Data Base Implementation

RAFAŁ GRZEGORZ STAŃCZAK

BACHELOR'S DEGREE YEAR II / SEM. II

Field of study: Computer Engineering

Course: Data Bases

Teacher in charge: Prof. Paulo Nogueira Martins

Teacher: Prof. Daniel Moreira Lopes Alexandre

# Contents:

[1. Entity-Relationship (E-R) diagram from a database: 3](#_Toc178086147)

[2. Mapping the E-R diagram to the relational model 4](#_Toc178086148)

[3. Normalization of the relational model up to the 3rd Normal Form 11](#_Toc178086149)

[4. Relational Model: 13](#_Toc178086150)

[5. Implementation of the physical model of the database with the respective integrity restrictions, using the SQL language 14](#_Toc178086151)

[6. Creation of the database diagram (graphical presentation of tables and their relationships) 16](#_Toc178086152)

[Exercise 1 17](#_Toc178086153)

[Data Insertion to Database 17](#_Toc178086154)

[Exercise 2 19](#_Toc178086155)

[Write the SQL language that answers each of the following questions 19](#_Toc178086156)

[2.1 Who was the 1st Hired Teacher? [Teacher (Name, Gender), Date] 19](#_Toc178086157)

[2.2 How many different Students have enrolled each University? [University (Name), N\_Students] 19](#_Toc178086158)

[2.3 Which students attended school the longest? [Students (Name), Number of Years] 19](#_Toc178086159)

[2.4 Which Teachers were Hired in the last 60 days with salaries below 1200€? [Teacher (Name), Date, Salary] 20](#_Toc178086160)

[2.5 Which universities currently have courses with no students enrolled? [Universities (Name)] List them in alphabetical order. 20](#_Toc178086161)

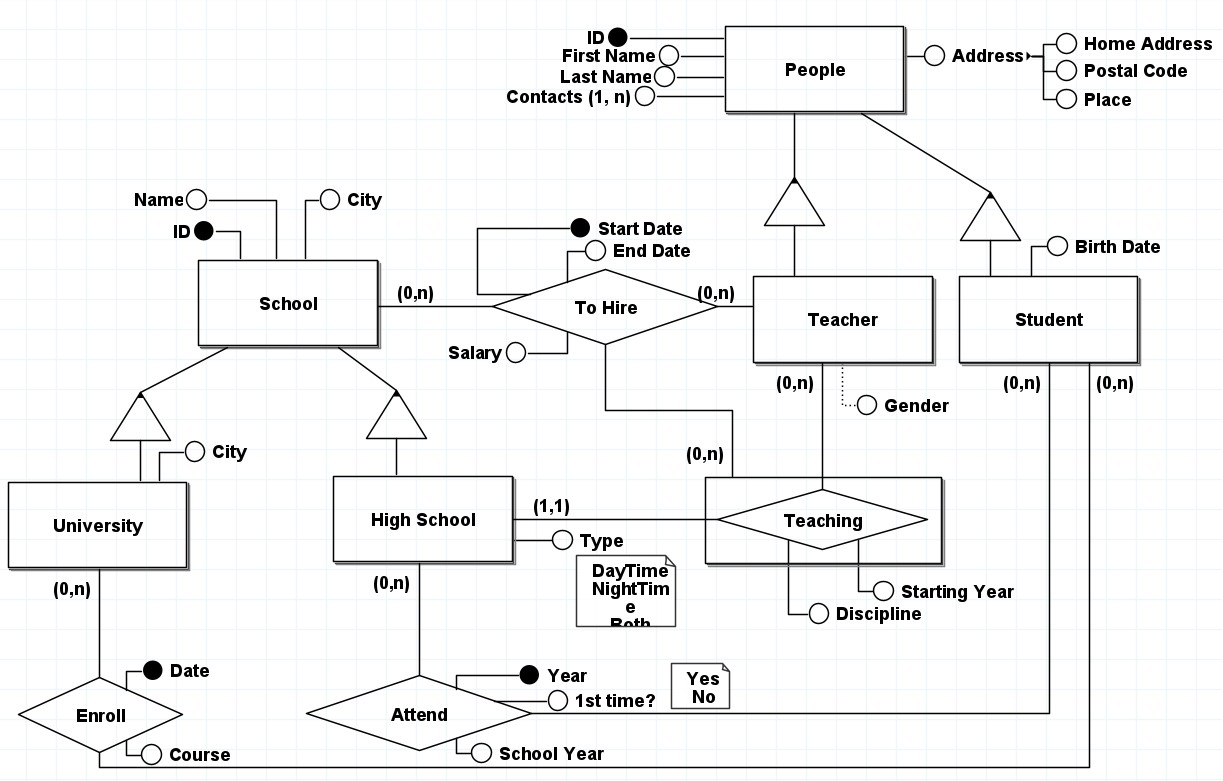
[Exercise 3 21](#_Toc178086162)

