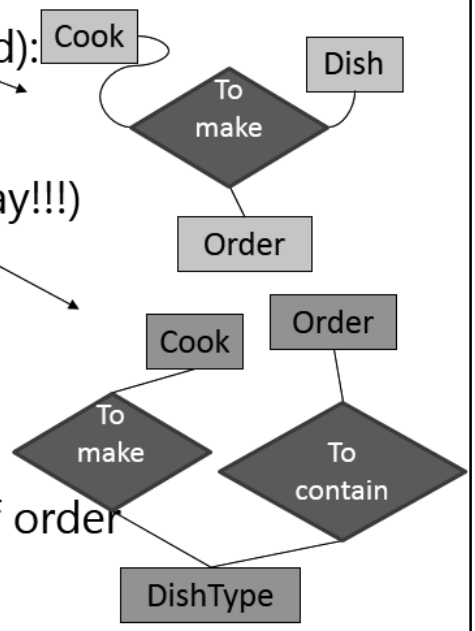
Example 1- In the Theater

- Option1- Ternary (correct method):
  - An artist has a role (acts, produces, edits, etc.) in a movie.
- Option 2- two Binaries (wrong way!!!)
  - An artist has a role.
  - An artist participated in a movie.
- With option 2, you don't know who acted in this movie and who produced it.
  - Woody Allen was an actor in the movie – "Sleeper", but the producer of the movie - "Bananas"



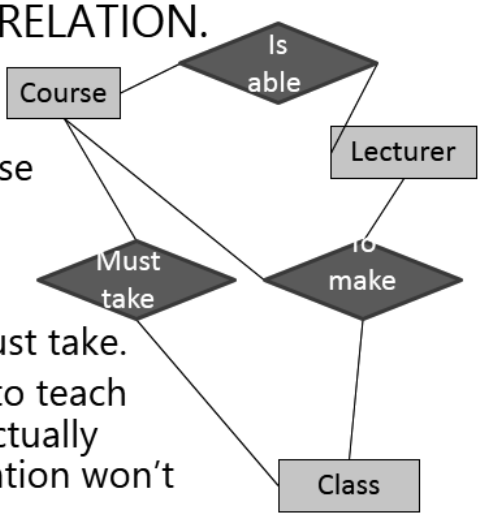
Example 2 - In the Restaurant

- Option1- Ternary (correct method):
  - A cook makes a dish (type) for a specific order.
- Option 2- two Binaries (wrong way!!!)
  - A cook makes many dishes (type).
  - An order contains many dishes (types).
- In option 2, you will not know which cook burned the steak ordered within the framework of order number 73.



Ternary RELATION, and Binary RELATIONS - Between 3 ENTITIES

- Example: A lecturer gives courses (types) to a specific class - this is a ternary RELATION.
  - Each lecturer is able to teach several course types.
  - Each class must take several course types (university requirements).
  - The ternary RELATION doesn't contain data regarding lecturer quality or course types a class must take.
  - If we have a lecturer who is able to teach several course types, but never actually teaches any class, the ternary relation won't contain that data.



RELATION Level - Ternary

- Common mistakes:
  - Three binary RELATIONS are not equivalent to one ternary RELATION.
  - A ternary RELATION doesn't "save" your RELATIONS, it represents different data and information retrieval options.

There are some common mistakes with regard to the ternary RELATION. The first one is that three binary RELATIONS are equivalent to one ternary RELATION. This is very wrong since not always the three binary RELATIONS are meaningful to create a ternary RELATION. A ternary RELATION must have a meaning which relates the three ENTITIES in a relation, and without the RELATION we won't have that meaning.

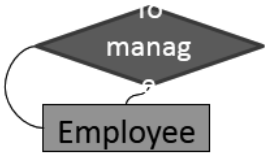
The second mistake is that ternary RELATION saves RELATIONS. That is also wrong. It represents a RELATION between three ENTITIES which represents different data and information retrieval options but nothing more.

Class Exercise - Is This a Ternary RELATION?

- A student takes a course. A course is given by a lecturer.
- An actor wins a prize (Oscar, Cannes, Golden Globe) for his or her part in a movie.
- Dr. Do-Little examines many animals. Each animal has an owner.
- A car is taken for a service type (refuel, wash, oil check, etc.) in a specific gas station.
- A reader borrows a copy of a book.
- A salesman sells a product type
- (T-shirt, cap, tie, etc.) to a customer.

Unary RELATION

- Also called a “reflexive” RELATION.
- A RELATION between an ENTITY and itself.
- Means that there is a connection between one entity of the ENTITY and another entity of the same ENTITY.
  - Each employee is managed by another employee (except for the BIG boss).
  - Each citizen is a child of another citizen.
  - (Many) ingredients can be replaced by other ingredients in the recipe (I hope I never have to eat in that restaurant).
  - A project could be part of another project.
- A unary RELATION is drawn as follows:
- Class exercise:
  - Provide 5 examples of unary RELATIONS.



Unary RELATION is a situation in which one entity will "point" to another entity with the same type. It is also called a “reflexive” RELATION. It sometimes refers to hierarchies or father and child relations but not always. It isn’t common in databases like binary RELATION but it is very useful.

RELATION Functionality

- Means: whether we can build a matching function between the ENTITIES on either side of the RELATION.
- Means: for each entity on one side of the relation, how many entities can we match on the other side.
- Examples:
  - Each bus is driven by one driver. 1 to many (1:N)
  - Each student takes many courses. Each course is taken by many students. Many to Many (N:M).
- Possible RELATION functionality:
  - 1:1, 1:N, M:N

For the mathematically minded.

For mere mortals.

We discovered what is RELATION level and the next stop is the RELATION functionality. What RELATION functionality is all about; that for each entity on one side of the relation, how many entities can be its match. There are three possible RELATION functionalities:

- 1:1 – one to one. For each entity on one side of the relation, there can be only one entity on the other side of the relation.
- 1:N – one to many. For each entity on one side of the relation, there can be a lot of entities related on the other side of the relation.
- M:N – many to many. There can be a relation between many entities in one side of the relation to many entities on the other side of the relation.

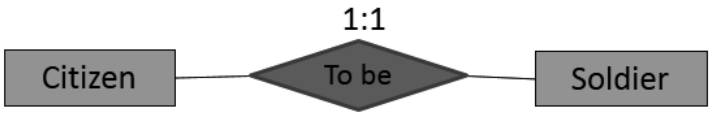
RELATION Functionality (contd.)

- 1:1 One to One -
  - For each entity on one side of the RELATION, there is no more than one (could also be zero, not even one) entity on the other side of the RELATION.
    - Each Citizen might be a soldier (but might not...) .
    - Each Department has one manager. A manager cannot manage more than one department.
    - Each mobile phone has one line.
  - To distinguish between a Unary RELATION and one to one functionality - there is no necessary connection.

For each entity on one side of the relation, there can be only one entity on the other side of the relation.

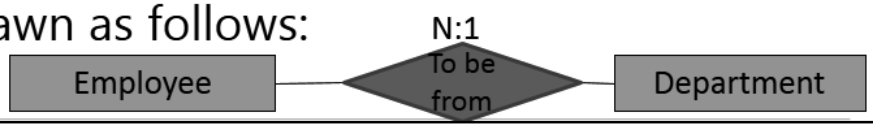
RELATION Functionality (contd.)

- A 1:1 RELATION is drawn as follows:



1:N One to Many

- For each entity on the "first" side of the RELATION there are unlimited number of entities on the "second" side, but not vice versa.
- Examples:
  - A copy is of a particular book, but each book can have many copies.
  - A company produces many cars of each car type. Each car is an instance of only one car type.
  - A product type is produced by its manufacturer. A manufacturer can produce many product types.
- 1:N is drawn as follows:



1:N or N:M

- In some cases, the RELATION between instances of an ENTITY is one to many at a specific point in time, but many to many, historically.
  - An employee works in one department.
  - If we wish to store data regarding the current status, a one to many RELATION provides that information.
  - In storing the employee's history and his former and current departments, we will use a many to many RELATION.
    - What PROPERTIES does the RELATION have? (answer: "starting date" and "finish date").

There are times that the RELATION between instances of an ENTITY are not certain enough so we have to make a business decision of which functionality to imply on the RELATION, 1:N or N:M. Since the organization demands might change in the future, this decision is sometimes crucial for later maintenance of the model.

N:M Many to Many

- For each instance on one side of the RELATION, there are multiple matching instances on the other side, and vice versa.
  - A Student takes many courses. Each course is taken by many students.
  - Each computer contains many part types, each part type is used in many computers
  - Each customer has multiple bank accounts. One bank account can be owned by more than one customer.
  - Many customers order a book title (not a copy). Each customer can order several books.
  - An author writes many books. Several authors can write a book together.

N:M Many to Many (contd.)

- A many to many RELATION is drawn as follows:

RELATION Functionality – Remarks

- RELATION that can stand alone and has meaning without the ENTITIES (on its sides), can become a separate ENTITY.
  - Many passengers fly in an airplane.
  - A passenger can fly in several planes.
  - The flight is an independent item. We want to store data about the flight without regard to the passengers or the airplane. (Data such as.... flight path, correct!!! departure and landing time).

There can be situations when we can transform a RELATION into an ENTITY. The example above shows such a situation where the RELATION – the flight which is the RELATION between passengers and planes, can become an ENTITY, that holds data which concerns a flight's ENTITY.

Class Exercise: RELATION Functionality

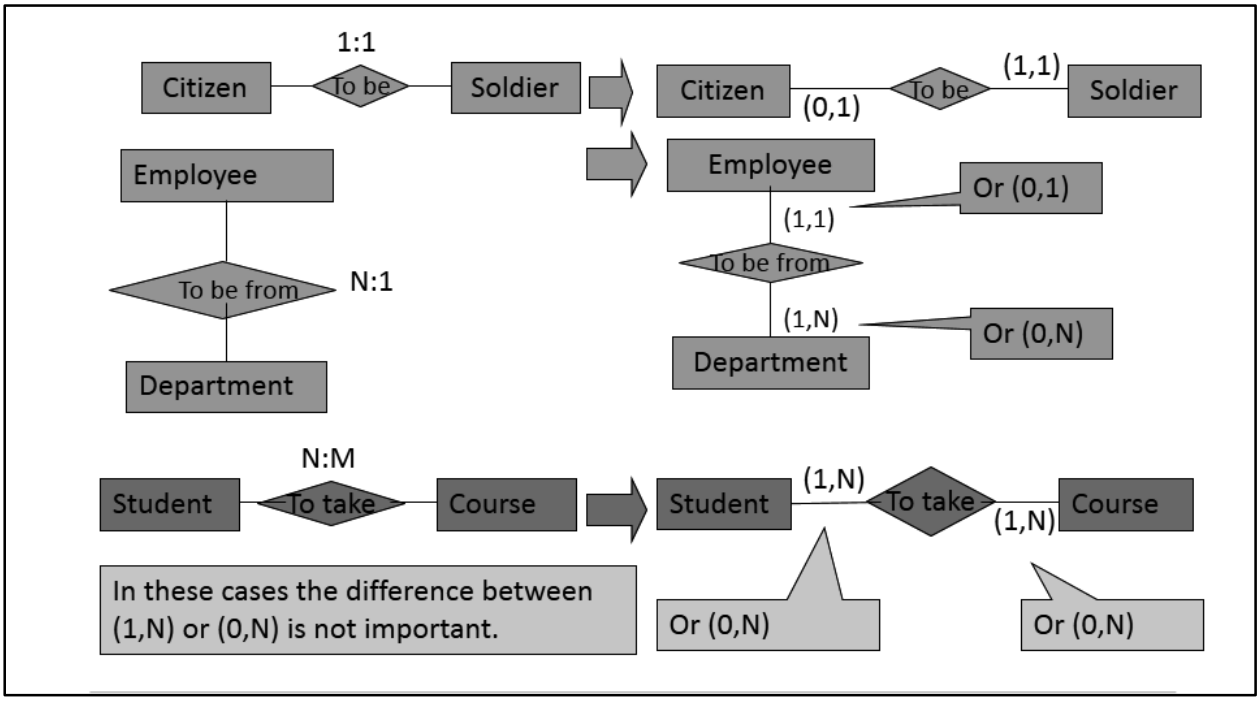
- Find the ENTITIES, RELATIONS and functionality in the following sentences:
  - Each song is of a song type (rock, pop, classical, etc.).
  - Each television talk show has several guests.
  - A course is of a course type (criminal law, domestic affairs, commercial contracts, biology, etc.).
  - A telephone call is made from one area to another area.
  - A course is taught by a single lecturer.

RELATION Cardinality

- RELATION cardinality:
  - Is a different way of demonstrating functionality.
  - It contains both the minimum and the maximum.
- In most cases the minimum is not important to the system designer.
- RELATION Cardinality possibilities:
  - (0:1), (1:1), (1:N)

We saw earlier that RELATION functionality provides the maximum number of instances on one side of the RELATION for instances on the other side (i.e., 1:N book and copies). Cardinality is a different way of looking at functionality. It also adds the minimum number of instances. In some places, the minimum isn't relevant but there are some RELATION which won't make sense without the minimum. For example: In some cultures, a husband can have one and only one wife (1:1). Can a husband have zero wives?

RELATION Cardinality: Example



The example above shows an ERD that uses the cardinality to explain how the relations between entities are created. For example, we can see that a citizen has a one to one relation with soldier but in cardinality 0..1 in regard of citizen to one in regard of soldier. That means that a citizen might be a soldier but sometimes he might not be. The opposite way is that a soldier must be a citizen.

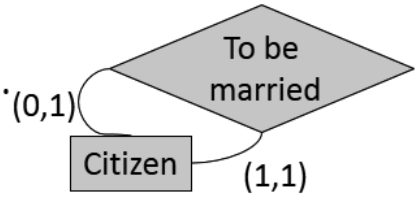
Level * Functionality * Cardinality

- In the ERD (coming up...), we will write - level and cardinality.
- We will speak in terms of functionality but in terms of the ERD, we will write cardinality.
- Possible RELATION (Level * Cardinality):
  - Unary 1:1
  - Unary 1:N
  - Binary 1:1
  - Binary 1:N
  - Binary N:M
  - Ternary N:M

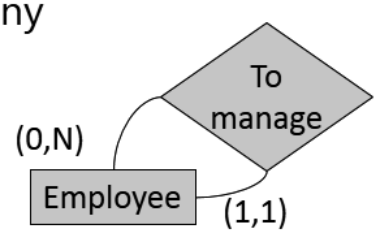
ERDs are very strict, which means that all the details must be within them, in order to create a meaningful model. Even though, the minimum part of a RELATION (which is added by the cardinality to the functionality) might be ignored by the system analyzer. In ERDs, we use cardinality and not functionality.

Level * Cardinality, Examples

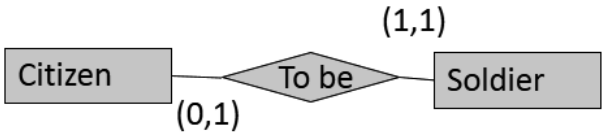
- Unary 1:1
  - A citizen is married to one citizen.



- Unary 1:N
  - An employee is a manager of many employees.

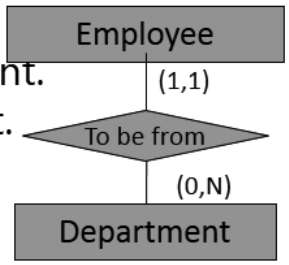


- Binary 1:1
  - A citizen might be a soldier.

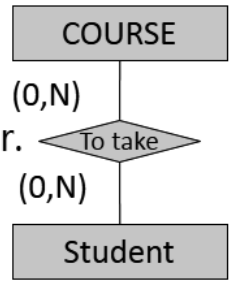


Level * Cardinality, Examples (contd.)

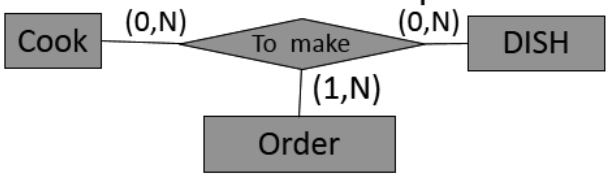
- Binary 1:N
  - Each Employee works in one Department.
    Many Employees work in a Department.



- Binary N:M
  - Students take many COURSES.
    A COURSE is taken by many Students.



- Ternary N:M
  - A Cook makes a DISH for a specific Order.



Class Exercise - Level * Cardinality

- Change the functionality (you wrote in the previous exercise) to cardinality:
  - Each song is of a song type(rock, pop, classical, etc.)
  - Each television talk show has several guests.
  - A course is of a course type (criminal law, domestic affairs, commercial contracts, biology, etc.).
  - A call is made from one area to another area.
  - A course is taught by a single lecturer.

Some Important Dilemmas and Issues

- The “time dilemma”:
  - A RELATION should be checked over the course of a significant period (as opposed to a specific moment).
    - Incorrect: Student can take only one COURSE at a specific moment.
    -
    - OK: Student takes many COURSES over a long period.
    - OK: a Copy of a book is a copy of one book even over an extended period.

Organizations are always changing and with them their business models. RELATIONS can change, ENTITIES can change, ENTITIES’ PROPERTIES can change over time. So, the models we build should be able to shift with time. Every time we should refresh the models if they don’t follow the organizational changes.

Some Important Dilemmas and Issues (contd.)

- A PROPERTY becomes an ENTITY when:
  - The PROPERTY is alphanumeric, but not a fixed list, and the user wants to filter by this PROPERTY.
    - An alphanumeric property, but not a fixed list: product color, DISH category (salad, fried, dessert, etc.), client's flight class (VIP, Business, Economy, etc.).
    - An alphanumeric property, and a fixed list: Gender (M,F), Family status (single, married, divorced, widow/er).
- The ENTITY has a relation of one to many with the PROPERTY:
    - An employee receives a different salary each month.
    - The employee's children's names.
    - A stock unit's value.

Sometimes a PROPERTY can become an ENTITY. These occasions occur in some cases such as:

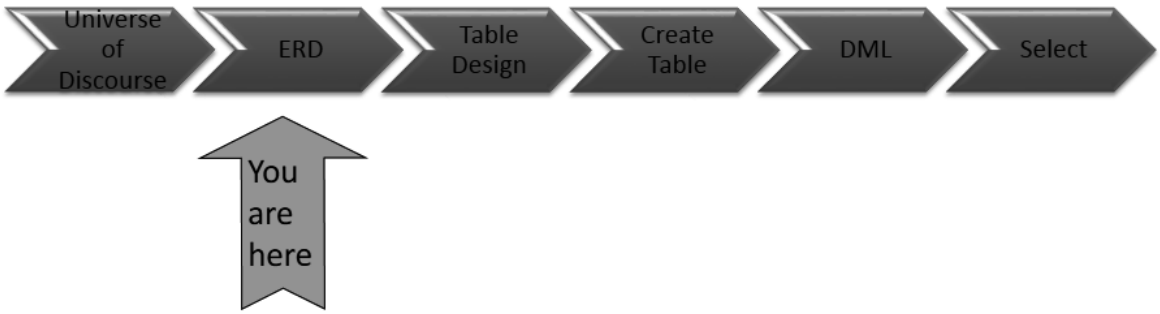
- The PROPERTY is alphanumeric and the user wants to filter by this PROPERTY. In this case we will create a "Lookup" ENTITY that will have a ID and a Description which will be the alphanumeric property. Lookup tables are very useful in a normalization process which will be discussed later.
- The ENTITY has a relation of one to many with the PROPERTY. In such cases, we will want to hold it in another ENTITY, with a one to many relation between these two ENTITIES.

Some Important Dilemmas and Issues (contd.)

- A PROPERTY becomes an ENTITY when:
  - The organization needs to store data regarding the PROPERTY with no relation to the source ENTITY.
    - A book's author, a product's manufacturer, a computer's CPU type.
- Class exercise:
  - For each PROPERTY, write whether that PROPERTY should become an ENTITY. Explain:
    - A product's color.
    - A customer's city.
    - A customer's street.
    - A Cigar's tobacco type.

From Universe of Discourse (UOD) to ENTITY RELATION Diagram (ERD)

- Up until this point, we have been dealing only with aspects of the UOD.
- We will now work on a complete UOD. We will build an ERD from a given UOD.



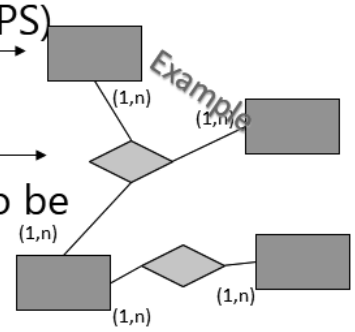
From Universe of Discourse (UOD) to ENTITY RELATION Diagram (ERD) (contd.)

- (Mentioned previously), UOD is the whole organizational data "story".
- Converting a UOD to an ERD checklist:
  - Mark all ENTITIES.
  - Write PROPERTIES for each ENTITY
  - Draw ENTITIES in the ERD (in single, CAPS)
  - Draw RELATION between ENTITIES.
  - Record the relation's cardinality.
  - Check the ERD based on CODD rules (to be explained later).

Example

Clients make many orders
Order contains many Products
Employee takes Order

Example



Following the creation of UOD which describes the data "story", we will create the ERD which is the UOD's graphical representation.

When you want to transform UOD to ERD the following checklist will help:

- Mark all ENTITIES.
- Write PROPERTIES for each ENTITY
- Draw ENTITIES in the ERD.
- Draw RELATION between ENTITIES.
- Record the relation's cardinality.
- Check the ERD based on CODD rules.

Check Your ERD - CODD Rules

- Provide a more mathematical way of viewing the normalization process.
- Run from the bottom up (i.e., from entities and properties to ENTITIES and PROPERTIES)
- Used to determine whether a DBMS can be considered as relational DBMS.
- First 3 rules:
  - Tables will include columns with atomic values and a primary key. Column or row order isn't significant.

CODD rules provide a mathematical way, of how we see the normalization process when we create an ERD. There are thirteen rules in the document. With these rules you can determine whether a given database can be considered a relational database. The rules (taken from Wikipedia):

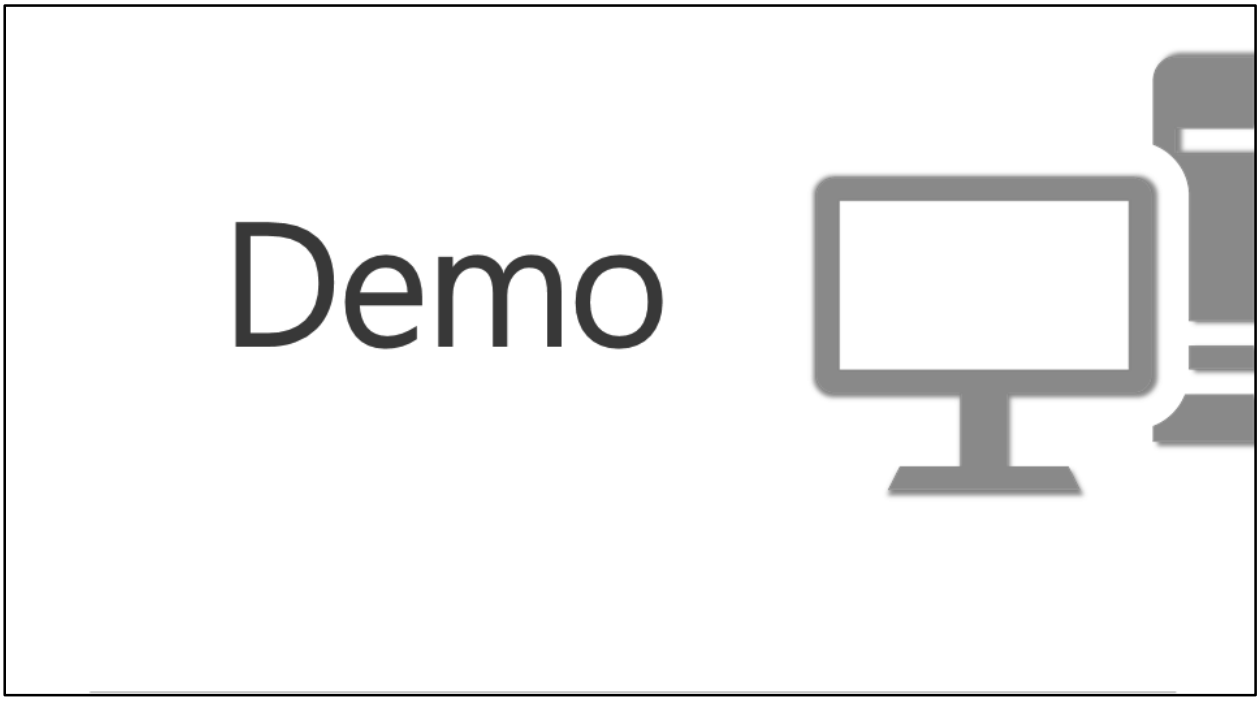
- **Rule 0:** The system must qualify as relational, as a database, and as a management system.
- **Rule 1:** The information rule.
- **Rule 2:** The guaranteed access rule.
- **Rule 3:** Systematic treatment of null values.
- **Rule 4:** Active online catalog based on the relational model.
- **Rule 5:** The comprehensive data sublanguage rule.
- **Rule 6:** The view updating rule.
- **Rule 7:** High-level insert, update, and delete.
- **Rule 8:** Physical data independence.
- **Rule 9:** Logical data independence.
- **Rule 10:** Integrity independence.
- **Rule 11:** Distribution independence.
- **Rule 12:** The nonsubversion rule.

  You can farther read about the rules and their meanings in the following link:
  http://en.wikipedia.org/wiki/Codd's_12_rules

Check Your ERD - CODD Rules (contd.)

- Each column that isn't part of the primary key will be dependent on the complete value of the primary key.
- A column will be independent of a column that isn't the primary key.

E.R.D. – Class Examples

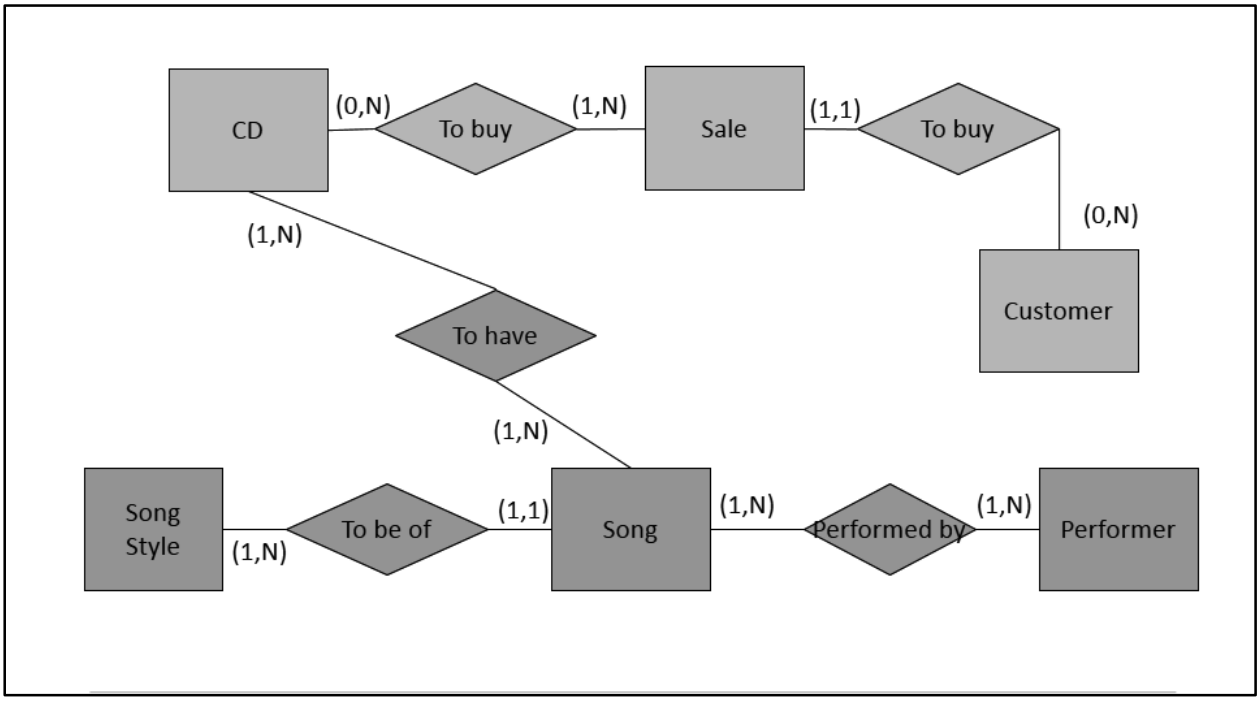


Class Example 1: “See This”

- In the CD store there are many CDs.
- Each CD contains various songs (“Let it Be”, “Hey Jude”, etc).
- Each song can appear on many CDs.
- Each song is of a song style (rock, pop, classic, etc.).
- Each song is performed by a certain performer or performers.
- A Sale is made by a customer.
- Is a sale, customers buy CDs .

Remark: We want to store the song style and performer/s for each song, to make CDs such as collections and soundtracks.

Solution - Class Example 1: “See This”



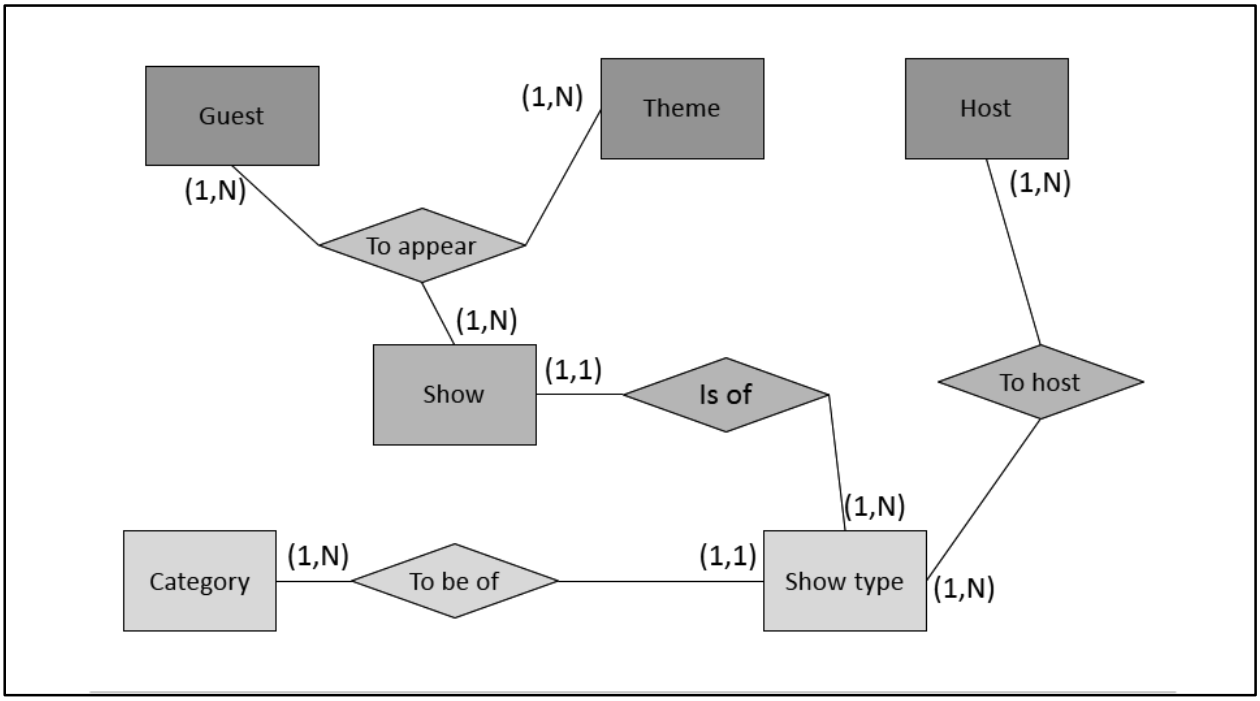
Class Example 2: “Tell-vision”

- There are many TV shows on the Tell-Vision TV network.
- Each show is of a show type (60 Minutes, Friends, E.R., Jay Lenno, etc.)
- For each show type, there are one or more hosts.
- Guests appear on shows with relation to a theme (art, politics, economics, etc.).
- Each show type is of a different category (game show, news, talk show, sitcom, drama, etc.).

Remarks:

- Each show is of a scheduled show type.
- Remark: Each Guest can appear with relation to several themes.

