**Министерство образования Республики Беларусь**

**Учреждение образования**

**«Белорусский государственный университет информатики и радиоэлектроники»**

Факультет непрерывного и дистанционного обучения

Специальность «ИиТП»

**КОНТРОЛЬНАЯ РАБОТА № 1**

по предмету «Технологии разработки программного обеспечения»

по теме «Проектирование и создание БД»

Студент дистанционной формы

обучения 5 курса

Группы № 493551

Авхимович Алексей Валерьевич

г.Борисов, ул. Чапаева, 29/116

Тел. +375-44-7900085

Минск 2019

Содержание

[1 Постановка задачи 4](#_Toc484595212)

[2 Разработка логической модели данных 5](#_Toc484595213)

[3 Разработка физической модели данных 8](#_Toc484595214)

[Приложение А DDL-скрипт генерации базы данных 13](#_Toc484595215)

# Постановка задачи

Разработка логической и физической модели БД на языке IDEF1.X не менее чем с 6-ю сущностями, генерация DDL скриптов, создание БД, получение отчетов.

Предметная область: Гостиница. Отдел клиентского сервиса. Процесс бронирования и сдачи номеров.

При выполнении работы использовать ERwin Data Modeler.

Тип СУБД – Microsoft SQL Server.

# Разработка логической модели данных

Проведенный анализ предметной области позволил выявить следующий набор сущностей:

* виды движений постояльцев (Actions);
* движения постояльцев (Moves);
* роли пользователей (Roles);
* гостиничные номера (Rooms);
* типы гостиничных номеров (Types);
* пользователи (Users).

Схема данных в нотации IDEF1.X – логическая ER-модель, приведенная на рисунке 2.1, показывает сущности, их атрибуты, связи между сущностями, первичные и внешние ключи.



Рисунок . – Схема логической модели данных

Между сущностями установлены следующие отношения:

* пользователи относятся к одной роли, к одной роли могут относиться много пользователей (1 : М);
* гостиничные номера относятся к одному типу, одному типу могут соответствовать много номеров (1 : М);
* движение относится к определенному виду, одному виду могут соответствовать много движений (1 : М);
* движение относится к одному пользователю, одному пользователю могут соответствовать много движений (1 : М);
* движение относится к одному гостиничному номеру, одному гостиничному номеру могут соответствовать много движений (1 : М).

Проверка перечисленных объектов на соответствие различным нормальным формам позволила сделать вывод о нахождении отношений в третьей нормальной форме.

Рассмотрим атрибуты сущностей.

Сущность «Роли» (Roles) включает следующие атрибуты:

* код роли (RoleId) – уникальный идентификатор роли, первичный ключ;
* наименование (Name) – обозначение роли.

Сущность «Пользователи» (Users) включает следующие атрибуты:

* код пользователя (UserId) – уникальный идентификатор пользователя, первичный ключ;
* наименование (Name) – имя пользователя;
* пароль (Password) – пароль пользователя для входа в систему;
* код роли (RoleId) – ссылка на идентификатор роли, внешний ключ.

Сущность «Типы номеров» (Types) включает следующие атрибуты:

* код типа (TypeId) – уникальный идентификатор типа, первичный ключ;
* обозначение (Code) – международный код типа гостиничного номера;
* наименование (Name) – имя типа на русском языке.

Сущность «Гостиничные номера» (Rooms) включает следующие атрибуты:

* код номера (RoomId) – уникальный идентификатор номера, первичный ключ;
* этаж (Floor) – номер этажа, на котором находится номер;
* номер на этаже (Number) – порядковый номер комнаты на этаже;
* код типа (TypeId) – ссылка на идентификатор типа номера, внешний ключ;
* максимальное количество человек (PersonsMax) – максимально доступное количество постояльцев в номере.

Сущность «Виды движений» (Actions) включает следующие атрибуты:

* код вида движения (ActionId) – уникальный идентификатор вида движения, первичный ключ;
* наименование (Name) – обозначение движения.