[Assuming that each Student has a cost of 50€/day for the school, create a procedure that for a given month calculates the Total Value that each school spent, presenting a table with the names of the schools and the total spent in each one in that month. The procedure must return the total amount spent by all the schools. 21](#_Toc178086163)

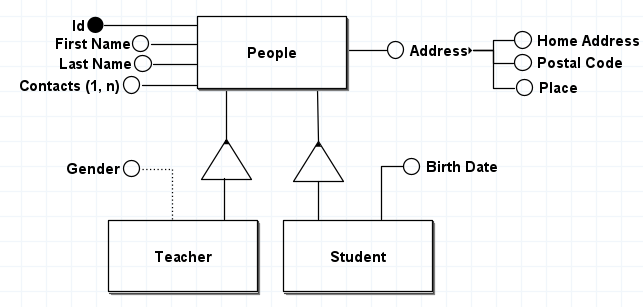
[Exercise 4 23](#_Toc178086164)

[A student cannot attend more than 2 Courses at the same time. Create a trigger that only lets you insert a new record in the Enroll relationship if this student is not yet enrolled in 2 courses. 23](#_Toc178086165)

# Entity-Relationship (E-R) diagram from a database:



# Mapping the E-R diagram to the relational model



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **People** | | | | |
| ID | FirstName | Contacts (1,n) | LastName | Address |
| 1 | John | 26 | Doe | 123 Main St 12345 City1 |
| 2 | Jane | 27 | Smith | 456 Elm St 54321 City2 |
| 3 | Alice | 28 | Johnson | 789 Oak St 67890 City3 |
| 4 | Bob | 29 | Williams | 101 Pine St 13579 City4 |
| 5 | Emily | 30 | Brown | 202 Maple St 97531 City5 |
| 6 | Rafał | 31 | Kawka | 203 Maple St 91531 City6 |
| 7 | Mary | 32 | White | 204 Maple St 91921 City7 |
| 8 | Kamil | 33 | Yellow | 205 Maple St 17531 City8 |
| 9 | Józef | 34 | Sikora | 206 Maple St 37531 City9 |
| 10 | Emily | 35 | Kowalski | 207 Maple St 44531 City10 |

|  |  |
| --- | --- |
| **Teacher** | |
| ~~Teacher\_ID~~ | Gender |
| 1 | Male |
| 2 | Female |
| 3 | Male |
| 4 | Female |
| 5 | Male |

|  |  |
| --- | --- |
| **Student** | |
| ~~Student\_ID~~ | Birth Date |
| 6 | 01.01.2000 |
| 7 | 03.02.2001 |
| 8 | 04.03.2000 |
| 9 | 05.04.2000 |
| 10 | 06.05.2000 |

In the Relational Model it becomes:

**People** (ID, First Name, Last Name, Contacts (1,n), Address)

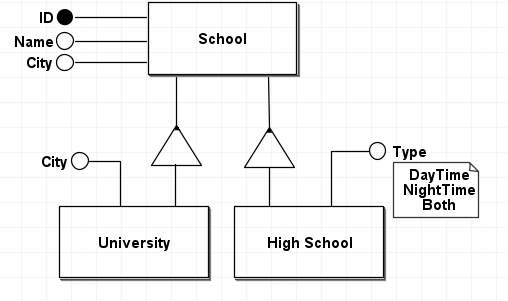
**Teacher** (~~Teacher\_ID,~~ Gender)

*Teacher\_ID references People*

*\*Gender is optional attribute - can be null in database*

**Student** (~~Student\_ID~~, Birth Date)

*Student\_ID references People*



|  |  |  |
| --- | --- | --- |
| **School** | | |
| ID | Name | City |
| 100 | School1 | City1 |
| 101 | School2 | City2 |
| 102 | School3 | City3 |
| 103 | School4 | City4 |
| 104 | School5 | City5 |
| 105 | School6 | City6 |
| 106 | School7 | City7 |
| 107 | School8 | City8 |
| 108 | School9 | City9 |
| 109 | School10 | City10 |

|  |  |
| --- | --- |
| **University** | |
| ~~University\_ID~~ | City |
| 100 | City1 |
| 101 | City2 |
| 102 | City3 |
| 103 | City4 |
| 104 | City5 |

|  |  |
| --- | --- |
| **High School** | |
| ~~HighSchool\_ID~~ | Type |
| 105 | DayTime |
| 106 | NightTime |
| 107 | Both |
| 108 | DayTime |
| 109 | NightTime |

In the Relational Model it becomes:

