Міністерство освіти і науки України

ПВНЗ “Європейський університет”

Кафедра інформаційних систем та математичних дисциплін

КУРСОВА РОБОТА

на тему: “База даних працівників підприємства”

**Підготував:** студент ІІ курсу заочного відділу факультету інформаційних систем та технологій (група 221/1)

Іванов Антон Володимирович

**Перевірив:**

Київ 2018

СОДЕРЖАНИЕ

[ВСТУПЛЕНИЕ 3](#_Toc515958548)

АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

[ТРЕБОВАНИЯ 4](#_Toc515958549)

[СТРУКТУРА 5](#_Toc515958550)

[РЕАЛИЗАЦИЯ 7](#_Toc515958551)

[Создание структуры 7](#_Toc515958552)

[Создание хранимых процедур 9](#_Toc515958553)

[ТЕСТОВЫЙ ПРИМЕР 14](#_Toc515958554)

[ВЫВОДЫ 18](#_Toc515958555)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 19](#_Toc515958556)

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ (общие сведения о БД, постановка задачи???)
2. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ (что из себя представляет система)
3. Концептуальное (инфологическое) проектирование
4. Логическое (даталогическое) проектирование (модель даных)
5. Физическое проектирование

# **ВСТУПЛЕНИЕ**

В наши дни практически каждое предприятие имеет базу данных для учета распределения как человеческих, так и финансовых. Как правило – это реляционная база данных, представляющая собой множество таблиц, данные из которых можно запросить в разных представлениях без необходимости реструктурировать сами таблицы Стандартным же программным интерфейсом (API) для взаимодействия с реляционной базой данных, как для пользователя, так и для приложения является структурированный язык запросов (Structured Query Language - SQL).

Целью этой работы будет проектирование и реализация упрощенного варианта подобной базы данных для демонстрации ее основных возможностей.

# **ТРЕБОВАНИЯ**

Прежде чем начать проектировать БД, определимся с минимальным набором требований к ней. Это поможет нам более четко представить какой должна быть ее структура.

1. Ключевым логическим элементом БД должен быть «работник», о котором должны быть известны следующие сведения:
   1. Имя;
   2. Фамилия;
   3. Должность;
   4. Начальник;
   5. Проект;
   6. Месячный оклад;
   7. Период работы;
2. Предоставить возможность добавления нового работника.
3. Предоставить возможность изменения текущей должности, начальника, проекта и оклада работника. При этом должна поддерживаться история должностей, проектов и окладов.
4. Предоставить возможность увольнения работника с возможностью его дальнейшего восстановления. При этом должна поддерживаться история принятий и увольнений.
5. Предоставить возможность выборки работников по проекту, позиции, начальнику с тем, чтобы видеть распределение человеческих и финансовых ресурсов. Результат должен содержать:
   1. Название среза (проект, позиция, начальник);
   2. Количество людей;
   3. Финансовые расходы в месяц.
6. Предоставить возможность выборки истории по работнику.

# **СТРУКТУРА**

Теперь зная основные требования, мы можем перейти к определению структуры нашей базы данных.



Рис. 1 Структура базы данных работников

Как видно из диаграммы, центральной таблицей является “Employee”. В ней имеется колонка “EmployeeId” выполняющая роль первичного ключа (primary key), то есть позволяет уникально идентифицировать запись в таблице, а также задать порядок следования записей в таблице для быстрого поиска (в нашем случае создается clustered index). Также, из интересных колонок можно отметить “SuperiorId” являющееся, по сути, внешним ключом (foreign key), который позволяет установить отношение начальник-подчиненный внутри таблицы, причем отношение здесь один-ко-многим, так один начальник может быть связан со многими подчиненными, но не наоборот.

Следующая таблица “Position” содержит список доступных должностей внутри предприятия.

Так как работник в течении своей работы может иметь разные должности, то нам понадобилась таблица “EmployeePosition” для установления отношения много-ко-многим (many-to-many) между “Employee” и “Position”. Колонка “EffectiveFrom” в таблице “EmployeePosition” задает дату вступления в должность, в то время как “EmployeeTo” – дату прекращения нахождения в текущей должности. “EffectiveTo” поддерживает признак NULL, который как раз и определяет актуальность должности для работника.

