University of Washington

iSchool Info 330

# Module 02 - Enhancing a Database

In this module, we **continue** talking about **how databases are designed** and **then move on** to what **additional features** can be added to enhance a database's design. You will learn how databases are made better by **restricting what is allowed** into databases and by forcing people to **use an abstraction layer** to access them.

## Outline

Here is a general outline of what we will be doing this module:

|  |
| --- |
| **Module02: Enhancing a Database** |
| Session01 Lectures and Labs < 110 mins |
| Normalization Issues and Common Errors - 30 |
| Lab: Is this existing database well designed? - 20 |
| Constraints - 30 |
| Lab: What constraints would you recommend? - 20 |
| Session02 - Lab |
| Lab: Create a database with constraints - 50 |
| Session03 Lectures and Lab < 110 mins |
| Abstraction Layers - 50 |
| Lab: How can you create a database with abstraction layers? - 50 |

**Note**: Times are only estimates and may change without notice!

# Session01 < 110 mins

In this session, we explore a couple of relational databases, **look for normalization issues and other errors**. We will also look at adding constraints to a database and how it is beneficial.

## Normalization Issues and Common Errors - 30

We have learned what normalization is and how it should be used when designing databases. In this lecture, we will **look at two databases** and **note which normalization errors** became part of their designs. We will also **review other common errors** found in these databases, and talk about how even today these same issues are common in the industry!

### The Pubs Database

To learn about the process of making databases, it is sometimes useful to **look at existing** databases and **ask what could make it better**. Pubs is a good example of a database that has lots of room for improvements!

"For roughly the **first ten years of SQL Server**, when documentation was still printed on paper—a lot of paper— the Pubs database was the database for SQL Server documentation. Pubs was **originally developed by Sybase** and came to Microsoft in **the early 90s** via the Microsoft-Sybase partnership.

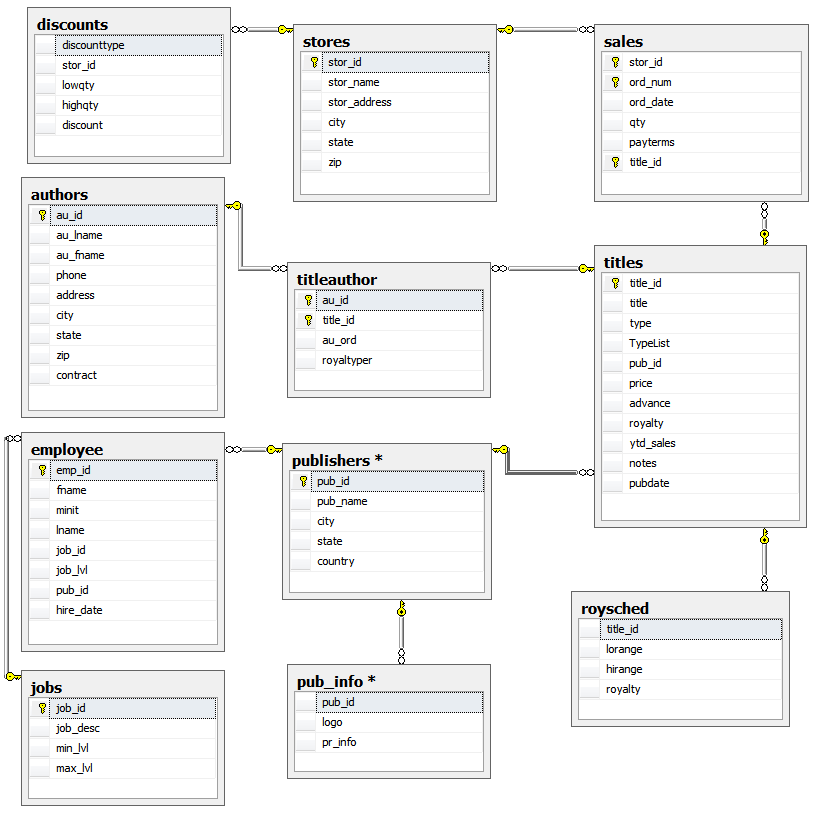
Although this database served its purpose for documentation for many years, it wasn’t free of complaints. People thought that Pubs was too simple and that the database needs of a book publishing company had nothing to do with the database needs of real SQL Server customers. After almost ten years in the market, it was time for something different." (<https://blogs.msdn.microsoft.com/writingdata_services/2013/09/09/revisited-the-story-of-pubs-northwind-and-adventureworks>, Matt Stroshane, 2017)

Let's look at this database and see what issues we can spot.

**TIP**: Your instructor will walk you through this review, but **later you will be asked to repeat this process** on our own.