**School** (ID, Name, City)

School is associated with University by IsA Relationship

School is associated with High School by IsA Relationship

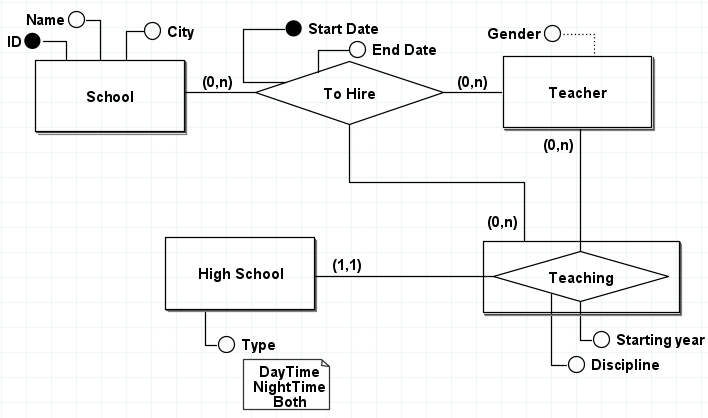
**University** (~~University\_ID~~, City)

*University\_ID references School*

**High** **School** (~~HighSchool\_ID~~, Type (DayTime, NightTime, Both))

*HighSchool\_ID references School*

*\*Attribute Type can only contain one of three values: DayTime, NightTime, Both*

**

|  |  |  |  |
| --- | --- | --- | --- |
| **Teaching** | | | |
| ~~Teacher\_ID~~ | ~~HighSchool\_ID~~ | Starting Year | Discipline |
| 1 | 105 | 2010 | Math |
| 2 | 106 | 2011 | Engineering |
| 3 | 107 | 2012 | Phyiscs |
| 4 | 108 | 2013 | Math |
| 5 | 109 | 2014 | Phyiscs |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **To Hire** | | | | |
| ~~ID~~ | ~~Teacher\_ID~~ | T\_ID | Start Date | End Date |
| 100 | 1 | 3 | 01.01.2020 | 01.01.2021 |
| 101 | 3 | 3 | 02.02.2020 | 02.02.2021 |
| 105 | 1 | 3 | 03.03.2020 | 03.03.2021 |
| 106 | 2 | 5 | 04.04.2020 | 04.04.2021 |
| 109 | 2 | 5 | 05.05.2020 | 05.05.2021 |

In the Relational Model it becomes:

**High** **School** (~~HighSchool\_ID~~, Type (DayTime, NightTime, Both))

**Teacher** (~~Teacher\_ID,~~ Gender)

**Teaching** (~~T\_ID~~, ~~HighSchool\_ID~~, Starting Year, Discipline)

*HighSchool\_ID references High School*

*T\_ID references Teacher*

*\** *With 1:N relation Teaching is taking primary key from N side*

**School** (ID, Name, City)

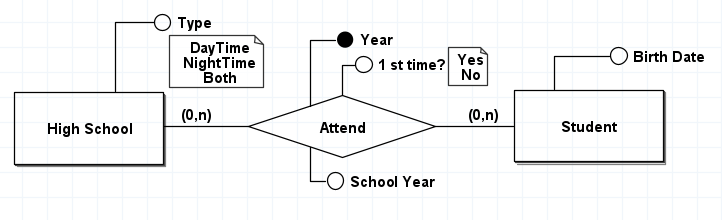
Ternary relationship (Association among three entities):

**To Hire** (~~ID~~, ~~Teacher\_ID~~, ~~T\_ID~~, Start Date, End Date)

*Teacher\_ID references Teacher*

*T\_ID references Teaching*

*ID references School*



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attend** | | | | |
| Year | School Year | FirstTime (Yes, No) | ~~HighSchool\_ID~~ | ~~Student\_ID~~ |
| 2020 | 1 | Yes | 105 | 6 |
| 2020 | 2 | No | 106 | 7 |
| 2020 | 3 | Yes | 108 | 8 |
| 2020 | 4 | No | 108 | 9 |
| 2020 | 5 | Yes | 109 | 10 |

In the Relational Model it becomes:

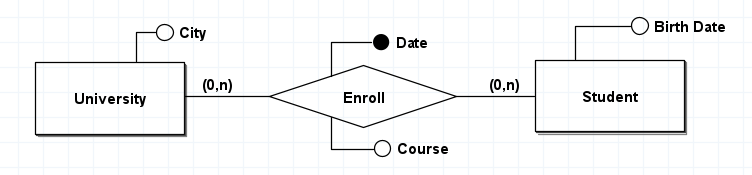
**High** **School** (~~HighSchool\_ID~~, Type (DayTime, NightTime, Both))

