

Week 14

ICT50220 (ICTPRG430 ICTPRG549 OOP Cluster)

- How to plan data for the database for AT3
- Reusable styles
- Multiwindows GUI
- SQL scripting
- Two foreign keys on the same table
- UI planning for AT3 (stack panel, date picker)



Why does one need more than one table?

Using multiple tables in a relational database instead of putting all data in one table is essential for organising and efficiently managing data. This approach adheres to the **principles of database normalisation**, which offers several benefits:

1. **Minimises Data Redundancy:** Multiple tables allow you to store related information separately, reducing the need to duplicate data. This **minimises the risk of inconsistencies** and saves storage space.
2. **Improves Data Integrity:** By avoiding redundant data, you **reduce the chances of errors and inconsistencies in your database. Updates and modifications are less error-prone, ensuring data accuracy.**
3. **Enhances Query Performance:** Organizing data into multiple tables enables the use of indexes and optimised query execution plans. This leads to **faster and more efficient data retrieval, especially when dealing with large datasets.**
4. **Facilitates Data Maintenance:** Maintaining and updating specific data pieces in their respective tables is easier. This simplifies data management tasks and makes it easier to scale and adapt the database structure over time.
5. **Supports Data Relationships:** Relational databases manage complex relationships between different data types. Multiple tables can effectively represent these relationships, such as one-to-many or many-to-many.

In summary, using multiple tables in a relational database ensures data organisation, integrity, and efficient retrieval, making it a fundamental practice in database design.

Principles of database normalisation

Database normalisation is a set of principles that help structure databases efficiently by **reducing redundancy and maintaining data integrity**. Here are the main principles in simple terms:

- **First Normal Form (1NF):** Each column in a table should contain no repeating groups or arrays. **This ensures that each piece of data is unique and doesn't contain multiple values.**
- **Second Normal Form (2NF):** Building on 1NF, a table in 2NF should have a **primary key**, and **every non-key column should be fully functionally dependent on that key.**
- **Third Normal Form (3NF):** In 3NF, a table should be in 2NF, and **no non-key column should depend on another non-key column.** This prevents transitive dependencies, ensuring that data is organised without unnecessary relationships.
- **Boyce-Codd Normal Form (BCNF):** BCNF goes a step further by stating that, besides being in 3NF, **every non-key attribute should be functionally dependent solely on the primary key. It aims to eliminate any anomalies related to key attributes.**
- **Fourth and Fifth Normal Forms (4NF and 5NF):** These are more advanced normalisation forms and are rarely needed in most database designs. They address multi-valued and join dependencies, respectively, but applying them can be complex and may not always be necessary.

The goal of normalisation is to reduce data redundancy and dependency, making databases more efficient, easier to maintain, and less prone to data anomalies.

First Normal Form (1NF)

Suppose we have a table for storing information about students and their enrolled courses. Each student can be enrolled in multiple courses, and each course can have multiple students.

A non-1NF table might look like this:

EMPLOYEE E_ID	NAME	JOB_CODE	JOB	STATE_CODE	HOME_STATE
E001	Alice	J01	Chef, Waiter	26	Michigan
E002	Bob	J02	Waiter, Bartender	56	Wyoming
E003	Alice	J01	Chef	56	Wyoming

- A single cell does not hold more than one value (atomicity)
- There is a primary key for identification
- No duplicated rows or columns
- Each column has only one value for each row in the table

To bring it to 1NF, we split the Courses column into multiple rows:

EMPLOYEE E_ID	NAME	JOB_CODE	JOB	STATE_CODE	HOME_STATE
E001	Alice	J01	Chef	26	Michigan
E001	Alice	J02	Waiter	26	Michigan
E002	Bob	J02	Waiter	56	Wyoming
E002	Bob	J03	Bartender	56	Wyoming
E003	Alice	J01	Chef	56	Wyoming

Second Normal Form (2NF):

The 1NF only eliminates repeating groups, not redundancy.

That's why there is 2NF.

A table is said to be in 2NF if it meets the following criteria:

- It's already in 1NF
- Has no partial dependency.
That is, all non-key attributes are fully dependent on a primary key.

EMPLOYEES TABLE			
EMPLOYEE_ID	NAME	STATE_CODE	HOME_STATE
E001	Alice	26	Michigan
E002	Bob	56	Wyoming
E003	Alice	56	Wyoming

EMPLOYEE_ROLES TABLE		
ROLE_ID	EMPLOYEE_ID	JOB_CODE
1	E001	J01
2	E001	J02
3	E002	J02
4	E002	J03
5	E003	J01

JOBS TABLE	
JOB_CODE	JOB
J01	Chef
J02	Waiter
J03	Bartender

Third Normal Form (3NF)

When a table is in 2NF, it eliminates repeating groups and redundancy, but it does not eliminate transitive partial dependency.

This means a non-prime attribute (an attribute that is not part of the candidate's key) is dependent on another non-prime attribute. This is what the third normal form (3NF) eliminates.

So, for a table to be in 3NF, it:

- is in 2NF
- has no transitive partial dependency

https://www.youtube.com/watch?v=J-drts33N8g&ab_channel=LearnLearnScratchTutorials

STATE_CODE	HOME_STATE
26	Michigan
56	Wyoming

EMPLOYEES TABLE		
EMPLOYEE_ID	NAME	STATE_CODE
E001	Alice	26
E002	Bob	56
E003	Alice	56

JOBS TABLE	
JOB_CODE	JOB
J01	Chef
J02	Waiter
J03	Bartender

EMPLOYEE_ROLES TABLE		
ROLE_ID	EMPLOYEE_ID	JOB_CODE
1	E001	J01
2	E001	J02
3	E002	J02
4	E002	J03
5	E003	J01

Primary and Foreign Key in SQL

A Primary key

A Primary key is a unique column we set in a table to identify and locate data in queries easily. A table can have only one primary key.

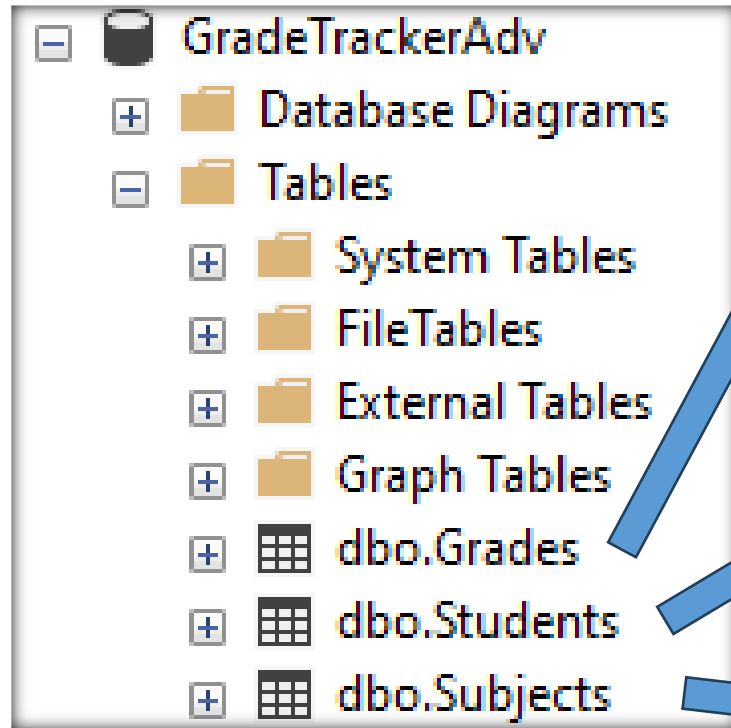
- The primary key column has a unique value and doesn't store repeating values. A Primary key can never take NULL values.
- For example, in the case of a student, when identification needs to be done in the class, the roll number of the student plays the role of Primary key.
- Similarly, when we talk about employees in a company, the employee ID functions as the Primary key for identification.

