# LærerVikaren 03

|  |  |
| --- | --- |
| **Læringsmål** | Du kan:   * **1Pf4:** anvende centrale metoder og teknikker til at realisere modeller i et databasesystem […] |
| **Forventet læringsudbytte [SOLO]** | Programmeringssporet:   * [Multistrukturel] Du kan beskrive flere kendetegn ved **database-tilgang via C#/.NET** |
| **Din forberedelse** | Programmeringssporet:   * Genopfrisk efter behov C#-klasserne:   + SqlConnection:     - Connection string     - Open(), Close()   + SqlCommand:     - Parameters     - ExecuteReader() , ExecuteScalar(), ExecuteNonQuery()   + SqlDataReader:     - Read() * Null-værdier:   + [Nullable value types](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/nullable-value-types)   + [C# - Nullable Types](https://www.tutorialsteacher.com/csharp/csharp-nullable-types)   + Håndtering af DBNulls i C# (se dokument i planer i ItsLearning) * Genopfrisk evt. Unit-tests   + F.eks. genlæs: [A Basic Introduction To C# Unit Test For Beginners](https://www.c-sharpcorner.com/article/a-basic-introduction-of-unit-test-for-beginners/) |

Denne opgave er den næste i en serie af opgaver, hvor du skal arbejde videre med database­temaet primært med udgangspunkt i LærerVikar-casen, som du er introduceret til i systemudvikling.

Løs den forrige opgave Ex46-LærerVikaren02 for en udvalgt iteration (tabel i UML-databasemodellen), før du går i gang med denne opgave.

Denne opgave har fokus på den fase C: **database-tilgang via C# og .NET**:

* Fase A: Database-design (fra domænemodel til UML-databasemodel via normaliserede relationsskemaer)
* Fase B: Database-implementering (fra UML-databasemodel til konkret database i et DBMS via SQL DDL)
* **Fase C: Database-tilgang via C# og .NET**

Dagens ord

“The difference between good and bad architecture is the time you spend on it”  
David Chipperfield

# Øvelse 1: Terminologi

Del teamet op i to mindre grupper, og brug Ordet rundt til at reflektere over begreberne ”SqlConnection”, ”SqlCommand”, ”SqlDataReader”, ”Nullable typer”, ”DBNulls” og ”unit test”.  
Sørg for, at alle får mulighed for at tale.

*Tidsramme: 20 minutter*

# Øvelse 2: Fase B: Database-implementering (SQL)

Hvis du ikke blev færdig med at etablere en SQL-database i Microsoft SQL Server i forrige opgave Ex46-LærerVikaren02 (iterativt), da sørg for at gøre dette nu. Du skulle gerne have realiseret en konkret SQL-database for LærerVikaren i Microsoft SQL Server, før du starter; gerne fyldt med testdata.

**Benyt parprogrammering i alle følgende øvelser.**

# Øvelse 3: Fase C: Database-tilgang via C#/.NET

Du skal nu til at implementere en applikation til LærerVikaren, hvor du sikrer persistens af data ved at bruge den database, du har designet og realiseret på Microsoft SQL Server i den forrige opgave Ex46.

## Øvelse 3.1: Klargøring

I første omgang er det ikke så vigtigt, om din implementering af LærerVikaren er en konsolapplikation eller en WPF-applikation. Du skal nemlig udelukkende fokusere på Model-laget og Persistens-laget i denne opgave uden at overveje UI-laget eller Applikationslaget (ViewModel/Controller).

Udfør følgende:

* Opret en Visual Studio løsning med navnet ’LærerVikaren’. Vælg enten en WPF- eller konsol-projekttype.
* Opret et .NET Class Library med navnet ’ModelPersistence’, et bibliotek hvori du implementerer både dit Persistens-lag og Model-lag. Man behøver ikke at vælge et særskilt projekt til disse to lag, men du gør det i denne omgang for at tydeliggøre lagdelingen i løsningen.
* I ModelPersistence-biblioteket opretter du nu en mappe med navnet ’Models’ til alle domæneklasser i Model-laget og en mappe ’Persistence’ til alle Repository-klasser i Persistens-laget.