Сущность «Движения постояльцев» (Moves) включает следующие атрибуты:

* код движения (MoveId) – уникальный идентификатор движения, первичный ключ;
* дата (Dates) – дата регистрации движения;
* код вида движения (ActionId) – ссылка на идентификатор вида движения, внешний ключ;
* код номера (RoomId) – ссылка на идентификатор номера, внешний ключ;
* дата заселения (DatesIn) – дата заселения постояльцев в номер;
* количество дней (DaysCnt) – количество дней проживания;
* количество человек (PersonsCnt) – количество постояльцев;
* код пользователя (UserId) – ссылка на идентификатор пользователя, внешний ключ;
* дата выселения (DatesOut).

# Разработка физической модели данных

Для осуществления генерации базы данных построим физическую модель, указав при этом связи, типы и размерности атрибутов.

Такая модель представлена на рисунке 3.1.



Рисунок . – Схема физической модели данных

Построенная физическая модель данных получена путем проецирования логической модели на СУБД Microsoft SQL.

К таблице Rooms добавим два триггера, отвечающие за формирование уникального кода – цифрового обозначения гостиничного номера по формуле:

Floor \* 100 + Number

где Floor – номер этажа;

Number – порядковый номер комнаты на этаже.

Первый триггер выполняется при вставке записи, заменяя собой соответствующую процедуру:

CREATE TRIGGER [dbo].[Trigger\_Rooms\_Insert\_For\_Id]

ON [dbo].[Rooms]

INSTEAD OF INSERT

AS

DECLARE @R\_Id int, @R\_Floor int, @R\_Number int, @R\_TypeId int, @R\_Persons int;

DECLARE InsCursor CURSOR FOR Select Floor, Number, TypeId, PersonsMax FROM Inserted;

BEGIN

OPEN InsCursor;

FETCH NEXT FROM InsCursor INTO @R\_Floor, @R\_Number, @R\_TypeId, @R\_Persons;

WHILE @@FETCH\_STATUS = 0

BEGIN

SET @R\_Id = @R\_Floor \* 100 + @R\_Number;

INSERT INTO Rooms (Id, Floor, Number, TypeId, PersonsMax) VALUES (@R\_Id, @R\_Floor, @R\_Number, @R\_TypeId, @R\_Persons)

FETCH NEXT FROM InsCursor INTO @R\_Floor, @R\_Number, @R\_TypeId, @R\_Persons;

END;

CLOSE InsCursor;

DEALLOCATE InsCursor;

END

Второй триггер выполняется после обновления записи:

CREATE TRIGGER [dbo].[Trigger\_Rooms\_Update\_For\_Id]

ON [dbo].[Rooms]

AFTER UPDATE

AS

DECLARE @R\_Floor int, @R\_Number int;

BEGIN

SELECT @R\_Floor = Floor, @R\_Number = Number FROM inserted

UPDATE Rooms SET Id = @R\_Floor \* 100 + @R\_Number

WHERE (Floor = @R\_Floor AND Number = @R\_Number)

END

К таблице Moves добавим триггер, проверяющий значение поля PersonsCnt (количество постояльцев) и приводящий его в соответствие с максимально допустимым количеством постояльцев, указанным в таблице Rooms:

CREATE TRIGGER [dbo].[Trigger\_Moves\_PersonsCnt]

ON [dbo].[Moves]

AFTER INSERT, UPDATE

AS

DECLARE @P\_Max int, @P\_Current int, @M\_Id int, @M\_RoomId int;

BEGIN

Select @P\_Current = PersonsCnt, @M\_Id = Id, @M\_RoomId = RoomId FROM inserted;

Select @P\_Max = PersonsMax FROM Rooms WHERE Id = @M\_RoomId;

IF (@P\_Current > @P\_Max) UPDATE Moves SET PersonsCnt = @P\_Max WHERE Id = @M\_Id;

END

Триггер выполняется после операций вставки и обновления записей таблицы Moves.