A Foreign key

A Foreign key is beneficial when we connect two or more tables so that data from both can be used parallelly.

- A foreign key is a field or collection of fields in a table that refers to the **Primary key of the other table**. It is responsible for managing the relationship between the tables.
- The table that contains the foreign key is often called the child table, and the table whose primary key is being referred to by the foreign key is called the Parent Table.
- For example, When we talk about students and the courses they have enrolled in, if we try to store all the data in a single table, the problem of redundancy arises.
- To solve this table, we make two tables, one the student detail table and the other the department table. In the student table, we store the details of students and the courses they have enrolled in.
- And in the department table, we store all the department details. Here, the course acts as the Primary key for the department table, whereas it acts as the Foreign key in the student table.

Grades Tracker database in 3NF



	Column Name	Data Type	Allow Nulls
PK	GradeId	int	<input type="checkbox"/>
	StudentId	int	<input type="checkbox"/>
	SubjectId	int	<input type="checkbox"/>
	Percentage	int	<input type="checkbox"/>
			<input type="checkbox"/>

	Column Name	Data Type	Allow Nulls
PK	StudentID	int	<input type="checkbox"/>
	FirstName	varchar(50)	<input type="checkbox"/>
	LastName	varchar(50)	<input type="checkbox"/>
	Email	varchar(50)	<input type="checkbox"/>
			<input type="checkbox"/>

	Column Name	Data Type	Allow Nulls
PK	SubjectId	int	<input type="checkbox"/>
	SubjectName	varchar(50)	<input type="checkbox"/>
			<input type="checkbox"/>

SQL INNER JOIN

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name =
   table2.column_name;
```

https://www.w3schools.com/sql/sql_ref_inner_join.asp

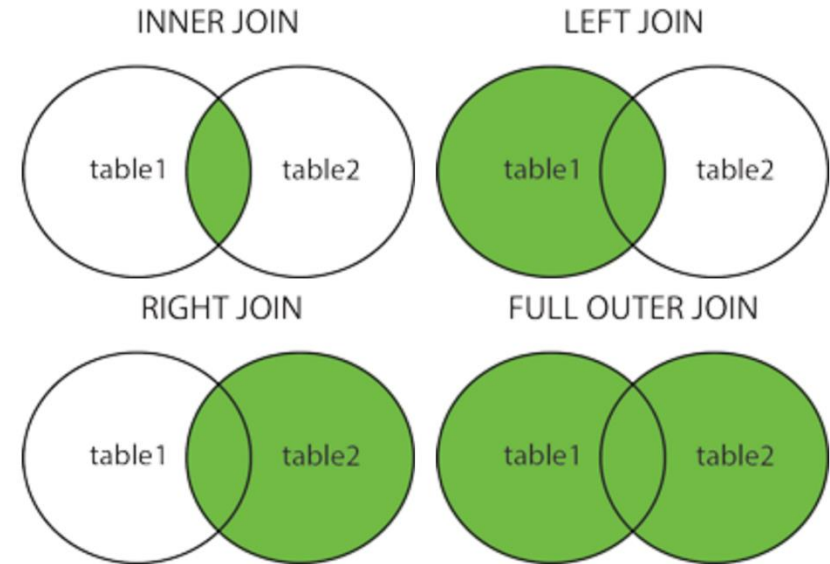


TABLE 1

Products	Price
Kiwis	\$6
Onions	\$3
Tomatoes	\$7

TABLE 2

Products	Quantity
Kiwis	10
Onions	6
Broccoli	5

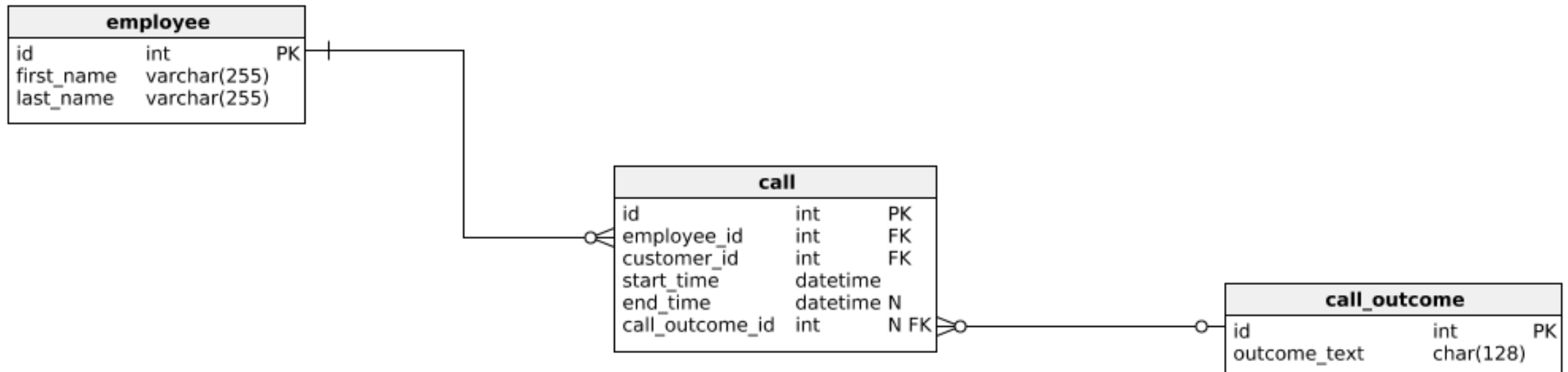


Products	Price	Quantity
Kiwis	\$6	10
Onions	\$3	6

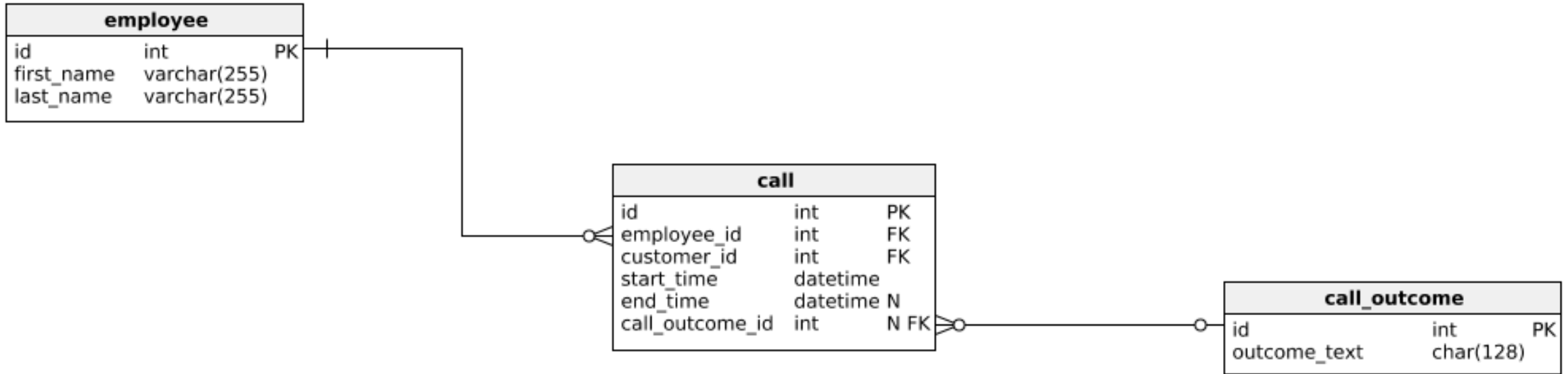
SQL INNER JOIN (Call centre)

We need to list all calls with their start time and end time. For each call, we want to display the outcome as well as the first and last name of the employee who made that call. We'll sort our calls by start time ascending.

Before we write the query, we'll identify the tables we need to use. To do that, we need to determine which tables contain the data we need and include them. Also, we should include all tables along the way between these tables – tables that don't contain data needed but serve as a relation between tables that do (that is not the case here).



SQL INNER JOIN



