Fjerde undervisningsgang

Database

Denne undervisningsgang

- Lektier fra sidst
- Dagens eksempel database
- Transaktioner fortsat
- Views
- Sikring af adgang til databaser
- Sikring af input til databaser
- C# og databaser
- C++ og databaser
- Java og databaser
- Anbefalet læsning
- Andre fag fremover
- Eksamen og krav

Lektier fra sidst

Det der er produceret vises og forklares til underviseren og resten af klassen

Dagens eksempel database

Hent de to sakila sql filer fra dagens lektion på Fronter

Start derefter MAMP og MySQL Workbench og åbn forbindelsen til din lokale databaseserver og importer de to scripts, schema først og derefter data

Den er beskrevet i detaljer i den officielle dokumentation i sakila_en.pdf filen

Transaktioner fortsat

Opsummerende eksempel på hvorfor man bruger transaktioner

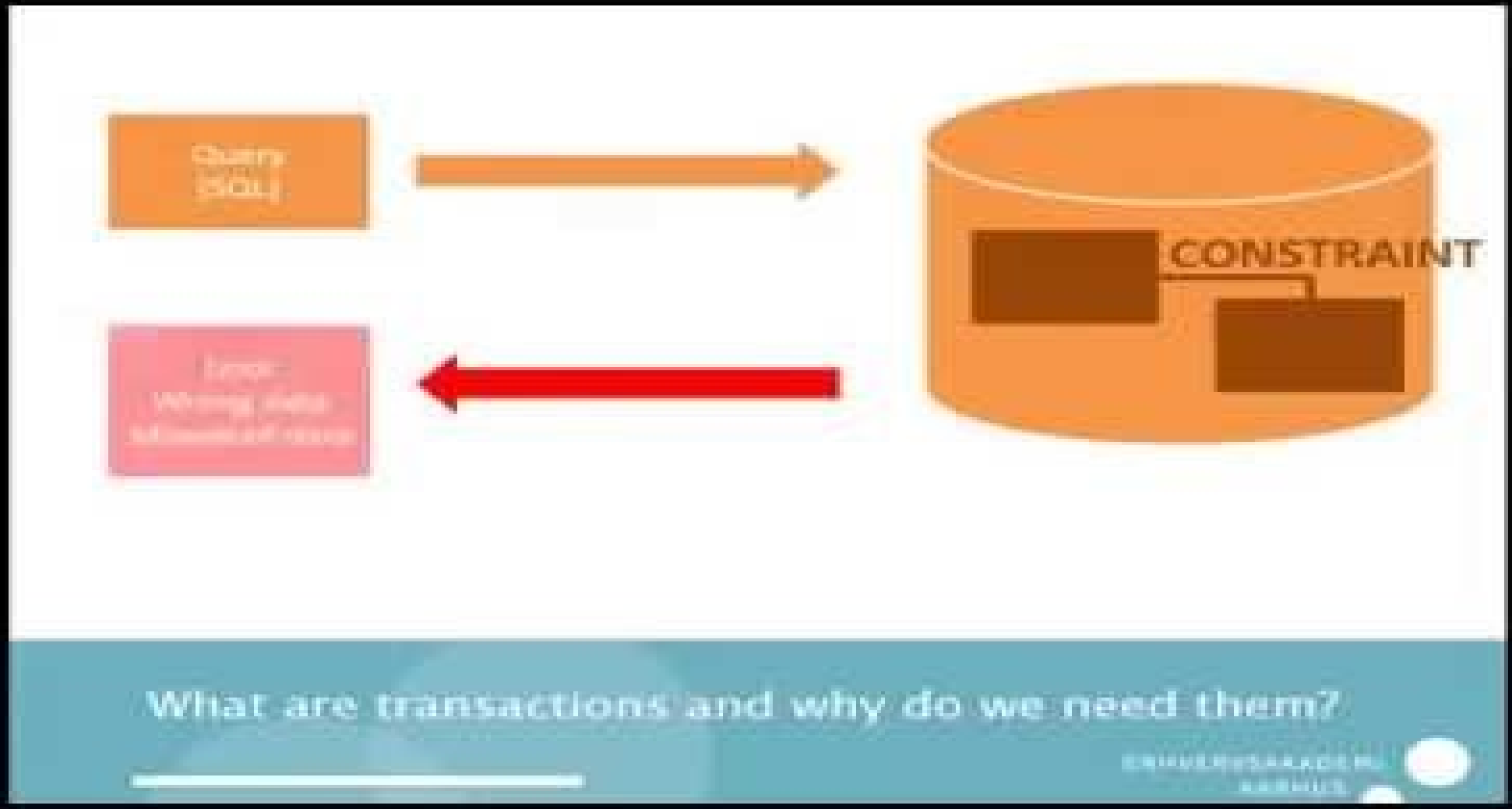
- En transaktion er et sæt af operationer udført, så alle operationer er garanteret at lykkes eller fejler som en enhed.
  - Altså alt eller intet
- Et klassisk eksempel på en transaktion er processen med at overføre penge fra en lønkonto til en opsparingskonto.
  - Dette involverer to operationer:
    - Udtræk af penge fra lønkontoen og
    - Tilføjelse af dem til opsparingskontoen.
  - Begge skal lykkes sammen, og ændringerne skal indgå i regnskabet, eller skal begge fejle sammen og rulles tilbage, så regnskaberne opretholdes i en konsekvent tilstand.
  - Under ingen omstændigheder skal penge fratrækkes lønkontoen, men ikke tilføjes til opsparingskontoen (eller omvendt).
  - Ved at bruge et transaktionskoncept kan både transaktionerne, nemlig debet og kredit, garanteres at lykkes eller fejle sammen. Så begge konti forbliver i konstant tilstand hele tiden.