**Student** (~~Student\_ID~~, Birth Date)

**Attend** (Year, SchoolYear, FirstTime (Yes, No), ~~HighSchool\_ID~~, ~~Student\_ID~~)

*HighSchool\_ID references High School*

*Student\_ID references Student*



|  |  |  |  |
| --- | --- | --- | --- |
| **Enroll** | | | |
| ~~Student\_ID~~ | ~~University\_ID~~ | Date | Course |
| 6 | 101 | 01.01.2020 | Managment |
| 7 | 101 | 02.02.2020 | Safety and Health |
| 8 | 103 | 03.03.2020 | Managment |
| 9 | 104 | 04.04.2020 | Safety and Health |
| 10 | 102 | 05.05.2020 | Managment |

In the Relational Model it becomes:

**Student** (~~Student\_ID~~, Birth Date)

**University** (~~University\_ID~~, City)

**Enroll** (~~Student\_ID~~, ~~University\_ID~~, Date, Course)

*Student\_ID references Student*

*University\_ID references University*

# Normalization of the relational model up to the 3rd Normal Form

**A relation is said to be in First Normal Form (1NF) when:**

* Does not contain multivalued attributes;
* Does not contain repeating groups.

**People** (ID, First Name, Last Name, Contacts (1,n), Address)

Address is multivalued attribute. It has to be unfolded.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **People** | | | | | | |
| ID | FirstName | LastName | Contact\_ID | HomeAddress | PostalCode | Place |
| 1 | John | Doe | 26 | 123 Main St | 12345 | City1 |
| 2 | Jane | Smith | 27 | 456 Elm St | 54321 | City2 |
| 3 | Alice | Johnson | 28 | 789 Oak St | 67890 | City3 |
| 4 | Bob | Williams | 29 | 101 Pine St | 13579 | City4 |
| 5 | Emily | Brown | 30 | 202 Maple St | 97531 | City5 |
| 6 | Rafał | Kawka | 31 | 203 Maple St | 91531 | City6 |
| 7 | Mary | White | 32 | 204 Maple St | 91921 | City7 |
| 8 | Kamil | Yellow | 33 | 205 Maple St | 17531 | City8 |
| 9 | Józef | Sikora | 34 | 206 Maple St | 37531 | City9 |
| 10 | Emily | Kowalski | 35 | 207 Maple St | 44531 | City10 |

In the Relational Model it becomes:

**People** (ID, First Name, Last Name, Contacts (1,n), HomeAddress, PostalCode, Place)

**A relation is said to be in Second Normal Form (2NF) :**

* Is in First Normal Form (1NF);
* All non-key attributes functionally depend on the entirety of the key.

Contacts is multi-valued attribute so we are creating a separate table to represent the contacts. Include a foreign key in this table to reference the primary key of the People table. In this setup, each row in the Contacts table represents a single contact associated with a person.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **People** | | | | | |
| ID | FirstName | LastName | HomeAddress | PostalCode | Place |
| 1 | John | Doe | 123 Main St | 12345 | City1 |
| 2 | Jane | Smith | 456 Elm St | 54321 | City2 |
| 3 | Alice | Johnson | 789 Oak St | 67890 | City3 |
| 4 | Bob | Williams | 101 Pine St | 13579 | City4 |
| 5 | Emily | Brown | 202 Maple St | 97531 | City5 |
| 6 | Rafał | Kawka | 203 Maple St | 91531 | City6 |
| 7 | Mary | White | 204 Maple St | 91921 | City7 |
| 8 | Kamil | Yellow | 205 Maple St | 17531 | City8 |
| 9 | Józef | Sikora | 206 Maple St | 37531 | City9 |
| 10 | Emily | Kowalski | 207 Maple St | 44531 | City10 |

|  |  |
| --- | --- |
| **Contacts** | |
| Contact\_ID | People\_ID |
| 26 | 1 |
| 27 | 2 |
| 28 | 3 |
| 29 | 4 |
| 30 | 5 |

In the Relational Model it becomes:

**People** (ID, First Name, Last Name, HomeAddress, PostalCode, Place)

**Contacts** (Contact\_ID, ~~People\_ID~~)

People\_ID references People

**A relation is said to be in Third Normal Form (3NF) :**

* It is in Second Normal Form (2NF);
* All non-key attributes are not functionally dependent on each other;
* In the 3NF verification, derived attributes should also be eliminated, as they would imply a functional dependency between non-key elements.

After normalization the relational model is ready to be implemented to physical model using SQL.