Подобным образом определены таблицы “Project” и “EmployeeProject”, содержащие информацию об имеющихся проектах и истории проектов для работников соответственно.

Таблица “Salary” содержит информацию о текущей месячной зарплате по работнику и историю.

# **РЕАЛИЗАЦИЯ**

Теперь, имея структуру базы, перейдем к непосредственному написанию сценариев для ее конструирования.

Стоит заметить, что в этой работе используется в качестве сервера базы данных Microsoft SQL Server Express 2017 и в качестве приложения для управления базой данных Microsoft SQL Server Management Studio 17.7.

## **Создание структуры**

Вначале создадим саму базу “EmployeeDB”. Для этого выполним следующий запрос:

CREATE DATABASE EmployeesDB;

GO

Следующим шагом будет создание таблиц:

USE [EmployeesDB];

GO

-- Remove objects first

IF OBJECT\_ID('EmployeesDB.dbo.EmployeeProject') is not null

BEGIN

DROP TABLE [dbo].[EmployeeProject]

END

IF OBJECT\_ID('EmployeesDB.dbo.Project') is not null

BEGIN

DROP TABLE [dbo].[Project]

END

IF OBJECT\_ID('EmployeesDB.dbo.EmployeePosition') is not null

BEGIN

DROP TABLE [dbo].[EmployeePosition]

END

IF OBJECT\_ID('EmployeesDB.dbo.Position') is not null

BEGIN

DROP TABLE [dbo].[Position]

END

IF OBJECT\_ID('EmployeesDB.dbo.Salary') is not null

BEGIN

DROP TABLE [dbo].[Salary]

END

IF OBJECT\_ID('EmployeesDB.dbo.Employee') is not null

BEGIN

DROP TABLE [dbo].[Employee]

END

GO

CREATE TABLE [dbo].[Employee]

(

[EmployeeId] INT IDENTITY(1,1) PRIMARY KEY,

[FirstName] VARCHAR(40) NOT NULL,

[LastName] VARCHAR(40) NOT NULL,

[SuperiorId] INT NULL FOREIGN KEY REFERENCES [dbo].[Employee](EmployeeId),

[EffectiveFrom] DATE NOT NULL,

[EffectiveTo] DATE NULL,

CONSTRAINT UC\_Employee\_FLName UNIQUE (FirstName, LastName),

CONSTRAINT CK\_Employee\_EffectiveFrom\_To CHECK (EffectiveTo IS NULL OR EffectiveFrom <= EffectiveTo)

);

GO

CREATE TABLE [dbo].[Project]

(

[ProjectId] INT IDENTITY(1,1) PRIMARY KEY,

[ProjectName] VARCHAR(40) NOT NULL UNIQUE

);

GO

CREATE TABLE [dbo].[EmployeeProject]

(

[EmployeeId] INT NOT NULL FOREIGN KEY REFERENCES [dbo].[Employee](EmployeeId),

[ProjectId] INT NOT NULL FOREIGN KEY REFERENCES [dbo].[Project](ProjectId),

[EffectiveFrom] DATE NOT NULL,

[EffectiveTo] DATE NULL,

CONSTRAINT CK\_EmployeeProject\_EffectiveFrom\_To CHECK (EffectiveTo IS NULL OR EffectiveFrom <= EffectiveTo)

);

GO

CREATE TABLE [dbo].[Position]

(

[PositionId] INT IDENTITY(1,1) PRIMARY KEY,

[PositionName] VARCHAR(40) NOT NULL UNIQUE

);

GO

CREATE TABLE [dbo].[EmployeePosition]

(

[EmployeeId] INT NOT NULL FOREIGN KEY REFERENCES [dbo].[Employee](EmployeeId),

[PositionId] INT NOT NULL FOREIGN KEY REFERENCES [dbo].[Position](PositionId),

[EffectiveFrom] DATE NOT NULL,

[EffectiveTo] DATE NULL,

CONSTRAINT CK\_EmployeePosition\_EffectiveFrom\_To CHECK (EffectiveTo IS NULL OR EffectiveFrom <= EffectiveTo)

