**Databasdokumentation**

**Översikt**  
Jag har fått i uppdrag att designa och implementera en databas för användarlogin till ”Hederlige Harrys bilar”. Databasen hanterar användarautentisering för åtkomst till applikationen.

Den innehåller tabeller för:

* **Användardata**-lagring av användarinformation och behörigheter.
* **Inloggningsloggar**-registrering av lyckade och misslyckade inloggningsförsök.
* **Säkerhetsövervakning**- spårning av misslyckade inloggningar för att identifiera potentiella säkerhetshot.

**Skapande av databasen ”Harrys Bilar”**

Denna SQL-script används för att kontrollera om det finns en databas med namnet ”HarrysBilar”. Om det finns en tidigare, stänger den av alla andra användare genom att sätta databasen i SINGLE\_USER-läge och rullar tillbaka eventuella pågående transaktioner med ROLLBACK IMMEDIATE, innan den raderar (DROP) databasen. Observera att detta radera den befintliga databasen och all dess data. Används i normalfall för testning och utveckling av databasen.

Efter att databasen har raderats, skapas en ny databas, genom koden ”Create Database” med namnet, ”HarrysBilar” och aktiverar den med kommandot USE, vilket initierar databasen för vidare användning. A screenshot of a computer code

AI-generated content may be incorrect.

**Tabeller och Kolumner**

Databasen består av två huvudtabeller. Vi har en Log-tabell som har Many-to-one relation till Users-tabellen.

1. **Users**

Denna tabell lagrar information om varje användare i systemet, inklusive autentiseringsuppgifter och status.

* **Primärnyckel (PK)**: UserID
* **Email** – Unik e-postadress för varje användare.
* **FullName** – Användarens namn.
* **Role** – Användarens roll, admin eller kund.
* **Adress** –Adress.
* **PostalCode** – Postnummer.
* **City** – Staden.
* **Country** – Land.
* **PasswordHash** **och Salt** - Säker lösenordshantering.
* **FailedAttempts** – Antal misslyckade inloggningsförsök.
* **VerificationToken** – En unik token som används för att verifiera e-post eller återställa lösenord.
* **TokenExpiry** – Anger när VerficationToken upphör att gälla, för att förhindra missbruk.
* **IsVerified**-Indikerar om användaren är verifierad.
* **IsActive** – Indikerar om användare är aktiv.
* **IsLocked**-Indikerar om kontot är låst.
* **CreatedAt**- Tidpunkt när kontot skapades.
* **ValidTo**- Tidpunkt hur länge kontot är giltig.

1. **Log**

Denna tabell lagar information om varje inloggningsförsök, både lyckade och misslyckade.

* **Primärnyckel (PK)**: LogID
* **Främmande nyckel (FK)**: UserID som regerar till Users-tabellen.
* **Email** – E-postadressen som används vid inloggningen.
* **LogTimestamp** – Tidsstämpel för inloggningsförsöket.
* **Sucess** – Indikerar om inloggningen är lyckad.
* **IPAddress** – Användarens IP-adress vid inloggningen.
* **LogMessage** – Beskrivning av händelsen till exempel ¨Felaktigt lösenord¨

**ER-Diagram  
A screenshot of a computer

AI-generated content may be incorrect.**

**Säkerhet  
Hashning och lösenordshantering:**

För att säkerställa att användarens lösenord hanteras på ett säkert sätt, används lösenordshashning tillsammans med en unikt salt-sträng. Varje användare får en unik salt-sträng, som generas med NEWID(). Lösenordet kombineras med salten och hashas med SHA2-512 via HASHBYTES(). Detta förhindrar att lösenord kan återställas från hashen.

**Åtgärder mot brute-force-attacker:**

För att skydda användare mot brute-force-attacker implementeras åtgärder som begränsar antalet inloggningsförsök. Efter 3 misslyckade inloggningsförsök inom 15 minuter kommer kontot att låsas, vilket förhindrar angripare från att prova olika lösenord i kort följd.

**Roll – och behörighetshantering:**

Det styr vilka rättigheter och åtkomstnivåer olika användare har. **Admin** har full åtkomst till systemets inställningar och användardata. **Customer** har åtkomst till sin egen kontoinformation och transaktioner.

