A logo for a company

AI-generated content may be incorrect.

Hederlige Harrys Bilar

James William Martinsson Co

Welcome to the documentation for Hederlige Harrys Bilar database. In this document, you will find the different functionalities and descriptions of why this database was created and how you can use it to its fullest capabilities. The script is written using SQL Server Management Studio (SSMS).

This is the final assignment for my course in Nackademin called SQL 2 – Advanced (Fördjupning). Some functions in this document would normally be handled by external programs. However, for this project, we decided to be creative and implement them solely in SSMS.

Table of Contents

[**Files** 3](#_Toc191054307)

[**Database Model (ER-Diagram)** 3](#_Toc191054308)

[**Tables** 4](#_Toc191054309)

[**Email settings and activation** 6](#_Toc191054310)

[**Views** 6](#_Toc191054311)

[**Stored Procedures** 7](#_Toc191054312)

[**Indexes** 8](#_Toc191054313)

[**Long-Term Improvements** 9](#_Toc191054314)

[**HHB – SQLQueries** 11](#_Toc191054315)

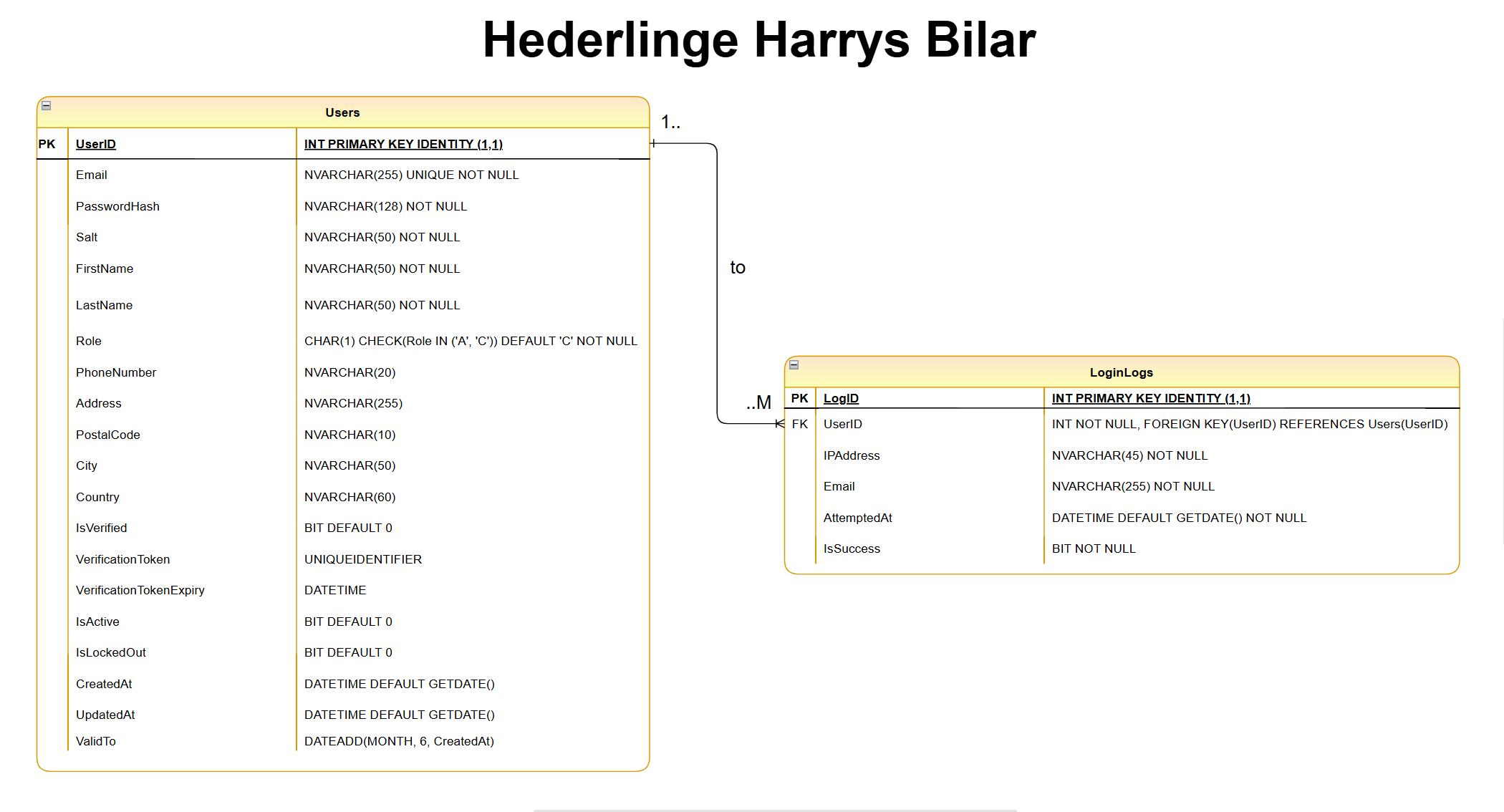
# *Note: Shortcut! You can click on one of the titles in the Table of Contents if you want to jump to those sections.*

