

**MARMARA UNIVERSITY**

**FACULTY OF ENGINEERING**

**DEPARTMENT OF INDUSTRIAL**

**ENGINEERING**

**Recycling Garbage Collector Application (RGCA)**

**DENİZ YAĞCI 150317504**

**GAYE KARA 150317028**

July,2019

**TABLE OF CONTENTS**

[RECYCLING GARBAGE COLLECTOR APPLICATION 4](#_Toc30090516)

[ENTITY NAMES 5](#_Toc30090517)

[ENTITY (ER) DIAGRAM BY USING ORACLE NOTATION 6](#_Toc30090518)

[ENTITY (ER) DIAGRAM BY USING ERWIN 7](#_Toc30090519)

[MS SQL SERVER 10](#_Toc30090520)

[DATA TYPES AND KEYS 11](#_Toc30090521)

[RELATIONS 15](#_Toc30090522)

[SQL DATA DEFINITION LANGUAGE-DDL STATEMENT 16](#_Toc30090523)

[SQL EXPERIENCE TO THREE TABLE 62](#_Toc30090524)

[A)DDL STATEMENTS 62](#_Toc30090525)

[**A.1)CREATE/ALTER VIEW EXAMPLES** 62](#_Toc30090526)

[**A.2)CREATE/ALTER INDEX EXAMPLES** 64](#_Toc30090527)

[B)DML STATEMENTS-DATA MANIPULATION LANGUAGE 67](#_Toc30090528)

[**B.1)INSERT INTO** 67](#_Toc30090529)

[**B.2)UPDATE STATEMENTS** 68](#_Toc30090531)

[**B.3)DELETE STATEMENTS** 69](#_Toc30090532)

[**B.4)STORE PROCEDURE** 70](#_Toc30090533)

[C)DQL STATEMENTS-DATA QUERY LANGUAGE 71](#_Toc30090534)

[**C.1)SIMPLE JOINS-JOINING TWO TABLES** 71](#_Toc30090535)

[**C.2)INNER/OUTER JOINS** 72](#_Toc30090536)

[**C.3)NESTED QUERIES** 73](#_Toc30090537)

[**C.4)SORTING** 74](#_Toc30090538)

[**C.5)GROUP BY** 75](#_Toc30090539)

[**C.6)USING SELECT STATEMENTS WITH STORED PROCEDURE** 76](#_Toc30090540)

[D)DCL STATEMENTS-DATA CONTROL LANGUAGE 77](#_Toc30090541)

[INTEGRETY TEST 78](#_Toc30090542)

[1)TEST 1 78](#_Toc30090543)

[2)TEST 2 79](#_Toc30090544)

[4)TEST 4 81](#_Toc30090545)

[5)TEST 5 82](#_Toc30090546)

[7)TEST 7 84](#_Toc30090547)

[8)TEST 8 85](#_Toc30090548)

[9)TEST 9 86](#_Toc30090549)

[RGC Application(Visual Studio/C#) 87](#_Toc30090550)

[A)Photos of App 88](#_Toc30090551)

[B)Codes of App 98](#_Toc30090552)

# **RECYCLING GARBAGE COLLECTOR APPLICATION**

This project is aimed at optimizing the collection of materials suitable for recycling in universities. This type of garbage types, the location of the garbage cans, the names of the employees involved in the waste bins and discharge points were optimized.

As a result of the social and economic activities of people, all kinds of substances that have expired and become harmful to nature are called waste. Materials such as cardboard, glass, metal, plastic can be classified as waste. Recycling of waste is only possible through recycling. Recycling is the recycling of these wastes, which have the possibility of re-use, to physical or chemical processes and converting them to a second raw material. The purpose of recycling is to prevent excessive use of resources, to ensure the separation of wastes in their sources, to reduce the amount of waste garbage.

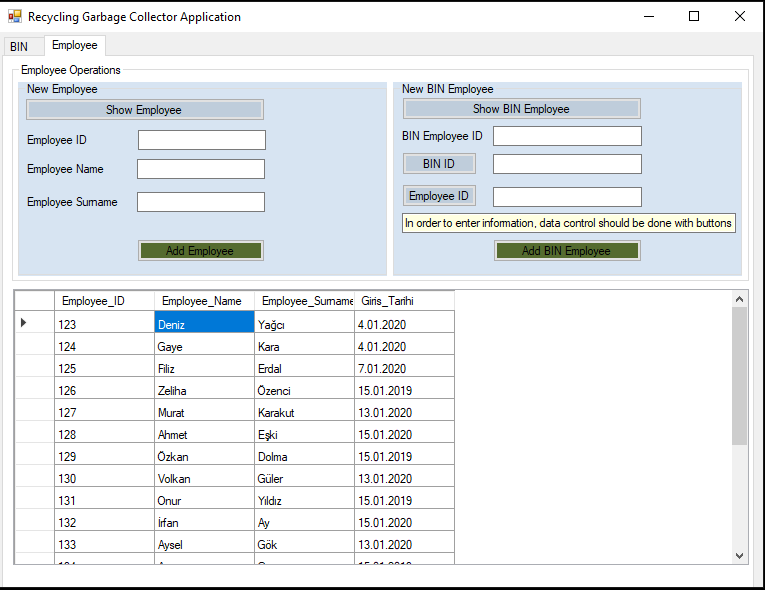
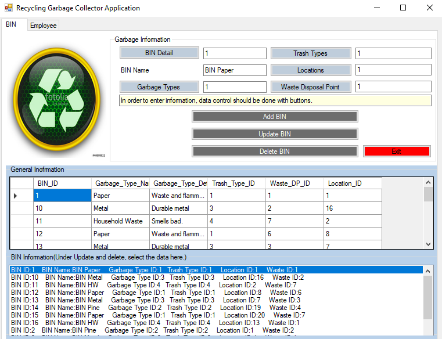
Recycling of materials such as paper, cardboard, glass, plastic and metal will prevent the depletion of natural resources. Recycling, which contributes greatly to the economy of the country, will reduce the amount of foreign currency paid to imported scrap and will significantly save energy use. Recycling of used paper reduces air pollution by 74-94%, water pollution by 35% and water use by 45%. Adding 1 ton of waste paper to the pulp prevents 8 trees from being cut.

Recycling is a very efficient economic investment in the long run. Human beings are faced with many economic problems in the future as a result of the depletion of raw materials and natural resources. On the other hand, many recycled wastes will contribute to the prevention of environmental pollution.