Кроме таблиц базы данных к физической модели добавлены три представления:

* MovesView;
* UsersView;
* RoomsView.

Представление UsersView включает данные пользователей, собираемые из таблиц Users и Roles. Данные объединяются с помощью внутреннего соединения (Inner Join):

CREATE VIEW [dbo].[UsersView]

AS SELECT Users.Id, Users.Name, Users.Password, Users.RoleId, Roles.Name AS RoleName

FROM Users INNER JOIN Roles ON Users.RoleId = Roles.Id

Представление RoomsView включает данные гостиничных номеров, собираемые из таблиц Rooms и Types. Данные объединяются с помощью внутреннего соединения (Inner Join):

CREATE VIEW [dbo].[RoomsView]

AS SELECT Rooms.Id, Rooms.Floor, Rooms.Number, Rooms.TypeId, Types.Code, Rooms.PersonsMax

FROM Rooms INNER JOIN Types ON Rooms.TypeId = Types.Id

Представление MovesView включает данные движения постояльцев, собираемые из таблиц Moves, Actions и Users. Данные объединяются с помощью внутреннего соединения (Inner Join):

CREATE VIEW [dbo].[MovesView]

AS SELECT Moves.Id, Moves.Dates, Moves.ActionId, Actions.Name AS ActionName, Moves.RoomId, Moves.DatesIn, Moves.DaysCnt, Moves.DatesOut, Moves.PersonsCnt, Moves.UserId, Users.Name AS UserName

FROM Moves INNER JOIN Actions ON Moves.ActionId = Actions.Id

INNER JOIN Users ON Moves.UserId = Users.Id

Далее разработаем хранимые процедуры, реализующие добавление, обновление и удаление записей для таблиц Moves и Rooms.

Хранимая процедура MoveDelete реализует удаление записи гостиничного номера:

CREATE PROCEDURE [dbo].MoveDelete

(

@Id int

)

AS

SET NOCOUNT OFF;

DELETE FROM Moves

WHERE (Id = @Id)

Хранимая процедура MoveInsert реализует добавление записи гостиничного номера:

CREATE PROCEDURE [dbo].MoveInsert

(

@Dates date,

@ActionId int,

@RoomId int,

@DatesIn date,

@DaysCnt int,

@PersonsCnt int,

@UserId int

)

AS

SET NOCOUNT OFF;

INSERT INTO Moves (Dates, ActionId, RoomId, DatesIn, DaysCnt, PersonsCnt, UserId)

VALUES (@Dates, @ActionId, @RoomId, @DatesIn, @DaysCnt, @PersonsCnt, @UserId)

Хранимая процедура MoveUpdate реализует обновление записи гостиничного номера:

CREATE PROCEDURE [dbo].MoveUpdate

(

@Dates date,

@ActionId int,

@RoomId int,

@DatesIn date,

@DaysCnt int,

@PersonsCnt int,

@Id int

)

AS

SET NOCOUNT OFF;

UPDATE Moves

SET Dates = @Dates, ActionId = @ActionId, RoomId = @RoomId, DatesIn = @DatesIn, DaysCnt = @DaysCnt, PersonsCnt = @PersonsCnt

WHERE (Id = @Id)

Аналогично реализуем хранимые процедуры для таблицы Rooms:

* RoomDelete – удаление записи гостиничного номера;
* RoomInsert – добавление записи гостиничного номера;
* RoomUpdate – обновление гостиничного номера.

Далее выполним генерацию DDL и формирование отчета.