```
SELECT employee.first_name, employee.last_name, call.start_time, call.end_time, call_outcome.outcome_text
FROM employee
INNER JOIN call ON call.employee_id = employee.id
INNER JOIN call_outcome ON call.call_outcome_id = call_outcome.id
ORDER BY call.start_time ASC;
```

SQL INNER JOIN

employee		
id	int	PK
first_name	varchar(255)	
last_name	varchar(255)	

```
SELECT employee.first_name, employee.last_name, call.start_time, call.end_time, call_outcome.outcome_text
FROM employee
INNER JOIN call ON call.employee_id = employee.id
INNER JOIN call_outcome ON call.call_outcome_id = call_outcome.id;
```

call			
id	int	PK	
employee_id	int	FK	
customer_id	int	FK	
start_time	datetime		
end_time	datetime	N	
call_outcome_id	int	N FK	

call_outcome		
id	int	PK
outcome_text	char(128)	

first_name	last_name	start_time	end_time	outcome_text
------------	-----------	------------	----------	--------------

Dapper

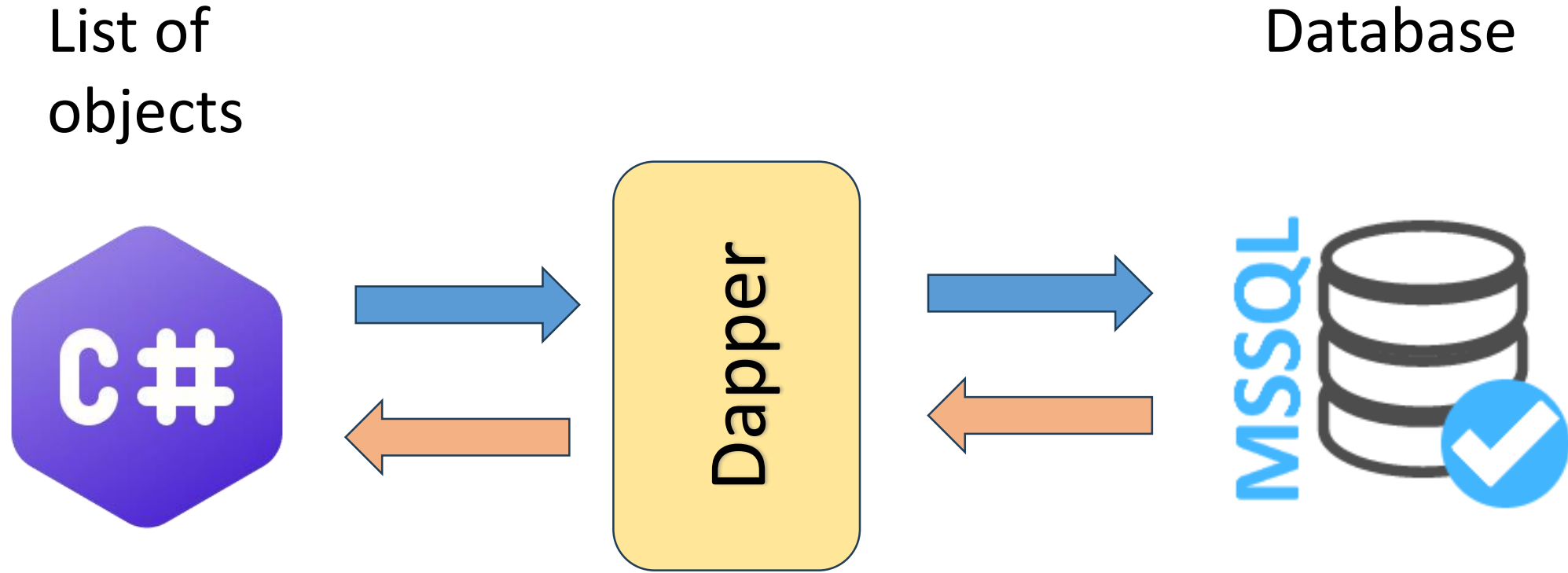
SQL INNER JOIN

```
SELECT employee.first_name, employee.last_name, call.start_time, call.end_time, call_outcome.outcome_text
FROM employee
INNER JOIN call ON call.employee_id = employee.id
INNER JOIN call_outcome ON call.call_outcome_id = call_outcome.id
ORDER BY call.start_time ASC;
```

Results Messages

	first_name	last_name	start_time	end_time	outcome_text
1	Thomas (Neo)	Anderson	2020-01-11 09:00:15.000	2020-01-11 09:12:22.000	finished - successfully
2	Agent	Smith	2020-01-11 09:02:20.000	2020-01-11 09:18:05.000	finished - unsuccessfully
3	Thomas (Neo)	Anderson	2020-01-11 09:14:50.000	2020-01-11 09:20:01.000	finished - successfully
4	Thomas (Neo)	Anderson	2020-01-11 09:24:15.000	2020-01-11 09:25:05.000	finished - unsuccessfully
5	Thomas (Neo)	Anderson	2020-01-11 09:26:23.000	2020-01-11 09:33:45.000	finished - successfully
6	Thomas (Neo)	Anderson	2020-01-11 09:40:31.000	2020-01-11 09:42:32.000	finished - successfully
7	Agent	Smith	2020-01-11 09:41:17.000	2020-01-11 09:45:21.000	finished - successfully
8	Thomas (Neo)	Anderson	2020-01-11 09:42:32.000	2020-01-11 09:46:53.000	finished - unsuccessfully
9	Agent	Smith	2020-01-11 09:46:00.000	2020-01-11 09:48:02.000	finished - successfully
10	Agent	Smith	2020-01-11 09:50:12.000	2020-01-11 09:55:35.000	finished - successfully

View Models & Multi-Table Requests



To Recap. Expenses Tracker

The screenshot shows a desktop application window titled "MainWindow". On the left is a blue sidebar with three buttons: "USERS", "EXPENSES", and "CATEGORIES". The main area contains a table with 6 columns: Id, Date, UserName, Price, CategoryName, and an empty column. The table has 5 rows of data. To the right of the table is a form with fields for ID, USER (a dropdown), EXPENSE (a dropdown), PRICE \$ (a text input), and DATE (a date picker showing 15). Below these fields are "SAVE" and "CLEAR FORM" buttons. At the bottom left of the main area is a red "DELETE RECORD" button.

Id	Date	UserName	Price	CategoryName	
1	4/24/2023 12:00:00 AM	Troy Vaughn	500.00	Invoices	
5	4/21/2023 12:00:00 AM	Zack Wilson	29.99	Petty Cash	
2	3/16/2023 9:30:00 AM	Sally Smith	200.00	Travel	
3	2/4/2023 12:00:00 AM	Sally Smith	150.00	Stock Purchases	
4	2/4/2023 12:00:00 AM	Sally Smith	25.00	Payroll	

USERS

EXPENSES

CATEGORIES

ID:

USER:

EXPENSE:

PRICE \$:

DATE:

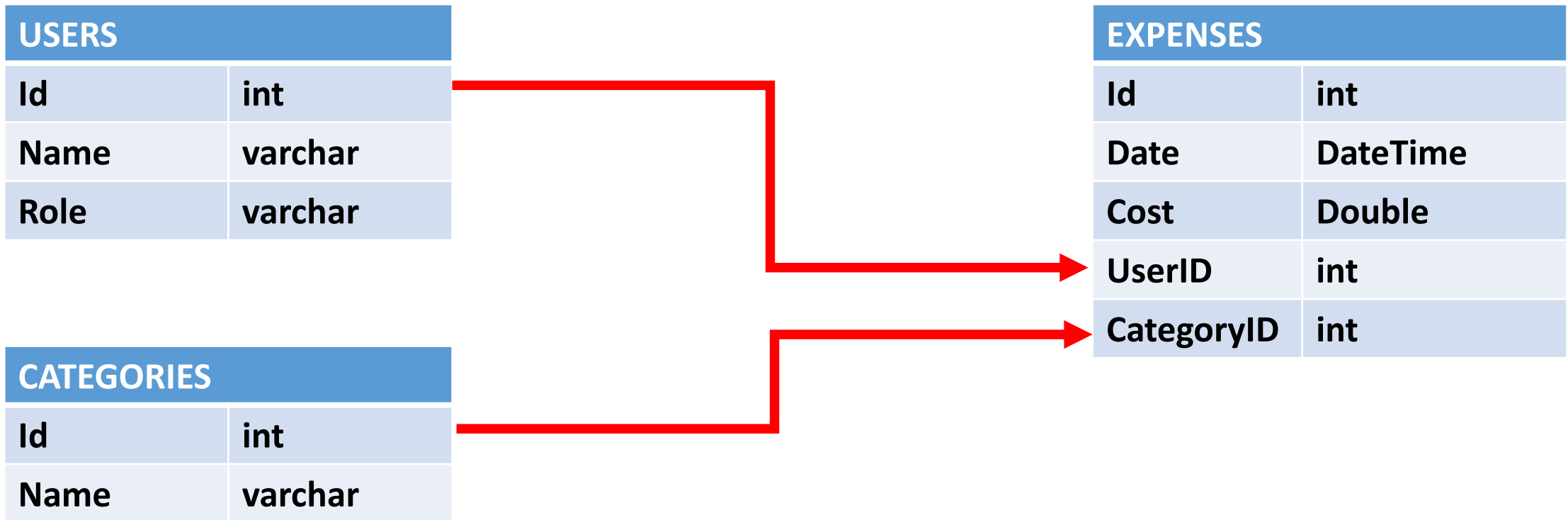
Select a date 15

SAVE

CLEAR FORM

DELETE RECORD

Expense tracker. Data structure. 3N-Form



How to retrieve the result of the query using Dapper if the names of the columns are duplicated!? Use ALIAS!

- SQL aliases are used to give a table, or a column in a table, a temporary name.
- Aliases are often used to make column names more readable.
- An alias only exists for the duration of that query.
- An alias is created with the **AS** keyword.