# Relational Model:

**People** (ID, First Name, Last Name, Home Address, Postal Code, Place)

**Contacts** (Contact\_ID, ~~People\_ID~~)

People\_ID references People

**Teacher** (~~Teacher\_ID,~~ Gender)

*Teacher\_ID references People*

**Student** (~~Student\_ID~~, Birth Date)

*Student\_ID references People*

**School** (ID, Name, City)

**University** (~~University\_ID~~, City)

*University\_ID references School*

**High** **School** (~~HighSchool\_ID~~, Type (DayTime, NightTime, Both))

*HighSchool\_ID references School*

**Teaching** (~~HighSchool\_ID~~, ~~T\_ID~~, Starting Year, Discipline)

*HighSchool\_ID references High School*

*T\_ID references Teacher*

**To Hire** (~~ID~~, ~~Teacher\_ID~~, ~~T\_ID~~, Start Date, End Date)

*Teacher\_ID references Teacher*

*T\_ID references Teaching*

*ID references School*

**Attend** (Year, SchoolYear, FirstTime (Yes, No), ~~HighSchool\_ID~~, ~~Student\_ID~~)

*HighSchool\_ID references High School*

*Student\_ID references Student*

**Enroll** (~~Student\_ID~~, ~~University\_ID~~, Date, Course)

*Student\_ID references Student*

*University\_ID references University*

# Implementation of the physical model of the database with the respective integrity restrictions, using the SQL language

After normalizing the relational model, the physical model can be implemented. This implementation typically involves creating tables in a database using SQL. These tables are then connected through foreign keys.

use master;

-- Create new database;

CREATE DATABASE Experimental\_Work;

-- Use created database

use Experimental\_Work;

-- Daniel -- 3rd Normal Form (3FN) --

CREATE TABLE PostalCode(

PostalCode varchar(8) not null,

Place varchar(50) not null,

PRIMARY KEY (PostalCode),

CHECK PostalCode LIKE ('[1-9][1-9][1-9][1-9]-[1-9][1-9][1-9]')

)

CREATE TABLE People (

ID integer not null,

FirstName varchar(50) not null,

LastName varchar(50) not null,

HomeAddress varchar(50) not null,

PostalCode varchar(8) not null,

FOREIGN KEY (PostalCode) REFERENCES PostalCode(PostalCode),

PRIMARY KEY (ID)

)

-- Create Contacts Table

CREATE TABLE Contacts (

Contact\_ID integer not null,

People\_ID integer not null,

FOREIGN KEY (People\_ID) REFERENCES People(ID),

PRIMARY KEY (Contact\_ID)

)

-- Create Teacher Table

CREATE TABLE Teacher (

Teacher\_ID integer not null,

Gender varchar(50),

FOREIGN KEY (Teacher\_ID) REFERENCES People(ID),

PRIMARY KEY (Teacher\_ID)

)

-- Create Student Table

CREATE TABLE Student (

Student\_ID integer not null,

BirthDate date not null,

FOREIGN KEY (Student\_ID) REFERENCES People(ID),

PRIMARY KEY (Student\_ID)

)

-- Create School Table

CREATE TABLE School (

ID integer not null,

Name varchar(50) not null,

City varchar(50) not null,

PRIMARY KEY (ID)

)

-- Create University Table

CREATE TABLE University (

University\_ID integer not null,

City varchar(50) not null,

FOREIGN KEY (University\_ID) REFERENCES School(ID),

PRIMARY KEY (University\_ID)

)

-- Create High School Table

CREATE TABLE HighSchool (

HighSchool\_ID integer not null,

Type\_ varchar(50) not null,

FOREIGN KEY (HighSchool\_ID) REFERENCES School(ID),

PRIMARY KEY (HighSchool\_ID),

CHECK(Type\_ IN ('DayTime','NigthTime','Both'))

)

-- Create Teaching Table

CREATE TABLE Teaching (

T\_ID integer not null,

HighSchool\_ID integer not null,

StartingYear integer not null,

Discipline varchar(50) not null,

FOREIGN KEY (T\_ID) REFERENCES Teacher(Teacher\_ID),

FOREIGN KEY (HighSchool\_ID) REFERENCES HighSchool(HighSchool\_ID),

PRIMARY KEY (T\_ID)

)

-- Create ToHire Table

CREATE TABLE ToHire (

ID integer not null,

Teacher\_ID integer not null,

T\_ID integer not null,

StartDate date not null,

EndDate date not null,

Salary money not null,

FOREIGN KEY (Teacher\_ID) REFERENCES Teacher(Teacher\_ID),

FOREIGN KEY (T\_ID) REFERENCES Teaching(T\_ID),

FOREIGN KEY (ID) REFERENCES School(ID),

PRIMARY KEY (ID, Teacher\_ID, T\_ID, StartDate)

)

-- Create Attend Table

CREATE TABLE Attend (

Year\_ integer not null,

SchoolYear integer not null,

FirstTime varchar(3) not null,

HighSchool\_ID integer not null,

Student\_ID integer not null,

FOREIGN KEY (HighSchool\_ID) REFERENCES HighSchool(HighSchool\_ID),

FOREIGN KEY (Student\_ID) REFERENCES Student(Student\_ID),

PRIMARY KEY (Year\_, HighSchool\_ID, Student\_ID),

CHECK (FirstTime IN ('Yes','No'))

