Instituut voor
Communicatie,
Media & Informatietechnologie

Naam: ………..……………………………..

Studentnr: ……………………………….

Klas: …………………..……………………

General information

Course code	:	INFDTB01-D, INFDTB21-D
Course name	:	Development Database
Full-time / Part-time	:	Full-time and Part-time (both)
Type of exam	:	Resit
Exam form	:	VS Code, Visual Studio, CodeGrade, GitHub
Period	:	2
Classes	:	INF2A-G, DINF2, Retakers
Duration in minutes	:	120 Minutes, (BO 160 Minutes)
Course holder	:	Saleem Anwar
Author	:	A. IBBA
Second reader	:	S. Anwar

This exam consists of:
☐ A cover page with …12… numbered pages
☐ Multiple choice
☐ Case
☒ Open question

Allowed tools:
☐ Non-graphic calculator
☐ Scrap paper
☐ Dictionary, type ……………………………………………………………………………
☐ Cheat sheet
☒ Other, namely …Laptop (with visual studio/ (VS Code), .Net, Docker, Browser (CodeGrade, GitHub, GitLab, pgadmin web-pages only)) …………

Write your answers and calculations:
☐ on the examination paper handed out, so not on the exam
☒ Submit the solution file(s) (Solution.cs, in CodeGrade and Teams assignment)
☐ on the enclosed answer form

Type of draft paper for elaboration:
☐ lined paper
☐ squared paper

Finally:
You are required to submit exam solution file(s) (Solution.cs) in CodeGrade and in the MSTeams assignment under invigilator/technical teacher presence. Good luck!

Setup Exam INFDTB

Steps that can be done BEFORE the exam

Running Docker-Compose:

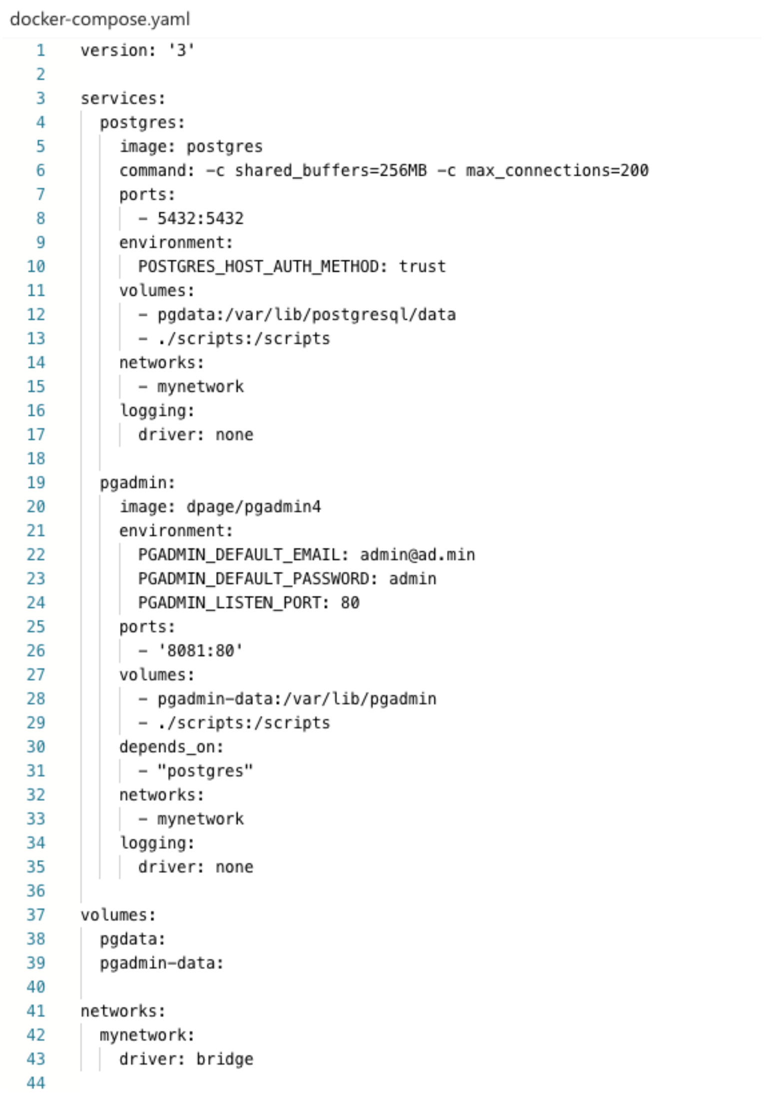
https://docs.docker.com/compose/gettingstarted/
You need to have Docker Engine and Docker Compose on your machine. You can either:

- Install Docker Engine and Docker Compose as standalone binaries

OR

- Install Docker Desktop which includes both Docker Engine and Docker Compose

Define services in a Compose file by Creating a file called docker-compose.yaml in a directory (for instance your project directory) and use the following (this file has been provided previously and can also be found in the Exam assignment):


docker-compose.yaml

```yaml
version: '3'

services:
  postgres:
    image: postgres
    command: -c shared_buffers=256MB -c max_connections=200
    ports:
      - 5432:5432
    environment:
      POSTGRES_HOST_AUTH_METHOD: trust
    volumes:
      - pgdata:/var/lib/postgresql/data
      - ./scripts:/scripts
    networks:
      - mynetwork
    logging:
      driver: none

  pgadmin:
    image: dpage/pgadmin4
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@ad.min
      PGADMIN_DEFAULT_PASSWORD: admin
      PGADMIN_LISTEN_PORT: 80
    ports:
      - '8081:80'
    volumes:
      - pgadmin-data:/var/lib/pgadmin
      - ./scripts:/scripts
    depends_on:
      - "postgres"
    networks:
      - mynetwork
    logging:
      driver: none

volumes:
  pgdata:
  pgadmin-data:

networks:
  mynetwork:
    driver: bridge