```
SELECT Expenses.Id, Expenses.Date, Expenses.Price, Users.Name AS UserName, Categories.Name  
AS CategoryName FROM Categories INNER JOIN Expenses ON Categories.Id = Expenses.CategoryId  
INNER JOIN Users ON Expenses.UserId = Users.Id ORDER BY Expenses.Date DESC;
```

https://www.w3schools.com/sql/sql_alias.asp

ResultView

```
SELECT Expenses.Id, Expenses.Date, Expenses.Price, Users.Name AS UserName, Categories.Name AS  
CategoryName FROM Categories INNER JOIN Expenses ON Categories.Id = Expenses.CategoryId  
INNER JOIN Users ON Expenses.UserId = Users.Id ORDER BY Expenses.Date DESC;
```

The diagram illustrates the data flow from a SQL query to a C# class. A blue bracket connects the SQL query to a table with five columns: Id, Date, Price, UserName, and CategoryName. A blue arrow points from this table to a yellow box labeled 'Dapper'. Another blue arrow points from 'Dapper' to the C# code for the ExpenseView class.

Id	Date	Price	UserName	CategoryName
----	------	-------	----------	--------------

Dapper

```
public class ExpenseView  
{  
    public int Id { get; set; }  
    public DateTime Date { get; set; }  
    public string UserName { get; set; }  
    public decimal Price { get; set; }  
    public string CategoryName { get; set; }  
}
```



Scenario

- Mark intends to run the system you develop on his personal laptop so he can enter data directly at events, he is also familiar with using a **relational database system (SQL Server)** as he has used it in the past and is capable of backing it up himself if needed.
- Therefore, he has asked that the system meet the below requirements:
 - **Must be a Windows Desktop Application.**
 - **Must use a relational database system — SQL Server.**
 - **Must provide functionality to Create, View, Update and Delete: Team Details, Events, Games Played and Team Results.**

Mark requires the system to be able to store the following data:

Team Details

- Team Name
- Primary Contact
- Contact Phone
- Contact Email
- Competition Points

Events

- Event Name
- Event Location
- Event Date

Games Played

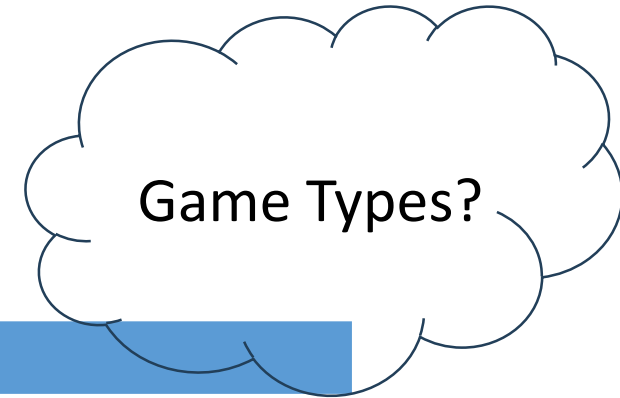
- Game Name
- Game Type (Team / Solo)

Team Results

- Event Held
- Game Played
- Team
- Opposing Team
- Result (Win / Loss / Draw)

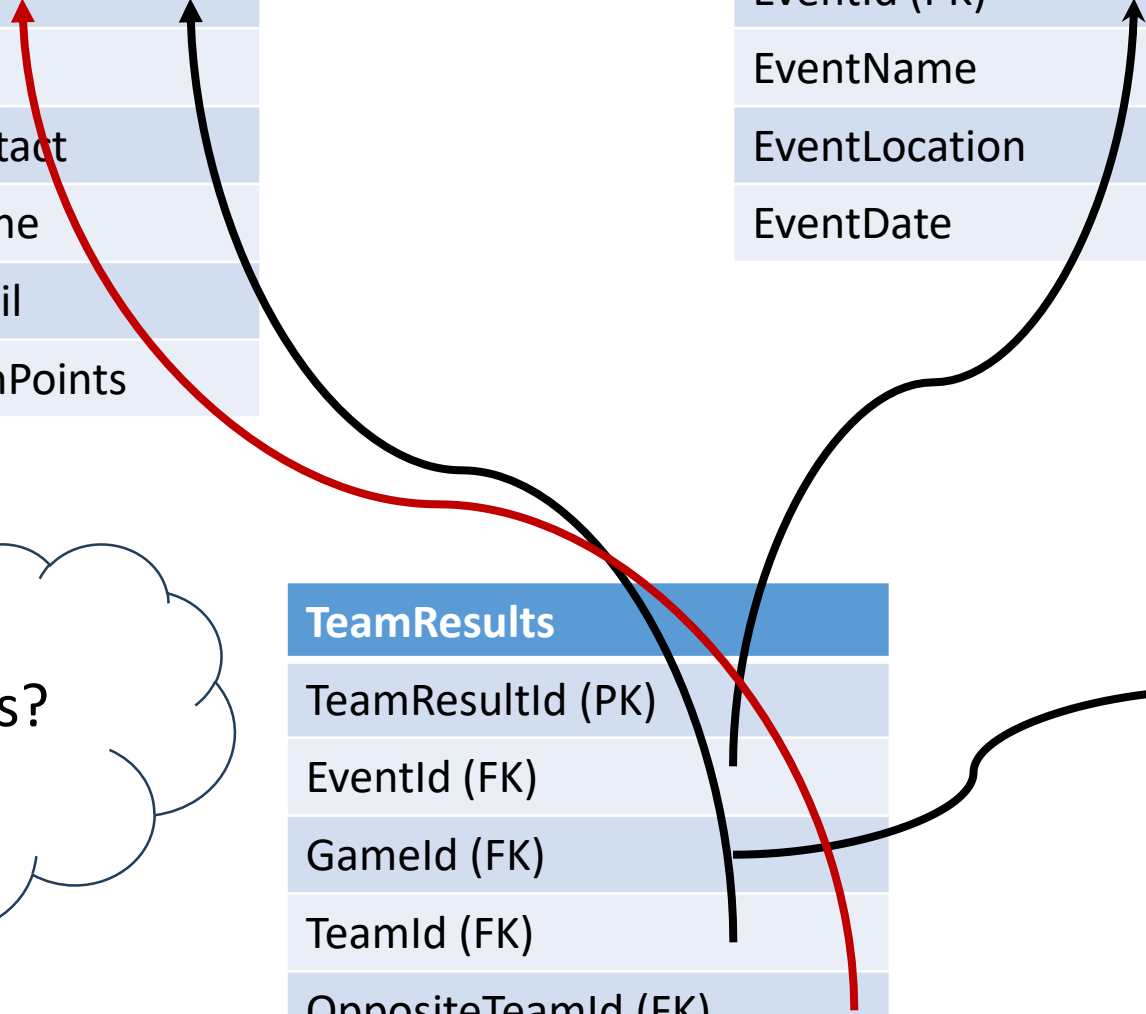
Teams
TeamId (PK)