);

GO

CREATE TABLE [dbo].[Salary]

(

[EmployeeId] INT NOT NULL FOREIGN KEY REFERENCES [dbo].[Employee](EmployeeId),

[MonthlyPay] MONEY NOT NULL,

[EffectiveFrom] DATE NOT NULL,

[EffectiveTo] DATE NULL,

CONSTRAINT CK\_Salary\_EffectiveFrom\_To CHECK (EffectiveTo IS NULL OR EffectiveFrom <= EffectiveTo)

);

GO

## **Создание хранимых процедур**

Согласно требованиям мы должны уметь добавлять новых работников. Для этого реализуем хранимую процедуру (stored-procedure) “*dbo.AddEmployee*”:

USE [EmployeesDB];

GO

IF OBJECT\_ID('[dbo].[AddEmployee]', 'P') IS NOT NULL

BEGIN

DROP PROCEDURE [dbo].[AddEmployee]

END

GO

CREATE PROCEDURE [dbo].[AddEmployee]

@FirstName VARCHAR(40),

@LastName VARCHAR(40),

@PositionId INT,

@ProjectId INT,

@SuperiorId INT = NULL,

@MonthlyPay MONEY,

@EffectiveFrom DATE = NULL, -- Use current date if it's null

@EmployeeId INT = NULL OUT

AS

SET NOCOUNT ON; -- Don't report the number of processed row to client

SET XACT\_ABORT ON; -- Enable transaction aborting on constraint failure

IF @EffectiveFrom IS NULL

BEGIN

SET @EffectiveFrom = GETDATE()

END

BEGIN TRY

--Use transactions to guarantee atomic insertion

BEGIN TRAN

INSERT INTO dbo.Employee (FirstName, LastName, SuperiorId, EffectiveFrom)

VALUES (@FirstName, @LastName, @SuperiorId, @EffectiveFrom)

SET @EmployeeId = @@IDENTITY

--NB. Error should be reported if the position id is invalid

INSERT INTO dbo.EmployeePosition (EmployeeId, PositionId, EffectiveFrom)

VALUES (@EmployeeId, @PositionId, @EffectiveFrom)

--NB. Error should be reported if the project id is invalid

INSERT INTO dbo.EmployeeProject(EmployeeId, ProjectId, EffectiveFrom)

VALUES (@EmployeeId, @ProjectId, @EffectiveFrom)

INSERT INTO dbo.Salary(EmployeeId, MonthlyPay, EffectiveFrom)

VALUES (@EmployeeId, @MonthlyPay, @EffectiveFrom)

COMMIT TRAN

END TRY

BEGIN CATCH

IF (XACT\_STATE()) = -1

ROLLBACK;

THROW;

END CATCH

GO

Основной особенностью этой процедуры и ниже является использование транзакций для гарантирования целостности данных, в случае если произойдет ошибка во время вставки/обновления в одну/одной из таблиц. Примером ошибки может быть нарушения уникальности ограничений имени и фамилии работника или недействительный идентификатор начальника.

Следующей процедурой будет “*dbo.UpdateEmployee*”, используемой для обновления одного или несколько полей работника:

USE [EmployeesDB];

GO

IF OBJECT\_ID('[dbo].[UpdateEmployee]', 'P') IS NOT NULL

BEGIN

DROP PROCEDURE [dbo].[UpdateEmployee]

END

GO

CREATE PROCEDURE [dbo].[UpdateEmployee]

@EmployeeId INT,

@FirstName VARCHAR(40) = NULL,

@LastName VARCHAR(40) = NULL,

@PositionId INT = NULL,

@ProjectId INT = NULL,

@MonthlyPay MONEY = NULL,

@EffectiveFrom DATE = NULL -- Use current date if it's null. It's expected to be greater than currently used for the active record.