Solution - Class Example 2: “Tell -Vision”

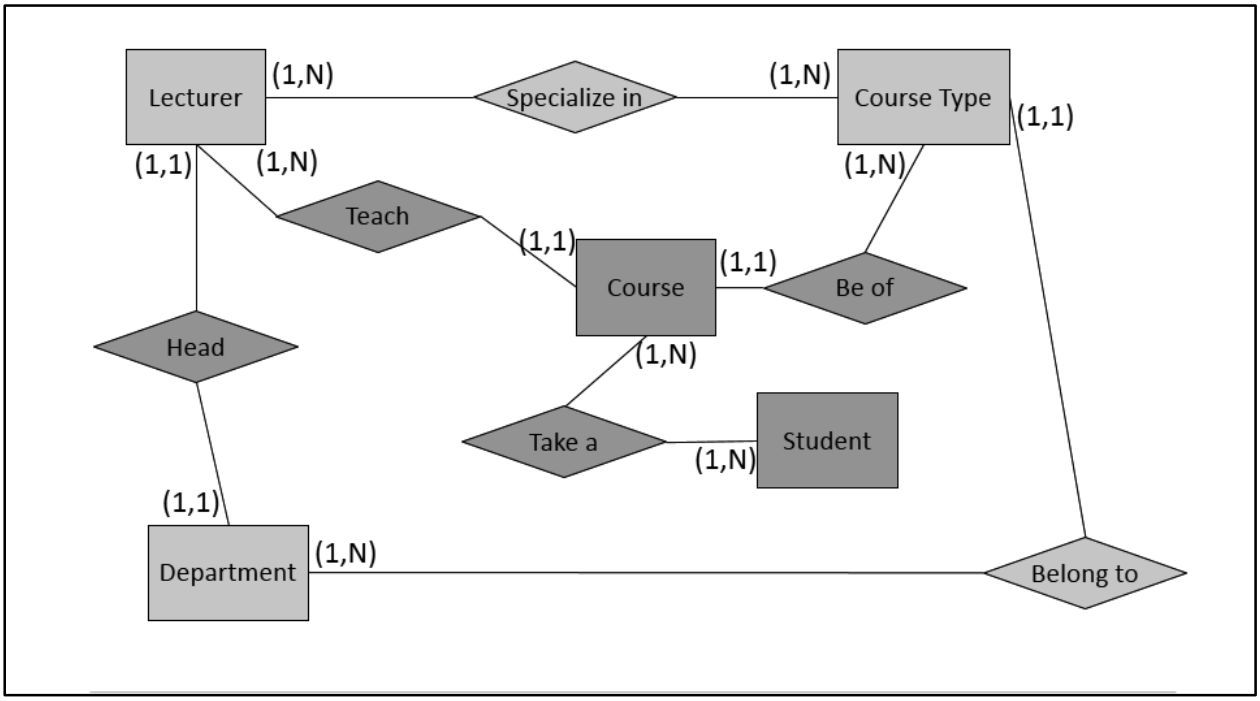


Class Example 3: DeepLawMass”

- The DLM Law Academy has courses, lecturers and students.
- A student takes several courses.
- A course is of a course type (criminal law, matrimonial law, commercial contracts, corporate law, etc.), and has a start date.
- A course is taught by a single lecturer.
- A lecturer specializes in a course type/s.
- A course type belongs to a certain department.
- One lecturer heads each department.

Remark: a lecturer heads no more than one department.

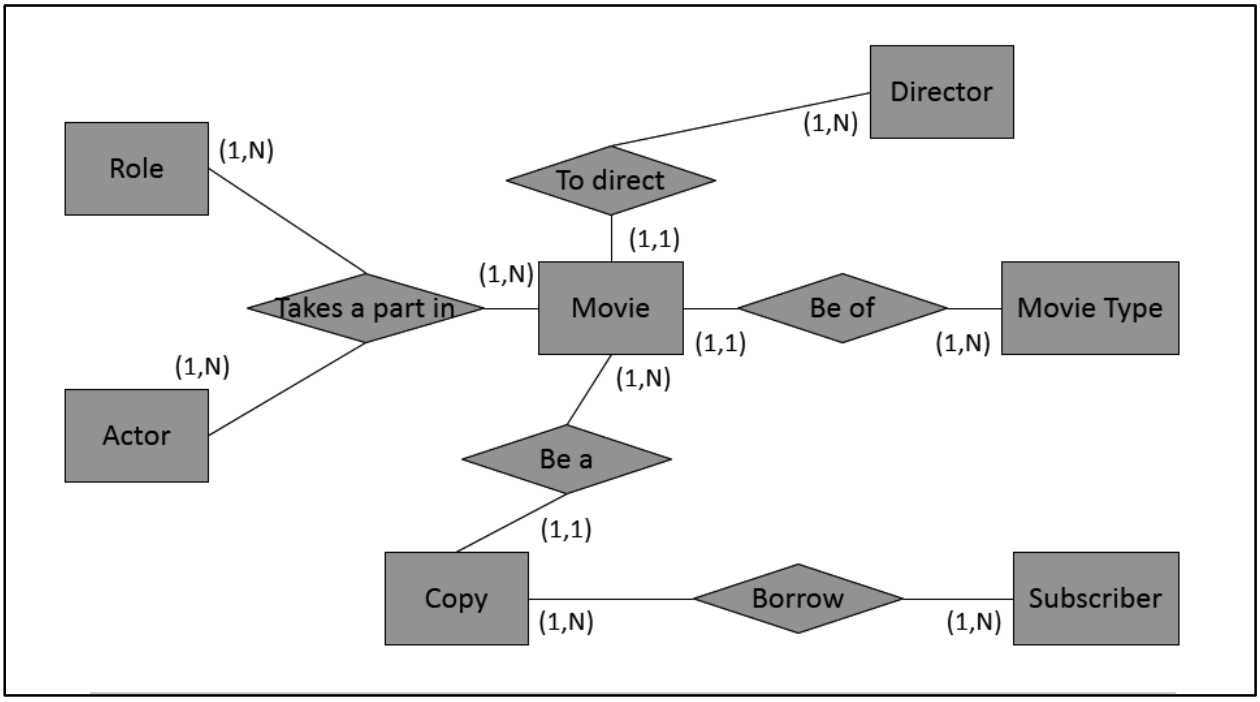
Solution Class Example 3: “DeepLawMass”



Class Example 4: "Joe's Movies" (Ternary connection between entities)

- Joe's video rental store has many movies.
- Each movie is of one movie type (drama, action, comedy, etc.).
- Actors appear in movies, each actor can play a different role/s in each movie (star, co-star, extra, stuntman, etc.).
- Each movie is directed by a director in a specific year.
- A subscriber borrows a copy or copies of a movie/s on a certain date and returns it/them on a certain date.

Solution Class Example 4: "Joe's Movies"



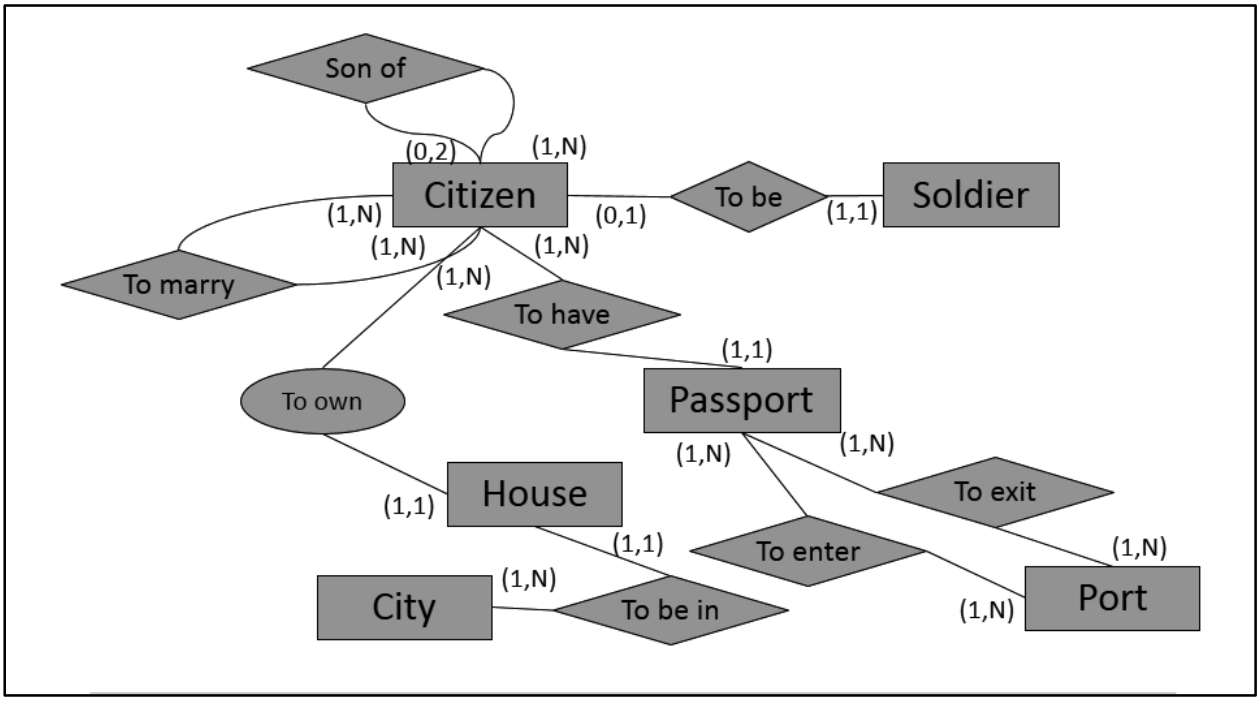
Class Example 5:
"Banana Republic"

- There are many citizens and soldiers in the Republic.
- A citizen can be a soldier.
- A soldier has an ID, a rank and a service start date.
- A citizen has a mother and a father (who are also citizens).
- A citizen can be married to another citizen.
- A citizen can get divorced and remarry.
- A citizen can own a house or houses.
- A house is in a certain city.

Class Example 5:
"Banana Republic" (contd.)

- A citizen can have many passports (only the Republic's), but only one of them is valid (the others have expired).
- A citizen can use his valid passport to enter and exit the country from different ports.

Solution - Class Example 5: "Banana Republic"



Lab: E.R.D. – Class Exercises

Class Exercise: “Fishy Beauty”

- The fish store has various products and fish.
- A product is of a product type (air pump 750, dry food 500g, etc.).
- A fish is of a fish type (goldfish, tiny shark, guppy, etc.).
- Each product type is supplied by one supplier.
- Product type has a certain size, price and weight.
- Customers buy products and fish.
- Each fish type is supplied by one supplier.

Remarks:
In order to simplify matters, a fish and a product type are not the same ENTITY.
no data is stored about each product or fish. The store maintains data on product types and quantities of fish type in stock.

Class Exercise : “Fishy Beauty” (contd.)

- A fish type is from a certain country.
- A fish needs certain product types to survive (certain food or the presence of a certain product/s).

Class Exercise : The “EUROVISION”

- In the annual European song contest, there are many songs, each one representing a different country.
- A song is sung by a singer or singers.
- A singer can sing only one song each year.
- Musicians play various instrument types in each song.
- A musician can play different instrument types in different songs.
- Each song’s lyrics are written by a songwriter.
- Each song is composed by a music composer.

Class Exercise : The “EUROVISION” (contd.)

- There is only one winner in each contest.
- Each singer is either a lead singer or a backup singer.

Class Exercise : “Hotel California” (Time Period as an Entity)

- Guests stay at the hotel on certain dates.
- A guest stays in a room or rooms.
- A stay is of a stay type (B&B, half board, full board, etc.).
- During a certain stay, guests order drinks.
- A room has a number of beds of type single, double or child (no other types).
- The room is of a certain type (regular, suite, etc.).
- A guest makes a reservation for a certain period.

Remark: reservations and stays might not be on the same date.

Class Exercise : “Dr. Do-Little”

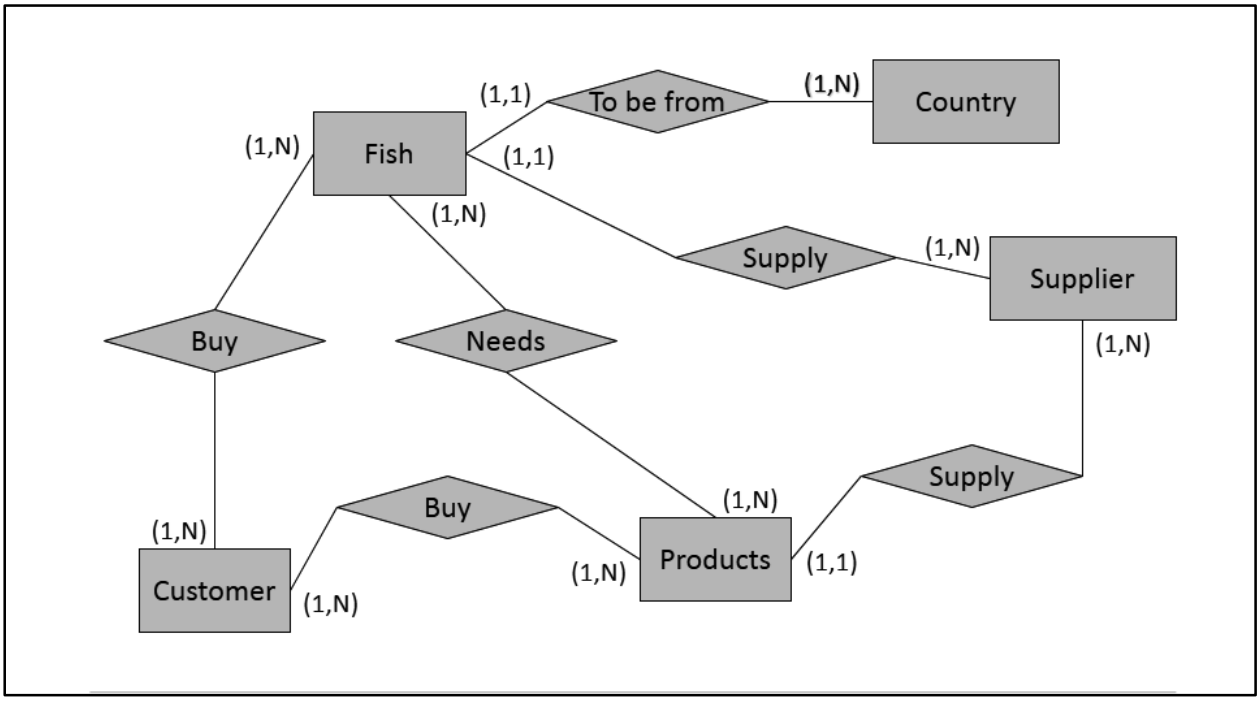
- The Dr. diagnoses pets.
- People own pets, pets are owned by 1 owner.
- A diagnosis for the pet is given to its owner.
- Several treatments are given for each diagnosis.
- One treatment could be given for several diagnoses.
- A pet has a history of treatments for all diagnoses.
- A pet (cat named “Mitzi”, dog named “Goofy”, etc.) is of a certain pet type (bull-terrier, Siamese, viper, boxer, etc.).

Class Exercise :"Talk To Me"

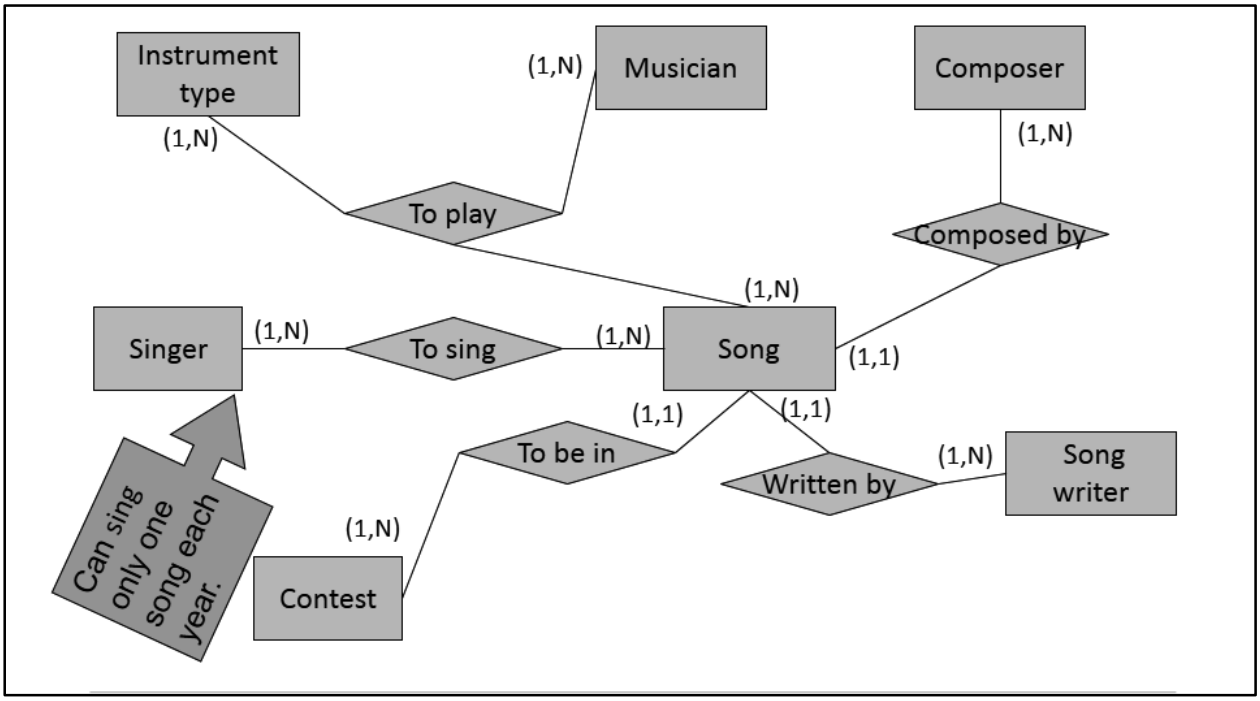
- There are many calls in the TTM Cellular Phone Services Company.
- Customers own cellular devices.
- A line is assigned to 1 or 2 devices maximum (mobile and car phone).
- A call is made from an 'area cell' and from a certain line.
- A call is made to line/s that are in an 'area cell'/s (between 2 lines or conference calls).
- A line belongs to a program (students, one line for half price, leisure time, etc.).

Lab: E.R.D. – Class Exercise Solutions

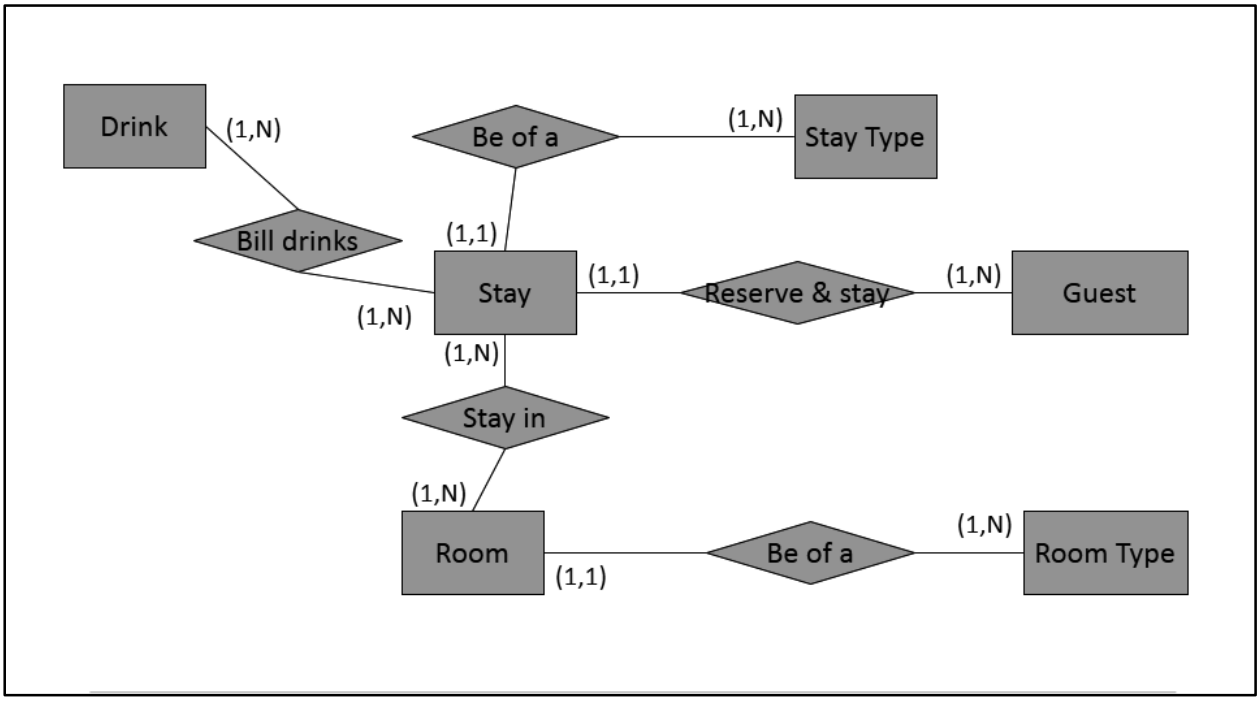
Class Exercise : “Fishy Beauty” Solution



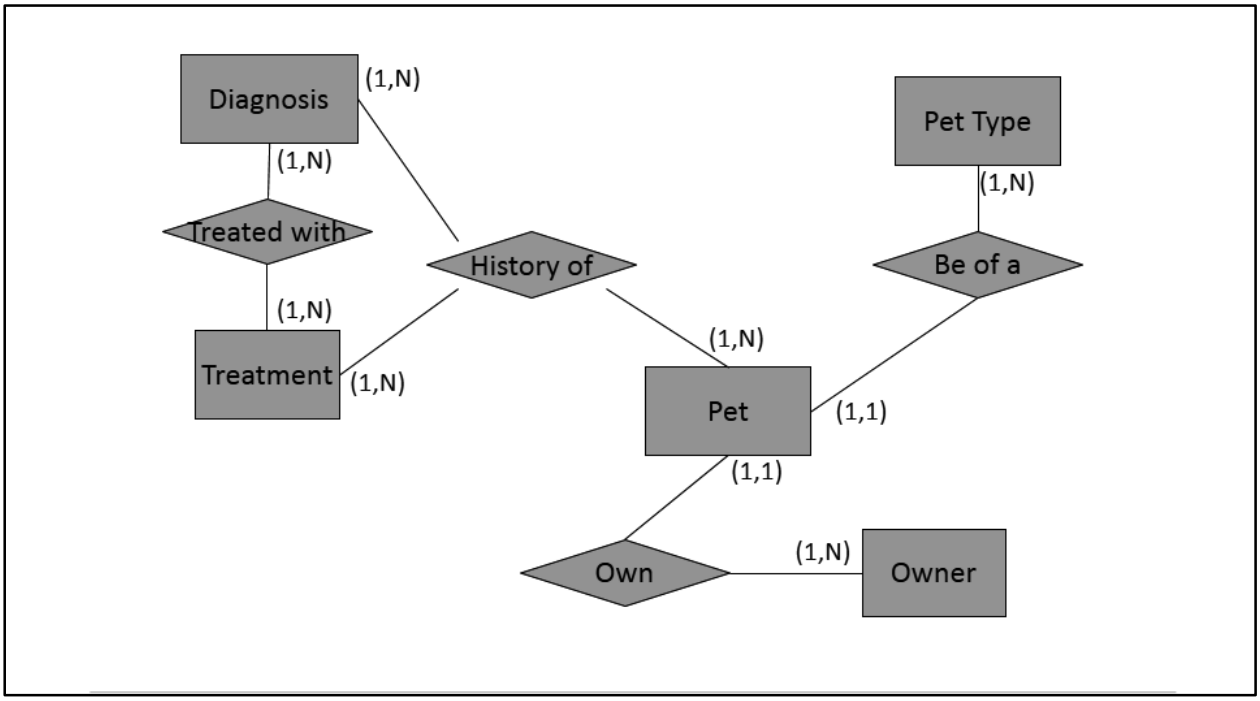
Class Exercise : “EUROVISION” Solution



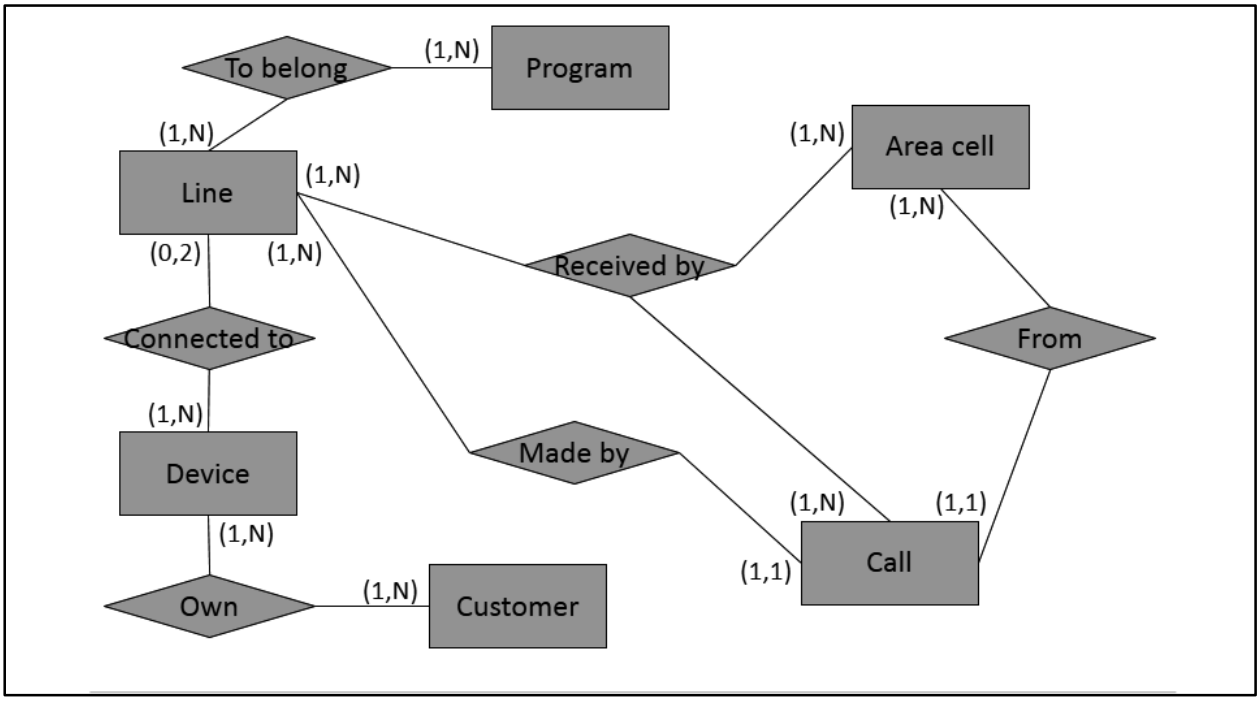
Class Exercises : “Hotel California” Solution



Class Exercises : “Dr. Do-little” Solution



Class Exercise : “Talk To Me” Solution



Final Exercises – UOD, ERD – Exercise 1

- Part I –
  Find the Entities and their properties (15 points)
  The following sentences were taken from different universes of discourse.
  Find out the Entities and mark them. What are their properties?
  - Example:
    Q: An employee has an employee number, employee name and employee start date.
    What is the Entity and what are its properties?
    A: Employee.

Final Exercises – UOD, ERD – Exercise 1 (contd.)

- Properties:
  - employee num,
  - employee name,
  - start date.

Final Exercises – UOD, ERD – Exercise 1 (contd.)

Part II – Find the Entities and their relations (25 points)

Draw the Entities and their relations in the following statements.

- Example: An employee belongs to a department.

1. At the Sela Institute there are many courses. Students take many courses.
2. A citizen can be a soldier as well.
3. Many bands participate in each contest.
4. A teacher teaches various types of courses.
5. A mechanic is certified to handle too many faults.
6. Each dish in a restaurant has many ingredients.

Final Exercises – UOD, ERD – Exercise 1 (contd.)

7. An actor participates in a film.
8. A dish is served by a waiter.
9. Each musical instrument belongs to a category (strings, brass, percussion).
10. Dancers form a band.

Final Exercises – UOD, ERD – Exercise 1 (contd.)

- Part III – ERD, basic models (60 points)

  Assume that you already have a brief from the system analyst.
  Don't add items to the described universe of discourse.
  This exercise is tutorial, and is not intended to analyze the 'real world'.

  According to the described design you should build an ERD that includes:

  Finding the right Entities, finding the right relation Entities.
  Even if you've already learned cardinality, you don't have to mark it.

Final Exercises – UOD, ERD – Exercise 1 (contd.)

- Design No. 1: National Library
  - In the library, there are books.
  - Each book has several copies.
  - Subscribers order books but borrow copies.
  - A book has a subject.
  - Below are some of the Entities that should appear in your ERD:
  - Book, Copy, Subscriber, Subject.

Final Exercises – UOD, ERD – Exercise 1 (contd.)

- Design no. 2: The City Conservatory of Music
  - Teachers can teach one instrument or more.
  - Each instrument belongs to one instrument type.
  - Teachers teach students to play one or more instruments.
  - Below are some of the Entities that should appear in your ERD:
    - Instrument, InstType, Student.

Final Exercises – UOD, ERD – Exercise 1 (contd.)

- Design no. 3: Gas Company
  - The company has many gas stations all over the company.
  - In each station there are employees.
  - The employees work in shifts.
  - Each shift has a starting hour and a finishing hour.
  - Each employee works at one station only.
  - Vehicles arrive at stations for refueling.

Final Exercises – UOD, ERD – Exercise 1 (contd.)

- Each vehicle is filled by one of the employees at the station.
- Below are some of the Entities that should appear in your ERD:
  - Station, Employee, Vehicle, Shift

Final Exercises – UOD, ERD – Exercise 1 (contd.)

- Design no. 4: Estate Agency
  - Houses for sale are located at a specific region in the city (in the north, center, south, etc.).
  - Each house is from a house type, such as: villa, apartment, penthouse, etc.
  - A House is sold by a salesman.
  - A sale is made from an existing owner to a new owner.
  - Each salesman specializes in a different house type, and sometimes there are several salesmen that specialize in the same house type.
  - Below are some of the Entities that should appear in your ERD:
    - House, Region, Owner, Salesman, House Type, Sale.

Final Exercises – UOD, ERD – Exercise 2

- Part I – ERD, complex models
  - Assume that you already have a brief from the system analyst.
  - These exercises are tutorials and are not intended to analyze the 'real world'.
  - According to the described design, you should build an ERD that includes:
    - Finding the right ENTITIES.
    - Finding the right relations between the ENTITIES.
    - Marking the cardinality.

Final Exercises – UOD, ERD – Exercise 2 (contd.)

- Design No. 1: Taxi Company
  - A taxi company manages many drivers and their taxies.
  - A driver drives a taxi. Each driver has his own taxi and does not switch taxies with any other drivers.
  - The taxies are of different types (van, compact, station wagon, etc.).
  - A driver drives a customer in his taxi and gets a payment for it.
  - Each driver is under another driver who supervises him. There is only one driver who is the manager himself (the boss).
  - A taxi has faults (once in a while). Each fault is of a given fault type (changing oil, bodywork repair, etc.).

Final Exercises – UOD, ERD – Exercise 2 (contd.)

- Below are some of the Entities:
  - Taxi, Customer, Driver, Fault.

Final Exercises – UOD, ERD – Exercise 2 (contd.)

- Design no. 2: Wood Company
  - The wood company creates many wood products.
  - A product type is from a specific series (e.g.: a product type named dining table belongs to the home furniture series, and a puzzle belongs to the games series).
  - A supplier supplies products to stores.
  - The stores are in cities, and they sell the products to the consumers.
  - Each supplier supplies several company products.

  - Below are some of the Entities:
    - Consumer, Product Type, Supplier, City, Store, Series.

Final Exercises – UOD, ERD – Exercise 2 (contd.)

- Design No. 3: Party hall Company
  - The company deals with the management of special events.
  - The company has halls where the events take place.
  - Each event is of a specific event type (wedding, Bar-Mitzvah, circumcision, etc.).
  - The maximum number of people in a hall is 300 – for safety reasons such as fire hazards, etc.
  - An event is ordered by a customer.
  - The company provides performers for events.
  - Each performer specializes in a certain performance type (singer, dancer, etc.)

Final Exercises – UOD, ERD – Exercise 2 (contd.)

- Below are some of the entities:
  - Event, Customer, Performance Type, Performer, Hall, Event Type

Final Exercises – UOD, ERD – Exercise 2 (contd.)

- Part II – ERD and Normalization (50 points) for the following UODs

1. Build an ERD.
2. Convert it into a table design.
   - Implement the normalization rules and add new tables if needed.
   - Draw a table design that includes the tables, their properties, primary keys and foreign keys as needed.

Final Exercises – UOD, ERD – Exercise 2 (contd.)

- Design Number 1: Para Para Restaurant The "Para Para" restaurant is a dairy restaurant.
  - After some meetings with the restaurant manager, you have made the
- following summary:
  - Customers sit in the restaurant.
  - A customer makes an order.
  - Each order includes some dishes.
  - A waiter takes an order.
  - A chef makes the dish especially for the order.
- Below are some of the entities:
  - Customer, Order, Dish, Waiter, Chef

Final Exercises – UOD, ERD – Exercise 2 (contd.)