TeamName
PrimaryContact
ContactPhone
ContactEmail
CompetitionPoints

Events
EventId (PK)
EventName
EventLocation
EventDate



TeamResults
TeamResultId (PK)
EventId (FK)
GameId (FK)
TeamId (FK)
OppositeTeamId (FK)
Result

GamesPlayed
GameId (PK)
GameName
GameType



Mark has also requested that the application be able to **generate reports** based upon the following options:

- Team Details — Ordered by Competition points
- Team Results — Ordered by Event
- Team Results — Ordered by Team

Each report **type needs to be filterable or searchable by Team Name and the results need to be shown in the application UI.**

Each report needs to also be exportable as a .csv file and be populated based on the current filters and search parameters.

The application also needs to be able to perform the following functionality for Mark and his team.

- When a team has a result entered, the outcome of the result (Win/Loss/Draw) needs to update the team's competition points by 2 points for a win and 1 point for a draw.
- When entering team results, the user should be able to enter both teams who competed and set the results for each at the same time.

A form for entering team results. It features two input fields for team names, a central 'VS' label, and checkboxes for 'WINNER' and 'DRAW' for each team. A 'SAVE' button is located at the bottom right.

TEAM 1 NAME ☐ WINNER

VS ☐ DRAW

TEAM 2 NAME ☐ WINNER

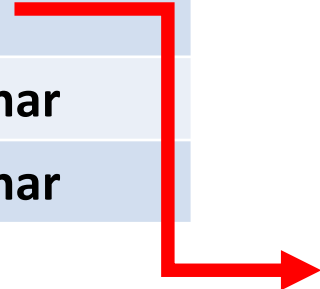
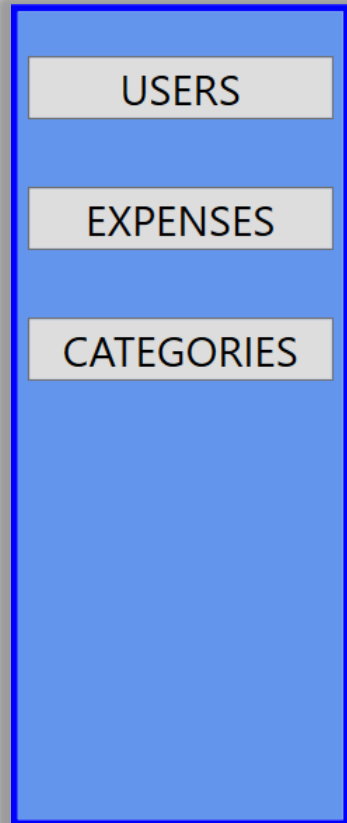
SAVE

Expenses Tracker

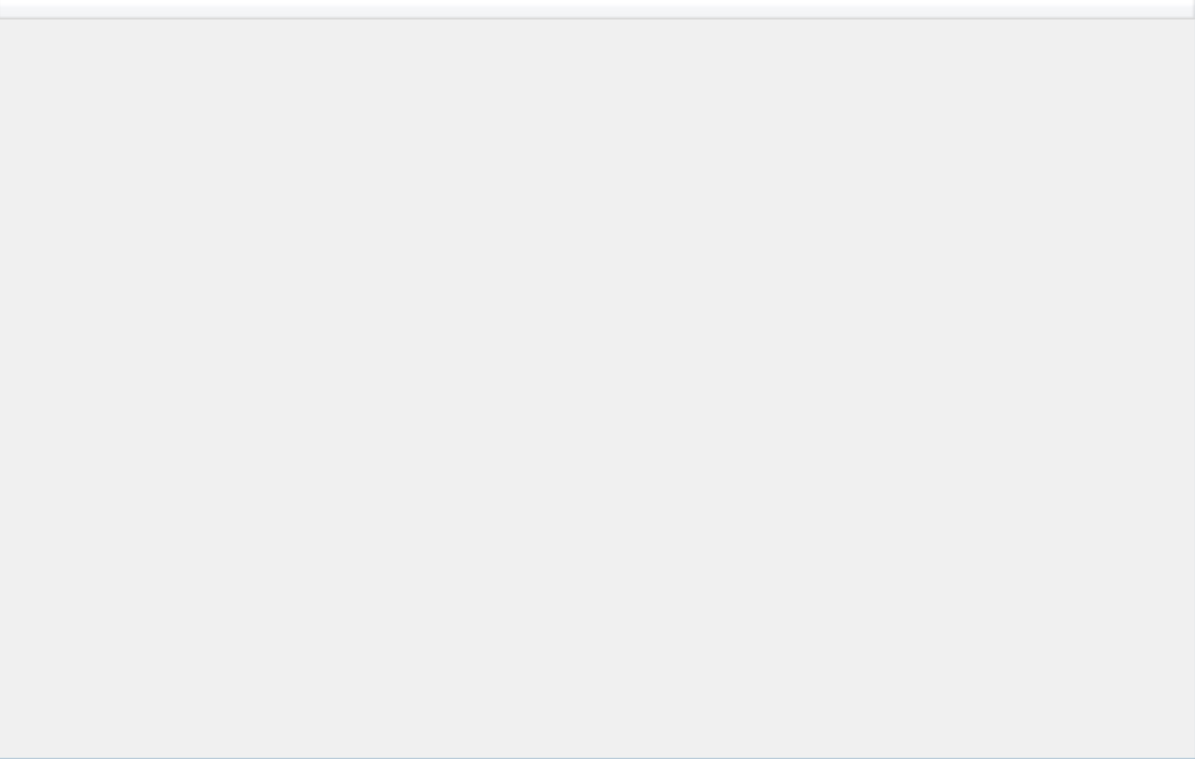
USERS	
Id	int
Name	varchar
Role	varchar

EXPENSES	
Id	int
Date	DateTime
Cost	Double
UserID	int
CategoryID	int

CATEGORIES	
Id	int
Name	varchar



User panel

	ID:
	<input type="text"/>
	NAME:
	<input type="text"/>
	ROLE:
	<input type="text"/>
	<input type="button" value="SAVE"/>
	<input type="button" value="CLEAR FORM"/>
<input type="button" value="DELETE RECORD"/>	

Expense panel

	ID:	<input type="text"/>
	USER:	<input type="text"/>
	EXPENSE:	<input type="text"/>
	PRICE \$:	<input type="text"/>
	DATE:	<input type="text" value="Select a date"/>
	<div>SAVE</div> <div>CLEAR FORM</div>	
<div>DELETE RECORD</div>		

One-Window GUI VS Multi-Windows GUI

- <https://www.youtube.com/watch?v=FACL4eTZ8uA> (at home)
- <https://www.youtube.com/watch?v=YBbeZp17Tho>