Hvornår giver det mening at bruge transaktioner

- Nedenstående er nogle hyppige scenarier, hvor brug af transaktioner anbefales:
  - I batchbehandling, hvor flere rækker skal indsættes, opdateres eller slettes som en enkelt enhed
  - Når en ændring til en tabel kræver, at andre tabeller holdes konsistente
  - Når man ændrer data i to eller flere databaser samtidigt
  - I distribuerede transaktioner, hvor data manipuleres i databaser på forskellige servere
- Når man bruger transaktioner, sætter man låse på data, der er under behandling for permanent ændring i databasen.
  - Ingen andre operationer kan finde sted på låst data, indtil den overtagne lås er frigivet.
  - Du kan låse alt fra en enkelt række til hele databasen.
- Dette kaldes samtidighed, hvilket betyder, hvordan databasen håndterer flere opdateringer ad gangen.
- I bankeksemplet tidligere sikrer låsene, at to separate transaktioner ikke får adgang til de samme konti på samme tid. Hvis de gør det, kan enten indskud eller udbetalinger gå tabt.

Alt foregående kan opsummeres i ACID

- Betegnelsen ACID blev defineret af Andreas Reuter i 1983.


CONSTRAINT
What are transactions and why do we need them?

Views

Hvad er et VIEW

- Et VIEW er et virtuelt table, hvorigennem en selektiv del af dataene fra en eller flere tabeller kan ses.
  - VIEWs indeholder ikke egen data.
- De bruges til at begrænse adgangen til databasen eller til at skjule datakompleksitet.
- Et VIEW er ikke mere end en SQL-sætning, der er gemt i databasen med et tilknyttet navn. Et VIEW er faktisk en sammensætning af et table i form af en foruddefineret SQL-forespørgsel.
- Et VIEW kan indeholde alle rækker fra en tabel eller vælge rækker fra en tabel.
- Et VIEW kan oprettes fra en eller flere tabeller, der afhænger af den skrevne SQL-forespørgsel til at oprette et VIEW.

VIEWs er altså virtuelle tabeller der kan…

- Strukturen data på en måde, som brugere eller klasser af brugere finder naturlige eller intuitive.
- Begrænse adgangen til data på en sådan måde, at en bruger kan se og (nogle gange) ændre præcis, hvad de har brug for og ikke mere.
- Opsummere data fra forskellige tabeller, der kan bruges til at generere rapporter.

Syntaksen for VIEW

```
CREATE VIEW view_name
AS
SELECT column_list
FROM table_name [WHERE condition];
```

- view_name er navnet på VIEWet.
- SELECT statementet bruges til at definere de kolonner og rækker, som du vil vise i VIEWet.

Syntaksen for VIEW

- Lad os lave (endnu) et VIEW hvor vi ser skuespillerne

```
CREATE VIEW view_actor
AS
SELECT first_name, last_name
FROM actor;
```

WITH CHECK OPTION

- WITH CHECK OPTION er en CREATE VIEW statement option.
- Formålet med WITH CHECK OPTION er at sikre at alle UPDATE og INSERTs opfylder krav(ene) I view definitionen.
  - Hvis de ikke opfylder krav(ene) vil UPDATE og INSERT returnere en fejl.

```
CREATE VIEW view_actor
AS
SELECT first_name, last_name
FROM  actor
WHERE last_name IS NOT NULL
WITH CHECK OPTION;
```

Opdatering af et VIEW

- Et VIEW kan opdateres under følgende omstændigheder:
  - SELECT-klausulen må ikke indeholde søgeordet DISTINCT.
  - SELECT-klausulen må ikke indeholde oversigtsfunktioner (summary functions).
  - SELECT-klausulen må ikke indeholde indstillede funktioner (set functions).
  - SELECT-klausulen må ikke indeholde indstillede operatører.
  - SELECT-klausulen må ikke indeholde en ORDER BY klausul.
  - FROM-klausulen må ikke indeholde flere tabeller
  - WHERE-klausulen må ikke indeholde subqueries.
  - Forespørgslen må ikke indeholde en GROUP BY eller HAVING.
  - Beregnede kolonner må ikke opdateres.
  - Alle NOT NULL kolonner fra grund tabellen skal medtages i visningen for at INSERT-forespørgslen skal fungere.

Opdatering af et VIEW

- Når kravene på sidste slide er opfyldt må vi opdatere et VIEW, således vil følgende virke

```
UPDATE view_actor
SET last_name = HANKS
WHERE first_name = 'PENELOPE';
```

Indsætning og sletning af rækker i et VIEW

- Rækker af data kan indsættes i en visning.
  - De samme regler som gælder for UPDATE-kommandoen gælder også for INSERT-kommandoen.
- Efter vores WITH CHECK kan vi dog ikke indsætte rækker i view_actor, da vi ikke har inkluderet alle NOT NULL-kolonnerne i denne visning, ellers indsættes rækker i en visning på samme måde som man indsætter dem i en tabel.

- Rækker af data kan også slettes fra en visning.
- De samme regler som gælder for UPDATE og INSERT kommandoerne gælder for DELETE kommandoen.

Sletning af rækker i et VIEW

- Følgende er et eksempel for at slette en post med `first_name = 'PENELOPE';`.

```
DELETE FROM view_actor
WHERE first_name = 'PENELOPE';
```

Sletning af VIEWs

- Man kan selvfølgelig slette et VIEW når man ikke har behov for det mere. Dette gøres ligesom sletning af tabeller og databaser.

```
DROP VIEW view_name;
```

- Således sletter vi det VIEW vi har arbejdet med:

```
DROP VIEW view_actor;
```

Data fra VIEW til ny tabel

- Man kan let få data fra et VIEW over i en ny tabel:

```
CREATE TABLE new_actor
AS(SELECT * FROM view_actor);
```

Opsummering af VIEWs

- Med et VIEW har man adgang til tabel data uden at man har direkte adgang til tabellen
  - Dette betyder at man ikke kan slette data i tabellen ved et uheld
- Man kan de mest essentielle funktioner fra en tabel
- Man kan let få ens VIEWs gemt som ny tabeller

Sikring af adgang til databaser

Folk skal ikke have adgang til noget de ikke har behov for

Sikring af en SQL server – 10 ting

1. Det fysiske miljø

Sikring af det fysiske miljø på din databaseserver er afgørende.