# **Files**

These are the files currently being used for this project:

* HHB\_Documentation – Documentation for the Hederlige Harrys Bilar project (This document).
* HHB\_ERDiagram - Database Model for this project (Available in two formats: DrawIO and PNG).
* HHB\_SQLScript - The main code, just execute this file into your SSMS and you’re good to go.
* HHB\_SQLQueries - This is where you execute the stored procedures, views and other things you need to check that the actual Script works.

# **Database Model (ER-Diagram)**



# **Tables**

**Schema** Users **Primary Key:** 🔑

**Tables** Users **Foreign Key: 🔑➡**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Key** | **Name** | **Data Type** | **Null** | **Attributes** | **References** | **Description** |
| 1. | 🔑 | UserID | INT | No | PRIMARY KEY, IDENTITY(1,1) | - | Unique identifier for users |
| 2. |  | Email | NVARCHAR(255) | No | UNIQUE | - | User’s email (must be unique) |
| 3. |  | PasswordHash | NVARCHAR(128) | No |  | - | Hashed password (SHA2\_512) |
| 4. |  | Salt | NVARCHAR(50) | No |  | - | Salt for password hashing |
| 5. |  | FirstName | NVARCHAR(50) | No |  | - | User’s first name |
| 6. |  | LastName | NVARCHAR(50) | No |  | - | User’s last name |
| 7. |  | Role | CHAR(1) | No | CHECK(Role IN ('A', 'C')) DEFAULT 'C' | - | Users role (A = Admin or C = Customer), Default C |
| 8. |  | PhoneNumber | NVARCHAR(20) | Yes |  | - | User’s phone number |
| 9. |  | Address | NVARCHAR(255) | Yes |  | - | User’s address |
| 10. |  | PostalCode | NVARCHAR(10) | Yes |  | - | Postal code |
| 11. |  | City | NVARCHAR(50) | Yes |  | - | City name |
| 12. |  | Country | NVARCHAR(60) | Yes |  | - | Country name |
| 13. |  | IsVerified | BIT | No | DEFAULT 0 | - | If user is verified = 1,  otherwise 0 |
| 14. |  | VerificationToken | UNIQUEIDENTIFIER | Yes |  | - | Token for email verification |
| 15. |  | VerificationTokenExpiry | DATETIME | Yes |  | - | Expiration date of verification token |
| 16. |  | IsActive | BIT | No | DEFAULT 0 | - | If account is active = 1, otherwise 0 |
| 17. |  | IsLockedOut | BIT | No | DEFAULT 0 | - | If account is locked = 1, otherwise 0 |
| 18. |  | CreatedAt | DATETIME | No | DEFAULT GETDATE() | - | Timestamp when the user was created |
| 19. |  | UpdatedAt | DATETIME | No | DEFAULT GETDATE() | - | Timestamp when the user was last updated |
| 20. |  | ValidTo | Computed (AS) | - | AS DATEADD(MONTH, 6, CreatedAt) | - | Computed column: user valid for 6 months |

**Schema** Users **Primary Key:** 🔑

**Tables** LoginLogs **Foreign Key: 🔑➡**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Key** | **Name** | **Data Type** | **Null** | **Attributes** | **References** | **Description** |
| 1. | 🔑 | LogID | INT | No | PRIMARY KEY, IDENTITY(1,1) | - | Unique identifier for each login attempt |
| 2. | **🔑➡** | UserID | INT | No | FOREIGN KEY | Users.Users(UserID) | References the user attempting to login |
| 3. |  | IPAddress | NVARCHAR(45) | No |  | - | IP address of the user attempting to login |
| 4. |  | Email | NVARCHAR(255) | No |  | - | Email used for login attempt |
| 5. |  | AttemptedAt | DATETIME | No | DEFAULT  GETDATE() | - | Timestamp of login attempt |
| 6. |  | IsSuccess | BIT | No |  | - | If login was successful = 1, otherwise 0 |

# **Email settings and activation**

A screenshot of a computer code

AI-generated content may be incorrect.