Below are the tables that make up the Pubs database:

* authors
* discounts
* employee
* jobs
* pub\_info
* publishers
* roysched
* sales
* stores
* titleauthor
* titles



##### Figure 1: An ERD of the Pubs Database

Here is some code you could use in MS SQL Server to **list the tables** of this database:

use Pubs;

Select \*

From Sys.Tables

Where type = 'u' -- U is for "User Created"

Order By Name;

### Reviewing A Table's Data

You might start your inspection by **looking at each table's data** and could do so using this command:

Select \* From Sales;

In this example, the results are returned as so:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **stor\_id** | **ord\_num** | **ord\_date** | **qty** | **payterms** | **title\_id** |
| 6380 | 6871 | 9/14/1994 | 5 | Net 60 | BU1032 |
| 6380 | 722a | 9/13/1994 | 3 | Net 60 | PS2091 |
| 7066 | A2976 | 5/24/1993 | 50 | Net 30 | PC8888 |
| 7066 | QA7442.3 | 9/13/1994 | 75 | ON invoice | PS2091 |
| 7067 | D4482 | 9/14/1994 | 10 | Net 60 | PS2091 |
| 7067 | P2121 | 6/15/1992 | 40 | Net 30 | TC3218 |
| 7067 | P2121 | 6/15/1992 | 20 | Net 30 | TC4203 |
| 7067 | P2121 | 6/15/1992 | 20 | Net 30 | TC7777 |
| 7131 | N914008 | 9/14/1994 | 20 | Net 30 | PS2091 |
| 7131 | N914014 | 9/14/1994 | 25 | Net 30 | MC3021 |
| 7131 | P3087a | 5/29/1993 | 20 | Net 60 | PS1372 |
| 7131 | P3087a | 5/29/1993 | 25 | Net 60 | PS2106 |
| 7131 | P3087a | 5/29/1993 | 15 | Net 60 | PS3333 |
| 7131 | P3087a | 5/29/1993 | 25 | Net 60 | PS7777 |
| 7896 | QQ2299 | 10/28/1993 | 15 | Net 60 | BU7832 |
| 7896 | TQ456 | 12/12/1993 | 10 | Net 60 | MC2222 |
| 7896 | X999 | 2/21/1993 | 35 | ON invoice | BU2075 |
| 8042 | 423LL922 | 9/14/1994 | 15 | ON invoice | MC3021 |
| 8042 | 423LL930 | 9/14/1994 | 10 | ON invoice | BU1032 |
| 8042 | P723 | 3/11/1993 | 25 | Net 30 | BU1111 |
| 8042 | QA879.1 | 5/22/1993 | 30 | Net 30 | PC1035 |

Table 1: The results of the query, "select all data from sales"

You then might **locate normalization issues** by asking questions like:

* Which values can be considered candidate keys?
* Are there repeating values?
* Are there Multi-part values?
* Are there Multi-value fields?

#### Question: Do you see any examples of repeating, multi-part, or multi-value data?

You can also ask what kind of errors might arise due to **Insert, Update, or Delete anomalies** due to the table's design. There are some simple examples of these anomalies on Wikipedia:

<https://en.wikipedia.org/wiki/Database_normalization> (external site)

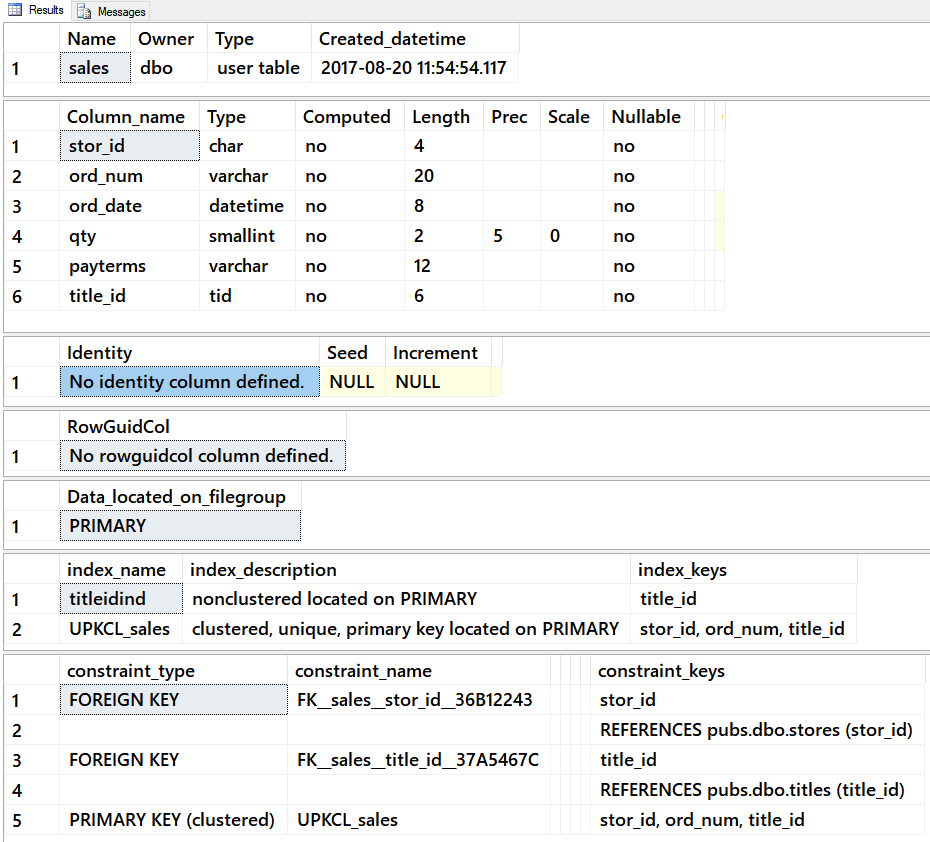
#### Question: Do you see any possible modification anomaly issues using this table design?