)

-- Create Enroll Table

CREATE TABLE Enroll (

Student\_ID integer not null,

University\_ID integer not null,

Date\_ date not null,

Course varchar(50) not null,

FOREIGN KEY (Student\_ID) REFERENCES Student(Student\_ID),

FOREIGN KEY (University\_ID) REFERENCES University(University\_ID),

PRIMARY KEY (Student\_ID, University\_ID, Date\_)

)

# Creation of the database diagram (graphical presentation of tables and their relationships)

A graphical presentation of tables and their relationships is compatible with the original diagram. The visual representation of tables and the connections between them in a graphical format aligns with the initial diagram.

**Exercise 1**

**Data Insertion to Database**

To extend database with sample data, we use the ***INSERT INTO*** method. This allows us to add records to tables by specifying the table name and providing the relevant data using the ***VALUES*** keyword.

-- Insert sample data into PostalCode Table

insert into PostalCode (PostalCode, Place) VALUES

('1234-567', 'City1'),

('2345-678', 'City2'),

('3456-789', 'City3');

-- Insert sample data into People Table

insert into People values

(1,'John', 'Doe', '123 Main St', '1234-567'),

(2,'Jane', 'Smith', '456 Elm St', '2345-678'),

(3,'Alice', 'Johnson', '789 Oak St', '1234-567'),

(4,'Bob', 'Williams', '101 Pine St', '2345-678'),

(5,'Emily', 'Brown', '202 Maple St', '1234-567'),

(6,'Rafał', 'Kawka', '203 Maple St', '3456-789'),

(7,'Mary', 'White', '204 Maple St', '3456-789'),

(8,'Kamil', 'Yellow', '205 Maple St', '1234-567'),

(9,'Józef', 'Sikora', '206 Maple St', '1234-567'),

(10,'Emily', 'Kowalski', '207 Maple St', '2345-678');

-- Check Table

SELECT \* FROM People;

-- Insert sample data into Contacts Table

insert into Contacts values

(26, 1),

(27, 2),

(28, 3),

(29, 4),

(30, 5),

(31, 6)

-- Insert sample data into Teacher Table

insert into Teacher values

(1, 'Male'),

(2, 'Female'),

(3, 'Male'),

(4, 'Female'),

(5, 'Male'),

(6, 'Male')

-- Insert sample data into Student Table

insert into Student values

(6,'2000-01-01'),

(7,'2001-02-03'),

(8,'2002-03-04'),

(9,'2003-04-05'),

(10,'2004-05-06')

-- Insert sample data into School Table

insert into School values

(100,'School1', 'City1'),

(101,'School2', 'City2'),

(102,'School3', 'City3'),

(103,'School4', 'City4'),

(104,'School5', 'City5'),

(105,'School6', 'City6'),

(106,'School7', 'City7'),

(107,'School8', 'City8'),

(108,'School9', 'City9'),

(109,'School10', 'City10')

-- Insert sample data into University Table

insert into University values

(100, 'City1'),

(101, 'City2'),

(102, 'City3'),

(103, 'City4'),

(104, 'City5')

-- Insert sample data into High School Table

insert into HighSchool values

(105, 'DayTime'),

(106, 'NigthTime'),

(107, 'Both'),

(108, 'DayTime'),

(109, 'NigthTime')

-- Insert sample data into Teaching Table

insert into Teaching values

(1, 105, 2010, 'Math'),

(2, 106, 2011, 'Engineering'),

(3, 107, 2012, 'Phyiscs'),

(4, 108, 2013, 'Math'),

(5, 109, 2014, 'Phyiscs')

-- Insert sample data into ToHire Table

insert into ToHire values

(100, 1, 3, '2020-01-01', '2021-01-01', 1000),

(101, 3, 3, '2020-02-02', '2021-02-02', 1100),

(105, 1, 3, '2020-03-03', '2021-03-03', 2000),

(106, 2, 5, '2020-04-04', '2021-04-04', 2100),

(109, 2, 5, '2020-05-05', '2021-05-05', 3000),

(107, 2, 5, '2024-05-05', '2025-05-05', 1000)

-- Insert sample data into Attend Table

insert into Attend values

(2020, 1, 'Yes', 105, 6),

(2021, 2, 'No', 106, 7),

(2022, 3, 'Yes', 108, 8),

(2022, 4, 'No', 108, 9),

(2020, 5, 'Yes', 109, 10)

-- Insert sample data into Enroll Table

insert into Enroll values

(6, 101, '2020-01-01', 'Managment'),

(7, 101, '2020-02-02', 'Safety and Health'),

(8, 103, '2020-03-03', 'Managment'),

(9, 104, '2020-04-04', 'Safety and Health'),

(10, 102, '2020-05-05', 'Managment')

Each table now holds some data, ensuring our database is sufficiently populated for testing and analysis.