Emails would normally be sent from an external program, but we don’t have access to that in this case. This code is added for us to be able to send an email through our stored procedures like:

* Register – An email will be sent with the verification code when a user has been created through this procedure.
* ForgottenPassword – An email will be sent with the new verification code / reset code when a user has requested to recover their password through this procedure.

# **Views**

UserLoginActivity

This view shows each user’s last successful and unsuccessful login attempts.

UserLoginAttempts

This view displays each user’s total amount of login attempts as well as the average successful attempts per user

# **Stored Procedures**

Register

This stored procedure allows a user to create a new account and adds the user to the system.

* The stored procedure sends a verification email to the user.
* The procedure has a specific format for their email that must be followed to prevent non-email formatted words to be used as an email.
* This procedure ensures that a password must have a min. of 8 letters, 1 uppercase letter, 1 number and 1 symbol. If all these requirements aren’t met, then the registration will fail.
* The procedure makes sure that the user’s password is hashed and salted for better security. SHA2\_512 is the hash function used for password hashing.
* The verification code is set to only be valid for one day and a one-time use.
* When a successful registration is done, the user’s information will be inserted into the table “Users.Users”.

Verification

This stored procedure allows the user to verify their newly created account with the verification code they’ve gotten from the verification email.

* Verify your account by entering your email and the verification code you got from the verification email.
* Checks if the email exists.
* Activates and verifies the account, empties the verification token.
* Checks if the verification code is valid.

Forgotten Password

This stored procedure allows a user who has forgotten their password to receive an email with a new verification code to reset their password.

* Ensure that the password is not changed if the account is locked.
* Checks if the email exists.
* Generates a new verification token that is only valid for one day and a one-time use.
* Send an email to the user with the new verification code.

Reset Password

SetForgottenPassword, this stored procedure allows the user to use the verification code they recently got from the Forgotten Password procedure and reset their password.

* Email, verification token and creating your new password.
* The new password is hashed and salted.
* Checks if the email and verification token is valid.
* This procedure ensures that a password must have a min. of 8 letters, 1 uppercase letter, 1 number and 1 symbol. If all these requirements aren’t met, then the registration will fail.
* Ensures that the new password cannot be the same as the old password for better security.
* When all requirements are met, the procedure will update the user’s new password, empty the verification token and activate and verify the user’s account

Login

This stored procedure allows the user to login to their account.

* Login using your email, password and IP Address (X.X.X.X ex: 192.0.2.146. You can create a hypothetical IP Address if you don’t want to use your own). This procedure does not gather the users IP Address automatically hence why you must insert it manually (In reality, an external program would be gathering the users IP Address automatically but in this case it doesn’t).
* Check if the email exists.
* If a user has tried to login more than three times within 15 minutes and failed, then the account gets locked.
* Checks if the account is locked due to too many login attempts.
* Checks if the account is verified.
* Check if account is active.
* The account must be verified, active and not locked to be able to login.

# **Indexes**

Users Table

*idx\_users\_email*

This is a unique index that ensures that no duplicate email exists in the database while improving the performance of queries searching for users by email.

*idx\_users\_role*

This index enhances the performance of queries filtering users based on their role (A or C), making role-based lookups more efficient.

*idx\_users\_email\_lockedout*

This composite index improves the performance of queries filtering users by email and lockout status, optimizing account security checks.

LoginLogs Table

*idx\_loginlogs\_email*

This index speeds up queries searching for login attempts associated with a specific email address.

*idx\_loginlogs\_ipaddress*

This index enhances the performance of queries filtering login attempts by IP address, useful for tracking suspicious activity.

*idx\_loginlogs\_attemptedat*

This index improves the efficiency of queries filtering or sorting login attempts based on timestamp, optimizing login activity analysis.

# **Long-Term Improvements**

User Account Management

For educational purposes, this project focused on creating a functional database and user account management system for a car dealership called “Hederlige Harrys Bilar”. We created a SQL script that allows the user to register and log in. If this were a real project, we would develop a website using one of three common backend technologies: PHP, Python, or Node.js.