```

Build and run your app with Compose from the chosen directory, start up your containers by running : `docker compose up -d`

You may use **MS Visual Studio**, or **VS Code**

It is advisable to install and use **.Net 7**

It is advisable to create and clone on your machine git repository (if you prefer submission via git)

A project can be created using the following these steps:

Creating app:

Visual Studio

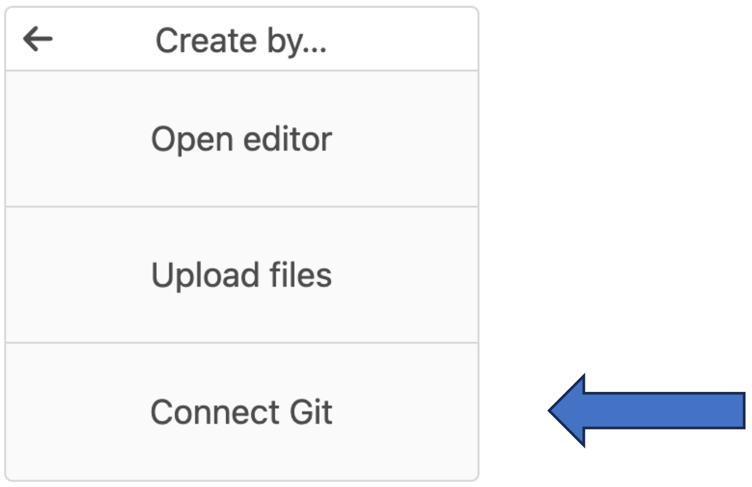
a. Create a console app with logical name (say DBExam)
File -> New -> Project ->Console App (c#) -> next [ProjectName]-> Finish
b. Add the following NuGet Packages (right click in solution explorer and goto Manage
NuGet Packages OR Tools-> NuGet Package Manager-> Manage NuGet Package)
`Microsoft.EntityFrameworkCore`
`Microsoft.EntityFrameworkCore.Tools`
`Microsoft.EntityFrameworkCore.Design`
`Npgsql.EntityFrameworkCore.PostgreSQL`

Visual Studio Code / CLI

a. Create a console app with logical name (say DBExam)
`dotnet new console -o DBExam`
`cd DBExam`
b. Install dotnet-ef (only needed once):
`dotnet tool install --global dotnet-ef`
c. Add the following NuGet Packages
these steps are NOT needed when using the provided csproj file
`dotnet add package Microsoft.EntityFrameworkCore`
`dotnet add package Microsoft.EntityFrameworkCore.Tools`
`dotnet add package Microsoft.EntityFrameworkCore.Design`
`dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL`
c. Open project directory in VS Code (you may add packages via terminal in vs code)

DURING the Exam

1. In CodeGrade connect the git repositories to the exam Question section.



2. Download and add to the respective sections (Linq Questions) the files available in Teams under assignment
   2324 Retake Exam Database.

3. Content of the provided csproj files should replace the respective ones in the respective project.

4. **Run the migrations** (Tools/NuGet Package Manager/PMC ▶ CLI Terminal and make sure Database created by looking into PgAdmin.

Visual Studio

```
Add-Migration migrationName
Update-Database
```

Visual Studio Code / CLI

```
dotnet ef migrations add migrationName
dotnet ef database update
```

Answers can be provided by changing the relevant methods (ONLY) in the file **Solution.cs** (Linq Questions section).

Submission in CodeGrade can happen an indefinite number of times using one of these three ways: using the online editor, uploading the relevant file (Solution.cs), or with a git push.

Good luck with the exam!

At the END of the Exam:

Before leaving the room submit in CodeGrade **AND** return modified Solution.cs in Teams as a proof of your exam and for evaluation/grade, this will happen under the supervision of invigilator/technical teacher.

General Rules:

1. Answers can be provided by changing the relevant methods in the file **Solution.cs** (Linq Questions section). Submission in CodeGrade can happen an indefinite number of times using one of these three ways: using the online editor, uploading the relevant files (Solution.cs), or with a git push.

2. The following actions:
   Seeding the DB differently than what already setup; using Raw SQL queries; using Include/ThenInclude methods; using variable/method names containing the following words: File, Process, Start, Correct, printing on Console.
   are **NOT** allowed.
   Very IMPORTANT: the following words **cannot** be used (*even not in comments*): **SQL, Include, ThenInclude, Console, Write, WriteLine, File, Process, Start, Correct** (case-insensitive for all characters).

3. Very IMPORTANT: Submit in CodeGrade and return Modified Solution.cs file in Teams as a proof of your exam and for evaluation/grade. No submission, no grade.

4. Academic honesty: we expect academic honesty from all of the students, a student found using illegal means will face inquiry and disciplinary actions.
5. During exam you are allowed to use Visual Studio/ VS code, Docker, Browser (CodeGrade, GitHub, GitLab, pgadmin web-pages) only, and Teams for final submission under supervision of invigilator/technical teacher.
6. The exam consists of 10 points in total, you need to secure 5.5 points to pass the exam.