Importance of Recycling;

* Prevents the reduction of natural resources.
* It provides energy saving.
* The amount of solid waste to be disposed of is reduced.
* Contributes to the economy.
* It provides a clean environment for future generations.
* It contributes to the reduction of greenhouse gas emissions.
* Recycling of domestic wastes by composting method increases the soil yield. Thus, both organic and high nutritional products are obtained.

In order for recycling to take place in our society, first of all, recycling should be provided at the highest level in universities.

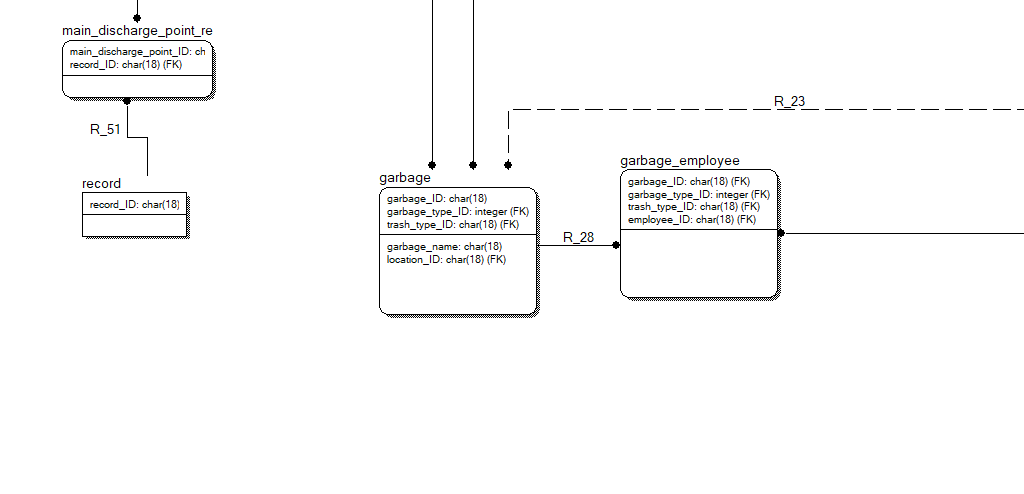
This application has been created for the purposes of the above symptoms. The following pages provide information about the application.

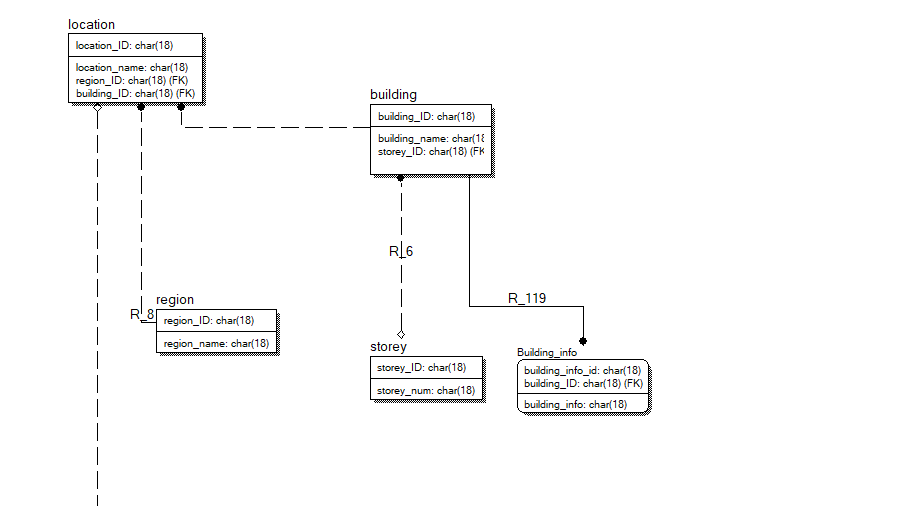
# **ENTITY NAMES**

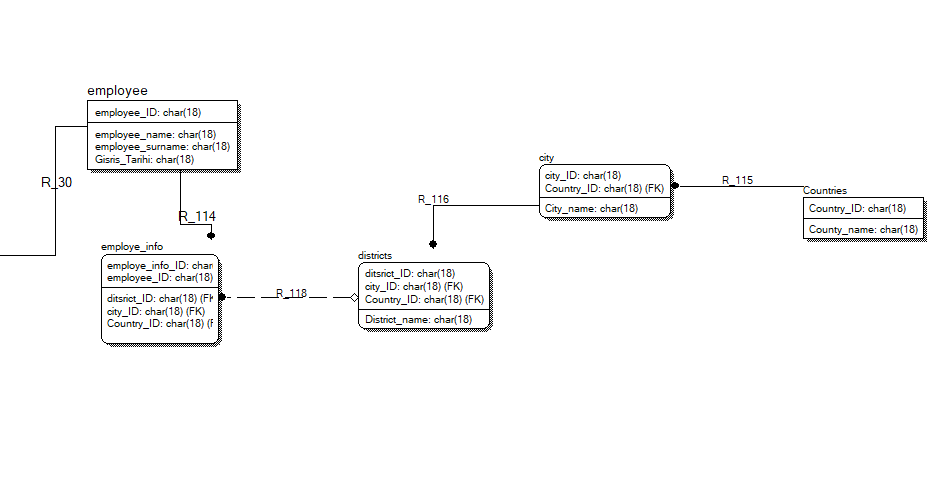
1. garbage\_type
2. trash\_type
3. trash\_company
4. capacity
5. main\_discharge\_point
6. garbage\_employee
7. garbage
8. employee
9. employee\_info
10. districts
11. city
12. countries
13. main\_discharge\_point\_record
14. record
15. region
16. location
17. building
18. storey
19. building info
20. building\_date