**PHP**

It’s one of the go-to language for server-side web development. Using MySQL or MariaDB as the database backend. It’s widely used for building dynamic web applications that include user registration, login flows and password resets.

**Python**

Python is a versatile and highly readable programming language with frameworks like Flask and Django. It provides powerful tools for web development. Due to its clear and concise syntax making it easy to write maintainable and secure authentication systems. Python also includes libraries such as **bcrypt** and **PyJWT**, which enable strong password hashing and token-based authentication.

**Node.js**

It’s powered by JavaScript and is well known for fast, event-driven and non-blocking I/O models. Like the other two backend technologies, Node.js efficiently handles user registrations, logins, and password management. Node.js is perfect for handling high concurrency for websites/applications that’s expecting frequent user interactions.

With the help of one of these backend technologies, we would provide Hederlige Harrys Bilar with a functional database and User Account Management system.

User Experience

Improving the user experience is one of the areas we would focus on, making it easier for our users to use our website would help the overall effectiveness of online purchases.

Just like many other websites, we would implement “**Social Login Integration**” which allows a user to login through third-party accounts such as Google, Facebook or GitHub using OAuth2.0 or OpenID Connect for faster and more convenient registration process.

Ensuring that the website has a “**Responsive design**” for mobile devices and tablets enhances the user experience for a wider audience.

Email

Our script allows SQL to send an email with the verification token. However, sending emails directly from SQL is not ideal due to security concerns and because SQL’s primary purpose is data storage. Just like User Account Management, we would be using one of the three backend technologies we mentioned to send our verification emails.

Two-Factor Authentication (2FA)

2FA adds an extra layer of security by requiring the user to provide two forms of verification. Usually by a password they themselves set and something they have (a device or token) such as Google Authenticator, Authy or Microsoft Authenticator. These authenticators generate temporary codes that the user must enter after providing their password. It could also be SMS-Based 2FA although less secure where the user gets sent a one-time password via SMS to the user’s mobile phone.

Database

For a database that stores user’s data, we will need to create a more secure database than we’ve done with this current project. That would mean that we would need to look to implement *audit trails* for sensitive user actions like password changes, login attempts or email updates.

*Backup & Disaster Recovery* is also instrumental in ensuring a safer database for a company with plenty of customers. By setting up automated database backups and regular testing of disaster recovery procedures, you can ensure your data is safe and can be restored in case of an issue.

*Automated Cleanup* for logs in our database such as our Login Logs table will be crucial to ensure that old data doesn’t take up too much space.

*Indexing and* adding proper indexes for a big database is crucial for faster performance.

User Roles & Permissions

Although there are two roles, "A" for Admin and "C" for Customer (the default role) no specific permissions have been assigned to them yet. In future improvements, we plan to implement role-based permissions to enhance security. Customers should only have access to the information necessary for their interactions, while administrators will be granted the appropriate permissions to perform their tasks efficiently.

# **HHB – SQLQueries**

OBS! Execute the HHB\_SQLScript file before trying to run the commands for this file and use the zoom to read the screenshots that are unreadable due to size otherwise.

In this section we will review the provided SQL file, which contains execution commands for stored procedures, views and SELECT statements to retrieve data from the Users and LoginLogs tables. Screenshots will be included to show the expected results, along with detailed explanations of how each query functions. This will help ensure that when you run the queries yourself, you achieve the same or similar results, depending on the data you input. You will need to execute some execution commands separately or you’ll get errors due to formats set in the Stored Procedures.

**SELECT Statements**

A screenshot of a computer program

AI-generated content may be incorrect.

Start off by using the “USE HederligeHarrysBilar;” code to ensure that it’s the database that’s selected in SSMS.

You can also select the whole code in that screenshot, and you’ll get a result for Users.Users and Users.LoginLogs:

A screenshot of a computer

AI-generated content may be incorrect.

The data is all generated through ChatGPT so we don’t have to worry about GDPR here, these are the two tables built for this project, and you can see all the different columns and the data inside it. Notice how the VerificationToken and VerificationTokenExpiry columns are “NULL”, that’s because these accounts were manually added and not created through our “Register” procedure. Other explanations for that are as documented, the data inside those two columns gets removed after being used once. It’s a table for every users account information registered in the system.

If we look at LoginLogs you can see 4 different users who’ve tried to log in at different points of time.