Retake Exam INFDTB (2023-24)

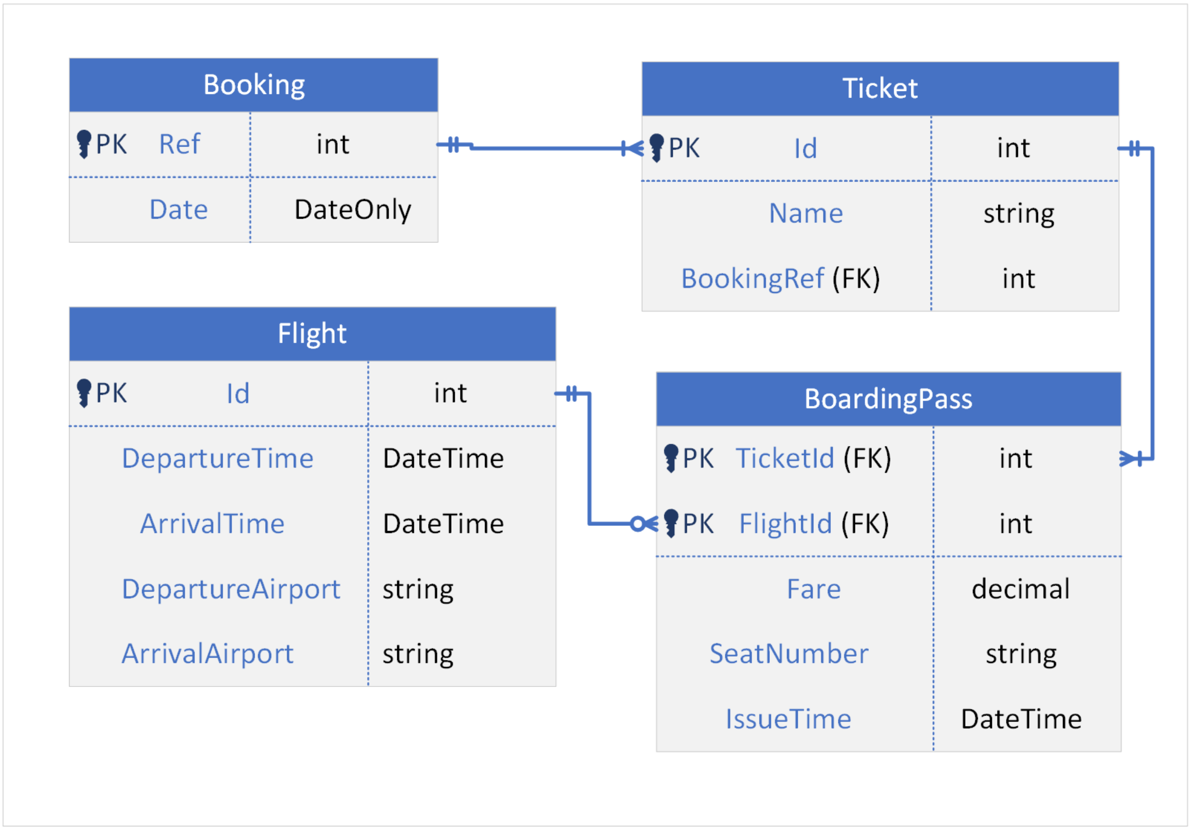
This exam ONLY consists of the **Linq Questions** section

CodeGrade enrollment-link: will be provided during the exam via teams.