```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition Width="3*"/>
  </Grid.ColumnDefinitions>

  <Border Background="CornflowerBlue" BorderBrush="Blue" BorderThickness="4"/>
  <StackPanel Margin="10">
    <Button x:Name="btnUsers" Margin="0,20" FontSize="25" Click="btnUsers_Click">USERS</Button>
    <Button x:Name="btnExpenses" Margin="0,20" FontSize="25" Click="btnExpenses_Click">EXPENSES</Button>
    <Button x:Name="btnCategories" Margin="0,20" FontSize="25">CATEGORIES</Button>
  </StackPanel>

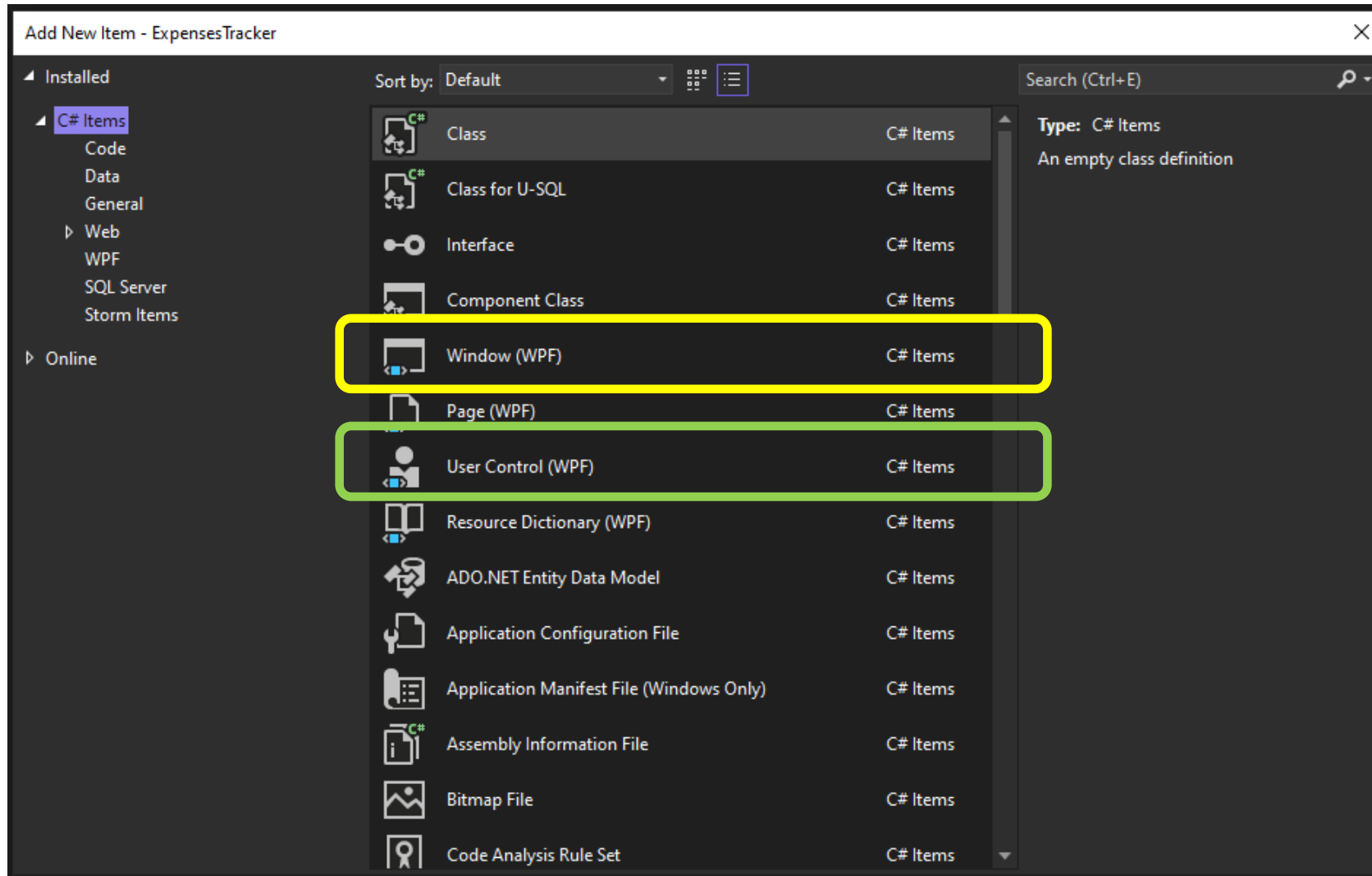
  <ContentControl x:Name="conMain" Grid.Column="1"/>
</Grid>
```

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }

    private void btnNormal_Click(object sender, RoutedEventArgs e)
    {
        NormalWindow normalWindow = new NormalWindow();
        normalWindow.Show();
    }

    private void btnModal_Click(object sender, RoutedEventArgs e)
    {
        ModalWindow modalWindow = new ModalWindow();
        modalWindow.ShowDialog();
    }
}
```

User Control (WPF) vs Window (WPF)



User Control (WPF)

MainWindow

USERS

EXPENSES

CATEGORIES

Id	Date	UserName	Price	CategoryName
1	4/24/2023 12:00:00 AM	Troy Vaughn	500.00	Invoices
5	4/21/2023 12:00:00 AM	Zack Wilson	29.99	Petty Cash
2	3/16/2023 9:30:00 AM	Sally Smith	200.00	Travel
3	2/4/2023 12:00:00 AM	Sally Smith	150.00	Stock Purchases
4	2/4/2023 12:00:00 AM	Sally Smith	25.00	Payroll

ID:

USER:

EXPENSE:

PRICE \$:

DATE:

SAVE

CLEAR FORM

DELETE RECORD

ID:

USER:

EXPENSE:

PRICE \$:

DATE:

SAVE

CLEAR FORM

DELETE RECORD

ID:

NAME:

ROLE:

SAVE

CLEAR FORM

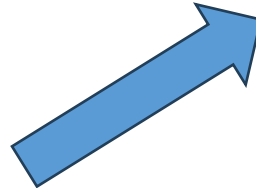
DELETE RECORD

Window (WPF)

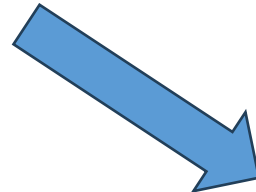
The screenshot shows a WPF application window titled "MainWindow". On the left is a blue sidebar with three buttons: "USERS", "EXPENSES", and "CATEGORIES". The main area contains a table with the following data:

Id	Date	UserName	Price	CategoryName
1	4/24/2023 12:00:00 AM	Troy Vaughn	500.00	Invoices
5	4/21/2023 12:00:00 AM	Zack Wilson	29.99	Petty Cash
2	3/16/2023 9:30:00 AM	Sally Smith	200.00	Travel
3	2/4/2023 12:00:00 AM	Sally Smith	150.00	Stock Purchases
4	2/4/2023 12:00:00 AM	Sally Smith	25.00	Payroll

Below the table is a large grey rectangular area. To the right of the table is a form with fields for ID, USER, EXPENSE, PRICE \$, and DATE, along with "SAVE" and "CLEAR FORM" buttons. At the bottom right of the sidebar area is a red "DELETE RECORD" button.