-- Ideally, I'd test it but I won't do it purposefully to minimize checks

AS

SET NOCOUNT ON; -- Don't report the number of processed row to client

SET XACT\_ABORT ON; -- Enable transaction aborting on constraint failure

IF @EffectiveFrom IS NULL

BEGIN

SET @EffectiveFrom = DATEADD(DAY, 1, GETDATE())

END

BEGIN TRY

--Use transactions to guarantee atomic update

BEGIN TRAN

IF @EmployeeId IS NULL

RAISERROR('@EmployeeId can''t be NULL', 11, 1)

IF (@FirstName IS NOT NULL AND @FirstName != (SELECT FirstName FROM dbo.Employee WHERE EmployeeId = @EmployeeId))

UPDATE dbo.Employee SET FirstName = @FirstName WHERE EmployeeId = @EmployeeId

IF (@LastName IS NOT NULL AND @LastName != (SELECT LastName FROM dbo.Employee WHERE EmployeeId = @EmployeeId))

UPDATE dbo.Employee SET LastName = @LastName WHERE EmployeeId = @EmployeeId

IF (@PositionId IS NOT NULL AND @PositionId != (SELECT PositionId FROM dbo.EmployeePosition WHERE EmployeeId=@EmployeeId AND EffectiveTo IS NULL))

BEGIN

-- Close the current position

UPDATE dbo.EmployeePosition SET EffectiveTo = DATEADD(DAY, -1, @EffectiveFrom)

WHERE EmployeeId = @EmployeeId AND EffectiveTo IS NULL

-- Create a new one

INSERT INTO dbo.EmployeePosition (EmployeeId, PositionId, EffectiveFrom)

VALUES (@EmployeeId, @PositionId, @EffectiveFrom)

END

IF (@ProjectId IS NOT NULL AND @ProjectId != (SELECT ProjectId FROM dbo.EmployeeProject WHERE EmployeeId=@EmployeeId AND EffectiveTo IS NULL))

BEGIN

-- Close the current project

UPDATE dbo.EmployeeProject SET EffectiveTo = DATEADD(DAY, -1, @EffectiveFrom)

WHERE EmployeeId = @EmployeeId AND EffectiveTo IS NULL

-- Create a new one

INSERT INTO dbo.EmployeeProject (EmployeeId, ProjectId, EffectiveFrom)

VALUES (@EmployeeId, @ProjectId, @EffectiveFrom)

END

IF (@MonthlyPay IS NOT NULL AND @MonthlyPay != (SELECT MonthlyPay FROM dbo.Salary WHERE EmployeeId=@EmployeeId AND EffectiveTo IS NULL))

BEGIN

-- Close the current salary

UPDATE dbo.Salary SET EffectiveTo = DATEADD(DAY, -1, @EffectiveFrom)

WHERE EmployeeId = @EmployeeId AND EffectiveTo IS NULL

-- Create a new one

INSERT INTO dbo.Salary (EmployeeId, MonthlyPay, EffectiveFrom)

VALUES (@EmployeeId, @MonthlyPay, @EffectiveFrom)

END

COMMIT TRAN

END TRY

BEGIN CATCH

IF (XACT\_STATE()) = -1

ROLLBACK;

THROW;

END CATCH

GO

Стоит отметить, что эта процедура не поддерживает изменения начальника, так как “*SuperiorId*” может не иметь значение, т.е. быть *NULL* и поэтому не вписывается в ее интерфейс. Для этих целей мы реализуем другую процедуру “*dbo.UpdateEmployeeSuperior*”:

USE [EmployeesDB];

GO

IF OBJECT\_ID('[dbo].[UpdateEmployeeSuperior]', 'P') IS NOT NULL

BEGIN

DROP PROCEDURE [dbo].[UpdateEmployeeSuperior]

END

GO

CREATE PROCEDURE [dbo].[UpdateEmployeeSuperior]

@EmployeeId INT,

@SuperiorId INT

AS

SET NOCOUNT ON; -- Don't report the number of processed row to client

IF @EmployeeId IS NULL

RAISERROR('@EmployeeId can''t be NULL', 11, 1)

IF (ISNULL(@SuperiorId, 0) != ISNULL((SELECT SuperiorId FROM dbo.Employee WHERE EmployeeId = @EmployeeId), 0))

UPDATE dbo.Employee SET SuperiorId = @SuperiorId WHERE EmployeeId = @EmployeeId

GO

Здесь мы используем системную функцию *ISNULL* для замены *NULL* на 0 при сравнении текущего и нового начальника, так как по стандарту *NULL* <> *NULL*. Мы можем безопасно использовать 0, так как идентификатор работника начинается с 1.