- Design No. 2: El-Al Airlines
  Our design is El Al-Airlines
- After some meetings with the company's managers, you have made the following summary:
  - The company has several set routes (lines).
  - Each line starts at one airport and ends at another airport.
  - Each airport is located in a certain city.
  - Each line has its flights.
  - In each flight there is a certain airplane.
  - Each passenger has a maximum of 2 suitcases with different weight (we don't want to keep information about suitcases except their weight).

Final Exercises – UOD, ERD – Exercise 2 (contd.)

- A flight takes off and lands at a certain hour.
- Passengers order tickets for flights.

- Below are some of the entities:
  - Passenger, Line, Flight, Airport, City

Questions

Module 04 - Converting the ERD to a Table Design

Contents:

In This Chapter We Will 5
Translating the ERD to Tables and Columns 6
What is a Table? 7
The Process of Converting the ERD Into a Table Design 8
Converting ENTITIES Into Tables 9
Tables 10
Tables (contd.) 11
Class Exercise 01 12
Converting Relations 13
Binary 1:N 14
Binary 1:N (Implementation) 15
Class Exercise 02 16
Binary N:M 17
Binary N:M (contd.) 18
Binary N:M (contd.) 19
Binary N:M (contd.) 20
Binary N:M (contd.) 21
Class Exercise 03 22
Binary 1:1 23
Binary 1:1 (Implementation) Option A 24
Binary 1:1 (Implementation) Option B 25
Class Exercise 04 26
Ternary (or more) N:M 27
Ternary (or more) N:M (contd.) 28

Ternary (or more) N:M – cont'd 29
Class Exercise 05 - Ternary 30
Unary 1:N 31
Class Exercise 06 - Unary 1:N 32
Unary 1:1 33
Class Exercise 07 - Unary 1:1 34
From ERD to Tables - Summary 35
From ERD to Tables - Summary (contd.) 36
From ERD to Tables - Remarks 37
From ERD to Tables - Remarks (contd.) 38
From ERD to Tables - Remarks (contd.) 39
Summery 40
Transition to the Table Model Class Examples 41
Class Example 1: "See This" 42
Solution - Class Example 1: "See This" 43
Transition to the Table Model 1: "See This" 44
Class Example 2: "Tell-Vision" 45
Class Example 2: "Tell-Vision" (contd.) 46
E.R.D. - Class Example 2: "Tell-Vision" 47
Transition to the Table Model 2: "Tell-Vision" 48
Class Example 3: DeepLawMass" 49
E.R.D. Class Example 3: DeepLawMass" 50
Transition to the Table Model 3: "DeepLawMass" 51
Class Example 4: "Joe's Movies" 52
E.R.D. Class Example 4: "Joe's Movies" 53
Transition to the Table Model 4: "Joe's Movies" 54
Class Example 5 : "Banana Republic" 55
Class Example 5 : "Banana Republic" (contd.) 56
Transition to the Table Model 5: "Banana Republic" 58
Lab: Transition to the Table Model Class Exercises 59
Class Exercise 1: "Fishy Beauty" 60

Class Exercise 1: “Fishy Beauty” (contd.) 61
Class Exercise - “Fishy Beauty” ERD1 62
Class Exercise 2: The “EUROVISION” 63
Class Exercise - “EUROVISION” ERD 2 64
Class Exercise 3: “Hotel California” 65
Class Exercise – “Hotel California” ERD 3 66
Class Exercise 4: “Dr. Do-little” 67
Class Exercise - “Dr. Do-little” ERD 4 68
Class Exercise 5: “Talk To Me” 69
Lab: Transition to the Table Model: Class Exercise Solutions 71
Class Exercise 1 “Fishy Beauty” Solution: 72
Class Exercise 2 “EUROVISION” Solution: 73
Class Exercise 3 “Hotel California” Solution: 74
Class Exercise 4 “Dr. Do-little” Solution: 75
Class Exercise 5 “Talk To Me” Solution: 76

SELA

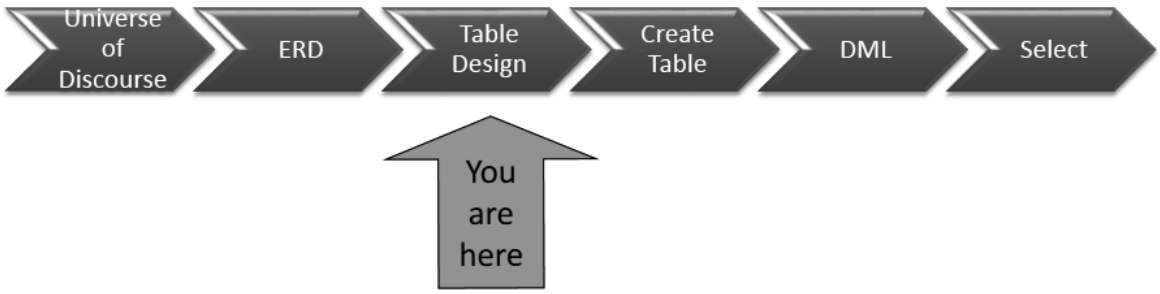
Introduction to Relational Databases and SQL

Module 04 - Converting the ERD to a Table Design

© Copyright SELA software & Education Labs Ltd. | 14-18 Baruch Hirsch St Bnei Brak, 51202 Israel | www.selagroup.com

In This Chapter We Will

- Translate the ERD to tables and columns.
- Study the Relational Model.



Translating the ERD to Tables and Columns

- Why?
  - The ERD is a design model.
  - The ERD is not computable.
  - The ERD does not display PROPERTIES.
  - The Engine requires that the database be structured in tables and columns.
- Therefore:
  - Table design will be a computable design based on the ERD.
  - Table design will include columns.
  - Table design will contain the necessary links between the ERD'S ENTITIES.

When we want to build a database, first we will build it's ERD. After we have an ERD, it is much easier to create a table design which will convert the design model (the ERD) into a computable design. The table design will be translated in future into a real database table. The table design includes all aspects of a real database. Tables have columns, (which are the PROPERTIES that aren't part of the ERD), constraints and more. Also, relations will be a part of the table design, because we need to connect the tables (By the RELATIONS in the ERD).

What is a Table?

- The design of a table will look like this:

Client

Name	Type	Size
ID	Long	
Name	Char	20
Address	Char	25
PhoneNumber	Char	12

Column

- The table's data will look like this:

Row

Client

ID	Name	Address	PhoneNumber
101	John Smith	102 3rd Ave., New York	212-555-1212
102	Rob Johnson	2005 Lake Dr., Chicago	847-432-1005
103	Hanna Robins	505 Main St., White Plains	914-946-8877

A table is a data holder which holds the data in rows. Every row is an entity inside the table. Tables have columns which have type and size. The type of the column will decide the data it can hold and the size will restrict this data (if we have restriction) to a firm size. These columns are the properties from the ERD.

The Process of Converting the ERD Into a Table Design

- ENTITY:
  - Convert each ENTITY into a table.
  - Convert the RELATION into a foreign key or a RELATIONS table.
- Columns:
  - Add columns to the table.
  - Set a primary key.
  - Set foreign keys.
  - Set the column's data type.

Just a checklist, everything will be explained later.

When we want to convert the ERD into table design we should start with the ENTITIES. As mentioned earlier, every ENTITY can be seen as a table. Every RELATION will be converted into a foreign key or a relation table which is a table that acts as a relation. The PROPERTIES will be added to the table as columns. After we have the table columns, it is time to decide what is the primary key (the key that is an identity of the entity), set the foreign keys and set the column data type. The choosing of a data type will be explained in details in the next chapters.

Converting ENTITIES Into Tables

- Each ENTITY will be converted into a table.
  - The table name will be in English, singular, with no spaces.
    - Example: Flight, Book, Customer, BookType
  - The ENTITY'S PROPERTIES will be columns in the table.
  - The column's name should be the PROPERTY name.
    - Example: Date, Name, Price.
  - Each entity instance (not ENTITY) is represented by a row.

102	Rob Johnson	2005 Lake Dr., Chicago	847-432-1005
103	Hanna Robins	505 Main St., White Plains	914-946-8877

When we convert an ENTITY into table we will start with the table name which will be in English, singular and with no spaces. After we have a table name it is time to convert the PROPERTIES into columns. Every column name should be called after the PROPERTY name. When we have the table design we will be able to insert data into the table as rows which are an entity (not ENTITY).

Tables

- The primary key:
  - Each table should have a primary key.
  - A primary key is one or more columns containing a unique value for each entity/instance.
    - Examples: passport number, license number, ID number.
  - If a column containing a unique value doesn't exist, add an extra column and call it TableNameID.
    - Example: OrderID for orders table.

After we have the table we need to decide which column (or columns) will be the primary key for the table. A primary key is a column that has a unique value for each entity and it will differentiate between each row in the table. Having a primary key will be useful for indexes (will be covered later) and for data queries. A primary key can be a compound key which means that it is built from more than one columns. Pay attention that sometimes it is better to have an ID column which isn't compound for better performance instead of compound primary key.

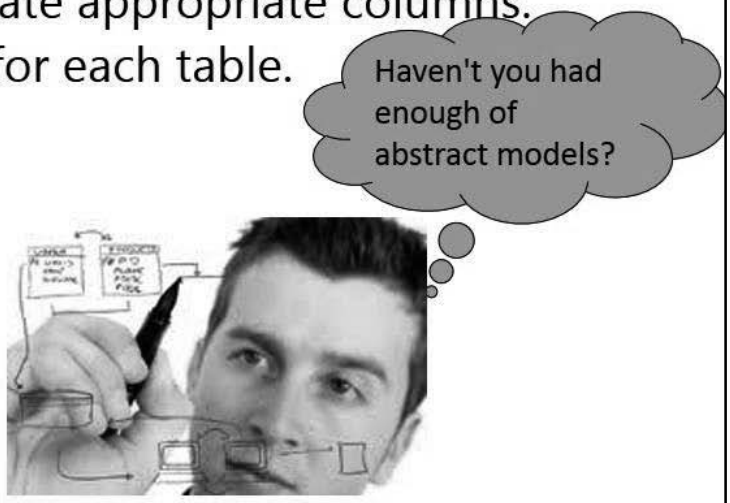
Tables (contd.)

- Additional columns are based on the organization's needs.

"Hello, it's your database developer speaking. I forgot to ask you what additional details you want to maintain for your customers besides name and telephone number !!"

Class Exercise 01

- Convert the following ENTITIES into tables.
- For each table, create appropriate columns.
- Set a primary key for each table.
  - Flight;
  - Course type;
  - Dish type;
  - Helpdesk call;
  - Music record;
  - Car;
  - Book copy;
  - Cellular phone call.



Converting Relations

- When converting a RELATION, we examine:
  - RELATION level (Unary, Binary, Ternary).
  - RELATION Functionality (1:1, 1:N, N:M).
- Each case will be translated in a different way:
  - Unary 1:1 - Unary 1:N - Unary N:M
  - Binary 1:1 - Binary 1:N - Binary N:M
  - ~~Ternary~~ 1:N
  - Ternary N:M
- For methodological reasons - Binary RELATIONS will be taught first.

Wrong!! if it's not N:M its not a ternary

Converting an ENTITY to a table is easy. Converting RELATIONS is more harder. When we want to convert a RELATION, we need to pay attention to the level and functionality of the RELATION. These two data pieces will enable us to understand how to build the relation in the table design model. The following are the different relations that we can convert:

- Unary 1:1
- Unary 1:N
- Unary N:M
- Binary 1:1
- Binary 1:N
- Binary N:M
- Ternary 1:N
- Ternary N:M

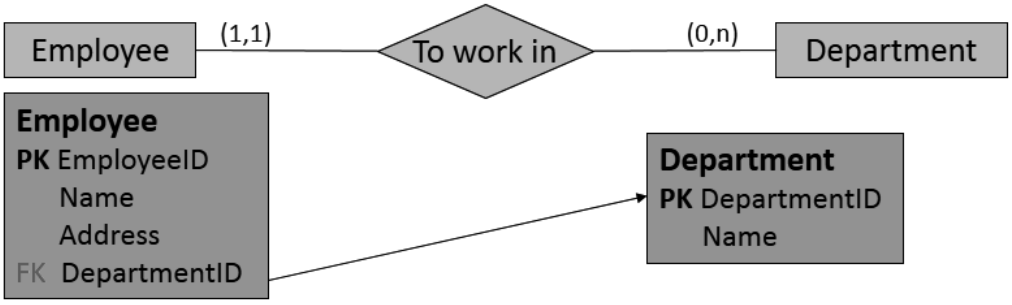
Binary 1:N

- Question?
  - How can we connect one row in the first table to many rows in a second table?
    - Example: an employee belongs to one department. One department has many employees. How can we connect John's row with the sales department's row?
- Solution:
  - We will add a column to the N side that will "point" to the equivalent row on the 1 side.
    - Example: John will have a column in his row with the value 800.
    - 800 is the sales department ID.

We start first with binary relations since they are the simplest. When we have a one to many relation between tables we do the following:
We add a column to the many side of the relation (in the above example to the employee table) that will hold the equivalent primary key of the row on the one side of the relation. After we do so we can establish a 1:N relation by adding a foreign key between the tables.

Binary 1:N (Implementation)

- The primary key of the table on the N side will be added as a new column on the 1 side.
- The new column will be designated as a foreign key



EmpID	Name	Address	DepartmentID
1	Nancy Davolio	507 - 20th Ave. E.	2
2	Andrew Fuller	908 W. Capital Way	1
3	Janet Leverling	722 Moss Bay Blvd.	2

DepartmentID	Name
1	Management
2	Sales
3	Engineering

Class Exercise 02

- Convert the following sentences to ERD diagrams
- Draw the ENTITIES and RELATIONS
- Mark the cardinality
- Translate the diagram into tables. Add columns
  - Dish Type to be from Category (Desert, Salads, Main Course…)
  - A Car is of one Car Type
  - A music record is from one Style (Classic, Rock, etc.)
  - A book copy is of one Book.
  - A cellular phone call is from one area (called CELL).

Binary N:M

- Question?
  - How can a row in the first table be connected to many rows in the second table, and at the same time, a row from the second table be connected to many rows in the first table?
    - Examples: A student takes many courses and a course is taken by many students.
    - John takes biology, math and computers. Biology is taken by Hanna, Sharon and Ron too.
  - The 1:N solution cannot be used in this situation.
- Solution:
  - Add a table, which will connect the 2 ENTITIES.

When we have a many to many relation between tables we do the following:
We add a new table to the table design that will connect the two ENTITIES in the RELATION. That table will contain one column that will point to the first entity and a second column that will point to the second entity. The pointers will be the primary key of the tables. After that we will create foreign key relation between these columns and the primary keys of the entities.

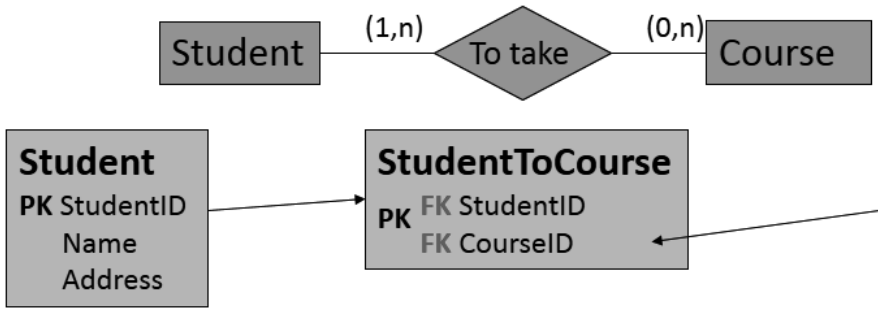
Binary N:M (contd.)

- The new table will contain:
  - A column pointing to the first ENTITY.
  - A column pointing to the second ENTITY.
  - Properties of the relation.

Binary N:M (contd.)

- Implementation:
  - Add a new table to the table design.
  - The table name will be a RELATION name or the names of the ENTITIES with to between them.
  - Primary keys from both ENTITIES will be added to the relation table. Each column will be a foreign key.
  - Column names will be: Relation table initials + primary key from the source ENTITY.

Binary N:M (contd.)



StdID	Name	Address
1	Nancy Davolio	507 - 20th Ave. E.
2	Andrew Fuller	908 W. Capital Way
3	Janet Leverling	722 Moss Bay Blvd.

StdID	CourseID
1	1
1	2
2	1
2	3
3	1

CourseID	Name
1	SQL
2	C#
3	Linq

Binary N:M (contd.)

- Add RELATION PROPERTIES as new columns to the RELATION table.
  - Example:

StudentToCourse
StudentID (FK)
CourseID (FK)
Year
Mark

- Primary key:
  - If you want the RELATION table to have a primary key - join two foreign keys to form a primary key.
    - Example:

StudentToCourse
StudentID (FK)
CourseID (FK)
Year
Mark

PK

If the RELATION has PROPERTIES (which means that the relation adds more data) we convert them into columns in the new created table. The relation table doesn't have to have a primary key (since it is a table that was created for relation purpose) but if you want to add a primary key you can use a compound primary key that will be built from the two foreign keys.

Class Exercise 03

- Convert the following sentences into diagrams.
- Draw the ENTITIES and RELATIONS.
- Mark the cardinality.
- Translate the diagram into tables, add columns.
  - A cook makes many types of dish.
  - A car is owned by many owners.
  - A music record contains many songs. A song can appear on many different records.
  - A book has multiple authors.
  - Many passengers take flights. A passenger can take several flights.

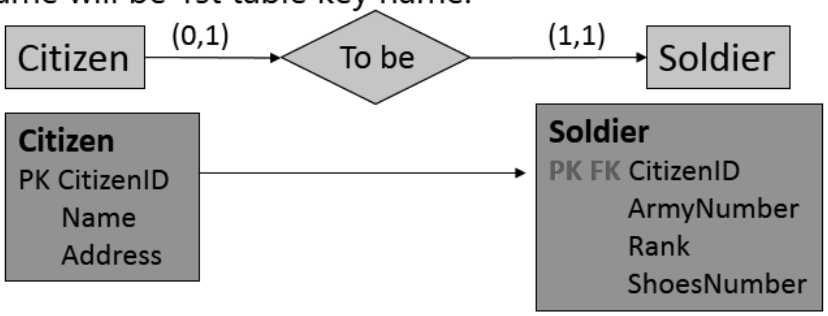
Binary 1:1

- A reminder: Why did we separate the ENTITY into 2 ENTITIES with 1:1 relations?
- Question:
  - How can we connect 1 row in the first table to 1 row in the second table?
    - Example: Each citizen could be one and only one soldier. Each soldier is not more than one citizen.
- Solution:
  - Decide which table is the “stronger” table and “weaker”.
    - Usually the RELATION between ENTITIES in a 1:1 RELATION will be (0,1) on one side, and (1,1) on the other. The (0,1) side is the “stronger” side.
  - Example: A citizen might be a soldier. A soldier must be a citizen.
  - Add a column to the “weaker” table, which points to the “stronger” table.

When we have a one to one relation, we will need to decide which side of the relation is the stronger side. The stronger side is the side that without it, the relation can’t have a meaning. In terms of cardinality, this means that the stronger side will be a (0,1) and the weaker side will be (1,1) (you also can look at it as a 0..1:1 relation). After we decide, which side is the stronger side, we add to the table of the weaker side a new foreign key column, which will point to the primary key of the stronger side.

Binary 1:1 (Implementation) Option A

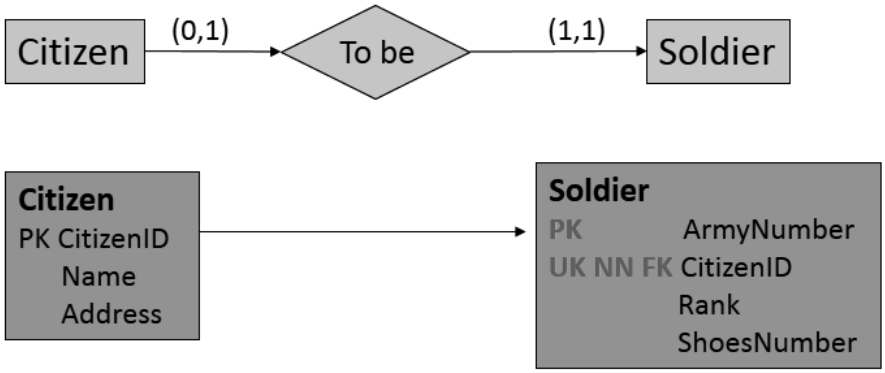
- The primary key of the "stronger" table will be added as a new column in the "weaker" table.
- The new column will be designated as a foreign key. The foreign key name will be 1st table key name.



CitizenID	Name	Address
0300000	Nancy Davolio	507 - 20th Ave. E.
0300001	Andrew Fuller	908 W. Capital Way
0300002	Janet Leverling	722 Moss Bay Blvd.

CitizenID	ArmyNumber	Rank	ShoesNumber
0300000	611111	sergeant	40
0300001	622222	commander	45
0300002	633333	sergeant	39

Binary 1:1 (Implementation) Option B



CitizenID	Name	Address
0300000	Nancy Davolio	507 - 20th Ave. E.
0300001	Andrew Fuller	908 W. Capital Way
0300002	Janet Leverling	722 Moss Bay Blvd.

ArmyNumber	CitizenID	Rank	ShoesNumber
611111	0300000	sergeant	40
622222	0300001	commander	45
633333	0300002	sergeant	39

Class Exercise 04

- Convert the following sentences into diagrams.
- Draw the ENTITIES and RELATIONS.
- Mark the cardinality.
- Translate the diagrams into tables. Add columns.
  - An employee can be a member of the company's basketball team.
  - Each line is connected to one cellular phone.
  - In SHIPS LTD., each ship has one specific line.
  - Separation for tax purposes: each order receives an invoice.

Ternary (or more) N:M

- This is very similar to the Binary N:M.
- We cannot store the RELATION in one of the ENTITIES table.
- Therefore:
  - We must add a table that will connect the 3 ENTITIES.
  - The table will contain:
    - A column pointing to the first ENTITY.
    - A column pointing to the second ENTITY.
    - A column pointing to the third ENTITY.
    - PROPERTIES of the RELATION.

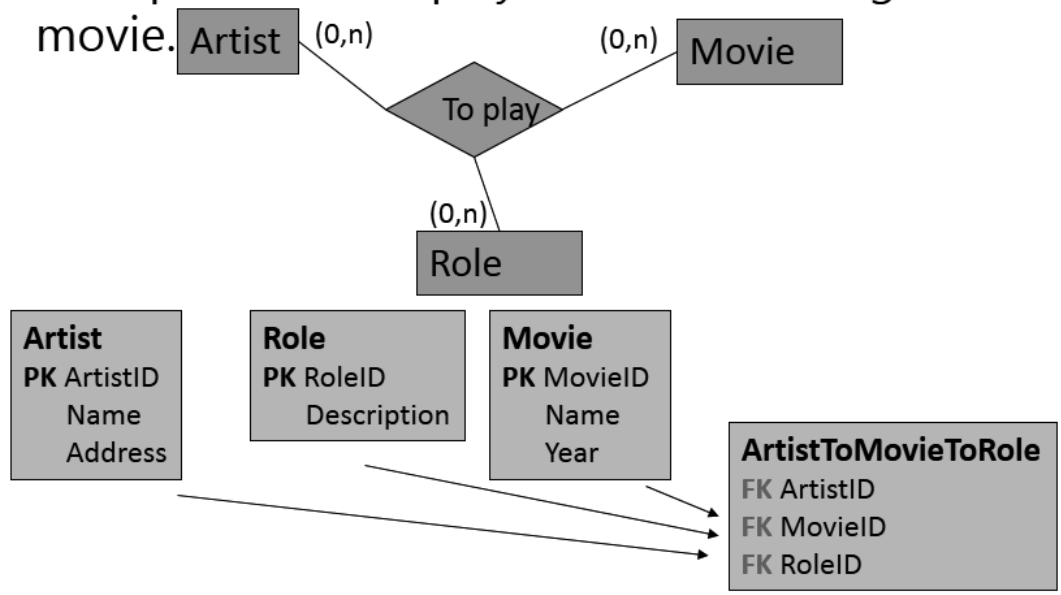
A ternary of N:M will be treated as a binary N:M, but we will add one more foreign key column to the added relation table in order to imply the connection to three ENTITIES.

Ternary (or more) N:M (contd.)

- Implementation:
  - Add a new table to the table design.
  - The table's name will be the RELATION name or the names of the 3 ENTITY with To between them.
  - Primary keys from each of the 3 ENTITIES will be added to the relation table.
  - Each column will be a foreign key.
  - Column names will be: the primary key from the source ENTITY.

Ternary (or more) N:M – cont'd

- Example: An artist plays a role in making a movie.

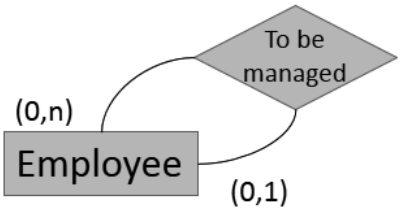


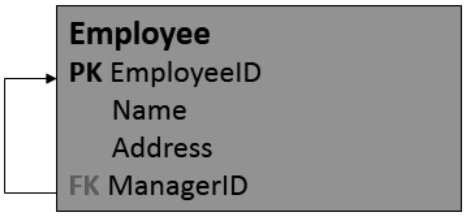
Class Exercise 05 - Ternary

- Convert the following sentences into diagrams.
- Draw the ENTITIES and RELATIONS.
- Mark the cardinality.
- Translate the diagrams into tables, add columns.
  - An employee accepts a position (team leader, programmer, debugger, etc.) in a certain project.
  - A cook makes a dish type for an order.
  - A certain doctor (in the hospital) diagnoses a diagnosis type for a certain patient (hopefully, just the flu... AAAA-CHOOO!!!)
  - The mechanic fixes different problem types in a car.

Unary 1:N

- The primary key of the table will be added as a new column to the same table.
- The new column will be designated as a foreign key.
- The foreign key name will be the RELATION meaning + ID.





EmpID	Name	Address	MngID
1	Nancy Davolio	507 - 20th Ave. E.	
2	Andrew Fuller	908 W. Capital Way	1
3	Janet Leverling	722 Moss Bay Blvd.	1
4	Robert King	28 Hollow Winchester	2

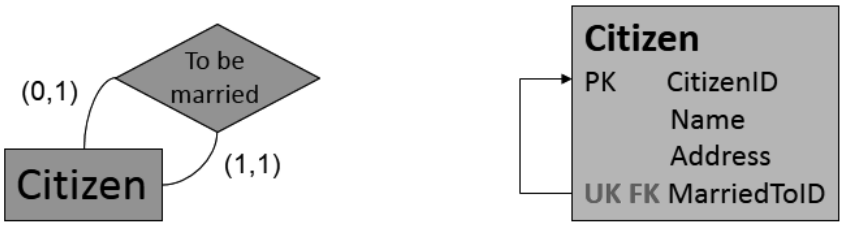
When we have a unary relation, this means that the ENTITY is connected to itself. To add a unary 1:N relation, we will add a new column to the table which will be a foreign key to the primary key of the table and that's it.

Class Exercise 06 - Unary 1:N

- Convert the following sentences into diagrams.
- Draw the ENTITIES and RELATIONS.
- Mark the cardinality.
- Translate the diagrams into tables, add columns.
  - A priest is ordained by another priest.
  - A book could be a translation of another book.
  - A product could be a component of a different product.
  - An ingredient type in the recipe could be replaced by another ingredient.

Unary 1:1

- The primary key of the table will be added as a new column in the same table.
- The new column will be mentioned as a foreign key.
- Foreign key name will be: RELATION meaning + ID.



CitizenID	Name	Address	MarriedToID
1	Nancy Davolio	507 - 20th Ave. E.	
2	Andrew Fuller	908 W. Capital Way	3
3	Janet Leverling	722 Moss Bay Blvd.	2
4	Robert King	28 Hollow Winchester	

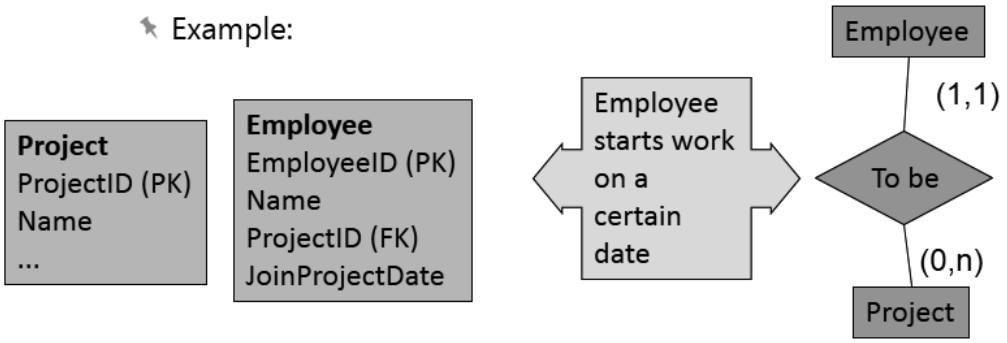
Unary 1:1 resembles the unary 1:N relation in its creation. We add a new foreign key to the table which will point to the table's primary key.

Class Exercise 07 - Unary 1:1

- Convert the following sentences into diagrams.
- Draw the ENTITIES and RELATIONS.
- Mark the cardinality.
- Translate the diagram into tables, add columns.
  - Each project can be a continuation of a different project.
  - An order can be an extension of a previous order.
  - A program is an upgrade/new version of a different program.

From ERD to Tables - Summary

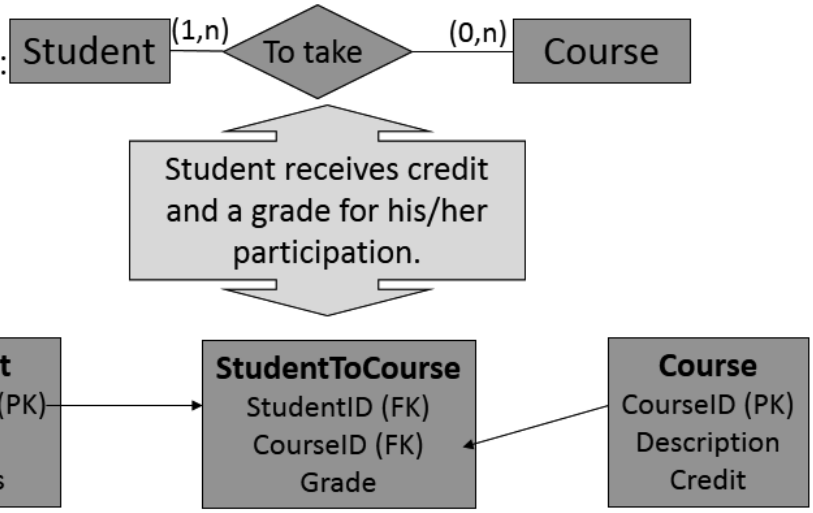
- An ENTITY'S PROPERTIES:
  - Become columns in the ENTITY table.
- A RELATION'S PROPERTIES:
  - In 1:N -
    - PROPERTIES become columns in the "one" table.
      - Example:



We saw in the chapter, how we transform an ERD to table design. The way is not hard, but it can lead to poor design if you don't follow the rules. We start converting an ENTITY into a table, the ENTITIES' properties into columns. Then we transform RELATIONS into table connections as we have seen earlier. Few things that weren't mentioned; what to do when a 1:N RELATION has PROPERTIES? The answer is simple. Add the PROPERTIES as columns in one of the tables.

From ERD to Tables - Summary (contd.)

- In N:M -
  - PROPERTIES become columns in the RELATION table.
    - Example:



When we have a N:M relation which holds PROPERTIES, we add them to the created relation table.

From ERD to Tables - Remarks

- The purpose of the foreign key:
  - To ensure that users do not insert unknown data into foreign key columns.
    - If a foreign key exists between employee and department, then the employee would not be able to insert a department number which doesn't exist in the department table.
- Calculated columns:
  - Calculated columns such as, age (current date - date of birth), or order totals should not be stored in the database except for special cases (performance or update).

Other things to note about table design:
The purpose of the foreign keys is data integrity. That means the foreign keys ensure that a user won't be able to insert unknown data. Unknown data is data that doesn't exist in the tables that are constraint by the foreign key.

Calculated columns should not be a part of a database. When we need a calculated column, it is probably a part that needs to sit in an application that uses the database. The only special case for that is when we need performance on the calculated column which can be achieved in the database.

From ERD to Tables - Remarks (contd.)

- When does a PROPERTY turn into a table?
  - In the ERD: When a PROPERTY becomes an ENTITY:
    - Alphanumeric and...
    - Has a changeable list of values and ...
    - We want to sort or filter by that PROPERTY.
  - "One UOD PROPERTY is another UOD ENTITY":
    - City is just another property in the STUDENT ENTITY, if we are speaking about the UOD of a University.
    - City is an ENTITY (an important one at that), if we are discussing UOD of the Mail service.