The case study: Flight booking system

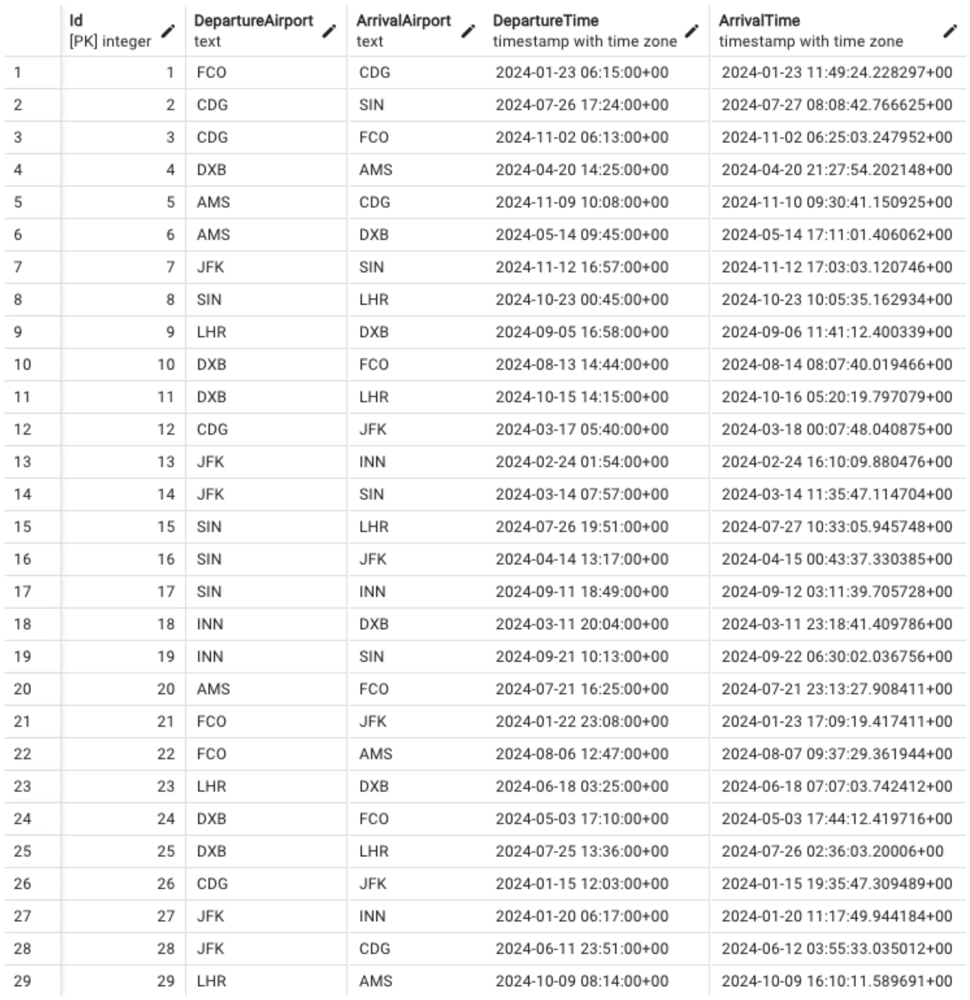
- This case study is about a flight booking system. Following data is recorded:
- The booking system is organised into **Flights**, **Bookings**, **Tickets**, **BoardingPasses**.
- The Flights have these attributes: an Id, a departure and arrival airports, departure and arrival times.
- A Booking with as attributes a reference and a date (Ref, Date) will (refer) be linked to a *number* of Tickets (attributes: Id, Name, BookingRef), each Ticket (contains or) can be associated to a *number* of Flights (through the entity BoardingPass) and can be referred to only one passenger ("Name" attribute of Ticket).
- Notice a Many To Many (optional) relationship exists between Tickets and Flights: a Ticket contains at least one or many Flights, a Flight might be present in one/many Ticket(s), or none. The mentioned Many To Many (optional) relationship is resolved through the entity BoardingPass.
- The BoardingPasses (attributes: FlightID, TicketID, Fare, SeatNumber, IssueTime) will be issued for unique { FlightID, TicketID } combinations: a specific Ticket issued for a passenger travelling on a specific Flight.
- For each Flight BoardingPasses might be issued therefore reserving a SeatNumber per BoardingPass. For some Flights there might be no BoardingPasses issued.
- A Flight might therefore be linked to many BoardingPasses (or no BoardingPasses at all), a Ticket will be linked to at least one or many BoardingPasses (depending on how many Flights will be present in the itinerary of the Ticket).

ER Diagram:



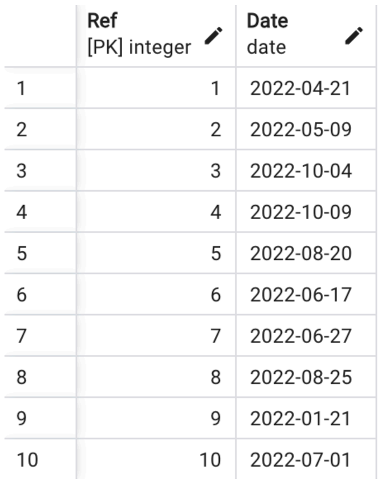
Below you may find sample data for each entity/table stored in database, the snippet also give away information about column names and data types. This is dummy data generated **randomly** so data and result may differ for each student.