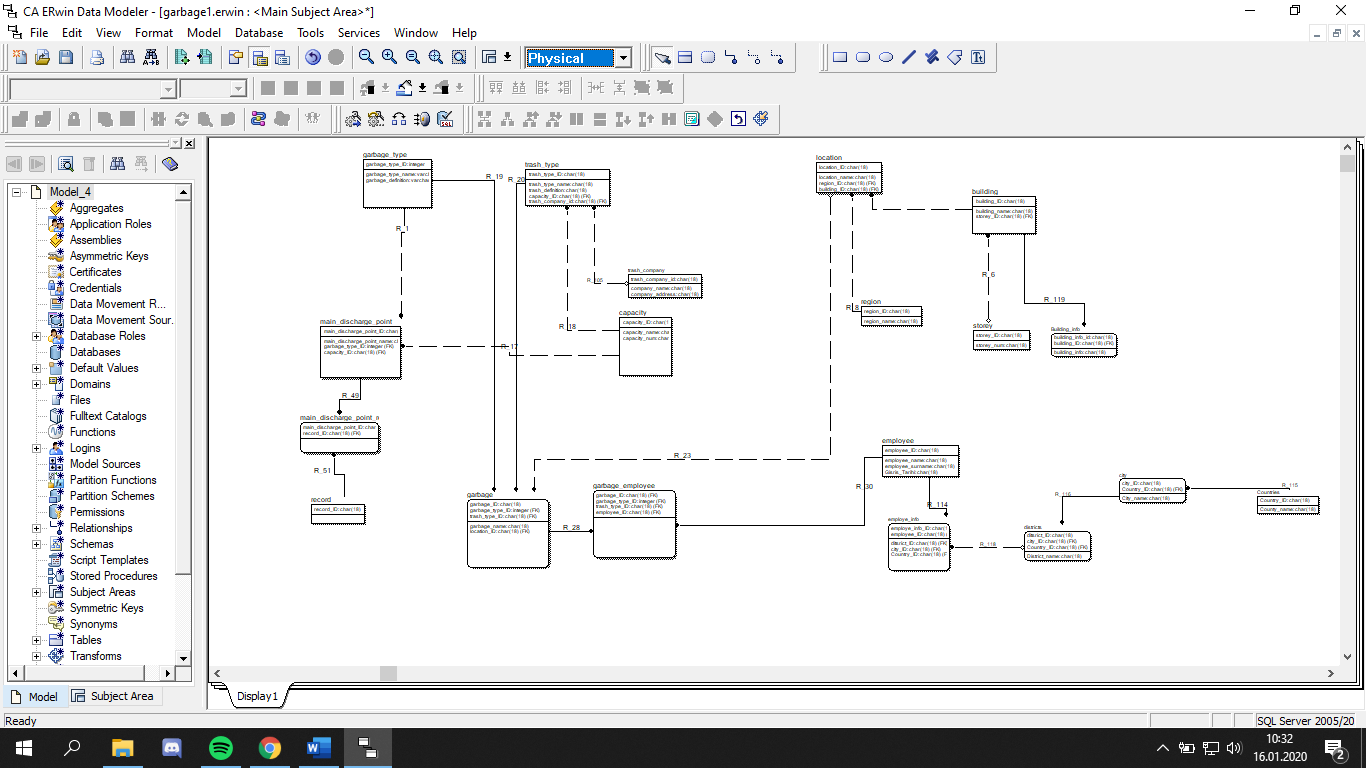
# **ENTITY (ER) DIAGRAM BY USING ORACLE NOTATION**

# **ENTITY (ER) DIAGRAM BY USING ERWIN**









# **MS SQL SERVER**

The SQL Server is a relational database management system from Microsoft. The system is designed and built to manage and store information. The system supports various business intelligence operations, analytics operations, and transaction processing. The information is stored on the server. However, since the system is much more than a database, it also includes a management system. SQL stands for Structured Query Language, a computer language that manages and administers the server. There are many versions of the SQL server, each subsequent version being an improved model of its predecessor.

Microsoft SQL Server has numerous applications in the business world. The first and most obvious one is the database. However, businesses that hold sensitive customer information, such as personal details, credit card information, and other confidential information will benefit from increased security. The system also provides data sharing by computers in the same network, a factor that increases reliability. The SQL server is also used to increase the speed with which data is processed, allowing large operations to be executed with ease. With the information stored in the database, businesses will have a reliable backup system.

# **DATA TYPES AND KEYS**

|  |  |  |
| --- | --- | --- |
| **Building** |  |  |
| Column Name | Data Type | Allow Nulls |
| **Building\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **Building\_Name** | **nvarchar(30)** | **Checked** |
| **Storey\_ID(FK)** | **nvarchar(10)** | **Checked** |

|  |  |  |
| --- | --- | --- |
| **Garbage\_Type** |  |  |
| Column Name | Data Type | Allow Nulls |
| **Garbage\_Type\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **Garbage\_Type\_Name** | **nvarchar(30)** | **Checked** |
| **Garbage\_Type\_Definition** | **nvarchar(30)** | **Checked** |

|  |  |  |
| --- | --- | --- |
| **Building\_info** |  |  |
| Column Name | Data Type | Allow Nulls |
| **building\_info\_id(PK)** | **nvarchar(10)** | **Checked** |
| **building\_ID(FK)** | **nvarchar(10)** | **Checked** |
| **building\_info** | **nvarchar(100)** | **Checked** |

|  |  |  |
| --- | --- | --- |
| **Location** |  |  |
| Column Name | Data Type | Allow Nulls |
| **Location\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **Location\_name** | **nvarchar(30)** | **Checked** |
| **Region\_ID(FK)** | **nvarchar(10)** | **Checked** |
| **Building\_ID(FK)** | **nvarchar(10)** | **Checked** |

|  |  |  |
| --- | --- | --- |
| **Capacity\_Type** |  |  |
| Column Name | Data Type | Allow Nulls |
| **Capacity\_Type\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **Capacity\_Type\_Name** | **nvarchar(30)** | **Checked** |
| **Capacity\_Amount** | **nvarchar(30)** | **Checked** |

|  |  |  |
| --- | --- | --- |
| **Main\_Discharge\_point\_record** |  |  |
| Column Name | Data Type | Allow Nulls |
| **Main\_Discharge\_point\_record\_ID(PK)** | **nvarchar(10)** | **Checked** |

|  |  |  |
| --- | --- | --- |
| **City** |  |  |
| Column Name | Data Type | Allow Nulls |
| **City\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **country\_ID(FK)** | **nvarchar(10)** | **Checked** |
| **City\_name** | **nvarchar(30)** | **Checked** |

|  |  |  |
| --- | --- | --- |
| **Record** |  |  |
| Column Name | Data Type | Allow Nulls |
| **record\_ID(PK)** | **nvarchar(10)** | **Checked** |

|  |  |  |
| --- | --- | --- |
| **Countries** |  |  |
| Column Name | Data Type | Allow Nulls |
| **Country\_ID(PK)** | **nvarchar(10)** | **Checked** |
| **Country\_name** | **nvarchar(30)** | **Checked** |

|  |  |  |
| --- | --- | --- |
| **district** |  |  |
| Column Name | Data Type | Allow Nulls |
| **District\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **City\_ID(FK)** | **nvarchar(10)** | **Checked** |
| **District\_name** | **nvarchar(30)** | **Checked** |