### Reviewing A Tables Structure

Besides the data, you also need to **look at the structure of a table**. For example, to see information about the design of an individual table in Microsoft's Pubs sample database you can use the MS SQL Server procedure:

Exec sp\_help sales;

Here is the output from sp\_help:



##### Figure 2: The results of sp\_help using the “authors” table in Pubs

If you were using the **Oracle** RDMS or the Open Source **MySQL** RDMS (owned by Oracle now) the command would be similar. Here is what that **command** would look like, assuming there was a database with a sales table:

**DESCRIBE** 'sales';

### Naming Conventions and Data Type Consistency

Once you find information on the structure of a table, you try to i**dentify issues with naming conventions and data type** consistency.

Users of data should expect the names of its objects (tables and columns) to be **predictable, consistent, and self-describing**. When these are not the case it wastes time and money, adds to a user's stress, and increases the possibility of errors.

When looking at a current table's design ask the following questions:

* Is the table's name self-describing?
* Is each table's naming convention consistent and predictable?
* Is each column's name self-describing?
* Is each column's naming convention consistent and predictable?
* Is each column's data type consistent and predictable?
* Does the choice of Primary Key column(s) seem obvious?

Some issues, such as a column’s data type, will seldom be noticed by end-users, while others can be an ever-present frustration. You should always try to **make the experience of using the database as productive and stress-free as possible**.

## Lab 1: Were These Existing Tables Designed Well? - 20

In this lab, you will **review some existing tables** and ask yourself, "Were these existing tables designed well?" We will be using Microsoft's Northwind database.

"**Northwind was a good database for SQL Server – a better fit than Pubs**. By **also** appearing **in the Access documentation**, it would help show customers how Access and SQL Server could accomplish many of the same tasks. To illustrate new SQL Server features, the team was also free to make additive changes to the database. The summation of that work is the Northwind database script that **shipped with SQL Server 2000**.

By the time SQL Server 2000 released, much of the SQL Server documentation still referenced Pubs. This is why both databases shipped with the product. " (<https://blogs.msdn.microsoft.com/writingdata_services/2013/09/09/revisited-the-story-of-pubs-northwind-and-adventureworks>, Matt Stroshane, 2017)

You will work on your own for the first 10 minutes, then we will review the answers together in the last 10 minutes.

**Note**: This lab is done individually or with a group of up to three people.

### Step 1: Review Database Tables

Run the following code in a SQL query editor. Review the structure and data of the orders and order details tables, then note any design issues you find.

Use Northwind;

Select \*

From Sys.Tables

Where type = 'u' -- U is for "User Created"

Order By Name;

Select \* From Orders;

Select \* From [Order Details];

Exec sp\_help 'Orders';

Exec sp\_help 'Order Details';

### Step 2: Review Your Work

Now, you will review your work with your instructor.

**NOTE: Unlike assignments, labs do not need to be turned in to Canvas!**

## Constraints - 30

### Constraints

Having a **database** that you **trust** is as important as it is being **easy to use**. **In order to trust data, it must be consistent**. Database designers and administrators place constraints on columns and rows to keep their data consistent. Consistent table row data (**entity integrity**), consistent column data (**domain integrity**), and consistent data between connected rows (**referential integrity**) it is vital for making the database usable and trustworthy.

There are **several constraints** in RDMS software. The ones we cover in this course are:

* Primary Key
* Foreign Key
* Unique
* Check
* Nullability

Most database software includes **options to automatically constrain values**. For example, Oracle RDMS has the following information about its constraints:

"Use a constraint to define an integrity constraint—a rule that restricts the values in a database**. Oracle Database lets you create six types of constraints** and lets you declare them in two ways.

The six types of integrity constraint are described briefly here and more fully in ["Semantics "](https://docs.oracle.com/cd/B12037_01/server.101/b10759/clauses002.htm#i1002038):

* **A NOT NULL** constraint prohibits a database value from being null.
* A unique constraint prohibits multiple rows from having the same value in the same column or combination of columns but allows some values to be null.
* A **primary key** constraint combines a NOT NULL constraint and a unique constraint in a single declaration. That is, it prohibits multiple rows from having the same value in the same column or combination of columns and prohibits values from being null.
* A **foreign key** constraint requires values in one table to match values in another table.
* A **check** constraint requires a value in the database to comply with a specified condition.
* A **REF** column by definition references an object in another object type or in a relational table. A REF constraint lets you further describe the relationship between the REF column and the object it references." (<https://docs.oracle.com/cd/B12037_01/server.101/b10759/clauses002.htm>, 2017)

While the **SQL Server** RDMS has the following information about its constraints:

"SQL Server **supports the following classes of constraints**:

**NOT NULL** specifies that the column does not accept NULL values. For more information, see Allowing Null Values.

**CHECK** constraints enforce domain integrity by limiting the values that can be put in a column. For more information, see CHECK Constraints.