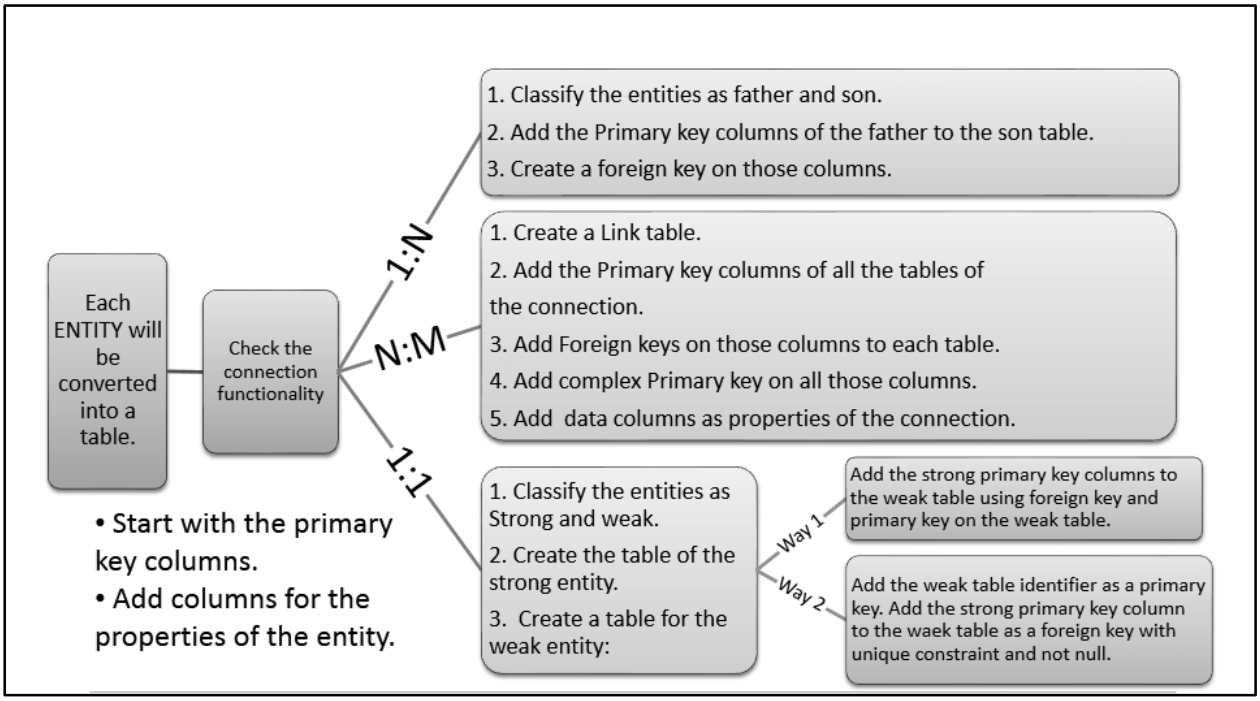
Module 3

In chapter 3, we learnt the situations that enable us to turn a PROPERTY into a table. If you don't remember the reasons you should return to chapter 3 to further investigate.

From ERD to Tables - Remarks (contd.)

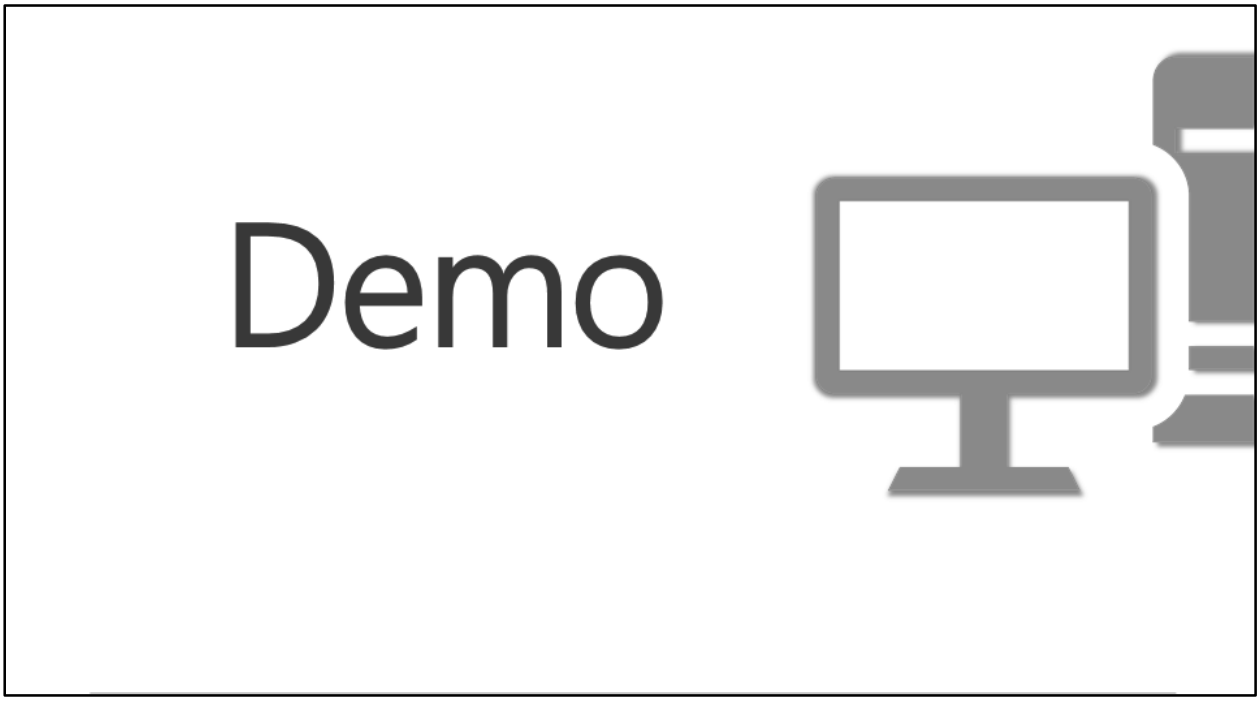
- When does a PROPERTY turn into a table? (contd.):
  - The PROPERTY has a 1:N or N:M relation with the ENTITY.
    - A stock has a daily value.
    - The product type has many colors.
  - We want to store the history of that PROPERTY.
    - Each employee receives a different salary each month.
  - The PROPERTY has its own independent properties (and the organization wants to maintain that data).
    - A diver has a 1,2,3,m stars diving license (I want to store independent data on each license type)
- "Intentional" duplications:
  - The recommended price vs. the actual price or order date vs. delivery date might be stored (sometimes in different tables).

Summery



- In 1:1 connection if we cannot classify the entities as Strong and weak we have to decide which table will point to the other and use the second way without the not null constraint.

Transition to the Table Model
Class Examples

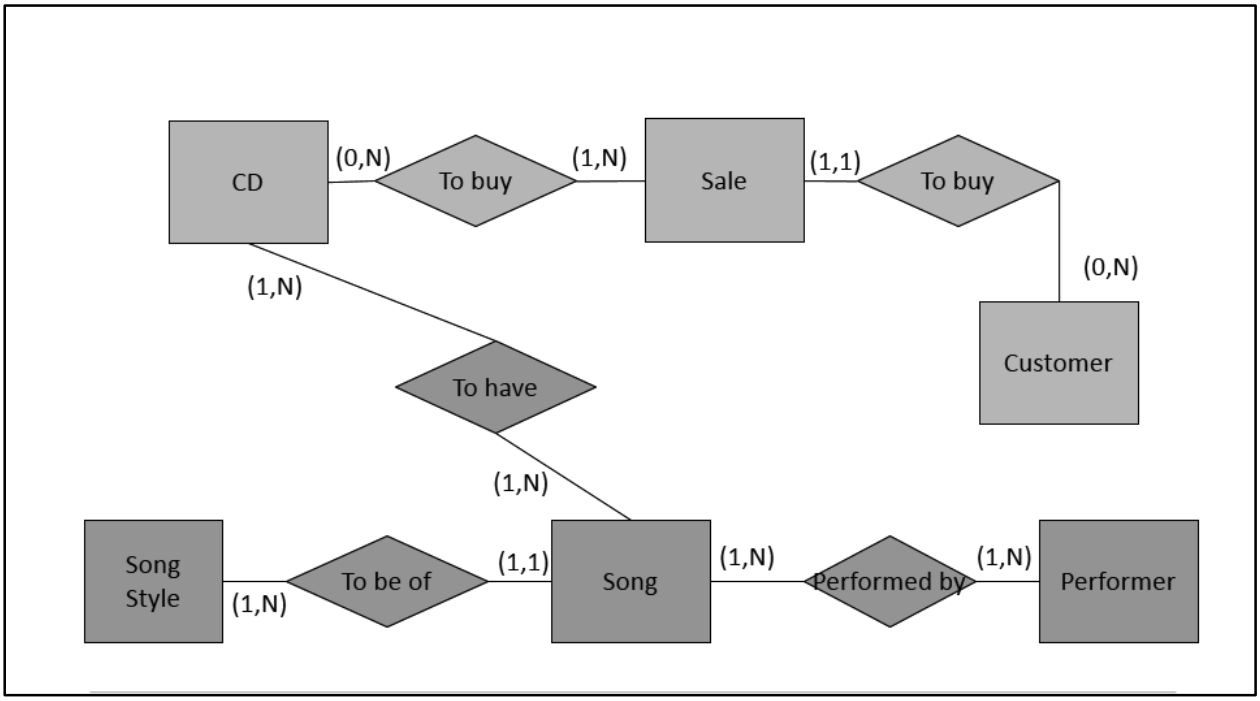


Class Example 1: “See This”

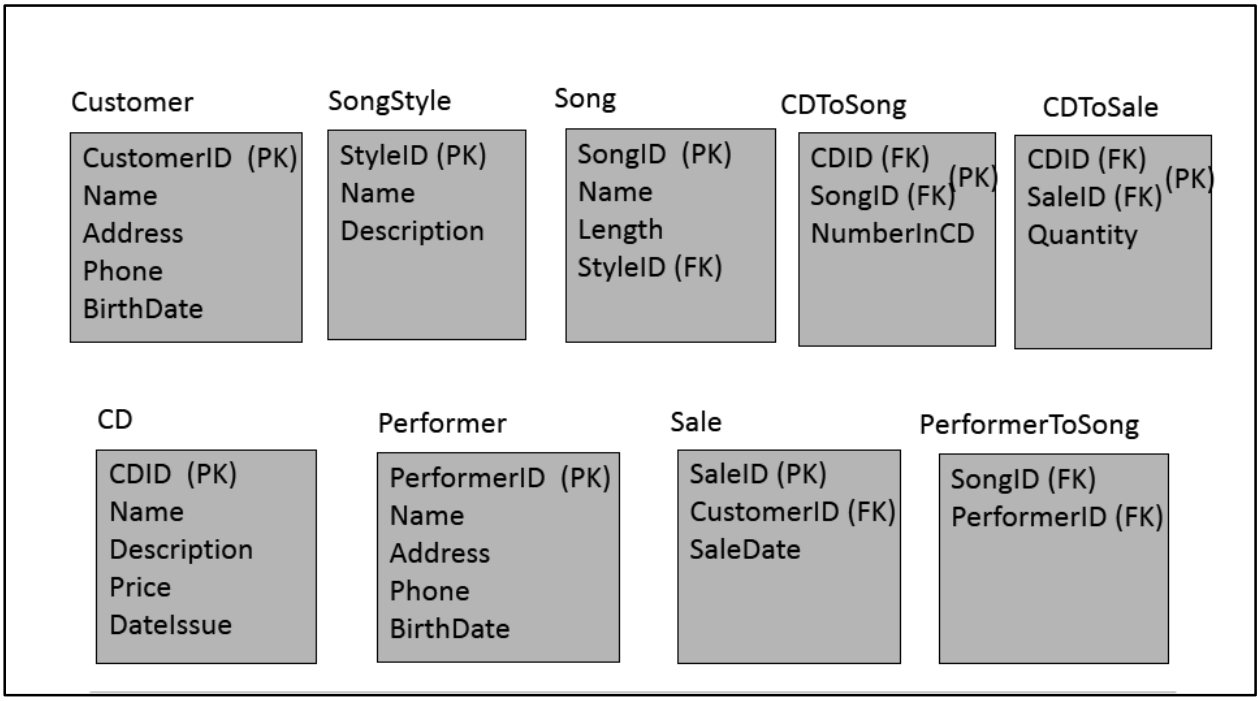
- In the CD store there are many CDs.
- Each CD contains various songs (“Let it Be”, “Hey Jude”, etc).
- Each song can appear on many CDs.
- Each song is of a song style (rock, pop, classic, etc.).
- Each song is performed by a certain performer or performers.
- A Sale is made by a customer.
- Is a sale, customers buy CDs .

Remark: We want to store the song style and performer/s for each song, to make CDs such as collections and soundtracks.

Solution - Class Example 1: “See This”



Transition to the Table Model 1: "See This"



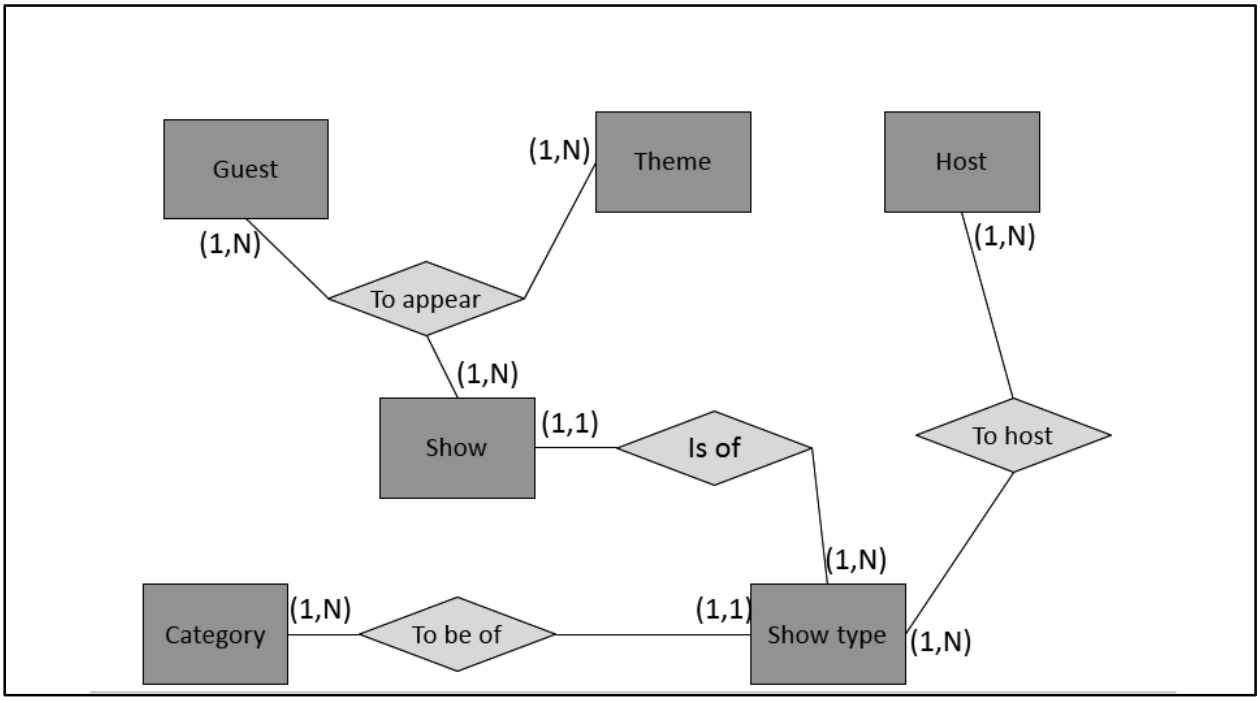
Class Example 2: "Tell-Vision"

- There are many TV shows on the Tell-Vision TV network.
- Each show is of a show type (60 Minutes, Friends, E.R., Jay Lenno, etc.).
- Remark: Each show is a scheduled show type.
- In each show type, there are one or more hosts.
- Guests appear in the shows with relation to a theme (Art, Politics, economy, etc.).
- Remark: Each guest can appear in relation to several themes.

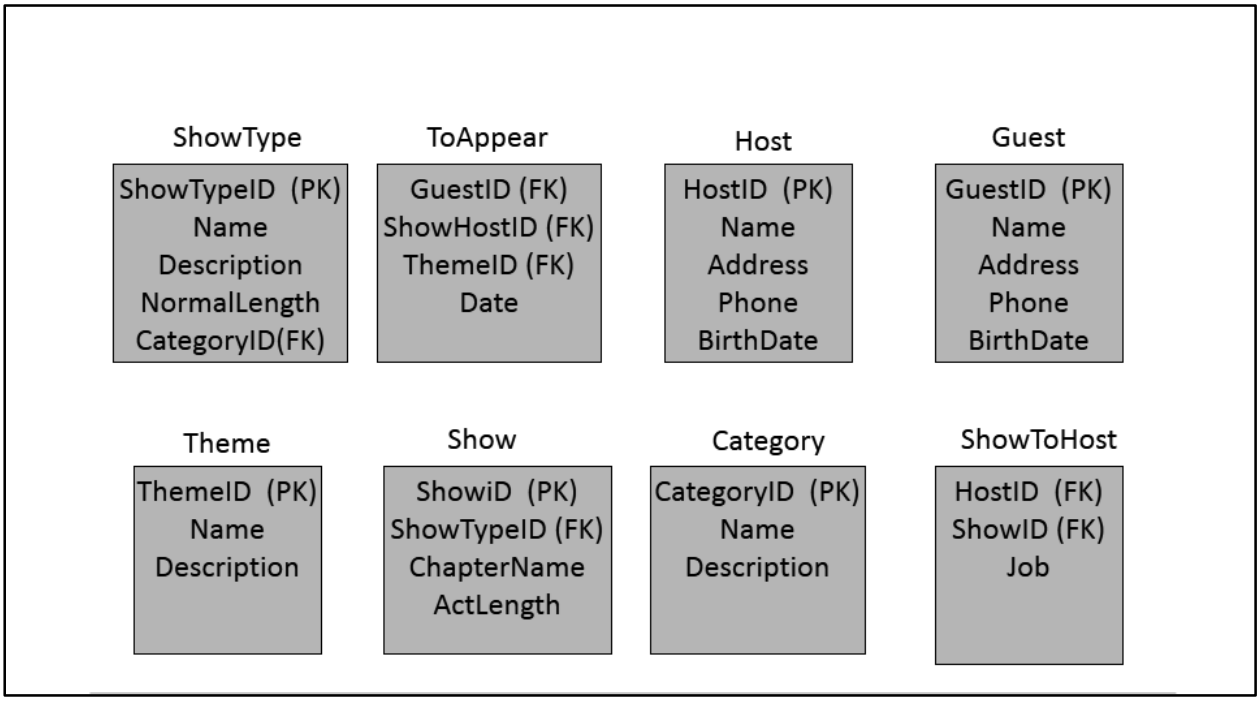
Class Example 2: “Tell-Vision” (contd.)

- Each show type is of a different category (game show, news, talk show, sitcom, drama, etc.).

E.R.D. - Class Example 2: “Tell-Vision”



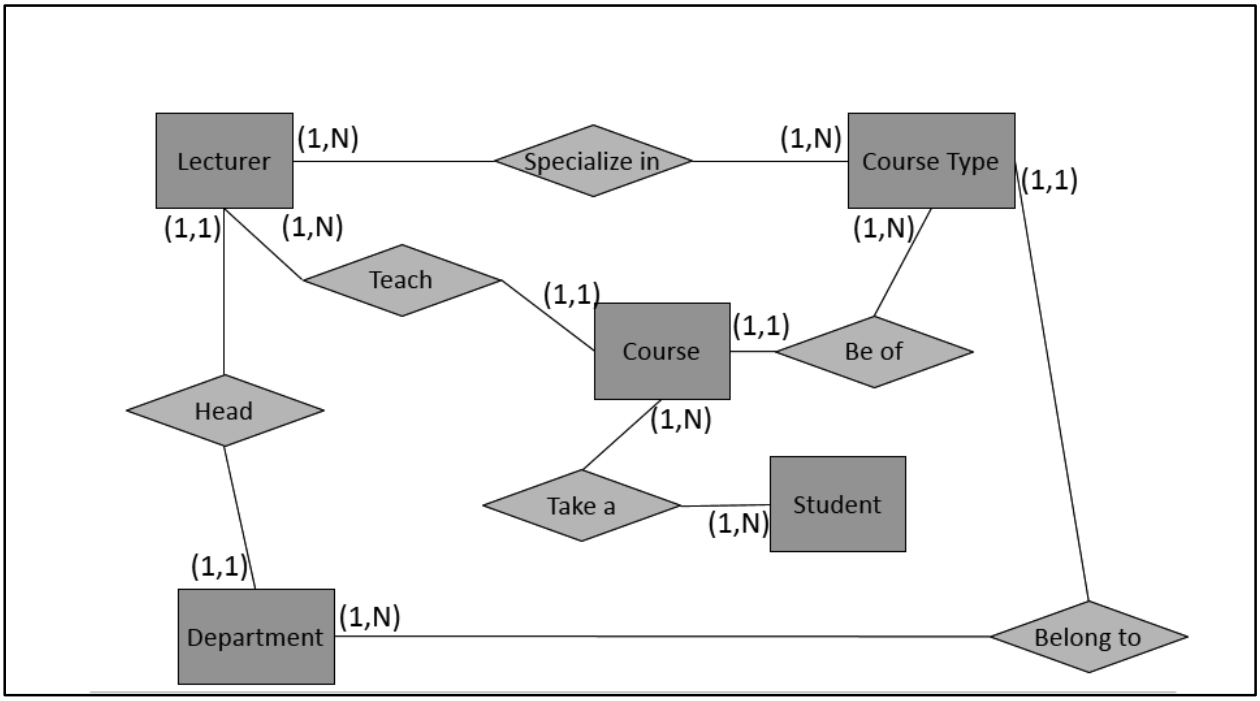
Transition to the Table Model 2: "Tell-Vision"



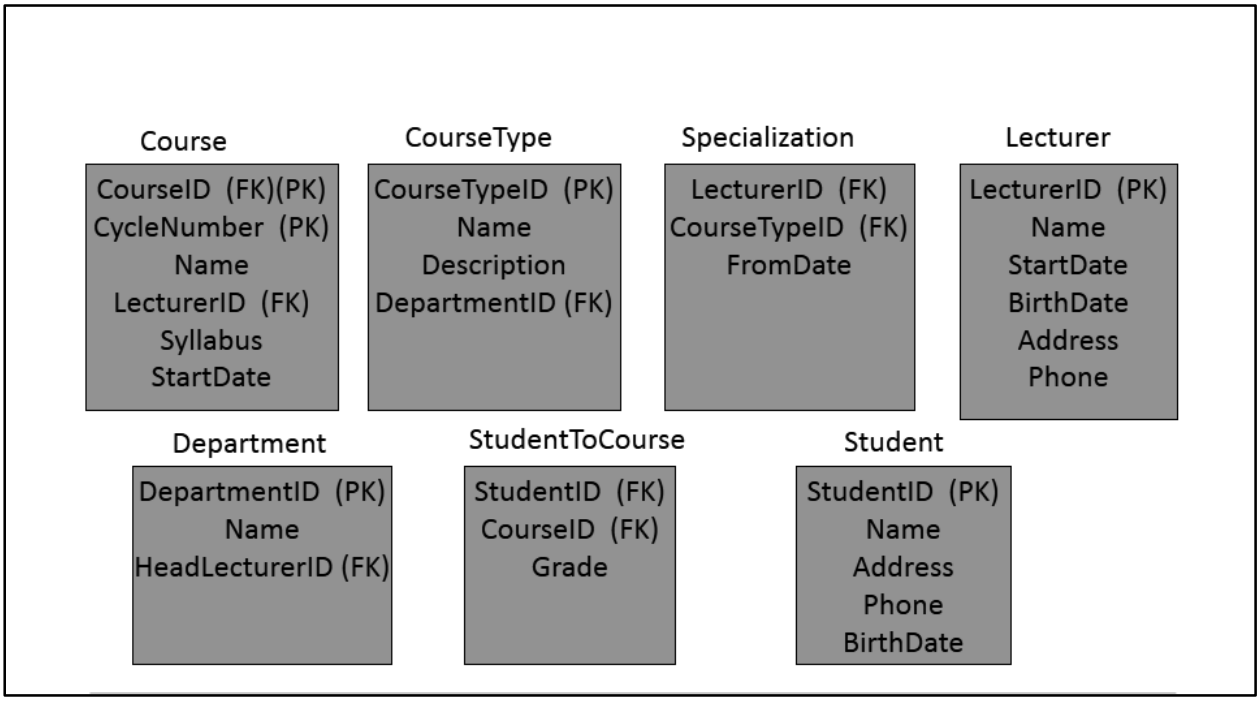
Class Example 3: DeepLawMass”

- The DLM Law Academy has courses, lecturers and students.
- A student takes several courses.
- A course is of a course type (criminal law, matrimonial law, commercial contracts, corporate law, etc.), and has a start date.
- A course is taught by a single lecturer.
- A lecturer specializes in a course type/s.
- A course type belongs to a certain department.
- One lecturer heads each department.
- Remark: lecturer heads no more than one department.

E.R.D. Class Example 3: DeepLawMass”



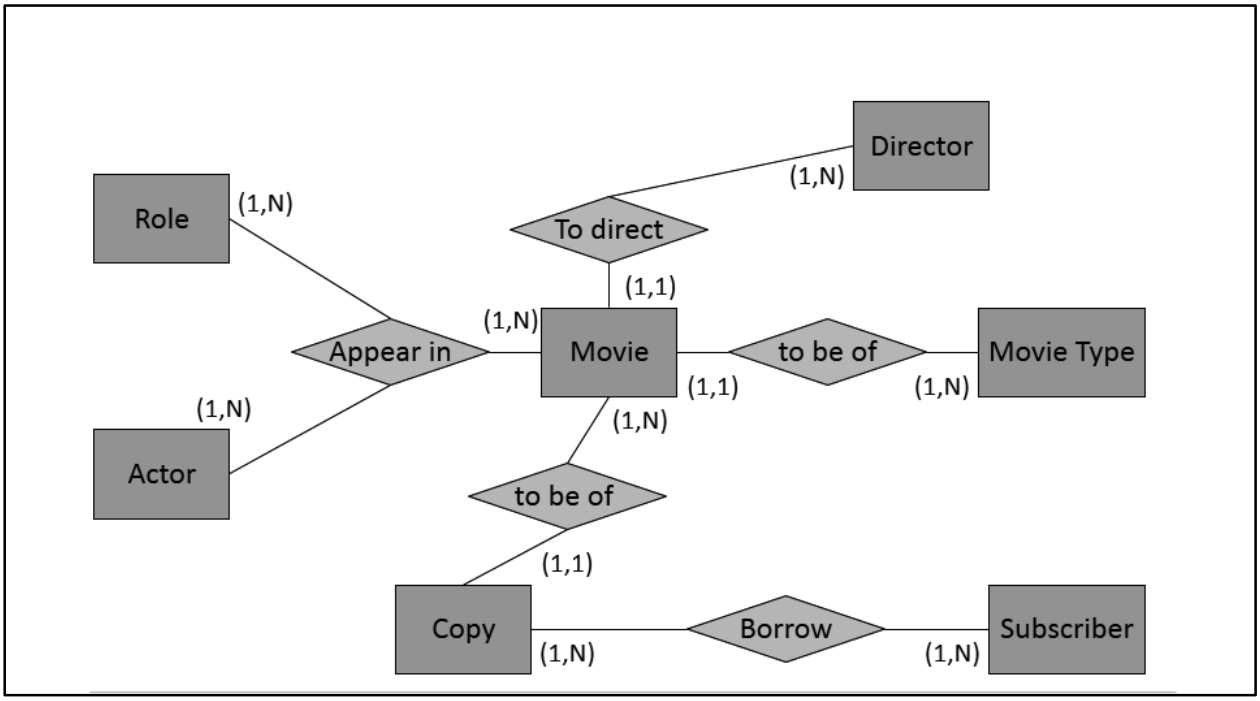
Transition to the Table Model 3: "DeepLawMass"



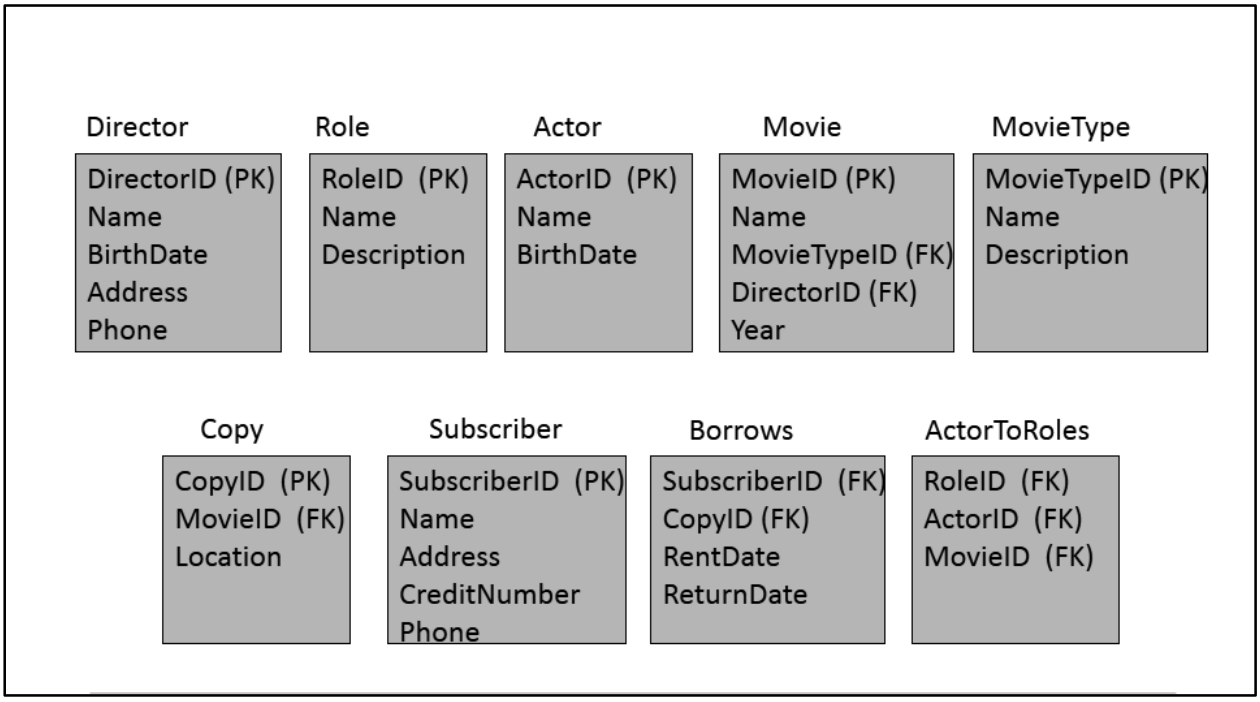
Class Example 4: "Joe's Movies"

- There are many movies in Joe's video rental store.
- Each movie is of a movie type (drama, action, comedy, etc.).
- Actors appear in movies, each actor plays a role/s in each movie (star, co-star, extra, stuntman, etc.).
- Each movie was directed by one director during a specific year.
- A subscriber borrows a copy or copies of a movie/s on a certain date and returns it/them on a certain date.

E.R.D. Class Example 4: "Joe's Movies"



Transition to the Table Model 4: "Joe's Movies"



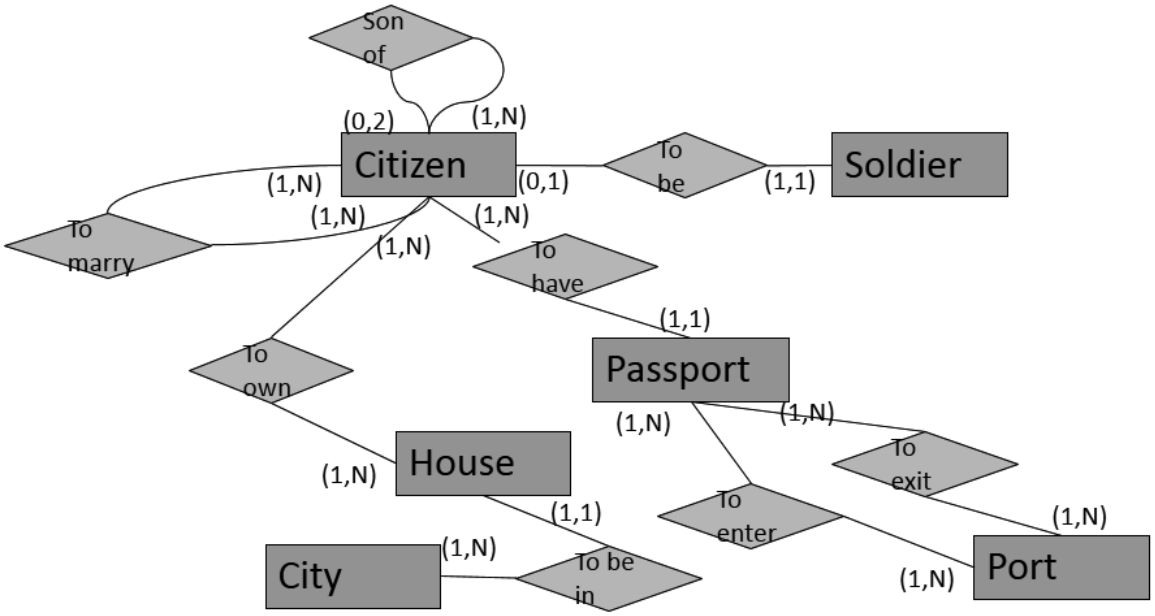
Class Example 5 : "Banana Republic"

- There are many citizens and soldiers in the Republic.
- A citizen can be a soldier.
- A soldier has an id, a rank and a service start date.
- A citizen has a mother and a father (who are also citizens).
- A citizen can be married to another citizen.
- A citizen can get divorced and remarry.
- A citizen can own a house or houses.
- A house is in a certain city.