**Databasfunktionalitet**

För att effektivisera datahantering i systemet används både Views (vy) och Stored Procedures. Views används för att snabbt kunna hämta fördefinierade data utan att behöva upprepa komplexa SQL-frågor. Stored Procedures används för att hantera databaslogik och automatisera uppgifter, till exempel låsa konton eller lösenordåterställning.

**Views**

1. **UsersLogInformation**

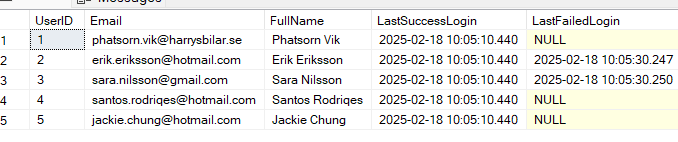
View med CTE (Common Table Expression) som tar fram alla användares senaste datum för lyckade och misslyckade inloggningsförsök. Genom att använda CTE och CASE WHEN, aggregeras den senaste inloggningsinformationen direkt från Log-tabellen.

Använder MAX() i kombination med CASE WHEN för att få den senaste lyckade (Success = 1) och misslyckade (Success = 0) inloggningen. GROUP BY UserID säkerställer att varje användare endast får en post med sin senaste inloggningshistorik.

Hämtar sedan användarens UserID, Email, FullName från Users-tabellen. Använder en LEFT JOIN mot LatestLogStatus\_CTE för att koppla samman användarinformation med inloggninshistorik.

Exempel:





1. **UserLoginDetails**Denna view används för att analysera inloggningsförsök per IP-adress.En view med fönsterfunktion som tar fram användarens e-port, IP-adress, loggningstid, totala försök, lyckade och misslyckade inloggningar samt genomsnittet på de lyckade inloggningsförsöken. Detta görs genom att använda COUNT(), SUM() och AVG() med fönsterfunktioner (OVER) för att partitionera datan baserat på IP-adress och ordna den efter inloggningstiden.

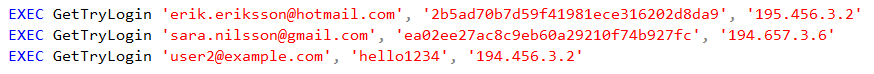
**Stored Procedures**

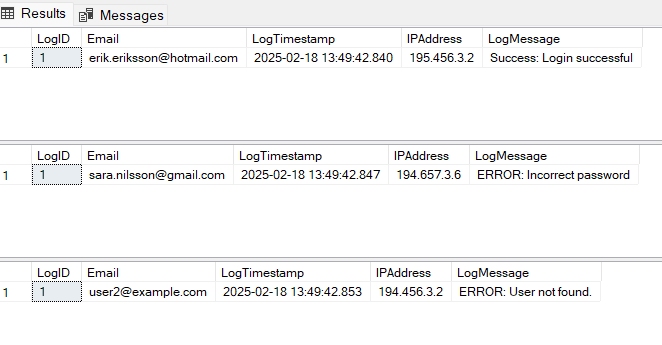
1. **GetTryLogin**Denna lagrade procedur, GetTryLogin, hanterar inloggningsförsök för användare. Den tar emot tre parametrar: användarens e-postadress (@Email), lösenord (@Password) och IP-adress(@IP-Address).

* **Användarverifiering:** Proceduren hämtar användarens ID, lösenord (hashat), salt, antal misslyckade inloggningsförsök och kontots status från Users-tabellen baserat på den angivna e-postadressen.
* **Felhantering:** Om användaren inte finns eller om kontot är låst, loggas felmeddelande och inloggning nekas.
* **Misslyckade försök**: Om det finns fler än 3 misslyckade försök på 15 minuter låses kontot.
* **Lösenordsverifiering**: Om lösenordet inte matchar det lagrade lösenordet, loggas ett felmeddelande och antalet misslyckade försök ökas.
* **Lyckad inloggning**: Vid lyckad inloggning återställs misslyckade försök och lyckad inloggning loggas.

Alla aktiviteter, inklusive misslyckade och lyckade inloggningsförsök, loggas i en temporär tabell och i den permanenta Log-tabellen för framtida granskning.

Exmepel:

Kallar på GetTryLogin procedure.  