...

**UNIQUE** constraints enforce the uniqueness of the values in a set of columns.

In a UNIQUE constraint, no two rows in the table can have the same value for the columns. Primary keys also enforce uniqueness, but primary keys do not allow for NULL as one of the unique values. For more information, see UNIQUE Constraints.

**PRIMARY KEY** constraints identify the column or set of columns that have values that uniquely identify a row in a table. For more information, see PRIMARY KEY Constraints.

...

**FOREIGN KEY** constraints identify and enforce the relationships between tables. For more information, see FOREIGN KEY Constraints." ( <https://technet.microsoft.com/en-us/library/ms189862(v=sql.105).aspx> , 2017)

### Creating Constraints in MS SQL Server

**Continuing our use of SQL Server** as our example RDMS, let's look at how you create tables with constraints.

You can create a table and **include the constraint immediately**.

Create -- DROP

Table Demo (

col1 int **Primary Key**,

col2 int **Unique**,

col3 int **Check** (col3 > 0),

col4 int **Foreign** **Key** References Demo(col2),

col5 int **Default**(0),

col6 int **NOT NULL**

);

You can also create the table first and then **add the constraints afterward**.

**CREATE**

**TABLE** dbo.Products (

ProductID int NOT NULL,

ProductName nvarchar (40) NOT NULL,

SupplierID int NULL,

UnitPrice money NULL

UnitsInStock NULL

);

**ALTER TABLE** dbo.Products

**ADD CONSTRAINT** PK\_Products PRIMARY KEY CLUSTERED (ProductID);

Go

#### Primary Keys Constraints

Primary Key (PK) constraints allow only **unique**, **non-null**, values into a column. They also **indicate** which column(s) has been **chosen**, from among the candidate keys, **to identify one row from another**. In addition, you can have **only one** Primary Key constraint **per table**.

CREATE TABLE dbo.Customers (

CustomerID int **primary key**,

CustomerName nvarchar(100)

*);*

#### Unique Constraints

Unique constraints allow only **unique** values into a column. These allow **one null** value. You can have **multiple** unique constraints **per table**.

CREATE TABLE dbo.Customers (

CustomerID int primary key,

CustomerName nvarchar(100),

CustomerEmail nvarchar(100) **unique**

);

#### Foreign Key Constraints

Foreign Key (FK) constraints allow **only** values into a column **if they match another value in a referenced column**. Usually, these columns are in separate tables, but they do not have to be. You can have **multiple** FKs **per table.**

Create Table Orders (

OrderId int primary key,

OrderDate date,

CustomerId int **foreign key** references Customers(CustomerID)

);

#### Check Constraints

Check constraints allow only values that **match** a **pattern or condition**. For example, a value must be greater than another, or a value must match the pattern of a typical US phone number ((206)111-2222). You can have **multiple** check constraints **per table**.

CREATE TABLE dbo.Customers (

CustomerID int primary key,

CustomerName nvarchar(100),

CustomerEmail nvarchar(100) unique,

CustomerPhone nvarchar(10)

**check** (CustomerPhone like '**(**[0-9][0-9][0-9]**)**[0-9][0-9][0-9]**-**[0-9][0-9][0-9][0-9]')

);

Here are some **additional patterns**:

**check** (

PostalCode LIKE '[0-9][0-9][0-9][0-9][0-9]' OR

PostalCode LIKE '[0-9][0-9][0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]' OR

PostalCode LIKE '[A-Y][0-9][A-Z][0-9][A-Z][0-9]')

**check** (Salary > 10.00 and Salary < 150000.00)

**check** (EmployeeType in ('full-time','part-time','short-time'))

**check** (email like '%@%.%')

#### Regular Expression Check Constraints

Regular Expressions are used to **find complex patterns in a string of text**. These are powerful tools, but can be **confusing** to use **without experience**.

"A regular expression (regex or regexp for short) is a special text string for describing a search pattern. You can think of regular expressions as wildcards on steroids. You are probably familiar with wildcard notations such as **\*.txt** **to find all text files** in a file manager. **The regex equivalent is ^.\*\.txt$." (**<https://www.regular-expressions.info/>**,** 2017**)**

While MS SQL Server's **T-SQL language does not include** a simple way to use Regular Expressions, although this can be done using advanced features, **other RDMS do**. Here is an example of **Oracle's** PL SQL language:

CHECK (**REGEXP\_LIKE** (emailID, '^(\S+)\@(\S+)\.(\S+)$')))

#### Not Null "Constraint"

Not Null constraints, **do not allow null values** in a column.

Create Table Orders (

OrderId int primary key **not null**,

OrderDate date **not null**,

CustomerId int foreign key references Customers(CustomerID) **not null**

);

#### Default "Constraint"

Default constraints allow an implicit default value to be placed in a column if one is not explicitly given. While not actually constraining a value, it is classified as a constraint in some Microsoft documentation.