- Forestil dig at have din SQL Server-forekomst hærdet til det maksimale sikkerhedsniveau, men efterlader den fysiske placering af databaseserveren med svag sikkerhed.
- Dette ville virkelig være en modsigelse.
- Som sådan skal du begrænse den fysiske adgang til din fysiske databaseserver. For at opnå dette skal du etablere de korrekte procedurer, der skal følges sammen med tilstrækkelige kontroller, så kun autoriseret personale har fysisk adgang til serverne.

2. Operativ system

Sikring af operativsystemet, på hvilket SQL serveren er installeret, er også vigtigt.

Hvis operativsystemet ikke er sikret, kan en potentiel angriber f.eks. få adgang til dine SQL Server-instans data og logfiler og dermed få adgang til dine data.

For at sikre operativsystemet kan du følge nedenstående retningslinjer:

- Hold det ajour med de nyeste patches og service packs (efter at du har kontrolleret, at de ikke vil påvirke databaseserverens funktion på nogen måde ved korrekt at teste dem).
- Følg princippet om mindst privilegium for servicekonti
- Følg også den mindst privilegerede konto for andre konti.
- Begræns adgangen til SQL serverens fysiske filer ved at indstille den korrekte mappe og filsikkerheds-rettigheder.

Sikring af en SQL server – 10 ting

3. Netværk
   - Alle data i en organisation rejser gennem netværket.
     - Der er databaseservere, applikationsservere, klienter, Storage Area Network (SAN) etc.
   - Netværket skal sikres for at begrænse adgangen til ressourcer fra uautoriserede kilder, samt ikke tillade data at strømme til uautoriserede destinationer.
   - For at opnå dette sikkerhedsniveau skal man konfigurere firewalls korrekt.
     - For eksempel skal man sætte en firewall mellem databaseserveren og internettet.
4. Applikation

   Når det kommer til databasesikkerhed, handler det ikke kun om at sikre ens SQL Server-forekomster. Det har også at gøre med at sikre applikationen, som forbinder til SQL Server-forekomsten.

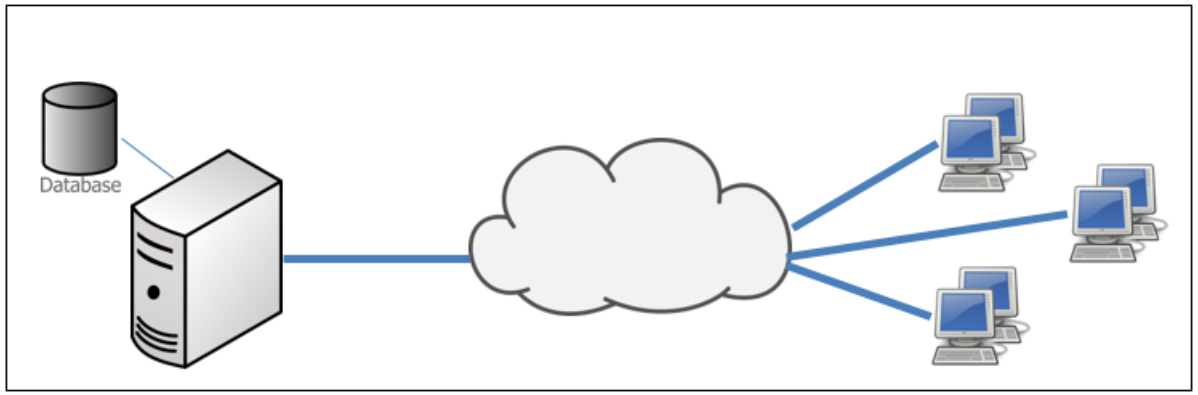
   Årsagen er, at datastrømme finder sted mellem SQL Server-forekomsten og applikationen, så du skal sørge for, at alle tre enheder, der deltager i denne meddelelse, er blevet sikret.

Sikring af en SQL server – 10 ting

4. Applikation

Der er typisk tre entitetstyper, der deltager i sådan datastrøm:

- Database server og instans
- Klienter (dvs. applikationsserver eller direkte klientforbindelser)
- Netværks forbindelse



Sikring af en SQL server – 10 ting

4. Applikation

For at sikre din applikation bør man sørge for:

- Udsæt ikke brugeradgangskoder i kode eller i eksterne filer (dvs. fil med forbindelsesstreng), der bruges af applikationen. Brug i stedet krypterede forbindelsesstrenge.
- Opret en krypteret forbindelse til ens SQL server-forekomst. Med en krypteret forbindelse, selv hvis en potentiel angriber opfanger netværkstrafikken mellem ens klient og SQL-server-forekomsten, kan han eller hun ikke læse data, fordi det ikke er klar tekst, men snarere krypterede datapakker.

Sikring af en SQL server – 10 ting

5. Adgang til serveren

   Overvej om der skal være fjernadgang til kritiske administrations funktioner på serveren eller om de kun skal kunne tilgås lokalt fra en administrator der sidder ved den fysiske server.

6. Bruger roller

   Overvej hvilke dele de enkelte brugere har behov for at have adgang til og lav konti med begrænsede rettigheder.

7. Fjern inaktive brugere

   Slet brugere så snart de ikke benyttes mere.