Flights



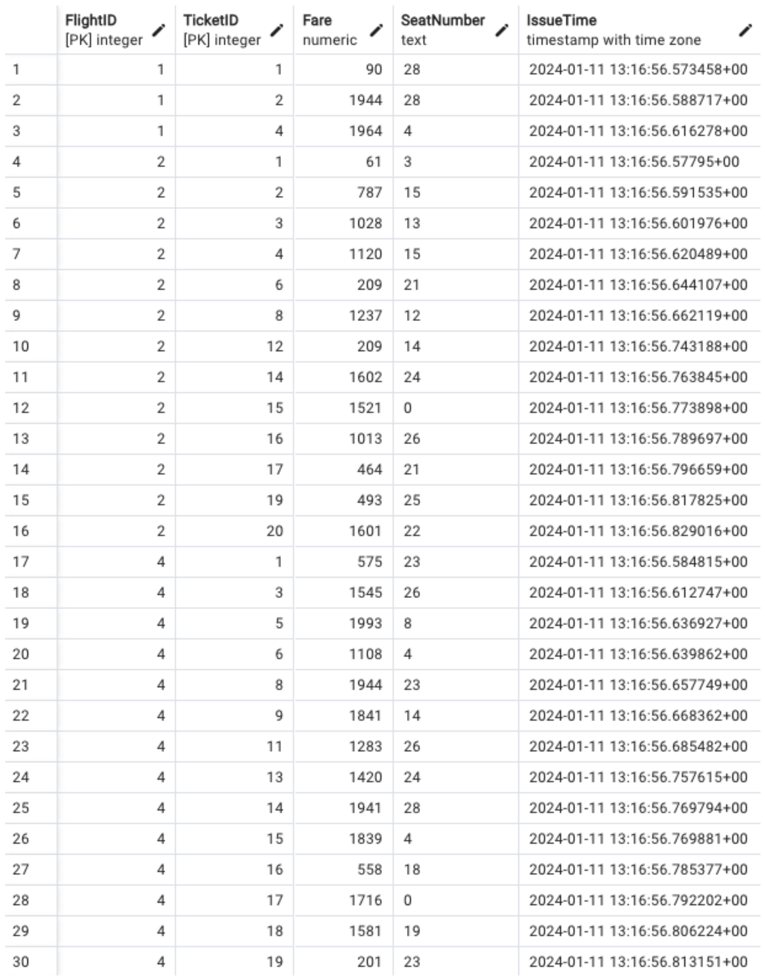
	Id [PK] integer	DepartureAirport text	ArrivalAirport text	DepartureTime timestamp with time zone	ArrivalTime timestamp with time zone
1	1	FCO	CDG	2024-01-23 06:15:00+00	2024-01-23 11:49:24.228297+00
2	2	CDG	SIN	2024-07-26 17:24:00+00	2024-07-27 08:08:42.766625+00
3	3	CDG	FCO	2024-11-02 06:13:00+00	2024-11-02 06:25:03.247952+00
4	4	DXB	AMS	2024-04-20 14:25:00+00	2024-04-20 21:27:54.202148+00
5	5	AMS	CDG	2024-11-09 10:08:00+00	2024-11-10 09:30:41.150925+00
6	6	AMS	DXB	2024-05-14 09:45:00+00	2024-05-14 17:11:01.406062+00
7	7	JFK	SIN	2024-11-12 16:57:00+00	2024-11-12 17:03:03.120746+00
8	8	SIN	LHR	2024-10-23 00:45:00+00	2024-10-23 10:05:35.162934+00
9	9	LHR	DXB	2024-09-05 16:58:00+00	2024-09-06 11:41:12.400339+00
10	10	DXB	FCO	2024-08-13 14:44:00+00	2024-08-14 08:07:40.019466+00
11	11	DXB	LHR	2024-10-15 14:15:00+00	2024-10-16 05:20:19.797079+00
12	12	CDG	JFK	2024-03-17 05:40:00+00	2024-03-18 00:07:48.040875+00
13	13	JFK	INN	2024-02-24 01:54:00+00	2024-02-24 16:10:09.880476+00
14	14	JFK	SIN	2024-03-14 07:57:00+00	2024-03-14 11:35:47.114704+00
15	15	SIN	LHR	2024-07-26 19:51:00+00	2024-07-27 10:33:05.945748+00
16	16	SIN	JFK	2024-04-14 13:17:00+00	2024-04-15 00:43:37.330385+00
17	17	SIN	INN	2024-09-11 18:49:00+00	2024-09-12 03:11:39.705728+00
18	18	INN	DXB	2024-03-11 20:04:00+00	2024-03-11 23:18:41.409786+00
19	19	INN	SIN	2024-09-21 10:13:00+00	2024-09-22 06:30:02.036756+00
20	20	AMS	FCO	2024-07-21 16:25:00+00	2024-07-21 23:13:27.908411+00
21	21	FCO	JFK	2024-01-22 23:08:00+00	2024-01-23 17:09:19.417411+00
22	22	FCO	AMS	2024-08-06 12:47:00+00	2024-08-07 09:37:29.361944+00
23	23	LHR	DXB	2024-06-18 03:25:00+00	2024-06-18 07:07:03.742412+00
24	24	DXB	FCO	2024-05-03 17:10:00+00	2024-05-03 17:44:12.419716+00
25	25	DXB	LHR	2024-07-25 13:36:00+00	2024-07-26 02:36:03.20006+00
26	26	CDG	JFK	2024-01-15 12:03:00+00	2024-01-15 19:35:47.309489+00
27	27	JFK	INN	2024-01-20 06:17:00+00	2024-01-20 11:17:49.944184+00
28	28	JFK	CDG	2024-06-11 23:51:00+00	2024-06-12 03:55:33.035012+00
29	29	LHR	AMS	2024-10-09 08:14:00+00	2024-10-09 16:10:11.589691+00

Bookings



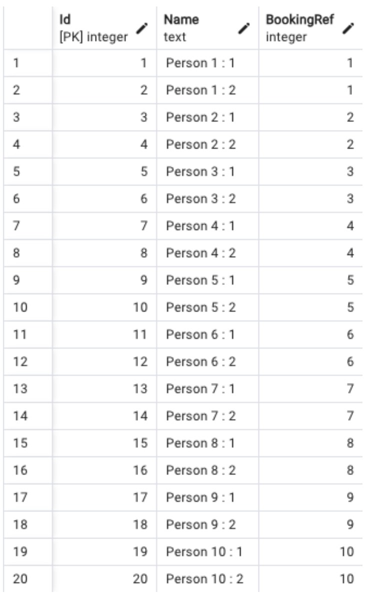
	Ref [PK] integer	Date date
1	1	2022-04-21
2	2	2022-05-09
3	3	2022-10-04
4	4	2022-10-09
5	5	2022-08-20
6	6	2022-06-17
7	7	2022-06-27
8	8	2022-08-25
9	9	2022-01-21
10	10	2022-07-01