Create Table Orders (

OrderId int primary key not null,

OrderDate date not null default GetDate(),

CustomerId int foreign key references Customers(CustomerID) not null

)

## Lab 2: What Constraints Would You Recommend? - 20

In this lab, you will examine an ERD and consider what constrains you would recommend for this database.

You will work on your own for the first 10 minutes then we will review the answers together in the last 10 minutes.

***Important****: I do not expect you to know how to do this! I want you to think about how you might do this, try out some of your ideas, and talk about it with others. Expect to not be as good at this as someone who has been trained in data.*

**Note**: This lab can be done individually or with a group of up to three people.

### Step 1: Review the ERD

Review the structure of this ERD.



Figure 3: An ERD of the Northwind database

### Step 2: Identifying the Constraints

Using this database design, choose columns in the database where you think a constraint would be beneficial. Provide three examples for each the following MS SQL Server constraints.

* Primary Key
* Unique
* Foreign Key
* Check
* Not Null
* Default

### Step 3: Review Your Work

Now, you will review your work with your instructor.

**NOTE: Unlike assignments, labs do not need to be turned in to Canvas!**

# Session02 - Lab < 50

Session 02, is a hand-on lab normally monitored by your TA. You are tasked to perform the following activities during this lab session.

## Lab 3: Create A Database With Constraints - 50

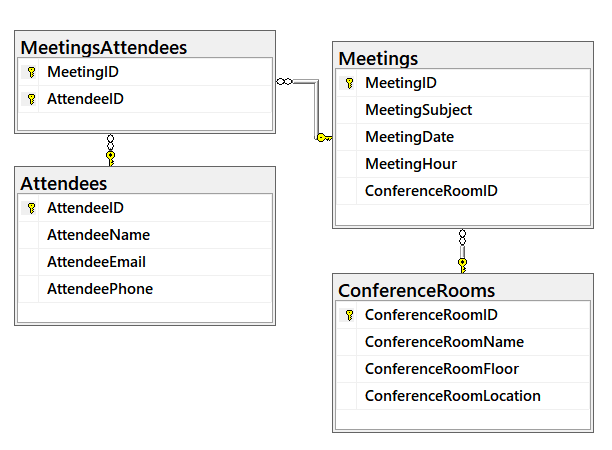
In this lab, you will create a database using a provided ERD and meta-data spreadsheet.

You will work on your own for the first 30 minutes then review the answers together in the last 20 minutes.

**Note**: This lab can be done individually or with a group of up to three people.

### Step 1: Review the ERD

Review the structure of this ERD.



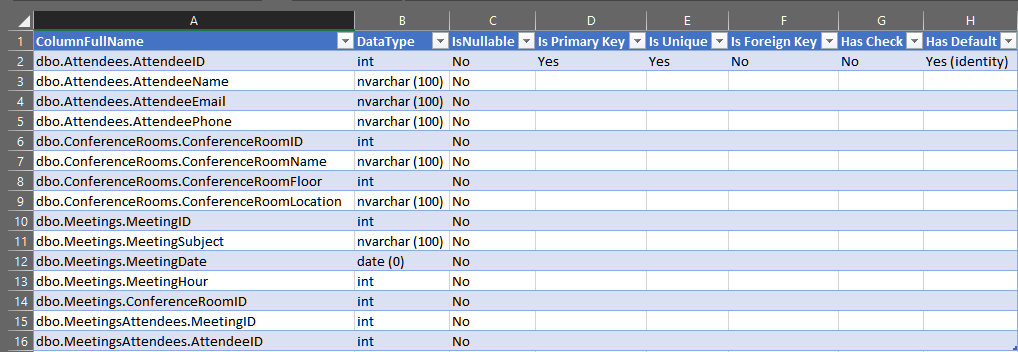
##### Figure 4: An ERD for Lab 3

### Step 2: Review a Meta-Data Spreadsheet

This lab includes a meta-data spreadsheet that describes the structure of the database. Please review this spreadsheet and make sure you understand how it is describing the database you will create in this lab.

### Step 3: Plan Constraints

Fill in the constraints section of the spreadsheet. The first row has been filled in for you as an example.



##### Figure 5: The Lab 3 meta-data spreadsheet

### Step 4: Create a Database

Using the metadata spreadsheet and the ERD, create a new database called YourNameHere\_Mod2\_Lab3 and include all the tables as planned. You can use code from the *Module02Code-Common Database Options.sql* file to create your database, tables, and constraints.

### Step 5: Review Your Work

Now, you will review your work with your instructor.

**NOTE: Unlike assignments, labs do not need to be turned in to Canvas!**

# Session03 < 110 mins

## Abstraction Layers - 50

Abstraction layers are a common concept used in software development.

"In computing, an abstraction layer or abstraction level **is a way of hiding the implementation details of a particular set of functionality**, allowing the separation of concerns to facilitate interoperability and platform independence. Software models that use layers of abstraction include the OSI 7-layer model for network protocols, the OpenGL graphics drawing library, and the byte stream input/output (I/O) model originated by Unix and adopted by MS-DOS, Linux, and most other modern operating systems[citation needed]." (<https://en.wikipedia.org/wiki/Abstraction_layer>, 2017)

In a **well-designed database**, the underlying structure of the database can **grow and change over time with little disruption** to how people use the database. One of the simplest and most effective ways to do this is by adding an abstraction layer of software objects between the tables you make and the people/software that uses those tables.