8. Adgangskode politikker

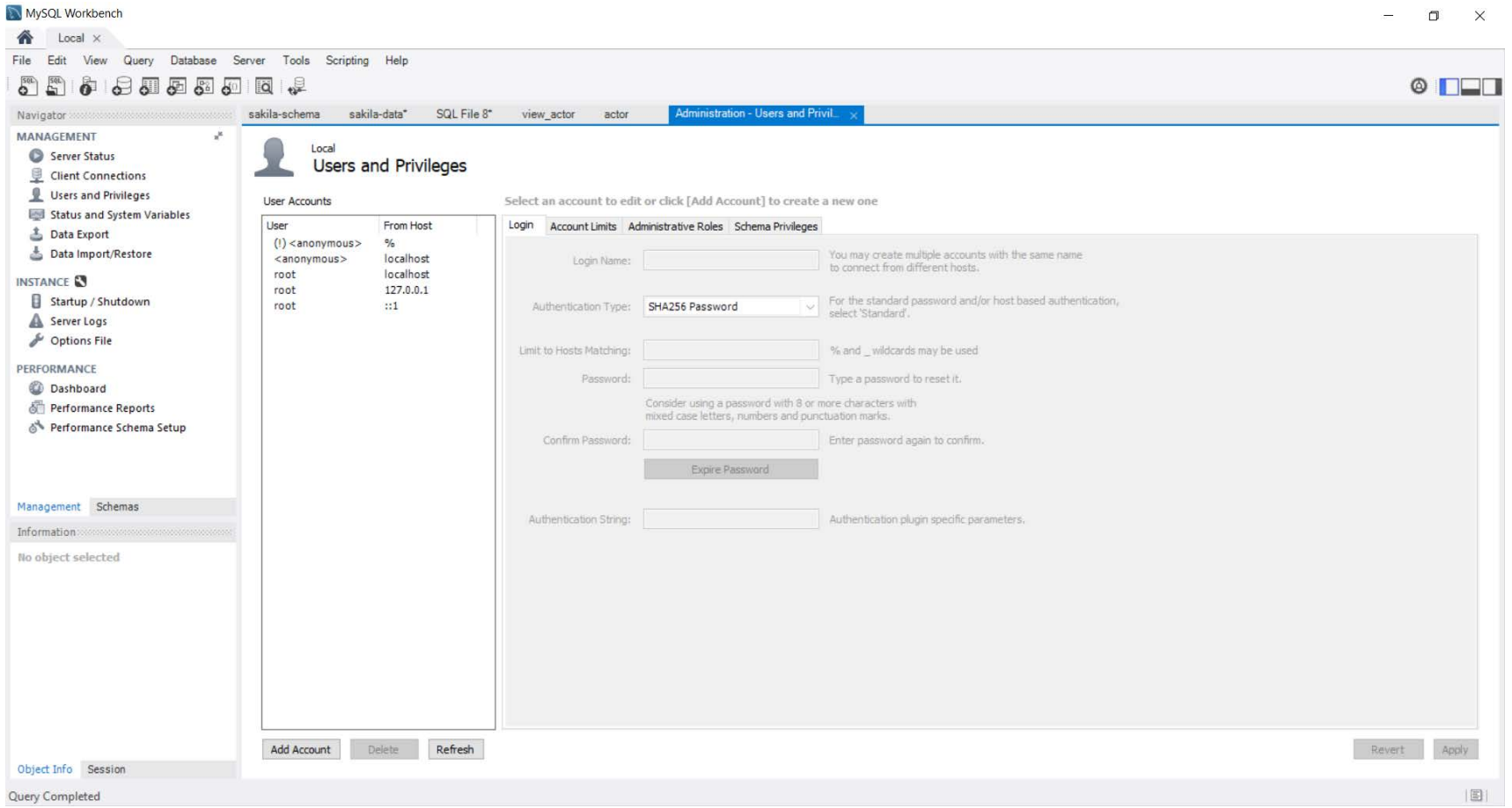
   Sørg for at der (tvunget) benyttes stærke, komplekse kodeord.

Sikring af en SQL server – 10 ting

9. Hold serverens software opdateret og patchet
10. Krypter databasen

    Hvis databasen indeholder følsom data, f.eks. persondata, så sørg for at den er krypteret. De fleste typer server-software understøtter forskellige former for dette.

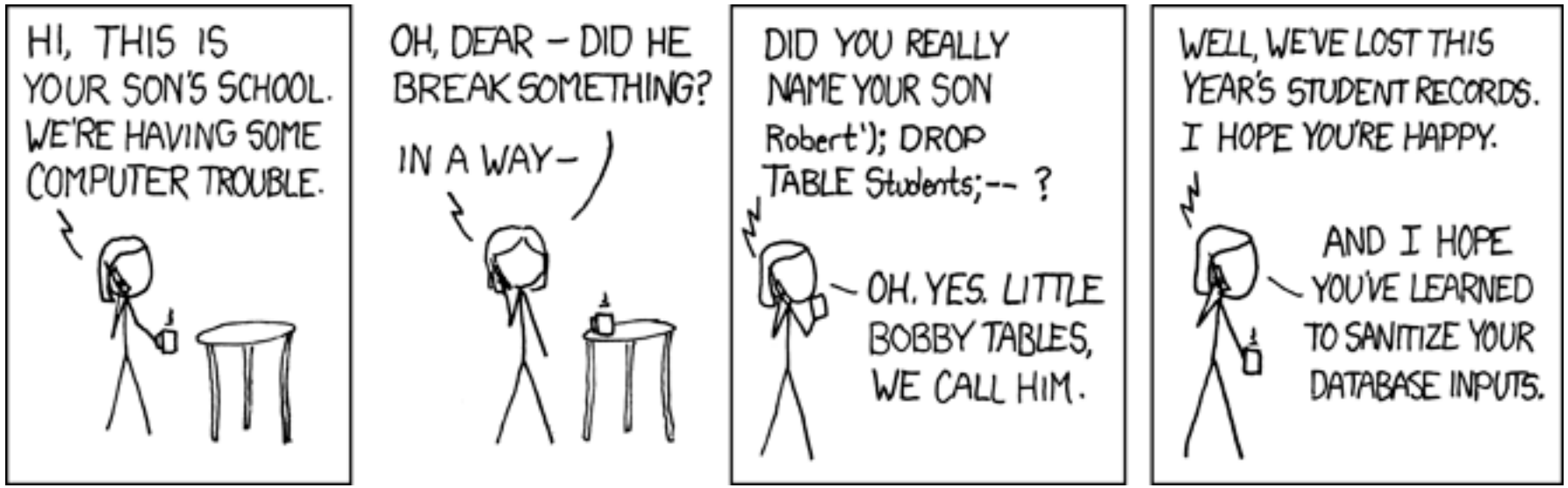
Roller – vi kigger på MySQL Server via Workbench



Sikring af input til databaser

Så brugerne ikke kan putte usikkert indhold i databasen

Den klassiske stribe



SQL injections introduktion

- SQL-injection fejl introduceres, når softwareudviklere opretter dynamiske databaseforespørgsler, der indeholder brugerleveret input.
- At undgå SQL-injections er principielt enkelt. Udviklere skal enten:
  A. stoppe med at skrive dynamiske forespørgsler; og / eller
  B. forhindre brugerleveret input, der indeholder ondsindet SQL, fra at påvirke logikken i den udførte forespørgsel.
- Her kigger vi på SQL database, men andre database typer kan også have problemer (f.eks. XPath og XQuery-injektion), og disse teknikker kan også bruges til at beskytte dem.
- Der er fire primære måder at undgå injections på, hvoraf de første tre er de mest effektive.