This view shows a detail record. It features a large grey rectangular area on the left. On the right is a form with fields for ID, USER, EXPENSE, PRICE \$, and DATE, along with "SAVE" and "CLEAR FORM" buttons. At the bottom left is a red "DELETE RECORD" button.



This view shows a detail record for a user. It features a large grey rectangular area on the left. On the right is a form with fields for ID, NAME, and ROLE, along with "SAVE" and "CLEAR FORM" buttons. At the bottom left is a red "DELETE RECORD" button.

Application Window Customisation and Reusable Style Resources

- <https://www.youtube.com/watch?v=Aon4tiUYlas>
- <https://www.youtube.com/watch?v=F7DiRnH6Yqo>

How to build an application database, tables app.config

```
<configuration>  
  <connectionStrings>  
    <add name="Default"  
          providerName="System.Data.SqlClient"  
          connectionString="Server=.\SQLEXPRESS; Integrated Security=True; Database=ETDB"/>  
  </connectionStrings>  
</configuration>
```

App.xaml and MainWindow.xaml

App.xaml and MainWindow.xaml are two crucial files in a WPF application developed using Visual Studio.

- MainWindow.xaml defines the user interface for the application's main window.
- The App.xaml file allows developers to set application-wide properties and handle application-wide events.
 - App.xaml is the declarative starting point of your application.
 - Visual Studio will automatically create it for you when you start a new WPF application, including a Code-behind file called App.xaml.cs.
 - Those two files are partial classes, working together to allow you to work in both markup (XAML) and Code-behind.

<https://wpf-tutorial.com/wpf-application/working-with-app-xaml/>

How to build an application database, tables

App.xaml.cs

```
1  using DataManagement;
2  using System;
3  using System.Collections.Generic;
4  using System.Configuration;
5  using System.Data;
6  using System.Linq;
7  using System.Security.Principal;
8  using System.Threading.Tasks;
9  using System.Windows;
10
11 namespace ExpensesTracker
12 {
13     /// <summary>
14     /// Interaction logic for App.xaml
15     /// </summary>
16     public partial class App : Application
17     {
18         public App()
19         {
20             //Create our Database building class.
21             DbBuilder builder = new DbBuilder();
22             //Tell it ti build the database. This will only do so if it doesn't already exist.
23             builder.CreateDatabase();
24             //Check if the database contains tables. If not, run the table building code.
25             if (builder.DoTablesExist() == false)
26             {
27                 //Runs the code to build and seed the database tables.
28                 builder.BuildDatabaseTables();
29             }
30         }
31     }
32 }
33
```

Search Solution Explorer (Ctrl+;)

Solution 'ExpensesTracker' (2 of 2 projects)

- ExpensesTracker
 - Dependencies
 - Models
 - Category.cs
 - Expense.cs
 - ExpenseView.cs
 - User.cs
 - DataAdapter.cs
 - DataAdapter
 - DbBuilder.cs
 - DbBuilder
 - Helper.cs
- ExpensesTracker
 - Dependencies
 - App.config
 - App.xaml
 - App.xaml.cs
 - AssemblyInfo.cs
 - ExpensePanel.xaml
 - ExpensePanel.xaml.cs
 - MainWindow.xaml
 - MainWindow.xaml.cs
 - UsersPanel.xaml
 - UsersPanel.xaml.cs

How to build an application database, tables

DbBuilder. Create database

```
public void CreateDatabase()
{
    //Our connection object to link to the database
    SqlConnection connection = Helper.CreateSQLServerConnection("Default");
    try
    {
        //Custom connection string to only connect to the server layer of your SQL Database
        string connectionString = $"Data Source={connection.DataSource}; Integrated Security = True";
        //Query to build new Database if it does not already exist.
        string query = $"IF NOT EXISTS (SELECT 1 FROM sys.databases WHERE name = '{connection.Database}')" +
            $" CREATE DATABASE {connection.Database}";
        using (connection = new SqlConnection(connectionString))
        {
            //A command object which will send our request to the Database <= Normally done for us by Dapper
            using (SqlCommand command = new SqlCommand(query, connection))
            {
                //Checks if the connection is currently open, if not, it opens the connection.<= Normally done for us by Dapper
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }
                //Executes an SQL Request that does not expect a response(Query) to be returned.
                command.ExecuteNonQuery();
                //Closes the connection to the database manually.<= Normally done for us by Dapper
                connection.Close();
            }
        }
    }
    catch (Exception e)
    {
    }
}
```