MS SQL Server, and most other RDMS, provide several tools to **create a database's abstraction layer**.

* **Views**
* **Functions**
* **Stored Procedures**

### Views

While the purpose of a table is to store data, **the purpose of a view is the present data**. This is done by **saving** a SQL Select statement in the database **as a text string**. When you "select" data from the view, it runs the stored code and **treats the result of the query as if it were a table**. This "table" of results is held in memory and its data can be ordered, filtered, or combined with other data.

"In database theory, a view **is the result set of a stored query** on the data, which the database users can query just as they would in a persistent database collection object. This pre-established query command is kept in the database dictionary. Unlike ordinary base tables in a relational database, a view **does not form part of the physical schema**: as a result set, it is a virtual table c**omputed or collated dynamically from data** in the database when access to that view is requested. Changes applied to the data in a relevant underlying table are reflected in the data shown in subsequent invocations of the view. **In some NoSQL databases, views are the only way to query data.**" (<https://en.wikipedia.org/wiki/View_(SQL)> , 2017)

You should consider creating a view for every table in a database. Since it is easy to do and may provide the benefits of an abstraction layer as the database evolves. For example, if I create a Customer table using the following code:

**Create Table** dbo.Customers (

CustomerID int primary key,

CustomerName nvarchar(100),

CustomerEmail nvarchar(100) unique,

CustomerPhone nvarchar(10)

check (CustomerPhone like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9]')

);

I would also create a view for this table with this code:

**Create View** dbo.vCustomers

As

Select

CustomerID,

CustomerName,

CustomerEmail,

CustomerPhone

From dbo.Customers;

I would then use the view by selecting from it as if it were a table:

Select \* from vCustomers;

**If** later the table is **re-designed** to include both a first and last name, people or software can keep using the view as they have been, as long as **I change the code to hide the change in the database's implementation**.

**Alter** View dbo.vCustomers

As

Select

CustomerID,

**CustomerName = CustomerFirstName + ' ' + CustomerLastName**,

CustomerEmail,

CustomerPhone

From dbo.Customers;

#### Naming Conventions

When creating tables and views you should consistently apply a naming convention. Two to consider are:

* Tables are named without a prefix while views are named with a prefix. Example Customers and **v**Customers.
* Tables are named without a prefix while views are named with a prefix. Example **tbl**Customers and Customers.

Other variations are acceptable as long as it is consistent in the database you are currently designing.

**NOTE**: We will talk more about views in a later module.

### Functions

Functions are very similar to view, in that they are **saved select statements**, but allow for filtering **parameters** to be included in their definition. We will talk about these in a later module.

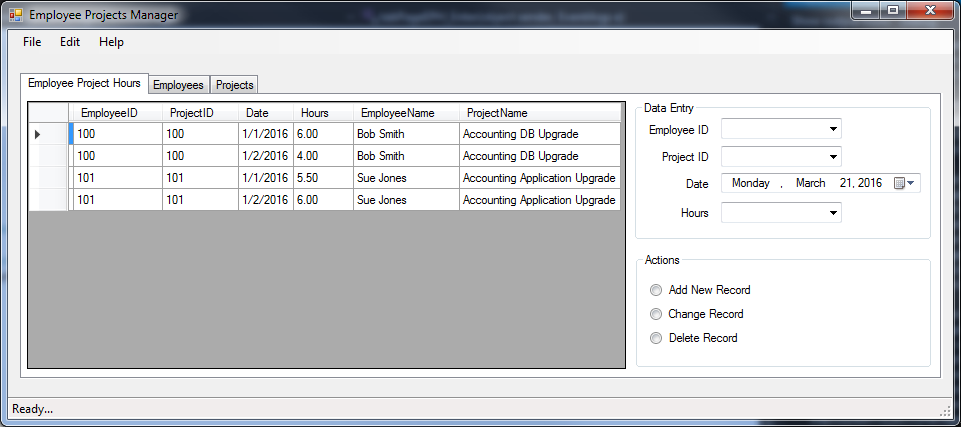
### Stored Procedures

Functions are very like views as well. They are also similar to functions in that they allow for filtering **parameters** to be included in their definition. One big difference is that stored procedures allow you to store not only select statements, but **also insert, update, and delete statements** as well. We will talk about these in a later module.