Du er nu klar til at komme i gang.

## Øvelse 3.2: Implementér Model-laget og Persistens-laget

**Vigtigt!** Der er jo en del klasser i domænemodellen for LærerVikaren og dermed også i DCD’et. Igen med den iterative tilgang i baghovedet, så hav fokus på at implementere de domæneklasser i DCD’et, som understøtter de use cases, du udvalgte i øvelse 4.1 i Ex45-LærerVikaren01.  
Fokusér gerne på en domæneklasse ad gangen i denne øvelse og i næste øvelse 3.3.

Med udgangspunkt i dit DCD og udvalgte use cases, udfør følgende i ModelPersistence-biblioteket:

* Implementér dine domæneklasser i Models-folderen.
* Implementér derefter dine Repository-klasser i Persistence-folderen (en for hver domæneklasse), så (vigtigt!) hver Repository-klasse vedligeholder **både** en repræsentation af data i din SQL-database (i en tabel) **og** en in-memory repræsentation af dine objekter (i en liste) for domæneklassen. Du er også velkommen til at definere et IRepository-interface, som hver Repository-klasse overholder, men det er frivilligt.  
  Sørg for at implementere alle CRUD-operationer (Add, GetAll, GetById, Update og Remove) i hver Repository-klasse med brug af SQL-sætninger. Vær opmærksom på at skulle håndtere tomme celler i en tabel (DBNulls) samt evt. typekonverteringer.

Denne øvelse kommer sikkert til at tage lidt tid, men hæng i. **Igen tag en domæneklasse ad gangen.**

Du skulle nu gerne være klar til at afprøve, om alt nu virker, som det skal. Men i stedet for at begynde at implementere en Controller-klasse eller ViewModel-klasse og efterfølgende en brugergrænseflade, så fokusér udelukkende på at lave unit-tests til dit Model-lag og Persistence-lag, så du også får repeteret unit-tests.

## Øvelse 3.3: Test af LærerVikaren

Klargøring:

* Opret et nyt testprojekt i løsningen med navnet ’LærerVikarUnitTest’. Benyt ’MSTest Test Project’-skabelonen.

Du skal nu i testprojektet oprette en testklasse for hver Repository-klasse, du har implementeret. For hver testklasse skal du teste både SQL- og in-memory-implementeringen for alle CRUD-metoder i den tilsvarende Repository-klasse, dvs. teste metoderne Add(), GetAll(), GetById(), Update() og Remove(), hvis det altså er det, du kalder dem. Find selv på passende data i dine tests.

Udfør følgende for hver Repository-klassen:

* Opret en testklasse i testprojektet (der er allerede oprettet et på forhånd, da du oprettede testprojektet, den næste skal du selv oprette). Navngiv testklassen, så det er tydeligt at se, hvilken Repository-klasse den tester.
* Benyt AAA-mønstret i hver testmetode i klassen, dvs. Arrange-Act-Assert. F.eks. opret et objekt i Repository-klassen med Add(). Hent objektet med GetById(), og tjek om det returnerede objekt har samme informationer, som det tilføjede. Slet dernæst objektet med Remove() og prøv at hente det igen med GetById(); det skulle gerne fejle.
* Formulér jeres testmetode, så du tester det, som du forventer vil ske.

Det var det hele for denne gang.

# Øvelse 4: Iterationer

Er du blevet færdig med den udvalgte iteration, da udfør ovenstående ’fase C’-øvelser for en ny domæne­klasse i DCD’et (evt. også for fase A og B, hvis der mangler domæneklasser). Gentag dette, så længe du har tid i dag.

Du har nu afsluttet fase C for LærerVikaren og dermed hele databaseprocessen.

Den kommende uges opgaver supplerer med yderligere emner, du kan inddrage i dine tidligere database-løsninger. Hvis du ikke er helt færdig med denne opgave, så vil der være tid til det i næste uge.