**Exercise 2**

**Write the SQL language that answers each of the following questions**

***SELECT*** – we are starting with this statement it is specifying the columns we want to retrieve from the database.

***FROM*** – this indicates the table from which to retrieve data.

***INNER JOIN*** – it is designation for the common type of Join, in which two or more tables are joined, linking them through the primary key of one and the foreign key of the other.

***ON*** – this specifies the condition for the join, indicating which columns from each table are used for the join.

***WHERE*** – this clause is used to filter the rows returned by the SELECT statement based on specified conditions.

***AS*** – this is used to provide an alias for a column or table name, making the output more readable.

1. **Who was the 1st Hired Teacher? [Teacher (Name, Gender), Date]**

--2.1. Who was the 1st Hired Teacher? [Teacher (Name, Gender), Date]

SELECT FirstName AS Name, Gender, StartDate AS Date

FROM ToHire

INNER JOIN Teacher ON Teacher.Teacher\_ID = ToHire.Teacher\_ID

INNER JOIN People ON Teacher.Teacher\_ID = People.ID

WHERE ToHire.StartDate = (SELECT MIN(StartDate) FROM ToHire);

***MIN()*** – This is a function that returns the smallest value of a selected column.

1. **How many different Students have enrolled each University? [University (Name), N\_Students]**

--2.2. How many different Students have enrolled each University?

--[University (Name), N\_Students]

SELECT School.Name, COUNT(DISTINCT Enroll.Student\_ID) AS N\_Students

FROM Enroll

INNER JOIN University ON University.University\_ID = Enroll.University\_ID

INNER JOIN School ON School.ID = University.University\_ID

GROUP BY School.Name;

***COUNT() –*** This is a function that returns the number of rows that match a specified condition.

***DISTINCT*** – This keyword is used to eliminate duplicate values when calculating the count.

***GROUP BY –*** This clause groups the result set by one or more columns. In this case, it groups the result set by the name of the school.

1. **Which students attended school the longest? [Students (Name), Number of Years]**

--2.3. Which students attended school the longest?

--[Students (Name), Number of Years]

SELECT People.FirstName AS Name, COUNT(DISTINCT Attend.Year\_) AS Number\_of\_Years

FROM Attend

INNER JOIN Student ON Attend.Student\_ID = Student.Student\_ID

INNER JOIN People ON Student.Student\_ID = People.ID

GROUP BY People.FirstName

ORDER BY Number\_of\_Years DESC;

***ORDER BY*** – This clause is used to sort the result set. In this case, it sorts the result set by the number of years in descending order, so the students who attended school the longest appear first.

1. **Which Teachers were Hired in the last 60 days with salaries below 1200€? [Teacher (Name), Date, Salary]**

--2.4. Which Teachers were Hired in the last 60 days with salaries below 1200€?

--[Teacher (Name), Date, Salary]

SELECT FirstName AS Name, LastName AS Surname, StartDate AS Date, Salary

FROM ToHire

INNER JOIN Teacher ON ToHire.Teacher\_ID = Teacher.Teacher\_ID

INNER JOIN People ON Teacher.Teacher\_ID = People.ID

WHERE StartDate >= DATEADD(day, -60, GETDATE()) AND Salary < 1200;

***DATEADD()*** – This is a function that adds or subtracts a specified time interval from a date.

***GETDATE()*** – This function returns the current system date and time.

1. **Which universities currently have courses with no students enrolled? [Universities (Name)] List them in alphabetical order.**

--2.5. Which universities currently have courses with no students enrolled?

--[Universities (Name)] List them in alphabetical order.

SELECT School.Name AS Name

FROM School

WHERE School.ID NOT IN (SELECT DISTINCT Enroll.University\_ID FROM Enroll)

ORDER BY Name ASC;

***NOT IN*** – This operator is used to negate the result of a subquery. In this case, it checks whether the ID of each school is not found in the result set returned by the operation.