### Application Development

All applications use data and most store that data in a database. This is because data used in an application always runs in the computer’s memory, but disappears once the application closes! So, an application's data must be saved somewhere else if it is to be used again. **A relational database is one of the most common places to store application data.**

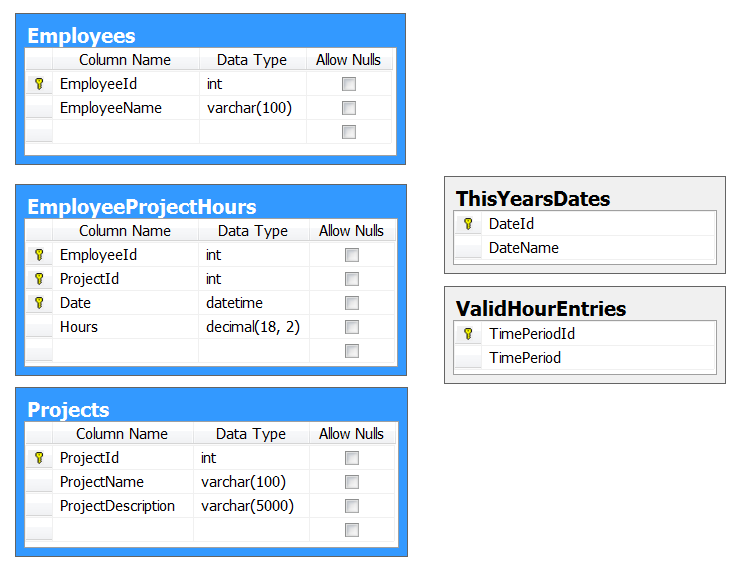
Consider a simple application that will allow employees and managers to tracks the hours that **employees work on a given project**.



##### Figure 6: A simple data-driven application's UI

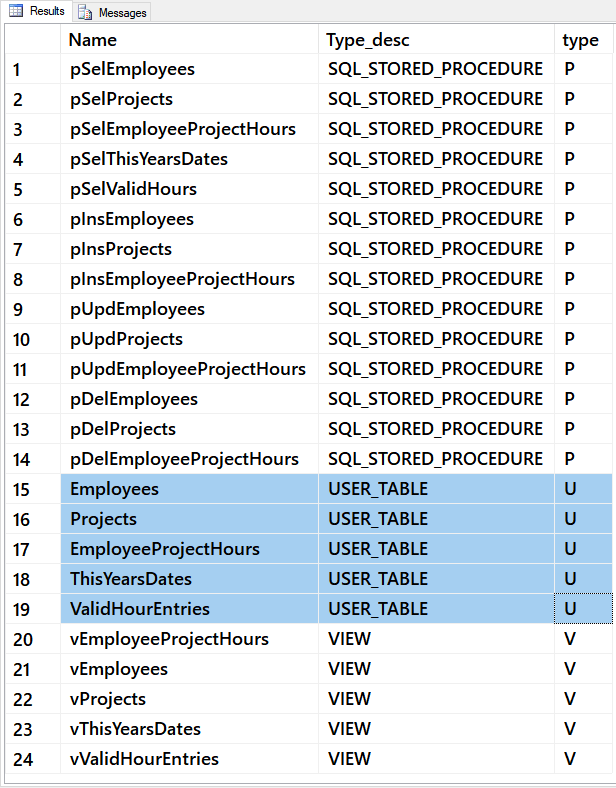
This application can **create, read, update**, and **delete** rows of data. The application's data is stored in a relational database. The database design is shown in the following image.

**Tip**: Though I think the acronym is silly, **create, read, update, and delete** are sometimes referred to as **CRUD**.



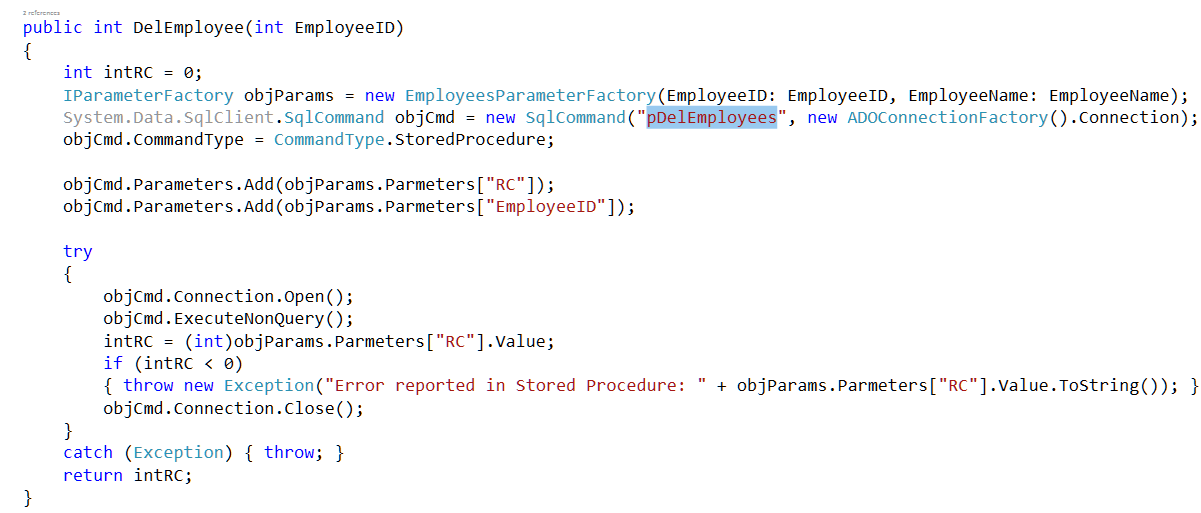
##### Figure 7: A simple database for storing employee project hours

**Each table** in the database **includes a view or stored procedure** for reading data (select) and stored procedures for adding data (insert), changing data (update) and deleting data (delete).