И последней хранимой процедурой у нас будет “*dbo.DismissEmployee*” для увольнения работника:

USE [EmployeesDB];

GO

IF OBJECT\_ID('[dbo].[DismissEmployee]', 'P') IS NOT NULL

BEGIN

DROP PROCEDURE [dbo].[DismissEmployee]

END

GO

CREATE PROCEDURE [dbo].[DismissEmployee]

@EmployeeId INT,

@EffectiveTo DATE = NULL -- Use current date if it's null

AS

SET NOCOUNT ON; -- Don't report the number of processed row to client

SET XACT\_ABORT ON; -- Enable transaction aborting on constraint failure

IF @EffectiveTo IS NULL

BEGIN

SET @EffectiveTo = GETDATE()

END

BEGIN TRY

--Use transactions to guarantee atomic update

BEGIN TRAN

IF @EmployeeId IS NULL

RAISERROR('@EmployeeId can''t be NULL', 11, 1)

-- Close the current position

UPDATE dbo.EmployeePosition SET EffectiveTo = @EffectiveTo

WHERE EmployeeId = @EmployeeId AND EffectiveTo IS NULL

-- Close the current project

UPDATE dbo.EmployeeProject SET EffectiveTo = @EffectiveTo

WHERE EmployeeId = @EmployeeId AND EffectiveTo IS NULL

-- Close the current salary

UPDATE dbo.Salary SET EffectiveTo = @EffectiveTo

WHERE EmployeeId = @EmployeeId AND EffectiveTo IS NULL

-- Close the current work period

UPDATE dbo.Employee SET EffectiveTo = @EffectiveTo

WHERE EmployeeId = @EmployeeId AND EffectiveTo IS NULL

COMMIT TRAN

END TRY

BEGIN CATCH

IF (XACT\_STATE()) = -1

ROLLBACK;

THROW;

END CATCH

GO

Задача этой процедуры закрыть текущие позицию, проект, зарплату и рабочий период. Единственный способ нанять уволенного работника снова при текущей реализации нашей базы – это добавить его заново посредством “*dbo.AddEmployee*”.

# **ТЕСТОВЫЙ ПРИМЕР**

Теперь имея все необходимые объекты, мы можем проверить их, написав небольшой сценарий.

Для начала заполним таблицы “*dbo.Position*” и “*dbo.Project*” данными:

DELETE FROM [dbo].[Position]

INSERT INTO [dbo].[Position] (PositionName) VALUES

('Software Developer'),

('QA Test Engineer'),

('Business Analyst'),

('Project Manager'),

('Chief Executive Officer')

DELETE FROM [dbo].[Project]

INSERT INTO [dbo].[Project] (ProjectName) VALUES

('Project A'),

('Project B'),

('Company Management')

GO

Дальше сделаем тоже самое для остальных таблиц, но используя наши хранимые процедуры:

DECLARE @CEOId INT

exec dbo.AddEmployee 'Donald', 'Trump', 5 /\*CEO\*/, 3 /\*Comp. Mgmt\*/, default, 10000, '1 Jan 2015', @CEOId OUT

-- Project A resources

DECLARE @PMAId INT

exec dbo.AddEmployee 'Bill', 'Gates', 4 /\*PM\*/, 1 /\*Proj. A\*/, @CEOId, 5000, '1 Feb 2015', @PMAId OUT

exec dbo.AddEmployee 'Mickey', 'Mouse', 1 /\*SD\*/, 1 /\*Proj. A\*/, @PMAId, 3000, '15 Feb 2015'