|  |  |  |
| --- | --- | --- |
| **Employee** |  |  |
| Column Name | Data Type | Allow Nulls |
| **Employee\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **Employee\_Name** | **nvarchar(30)** | **Checked** |
| **Employee\_Surname** | **nvarchar(30)** | **Checked** |
| **Giris\_Tarihi** | **date** | **Checked** |

|  |  |  |
| --- | --- | --- |
| **Garbage** |  |  |
| Column Name | Data Type | Allow Nulls |
| **BIN\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **BIN\_Name** | **nvarchar(30)** | **Checked** |
| **Garbage\_Type\_ID(FK)** | **nvarchar(10)** | **Checked** |
| **Trash\_Type\_ID(FK)** | **nvarchar(10)** | **Checked** |
| **Location\_ID(FK)** | **nvarchar(10)** | **Checked** |
| **Waste\_DP\_ID(FK)** | **nvarchar(10)** | **Checked** |

|  |  |  |
| --- | --- | --- |
| **Garbage\_Employee** |  |  |
| Column Name | Data Type | Allow Nulls |
| **Garbage\_Employee\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **BIN\_ID(FK)** | **nvarchar(10)** | **Checked** |
| **Employee\_ID(FK)** | **nvarchar(10)** | **Checked** |

|  |  |  |
| --- | --- | --- |
| **Main\_Discharge\_point\_record** |  |  |
| Column Name | Data Type | Allow Nulls |
| **Main\_Discharge\_point\_record\_ID(PK)** | **nvarchar(10)** | **Checked** |
|  |  |  |
|  |  |  |
|  |  |  |
| **Record** |  |  |
| Column Name | Data Type | Allow Nulls |
| **record\_ID(PK)** | **nvarchar(10)** | **Checked** |
|  |  |  |
|  |  |  |
| **Region** |  |  |
| Column Name | Data Type | Allow Nulls |
| **Region\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **Region\_Name** | **nvarchar(30)** | **Checked** |
|  |  |  |
|  |  |  |
| **Record\_Garbage\_Amount** |  |  |
| Column Name | Data Type | Allow Nulls |
| **Record\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **Record\_Amount** | **nvarchar(30)** | **Checked** |
| **Record\_Date** | **datetime** | **Checked** |
| **BIN\_ID(FK)** | **nvarchar(10)** | **Checked** |
| **Waste\_DP\_ID(FK)** | **nvarchar(10)** | **Checked** |
|  |  |  |
| **Storey** |  |  |
| Column Name | Data Type | Allow Nulls |
| **Storey\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **Storey\_Name** | **nvarchar(30)** | **Checked** |
|  |  |  |
|  |  |  |
| **System\_Users** |  |  |
| Column Name | Data Type | Allow Nulls |
| **User\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **User\_Name** | **nvarchar(30)** | **Checked** |
| **User\_Sifre** | **nvarchar(30)** | **Checked** |
|  |  |  |
|  |  |  |
| **trash\_Company** |  |  |
| Column Name | Data Type | Allow Nulls |
| **trash\_company\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **company\_name** | **nvarchar(30)** | **Checked** |
| **company\_address** | **nvarchar(30)** | **Checked** |
|  |  |  |
|  |  |  |
|  |  |  |
| **Trash\_Type** |  |  |
| Column Name | Data Type | Allow Nulls |
| **Trash\_Type\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **Trash\_Type\_Name** | **nvarchar(30)** | **Checked** |
| **Trash\_Type\_Definition** | **nvarchar(30)** | **Checked** |
| **Capacity\_Type\_ID(FK)** | **nvarchar(10)** | **Checked** |
| **trash\_company\_ID(FK)** | **nvarchar(10)** | **Checked** |
|  |  |  |
| **Waste\_Disposal\_Point** |  |  |
| Column Name | Data Type | Allow Nulls |
| **Waste\_DP\_ID(PK)** | **nvarchar(10)** | **Unchecked** |
| **Waste\_DP\_Name** | **nvarchar(30)** | **Checked** |
| **Capacity\_Type\_ID(FK)** | **nvarchar(10)** | **Checked** |
| **Garbage\_Type\_ID(FK)** | **nvarchar(10)** | **Checked** |
| **Region\_ID(FK)** | **nchar(10)** | **Checked** |

# **RELATIONS**

* Each Garbage must be defined one or only one Garbage Type.
* Each Garbage must be defined one or only one Trash Type.
* Each Garbage must be linked one or only one Garbage Employee.
* Each Garbage may be defined one or only one Location.
* Each Garbage Type must be connected one or more Garbage.
* Each Trash Type must be connected one or more Garbage.
* Each garbage Employee must be linked one or only one Garbage.
* Each Location may be connected one or more Garbage.
* Each Trash Type may be defined one or only one Trash Company.
* Each Trash Type may be defined one or only one Capacity.
* Each Trash Company may be connected one or more Trash Type.
* Each Capacity may be connected one or more Trash Type.
* Each Garbage Type may be defined one or more Main Discharge Point.
* Each Main Discharge Point may be connected one or only one Garbage Type.
* Each Main Discharge Point must be defined one or more Main Discharge Point Record.
* Each Main Discharge Point Record must be connected one or only one Main Discharge Point.
* Each Record must be connected one or more Main Discharge Point.
* Each Main Discharge Point must be defined one or only one Record.
* Each Building must be linked one or more Building Info.
* Each Location may be defined one or only one Region.
* Each Garbage Type must be connected one or more Garbage.
* Each Trash Type may be connected one or only one Trash Company.
* Each Trash Type may be connected one or only one Capacity.
* Each Garbage must be defined one or only one Garbage Employee.
* Each Garbage may be defined one or only one Location.
* Each Garbage Employee may be connected one or only one Employee.
* Each Region may be defined one or more Location.
* Each Main Discharge Point must be defined one or more Main Discharge Point Record.
* Each Main Discharge Point Record must be connected one or only one Main Discharge Point.
* Each Location may be defined one or only one Region.
* Each Garbage Type must be connected one or more Garbage.
* Each Region may be connected one or only one Trash Company.
* Each Trash Type may be connected one or only one Capacity.
* Each Capacity must be defined one or more Garbage.
* Each Garbage may be defined one or only one Location.