Eksempel på en SQL injection

```
String query = "SELECT account_balance FROM
user_data WHERE user_name = "
   + request.getParameter("customerName");


try {
    Statement statement =
connection.createStatement( … );
    ResultSet results = statement.executeQuery(
query );
 }
```

Prepared statements

- Brugen af prepared statements med variabel binding (aka parameteriserede forespørgsler) er, hvordan alle udviklere først bør lære at skrive databaseforespørgsler.
- De er enkle at skrive og lettere at forstå end dynamiske forespørgsler.
- Parameteriserede forespørgsler tvinge udvikleren til først at definere al SQL-koden og derefter sende hver parameter til forespørgslen senere.
- Denne kodestil gør det muligt for databasen at skelne mellem kode og data, uanset hvilken brugerindgang der leveres.
- Prepared statements sikrer, at en angriber ikke kan ændre hensigten med en forespørgsel, selvom SQL-kommandoer indsættes af en angriber.

Prepared statements

- Sprogspecifikke anbefalinger:
  - Java EE - brug PreparedStatement () med binde-variabler
  - .NET - brug parameteriserede forespørgsler som SqlCommand () eller OleDbCommand () med binde-variabler
  - PHP - brug PDO med stærkt typede parametriske forespørgsler (ved hjælp af bindParam ())
  - SQLite - brug sqlite3_prepare () for at oprette et sætningsobjekt
- Under sjældne omstændigheder kan prepared statements skade ydelsen. Når man konfronteres med denne situation, er det bedst enten at
  A. stærkt validere alle data eller
  B. escape alle brugerleverede input ved hjælp af en escape-rutine, der er specifik for ens databaseleverandør

Stored Procedures

- Stored procedures er ikke altid sikre mod SQL-injektion.
- Visse standard stored procedure programmerings konstruktioner har dog samme effekt som brugen af parametrerede forespørgsler, når de implementeres sikkert. De kræver, at udvikleren blot bygger SQL-sætninger med parametre, der automatisk parametriseres.
- Forskellen mellem prepared statements og stored procedures er, at SQL-koden for en stored procedure defineres og lagres i selve databasen og derefter kaldes fra applikationen.
- Begge disse teknikker har samme effektivitet i at forhindre SQL-injektion, så din organisation skal vælge hvilken tilgang der giver mest mening for dig.

White List Input Validation

- Forskellige dele af SQL-forespørgsler er ikke lovlige steder til brug af bindevariabler, såsom navnene på tabeller eller kolonner, og sorteringsordreindikatoren (ASC eller DESC).
- I sådanne situationer er input validering eller forespørgsel redesign det mest hensigtsmæssige forsvar.
- Ved navne på tabeller eller kolonner kommer disse værdier ideelt fra koden og ikke fra brugerparametre.
  - Hvis brugerparameterværdier bruges til forskel mellem tabelnavne og kolonnens navne, skal parameterværdierne kortlægges til den lovlige / forventede tabel eller kolonnens navne for at sikre, at uvalideret brugerindgang ikke ender i forespørgslen.
  - Bemærk venligst, dette er et symptom på dårlig design, og en fuld omskrivning bør overvejes, hvis tiden tillader det.

White List Input Validation

- Her er et eksempel på tabelnavn validering.

```
String tableName;
switch(PARAM):
  case "Value1": tableName = "fooTable";
                 break;
  case "Value2": tableName = "barTable";
                 break;

    ...
  default      : throw new
InputValidationException("unexpected value provided
for table name");
```

White List Input Validation

- tableName kan derefter tilføjes direkte til SQL-forespørgslen, da det nu er kendt at være en af de lovlige og forventede værdier for et tabelnavn i denne forespørgsel.
  - Husk, at generiske tabelvalideringsfunktioner kan føre til tab af data, da tabelnavne bruges i forespørgsler, hvor de ikke forventes.
- For noget simpelt som en sorteringsrækkefølge ville det være bedst, hvis det brugerleverede input blev konverteret til en boolsk værdi, og så bruges den boolske værdi til at vælge den sikre værdi, der skal tilføjes til forespørgslen.

White List Input Validation

- Den på sidste slide beskrevne arbejdsmåde er et meget udbredt standardbehov ved oprettelse af dynamiske forespørgseler.
- Et eksempel:

```
public String someMethod(boolean sortOrder) {
 String SQLquery = "some SQL ... order by Salary " +
(sortOrder ? "ASC" : "DESC");
  ...
```

- Når bruger input kan konverteres til en ikke-streng, som en dato, numerisk, boolean, taltype osv., inden den tilføjes en forespørgsel, eller bruges til at vælge en værdi, der skal tilføjes forespørgslen, sikres det at det er sikkert at gøre det.
- Input validering anbefales også som et sekundært forsvar i ALLE tilfælde, selv når du bruger bindevariabler som diskuteres senere.

Escaping af alt bruger input

- Denne teknik bør kun bruges som en sidste udvej, når de tidligere nævnte ikke er mulige.
  - Input validering er nok et bedre valg, da denne metode er svag i forhold til andre forsvar, og vi kan ikke garantere at det forhindrer alle SQL-injections indsprøjtning i alle situationer.
- At undslippe bruger input før man sætter det i en forespørgsel er meget databasespecifikt i implementeringen.
  - Det anbefales normalt kun at ændre koden, når implementering af input validering ikke er omkostningseffektiv.
  - Programmer, der er bygget fra bunden, eller programmer, der kræver lav risiko tolerance, skal skrives eller genskrives ved hjælp af parametriske forespørgsler, lagrede procedurer eller en slags Objektrelationel Mapper (ORM), der bygger dine forespørgsler for dig.
- Denne teknik virker som således: Hvert DBMS understøtter en eller flere character escaping schemes, der er specifikke for bestemte typer forespørgsler.
  - Hvis du så escaper alle indtastede indtastninger fra brugeren ved hjælp af det korrekte escape schema for den database, du bruger, vil DBMS ikke forveksle den indtastning med SQL-kode skrevet af udvikleren, hvilket forhindrer eventuelle mulige SQL-injektionssårbarheder.