DECLARE @DonaldDuckId INT

exec dbo.AddEmployee 'Donald', 'Duck', 2 /\*QA\*/, 1 /\*Proj. A\*/, @PMAId, 2000, '20 Feb 2015', @DonaldDuckId OUT

exec dbo.AddEmployee 'Goofy', 'Wolf', 3 /\*BA\*/, 1 /\*Proj. A\*/, @PMAId, 3500, '25 Feb 2015'

exec dbo.AddEmployee 'Daisy', 'Mouse', 1 /\*SD\*/, 1 /\*Proj. A\*/, @PMAId, 3000, '20 Mar 2015'

-- Project B resources

DECLARE @PMBId INT

exec dbo.AddEmployee 'Jackie', 'Chan', 4 /\*PM\*/, 2 /\*Proj. B\*/, @CEOId, 6000, '1 Jul 2016', @PMBId OUT

exec dbo.AddEmployee 'Samo', 'Hung', 1 /\*SD\*/, 2 /\*Proj. B\*/, @PMBId, 2000, '23 Jul 2016'

exec dbo.AddEmployee 'Bruce', 'Lee', 2 /\*SD\*/, 2 /\*Proj. B\*/, @PMBId, 4000, '1 Sep 2016'

exec dbo.AddEmployee 'Chuck', 'Noris', 2 /\*QA\*/, 2 /\*Proj. B\*/, @PMBId, 2000, '5 Sep 2016'

exec dbo.AddEmployee 'Jet', 'Lee', 3 /\*BA\*/, 2 /\*Proj. B\*/, @PMBId, 1000, '1 Dec 2016'

exec dbo.AddEmployee 'Panda', 'Kung Fu', 3 /\*BA\*/, 2 /\*Proj. B\*/, @PMBId, 5000, '1 Apr 2017'

-- Create career for Donald Duck

exec dbo.UpdateEmployee @DonaldDuckId, @MonthlyPay = 2100, @EffectiveFrom = '20 May 2015'

exec dbo.UpdateEmployee @DonaldDuckId, @PositionId = 1 /\*SD\*/, @EffectiveFrom = '20 Aug 2015'

exec dbo.UpdateEmployee @DonaldDuckId, @MonthlyPay = 2500, @EffectiveFrom = '1 Nov 2015'

exec dbo.UpdateEmployee @DonaldDuckId, @ProjectId = 2, @EffectiveFrom = '2 Jul 2016'

exec dbo.UpdateEmployeeSuperior @DonaldDuckId, @PMBId

exec dbo.DismissEmployee @DonaldDuckId, '1 Mar 2017'

GO

Теперь напишем запрос для получения используемых человечески и финансовых ресурсов по проекту:

-- SELECT BY PROJECT

SELECT

proj.ProjectName,

COUNT(empProj.EmployeeId) as PeopleCount,

SUM(sal.MonthlyPay) as TotalMonthlyPay

FROM dbo.EmployeeProject empProj

JOIN dbo.Project proj on empProj.ProjectId = proj.ProjectId

JOIN dbo.Salary sal on empProj.EmployeeId = sal.EmployeeId and sal.EffectiveTo is NULL

JOIN dbo.Employee emp on empProj.EmployeeId = emp.EmployeeId and emp.EffectiveTo is NULL

WHERE

empProj.EffectiveTo is NULL

GROUP BY proj.ProjectName

ORDER BY proj.ProjectName

|  |  |  |
| --- | --- | --- |
| ProjectName | PeopleCount | TotalMonthlyPay |
| Company Management | 1 | 10000 |
| Project A | 4 | 14500 |
| Project B | 6 | 20000 |

По позиции:

-- SELECT BY POSITION

SELECT

pos.PositionName,

COUNT(empPos.EmployeeId) as PeopleCount,

SUM(sal.MonthlyPay) as TotalMonthlyPay

FROM dbo.EmployeePosition empPos

JOIN dbo.Position pos on empPos.PositionId = pos.PositionId

JOIN dbo.Salary sal on empPos.EmployeeId = sal.EmployeeId and sal.EffectiveTo is NULL

JOIN dbo.Employee emp on empPos.EmployeeId = emp.EmployeeId and emp.EffectiveTo is NULL