# **SQL DATA DEFINITION LANGUAGE-DDL STATEMENT**

CREATE TABLE building

(

building\_ID char(18) NOT NULL ,

building\_name char(18) NULL ,

storey\_ID char(18) NULL

)

go

ALTER TABLE building

ADD CONSTRAINT XPKbuilding PRIMARY KEY CLUSTERED (building\_ID ASC)

go

CREATE TABLE Building\_info

(

building\_info\_id char(18) NOT NULL ,

building\_info char(18) NULL ,

building\_ID char(18) NOT NULL

)

go

ALTER TABLE Building\_info

ADD CONSTRAINT XPKBuilding\_info PRIMARY KEY CLUSTERED (building\_info\_id ASC,building\_ID ASC)

go

CREATE TABLE capacity

(

capacity\_ID char(18) NOT NULL ,

capacity\_name char(18) NULL ,

capacity\_num char(18) NULL

)

go

ALTER TABLE capacity

ADD CONSTRAINT XPKcapacity PRIMARY KEY CLUSTERED (capacity\_ID ASC)

go

CREATE TABLE city

(

city\_ID char(18) NOT NULL ,

City\_name char(18) NULL ,

Country\_ID char(18) NOT NULL

)

go

ALTER TABLE city

ADD CONSTRAINT XPKcity PRIMARY KEY CLUSTERED (city\_ID ASC,Country\_ID ASC)

go

CREATE TABLE Countries

(

Country\_ID char(18) NOT NULL ,

County\_name char(18) NULL

)

go

ALTER TABLE Countries

ADD CONSTRAINT XPKCountries PRIMARY KEY CLUSTERED (Country\_ID ASC)

go

CREATE TABLE districts

(

ditsrict\_ID char(18) NOT NULL ,

District\_name char(18) NULL ,

city\_ID char(18) NOT NULL ,

Country\_ID char(18) NOT NULL

)

go

ALTER TABLE districts

ADD CONSTRAINT XPKdistricts PRIMARY KEY CLUSTERED (ditsrict\_ID ASC,city\_ID ASC,Country\_ID ASC)

go

CREATE TABLE employe\_info

(

employe\_info\_ID char(18) NOT NULL ,

employee\_ID char(18) NOT NULL ,

ditsrict\_ID char(18) NULL ,

city\_ID char(18) NULL ,

Country\_ID char(18) NULL

)

go

ALTER TABLE employe\_info

ADD CONSTRAINT XPKemploye\_info PRIMARY KEY CLUSTERED (employe\_info\_ID ASC,employee\_ID ASC)

go

CREATE TABLE employee

(

employee\_ID char(18) NOT NULL ,

employee\_name char(18) NULL ,

employee\_surname char(18) NULL ,

Gisris\_Tarihi char(18) NULL

)

go

ALTER TABLE employee

ADD CONSTRAINT XPKemployee PRIMARY KEY CLUSTERED (employee\_ID ASC)

go

CREATE TABLE garbage

(

garbage\_ID char(18) NOT NULL ,

garbage\_name char(18) NULL ,

garbage\_type\_ID integer NOT NULL ,

trash\_type\_ID char(18) NOT NULL ,

location\_ID char(18) NULL

)

go

ALTER TABLE garbage

ADD CONSTRAINT XPKgarbage PRIMARY KEY CLUSTERED (garbage\_ID ASC,garbage\_type\_ID ASC,trash\_type\_ID ASC)

go

CREATE TABLE garbage\_employee

(

garbage\_ID char(18) NOT NULL ,

garbage\_type\_ID integer NOT NULL ,

trash\_type\_ID char(18) NOT NULL ,

employee\_ID char(18) NOT NULL

)

go

ALTER TABLE garbage\_employee

ADD CONSTRAINT XPKgarbage\_employee PRIMARY KEY CLUSTERED (garbage\_ID ASC,garbage\_type\_ID ASC,trash\_type\_ID ASC,employee\_ID ASC)

go

CREATE TABLE garbage\_type

(

garbage\_type\_ID integer NOT NULL ,

garbage\_type\_name varchar(20) NULL ,

garbage\_definition varchar(20) NULL

)

go

ALTER TABLE garbage\_type

ADD CONSTRAINT XPKgarbage\_type PRIMARY KEY CLUSTERED (garbage\_type\_ID ASC)

go

CREATE TABLE location

(

location\_ID char(18) NOT NULL ,

location\_name char(18) NULL ,

region\_ID char(18) NOT NULL ,

building\_ID char(18) NOT NULL

)

go

ALTER TABLE location

ADD CONSTRAINT XPKlocation PRIMARY KEY CLUSTERED (location\_ID ASC)

go

CREATE TABLE main\_discharge\_point

(

main\_discharge\_point\_ID char(18) NOT NULL ,

main\_discharge\_point\_name char(18) NULL ,

garbage\_type\_ID integer NOT NULL ,

capacity\_ID char(18) NOT NULL

)

go

ALTER TABLE main\_discharge\_point

ADD CONSTRAINT XPKmain\_discharge\_point PRIMARY KEY CLUSTERED (main\_discharge\_point\_ID ASC)

go

CREATE TABLE main\_discharge\_point\_record

(

main\_discharge\_point\_ID char(18) NOT NULL ,

record\_ID char(18) NOT NULL

)

go

ALTER TABLE main\_discharge\_point\_record

ADD CONSTRAINT XPKmain\_discharge\_point\_record PRIMARY KEY CLUSTERED (main\_discharge\_point\_ID ASC,record\_ID ASC)

go

CREATE TABLE record

(

record\_ID char(18) NOT NULL

)

go

ALTER TABLE record

ADD CONSTRAINT XPKrecord PRIMARY KEY CLUSTERED (record\_ID ASC)

go

CREATE TABLE region

(

region\_ID char(18) NOT NULL ,

region\_name char(18) NULL

)

go

ALTER TABLE region

ADD CONSTRAINT XPKregion PRIMARY KEY CLUSTERED (region\_ID ASC)

go

CREATE TABLE storey

(

storey\_ID char(18) NOT NULL ,

storey\_num char(18) NULL

)

go

ALTER TABLE storey

ADD CONSTRAINT XPKstorey PRIMARY KEY CLUSTERED (storey\_ID ASC)

go

CREATE TABLE trash\_company

(

trash\_company\_id char(18) NOT NULL ,

company\_name char(18) NULL ,

company\_address char(18) NULL

)

go

ALTER TABLE trash\_company

ADD CONSTRAINT XPKtrash\_company PRIMARY KEY CLUSTERED (trash\_company\_id ASC)

go

CREATE TABLE trash\_type

(

trash\_type\_ID char(18) NOT NULL ,

trash\_type\_name char(18) NULL ,

trash\_definition char(18) NULL ,

capacity\_ID char(18) NOT NULL ,

trash\_company\_id char(18) NULL

)

go

ALTER TABLE trash\_type

ADD CONSTRAINT XPKtrash\_type PRIMARY KEY CLUSTERED (trash\_type\_ID ASC)

go

ALTER TABLE building

ADD CONSTRAINT R\_6 FOREIGN KEY (storey\_ID) REFERENCES storey(storey\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE Building\_info