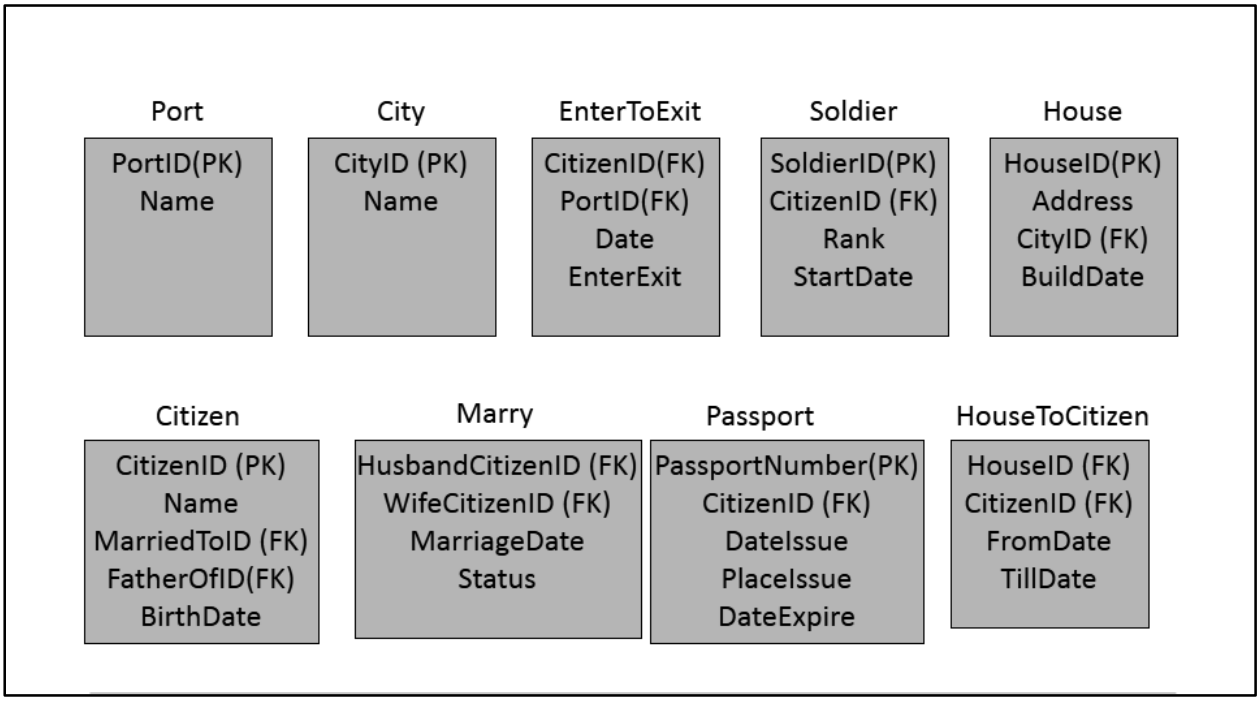
Class Example 5 :
"Banana Republic" (contd.)

- A citizen can have many passports (only the Republic's), but only one of them is valid (the others have expired).
- A citizen can use his valid passport to enter and exit the country from different ports.

E.R.D. - Class Example 5: "Banana Republic"



Transition to the Table Model 5: "Banana Republic"



Lab: Transition to the Table Model
Class Exercises

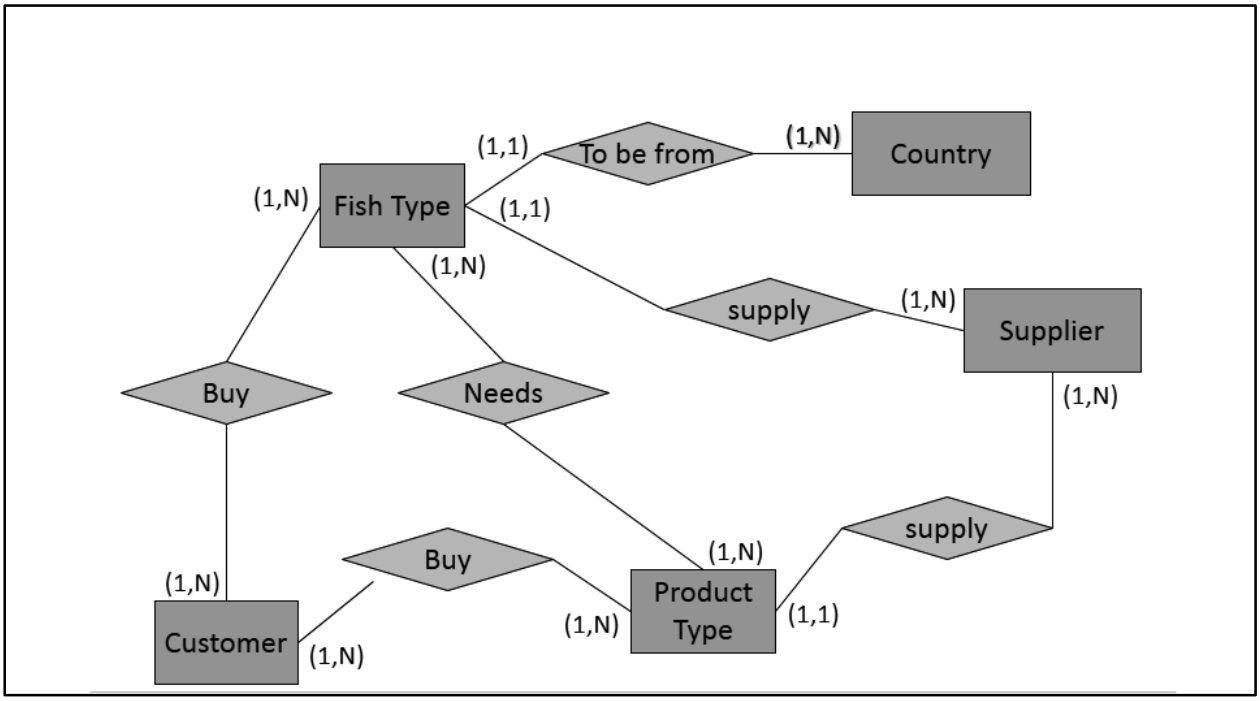
Class Exercise 1: “Fishy Beauty”

- The fish store has various products and fish.
- A product is of a product type. (air pump 750, dry food 500g, etc.).
- A fish is of fish type (goldfish, tiny shark, guppy, etc.).
- Remark: no data is stored on each product or fish. The store maintains data on product types and quantities of fish type in stock.
- Each product type is supplied by one supplier.
- A product type has a certain size, price and weight.
- Customers buy products and fish.

Class Exercise 1: "Fishy Beauty" (contd.)

- Remark: In order to simplify matters, a fish and a product type are not the same ENTITY.
- Each fish type is supplied by one supplier.
- A fish type is from a certain country.
- A fish needs certain product types to survive (certain food or the presence of a certain product/s).

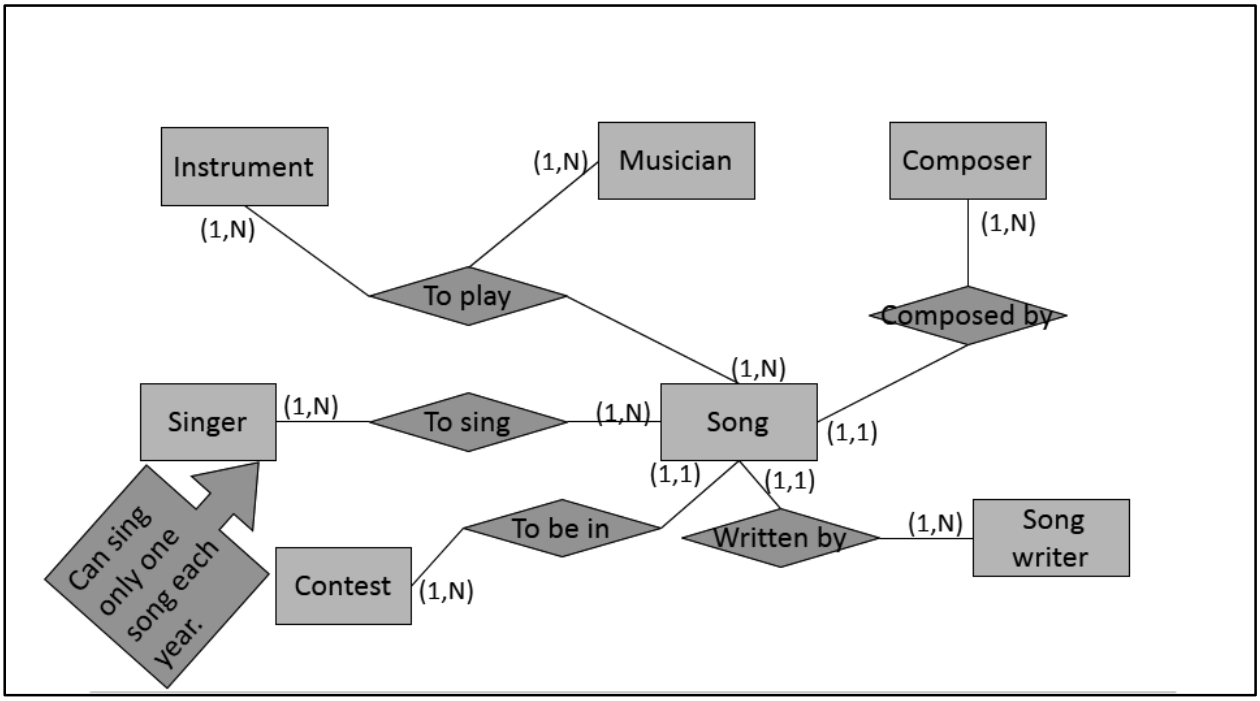
Class Exercise - “Fishy Beauty” ERD1



Class Exercise 2: The "EUROVISION"

- In the annual European song contest there are many songs, each one representing a different country.
- A song is sung by a singer or singers.
- A singer can sing only one song each year.
- Musicians play various types of instruments in a song.
- A musician can play different instruments in different songs.
- Each song's lyrics are written by a songwriter.
- Each song is composed by a music composer.
- There is only one winner in each contest.
- Each singer is either a lead singer or a backup singer.

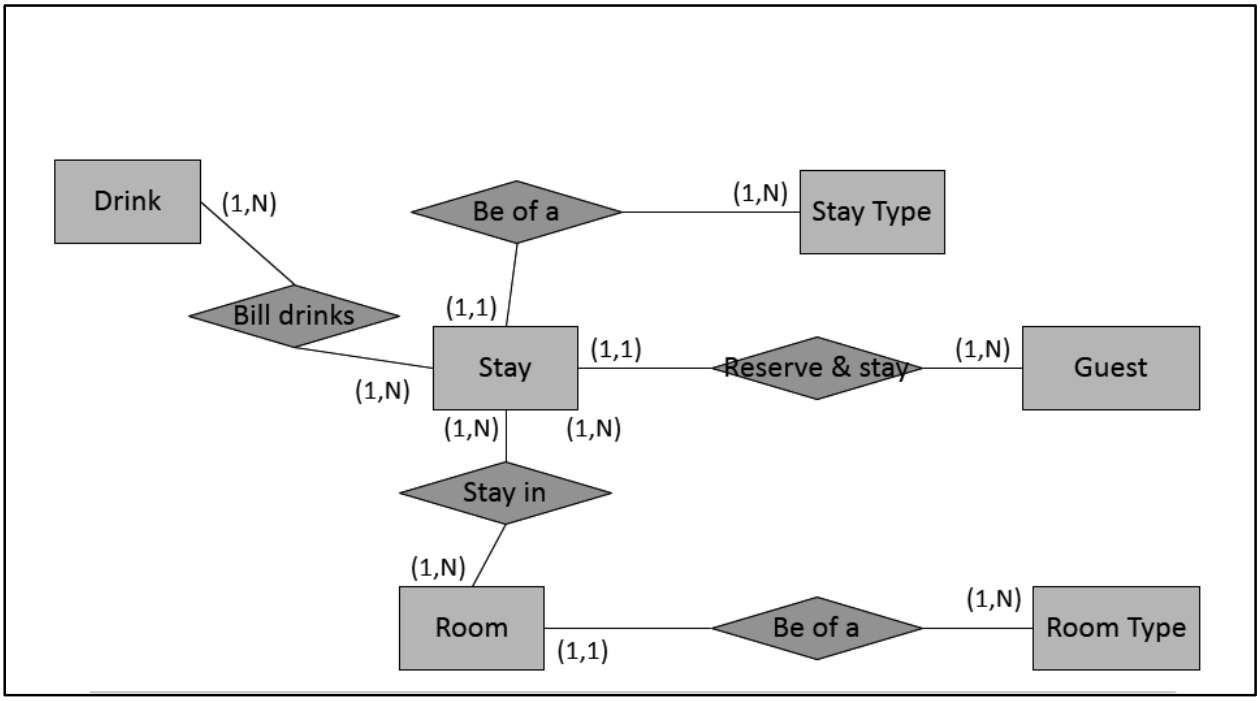
Class Exercise - “EUROVISION” ERD 2



Class Exercise 3: "Hotel California"

- Guests stay at the hotel on specific dates.
- A guest stays in a room or rooms.
- A stay is of a stay type (B&B, half board, full board, etc.).
- During a certain stay, guests order drinks.
- A room has a number of beds of type single, double or child (no other types).
- The room is of a certain type (regular, suite, etc.).
- A guest makes a reservation for a certain period.
- Remark: reservations and stay might not be on the same date.

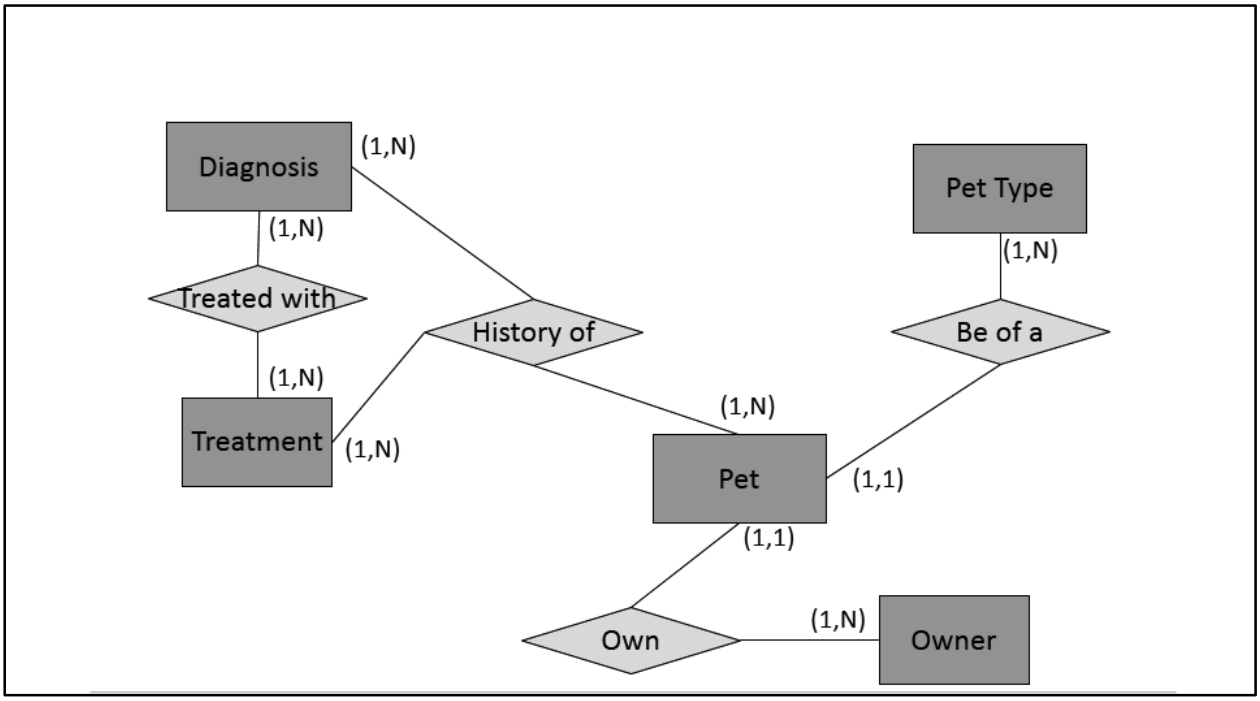
Class Exercise – "Hotel California" ERD 3



Class Exercise 4: "Dr. Do-little"

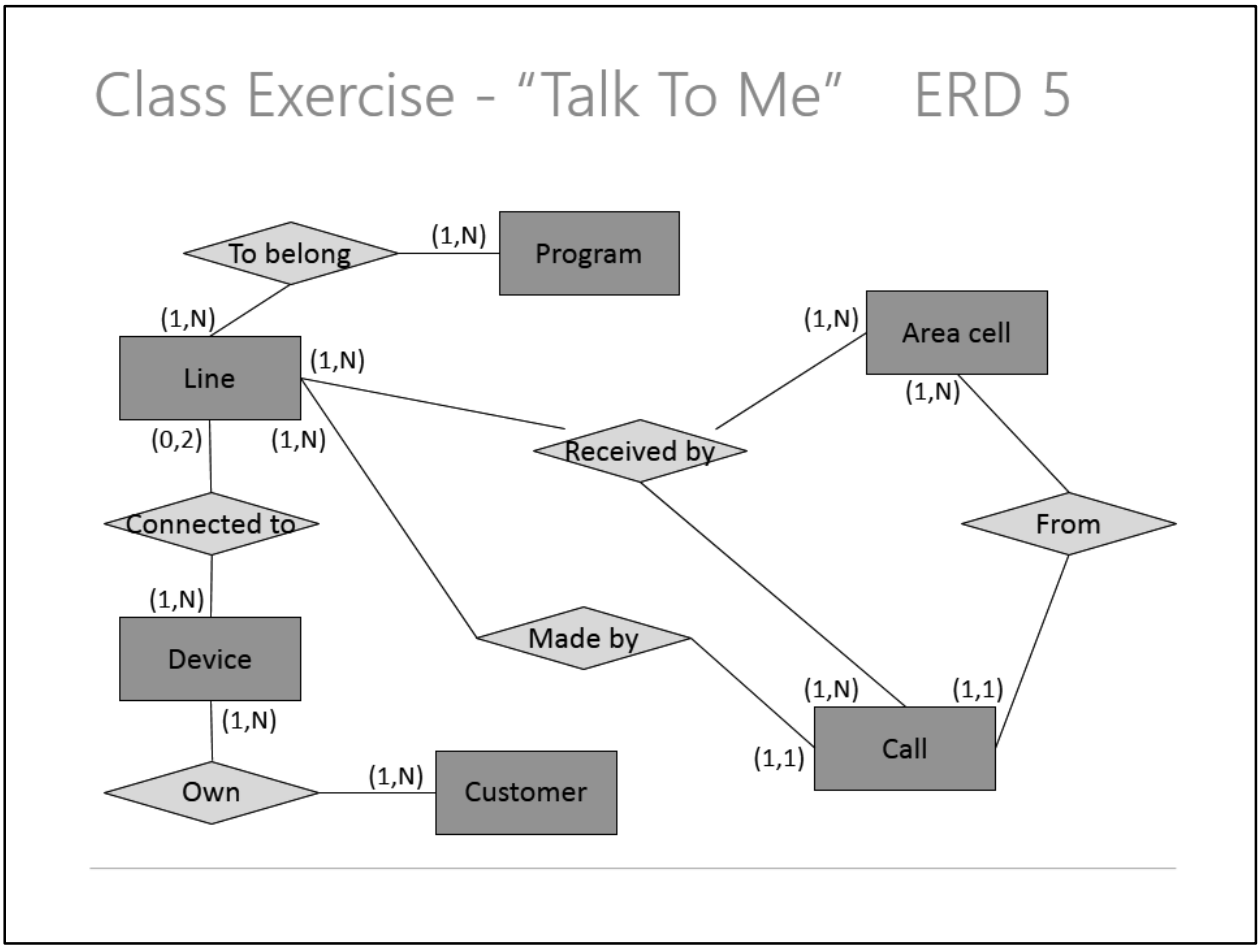
- The Dr. diagnoses pets.
- People own pets, pets are owned by 1 owner.
- A diagnosis is given to an owner for its pet .
- Several treatments are given for a diagnosis.
- One treatment could be given for several diagnoses.
- A pet has a history of treatments for all diagnoses.
- A pet (cat named "Mitzi", dog named "Goofy", etc.) is of a certain pet type (bull-terrier, Siamese, viper, boxer, etc.).

Class Exercise - “Dr. Do-little” ERD 4



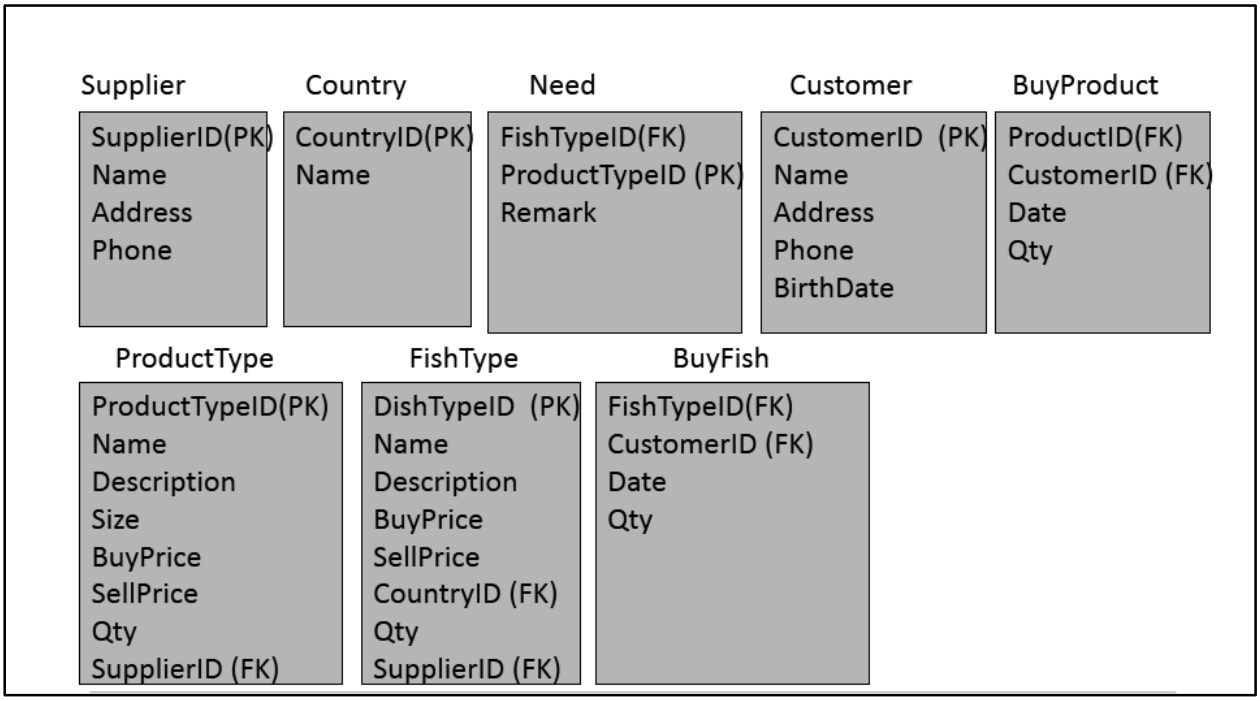
Class Exercise 5: "Talk To Me"

- There are many calls in the TTM Cellular Phone Services company.
- Customers own cellular devices.
- A line is assigned to 1 or 2 devices maximum (mobile and car phone).
- A call is made from an 'area cell' and from a certain line.
- A call is made to line/s that are in an 'area cell'/s (between 2 lines or conference calls).
- A line belongs to a program (students, one line for half price, leisure time, etc.).

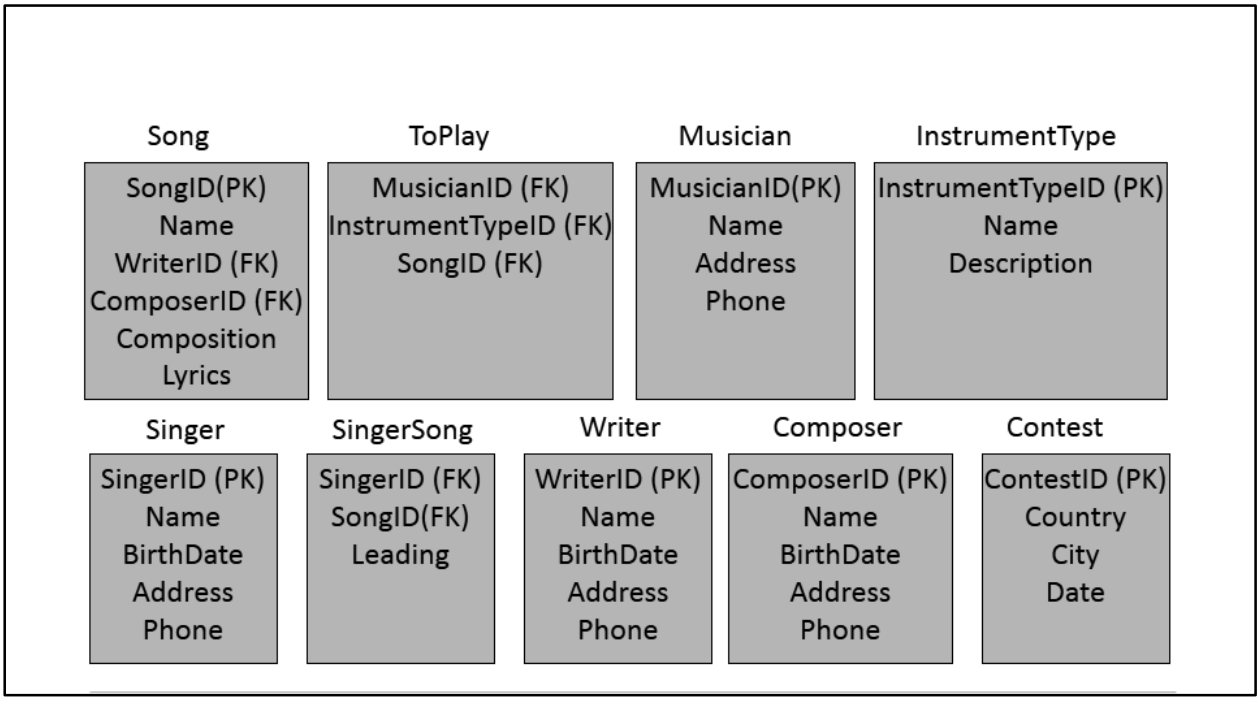

Class Exercise - "Talk To Me" ERD 5
To belong
(1,N)
Program
(1,N)
Line
(1,N)
(1,N)
Area cell
(1,N)
(0,2)
(1,N)
Received by
Connected to
From
(1,N)
Device
Made by
(1,N)
(1,1)
(1,N)
Call
(1,1)
Own
(1,N)
Customer

Lab: Transition to the Table Model: Class Exercise Solutions

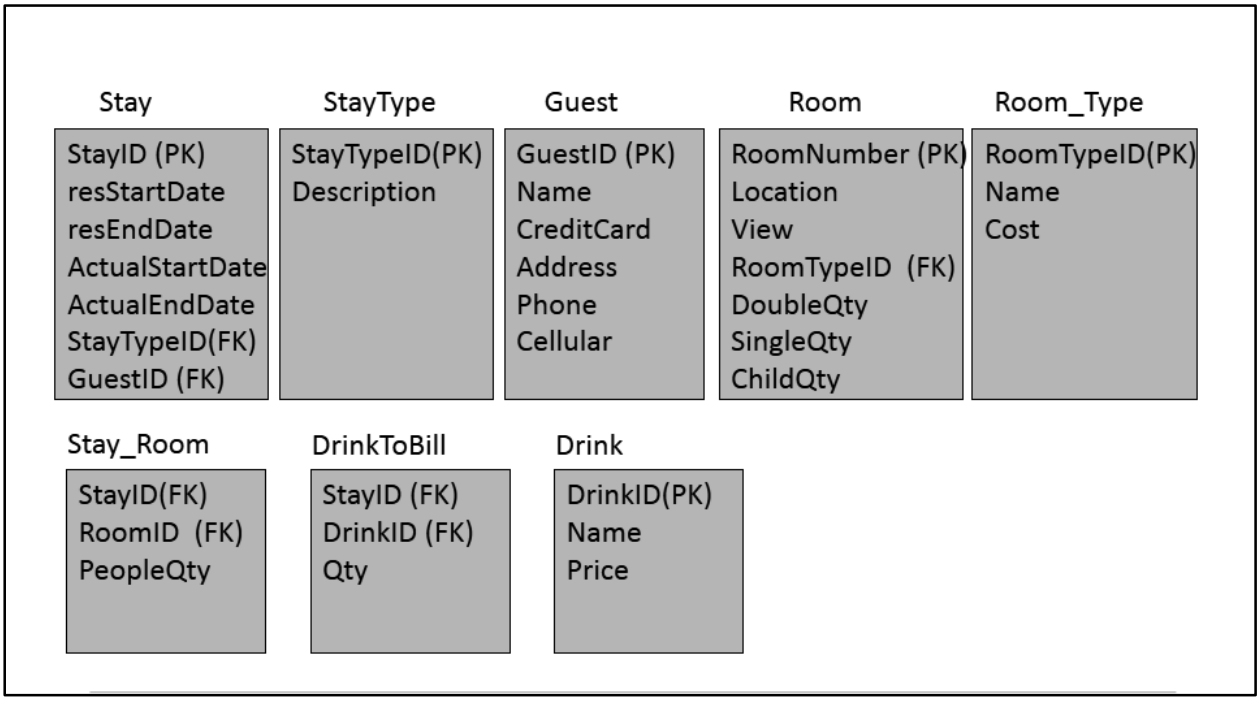
Class Exercise 1 “Fishy Beauty” Solution:



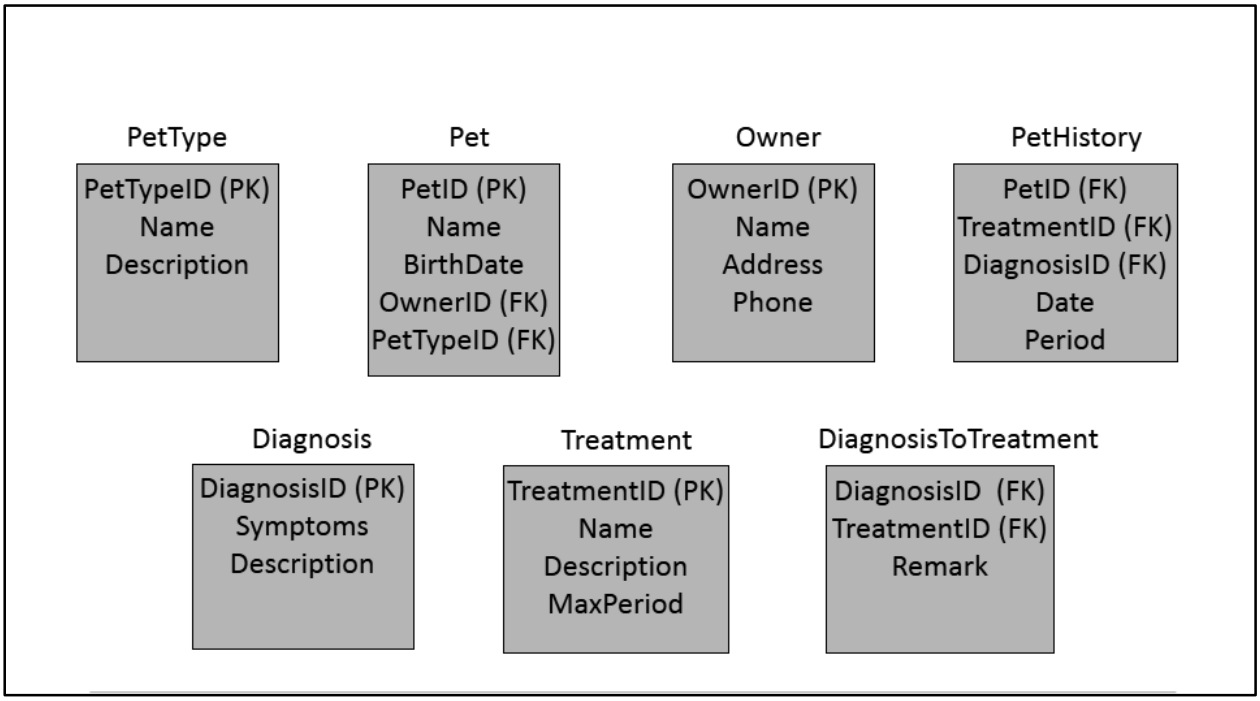
Class Exercise 2 “EUROVISION” Solution:



