Accessing data from program code

Project assignment

Learning outcomes					ALL
LO1	LO2	LO3	LO4	LO5	
20	20	20	20	20	100

INSTRUCTIONS

- The defense of the project takes place during the examination periods
- The student applies for the project the same way as for the written exams (through Infoeduka exams)
- All solutions should be submitted through GitHub version control service **3 days before the project defense**

LEARNING OUTCOMES:

LO1 (20 points):

- *minimum (10 points):* Create a software solution using a relational database on the cloud as a data source
- *desired (10 points)*: Create a relational database in the cloud and create a software solution using that database as a data source

LO2 (20 points):

- *minimum (10 points):* Create a software solution using solutions for storing unstructured data in the cloud as a data source
- *desired (10 points)*: Create an unstructured data storage in the cloud and create a software solution using that unstructured data storage as a data source

LO3 (20 points):

- *minimum (10 points):* Create a software solution using a non-relational cloud database as a data source
- *desired (10 points)*: Create a non-relational database in the cloud and create a software solution using that non-relational database as a data source

LO4 (20 points):

- *minimum (10 points):* Select the optimal conceptual data model and implement it
- *desired (10 points)*: Select the optimal complex conceptual data model and implement it

LO5 (20 points):

- *minimum (10 points):* Implement a software solution using selected ORM tools
- *desired (10 points)*: Implement a complex software solution using selected ORM tools

First Project - Apolon (LO1 - 20 points, LO2 – 10 points, LO4 - 20 points, LO5 - 20 points)

Your task is to write custom implementation of **Object Relational Mapper** (**ORM**) using any language that you prefer (C# is recommendation as most of the examples in the exercises are shown in that language). You can use code snippets and examples from the exercises and EF Core features as an inspiration.

Features that your implementation must support are:

- Mapping classes to database tables – basic concept of accessing data from rows by reading them as instances of entity classes
- Support datatype mapping for at least the following data types:
    - `VARCHAR`
    - `INT`
    - `DECIMAL`
    - `FLOAT`
    - `DATETIME`
- Column constraints:
    - `NULL / NOT NULL`
    - `DEFAULT`
    - `UNIQUE`
    - Auto generation of the primary key
- Navigational properties for related tables (1-many, many-1, 1-1)
- Well defined management of raw database connection
- Basic CRUD using the structure of your choice
    - Retrieving the data must support filtering and ordering (grouping is not required)
- Migrations (creation, execution and rollback)

Demonstrate the required functionalities by implementing a simple web or console application that uses your ORM implementation for CRUD operations on relational database schema. The application should support the **relational data model** which captures the scenario of a medical system responsible for managing **patients**, their **medical records**, **checkups** and **prescriptions**. Application must allow CRUD operations on patients, checkups, medications. Checkups can be of a certain type (GP, BLOOD, X-RAY, CT, MRI, ULTRA, EKG, ECHO, EYE, DERM, DENTA, MAMMO, EEG) and prescriptions detail which dosage of medication patient is taking (with start and optional end date).

All features must be demonstrated and shown on the project defense. You can also expect the questions about inner workings of Postgres that will be covered in the exercises.

The following features need to be demonstrated on the project defense. As mentioned before, you can use either simple console or web application to showcase how your ORM implementation works in realistic use-case scenario. You can also expect the questions related to understanding how underlying mechanisms work for the different data storages that you used in the project and that we covered in the exercises.

Learning outcome 1 (Minimum - 10 points) Create a database with required entities and relations located in a cloud Postgres database provisioned by Supabase service. It is mandatory to explain how the database was created as well as how to connect to a given instance. Using your own ORM implementation, implement a functionality to generate executable DDL queries based on the database schema.

Learning outcome 1 (Desired - 10 points) Implement support for CRUD operations on required related entities, using any approach that you like. It is necessary to additionally demonstrate the following:

- a detailed explanation of the communication between the application and the database
- understanding the Postgres architecture

Learning outcome 4 (Minimum - 10 points) Create the conceptual data model by introducing the required entities connected by various relationships, following the definition of 3rd Normal form. Ensure data integrity by implementing support for database constraints.

Learning outcome 4 (Desired - 10 points) Implement the retrieval of related data using navigational properties for different cardinalities (1-many, many-1, 1-1).