ADD CONSTRAINT R\_119 FOREIGN KEY (building\_ID) REFERENCES building(building\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE city

ADD CONSTRAINT R\_115 FOREIGN KEY (Country\_ID) REFERENCES Countries(Country\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE districts

ADD CONSTRAINT R\_116 FOREIGN KEY (city\_ID,Country\_ID) REFERENCES city(city\_ID,Country\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE employe\_info

ADD CONSTRAINT R\_114 FOREIGN KEY (employee\_ID) REFERENCES employee(employee\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE employe\_info

ADD CONSTRAINT R\_118 FOREIGN KEY (ditsrict\_ID,city\_ID,Country\_ID) REFERENCES districts(ditsrict\_ID,city\_ID,Country\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE garbage

ADD CONSTRAINT R\_19 FOREIGN KEY (garbage\_type\_ID) REFERENCES garbage\_type(garbage\_type\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE garbage

ADD CONSTRAINT R\_20 FOREIGN KEY (trash\_type\_ID) REFERENCES trash\_type(trash\_type\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE garbage

ADD CONSTRAINT R\_23 FOREIGN KEY (location\_ID) REFERENCES location(location\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE garbage\_employee

ADD CONSTRAINT R\_28 FOREIGN KEY (garbage\_ID,garbage\_type\_ID,trash\_type\_ID) REFERENCES garbage(garbage\_ID,garbage\_type\_ID,trash\_type\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE garbage\_employee

ADD CONSTRAINT R\_30 FOREIGN KEY (employee\_ID) REFERENCES employee(employee\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE location

ADD CONSTRAINT R\_8 FOREIGN KEY (region\_ID) REFERENCES region(region\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE location

ADD CONSTRAINT R\_9 FOREIGN KEY (building\_ID) REFERENCES building(building\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE main\_discharge\_point

ADD CONSTRAINT R\_1 FOREIGN KEY (garbage\_type\_ID) REFERENCES garbage\_type(garbage\_type\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE main\_discharge\_point

ADD CONSTRAINT R\_17 FOREIGN KEY (capacity\_ID) REFERENCES capacity(capacity\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE main\_discharge\_point\_record

ADD CONSTRAINT R\_49 FOREIGN KEY (main\_discharge\_point\_ID) REFERENCES main\_discharge\_point(main\_discharge\_point\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE main\_discharge\_point\_record

ADD CONSTRAINT R\_51 FOREIGN KEY (record\_ID) REFERENCES record(record\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE trash\_type

ADD CONSTRAINT R\_18 FOREIGN KEY (capacity\_ID) REFERENCES capacity(capacity\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

ALTER TABLE trash\_type

ADD CONSTRAINT R\_105 FOREIGN KEY (trash\_company\_id) REFERENCES trash\_company(trash\_company\_id)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go

CREATE TRIGGER tD\_building ON building FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on building \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* building location on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00030cba", PARENT\_OWNER="", PARENT\_TABLE="building"

CHILD\_OWNER="", CHILD\_TABLE="location"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_9", FK\_COLUMNS="building\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,location

WHERE

/\* %JoinFKPK(location,deleted," = "," AND") \*/

location.building\_ID = deleted.building\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete building because location exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* building Building\_info on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="building"

CHILD\_OWNER="", CHILD\_TABLE="Building\_info"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_119", FK\_COLUMNS="building\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,Building\_info

WHERE

/\* %JoinFKPK(Building\_info,deleted," = "," AND") \*/

Building\_info.building\_ID = deleted.building\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete building because Building\_info exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* storey building on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="storey"

CHILD\_OWNER="", CHILD\_TABLE="building"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_6", FK\_COLUMNS="storey\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,storey

WHERE

/\* %JoinFKPK(deleted,storey," = "," AND") \*/

deleted.storey\_ID = storey.storey\_ID AND

NOT EXISTS (

SELECT \* FROM building

WHERE

/\* %JoinFKPK(building,storey," = "," AND") \*/

building.storey\_ID = storey.storey\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last building because storey exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_building ON building FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on building \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@insbuilding\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* building location on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00037bcb", PARENT\_OWNER="", PARENT\_TABLE="building"

CHILD\_OWNER="", CHILD\_TABLE="location"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_9", FK\_COLUMNS="building\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(building\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,location

WHERE

/\* %JoinFKPK(location,deleted," = "," AND") \*/

location.building\_ID = deleted.building\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update building because location exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* building Building\_info on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="building"

CHILD\_OWNER="", CHILD\_TABLE="Building\_info"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_119", FK\_COLUMNS="building\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(building\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,Building\_info

WHERE

/\* %JoinFKPK(Building\_info,deleted," = "," AND") \*/

Building\_info.building\_ID = deleted.building\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update building because Building\_info exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* storey building on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="storey"

CHILD\_OWNER="", CHILD\_TABLE="building"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_6", FK\_COLUMNS="storey\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(storey\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,storey

WHERE

/\* %JoinFKPK(inserted,storey) \*/

inserted.storey\_ID = storey.storey\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

select @nullcnt = count(\*) from inserted where

inserted.storey\_ID IS NULL

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update building because storey does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_Building\_info ON Building\_info FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on Building\_info \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* building Building\_info on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00013de0", PARENT\_OWNER="", PARENT\_TABLE="building"

CHILD\_OWNER="", CHILD\_TABLE="Building\_info"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_119", FK\_COLUMNS="building\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,building

WHERE

/\* %JoinFKPK(deleted,building," = "," AND") \*/

deleted.building\_ID = building.building\_ID AND

NOT EXISTS (

SELECT \* FROM Building\_info

WHERE

/\* %JoinFKPK(Building\_info,building," = "," AND") \*/

Building\_info.building\_ID = building.building\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last Building\_info because building exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_Building\_info ON Building\_info FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on Building\_info \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@insbuilding\_info\_id char(18),

@insbuilding\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* building Building\_info on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0001517e", PARENT\_OWNER="", PARENT\_TABLE="building"

CHILD\_OWNER="", CHILD\_TABLE="Building\_info"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_119", FK\_COLUMNS="building\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(building\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,building