Här ser vi resultaten i #LogTable.

Allt loggas i Log-tabellen.



1. **GetForgotPassword**Denna procedur hanterar förfrågan om lösenordsåterställning genom att den generar ett unikt token för användare som har glömt sitt lösenord. Token gäller i 24 timmar och används för att kunna återställa lösenord. Den tar emot parametern e-postadressen (@Email) för att identifiera användaren.

* **Användarverifiering**: Hämtar användarens ID och nuvarande token från Users-tabellen baserat på den angivna e-postadressen.
* **Felhantering**: Om användaren inte finns i databasen loggas ett felmeddelande och proceduren avslutas.  
  Om det redan finns ett aktivt token (ej utgånget), loggas felmeddelande och användaren kan inte begära ett nytt token.
* **Token generering:** Om inga tokens finns, generas ett nytt token(@NewToken) och uppdaterar användarens VerificationToken och TokenExpiry i databasen. TokenExpiry är giltig i 24 timmar.
* **Bekräftelse:** Om token skapas utan problem, loggas ett bekräftelsemeddelande.

1. **GetSetForgottenPassword**

Denna procedur hanterar återställning av lösenord för användare som har begärt återställning via ett token. Den verifierar token och uppdaterar lösenordet om alla villkor är uppfyllda. Den tar emot 3 parametrar: e-postadressen (@Email), nya lösenordet (@NewPassword) och token som genererades för att tillåta lösenordsåterställning (@Token).

* **Användarverifiering:** Hämtar användarens UserID, token och tokenets utgångstid baserat på e-postadressen.
* **Felhantering:** Om användaren inte finns i databasen, loggas felmeddelande.   
  Om tokenet saknas, är ogiltigt eller inte matchar, loggas felmeddelande.   
  Om tokenet har gått ut, loggas felmeddelande.
* **Lösenordsuppdatering:** Om alla parametrar stämmer, generas ett nytt salt och lösenordet hashas. Användarens lösenord uppdateras, och token tas bort för att förhindra återanvändning.
* **Bekräftelse:** En bekräftelselogg skrivs ut om lösenordet har ändrats.

**Vidare utveckling och utbyggnad**

Denna databas och dess stored procedures utgör en grundläggande struktur för hantering av användarautentisering och loggning. Systemet kan byggas ut med fler funktioner för att öka säkerheten och förbättra användarupplevelsen. Några exempel på vidareutveckling kan vara en e-postverifikation, där vi kan ha en SP som genererar en verifikationskod och skickar den till användarens e-post vid registrering. En annan SP kan användas för att verifiera koden innan kontot aktiveras.

I dagsläge blir kontot låst efter 3 misslyckades försök inom 15 minuter. Denna låsning kan enbart låsas upp utav admin, en annan mer användarvänlig funktion skulle vara att kontot endast blir låst i 30 minuter och blir sedan tillgänglig för nya inloggningsförsök.

**Optimering**

För närvarande har databasen ett fåtal användare, vilket gör att prestandan är bra. Men i framtiden när databasen växer, kan sökningar och filtreringar bli mer tidskrävande. För att förebygga detta har jag implementerat non-clustered index på relevanta kolumner.

* **Email**: ett index på Email-kolumnen gör att vi snabbt kan identifiera en specifik användare och hämta dennes historik.
* **IP-adress**: Ett index på IPAddress-kolumnen underlättar analys av inloggningsförsök från specifika IP-adresser.

A screen shot of a computer

AI-generated content may be incorrect.

För att utvärdera optimeringen genomfördes en prestandatestning före och efter indexeringen.

**Före indexering**:

* SQL Server behövde 106 ms för att exekvera frågan.
* Users-tabell - Physical reads 1. Physical reads är antal sidor som SQL hämtar från disken.
* Log-tabell – Physical reads 1.

**Efter indexering**:

* Exekveringstiden minskade till 92 ms.
* Users-tabell minskade physical reads till 0.
* Log-tabell minskade physical reads till 0.

Även om skillnaden är marginell i nuläget så kommer dessa index att ge betydande förbättringar när databasen expanderar och hanterar större datamängder. Indexen bidrar till snabbare sökningar och mer effektiv frågeoptimering övertid.