BoardingPasses



	FlightID [PK] integer	TicketID [PK] integer	Fare numeric	SeatNumber text	IssueTime timestamp with time zone
1	1	1	90	28	2024-01-11 13:16:56.573458+00
2	1	2	1944	28	2024-01-11 13:16:56.588717+00
3	1	4	1964	4	2024-01-11 13:16:56.616278+00
4	2	1	61	3	2024-01-11 13:16:56.57795+00
5	2	2	787	15	2024-01-11 13:16:56.591535+00
6	2	3	1028	13	2024-01-11 13:16:56.601976+00
7	2	4	1120	15	2024-01-11 13:16:56.620489+00
8	2	6	209	21	2024-01-11 13:16:56.644107+00
9	2	8	1237	12	2024-01-11 13:16:56.662119+00
10	2	12	209	14	2024-01-11 13:16:56.743188+00
11	2	14	1602	24	2024-01-11 13:16:56.763845+00
12	2	15	1521	0	2024-01-11 13:16:56.773898+00
13	2	16	1013	26	2024-01-11 13:16:56.789697+00
14	2	17	464	21	2024-01-11 13:16:56.796659+00
15	2	19	493	25	2024-01-11 13:16:56.817825+00
16	2	20	1601	22	2024-01-11 13:16:56.829016+00
17	4	1	575	23	2024-01-11 13:16:56.584815+00
18	4	3	1545	26	2024-01-11 13:16:56.612747+00
19	4	5	1993	8	2024-01-11 13:16:56.636927+00
20	4	6	1108	4	2024-01-11 13:16:56.639862+00
21	4	8	1944	23	2024-01-11 13:16:56.657749+00
22	4	9	1841	14	2024-01-11 13:16:56.668362+00
23	4	11	1283	26	2024-01-11 13:16:56.685482+00
24	4	13	1420	24	2024-01-11 13:16:56.757615+00
25	4	14	1941	28	2024-01-11 13:16:56.769794+00
26	4	15	1839	4	2024-01-11 13:16:56.769881+00
27	4	16	558	18	2024-01-11 13:16:56.785377+00
28	4	17	1716	0	2024-01-11 13:16:56.792202+00
29	4	18	1581	19	2024-01-11 13:16:56.806224+00
30	4	19	201	23	2024-01-11 13:16:56.813151+00

Tickets



	Id [PK] integer	Name text	BookingRef integer
1	1	Person 1 : 1	1
2	2	Person 1 : 2	1
3	3	Person 2 : 1	2
4	4	Person 2 : 2	2
5	5	Person 3 : 1	3
6	6	Person 3 : 2	3
7	7	Person 4 : 1	4
8	8	Person 4 : 2	4
9	9	Person 5 : 1	5
10	10	Person 5 : 2	5
11	11	Person 6 : 1	6
12	12	Person 6 : 2	6
13	13	Person 7 : 1	7
14	14	Person 7 : 2	7
15	15	Person 8 : 1	8
16	16	Person 8 : 2	8
17	17	Person 9 : 1	9
18	18	Person 9 : 2	9
19	19	Person 10 : 1	10
20	20	Person 10 : 2	10

Important:

In the file Program.cs of the template, for each of the questions code examples can be found to test the implementation of the methods in Solution.cs
The mentioned code produces an output similar to the one of the hidden tests.

Exercise 1 [1.0 Points]:

Collect all the outbound and inbound flights at given airport, the arrival time should be used to order the result.

Method signature:

```
public static IQueryable<Flight> Q1(FlightContext db, string airport)
```

The record **Flight** (entity of the database) can be found in the file **FlightModel.cs**

Sample Result:

```
---Test Q1----
Chosen airport: AMS
Correct flights retrieved (7) .
Flights from or to AMS:
Flight { Id = 6, DepartureAirport = AMS, ArrivalAirport = FCO, DepartureTime = 1/17/2024 1:40:00 PM, ArrivalTime = 1/18/2024 7:32:32 AM }
Flight { Id = 15, DepartureAirport = FCO, ArrivalAirport = AMS, DepartureTime = 2/11/2024 10:26:00 AM, ArrivalTime = 2/11/2024 3:21:33 PM }
Flight { Id = 18, DepartureAirport = AMS, ArrivalAirport = DXB, DepartureTime = 2/26/2024 7:59:00 PM, ArrivalTime = 2/26/2024 11:06:05 PM }
Flight { Id = 17, DepartureAirport = AMS, ArrivalAirport = CDG, DepartureTime = 3/10/2024 2:06:00 PM, ArrivalTime = 3/11/2024 5:28:13 AM }
Flight { Id = 27, DepartureAirport = AMS, ArrivalAirport = FCO, DepartureTime = 5/7/2024 6:14:00 PM, ArrivalTime = 5/8/2024 10:45:20 AM }
Flight { Id = 16, DepartureAirport = DXB, ArrivalAirport = AMS, DepartureTime = 7/25/2024 7:41:00 PM, ArrivalTime = 7/26/2024 3:10:44 AM }
Flight { Id = 13, DepartureAirport = AMS, ArrivalAirport = FCO, DepartureTime = 10/24/2024 2:58:00 PM, ArrivalTime = 10/24/2024 7:50:10 PM }
```

Exercise 2 [1.0 Points]:

For given person Name (attribute of Ticket) find the boardingpasses (flight id, Ticket id, fare, seat number and issue date) with passenger name [BoardingPassWithName].

Method signature:

```
public static IQueryable<BoardingPassWithName> Q2(FlightContext db, string person)
```

The record **BoardingPassWithName** (inherited from record BoardingPass, entity of the db) can be found in the file **DataFormats.cs**

```
public record BoardingPassWithName : BoardingPass
{
    public string Name{get; set;}
    public BoardingPassWithName(int FlightID, int TicketID, decimal Fare, string SeatNumber, string name, DateTime issueDate)
        : base(FlightID, TicketID, Fare, SeatNumber, issueDate)
    {
        Name = name;
    }

    public BoardingPassWithName(BoardingPass bp, string name) : base(bp)
    {
        Name = name;
    }
}
```