WHERE

/\* %JoinFKPK(inserted,building) \*/

inserted.building\_ID = building.building\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update Building\_info because building does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_capacity ON capacity FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on capacity \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* capacity main\_discharge\_point on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0001f7c4", PARENT\_OWNER="", PARENT\_TABLE="capacity"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_17", FK\_COLUMNS="capacity\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,main\_discharge\_point

WHERE

/\* %JoinFKPK(main\_discharge\_point,deleted," = "," AND") \*/

main\_discharge\_point.capacity\_ID = deleted.capacity\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete capacity because main\_discharge\_point exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* capacity trash\_type on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="capacity"

CHILD\_OWNER="", CHILD\_TABLE="trash\_type"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_18", FK\_COLUMNS="capacity\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,trash\_type

WHERE

/\* %JoinFKPK(trash\_type,deleted," = "," AND") \*/

trash\_type.capacity\_ID = deleted.capacity\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete capacity because trash\_type exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_capacity ON capacity FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on capacity \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@inscapacity\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* capacity main\_discharge\_point on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00023b93", PARENT\_OWNER="", PARENT\_TABLE="capacity"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_17", FK\_COLUMNS="capacity\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(capacity\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,main\_discharge\_point

WHERE

/\* %JoinFKPK(main\_discharge\_point,deleted," = "," AND") \*/

main\_discharge\_point.capacity\_ID = deleted.capacity\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update capacity because main\_discharge\_point exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* capacity trash\_type on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="capacity"

CHILD\_OWNER="", CHILD\_TABLE="trash\_type"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_18", FK\_COLUMNS="capacity\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(capacity\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,trash\_type

WHERE

/\* %JoinFKPK(trash\_type,deleted," = "," AND") \*/

trash\_type.capacity\_ID = deleted.capacity\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update capacity because trash\_type exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_city ON city FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on city \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* city districts on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0002232b", PARENT\_OWNER="", PARENT\_TABLE="city"

CHILD\_OWNER="", CHILD\_TABLE="districts"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_116", FK\_COLUMNS="city\_ID""Country\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,districts

WHERE

/\* %JoinFKPK(districts,deleted," = "," AND") \*/

districts.city\_ID = deleted.city\_ID AND

districts.Country\_ID = deleted.Country\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete city because districts exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* Countries city on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="Countries"

CHILD\_OWNER="", CHILD\_TABLE="city"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_115", FK\_COLUMNS="Country\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,Countries

WHERE

/\* %JoinFKPK(deleted,Countries," = "," AND") \*/

deleted.Country\_ID = Countries.Country\_ID AND

NOT EXISTS (

SELECT \* FROM city

WHERE

/\* %JoinFKPK(city,Countries," = "," AND") \*/

city.Country\_ID = Countries.Country\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last city because Countries exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_city ON city FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on city \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@inscity\_ID char(18),

@insCountry\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* city districts on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="000271be", PARENT\_OWNER="", PARENT\_TABLE="city"

CHILD\_OWNER="", CHILD\_TABLE="districts"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_116", FK\_COLUMNS="city\_ID""Country\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(city\_ID) OR

UPDATE(Country\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,districts

WHERE

/\* %JoinFKPK(districts,deleted," = "," AND") \*/

districts.city\_ID = deleted.city\_ID AND

districts.Country\_ID = deleted.Country\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update city because districts exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* Countries city on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="Countries"

CHILD\_OWNER="", CHILD\_TABLE="city"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_115", FK\_COLUMNS="Country\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(Country\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,Countries

WHERE

/\* %JoinFKPK(inserted,Countries) \*/

inserted.Country\_ID = Countries.Country\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update city because Countries does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_Countries ON Countries FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on Countries \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* Countries city on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0000e55d", PARENT\_OWNER="", PARENT\_TABLE="Countries"

CHILD\_OWNER="", CHILD\_TABLE="city"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_115", FK\_COLUMNS="Country\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,city

WHERE

/\* %JoinFKPK(city,deleted," = "," AND") \*/

city.Country\_ID = deleted.Country\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete Countries because city exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_Countries ON Countries FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on Countries \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@insCountry\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* Countries city on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00010087", PARENT\_OWNER="", PARENT\_TABLE="Countries"

CHILD\_OWNER="", CHILD\_TABLE="city"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_115", FK\_COLUMNS="Country\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(Country\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,city

WHERE

/\* %JoinFKPK(city,deleted," = "," AND") \*/

city.Country\_ID = deleted.Country\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update Countries because city exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_districts ON districts FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on districts \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* districts employe\_info on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00027ef4", PARENT\_OWNER="", PARENT\_TABLE="districts"

CHILD\_OWNER="", CHILD\_TABLE="employe\_info"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_118", FK\_COLUMNS="ditsrict\_ID""city\_ID""Country\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,employe\_info

WHERE

/\* %JoinFKPK(employe\_info,deleted," = "," AND") \*/

employe\_info.ditsrict\_ID = deleted.ditsrict\_ID AND

employe\_info.city\_ID = deleted.city\_ID AND

employe\_info.Country\_ID = deleted.Country\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete districts because employe\_info exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* city districts on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="city"

CHILD\_OWNER="", CHILD\_TABLE="districts"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_116", FK\_COLUMNS="city\_ID""Country\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,city

WHERE

/\* %JoinFKPK(deleted,city," = "," AND") \*/

deleted.city\_ID = city.city\_ID AND

deleted.Country\_ID = city.Country\_ID AND

NOT EXISTS (

SELECT \* FROM districts

WHERE

/\* %JoinFKPK(districts,city," = "," AND") \*/

districts.city\_ID = city.city\_ID AND

districts.Country\_ID = city.Country\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last districts because city exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_districts ON districts FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on districts \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@insditsrict\_ID char(18),

@inscity\_ID char(18),

@insCountry\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* districts employe\_info on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0002bb90", PARENT\_OWNER="", PARENT\_TABLE="districts"

CHILD\_OWNER="", CHILD\_TABLE="employe\_info"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_118", FK\_COLUMNS="ditsrict\_ID""city\_ID""Country\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(ditsrict\_ID) OR

UPDATE(city\_ID) OR

UPDATE(Country\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,employe\_info

WHERE

/\* %JoinFKPK(employe\_info,deleted," = "," AND") \*/

employe\_info.ditsrict\_ID = deleted.ditsrict\_ID AND

employe\_info.city\_ID = deleted.city\_ID AND

employe\_info.Country\_ID = deleted.Country\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update districts because employe\_info exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* city districts on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="city"

CHILD\_OWNER="", CHILD\_TABLE="districts"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_116", FK\_COLUMNS="city\_ID""Country\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(city\_ID) OR