*Oliver Nilsson* – Tried to log in 2025-02-12 between 01:00 – 01:15 and was successful on his last attempt.

*Kristin Forsberg* – She was successful on both of her login attempts.

*Markus Holm* – Tried to log in 2025-02-14 between 03:00 – 03:15 but was unsuccessful on all his attempts and ended up getting his account locked.

*Josefine Eriksson* – She managed to log in on her first try.

**System Mail – Adding a Database Mail Account**

A white rectangle with blue and green text

AI-generated content may be incorrect.

This script adds a new Database Mail account in SQL Server using the sysmail\_add\_account\_sp stored procedure. The account is configured to send emails via Gmail’s SMTP server.

* Account Name: Hederlige Harrys Bilar AB (Can be modified if needed)
* Email Address: [Testmejl@gmail.com](mailto:Testmejl@gmail.com)
* SMTP Server: smtp.gmail.com (Port 587, SSL enabled)
* Authentication: Uses a Google App Password for secure authentication.

This setup allows stored procedures to send emails via this configured mail account. No modifications are necessary unless specific changes are required for a different email setup.

When executing this code, the result will be “Commands completed successfully”. You can just leave it as is, the purpose for this code is only for the procedure to have the function to send emails. Running this is not a must so you can technically skip this code.

**Register**

A computer code with text

AI-generated content may be incorrect.

This Stored Procedure allows you to register a new account to the database, insert your account information by replacing the data with “test” in them with your own credentials.   
A close-up of a computer screen

AI-generated content may be incorrect.

Results:

A close-up of a computer screen

AI-generated content may be incorrect.

By using the SELECT statement for Users.Users, a new account will be created for you:

A screenshot of a computer

AI-generated content may be incorrect.

Notice how your account has a verification token and an expiring date, that’s because you created your account by using the *Register* procedure.

Results with incorrect email format:

A screenshot of a computer

AI-generated content may be incorrect.

Results with incorrect password format:

A screenshot of a computer

AI-generated content may be incorrect.

Results if the inserted email exists:

The specified email address is already in use.

**Verification**

A screen shot of a computer program

AI-generated content may be incorrect.

Now that you’ve registered a new account, you will need to verify your account. Use the SELECT statement provided in the procedure before running the EXEC Verification code. Don’t forget to replace the existing test email with the email you created your account with on both the SELECT statement and EXEC Verification code.

Result:

A screenshot of a computer

AI-generated content may be incorrect.

Now copy the verification token and insert it to @VerificationToken.

A computer screen shot of a computer code

AI-generated content may be incorrect.

Run the “EXEC Verification” code.

Result:

A screen shot of a computer

AI-generated content may be incorrect.

What went wrong?

The answer to that is because we made a typo on the email. Notice how the email has a “2”?

Once we correct that, the result will be:

A screenshot of a computer

AI-generated content may be incorrect.

Now that the account is verified, IsVerified and IsActive will be changed to “1” and the VerificationToken and VerificationTokenExpiry columns will be emptied.

A screenshot of a computer

AI-generated content may be incorrect.

Results if email or verification code is incorrect and/or the verification code has expired:

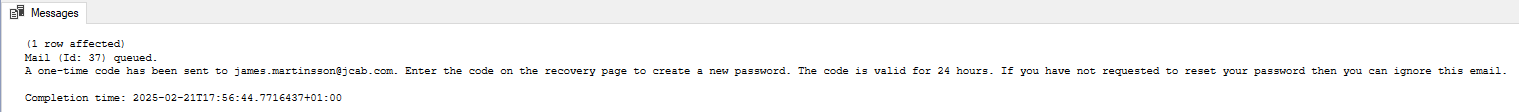
Verification failed. Invalid email or verification code and/or the verification code has expired.

**ForgottenPassword**



This code allows the user to receive a new verification code and reset their password via the “SetForgottenPassword” procedure.

Result:



Result if the email does not exist:  
A screenshot of a computer

AI-generated content may be incorrect.

**SetForgottenPassword**

A screen shot of a computer

AI-generated content may be incorrect.

After requesting a new verification code from the “ForgottenPassword” procedure, it’s time to use this procedure and reset your own password. Change the “test” from each row with your own credentials. Use the SELECT statement first to get your new verification code and then insert it in the “EXEC SetForgottenPassword” command.

A screenshot of a computer

AI-generated content may be incorrect.