***SELECT DISTINCT*** – This keyword is used to retrieve unique values from a column in a table.

**Exercise 3**

**Assuming that each Student has a cost of 50€/day for the school, create a procedure that for a given month calculates the Total Value that each school spent, presenting a table with the names of the schools and the total spent in each one in that month. The procedure must return the total amount spent by all the schools.**

CREATE PROCEDURE SchoolExpenses

@Month INTEGER,

@Year INTEGER

AS

BEGIN

DECLARE @TotalExpenses MONEY = 0;

-- Create a temporary table to hold the results

CREATE TABLE #SchoolExpenses (

SchoolName VARCHAR(50),

TotalSpent MONEY

);

DECLARE @DaysInMonth INTEGER;

-- Determine the number of days in the specified month

SET @DaysInMonth = CASE

WHEN @Month IN (1, 3, 5, 7, 8, 10, 12) THEN 31

WHEN @Month IN (4, 6, 9, 11) THEN 30

WHEN @Month = 2 THEN

CASE

WHEN @Year % 4 = 0 AND (@Year % 100 != 0 OR @Year % 400 = 0) THEN 29

ELSE 28

END

END;

-- Insert total expenses for each school into the temporary table

INSERT INTO #SchoolExpenses (SchoolName, TotalSpent)

SELECT

School.Name,

COUNT(Attend.Student\_ID) \* 50 \* @DaysInMonth AS TotalSpent

FROM

School

LEFT JOIN

Attend ON School.ID = Attend.HighSchool\_ID

AND YEAR(Attend.SchoolYear) = @Year

AND MONTH(Attend.SchoolYear) = @Month

GROUP BY

School.Name;

-- Calculate the total expenses across all schools

SELECT @TotalExpenses = SUM(TotalSpent) FROM #SchoolExpenses;

-- Return the detailed expenses for each school and the total expenses

SELECT \* FROM #SchoolExpenses;

SELECT @TotalExpenses AS TotalExpenses;

-- Clean up the temporary table

DROP TABLE #SchoolExpenses;

END;

-- Execute the procedure with example input

EXECUTE SchoolExpenses @Month = 2, @Year = 2020;

The SQL code defines procedure named "SchoolExpenses" responsible for calculating the total expenses that schools have to spend on students during specific month of the year.

It starts by declaring two input parameters, @Month and @Year.

Inside the procedure, it initializes a variable called @TotalExpenses to 0, and creates a temporary table named #SchoolExpenses with columns named for SchoolName and TotalSpent.

The procedure then calculates the number of days in the given month based on the provided month and year. How many of them depend on month, so there are possibilities: 31, 30, 29 or 28 days accordingly to not only month, but also to year.

The procedure is inserting the calculated expenses for each school into the temporary table, derived from the multiplication of 50 Euro per day for all days in the month. After summing up the total expenses stored in the temporary table, it outputs both the detailed expenses for each school and the overall total expenses.

The execution of the stored procedure is demonstrated with an example input of February 2020. Overall, this procedure efficiently computes and presents school expenses for a specified month and year.

**Exercise 4**

**A student cannot attend more than 2 Courses at the same time. Create a trigger that only lets you insert a new record in the Enroll relationship if this student is not yet enrolled in 2 courses.**

The provided code defines a trigger named EnrollLimit, which ensures that a student cannot enroll in more than two courses simultaneously. This trigger is activated instead of a standard insert operation on the Enroll table. Here's a detailed explanation of the trigger's functionality:

CREATE TRIGGER EnrollLimit

ON Enroll

INSTEAD OF INSERT

AS

BEGIN

DECLARE @StudentID INT;

DECLARE @CourseCount INT;

-- Check each student in the INSERTED table

DECLARE cur CURSOR FOR

SELECT Student\_ID FROM INSERTED;

OPEN cur;

FETCH NEXT FROM cur INTO @StudentID;

WHILE @@FETCH\_STATUS = 0

BEGIN

-- Count the number of currently enrolled courses for this student

SELECT @CourseCount = COUNT(\*)

FROM Enroll

WHERE Student\_ID = @StudentID;

-- Check if the student is already enrolled in 2 or more courses

IF @CourseCount >= 2

BEGIN

PRINT 'Cannot enroll in more than 2 courses at the same time for Student ID: ' + CAST(@StudentID AS VARCHAR);

END

ELSE

BEGIN

-- If the count is less than 2, insert the new record into Enroll

INSERT INTO Enroll (Student\_ID, University\_ID, Date\_, Course)

SELECT Student\_ID, University\_ID, Date\_, Course

FROM INSERTED

WHERE Student\_ID = @StudentID; -- Ensure we only insert for this student

END

FETCH NEXT FROM cur INTO @StudentID;

END

CLOSE cur;

DEALLOCATE cur;

END;

When the students try to enrol on the course the trigger is checking if they are already in two of them. This is done using an operation that counts the number of courses for the students specified in the **INSERTED** table.

The **COUNT(\*)** function is used to count the number of rows in the result set, which in this case is the number of courses the student is already enrolled in.

If the number of courses that student is enrolled to is 2 or more, an error message is displayed using the **PRINT** statement, and the insertion cannot be done.

In the case when the number of courses that student is enrolled to is less than 2, the new enrolment records from the **INSERTED** table are put into the Enroll table, so the trigger is allowing to enrol on the another course.