Escaping af bruger input

- Denne teknik bør kun bruges som en sidste udvej, når de tidligere nævnte ikke er mulige.
  - Input validering er nok et bedre valg, da denne metode er svag i forhold til andre forsvar, og vi kan ikke garantere at det forhindrer alle SQL-injections indsprøjtning i alle situationer.
- At undslippe bruger input før man sætter det i en forespørgsel er meget databasespecifikt i implementeringen.
  - Det anbefales normalt kun at ændre koden, når implementering af input validering ikke er omkostningseffektiv.
  - Programmer, der er bygget fra bunden, eller programmer, der kræver lav risiko tolerance, skal skrives eller genskrives ved hjælp af parametriske forespørgsler, lagrede procedurer eller en slags Objektrelationel Mapper (ORM), der bygger dine forespørgsler for dig.
- Hvert DBMS understøtter en eller flere character escaping schemes, der er specifikke for bestemte typer forespørgsler.
  - Hvis du så escaper alle indtastede indtastninger fra brugeren ved hjælp af det korrekte escape schema for den database, du bruger, vil DBMS ikke forveksle den indtastning med SQL-kode skrevet af udvikleren, hvilket forhindrer eventuelle mulige SQL-injektionssårbarheder.

Escaping af bruger input

- PHP g MySQL eksempel

```
<?php
// Connect
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_
password')
    OR die(mysql_error());

// Query
$query = sprintf("SELECT * FROM users WHERE user='%s' AND
 password='%s'",
             mysql_real_escape_string($user),
             mysql_real_escape_string($password));
?>
```

- mysql_real_escape_string () kalder MySQLs biblioteksfunktion mysql_real_escape_string, som forudstiller backslashes til følgende tegn: \x00, \n, \r, \, ',' og \x1a.

Yderligere forsvar

- Udover at benytte mindst et af de fire primære forsvar anbefales også at benytte begge disse yderligere forsvar.
- Disse yderligere forsvar er:

- Mindste privilegium
- Validering af white list input
  - Ja, igen, men på en lidt anden måde

Mindste privilegie

- For at minimere den potentielle skade ved et vellykket SQL-injection angreb, bør man minimere de rettigheder, der er tildelt hver databasekonto i ens miljø.
- Tildel *ikke* DBA eller administrator type adgangsrettigheder til ens applikations-konti.
  - Alt fungerer bare, når man gør det på denne måde, men det er meget farligt.
- Start fra bunden for at afgøre, hvilke adgangsrettigheder dine applikations-konti kræver, i stedet for at finde ud af hvilke adgangsrettigheder du skal fjerne.
- Sørg for, at konti, der kun skal have læse adgang, kun får læse adgang til de tabeller, de skal bruge adgang til.
- Hvis en konto kun behøver adgang til dele af en tabel, skal du overveje at oprette et VIEW, der begrænser adgangen til den del af dataene og tildeler kontoadgang til visningen i stedet for den underliggende tabel.
- Sjældent, hvis nogensinde, tildeler man oprette eller slette adgang til databasekonti.

Mindste privilegie

- Hvis man vedtager en politik, hvor man bruger stored procedures overalt, og ikke tillader, at applikationskonti direkte udfører deres egne forespørgsler, skal man begrænse disse konti til kun at kunne udføre de stored procedures, de har brug for. Giv dem ikke rettigheder direkte til tabellerne i databasen.
- SQL-injections er dog ikke den eneste trussel mod database data.
  - Angriberne kan simpelthen ændre parameterværdierne fra en af de lovlige værdier, de bliver præsenteret for, til en værdi, der er uautoriseret for dem, men selve applikationen kan få adgang til.
  - Som sådan reduceres sandsynligheden for sådanne uautoriserede adgangsforsøg ved at minimere privilegierne til ens applikation, selvom en angriber ikke forsøger at bruge SQL-injektion som en del af deres exploit.

Mindste privilegie

- Når man er i gang bør man minimere privilegierne på operativsystem-kontoen, som DBMS kører under.
- Kør ikke dit DBMS som root eller system!
- De fleste DBMSer kører fint med en meget kraftfuld systemkonto.
  - For eksempel kører MySQL som system på Windows med en standard konto.
  - Skift DBMS OS-konto til noget mere passende, med begrænsede rettigheder.

Mindste privilegie – Flere DB brugere

- Design af webapplikationer bør ikke kun undgå at bruge den samme ejer / admin-konto i webapplikationerne for at oprette forbindelse til databasen.
- Forskellige DB-brugere bør bruges til forskellige webapplikationer.
- Generelt kan hver separat webapplikation, der kræver adgang til databasen, have en udpeget databasebrugerkonto, som web-appen skal bruge til at oprette forbindelse til DBen.
  - På den måde kan applikationsdesigneren have god granularitet i adgangskontrollen og dermed reducere privilegierne så meget som muligt.
  - Hver DB-bruger vil derefter have adgang til, hvad kun denne kræver, og skriveadgang efter behov.
    - For eksempel kræver en login side læsadgang til brugernavn og adgangskodefelter i en tabel, men ingen skriveadgang til en formular (ingen indsætning, opdatering eller sletning).
    - Registreringssiden kræver dog bestemt indsætnings-privilegier til den pågældende tabel;
  - Denne begrænsning kan kun håndhæves, hvis disse webapps bruger forskellige DB-brugere til at oprette forbindelse til databasen.