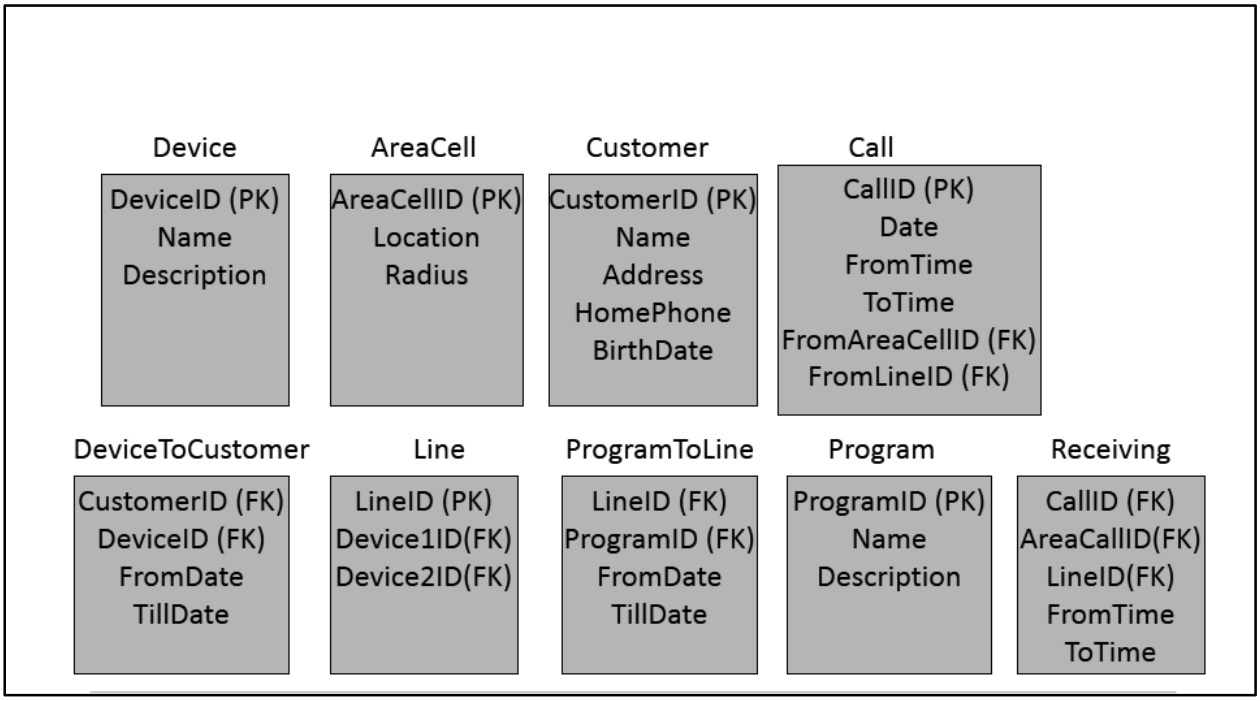
Class Exercise 3 “Hotel California” Solution:



Class Exercise 4 “Dr. Do-little” Solution:



Class Exercise 5 “Talk To Me” Solution:



Questions

Module 05 - The Select Query

Contents:

Chapter Goals 6
What is SQL? 7
SQL's Groups of Statements 8
SQL Language Elements 9
Database (DB) Management Environment 10
Class Example: Northwind - UOD 11
Class Example: Northwind – ERD 12
Class Example: NorthWind - Table Design (schema) 13
Select - The Logical Idea 14
Select - The Logical Idea (contd.) 15
The Select Command 16
The Select Statement 17
The Select Statement (contd.) 18
The Select Statement 19
Select - Caption 20
From 21
From (contd.) 22
Sql Server Functions 23
Common String Functions 24
Common String Functions 25
Common Date & Time Functions 26
Common Date & Time Functions 27
Common Date & Time Functions 28
Common Mathematical Functions 29
The Where Clause 30

The Where Clause (contd.) 31
Where, And, Or 32
Order By 33
Class Example - Select Statement 34
Class Example - Select Statement (contd.) 35
Class Exercise - Select 36
Join Tables 37
Join Tables (contd.) 38
Join Table - Examples 39
How Do You Know Which Tables You Need? 40
How Do You Know Which Tables You Need? (Example) 41
Class Exercise - Join Tables 42
Aggregation Query 43
Aggregation Query (contd.) 44
Aggregation Query (contd.) 45
Aggregation Function Without Group By 46
Group By <Column Name> 47
Group By - Example 48
Group By – Example (contd.) 49
Aggregation - Having 50
Aggregation - Having (contd.) 51
Where and Having 52
Aggregation - Remarks 53
Class Examples - Aggregation 54
More Class Examples - Aggregation 55
More Class Examples – Aggregation (contd.) 56
More Class Examples – Aggregation (contd.) 57
Class Exercise - Aggregation 58
Outer Join 59
An Outer Join 60
Outer Join - Class Exercise 61

A Sub Query 62
A Sub Query (contd.) 63
A Sub Query (contd.) 64
A Sub Query in Where – Option 1 65
A Sub Query In Where – Option 2 66
A Sub Query In Where – Option 2 (contd.) 67
A Sub Query In a Select Statement 68
A Sub Query In a Select Statement (contd.) 69
Class Examples - Sub Query 70
More Class Examples - Sub Query 71
Exists 72
Class Exercise - Sub Query 73
Using Views 74
Views in a Query - Syntax 75
Views in a Query 76
Class Example - View 77
More Class Examples - View 78
Class Exercise - Views 79
Set Operators 80
UNION and UNION ALL 81
UNION and UNION ALL 82
Except 83
Intersect 84
Class Exercise – Set Operators 85
How to Resolve a Query - A Check List 86
How to Resolve a Query - A Check List (contd.) 87
Chapter Summary 88
Class Exercise - Select Query 89
Class Exercise - Select Query (contd.) 90
Class Exercise - Select Query (contd.) 91
Final Exercises – Select Query – Exercise 1 92

Final Exercises – Select Query – Exercise 1 (contd.) 93
Final Exercises – Select Query – Exercise 1 (contd.) 94
Final Exercises – Select Query – Exercise 1 (contd.) 95
Final Exercises – Select Query – Exercise 1 (contd.) 96
Final Exercises – Select Query – Exercise 1 (contd.) 97
Final Exercises – Select Query – Exercise 1 (contd.) 98
Final Exercises – Select Query – Exercise 2 99
Final Exercises – Select Query – Exercise 2 (contd.) 100
Final Exercises – Select Query – Exercise 2 (contd.) 101
Final Exercises – Select Query – Exercise 2 (contd.) 102
Final Exercises – Select Query – Exercise 2 (contd.) 103
Final Exercises – Select Query – Exercise 2 (contd.) 104

SELA

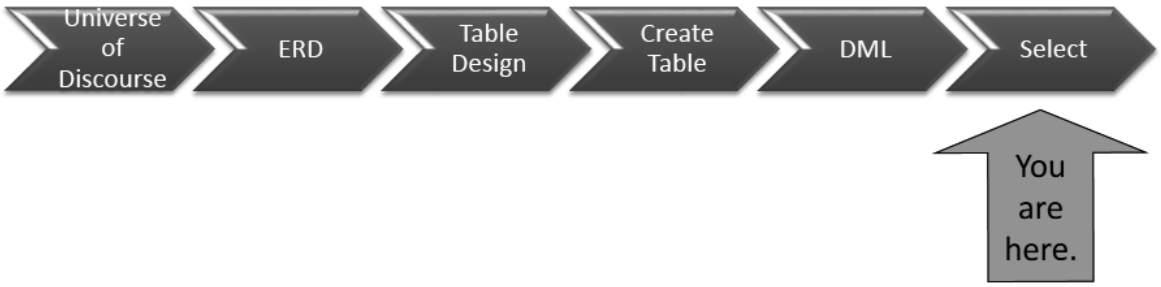
Introduction to Relational Databases and SQL

Module 05 - The Select Query

© Copyright SELA software & Education Labs Ltd. | 14-18 Baruch Hirsch St Bnei Brak, 51202 Israel | www.selagroup.com

Chapter Goals

- To understand what is SQL?
- To learn about SQL's various groups of statements.
- To learn specifically about the "powerful" SELECT statement.



What is SQL?

- SQL – **S**tructural **Q**uery **L**anguage.
- 4GL – It is a 4th generation language.
  - Machine language, Assembler, C are 1st – 3rd generation languages, respectively.
- It contains very little syntax, and is very simple.
- It contains a significant amount of logic.
- ANSI (1992) set standards. Most of the database engines accept ANSI standards.

A little history – SQL was developed at IBM in the early 1970s. The language was designed to help manipulating and retrieving data from IBM's database product (System R). Since then it was developed to the language that we know today. The SQL is shortcut of Structured Query Language. It is a language of queries which enables us to get the relevant data that is stored in the database. It contains little syntax and it is very simple but very powerful. For a language with little syntax it contains a significant amount of logic inside.

SQL's Groups of Statements

- DDL - Data Definition Language (Chapter 7):
  - Defining the data model.
- DML- Data Manipulation Language:
  - Action Query - Changes the data (Chapter 6).
    - Update, Insert, Delete statements.
  - Read-only Query (covered in this chapter).
    - The Select statement.
- DCL - Data Control Language (not within the scope of this course):
  - Deals with the user's accessibility to data.

SQL has a few group of statements which declare it language:

- The DDL – Data Definition Language is the language that help us construct the data model. The model is constructed from tables, indexes, primary and foreign keys and etc.
- The DML – Data Manipulation Language is a bunch of statements that enables us to manipulate data (CUD – Create/Update/Delete) and also enables the query of data (the Select statement).
- The DCL – Data Control Language is a bunch of statements that enables the security definition of user accessibility to resource of the database and for data.

SQL Language Elements

- Clauses
  - Components of statements and queries
- Expressions
  - Produces scalars or tables which consist returned data
- Predicates
  - Conditions that can be evaluated
- Queries
  - Retrieves specific data
- Statements
  - Affects schemas and data

SQL has a few language elements that we use in order to do things. The elements are:

- Clauses – the components of statements and queries.
- Expressions – enables the production of scalars or tables which consist returned data.
- Predicates – which are conditions that can be evaluated.
- Queries – which enable the retrieval of specific data.
- Statements – which affects schemas and data.

Database (DB) Management Environment

- Start the database engine and the DB file.
  - See Appendix I for instructions.
- Start the interface.
  - See Appendix I for instructions.
- The interface enables you to send SQL commands to the engine.
- In this course, we will work on a SQL Server DB that is called Northwind.

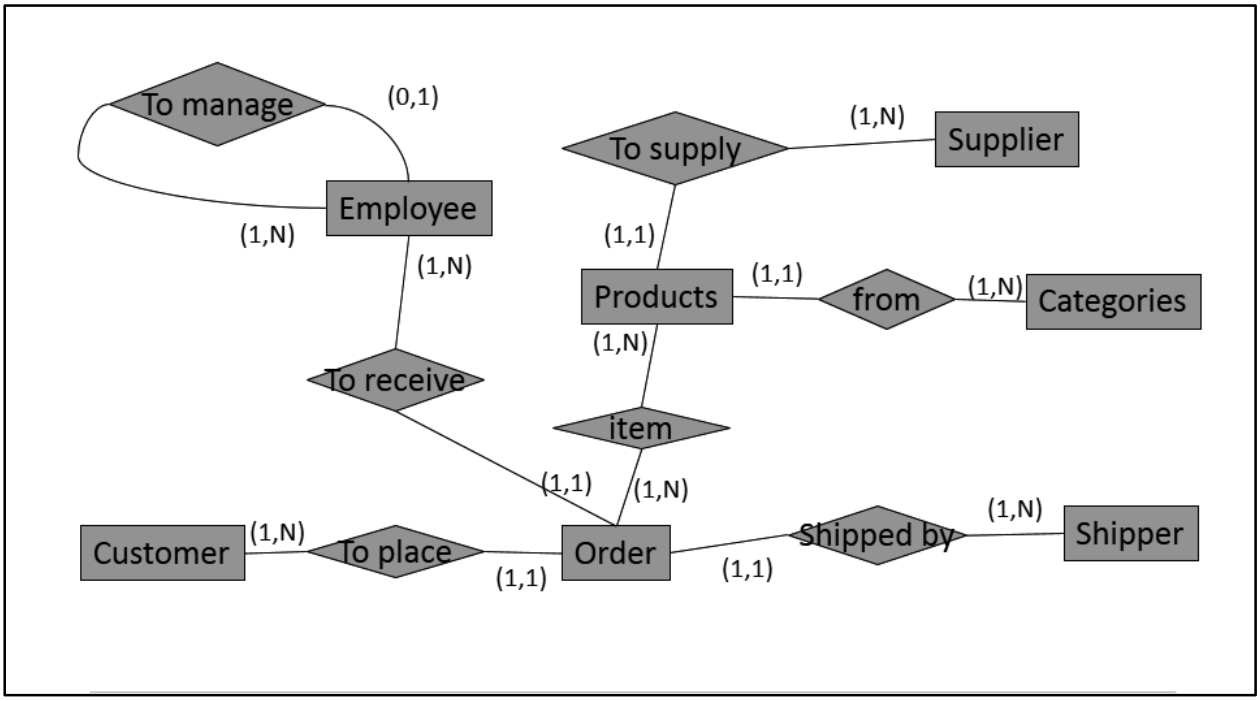
When we want to use a created database we first need to start the database engine. For example in SQL Server, we need to start the SQL Server service. After we start the database engine, we start the management tool that enables us to create or manipulate the database and tables. In the course we are going to use SQL Server and the Northwind database. The Northwind database contains data related to the sales management of a fabrics company, including its: products; customers; orders and employees.

Class Example: Northwind - UOD

- The Universe of Discourse for Northwind:
  - Customers place orders.
  - An order contains many products.
  - Employees take an order from a customer.
  - Employees work in one department.
  - Each employee has a manager (who is another employee).
  - Each department has a manager (who is one of the employees).

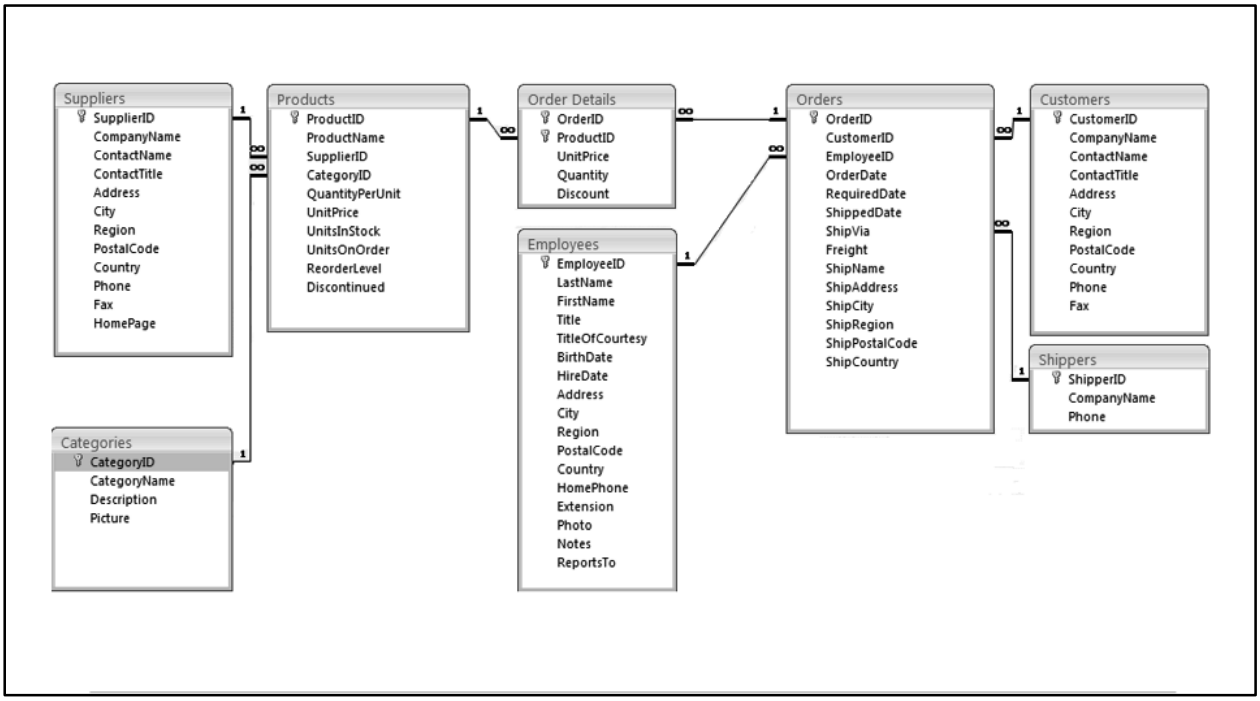
The slide contains the Universe of Discourse for Northwind database.

Class Example: Northwind – ERD



The ERD of Northwind database.

Class Example: NorthWind - Table Design (schema)



The table design of Northwind database.

Select - The Logical Idea

- We normalized the universe of discourse into tables. Why?
  - Mainly, to avoid data duplications.
- Now we will connect the tables again.
  - In order to question (queries) about the complete data structure.
- Each query requires data from different parts of the table design model:
  - E.g., I want to see departments and employees, or products and orders.

The query is the most common SQL operation. It's performed with the SELECT statement. SELECT retrieves data from one or many tables or other expressions. The SELECT enables us to shape the data that we want to retrieve and that makes it very powerful.

Select - The Logical Idea (contd.)

- In each query we will want to display various properties of the entities:
  - E.g., I want to display only the department name, employee name and salaries.
- Not all the rows are relevant to the query:
  - E.g., I want to display only employees from New York.
- Sometimes we will want to make calculations based on the data:
  - E.g., calculate the average salary for each department.

The select statement will help us to query the table's data, to display the properties of each entity (row) that we hold in the database, to retrieve the relevant entities (rows) or to return calculations or aggregations.

The Select Command

- The structure of the command:
  - Select [ALL|DISTINCT] <column list> or <*>
    From <tables list>
    Where clause
    Group by <column list>
    Having aggregate clause
    Order by <column list>;
- A clause is:
  - <column name>< test> <value|column name|sub query>
- An aggregate clause is:
  - function(<column name>)< test><value|column name|sub query>

Just the syntax, no explanations at this point.

The structure of the select command is very simple.

The Select Statement

- SELECT * FROM <table>
  - Show all columns from table or tables:

```
SELECT * FROM Employees;
```

- SELECT <column list> FROM <table>
  - Show only the columns specified, from a table:

```
SELECT ProductID,ProductName FROM Products;
```

- SELECT function(<column>) FROM <table>
  - Perform a calculation on a column's data:

```
SELECT OrderID, YEAR(OrderDate) FROM Orders;
```

The Select clause itself can be used to retrieve all the columns (*), column list or perform calculation on the column's data.
Even though we can use select * to select all the columns – a good practice is to use a column list instead. The reason is that if we add columns in the future, the using of * can impact the application that uses the database. All the slide's examples are only showing the use of the select statement and all the other clauses that can be added in order to add more functionality to the select statement, will be discussed later.

The Select Statement (contd.)

- SELECT <column><operator><column> FROM <table>
  - Perform a calculation on a column's data:

```
SELECT UnitInStock * UnitPrice FROM Products;
```

The Select Statement

- SELECT DISTINCT <column list> FROM <table>
  - Eliminate duplication of the same value in the query result.
  - Example: display the names of the states in which employees live.

```
SELECT City
FROM Employees
```

City
Seattle
Redmond
London
London
Seattle

```
SELECT DISTINCT City
FROM Employees
```

City
Seattle
Redmond
London
~~London~~
~~Seattle~~

The DISTINCT statement eliminates duplications of the same value in a query result. It acts as some sort of a filter that enables the retuning of a set.

Using the DISTINCT statement is considered a bad practice, since there are always ways to create a query that returns the same result set but doesn't use DISTINCT. Also, it might indicate that the database isn't normalized. Even then, we learn this statement because it's a part of the SQL language, and sometimes we don't have a choice and we use it.

Select - Caption

- SELECT <column name> AS 'caption' FROM <table>
  - To rename a column to a different title/caption:
    - SELECT EmployeeID AS 'EMPLOYEE NUMBER'
      FROM Employees;
  - Especially useful for calculated columns:
    - SELECT YEAR(getdate()) - YEAR(BirthDate) AS 'AGE'
      FROM Employees;
      SELECT COUNT(EmployeeID) AS 'NUMOFEMP'
      FROM Employees;
  - If no name is assigned to a calculated column, the engine will title the column 'expression'.

A part of relational algebra is the rename operation. We can rename returning columns in a very simple manner with the select statement. If we return a calculated column without renaming it, the database engine will pick a name for us. The default name that it picks is ‘expression’. The old syntax to do it is to use SELECT 'caption'= <column name> FROM <table> (for example: SELECT 'EMPLOYEE NUMBER' = EmployeeID FROM Employees)
The new way is mentioned in the slide

From

- FROM <table name>
  - This command determines which tables will be included in the query.
  - How do we decide which table to include in the query?
- FROM <table name> alias
  - Assigns a table an alias, use alias instead of the table's name.
  - This is important when:
    - Using more than one table in a FROM statement.
    - Calling a table more than once (to be discussed later).
    - Columns from different tables have the same column names.
    - Using Join operations (to be discussed later).

The from clause is very important and it determines the table or tables that will be used in the query. We decide which tables will be included inside a query, according to the results we seek. If we need employees then only the Employee table will be used in the from clause. If we need employees and also their customers then we probably are going to need more than one table in the from clause. Another reason to decide to use other tables is self joining (join the same table to itself) or using joins themselves.

From (contd.)

- An example of an alias:

```
SELECT E.EmployeeID
FROM Employees E;
```

Alias use

Alias name

- A table's "full path"
  - from <database>.<owner>.<table>
  - Sometimes it is better to give the table's "full path".
  - It is important in real world database architecture.
    - One query can call several databases:

```
SELECT LastName,FirstName, OrderDate

FROM DBO.Employees, DBO.Orders;
```

The use of “full path” indicates all the details a query needs in order to use a table. Since we can have the same table name in different databases, it is preferable to use “full path” then the shortened way.

In real world database architecture, it is very important to use the full path of a table since we can create only one query that calls several databases which can be addressed from one application (applications can work on different databases simultaneously).

Sql Server Functions

- SQL Server provides many built-in functions.
- The categories of built-in functions are:
  - Conversion Functions
  - Date and Time Data Types and Functions
  - Logical Functions
  - Mathematical Functions
  - String Functions
- Using a function example:

```
Select func(col1)
From table1;
```

Common String Functions

- Replace (string_expression, string_pattern, string_replacement)
  - Replaces all occurrences of a specified string value with another string value.

```
Select REPLACE('abcdef', 'de' , 'zz')

   abczzf
```

SUBSTRING (expression ,start , length)

– Returns part of a character text.

```
Select SUBSTRIG('abcdef', 2 ,3)

   bcd
```

Common String Functions

- LEN (string_expression)
  - Returns the number of characters of the specified string expression, excluding trailing blanks.

```
Select LEN('abcdef')

   6
```

LOWER (character_expression) | UPPER (character_expression)

– Change the case of the strings.

```
Select UPPER('aBcdEf'), LOWER('AbcD')

   ABCDEF    abcd
```

Common Date & Time Functions

- GETDATE ()
  - Returns the current database system timestamp as a **datetime** value

```
Select GETDATE()

 2010-08-10 17:21:02.310
```

YEAR (date) | MONTH (date) | DAY (date)

– Returns an integer that represents the year \ month \ day part of a specified *date*.

```
Select year('2010-08-10'), month('2010-08-10'), day('2010-08-10')
         2010  8      10
```

Common Date & Time Functions

- DATEPART (datepart , date)
  - Returns an integer that represents the specified *datepart* of the specified *date*.

```
Select DATEPART(year, GETDATE())

 2010
```

DATEDIFF (datepart , startdate , enddate)

– Returns the count (signed integer) of the specified *datepart* boundaries crossed between the specified *startdate* and *enddate*.

```
select DATEDIFF(month,'2010-08-10','2010-10-11')

          2
```

Common Date & Time Functions

- DATEPART (datepart , date)
  - Returns an integer that represents the specified *datepart* of the specified *date*.

```
Select DATEPART(year, GETDATE())

 2010
```

DATEDIFF (datepart , startdate , enddate)

– Returns the count (signed integer) of the specified *datepart* boundaries crossed between the specified *startdate* and *enddate*.

```
select DATEDIFF(month,'2010-08-10','2010-10-11')

          2
```

Common Mathematical Functions

- ROUND (numeric_expression , length)
  - Returns a numeric value, rounded to the specified length or precision.

```
Select ROUND(123.9994, 3), ROUND(123.9995, 3)

   123.9990    124.0000
```

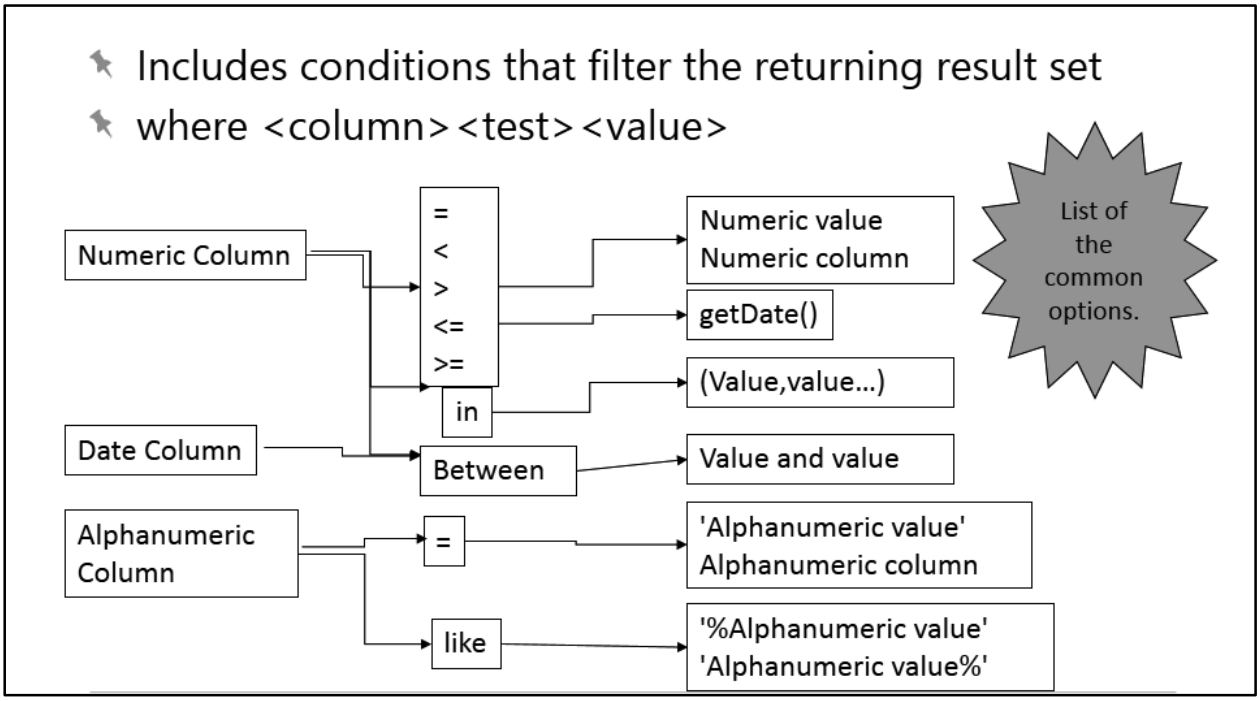
FLOOR (numeric_expression)

– Returns the largest integer less than or equal to the specified numeric expression.

```
SELECT FLOOR(123.45), FLOOR(-123.45)

         123    -124
```

The Where Clause



The Where clause is used to filter the returning data by using conditions. If an entity (row) doesn't pass the condition it won't be retrieved by the query. The Where clause can use binary operations (such as >, < and etc), In operator that enable checking inside a set, Between operator that checks whether the details are between two values, Equals operator that check for equality of two values and Like operator that checks whether the alphanumeric value contains a given value.

The Where Clause (contd.)

- If a column's data type is: int, real, double or date:
  - Operators are = |< | > | <= | >= | != | between | in

```
SELECT * FROM Products WHERE UnitPrice >= 10;
SELECT ProductName FROM Products WHERE ProductID != 300;
SELECT * FROM Orders WHERE YEAR(OrderDate) BETWEEN 1996 AND 1997;
```

- If a column's data type is char (alphanumeric)
  - Operators are = | in

```
SELECT * FROM Customers WHERE City = ‘London';
```

  - Operator is like if you want to use a wild card (* in DOS)

```
SELECT * FROM Employees WHERE FirstName LIKE 'AL%';
SELECT * FROM Products WHERE ProductName LIKE '%S%';
```

In numeric data type, we use the operator of numeric data like >,<, >= and etc.
In alphanumeric data type, we can use the =, IN, or LIKE operators.

Where, And, Or

- Use *and* or *or* to build a logical where clause. You may also use parentheses to force precedence:

```
SELECT * FROM Product WHERE UnitPrice >= 10 AND UnitsInStock
> 5;
SELECT * FROM Orders WHERE (YEAR(OrderDate) BETWEEN 1996 AND
1996) OR YEAR(OrderDate) > 1997;
```

- Where is also used to join tables - to be discussed later.

In the Where clause we can use and or or operators to compose filters that includes more logic. As in all mathematical equations, we can use parentheses to force the precedence of conditions. We can also use Where clauses for join operations. This will be explained in more details later on.

Order By

- Use ORDER BY to sort your query results.
- A sort can be in either ascending (ASC) or descending (DESC) order.

```
SELECT EmployeeID, LastName, FirstName
   FROM Employees
   ORDER BY LastName ASC , FirstName ASC;

SELECT *
   FROM Products
   ORDER BY UnitPrice DESC;
```

Order by is used to order the returning result set. It can be used on one column (the returning result set will be ordered by that column) or on more than one column (the returning result set will be ordered by the first column, then by the second column and so on).

Class Example - Select Statement

- Display a list of customers' states :

```
SELECT DISTINCT Country FROM Customers;
```

- Display the years of order dates:

```
SELECT DISTINCT YEAR(OrderDate) FROM Orders;
```

- Display all customers from OR state, display their names and addresses:

```
SELECT CustomerID, ContactName, Address FROM Customers WHERE
  Region='OR';
```

- Display a list of diary products, and their prices in US $ and in NIS (or another currency):

```
SELECT ProductID, ProductName, 'IN $'=UnitPrice,
  'IN NIS'=UnitPrice*4.20 // a random value for NIS
  FROM Products WHERE CategoryID=4;
```

Class Example - Select Statement (contd.)

- Display all the orders from 1996. Display order id, date and ship region, sorted by order's date.

```
SELECT OrderID, OrderDate, ShipRegion FROM Orders
   WHERE YEAR(OrderDate) = 1996
  ORDER BY OrderDate;
```

Class Exercise - Select

- Display all product ids, names and prices, sort by name.
- Display each employee's name, and the number of years he / she has worked at the company.
- For employees who were born after 1955, display their names, addresses and birth dates. Sort list by addresses.
- Display the names of those products whose stocked quantity is greater than 100 , sorted by name.

Join Tables

- Why join tables?
  - So far we have only worked with one table in each query.
  - However, information is derived from the relations between tables.
- How?

```
FROM Employees INNER JOIN Orders ON
Employees.EmployeeID = Orders.EmployeeID;

FROM Employees, Orders
WHERE Employees.EmployeeID = Orders.EmployeeID;
```

Call for two tables

Join tables by common column

- What do we receive?
  - For each entity in one table, we get the matching entity or entities in the second table.

Join operations help us to retrieve data through the relations that we have in the database. When we join tables we need a relation between them or else we will receive a Cartesian Product which means that every entity from the first table will be returned with all the possible entities from the second table. When we talk about the join in this slide we talk about an inner join. There are more join options that will be discussed later on.

Join Tables (contd.)

- If we don't join tables by matching columns we receive a "Cartesian Product".
  - Each entity from one table is displayed with all possible entities from the second table.

Join Table - Examples

- Display customer id, contact name, order id and order date.