WHERE

empPos.EffectiveTo is NULL

GROUP BY pos.PositionName

ORDER BY pos.PositionName

|  |  |  |
| --- | --- | --- |
| PositionName | PeopleCount | TotalMonthlyPay |
| Business Analyst | 3 | 9500 |
| Chief Executive Officer | 1 | 10000 |
| Project Manager | 2 | 11000 |
| QA Test Engineer | 2 | 6000 |
| Software Developer | 3 | 8000 |

По начальнику, со всеми подчиненными, как прямыми, так и не прямыми:

-- SELECT GRAND TOTAL BY A SUPERIOR

DECLARE @SuperiorId INT

SELECT @SuperiorId = EmployeeId from dbo.Employee WHERE FirstName= 'Donald' and LastName ='Trump'

;WITH cte\_Subordinates (EmployeeId)

AS

(

SELECT @SuperiorId

UNION ALL

SELECT emp.EmployeeId

FROM cte\_Subordinates sub

JOIN dbo.Employee emp ON sub.EmployeeId = emp.SuperiorId

)

SELECT

COUNT(sub.EmployeeId) as SubordinateCount,

ISNULL(SUM(sal.MonthlyPay), 0) as TotalMonthlyPay

FROM cte\_Subordinates sub

JOIN dbo.Salary sal on sub.EmployeeId = sal.EmployeeId and sal.EffectiveTo is NULL

JOIN dbo.Employee subEmp on sub.EmployeeId = subEmp.EmployeeId and subEmp.EffectiveTo is NULL

WHERE sub.EmployeeId != @SuperiorId

|  |  |
| --- | --- |
| SubordinateCount | TotalMonthlyPay |
| 10 | 34500 |

И напоследок напишем несколько запросов для просматривания истории по работнику.

История проектов:

DECLARE @EmployeeId INT

SELECT @EmployeeId = EmployeeId from dbo.Employee WHERE FirstName= 'Donald' and LastName ='Duck'

SELECT

sproj.ProjectName,

proj.EffectiveFrom,

proj.EffectiveTo

FROM dbo.EmployeeProject proj

JOIN dbo.Project sproj ON proj.ProjectId = sproj.ProjectId

WHERE proj.EmployeeId = @EmployeeId

|  |  |  |
| --- | --- | --- |
| ProjectName | EffectiveFrom | EffectiveTo |
| Project A | 20.02.2015 | 01.07.2016 |
| Project B | 02.07.2016 | 01.03.2017 |

История должностей:

SELECT

spos.PositionName,

pos.EffectiveFrom,

pos.EffectiveTo

FROM dbo.EmployeePosition pos

JOIN dbo.Position spos ON pos.PositionId = spos.PositionId

WHERE pos.EmployeeId = @EmployeeId

|  |  |  |
| --- | --- | --- |
| PositionName | EffectiveFrom | EffectiveTo |
| QA Test Engineer | 20.02.2015 | 19.08.2015 |
| Software Developer | 20.08.2015 | 01.03.2017 |

История зарплат:

SELECT

sal.MonthlyPay,

sal.EffectiveFrom,

sal.EffectiveTo

FROM dbo.Salary sal

WHERE sal.EmployeeId = @EmployeeId

|  |  |  |
| --- | --- | --- |
| MonthlyPay | EffectiveFrom | EffectiveTo |
| 2000 | 20.02.2015 | 19.05.2015 |
| 2100 | 20.05.2015 | 31.10.2015 |
| 2500 | 01.11.2015 | 01.03.2017 |

# **ВЫВОДЫ**

Таким образом мы сконструировали схему базы данных, реализовали минимальный набор к требованиям к ней и продемонстрировали ее в работе.

Потенциальных улучшениями могут быть:

* Комплексная выборка истории по работнику, в которой бы присутствовали проект, позиция и зарплата с указанием периодов;
* Возможность восстановления работника без необходимости заводить для него новую запись в “dbo.Employee ” таблице;
* Добавления отсутствующих ограничений;
* Более тщательная проверка на ошибки;
* Оптимизация запросов.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

|  |  |
| --- | --- |
| [1] | https://docs.microsoft.com/en-us/sql/t-sql/language-reference?view=sql-server-2017 |
| [2] | https://www.google.com.ua/ |
|  |  |
|  |  |