Mindste privilegie – VIEWs

- Man kan bruge SQL-VIEWs til yderligere at øge adgangen til granularitet ved at begrænse læsadgang til bestemte felter i en tabel eller joins af tabeller.
- Det kan potentielt have yderligere fordele:
  - For eksempel, antag, at systemet er påkrævet (måske på grund af nogle specifikke juridiske krav) til at gemme brugerens adgangskoder, i stedet for salted-hashed-adgangskoder.
    Designeren kan bruge VIEWs til at kompensere for denne begrænsning;
    Tilbagekald al adgang til tabellen (fra alle DB-brugere undtagen ejer / admin) og opret et VIEW, der udsender hash af adgangskodefeltet og ikke selve feltet
    Ethvert SQL-injektions-angreb, der lykkes i at stjæle DB-information, vil være begrænset til at stjæle kodeordets hash (kan endda være en indtastet hash), da ingen DB-bruger til nogen af webapplikationerne har adgang til selve tabellen.

Validering

- Ud over at være et primært forsvar, når intet andet er muligt (f.eks. Når en bindingsvariabel ikke er lovlig), kan input validering også være et sekundært forsvar, der bruges til at registrere uautoriseret input, før det sendes til SQL-forespørgslen.

C# og databaser

Især om LINQ og stored procedures

LINQ

- Akronymet LINQ star for Language INtegrated Query.
- Gennem LINQ queries får man let data adgang til in-memory objekter, databaser, XML documenter og mere til.
- LINQ blev indført i Visual Studio 2008 og er designet af Anders Hejlsberg.
- LINQ tillader en at skrive queries selv uden kendskab til query sprog som SQL, XML osv.
- LINQ queries kan skrives til forskellige datatyper.

```
1  using System;
2  using System.Linq;
3
   0 references
4  class Program
5  {
       0 references
6      static void Main()
7      {
8          string[] words = { "hello", "wonderful", "LINQ", "beautiful", "world" };
9          //Get only short words
10         var shortWords = from word in words
11                          where word.Length <= 5
12                          select word;
13
14         //Print each word out
15         foreach (var word in shortWords)
16         {
17             Console.WriteLine(word);
18         }
19         Console.ReadLine();
20     }
21 }
```

LINQ - Syntaks

Der er to syntakser I LINQ.

- Lamda (Method) Syntax

```
var longWords = words.Where( w => w.length > 10);
```

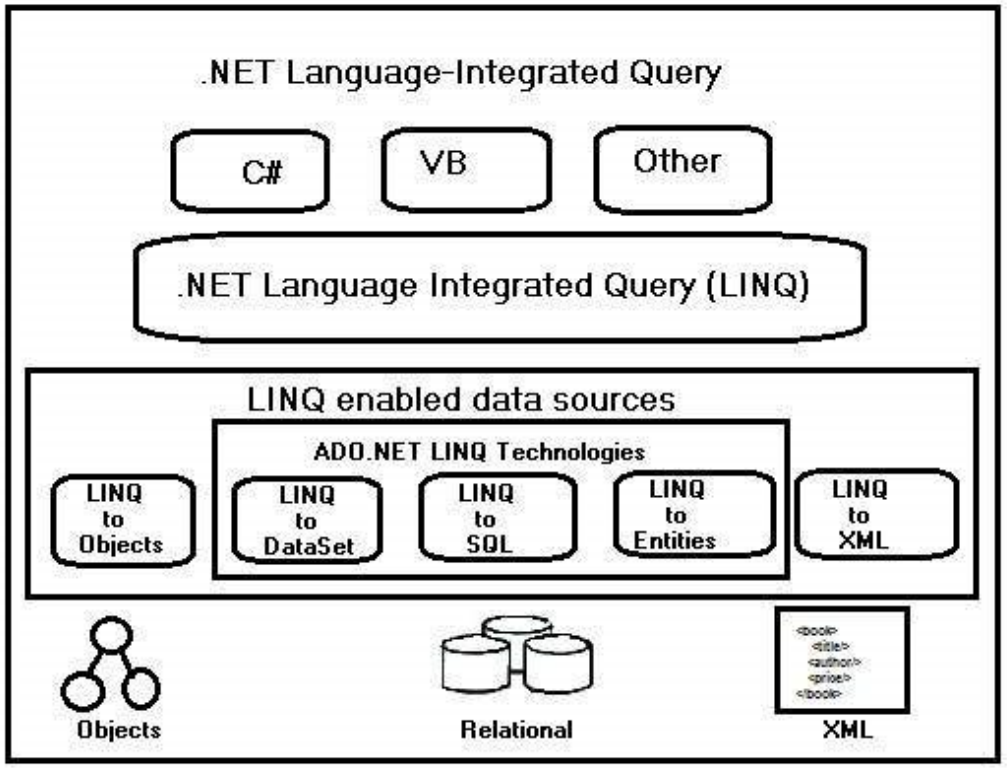
- Query (Comprehension) Syntax

```
var longwords = from w in words where w.length > 10;
```

- Query udtryk er ikke andet end en LINQ forespørgsel, udtrykt i en form, der svarer til SQL med query operatører som Select, Where og OrderBy.
- Query udtryk starter som regel med nøgleordet ”From".

LINQ – I .net

- LINQ har en 3-laget arkitektur i hvilken det øverste lag består af sprog (language) extensions og det nederste lag består af data kilder, der typisk er objekter der implementerer IEnumerable<T> eller IQueryable<T> generiske interfaces.



LINQ query

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5
6   namespace Operators
7   {
        0 references
8       class LINQQueryExpressions
9       {
            0 references
10          static void Main()
11          {
12              // Specify the data source.
13              int[] scores = new int[] { 97, 92, 81, 60 };
14
15              // Define the query expression.
16              IEnumerable<int> scoreQuery = from score in scores
17                                            where score > 80
18                                            select score;
19
20              // Execute the query.
21              foreach (int i in scoreQuery)
22              {
23                  Console.Write(i + " ");
24              }
25              Console.ReadKey();
26          }
27      }
28  }
```

LINQ vs stored procedures

Der er en række forskelle på LINQ og Stored procedures.

- Stored procedures er meget hurtigere end en LINQ forespørgsel da de følger en forventet udførelses plan.
- Det er nemmere at undgå run-time fejl under udførelse af en LINQ query i forhold til en stored procedure.
  - LINQ har Visual Studio Intellisense support samt full-type checking ved kompilerings-tid.
- LINQ tillader debugging igennem .NET debugger hvilket ikke er tilfældet for stored procedures.

LINQ vs stored procedures

- LINQ tilbyder understøttelse af flere databaser i modsætning til stored procedures, hvor det er vigtigt at omskrive koden til forskellige typer af databaser.
- Implementering af LINQ baserede løsninger er nemt og enkelt i forhold til indsættelsen af en række af stored procedures.

LINQ
uden og med

Førhen uden LINQ