Sample Result:

```
---Test Q2----
PersonName:  Person 5 : 2
Correct Boardingpasses retrieved (5).
Boardingpasses for person Person 5 : 2:
BoardingPassWithName { FlightID = 4, TicketID = 10, Fare = 706, SeatNumber = 0, IssueTime = 1/11/2024 5:08:15 PM, Name = Person 5 : 2 }
BoardingPassWithName { FlightID = 5, TicketID = 10, Fare = 472, SeatNumber = 8, IssueTime = 1/11/2024 5:08:15 PM, Name = Person 5 : 2 }
BoardingPassWithName { FlightID = 12, TicketID = 10, Fare = 367, SeatNumber = 21, IssueTime = 1/11/2024 5:08:15 PM, Name = Person 5 : 2 }
BoardingPassWithName { FlightID = 17, TicketID = 10, Fare = 1887, SeatNumber = 11, IssueTime = 1/11/2024 5:08:15 PM, Name = Person 5 : 2 }
BoardingPassWithName { FlightID = 27, TicketID = 10, Fare = 1181, SeatNumber = 24, IssueTime = 1/11/2024 5:08:15 PM, Name = Person 5 : 2 }
```

Exercise 3 [1.5 Points]:

Returns an instance of BookingOverview for a given booking, fields to be computed:
FlightDetails : List of Tuples containing Departure and Arrival airports for each Flight in the booking.
TotalFare: Compute the total fare of given booking, considering all the BoardingPasses involved.

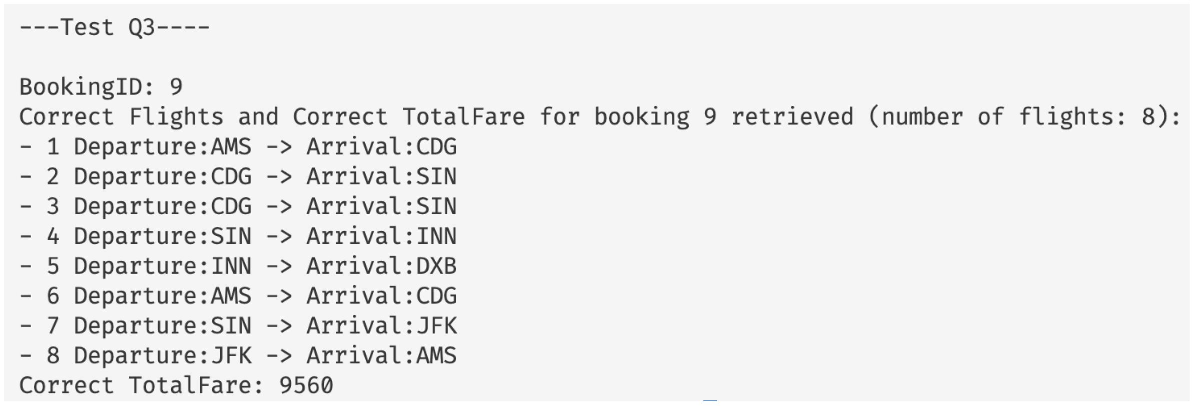
Method signature:

```
public static BookingOverview Q3(FlightContext db, int booking)
```

The record **BookingOverview** can be found in the file **DataFormats.cs**

```
public record BookingOverview(List<Tuple<string, string>> FlightDetails, decimal TotalFare);
```

Sample Result:



```
---Test Q3----

BookingID: 9
Correct Flights and Correct TotalFare for booking 9 retrieved (number of flights: 8):
- 1 Departure:AMS -> Arrival:CDG
- 2 Departure:CDG -> Arrival:SIN
- 3 Departure:CDG -> Arrival:SIN
- 4 Departure:SIN -> Arrival:INN
- 5 Departure:INN -> Arrival:DXB
- 6 Departure:AMS -> Arrival:CDG
- 7 Departure:SIN -> Arrival:JFK
- 8 Departure:JFK -> Arrival:AMS
Correct TotalFare: 9560
```

Exercise 4 [2 Points]:

List down the number of seats booked (TotalSeats) per flight (FlightID) [SeatsInFlight], do not forget to include flights with no Boarding passes issued (no seats booked), as well if any => LEFT JOIN.
Using the Sum method might be useful to compute TotalSeats.

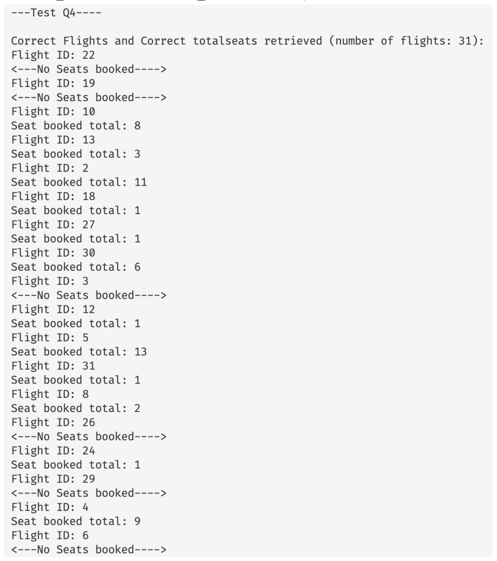
Method signature:

```
public static IQueryable<SeatsInFlight> Q4(FlightContext db)
```

The record **SeatsInFlight** can be found in the file **DataFormats.cs**

```
public record SeatsInFlight(int FlightID, int TotalSeats);
```

Sample Result (part of it):



```
---Test Q4----

Correct Flights and Correct totalseats retrieved (number of flights: 31):
Flight ID: 22
<---No Seats booked---->
Flight ID: 19
<---No Seats booked---->
Flight ID: 10
Seat booked total: 8
Flight ID: 13
Seat booked total: 3
Flight ID: 2
Seat booked total: 11
Flight ID: 18
Seat booked total: 1
Flight ID: 27
Seat booked total: 1
Flight ID: 30
Seat booked total: 6
Flight ID: 3
<---No Seats booked---->
Flight ID: 12
Seat booked total: 1
Flight ID: 5
Seat booked total: 13
Flight ID: 31
Seat booked total: 1
Flight ID: 8
Seat booked total: 2
Flight ID: 26
<---No Seats booked---->
Flight ID: 24
Seat booked total: 1
Flight ID: 29
<---No Seats booked---->
Flight ID: 4
Seat booked total: 9
Flight ID: 6
<---No Seats booked---->
```

Exercise 5 [1.5 Points]:
List down the Flights [if any] that were never booked (no Boarding passes issued).