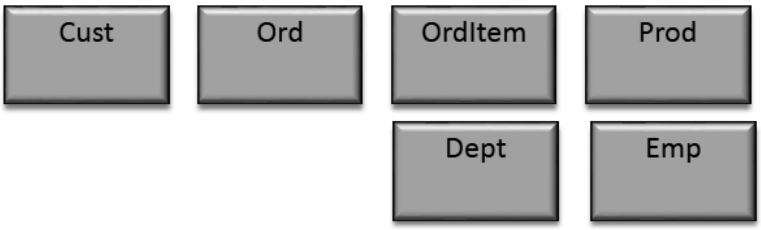
```
SELECT Customers.CustomerID, ContactName, OrderID, OrderDate
   FROM Customers INNER JOIN Orders ON
   Customers.CustomerID = Orders.CustomerID;
```

Display product name, description, price and orders containing this product.

```
SELECT ProductName, Products.UnitPrice, OrderID
FROM Products, "Order Details"
WHERE "Order Details".ProductID =Products.ProductID;
```

How Do You Know Which Tables You Need?

- All the tables which contain those columns the query needs to display.
- Tables that contain columns you need to filter by (where).
- Tables that contain columns you want to sort by.
- Tables that connect the previous tables.

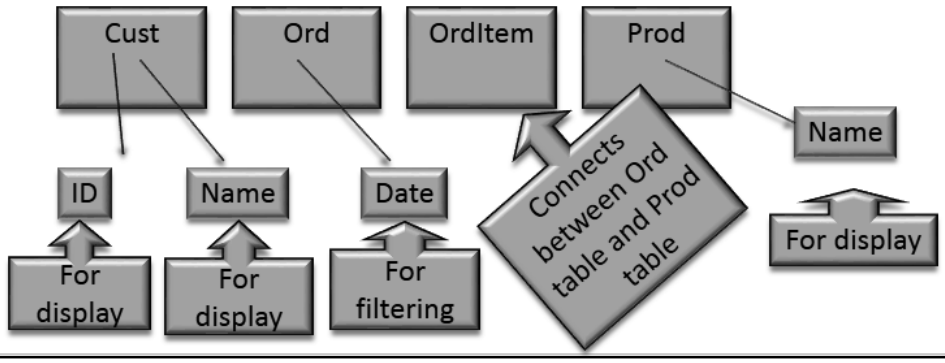


The decision which tables you need is a very crucial decision that can impact the performance of a given query (every join operation is very expensive) so think about the answer before you implement a query.

How Do You Know Which Tables You Need? (Example)

- Q: Display customer's ID and name, and the products they ordered in 1993, sorted by customer ID.

```
select C.CustomerID, ContactName, ProductName
   from Customers C, Orders O, "Order Details" OD, Products P
   where C.CustomerID = O.CustomerID and O.OrderID = OD.OrderID and
   OD.ProductID = P.ProductID and year(OrderDate) = 1996
   order by C.CustomerID;
```



Class Exercise - Join Tables

- Display Supplier id and company name and the names of the products it supplies.
- Display customer id, contact name and order id for: orders placed in 1996, and customers who live in London, sort by contact name and order id.
- Display customers' contact names and the products they ordered in 1998, only for products that cost more than $10
- Display Order IDs, product names and calculate product unit price * ordered quantity for each order item.
- Display employee name and the name of his / her manager (It is a unary relation in the employee table).

Aggregation Query

- Why do we need this?
  - Up until now, we made calculations only within the rows.
    - Calculate quantity * price, ship date - order date, etc.
  - We were not capable of performing calculations on values from different rows.
    - How many employees are there in the company?
    - What is an order's total cost?
  - We could not run functions on rows.
    - How old is the oldest employee?
    - What is the price of the most expensive product?

Aggregation queries are very useful in order to calculate or run functions on rows. Until now, we used only within the row calculations. There are some aggregate functions that we will discover in the next slide.

Aggregation Query (contd.)

- Therefore:
  - Aggregation (comes from the word combined) query.
  - Returns a single value from a given set
  - New functions: sum, average, count, min, max.
  - Aggregation queries require special treatment.

Aggregate functions are also referred to as combined queries. They return a single value from a given set since they make calculations on the rows. We have the following aggregate function for our use: Sum, Average, Count, Min and Max. When we use aggregate functions we need to be careful. The use of them has strict rules that we must follow or else we will get errors.

Aggregation Query (contd.)

- The syntax is as follows:
  - Select <column name>, <aggregation function>(<column name>)
  - From <Table>
  - Where <Clause>
  - Group by <column name>
  - Having <aggregation function>(<column name>) <test> <value>;
- Aggregation functions:
  - **Sum** - summates the values in a numeric column.
  - **Avg** - calculates the average values of a numeric column.
  - **Count** - counts the number of rows.
  - **Min** and **max** - finds the minimum or maximum value in a numeric or date column.

Aggregation Function Without Group By

- The calculation will be performed on all table rows.

```
SELECT SUM(UnitPrice * UnitsInStock)FROM Products;
SELECT MAX(BirthDate) FROM Employees;
SELECT AVG(UnitPrice) FROM [Order Details]
WHERE OrderID = 10248;
```

You cannot display columns that are not under an aggregation function.

```
SELECT ProductID, SUM(UnitPrice * UnitsInStock)
FROM Products;
```

Error

We saw that we use aggregate functions on columns and that they perform their calculation on all the rows. One restriction that we have when using aggregate functions is, that we can’t display columns that aren’t under the aggregate function. When we want to display other columns (not under the aggregate) we use the Group By clause.

Group By <Column Name>

- Why do we need this command?
  - To separate the aggregation function's calculations for each...
    - Display how many orders each customer ordered.
    - Display the total quantity of products sold in 1993.
    - Display the highest salary in each department.
- How do we do this? – We must ask 3 questions:
  - What is the grouping column?
  - What is the aggregation function?
  - On which column should the aggregation function work?

The Group By enables us to get data according to groups when we use an aggregate function. It groups the entities according to the columns that we choose to participate in the Group By clause.

Group By - Example

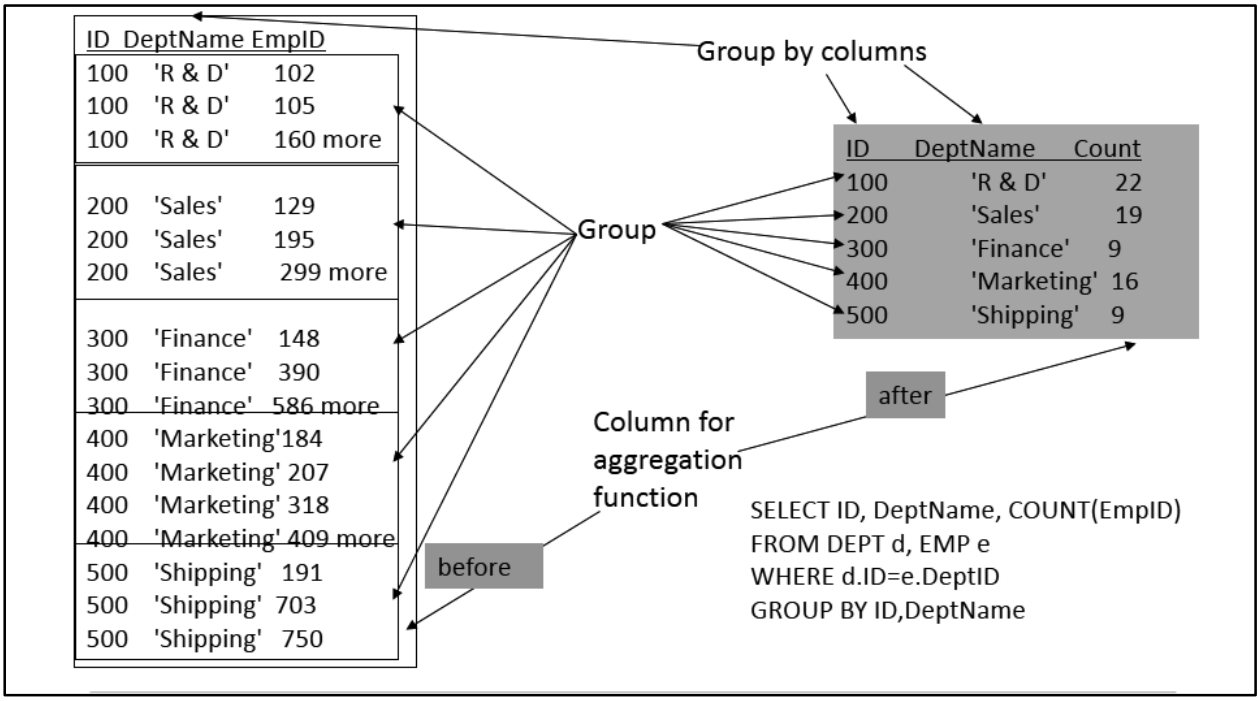
- Display the number of employees in each city:

```
SELECT City, COUNT(EmployeeID)
FROM Employees
GROUP BY City;
```

- Results:

City	Count(EmployeeID)
‘Kirkland'	1
‘London'	4
‘Redmond'	1
‘Seattle‘	2
‘Tacoma'	1

Group By – Example (contd.)



Aggregation - Having

- To filter the aggregation function's results.
  - Example: For cities containing more than 1 employee, show city's name and the number of employees in that city.
- Example:

```
SELECT City, COUNT(EmployeeID)
FROM Employees
GROUP BY City
HAVING COUNT(EmployeeID) > 1;
```

- Having appears and occurs after group by.

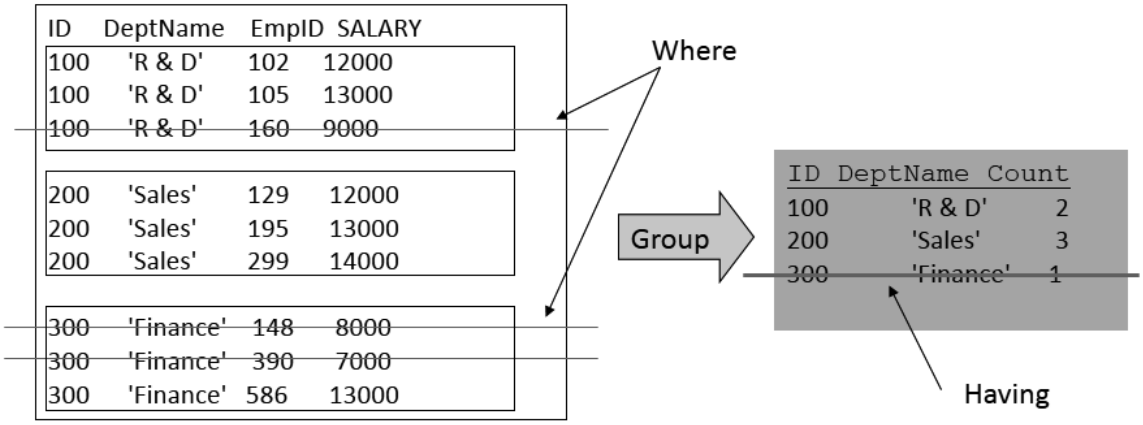
A Having clause enables the filtering of aggregate function result. It is always used after the Group By clause, if we need a filter. Don't be confused by Having and Where clause, they do different things, even though both of them filter the data.

Aggregation - Having (contd.)

- Having and Where are not interchangeable.
  - An aggregation function can be used only in conjunction with Having.
  - A simple WHERE clause can be used only in conjunction with Where.
    - Cannot use `...WHERE COUNT(EmployeeID)>10...`
    - Cannot use ...`HAVING UnitPrice < 10000;`

Having and Where clauses are very different clauses. Each one of them performs a filter at the end, but they can be used in different places. Having is only used with aggregate functions and can filter aggregate results. Where is used on columns that aren't aggregate functions results.

Where and Having




```
SELECT ID, DeptName, COUNT(EmpID)
FROM DEPT d, EMP e
WHERE d.ID=e.DeptID AND SALARY > 10000
GROUP BY ID,DeptName
HAVING COUNT(EmpID) >= 2;
```

In the example, we can see how select execution is performed. First the where clause is implied and it filters the result set and then grouping occurs. After grouping a having clause, filtering will be performed to filter the group's result set.

Aggregation - Remarks

- Every column in the select statement must be either in an aggregation function or appear in the group by statement.

```
SELECT City, COUNT(EmployeeID)
FROM Employees
GROUP BY City;
```

Column appears in GROUP BY.

Column is in the aggregation function.

You must pay attention to the restriction that every column in the select statement, must be either in an aggregate function or appear in the Group By clause, when we use aggregates. The result of not following this rule will be an error.

Class Examples - Aggregation

- How many customers from the customer table live in London?

```
SELECT COUNT(CustomerID)
FROM Customers
WHERE City = 'London'
```

- Display the category id and the quantity of product items which are in stock for category.

```
SELECT CategoryID, COUNT(UnitsInStock)
FROM Products
GROUP BY CategoryID
```

More Class Examples - Aggregation

- Display customer id, name and number of orders placed in 1996 for customers who live in ‘London' only.

```
SELECT Customers.CustomerID, ContactName, COUNT(*) AS "Order List"
    FROM Customers, Orders
    WHERE Customers.CustomerID = Orders.CustomerID AND YEAR(OrderDate) =
       1996 AND City = 'London'
    GROUP BY Customers.CustomerID, ContactName
```

Display order numbers and total cost of orders.

```
 SELECT Orders.OrderID, SUM(Quantity * UnitPrice)
FROM Orders, [Order Details]
WHERE Orders.OrderID = [Order Details].OrderID
GROUP BY Orders.OrderID
```

More Class Examples – Aggregation (contd.)

- Display customer name, order numbers and total order cost.

```
SELECT C.CustomerID, ContactName, O.OrderID, SUM(Quantity * UnitPrice)
FROM Customers C, Orders O, [Order Details] OD
WHERE C.CustomerID = O.CustomerID AND O.OrderID = OD.OrderID
GROUP BY C.CustomerID, ContactName, O.OrderID
```

Display customer name, order year and total order cost in that year (use: group by year (OrderDate)).

```
SELECT C.CustomerID, ContactName, YEAR(OrderDate), SUM(Quantity * UnitPrice)
FROM Customers C, Orders O, [Order Details] OD
WHERE C.CustomerID = O.CustomerID AND O.OrderID = OD.OrderID
GROUP BY C.CustomerID, ContactName, YEAR(OrderDate)
```

More Class Examples – Aggregation (contd.)

- Display customer name and total order cost for customers who placed orders totaling more than $5000.

```
SELECT C.CustomerID, ContactName, 'SCUM' = SUM(Quantity * UnitPrice)
    FROM Customers C, Orders O, [Order Details] OD
    WHERE C.CustomerID = O.CustomerID AND O.OrderID = OD.OrderID
    GROUP BY C.CustomerID, ContactName
    HAVING SUM(Quantity * UnitPrice) > 5000;
```

Class Exercise - Aggregation

- What is the average age of company's employees?
- How many customers live in each city (display only cities in which there is more than one customer).
- How many products does each supplier supply?
- For each year, display the dates of the first and last orders, the number of orders made and the total cost of orders.
- How many items were ordered of each product?
- Which employees' sales exceeded $20,000 in 1996?
- If we grant a $1 discount for each day between order date and ship date, what is the total amount of discounts given each year?

Outer Join

- A join (as previously discussed), is used to connect tables.
- Generally, a join is an inner join.
  - It means that only those rows that have a related row in the other side of the relation will be displayed.

```
SELECT C.CustomerID, O.OrderID
  FROM Customers C, Orders O
  WHERE C.CustomerID = O. CustomerID;
```

  - table.PK=table.FK

```
Employees.EmployeeID = Orders.EmployeeID;
```

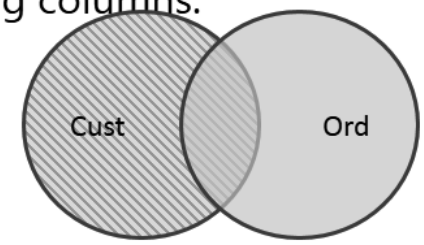
  - If a customer has no orders, his ID will not be displayed in the query's results.

Inner joins are commonly used in queries and we saw them earlier. They depend on the relations between tables to make the connection between them in a given query. What if we want results that include Customers who didn’t place an Order? In inner join it is not possible, since we won’t have the Customer ID in the Order’s table. The answer will be outer join.

An Outer Join

- An outer join:
  - Enables you to display rows even if they don't have matching rows on the other side of the relation.
- Use LEFT | RIGHT OUTER JOIN
  - The row in the LEFT | RIGHT side does not need a matching row in order to be displayed.
  - A column that has no value (because of an outer join) will display NULLs in the matching columns.
- Example:

```
SELECT C.CustomerID, ContactName, OrderID
FROM Customers C LEFT OUTER JOIN Orders O
 ON C.CustomerID = O.CustomerID
ORDER BY OrderID;
```



Outer joins enable the retrieval of entities (rows) that don't have a matching row on the other side of the relation.

Outer join is divided into LEFT outer join, RIGHT outer join and FULL outer join (which is a combination between LEFT and RIGHT outer join). Outer joins are very useful when we have an orphan record in a table (a record that doesn't have a matching row on the outer side of a relation) and we want to retrieve them in a query.

Outer Join - Class Exercise

- Display all employees. Order numbers should be displayed for those who accepted orders during August 1996.
- Display all products. Order numbers and dates should be displayed for those which were ordered in January 1997.
- Display employee name and the name of his/her manager for all employees (it is a unary relation in the employee table).
- Display all products and a summary of sales for products that were sold in sums exceeding $1,000 in 1998.

A Sub Query

- Why do we use a sub-query?
  - When a query has two meanings of the where clause:
    - For customers who made orders last year, display what they ordered this year.
  - You want to display details of a row you found in an aggregation query:
    - Who is the oldest employee?
    - What is the name of the most expensive product?
  - You want to display or calculate the result from another query in the select statement.
    - Display the most expensive, cheapest, average and each product's price, calculate the difference between average price and product price.

When we have ambiguous query that has two meanings we sometime use sub query. Another reason to use a sub query is to perform an aggregate of an aggregate. The last reason to use a sub query is that you need to calculate the result from another select result.

A Sub Query (contd.)

- Is a nested query inside a select statement.
- Can be used anywhere, an expression is permitted.
- Example (just to see the structure):

```
SELECT *
FROM Products
WHERE unitPrice = (SELECT max(unitPrice)
                   FROM Products)
```

Main query

Sub query

A sub query is a query that is composed of another query. It can be used in all the places where an expression can be used. It can sometimes resolve very hard problems but with a cost. When we use sub queries, we pay in terms of performance.

A Sub Query (contd.)

- There are three types of sub queries:
  - Sub query in where - Option1 (=,<,>,<=,>=,<>).
  - Sub query in where - Option2 (in, not in).
  - Sub query in select – Option 3.
- Connecting between the main query and the sub query.
  - A sub query returns one column.
  - A sub query returns one or more rows (depending on the type of sub query).

There are three kinds of sub queries which we can use: sub queries in a Where clause are used with a binary operator, sub query in where clause that use the set operators in or not in and a sub query in a select clause.
A sub query can return one or many rows. Depending on the value it is returning, we will be able to connect it to the parent (main) query.

A Sub Query in Where – Option 1

- A sub query returns one column and one row.
- Syntax:

```
select <column name>
   from <table>
   Where <column name> <test>
        (select  <column> from <table> where ….)
```

<test> is = | < | > | <= | >= | !=

Example: who is the oldest employee?

```
SELECT * FROM Employees
WHERE BirthDate =
       (SELECT MIN(BirthDate) FROM Employees);
```

A Sub Query In Where – Option 2

- A sub query returns one column but one or more rows
- Syntax:

```
select <column name>
from  <table>
Where <column name> <test>
(select  <column> from <table> where ….)
```

- <test> is in | not in

A Sub Query In Where – Option 2 (contd.)

- Example:
  - For those customers who ordered something in 1996, display their orders in 1997:
  -

```
SELECT *
FROM Customers C, Orders O
WHERE C.CustomerID = O.CustomerID
   AND YEAR(OrderDate) = 1997
   AND C.CustomerID IN
       (SELECT O.CustomerID
        FROM Orders O
          WHERE YEAR(OrderDate) = 1996)
```

A Sub Query In a Select Statement

- A sub query returns one column and one row.
- Syntax:

```
select <column name>,
(select  <column> from <table> where ….)
from  <table>
where <clause>
```

- You can calculate the sub query's results and the main query's columns.

A Sub Query In a Select Statement (contd.)

- Example:
  - Display the average product price as well as each product's price. Calculate the difference between the average and the product price.

```
SELECT
   (SELECT AVG(PR_PRICE)FROM PROD) AS AVGPRICE,
  PR_PRICE
FROM PROD;
```

Class Examples - Sub Query

- What is the name of the customer who placed the first order?

```
SELECT C.CustomerID, ContactName
FROM Customers C, Orders O
WHERE C.CustomerID = O.CustomerID
AND OrderDate = (SELECT MIN(OrderDate) FROM Orders)
```

- Who ordered the most expensive product in 1997?

```
 SELECT DISTINCT C.CustomerID, ContactName
        FROM Customers C, Orders O, [Order Details] OD
   WHERE C.CustomerID = O.CustomerID AND O.OrderID = OD.OrderID  AND
YEAR(OrderDate) = 1997 AND OD.UnitPrice =
         (SELECT MAX(UnitPrice) FROM [Order Details])
```

More Class Examples - Sub Query

- What products were included in orders whose total cost was greater than $10500?

```
SELECT DISTINCT P.ProductID, P.ProductName
FROM [Order Details] OD, Products P
WHERE OD.ProductID = P.ProductId
   AND OD.OrderID IN
   (SELECT O.OrderID
    FROM Orders O, [Order Details] OD, Products P
    WHERE OD.OrderID = O.OrderId
        AND OD.ProductID =     P.ProductID
    GROUP BY O.OrderId
     HAVING SUM(OD.Quantity * OD.UnitPrice) > 3500);
```

מ10500 גדול להיות צריך התנאי האחרונה בשורה

Exists

- Exists returns TRUE if a subquery contains any rows.
- Example: Display all order ids which contains products 7 and 16.

```
SELECT OrderID
FROM [Order Details] OutOD
WHERE ProductId=7
   and EXISTS (SELECT *
           FROM [Order Details] InOD
           WHERE  ProductId=16
               and OutOD.orderid=InOD.orderid)
```

The Outer select finds rows which contains the productID=7 and the inner select checks that there is a row of the same order with productId=16

Class Exercise - Sub Query

- What is the name of the most expensive product?
- What is the name of the most expensive product that has ever been ordered?
- Which products were demanded in orders for which the sales representative was the oldest employee ?
- Which products cost more than the average product price?
- Which products were ordered among the orders in which the most expensive product had been ordered?

Using Views

- What is a view?
  - A view is a query's results stored in a virtual table.
    - Not part of the physical database schema.
- You call a view the same way you call a table.
- When should a view be used?
  - Use a view for an "aggregation on aggregation" query.
    - Select (max(count(em_id)) - wrong !!!
    - It is impossible to perform an "aggregation on an aggregation" without using a view!
  - For more uses of views see Chapter 6.

Views are a very common way to transform a query to a stored virtual table. That virtual table can help us to avoid the use of a sub query sometimes. It can help us to create the behavior of aggregate on an aggregate and many other useful things. We use a View like we use tables, but as written earlier also, it isn't a table, it's a view on a table or tables.

Views in a Query - Syntax

- Syntax:
  - Create view <view name> as
    Select statement…;
- Example:
  - Creating a view:

```
CREATE VIEW V_NUM_OF_EMP_IN_COUNTRY AS
SELECT Country, 'CNT' = COUNT(EmployeeID)
FROM Employees
GROUP BY Country;
```

  - Using a view:

```
SELECT * FROM V_NUM_OF_EMP_IN_COUNTRY;
```

  - It is like "reselecting" the source query.

Views in a Query

- For Aggregation on Aggregation:
  - First, create a view that makes the first aggregation:

```
CREATE VIEW V_NUM_OF_EMP_IN_COUNTRY AS
SELECT Country, 'CNT' = COUNT(EmployeeID)
FROM Employees
GROUP BY Country;
```

- Create an aggregation query on the view you have already created:

```
SELECT MAX(CNT)
FROM V_NUM_OF_EMP_IN_COUNTRY;
```

Class Example - View

What was the biggest order placed in 1996?

```
CREATE VIEW V_ORD_SUM AS
SELECT OD.OrderID, SUM(Quantity * UnitPrice) AS VSUM
FROM [Order Details] OD, Orders O
WHERE OD.OrderID = O.OrderID AND YEAR(OrderDate) = 1996
GROUP BY OD.OrderID;


SELECT OrderID, VSUM
FROM V_ORD_SUM
WHERE VSUM =
   (SELECT MAX(VSUM) FROM V_ORD_SUM);
```

More Class Examples - View

- What is the name of the customer who ordered the highest amount?

```
CREATE VIEW V_CUST_SUM AS
SELECT O.CustomerID, SUM(OD.Quantity *
  OD.UnitPrice) AS [SUM]
FROM Orders O, [Order Details] OD
WHERE O.OrderID = OD.OrderID
GROUP BY O.CustomerID

SELECT C.CustomerID, C.ContactName,[SUM]
FROM Customers C, V_CUST_SUM
WHERE C.CustomerID = V_CUST_SUM.CustomerID AND
[SUM] =
  (SELECT MAX([SUM]) FROM V_CUST_SUM)
```

Class Exercise - Views

- How many items were sold by the best sales representative (in terms of selling the largest quantity of items) of the year, in every year?
- Who was that best sales representative of the year, in every year?
- Which product has been ordered the most?
- Which product has made the largest income?

Set Operators

- Set operators combine results from two or more queries into a single result set.
- SQL Server provides the following set operators:
  - Union (Union All)
  - Except
  - Intersect
- The following are basic rules for combining the result sets of two queries by using Set Operators:
  - The number and the order of the columns must be the same in all queries.
  - The data types must be compatible.

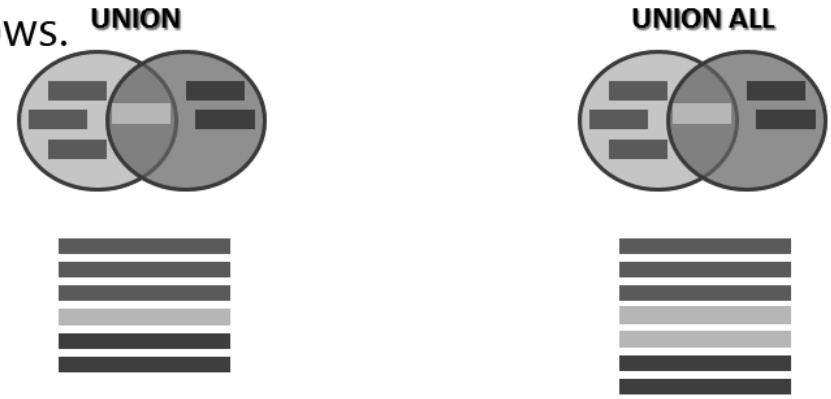
UNION and UNION ALL

- Combines the results of two queries into a single result set that includes all the rows that belong to all queries in the union.
- Example: select all customers names and workers in one list.

```
SELECT ContactName FROM Customers
UNION ALL
SELECT FirstName + ' ' + LastName  FROM Employees
```

UNION and UNION ALL

- **UNION** eliminate the duplicate rows from the combined results of the SELECT statements.
- **Union ALL** does not remove the duplicates rows.



Except

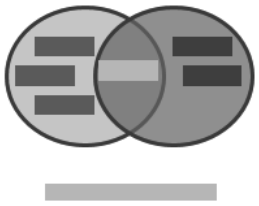
- EXCEPT returns any distinct values from the left query that are not also found on the right query.

- For Example: Display Product Ids that have never been ordered.

```
SELECT ProductID FROM Products
EXCEPT
SELECT ProductID FROM [Order Details]
```

Intersect

- INTERSECT returns any distinct values that are returned by both the query on the left and right sides of the INTERSECT operand.



Class Exercise – Set Operators

- Display all cities and countries of the customers and suppliers.
- Display all cities and countries of the customers, suppliers and employees.

How to Resolve a Query - A Check List

- Do you need an aggregation?
  - If yes, what is the grouping column?
  - What is the aggregation function?
  - What is the aggregation column?
  - Do you want to filter the aggregation results?
    - If yes, use HAVING.
- Add display columns:
  - In the select line.
- If you need a sub query, match columns from the main and the sub query.

How to Resolve a Query - A Check List (contd.)

- Does the query have two meaning request?
  - If yes, separate/rewrite the query as two separate queries.
  - Resolve two queries separately.
  - Use a sub query.
- Does the query ask for aggregation on aggregation?
  - If yes, use a view.
- You now have separated, non-ambiguous queries.
- Join the required tables.
- Add the additional WHERE criteria.

Contd.

Chapter Summary

- The select statement is both a simple and a complicated command.
- Solve queries in steps.
- Start with the data you have and move towards the information the query requests.

Class Exercise - Select Query

- For each employee, display name, address and phone number as well as the number of orders which belong to him, sort the output by employee id in descending order.
- List the products (id, name) that were not ordered by any customer in the second half of July 1996, and for which there are more than 50 items currently in stock.
- Display the best sellers from the products (by name, not id) in two categories: quantity and value (display product name, total value, total qty).

Class Exercise - Select Query (contd.)

- 4. Display all the customers who placed orders for products similar to those ordered by any customer from London, in the last two weeks of 1996.
- Display a list of all customers who did not place any order containing the bestsellers.
- Display a list of all products not ordered in December 1996 by customers from Sao Paulo.
- Display a list of all customers and the number of orders they placed in December 1997. For those who placed orders, display the total amount of those orders.

Class Exercise - Select Query (contd.)

- 8. For each city, display the total sales value on a per month basis for orders made in 1997.
- 9. Display the average quantity ordered of each product, on a monthly basis, in each year.

Final Exercises – Select Query – Exercise 1

- SQL – selects
  - The SQL statements in this exercise relate to the a3v32.db database,
- which is located in each student's home directory.
- You should put documentation inside your code (/* */). The documentation will explain the query (or the sub query) role and will help the reader understand the purposes of the written code.

- Write SQL queries for the following questions:

1. Show the first name, last name and telephone number for all the employees who live in the state 'MA'.

Final Exercises – Select Query – Exercise 1 (contd.)

2. Show the first name, last name and telephone number for all the employees, except those who live in a city named 'Lexington'.
3. Show all product details for products whose unit price is greater than $10 and are colored black. Order by product price.
4. Show the product name, description, size, color and stock quantity of the products that have 32 or more items in stock.

Final Exercises – Select Query – Exercise 1 (contd.)

5. Show all details for products whose colors are ‘Orange’, ‘White’ or ‘Black’. Order them by product name and color.
6. Show the first name, last name, telephone number and date of birth for the employees who are aged older than 45. Order them by last name in descending order.
7. Show the first name, last name and telephone number for the employees who started working in the company in 1980-1990.

Final Exercises – Select Query – Exercise 1 (contd.)

8. In which order numbers was the ordered quantity greater than 10% of the quantity in stock?
9. Show names of departments that have, a female as the department's manager.
10. What would be the discount for all the NY customers (together), if for every day of gap between the order date and shipping date they get a 10% discount per item they bought?

Final Exercises – Select Query – Exercise 1 (contd.)

11. Show the product code, name, description, stock quantity, price and total value (product price * stock quantity) for products whose total value is greater than $500.
12. Show for each order item – the customer name and order id, product name, product description, ordered quantity, product price and total price (orderedquantity * product price) and gap between ordered date and shipped date. Order by order id.

Final Exercises – Select Query – Exercise 1 (contd.)

13. Show the employee names, department names and department manager names ordered by department names (a clue: you can use table EMP twice with different alias).
14. Show each employee's name and the product name for the products that he has sold (each order has a sales representative code).

Final Exercises – Select Query – Exercise 1 (contd.)

15. Do customers like products starting with the first letter of their last names? Show customers and products where the first letters of their last name and their products ordered match (use LEFT(…) function).

Final Exercises – Select Query – Exercise 2

- SQL – Selects
  - The SQL statements in this exercise relate to the a3v32.db database, which is located in each student's home directory.
- You should put documentation inside your code (/* */). The documentation will explain the query (or the sub query) role and will help the reader understand the purposes of the code written.
- Write SQL queries for the following questions (submit a printout and a diskette):

Final Exercises – Select Query – Exercise 2 (contd.)

1. Show only for female employees: number of employees, lowest salary, highest salary and average salary.
2. Show the average price of products whose name contains the word 'shirt'.
3. Show the codes, names, salaries, cities and countries of salesmen who made at least 23 orders in 1994.
4. Show for each order: order number, customer name, order date, order region, salesman number, salesman name and phone number for each order that contains

Final Exercises – Select Query – Exercise 2 (contd.)

at least 60 items from 'Tee shirt' or 'Sweatshirt'.