```
SqlConnection sqlConnection = new SqlConnection(connectString);
SqlConnection.Open();
System.Data.SqlClient.SqlCommand sqlCommand = new SqlCommand();
sqlCommand.Connection = sqlConnection;
sqlCommand.CommandText = "Select * from Customer";
return sqlCommand.ExecuteReader (CommandBehavior.CloseConnection)
```

Med LINQ

```
Northwind db = new Northwind(@"C:\Data\Northwnd.mdf");
var query = from c in db.Customers
            select c;
```

C++ og databaser

Kort eksempel, del af et større program

Minder en del om C#

```
#include <cstdlib>
#include <sstream>
#include <iostream>
#include <iomanip>
…
BankTransaction::BankTransaction(const string
HOST,
const string USER, const string PASSWORD,
const string DATABASE)
{
```

Minder en del om C#

```
db_conn = mysql_init(NULL);
if(!db_conn)
message("MySQL initialization failed! ");
db_conn = mysql_real_connect(db_conn,
HOST.c_str(),
USER.c_str(), PASSWORD.c_str(),
DATABASE.c_str(), 0,
NULL, 0);
if(!db_conn)
message("Connection Error! ");
}
```

Java og databaser

Kort eksempel

Java er også et C sprog

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT Lname
FROM Customers WHERE Snum = 2001");
```

Opsummering af alle kodeeksemplerne

- Uanset sprog bruges SQL eller en SQL-lignende syntaks (LINQ) til at kommunikere med databasen.
- Da dette kursus er om databaser og ikke programmering blev der valgt ikke at gå i dybden med dette område.

Anbefalet læsning

Nu det er sidste officielle undervisningsgang er det jo ikke lektier

Anbefalet læsning

- http://www.c-sharpcorner.com/UploadFile/84c85b/understanding-transactions-in-sql-server/
- https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
- https://www.cisecurity.org/wp-content/uploads/2017/05/SQL-Injection-White-Paper.pdf

Andre fag fremover

Vi gennemgår studieordningen og snakker om (oplagte) muligheder at bygge videre med – især hvis målet er en AU i Informationsteknologi

Eksamen og krav

Tid til ekstra spørgsmål til krav og eksamensform

Eksamensspørgsmål

1. Beskriv hvad en relationel database er og perspektiver til andre former for dataopbevaring
2. Beskriv hvad CRUD er og kom med eksempler på SQL kode
3. Beskriv og illustrer hvad et ER diagram er
4. Beskriv hvad normalformer er og gennemgå de første tre
5. Beskriv hvad et JOIN er, herunder skal nævnes forskellige former, kom med eksempler på SQL kode

Eksamensspørgsmål

6. Beskriv matematikken bag databaser (relationel algebra og relationel kalkulus) og perspektiver det til SQL kommandoer
7. Beskriv konceptet bag database nøgler (keys), kom herunder ind på primary og foreign keys
8. Beskriv hvad en transaktion er og kom med eksempler på SQL kode
9. Beskriv hvad et VIEW er og kom med eksempler på SQL kode
10. Beskriv hvad en SQL injektion er og beskriv metoder til at beskytte sig mod det

Kilder

Materiale benyttet i denne undervisningsgang

Kilder

Dagens database

- https://dev.mysql.com/doc/sakila/en/

Transaktioner

- http://www.c-sharpcorner.com/UploadFile/84c85b/understanding-transactions-in-sql-server/
- https://www.universalclass.com/articles/computers/sql/what-are-transactions-in-sql.htm
- https://youtu.be/d7-tJUnXOmc

Kilder

Views

- https://www.tutorialspoint.com/sql/sql-using-views.htm
- http://beginner-sql-tutorial.com/sql-views.htm
- https://www.w3resource.com/sql/creating-views/creating-view.php
- https://www.w3schools.com/sql/sql_view.asp
- https://docs.oracle.com/database/121/SQLRF/functions003.htm#SQLRF20035
- http://www.dummies.com/programming/databases/sql-set-functions-2/

Kilder

Views

- https://community.teradata.com/t5/Database/SQL-to-Create-a-Table-from-a-View-Retrieve-the-DDL/m-p/15285
- https://stackoverflow.com/questions/6694430/create-table-from-view

Sikring af adgang til databaser

- http://searchsqlserver.techtarget.com/feature/Basic-SQL-Server-security-best-practices

Kilder

Sikring af adgang til databaser

- https://www.sqlshack.com/top-10-security-considerations-sql-server-instances/
- http://searchsqlserver.techtarget.com/tip/Microsoft-SQL-Server-security-best-practices-checklist

Sikring af input til databaser

- https://xkcd.com/327/
- https://www.explainxkcd.com/wiki/index.php/Little_Bobby_Tables

Kilder

Sikring af input til databaser

- https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
- https://www.cisecurity.org/wp-content/uploads/2017/05/SQL-Injection-White-Paper.pdf
- http://php.net/manual/en/function.mysql-real-escape-string.php

Kilder

C# og databaser

- https://www.codeproject.com/Articles/26657/Simple-LINQ-to-SQL-in-C
- https://www.dotnetperls.com/datagridview-tutorial
- https://www.youtube.com/watch?v=GVV-LUcmCOE
- https://www.youtube.com/watch?v=L1m1Znj9dZA
- https://www.youtube.com/watch?v=p2UeT7dBTEg
- https://www.youtube.com/watch?v=_PCBTiXL884

Kilder

C++ og databaser

- https://www.codeguru.com/cpp/data/database-programming-with-cc.html

Java og databaser

- https://alvinalexander.com/java/edu/pj/jdbc/jdbc0003