How to build an application database, tables

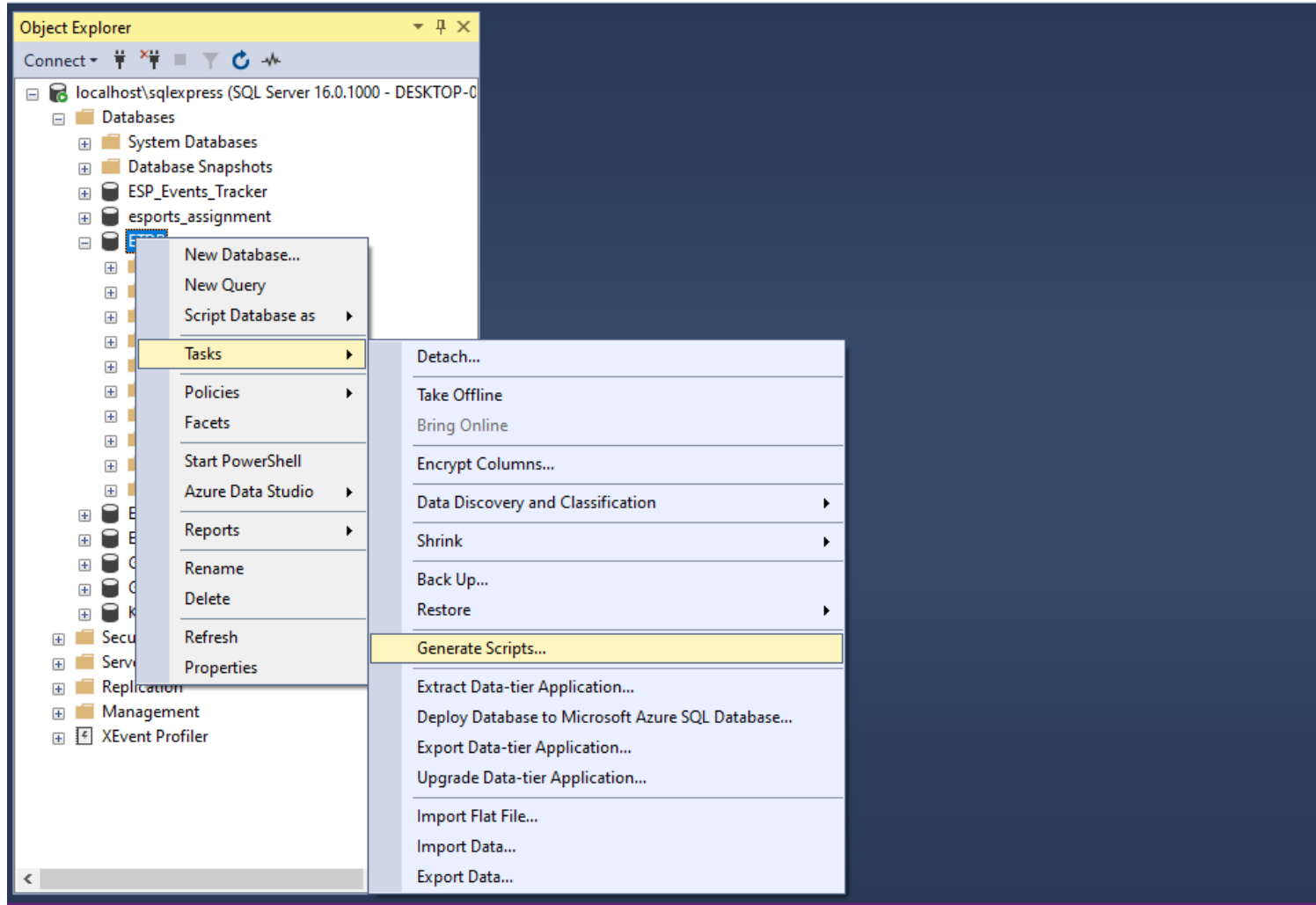
DbBuilder. Create tables

```
public bool DoTablesExist()
{
    //Our using statemtnqwhich builds our connection and disposes of it once finished.
    using (var connection = Helper.CreateSqlServerConnection("Default"))
    {
        //Quey to request the count of how many base tables are in the database structure. Base tables refers to user
        //built tables and ignores inbuild tables such as index tables and reference/settings tables.
        string query = $"SELECT COUNT(*) FROM {connection.Database}.INFORMATION_SCHEMA.TABLES " +
            $"WHERE TABLE_TYPE = 'BASE TABLE'";
        //Sends the query to the databse and stores the returned table count.
        int count = connection.QuerySingle<int>(query);
        //If the count is above 0 return true, otherwise return false to indicate whether the databse has tabes or not.
        if (count > 0)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

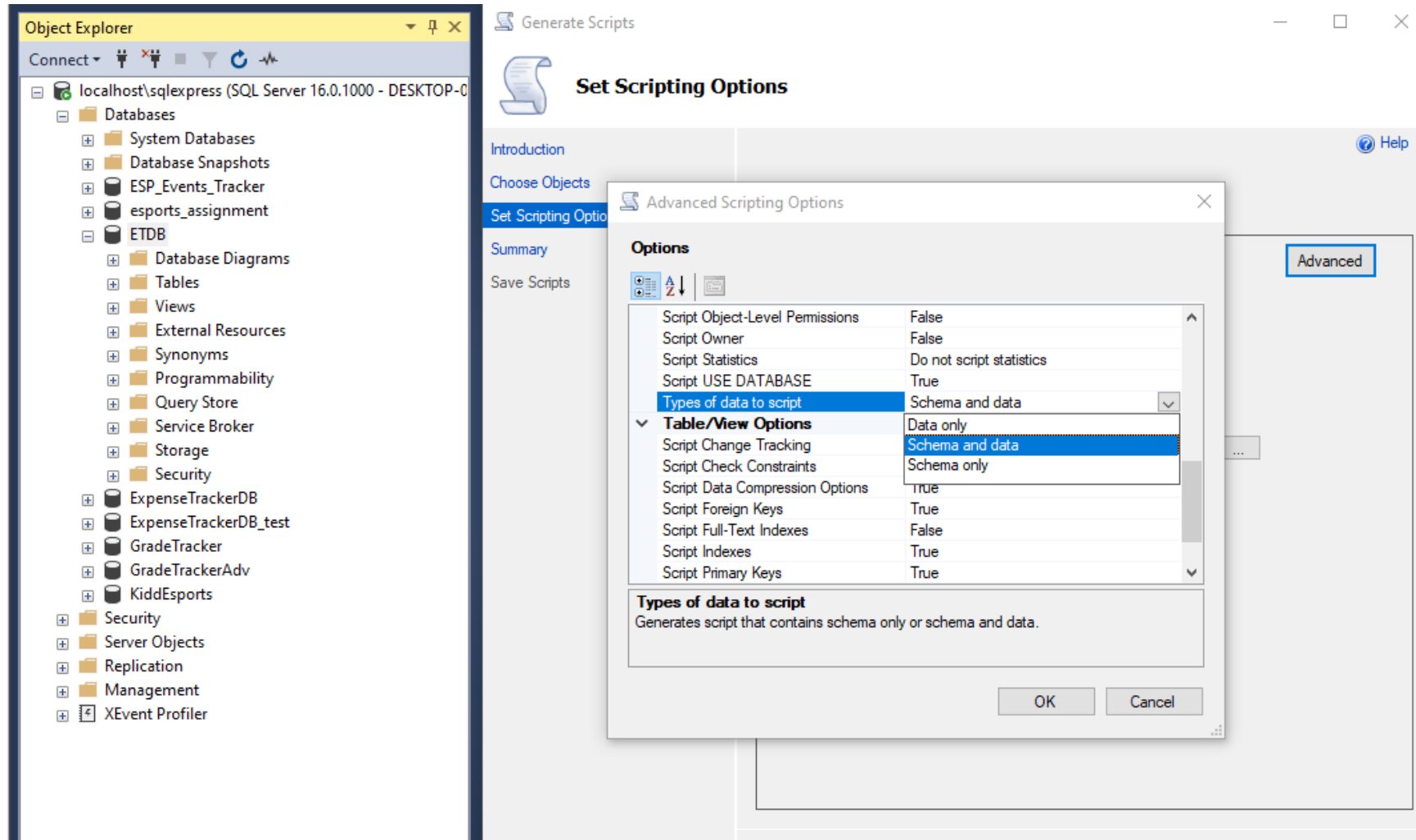
How to check the existence of one table.

```
public static bool TableExists(string tableName)
{
    //Our using statement which builds our connection and disposes of it once finished.
    using (var connection = Helper.CreateSQLServerConnection("Default"))
    {
        if (connection != null)
        {
            //Query to request the count of how many base tables are in the database structure. Base tables refers to user
            //built tables and ignores inbuilt tables such as index tables and reference/settings tables.
            //Query - method extension provided by Dapper library
            string query = $"SELECT COUNT(*) as Count FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME = '{tableName}'";
            var count = connection.Query<int>(query).FirstOrDefault();
            //If the count is above 0 return true, otherwise return false to indicate whether the database has tables or not.
            return (count > 0);
        }
    }
    return false;
}
```

SQL management studio. How to generate script. Step 1



SQL management studio. How to generate script. Step 2



More elements. Date Picker

- DatePicker - <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/controls/datepicker?view=netframeworkdesktop-4.8>
- https://www.youtube.com/watch?v=NOop2rC4fN8&t=3s&ab_channel=tips%27ntricks

More elements. Combo-Box

- ComboBox - <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/controls/combobox?view=netframeworkdesktop-4.8&viewFallbackFrom=netdesktop-6.0>
- https://www.youtube.com/watch?v=4rU9yAkgGdk&ab_channel=CodingUnderPressure