5. Show the employee number, name and salary, only for women whose salary is greater than the average salary.
6. For the company managers, show the manager code and number of employees they run, ordered by the number of employees.
7. For each order, show the order number, number of lines in the order and the total quantity of ordered items (sum of the quantity). Order the result by the total quantity of ordered items in descending order.

Final Exercises – Select Query – Exercise 2 (contd.)

8. Show the customer code, name and phone number for customer who made more than 3 orders in 1994.
9. Show for each product, for each year (of the order date): the product number, number of orders made for that product and the total amount delivered of that product.
10. Show for women employees: the employee number, first and last name and phone number of only those departments which have at least 7 women. Order by department and salary.

Final Exercises – Select Query – Exercise 2 (contd.)

11. Show the employee code, name and phone number only of employees who made no orders at all.
12. Show the employees: employee number and name and salary, only for those who have more than 30 orders during 1993 and 1994.
13. Show the manager code, name and salary of managers who manage more than 10 employees.
14. Show for each department: department's manager code, number of employees in the department, average salary and highest and lowest salary.

Final Exercises – Select Query – Exercise 2 (contd.)

15. Show the product code and name for products that were not ordered in 1994.
16. Show the employee id, name and age for employees whose period of service is greater than 6 years.
17. What's the name of the customer who ordered the highest sum?
18. What's the best sold product by price multiplied by quantity sold?
19. What's the average income per year (by price multiplied by quantity sold)?

Questions

Module 06 - Update, Insert, Delete

Contents:

Chapter Goals 3
Action Queries 4
Insert New Rows 5
The Insert Query - Remarks 6
The Insert Query - Class Exercise 7
The Update Query 8
The Update Query - Class Exercise 9
The Delete Query 10
The Delete Query - Class Exercise 11
Chapter Summary 12

SELA

Introduction to Relational Databases and SQL

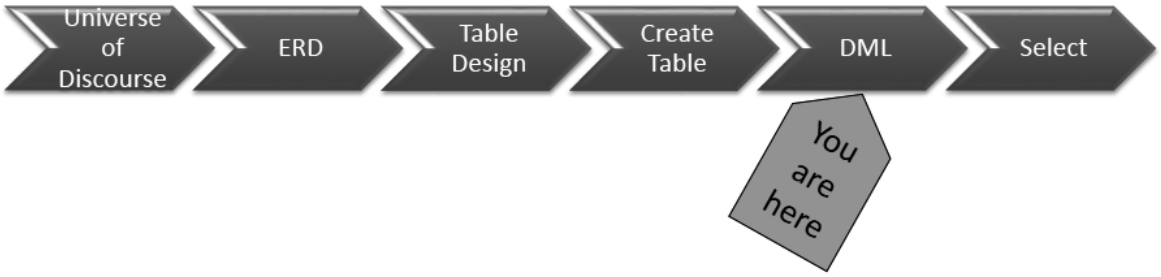
Module 06 - Update, Insert, Delete

© Copyright SELA software & Education Labs Ltd. | 14-18 Baruch Hirsch St Bnei Brak, 51202 Israel | www.selagroup.com

Chapter Goals

- In the previous chapter, we learnt about the Select statement.
- In this chapter, we will learn about the SQL 'action' statements:

Insert, Update, Delete



Action Queries

- The Select statement is only used in read only queries.
- As opposed to action queries, which actually change the data in the DB.
- Action queries include the following three types of queries:
  - Insert -
    - Inserts new rows into tables.
  - Update -
    - Revises data in existing rows.
  - Delete -
    - Deletes rows.

In the previous chapter, we saw the select query which is a part of DML (Data Manipulate Language) that enables the retrieval of read only data. The second part of DML is the use of action queries which include the CUD operations (Create/Update/Delete). When we want to create rows we use the Insert statement. When we want to update rows we will use the Update statement. When we want to delete rows we will use the Delete statement.

Insert New Rows

- An Insert statement inserts new rows into a single table.
- Syntax:
  - Insert into <table> [<column names>] values (value,value…)
- Example:

```
INSERT INTO DEPARTMENTS (ID,NAME) VALUES (600,'TQM');

INSERT INTO PRODUCTS VALUES (801, 'BOOTS', 'WINTER', 'ALL SIZES',
                                  'BLACK', 200, 50);
```

The Insert Query - Remarks

- Certain columns are mandatory, which means that values must be recorded.
- Default Values:
  - A default value can be assigned to a column (this will be explained in the section dealing with the create table statement).
- NULL:
  - This is used to assign a NULL value to a column.
- Example:

```
INSERT INTO PRODUCTS VALUES (901, 'sport shoes',
NULL, NULL, DEFAULT, 200, 35);
```

This will not actually work. Northwind, does not accept NULLS

There are strict rules for inserts:

- The number of columns and values need to be the same. The database engine doesn't need to guess in which column a given value should be added.
- There are mandatory columns (which can't be NULL for example). If values are not provided to these columns the insert operation will fail.
- If there are constraints on columns, you should provide the relevant data or the insert operation will fail.
- If a column is not specified in the insert operation, the default value for the column is used (if it exists).

The Insert Query - Class Exercise

- Insert new products into the product table.
- Insert new employees into the employee table (beware of mandatory columns).
- Insert new rows into the relation table [order details] (notice the values in ProductID and OrderID).

The Update Query

- Update revises the data in existing rows.
- Syntax:
  - Update <table>
    set <column name> = Value
    where <where clause>
- Example:

```
UPDATE Products
SET PRICE = PRICE * 1.1, Quantity=Quantity-1
WHERE NAME='T-SHIRT';
```

- Use a sub query (if necessary) in the where clause.

For an update operation to be successful, the user needs to have update privileges and the given value should not violate column and table constraints.
To "delete" one column - set <column> = NULL (if there isn't NOT NULL constraint on the column).

The Update Query - Class Exercise

- 40 units of *Tofu* have been bought, update the data to reflect the new stock value for this item.
- A client asks to add two more units of *Mascarpone Fabioli* to his order (order id - 10258).
- Update all *null* values in the *region* column of the *customers* table to the value of *city* of the relevant row.
- Update each *unitPrice* field in table *[order details]* to have the value of the relevant *unitPrice* field in the *products* table. (Note: This is a 2 tables update.)

The Delete Query

- This query deletes rows by criteria.
- Syntax:
  - Delete from <table>
    Where <clause>
- Example:

```
DELETE FROM ORDERS
WHERE YEAR(OrderDate) < YEAR(GETDATE());
```

- Remarks:
  - "Deleting" specific columns in a row is not considered to be a delete query (but an update to null).
  - "Deleting" an entire column is not considered as a delete query (but an "alter table" – this will be discussed in the following chapter).

When using delete operation there are two options:

- Delete all the rows in the table (if there is no where clause).
- Delete a group of rows that answer a given criteria (the where clause).

Delete operation should be used carefully:

- There can be triggers in the database that perform cascade deletes which can remove relevant data (if someone created them).
- When we don't provide a where clause all the table's data will be removed (as written early).

The Delete Query - Class Exercise

- Delete order 10248 (and [order details] records).
- Delete the order of minimum total quantity.
- Delete customer of *id* value *FRANK*.
- Delete the best customer (based on the quantity of orders).
- Delete the least ordered product. (is this possible?)

Chapter Summary

- We just learnt about the various action query commands:
  - Insert
  - Update
  - Delete
- Handle these queries with care – they revise data.
  These actions can be irreversible.

Questions

Module 07 - Data Definition Language

Contents:

Chapter Goals 4
Data Definition Language (DDL) 5
Data Definition Language (DDL) (contd.) 6
Create Table <Table> 7
Data Types 8
Constraints 9
Primary Key (PK) 10
The Primary Key - Syntax 11
The Primary Key- Syntax (contd.) 12
The Foreign Key (FK) 13
The FK - Syntax 14
The FK – Syntax (contd.) 15
The Constraints - Not Null, Unique 16
The Constraints - Check 17
Create Table - Miscellaneous 18
Drop Table 19
Class Exercise - Create Table 20
Indexes 21
Indexes 22
Indexes (contd.) 23
Index - Syntax 24
Create <view> 25
View 26
View - Remarks 27
View – Remarks (contd.) 28

Class Exercise - View 29
Alter <table> 30
Stored Procedures 31
Stored Procedures (contd.) 32
Alter <Stored Procedure> 33
Stored Procedures and Security 34
Class Exercise – Stored Procedures 35
User-defined Functions 36
User-defined Functions (contd.) 37
Class Exercise – User-defined Functions 38
Difference Between SP and UDF 39
Database Normalization Process 40
Normal Forms 41
Final Exercise – DDL and DML 42
Final Exercise – DDL and DML (contd.) 43
Final Exercise – DDL and DML (contd.) 44
Final Exercise – UOD, ERD, Table Design, DDL, DML and Queries 45
Final Exercise – UOD, ERD, Table Design, DDL, DML and Queries (contd.) 46
Final Exercise 47
Chapter Summary 48

SELA

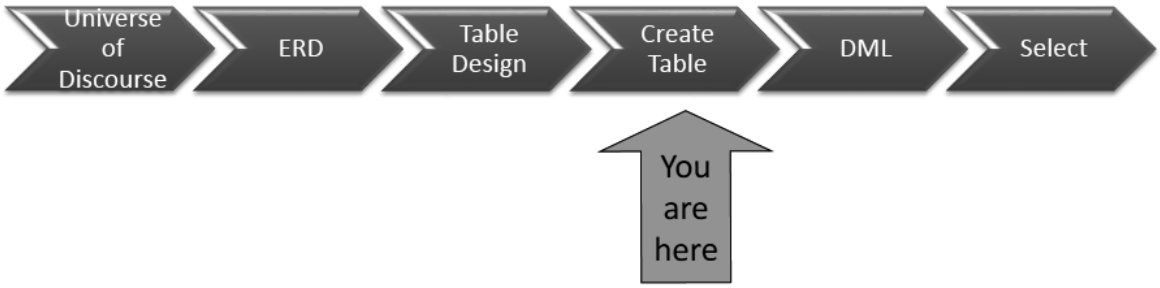
Introduction to Relational Databases and SQL

Module 07 - Data Definition Language

© Copyright SELA software & Education Labs Ltd. | 14-18 Baruch Hirsch St Bnei Brak, 51202 Israel | www.selagroup.com

Chapter Goals

- To learn about DDL.
- To learn the basic DDL commands in SQL.



Data Definition Language (DDL)

- The DML – Data Manipulation Language (chapters 4-5), changes or displays data.
- In contrast, the DDL creates the actual database file.
- The DDL defines the database structure, by:
  - Creating database objects:
    - Create table <table name>
    - Create view <view name>
    - Create index <index name>
  - Deleting database objects:
    - Drop table <table name>
    - Drop view <view name>
    - Drop index <index name>

We will use external tools, not SQL, for this.

In the previous two chapters we discussed the DML which enables the retrieval and manipulation of data. This chapter is all about the DDL (Data Definition Language). The DDL is the SQL means to create the actual database. The DDL enables the creation of database objects, the deletion of database objects and also the modification of database objects.

Data Definition Language (DDL) (contd.)

- Modifying table definitions:
  - Alter table <table name>
- And more.

Create Table <Table>

- It is used to create a new, empty table in the DB file.
- Syntax:

  CREATE TABLE <table name>
  (
  <column name> <data type><column constraints>,
)
- Example:

```
CREATE TABLE Car(
CarID INT PRIMARY KEY,
Model INT CHECK(Model > 1900),
PlateNumber CHAR(10),
EmployeeID INT REFERENCES Employees(EmployeeID));
```

See next pages for explanation

When we want to create a new table we will use the Create statement. In that statement we must provide the columns that we want to inhabit the table and also their name, data type and every relevant constraint that they must have.

Data Types

- This is used to assign the data type to each column.
- Classic types:

Varchar int decimal DateTime binary

When we create a column we need to assign it a data type. Assigning appropriate data types is very important. Using data types can have a performance impact on the database during runtime. If you don't know what data type to pick for a column, you should use the database engine documentation for assistance. Data types can be numeric, alphanumeric, datetime, money and more.

Constraints

- These will place constraints on the insert statement (chapter 5).
- There are different types of constraints:
  - Primary key;
  - Foreign key;
  - Not Null;
  - Unique;
  - Check.

Table constraints are constraints that we put on columns in order to deny some inserted data which doesn't follow the constraint.
Table constraints are one of the barriers that maintains data integrity. They also provide safety hatches against unwanted operations. Even then, they can affect the performance of the database in inserts and updates since they perform a check on the data.

Primary Key (PK)

- The PK will be one column, or a combination of more than one column, in each table.
- The PK column must have a value in each insert statement.
- The PK column's value must be unique in the table.
- This is similar to *not null unique.*

Will be explained later.

The primary key must be unique because it identifies a row.
It can be put on a column or a bunch of columns.
When we use Insert actions we must provide data for the primary key.
Failing to do so will raise an error.

The Primary Key - Syntax

- Option 1:

```
create table <table name>
(
<column name><type> primary key
);
```

- Example:

```
CREATE TABLE Person
(
PersonID INT PRIMARY KEY,
PersonName CHAR (15)
);
```

The Primary Key- Syntax (contd.)

- Option 2 (also called compound primary key):
  - Use this option when the PK contains more than one column (in which case you cannot use option1).

```
Create table <table name> (
<column name><type>,
<column name><type>,
Primary key (<column name>, <column name>)    );
```

- Example:

```
CREATE TABLE Flight (
 AirlineINI CHAR(2),
 FlightNumber INT,
 Date DATETIME,
 Terminal INT,
 PRIMARY KEY (AirlineINI, FlightNumber));
```

Compound primary keys are primary key that contains more than one column.
It is preferable to create an ID column instead of compound keys, since they can interfere in database indexes, which will be discussed later in this chapter.

The Foreign Key (FK)

- See the discussion regarding FKs in Module 4.
- FK – Remarks:
  - A foreign key is a column in a table, which has the same value as a column in another table.
  - A FK could contain more than one column.
  - The FK column usually points to a PK in another table.

The FK - Syntax

- Option 1:
  - If the FK is in one column:

```
Create table <table name> (
<column name><type>,
<column name><type> references <table name>(column name),
);
```

- Example:

```
CREATE TABLE Flight (
AirlineINI CHAR(2)PRIMARY KEY,
FlightNumber INT,
Date DATETIME,
TerminalID INT REFERENCES TERMINAL(TerminalID)
);
```

The FK – Syntax (contd.)

- Option 2 - If the FK is in more than one column:

```
create table <table name> (
<column name><type>,
<column name><type>,
foreign key (<columns names>)
references <table name>(columns names));
```

- Example (based on the PK Option2, regarding the flight table):

```
CREATE TABLE PassengerFlight (
 FlightAirline CHAR(2),
 FlightNumber INT,
 Date DATETIME,
 PassengerName CHAR(20),
 FOREIGN KEY (FlightAirline , FlightNumber) REFERENCES
 FLIGHT(AirlineINI, FlightNumber));
```

Foreign keys that are put in more than one column are very rare.

The Constraints - Not Null, Unique

- Not null:
  - Doesn't allow a null value to be inserted into the column.
    - LastName CHAR(15) NOT NULL,
- Unique:
  - Doesn't allow a value that already exists to be inserted into the column.
    - PlateNumber CHAR(7) UNIQUE,
- Not null unique:
  - This is similar to using a PRIMARY KEY, but...
  - It can be used more than once in a table.

There are more constraints other than, primary and foreign keys.
The not null constraint will check whether you try to insert a null value to a column and if so will raise an error.
The unique constraint will check that inserted data in a column is unique which means that the column doesn't hold the same value in previous rows.
The not null unique is a composition of the two previous constraints.

The Constraints - Check

- Check - Option1:
  - When the Check constraint is on one column only:
  - It restricts the insert statement's values for the column:

```
EmployeeGender CHAR(1)CHECK (EmployeeGender  IN('M','F')),
Grade INT CHECK (Grade BETWEEN 0 AND 100),
```

- Check - Option2:
  - When check is on more than one column:

```
GradeType CHAR(6),
Grade     CHAR(4),
CONSTRAINT GRADES CHECK((GradeType = 'USA' AND Grade IN('A','A-
','+B','C')) OR (GradeType = 'Europe' AND Grade
IN('GOOD','OK','POOR')))
```

The check constraint checks that a given rule isn’t violated when we insert data. The check can get a condition that must be followed. We can put a check constraint on one or more columns. Failing the check will raise an error.

Create Table - Miscellaneous

- Default:
  - If the insert statement utilizes default, the value in the column will be...
  - Salary INT DEFAULT 1000,
  - Use 'default' keyword in that column, in the insert statement.
- Identity:
  - Set an automatic increasing integer value (1,2,3..) for new rows.
  - This is useful for PKs.
  - Use it only once in a table.
    - EmployeeID INT PRIMARY KEY IDENTITY,

When can we have a default value for a column? When we use a default and if no value is provided in an insert statement then the default value will be created for the inserted row. The identity is an automatic increasing integer that is very useful for counters or for PK. An identity can only be used once in a table.

Drop Table

- This command deletes the entire table object.
- If other tables are referenced to this table, the engine won't allow you to delete the table.
- BE CAREFUL when using this!!!
- Syntax:
  - Drop table <table name>
- Example:

```
DROP TABLE Flight;
```

Entropy law:
It is much easier
to destroy than
to rebuild!!!

Drop with a table name is the statement that will enable the deletion of a table. It deletes the table with all its data, so be very careful when you use it. If you have an active relation between the table you try to drop and other tables you won't be able to commit the drop statement.

Class Exercise - Create Table

- Create a table for a company's cars.
  - The table's properties: plate number, frame number, color, model.
- Create a relation table between cars and employees.
  - The relation table's properties: from date, to date (an employee uses the car).
- Remarks:
  - Set a primary key, defaults, checks, and foreign keys.

Indexes

- Why do we need them:
  - An index decreases access time to the data.
  - Without an index, each select statement must execute a sorting algorithm to filter and sort the data - this is time consuming.
  - An index is a physical means (real, kept on the disk) of maintaining sorted “pointers” to the data.
  - When a table is queried, instead of first having to sort it, an index will help the engine retrieve the already sorted data.
  - An index saves a significant amount of query response time.

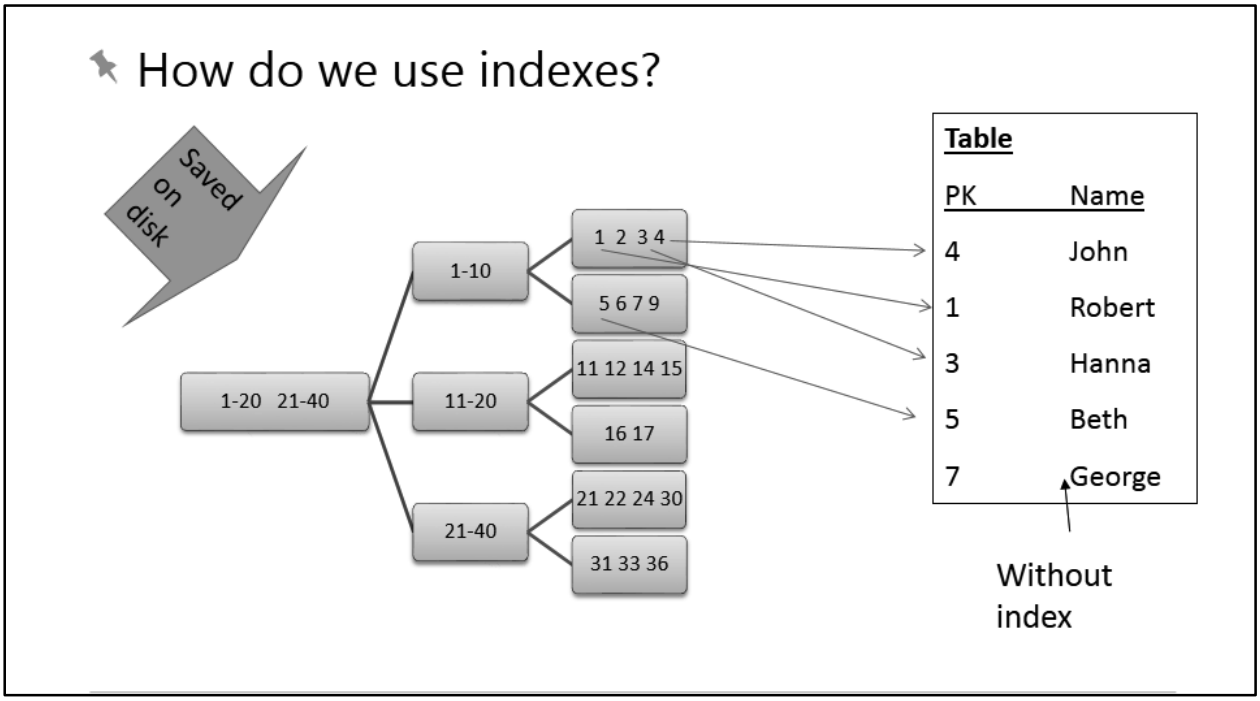
Indexes are a tool of database performance. They can boost the execution time of queries. The downside is in CUD (Create/Update/Delete) operations, where indexes can decrease their execution time (because in every CUD operation the index will be calculated by the database engine).

Indexes

- Why do we need them:
  - An index decreases access time to the data.
  - Without an index, each select statement must execute a sorting algorithm to filter and sort the data - this is time consuming.
  - An index is a physical means (real, kept on the disk) of maintaining sorted “pointers” to the data.
  - When a table is queried, instead of first having to sort it, an index will help the engine retrieve the already sorted data.
  - An index saves a significant amount of query response time.

Indexes are a tool of database performance. They can boost the execution time of queries. The downside is in CUD (Create/Update/Delete) operations, where indexes can decrease their execution time (because in every CUD operation the index will be calculated by the database engine).

Indexes (contd.)



- MySQL uses B-Tree data structure for indexes.
- Indexes updated automatically after each action query.
- Indexes save time in sorting, but consume time when updating, inserting or deleting.
- Create the index on columns that frequently appear in the WHERE clause.

Index - Syntax

- Create index <index name> on <table>(<column name>)

```
CREATE INDEX BYNAME ON Employees(LastName,
  FirstName);
CREATE INDEX BYORDDATE ON Orders(OrderDate);
```

- Drop index <index name>

```
DROP INDEX Employees.BYNAME;
```

Create <view>

- "Saves a select query".
- First, an Example

```
CREATE VIEW ViewEmployeeOrder AS
     SELECT EmployeeID, FirstName, LastName, HireDate, OrderID
     FROM Employees, Orders
     WHERE Employees.EmployeeID = Orders.EmployeeID;
SELECT * FROM ViewEmployeeOrder;
```

Create view

returns the select result

- Why do we need it?
  - For aggregation on aggregation queries (see chapter4).
  - For convenience (reusing the select code).
  - For data control purposes.

In previous chapters we talked about Views. The slide shows how we create a view from a query. The use of view is very common because they are very convenient.

View

- Drop <view>
  - Delete view object:
    - DROP VIEW ViewEmployeeOrder
- Views for data control purposes:
  - Permissions are for tables and views.
  - Create a view that selects certain rows and columns (for example LastName column and manager's rows).
  - Permit the view for the users. Restrict the table's permission from users.

One reason to use a view is for data control. We can create a view in order to restrict users the access for a table or tables. We will create the restriction on the tables and then create a view with permissions for these users. In this way we can gain data control.

View - Remarks

- New columns in view.
  - A computed column in the select will be displayed in the view as:
    - If no name was given - the new column will be called expression.
    - If a column name was given in the select - this name will be the column name in the view.

```
CREATE VIEW VEMP AS
SELECT EmployeeID, YEAR(getdate()) - YEAR(BirthDate)
  AS 'AGE'
FROM Employees
```

  - You can assign a column name in the create view.

```
CREATE VIEW VIEMP (CODE, FULLNAME) AS
SELECT EmployeeID, LastName + FirstName
FROM Employees
```

When we use a view, there are sometimes computed columns that we want to expose. If we do not want to give these columns a name, the database engine will call them expression. In order to be more meaningful, use the rename method we learnt in chapter 4, to rename the calculation or use the view declaration to expose a meaningful name.

View – Remarks (contd.)

- In what ways are view limited:
  - Not all the views are updateable.
  - The create view statement cannot contain the order by statement (in select).

One problem with views is that we can impose order on the view rows by using an order when we create the view. This will be handled by the queries of the view or by the application that consumes the view.

Class Exercise - View

- Create a view for customers and their order details.
  - Join tables: Customers, Orders and Products.
- Create a view for orders and their total cost (aggregation).

Alter <table>

- Allows you to make revisions in the table definitions.

  Syntax:

```
alter table <table>
[ADD <column name> <data type><constraint>];

alter table <table>
[DROP COLUMN <column name> ];
```

- Examples:

```
ALTER TABLE Car
ADD Color CHAR(15);

ALTER TABLE Car
DROP COLUMN Color;
```

Stored Procedures

- A procedure that is stored directly in the database
- Enables the creation of reusable functionality
- Syntax:

```
CREATE PROCEDURE <procedure name>
[List of parameters]
AS
[SP implementation]
```

- Example:

```
CREATE PROCEDURE dbo.GetAllEmployees
AS
SELECT EmployeeID, FirstName, LastName,
     HireDate
From Employees
```

Stored procedures are a very important part of a database.
They enable us to create reusable procedures that are stored directly in the database.
Stored procedures also allow you to put fragments of code or business logic inside the database.
They can help in security since we can grant, execution permissions on stored procedures, and therefore we can minimize the user's ability to corrupt the database.

Stored Procedures (contd.)

- To execute SP
  - Use the SP name
  - Use the EXECUTE statement
    - Example: EXECUTE GetAllEmployees
- Parsing and execution plan are built when SP is created
- Faster than dynamic SQL

When you create a stored procedure, the procedure is compiled by the database engine and an execution plan is built for its better performance. Since this is the case, stored procedure runs faster than a dynamic SQL, that is built inside a software that uses the database.
In order to execute a stored procedure we need to use its name or the EXECUTE statement following by the parameters for the SP (if there are any).

Alter <Stored Procedure>

- Allows you to make revisions in the stored procedure.
- Syntax:

```
ALTER PROCEDURE <procedure name>
[List of parameters]
AS
[SP implementation]
```

- Example:

```
ALTER PROCEDURE dbo.GetAllEmployees
AS
SELECT EmployeeID, FirstName, LastName,
HireDate, PhoneNumber
From Employees
```

Stored Procedures and Security

- We can grant users execute permissions on stored procedures.
- The following roles can create stored procedures:
  - dbo_owner
  - db_ddladmin
  - Other roles need to have GRANT permission in order to create a SP
    - Example:
      GRANT CREATE PROCEDURE TO [Some Database Role]

Stored procedures can help to make the database more secure.
We can grant execution permission on stored procedures and not allow permissions on objects manipulated by those stored procedures.
Not all users can create stored procedures. You need to be on the dbo_owner role or db_ddladmin role, in order to be able to create stored procedures. If you don't exist in those roles, you need to be granted permission by someone who has those permissions, in order to create or alter a stored procedure.

Class Exercise – Stored Procedures

- Create a stored procedure that returns all the customers and their order details
  - Join tables: Customers, Orders and Products.
- Alter the procedure to get a customer id and return only that customer and his order details

User-defined Functions

- A function that is stored directly in the database
- Enables a mechanism for extending the database functionality
- Kinds of UDF
  - Scalar function – returns a single value
  - Table function – returns a table that contains zero or more rows
- Syntax:
  CREATE FUNCTION <FunctionName>([List of parameters])
  RETURNS [Data type]
  [Function body]

UDFs resemble SP, but they are used for other purposes. They enable us to extend the database functionality.
They can be used anywhere in a given SQL statement, as opposed to SP, that can't be used anywhere.
If you have a simple and small functionality, UDFs are the place to start with. UDFs are divided into scalar functions that return a single scalar value and into table functions, that return a table that contain zero or more returning rows.

User-defined Functions (contd.)

- Example:

```
CREATE FUNCTION dbo.CelsiustoFahrenheit(Celsius FLOAT)
RETURNS FLOAT AS
RETURN (Celsius * 1.8) + 32
```

- Calling a UDF is simple – just use its name and if needed give it parameters.
  - Example:

```
SELECT dbo.CelsiustoFahrenheit(38)
```

Class Exercise – User-defined Functions

- Create a UDF that transforms centimeters to inches.
- Create a UDF that returns a table of all the customers with their order details.
- Create a UDF that derives speed, from a given distance and time
  - V (speed) = X (distance) * T (time)

Difference Between SP and UDF

- UDFs can be used anywhere in a given SQL statement (WHERE/HAVING/SELECT)
- UDFs that return tables can be treated as another table inside a JOIN statement
- Different syntax
- UDFs extend database functionality, whereas SP puts business logic inside the database

Database Normalization Process

- The process of organizing data in a given database.
- Has the following goals:
  - Helps preventing loss of data integrity.
  - Prevents unnecessary data.
  - Ensuring data dependencies makes sense.
- There are guidelines for normalization
  - Makes sure that a database can be called normalized.

Normalization is a process that helps to organize a given database through strict normal form rules.
Normalization has the following goals:
Helps preventing loss of data integrity meaning prevents anomalies in insert, update and delete operations.
Prevents unnecessary data meaning preventing the storage of the same data in more than one table.
Ensuring data dependencies makes sense, meaning storing related data in the same table.
There are guidelines for normalization which are numbered: First normal form (1NF) to Sixth normal form (6NF)
The guidelines makes sure that a database can be called normalized but are still guidelines that can be avoided when they don't answer given business rules. 1NF – 4NF are commonly used.
It's very rarely to see 5NF and 6NF.

Normal Forms

Taken from Wikipedia:

Normal form	Definition
First Normal Form (1NF)	Table faithfully represents a relation and has no repeating groups
Second Normal Form (2NF)	No non-prime attribute in the table is functionally dependent on a part of a candidate key
Third Normal Form (3NF)	Every non-prime attribute is non-transitively dependent on every key of the table
Forth Normal Form (4NF)	Every non-trivial multi valued dependency in the table is a dependency on a superkey
Fifth Normal Form (5NF)	Every non-trivial join dependency in the table is implied by the superkeys of the table
Sixth Normal Form (6NF)	Table features no non-trivial join dependencies at all

The higher the normal form, the table is less vulnerable to inconsistencies and change operation anomalies.
The above table definitions are only applicable to individual tables.

Final Exercise – DDL and DML

- SQL – create, update, delete, insert, alter and more…
  - The SQL statements in this exercise relate to the a3v32.db database, which is located in each student's home directory.
  - You should put documentation inside your code (/* */). The documentation will explain the query (or the sub query) role and will help the reader understand the purposes of the written code.
- Write SQL queries for the following questions (submit a printout and a diskette):
- 1. We want to build a golden customer club. We will maintain details about big customers. Create a new table named: gold_customer that will contain the following columns:

Final Exercise – DDL and DML (contd.)

- customer id (cust_id) integer PK
- year (cust_ord_year) char (4).
- yearly amount (cust_year_amnt) long/

2. Insert into the gold_customer table the code, year 1993 and amount for all the customers who placed more than 5 orders in 1993.

3. Add 10% to employees' salaries for employees whose period of employment in the company exceeds 11 years.

4. Add to the Emp table one more column called Em_Height.

Final Exercise – DDL and DML (contd.)

5. Drop the sex column from the Emp table.
6. Modify the Em-Height column from numeric into character.
7. Declare the employee's first name, last name and start date and date of birth as indexes.
8. Raise by 5% the product prices for products whose quantity sold in 1994 exceeded the average sales per product.

Final Exercise – UOD, ERD, Table Design, DDL, DML and Queries

- This final exercise is an attempt to practice practically every topic so far studied, compromising specification (UOD) volume, however comprising a full process. Starting with a spec and escalating all the way towards a live set of properly related tables, a DB.

A. Create an ERD for the following spec:

- Employees work in different departments within the company.
- Employees have different jobs, which they perform.
- An employee works in exactly one department at a time.
- An employee has exactly one job at a time.

Final Exercise – UOD, ERD, Table Design, DDL, DML and Queries (contd.)

- We would want to keep a record of each job performed by an employee. We would want to be able to extract department data as well. We would also like to extract start and end dates for each such job.
- Each employee reports to his / her direct manager (except maybe the CEO).
- Each department has exactly one manager who is an employee.

Final Exercise

A. Create the table model for your ERD.
B. Create a DB, create the tables and the relevant PKs and FKs.
C. Append dummy data to query on.
D. Write queries for the following :

1. Display employee details and the name of the department they currently work in.
2. Display employee id, name, job title, department name, start date and end date for employee named, 'Jack Daniels'.
3. Display the total number of employees in each department which has more than 10 employees.
4. Who's the employee that has been recruited first? recruited last?
5. Display the employees who have switched jobs since recruitment.

Chapter Summary

- In the Process of Data Definition we:

Created tables

Created PK, FK and constraints

Created views

Created indexes

Dropped views

Altered the table definitions

Dropped indexes

Created Stored Procedures

Dropped tables

Created User-defined Functions

Questions