Notice how a completely different verification code is generated from when you first registered your account? That’s because it’s made to generate a new verification code every time. When a new password is set, then those two columns will once again be emptied.

A screen shot of a computer

AI-generated content may be incorrect.

It’s time to run the “EXEC SetForgottenPassword” command.

Result:

A screenshot of a computer

AI-generated content may be incorrect.

Entering the same password as your old one is not allowed, you’ll have to insert a completely new password.

Correct Result:  
A screenshot of a computer

AI-generated content may be incorrect.

Result if incorrect email or verification code and/or the verification code has expired:

You cannot change the password. Invalid email or verification code and/or the verification code has expired.

**Login**

A close up of text

AI-generated content may be incorrect.

This is the last procedure and when you’ve registered and verified your account, you’ll be able to log in with this procedure. Replace the “test” with your own credentials.

But what about @IPAddress? Although it would be possible to obtain a user’s IP Address automatically, we decided to let the user for this project to insert an IP Address of their choosing (The idea of obtaining a user’s IP Address automatically is for it to be external and not through SQL for security purposes.), it doesn’t need to be a real IP Address.

A close up of a computer screen

AI-generated content may be incorrect.

Result:

A number and numbers on a white background

AI-generated content may be incorrect.

Result if incorrect password is entered:

A close-up of a computer screen

AI-generated content may be incorrect.

Result if a user has attempted to log in three times in the past 15 minutes:

A screenshot of a computer

AI-generated content may be incorrect.

You have now managed to lock your account, if we lock at users.users table, the column “IsLockedOut” on your account will now be changed from “0” to “1”:

A screenshot of a computer

AI-generated content may be incorrect.

If a user’s account is locked, it won’t be possible to request a verification code through the “ForgottenPassword” procedure. That means you won’t be able to reset your password if your account is locked. The result would be:

Your account is locked, please contact our customer service.

Result if your account isn’t verified after a login attempt:

Your account has not been verified, please verify your account via email.

Result if your account isn’t activated after a login attempt:

Your account is inactive.

**Views**

A white background with black text

AI-generated content may be incorrect.

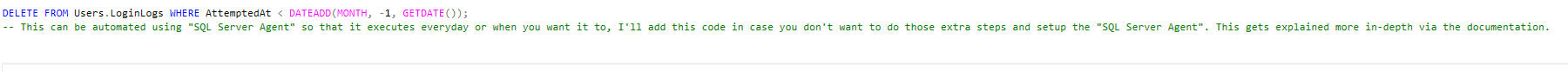
There’s not much to describe regarding these views since there’s already a segment in this documentation about Views and the descriptions for these views hence, we will just add the results here. You can see that aside from the manually inserted data, your own login attempts will be included when you run these views.

Result:

A screenshot of a computer

AI-generated content may be incorrect.

**Delete old logs**



This code is added in here just for the purpose of allowing a user to delete old logs that’s older than a month (It’s for the educator of this course, so that he doesn’t need to set up an SQL Server Agent Job to save him some time). If it were to be in a real project, then it would be automated using SQL Server Agent.

But if you want to automate it then here is a quick guide to how you can do it:

1. **Enable SQL Server Agent**

* Go to the “*Object Explorer*” and expand “*SQL Server Agent*”.
* If it’s disabled, then right-click *SQL Server Agent* and select “*start*”.

1. **Create an SQL Agent Job**

* Right-click *SQL Server Agent* hover *New* > click *Job*.
* In the general tab: give the job a name, optionally add a description

1. **Add a Job Step**

* Go to the Steps tab and click New.
* In the New Job Step window: Give the step a Step Name,
* Type: Transact\_SQL (T-SQL),
* Select Hederlige Harrys Bilar database
* Enter this query: DELETE FROM Users.LoginLogs WHERE AttemptedAt < DATEADD(MONTH, -1, GETDATE());
* Click *OK*.

1. **Schedule the Job**

* Go to the *Schedules* tab and click *New*.
* Give the schedule a name.
* Set the frequency:
* Occurs: Monthly,
* Recurs every: 1 month,
* Time: Choose a time when the database is the least busy.
* Click *OK*.

Once you’ve saved the job, you’re done, and it’ll remove logs that are older than 1 month from Users.LoginLogs.

*This concludes the documentation. We appreciate your time and hope this project meets your expectations. If you have any questions or require further assistance, please feel free to reach out.*