Method signature:

```
public static IQueryable<Flight> Q5(FlightContext db)
```

The record **Flight** (entity of the database) can be found in the file **FlightModel.cs**
Sample Result:

```
---Test Q5----

Correct (never booked) Flights (number of flights: 11):
Flight { Id = 1, DepartureAirport = AMS, ArrivalAirport = FCO, DepartureTime = 1/4/2024 4:36:00 AM, ArrivalTime = 1/4/2024 10:13:11 PM }
Flight { Id = 9, DepartureAirport = CDG, ArrivalAirport = FCO, DepartureTime = 10/7/2024 3:36:00 PM, ArrivalTime = 10/8/2024 5:16:46 AM }
Flight { Id = 10, DepartureAirport = JFK, ArrivalAirport = SIN, DepartureTime = 8/3/2024 11:43:00 AM, ArrivalTime = 8/3/2024 3:07:04 PM }
Flight { Id = 11, DepartureAirport = SIN, ArrivalAirport = LHR, DepartureTime = 4/19/2024 3:44:00 AM, ArrivalTime = 4/19/2024 11:49:21 PM }
Flight { Id = 15, DepartureAirport = AMS, ArrivalAirport = FCO, DepartureTime = 6/21/2024 3:08:00 PM, ArrivalTime = 6/21/2024 4:52:34 PM }
Flight { Id = 17, DepartureAirport = JFK, ArrivalAirport = SIN, DepartureTime = 3/19/2024 1:34:00 PM, ArrivalTime = 3/20/2024 9:05:06 AM }
Flight { Id = 18, DepartureAirport = SIN, ArrivalAirport = LHR, DepartureTime = 6/26/2024 2:57:00 AM, ArrivalTime = 6/26/2024 1:13:02 PM }
Flight { Id = 21, DepartureAirport = INN, ArrivalAirport = DXB, DepartureTime = 3/25/2024 1:05:00 AM, ArrivalTime = 3/25/2024 8:17:38 AM }
Flight { Id = 22, DepartureAirport = INN, ArrivalAirport = SIN, DepartureTime = 10/12/2024 11:48:00 AM, ArrivalTime = 10/13/2024 5:09:01 AM }
Flight { Id = 25, DepartureAirport = DXB, ArrivalAirport = LHR, DepartureTime = 8/8/2024 6:38:00 AM, ArrivalTime = 8/9/2024 4:50:30 AM }
Flight { Id = 26, DepartureAirport = CDG, ArrivalAirport = JFK, DepartureTime = 4/20/2024 6:23:00 AM, ArrivalTime = 4/20/2024 2:45:00 PM }
```

Exercise 6 [1.5 Points]:
Given two ticket IDs, merge the FlightInfo elements (projection of Flight entity) belonging to both given tickets WITHOUT repetitions.

Method signature:

```
public static List<FlightInfo> Q6(FlightContext db, int TicketID1, int TicketID2)
```

The record **FlightInfo** can be found in the file **DataFormats.cs**

```
public record FlightInfo(int FlightID, string Dept, string Arr);
```

Sample Result:

```
---Test Q6----
TicketID: 9
 - FlightInfo: (16, DXB, LHR)
 - FlightInfo: (30, LHR, AMS)
 - FlightInfo: (2, AMS, CDG)
 - FlightInfo: (20, CDG, SIN)
 - FlightInfo: (6, SIN, INN)
 - FlightInfo: (7, INN, DXB)
TicketID: 10
 - FlightInfo: (13, FCO, AMS)
 - FlightInfo: (2, AMS, CDG)
 - FlightInfo: (20, CDG, SIN)
 - FlightInfo: (6, SIN, INN)

Correct Flights (number of flights: 7):
FlightInfo { FlightID = 16, Dept = DXB, Arr = LHR }
FlightInfo { FlightID = 30, Dept = LHR, Arr = AMS }
FlightInfo { FlightID = 2, Dept = AMS, Arr = CDG }
FlightInfo { FlightID = 20, Dept = CDG, Arr = SIN }
FlightInfo { FlightID = 6, Dept = SIN, Arr = INN }
FlightInfo { FlightID = 7, Dept = INN, Arr = DXB }
FlightInfo { FlightID = 13, Dept = FCO, Arr = AMS }
```

Exercise 7 [1.5 Points]:

Create a new flight, new booking, new ticket and a new boarding pass and make the changes persistent.
HINT:
having a look at the implementation of the seed methods in Data class (FlightModel.cs) can be useful, as well as DateTimeUtils methods in DataFormats.cs

Method signature:

```
public static void Q7(FlightContext db)
```

Sample Result:

```
---Test Q7----
  1 flights
  1 bookings
  1 tickets
  1 boardingpasses
Correct number of Flights, Bookings, Tickets, BoardingPasses added in the Database
```