1. DDL-скрипт генерации базы данных

CREATE TABLE Actions

(

ActionId integer NOT NULL ,

Name nvarchar(20) NOT NULL

)

go

ALTER TABLE Actions

ADD CONSTRAINT XPKActions PRIMARY KEY CLUSTERED (ActionId ASC)

go

CREATE TABLE Moves

(

MoveId integer NOT NULL ,

Dates datetime NOT NULL ,

ActionId integer NOT NULL ,

RoomId integer NOT NULL ,

DatesIn datetime NOT NULL ,

DaysCnt integer NOT NULL ,

PersonsCnt integer NOT NULL ,

UserId integer NOT NULL ,

DatesOut datetime NULL

)

go

ALTER TABLE Moves

ADD CONSTRAINT XPKMoves PRIMARY KEY CLUSTERED (MoveId ASC)

go

CREATE TABLE Roles

(

RoleId integer NOT NULL ,

Name nvarchar(20) NOT NULL

)

go

ALTER TABLE Roles

ADD CONSTRAINT XPKRoles PRIMARY KEY CLUSTERED (RoleId ASC)

go

CREATE TABLE Rooms

(

RoomId integer NOT NULL ,

Floor integer NOT NULL ,

Number integer NOT NULL ,

PersonsMax integer NOT NULL ,

TypeId integer NOT NULL

)

go

ALTER TABLE Rooms

ADD CONSTRAINT XPKRooms PRIMARY KEY CLUSTERED (RoomId ASC)

go

CREATE TABLE Types

(

TypeId integer NOT NULL ,

Code nvarchar(20) NOT NULL ,

Name nvarchar(50) NOT NULL

)

go

ALTER TABLE Types

ADD CONSTRAINT XPKTypes PRIMARY KEY CLUSTERED (TypeId ASC)

go

CREATE TABLE Users

(

UserId integer NOT NULL ,

Name nvarchar(50) NOT NULL ,

Password nvarchar(50) NOT NULL ,

RoleId integer NOT NULL

)

go

ALTER TABLE Users

ADD CONSTRAINT XPKUsers PRIMARY KEY CLUSTERED (UserId ASC)

go

CREATE VIEW [dbo].[MovesView]

AS SELECT Moves.Id, Moves.Dates, Moves.ActionId, Actions.Name AS ActionName, Moves.RoomId, Moves.DatesIn, Moves.DaysCnt, Moves.DatesOut, Moves.PersonsCnt, Moves.UserId, Users.Name AS UserName

FROM Moves INNER JOIN Actions ON Moves.ActionId = Actions.Id

INNER JOIN Users ON Moves.UserId = Users.Id

go

CREATE VIEW [dbo].[RoomsView]

AS SELECT Rooms.Id, Rooms.Floor, Rooms.Number, Rooms.TypeId, Types.Code, Rooms.PersonsMax

FROM Rooms INNER JOIN Types ON Rooms.TypeId = Types.Id

go

CREATE VIEW [dbo].[UsersView]

AS SELECT Users.Id, Users.Name, Users.Password, Users.RoleId, Roles.Name AS RoleName

FROM Users INNER JOIN Roles ON Users.RoleId = Roles.Id

go

ALTER TABLE Moves

ADD CONSTRAINT R\_8 FOREIGN KEY (ActionId) REFERENCES Actions(ActionId)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE Moves

ADD CONSTRAINT R\_9 FOREIGN KEY (RoomId) REFERENCES Rooms(RoomId)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE Moves

ADD CONSTRAINT R\_10 FOREIGN KEY (UserId) REFERENCES Users(UserId)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE Rooms

ADD CONSTRAINT R\_7 FOREIGN KEY (TypeId) REFERENCES Types(TypeId)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE Users

ADD CONSTRAINT R\_6 FOREIGN KEY (RoleId) REFERENCES Roles(RoleId)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

CREATE PROCEDURE [dbo].MoveDelete

(

@Id int

)

AS

SET NOCOUNT OFF;

DELETE FROM Moves

WHERE (Id = @Id)

go

CREATE PROCEDURE [dbo].RoomInsert

(

@Floor int,

@Number int,

@TypeId int,

@PersonsMax int

)

AS

SET NOCOUNT OFF;