Learning outcome 5 (Minimum - 10 points) Implement the database connection management strategy as well as transaction management (Unit of Work pattern). Demonstrate these features in your application.

Learning outcome 5 (Desired - 10 points) Implement your own migration functionality. Ideally, you will implement separate console applications that can create new migrations, execute existing migrations (with tracking) and roll back to the previous migration.

Second project – Dionis (LO2 – 20 points, LO3 – 20 points)

Your task is to implement a data processing pipeline for bird sightings consisting of four steps. Main goal of this pipeline is to generate a dataset that contains information about birds and their sightings based on the

- audio files containing possible bird songs or calls
- data consumed from external ornithology services.

For this solution, it is mandatory to use **MongoDB** and **Minio** (or any other S3 compatible data storage) as storage providers. However, if you require any other external tool or library, you are permitted to use it.

First objective is to scrape all the data about bird species (lat. *aves*) and create a backbone by storing the scraped data into a MongoDB collection. Data is available on the https://aves.regoch.net (mock website that uses publicly available GBIF data). If the data already exists in the collection, this step needs to be skipped.

Secondly, it is necessary to read all messages present on the Kafka publisher at the time of the execution. Kafka messages contain information about bird sightings published by ornithologists, called **observations**. These messages do not contain any audio files. Rather, they must contain the bird taxonomy identification code and geographical position (longitude and latitude), but the present biological observation data can vary between different ornithological sources (e.g. body size, body temperature, migration status, flight pattern, habitat, etc.). All observations need to be stored in your database.

Third step of the pipeline should process audio files in a target directory which means uploading each file to MiniO (or any other cloud S3 compatible) bucket storage and subsequently requesting classification from the publicly available classification model () via the API interface. The classification response will provide information about birds present in the audio recording as well as confidence score. The audio file and classification results should be stored in the database for later retrieval.
The target directory containing audio files can be a local directory, but you can also optionally support cloud-based storage (e.g. Google Drive). Each file must be associated with a geographical position, similarly to the previous step (for the sake of simplicity, you can assume that all files in a certain folder are associated with a single geographical position).

The final step should generate statistics for all bird species that have at least one positive classification. Before creating the final output, make sure to clean the data and apply the appropriate transformations. Produced CSV should include species name, number of classified sightings, and any relevant observational data. When executing this pipeline, optional parameter can be set to provide a fuzzy filter for species names to narrow the report to specific birds.

To achieve maximum points, the workflow described should be split into multiple smaller scripts orchestrated with any tool that allows single entry point execution with the possibility of defining optional runtime parameters (since Python is a recommended language for this task, **Snakemake** is a recommended script execution orchestrator). For bonus points, enable the script execution through manually triggerable **Github Actions** workflow (as shown on the exercises) as well as visualization of the generated report, in any way that you prefer[1].

[1] You can use Python plotting libraries like Matplotlib, Seaborn or Plotly as well as Charts.js, Grafana or any other language library that supports visualizations

The following section details the points for each successfully implemented segment of the pipeline. The order is scrambled to accommodate the definitions and order of learning outcomes. For the actual step order, please refer to the description above.

Learning Outcome 2 (Minimal - 10 points): Implement the processing of files located in a target directory by uploading them to MinIO (or another S3-compatible storage) and ensuring that each file can be retrieved and uniquely identified. The uploaded files should be associated with metadata, such as location and filename, and this association must be stored in MongoDB. Optionally, you can add support for processing files found in other remote locations (e.g. Google Drive).

Learning Outcome 2 (Desired - 10 points): For each uploaded file, request an API call to the bird classifier model (deployed on:), storing request log in the MiniO (define any format that you see fit), while saving the classification results the appropriate MongoDB collection.

Learning Outcome 3 (Minimal - 10 points): Store the scraped bird species data from https://aves.regoch.net in the appropriate MongoDB collection while avoiding duplicate entries. After the classification process has finished in the third step, store the classification results in the appropriate collection while linking the bird observations to species information.

Learning outcome 3 (Desired – 5 points): Consume Kafka messages containing bird observations and store them in the MongoDB collection including species ID, location, and any provided biological observation data. Be careful, as different observations can contain different biological properties.

Learning outcome 3 (Desired – 5 points): Implement filtering (fuzzy sting matching) based on the species names for generating the final CSV report and apply appropriate data cleaning and transformations method before producing the final CSV output.