##### Figure 8: Stored Procedures in an Employee Project Hours database

These stored procedures and views are accessed by the application through functions that call the database abstraction objects.



##### Figure 9: A C# function that calls a database stored procedure

**Note**: Later in this course, you will learn how to create very simple reporting applications without having to program outside of the SQL language.

## Lab 4: Create A Database With Abstraction Layers - 50

In this lab, you will create a database with tables, constraints, and views.

**Note**: We will add stored procedures to our databases in later modules.

You will work on your own for the first 30 minutes then we will review the answers together in the last 20 minutes.

**Note**: This lab can be done individually or with a group of up to three people.

### Step 1: Review the ERD

Review the structure of this ERD. Ask yourself, "what purpose does each able serve?"

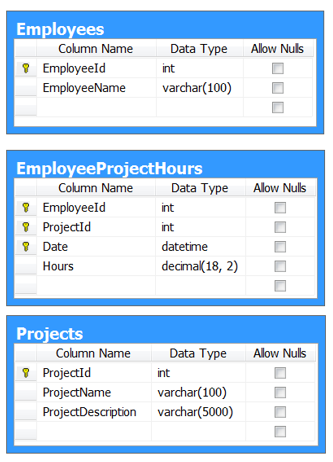
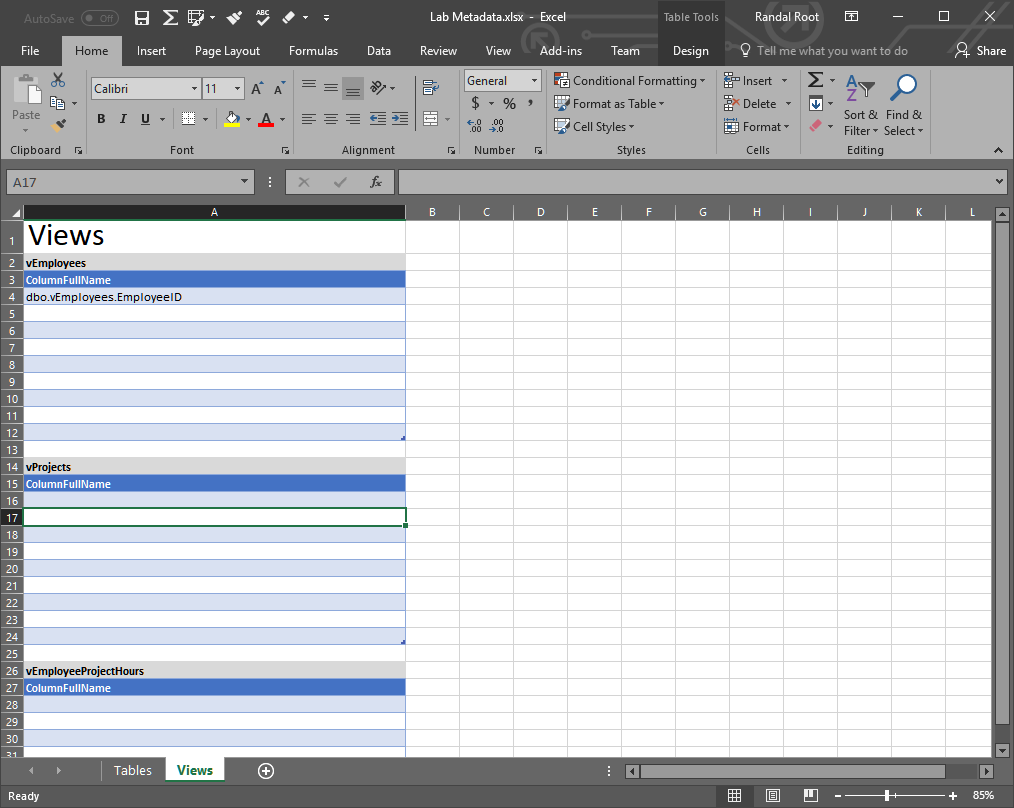


Figure 10: A Simple ERD for Lab 4

### Step 2: Create a Meta-Data Spreadsheet

This lab has you create your own meta-data spreadsheet which describes the structure of the database. A starter spreadsheet is included with the lab files.



##### Lab Figure 11: The Lab 4 meta-data spreadsheet

### Step 3: Plan Constraints

Fill in the constraints section of the spreadsheet. The first row has been filled in for you as an example.

### Step 4: Plan the Views

This time you will also include a set of views, one for each of the three tables.

### Step 5: Create a Database

Using the metadata spreadsheet create a new database called YourNameHere\_Mod2\_Lab4 and include all the tables as planned. You can use code from the *Module02Code-Common Database Options.sql* file to create your database, tables, views, and constraints.

### Step 6: Review Your Work

Now, you will review your work with your instructor.

**NOTE: Unlike assignments, labs do not need to be turned in to Canvas!**