INSERT INTO Rooms

(Floor, Number, TypeId, PersonsMax)

VALUES (@Floor,@Number,@TypeId,@PersonsMax)

SET NOCOUNT OFF;

UPDATE Rooms

SET Floor = @Floor, Number = @Number, TypeId = @TypeId, PersonsMax = @PersonsMax

WHERE (Id = @Id)

go

CREATE PROCEDURE RoomInsert

AS

go

CREATE PROCEDURE [dbo].RoomDelete

(

@Id int

)

AS

SET NOCOUNT OFF;

DELETE FROM Rooms

WHERE (Id = @Id)

go

CREATE PROCEDURE [dbo].MoveUpdate

(

@Dates date,

@ActionId int,

@RoomId int,

@DatesIn date,

@DaysCnt int,

@PersonsCnt int,

@Id int

)

AS

SET NOCOUNT OFF;

UPDATE Moves

SET Dates = @Dates, ActionId = @ActionId, RoomId = @RoomId, DatesIn = @DatesIn, DaysCnt = @DaysCnt, PersonsCnt = @PersonsCnt

WHERE (Id = @Id)

go

CREATE PROCEDURE [dbo].MoveInsert

(

@Dates date,

@ActionId int,

@RoomId int,

@DatesIn date,

@DaysCnt int,

@PersonsCnt int,

@UserId int

)

AS

SET NOCOUNT OFF;

INSERT INTO Moves (Dates, ActionId, RoomId, DatesIn, DaysCnt, PersonsCnt, UserId)

VALUES (@Dates,@ActionId,@RoomId,@DatesIn,@DaysCnt,@PersonsCnt,@UserId)

go

CREATE TRIGGER [dbo].[Trigger\_Moves\_PersonsCnt]

ON [dbo].[Moves]

AFTER INSERT, UPDATE

AS

DECLARE @P\_Max int, @P\_Current int, @M\_Id int, @M\_RoomId int;

BEGIN

Select @P\_Current = PersonsCnt, @M\_Id = Id, @M\_RoomId = RoomId FROM inserted;

Select @P\_Max = PersonsMax FROM Rooms WHERE Id = @M\_RoomId;

IF (@P\_Current > @P\_Max) UPDATE Moves SET PersonsCnt = @P\_Max WHERE Id = @M\_Id;

END

CREATE TRIGGER [dbo].[Trigger\_Rooms\_Update\_For\_Id]

ON [dbo].[Rooms]

AFTER UPDATE

AS

DECLARE @R\_Floor int, @R\_Number int;

BEGIN

SELECT @R\_Floor = Floor, @R\_Number = Number FROM inserted

UPDATE Rooms SET Id = @R\_Floor \* 100 + @R\_Number

WHERE (Floor = @R\_Floor AND Number = @R\_Number)

END

CREATE TRIGGER [dbo].[Trigger\_Rooms\_Insert\_For\_Id]

ON [dbo].[Rooms]

INSTEAD OF INSERT

AS

DECLARE @R\_Id int, @R\_Floor int, @R\_Number int, @R\_TypeId int, @R\_Persons int;

DECLARE InsCursor CURSOR FOR Select Floor, Number, TypeId, PersonsMax FROM Inserted;

BEGIN

OPEN InsCursor;

FETCH NEXT FROM InsCursor INTO @R\_Floor, @R\_Number, @R\_TypeId, @R\_Persons;

WHILE @@FETCH\_STATUS = 0

BEGIN

SET @R\_Id = @R\_Floor \* 100 + @R\_Number;

INSERT INTO Rooms (Id, Floor, Number, TypeId, PersonsMax) VALUES (@R\_Id, @R\_Floor, @R\_Number, @R\_TypeId, @R\_Persons)

FETCH NEXT FROM InsCursor INTO @R\_Floor, @R\_Number, @R\_TypeId, @R\_Persons;

END;

CLOSE InsCursor;

DEALLOCATE InsCursor;

END