UPDATE(Country\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,city

WHERE

/\* %JoinFKPK(inserted,city) \*/

inserted.city\_ID = city.city\_ID and

inserted.Country\_ID = city.Country\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update districts because city does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_employe\_info ON employe\_info FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on employe\_info \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* employee employe\_info on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0002d09f", PARENT\_OWNER="", PARENT\_TABLE="employee"

CHILD\_OWNER="", CHILD\_TABLE="employe\_info"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_114", FK\_COLUMNS="employee\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,employee

WHERE

/\* %JoinFKPK(deleted,employee," = "," AND") \*/

deleted.employee\_ID = employee.employee\_ID AND

NOT EXISTS (

SELECT \* FROM employe\_info

WHERE

/\* %JoinFKPK(employe\_info,employee," = "," AND") \*/

employe\_info.employee\_ID = employee.employee\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last employe\_info because employee exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* districts employe\_info on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="districts"

CHILD\_OWNER="", CHILD\_TABLE="employe\_info"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_118", FK\_COLUMNS="ditsrict\_ID""city\_ID""Country\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,districts

WHERE

/\* %JoinFKPK(deleted,districts," = "," AND") \*/

deleted.ditsrict\_ID = districts.ditsrict\_ID AND

deleted.city\_ID = districts.city\_ID AND

deleted.Country\_ID = districts.Country\_ID AND

NOT EXISTS (

SELECT \* FROM employe\_info

WHERE

/\* %JoinFKPK(employe\_info,districts," = "," AND") \*/

employe\_info.ditsrict\_ID = districts.ditsrict\_ID AND

employe\_info.city\_ID = districts.city\_ID AND

employe\_info.Country\_ID = districts.Country\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last employe\_info because districts exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_employe\_info ON employe\_info FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on employe\_info \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@insemploye\_info\_ID char(18),

@insemployee\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* employee employe\_info on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0003147a", PARENT\_OWNER="", PARENT\_TABLE="employee"

CHILD\_OWNER="", CHILD\_TABLE="employe\_info"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_114", FK\_COLUMNS="employee\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(employee\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,employee

WHERE

/\* %JoinFKPK(inserted,employee) \*/

inserted.employee\_ID = employee.employee\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update employe\_info because employee does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* districts employe\_info on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="districts"

CHILD\_OWNER="", CHILD\_TABLE="employe\_info"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_118", FK\_COLUMNS="ditsrict\_ID""city\_ID""Country\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(ditsrict\_ID) OR

UPDATE(city\_ID) OR

UPDATE(Country\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,districts

WHERE

/\* %JoinFKPK(inserted,districts) \*/

inserted.ditsrict\_ID = districts.ditsrict\_ID and

inserted.city\_ID = districts.city\_ID and

inserted.Country\_ID = districts.Country\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

select @nullcnt = count(\*) from inserted where

inserted.ditsrict\_ID IS NULL AND

inserted.city\_ID IS NULL AND

inserted.Country\_ID IS NULL

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update employe\_info because districts does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_employee ON employee FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on employee \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* employee employe\_info on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0001ee48", PARENT\_OWNER="", PARENT\_TABLE="employee"

CHILD\_OWNER="", CHILD\_TABLE="employe\_info"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_114", FK\_COLUMNS="employee\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,employe\_info

WHERE

/\* %JoinFKPK(employe\_info,deleted," = "," AND") \*/

employe\_info.employee\_ID = deleted.employee\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete employee because employe\_info exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* employee garbage\_employee on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="employee"

CHILD\_OWNER="", CHILD\_TABLE="garbage\_employee"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_30", FK\_COLUMNS="employee\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,garbage\_employee

WHERE

/\* %JoinFKPK(garbage\_employee,deleted," = "," AND") \*/

garbage\_employee.employee\_ID = deleted.employee\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete employee because garbage\_employee exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_employee ON employee FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on employee \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@insemployee\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* employee employe\_info on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="000241d9", PARENT\_OWNER="", PARENT\_TABLE="employee"

CHILD\_OWNER="", CHILD\_TABLE="employe\_info"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_114", FK\_COLUMNS="employee\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(employee\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,employe\_info

WHERE

/\* %JoinFKPK(employe\_info,deleted," = "," AND") \*/

employe\_info.employee\_ID = deleted.employee\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update employee because employe\_info exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* employee garbage\_employee on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="employee"

CHILD\_OWNER="", CHILD\_TABLE="garbage\_employee"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_30", FK\_COLUMNS="employee\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(employee\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,garbage\_employee

WHERE

/\* %JoinFKPK(garbage\_employee,deleted," = "," AND") \*/

garbage\_employee.employee\_ID = deleted.employee\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update employee because garbage\_employee exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_garbage ON garbage FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on garbage \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* garbage garbage\_employee on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0004d303", PARENT\_OWNER="", PARENT\_TABLE="garbage"

CHILD\_OWNER="", CHILD\_TABLE="garbage\_employee"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_28", FK\_COLUMNS="garbage\_ID""garbage\_type\_ID""trash\_type\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,garbage\_employee

WHERE

/\* %JoinFKPK(garbage\_employee,deleted," = "," AND") \*/

garbage\_employee.garbage\_ID = deleted.garbage\_ID AND

garbage\_employee.garbage\_type\_ID = deleted.garbage\_type\_ID AND

garbage\_employee.trash\_type\_ID = deleted.trash\_type\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete garbage because garbage\_employee exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* garbage\_type garbage on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="garbage\_type"

CHILD\_OWNER="", CHILD\_TABLE="garbage"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_19", FK\_COLUMNS="garbage\_type\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,garbage\_type

WHERE

/\* %JoinFKPK(deleted,garbage\_type," = "," AND") \*/

deleted.garbage\_type\_ID = garbage\_type.garbage\_type\_ID AND

NOT EXISTS (

SELECT \* FROM garbage

WHERE

/\* %JoinFKPK(garbage,garbage\_type," = "," AND") \*/

garbage.garbage\_type\_ID = garbage\_type.garbage\_type\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last garbage because garbage\_type exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* trash\_type garbage on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="trash\_type"

CHILD\_OWNER="", CHILD\_TABLE="garbage"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_20", FK\_COLUMNS="trash\_type\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,trash\_type

WHERE

/\* %JoinFKPK(deleted,trash\_type," = "," AND") \*/

deleted.trash\_type\_ID = trash\_type.trash\_type\_ID AND

NOT EXISTS (

SELECT \* FROM garbage

WHERE

/\* %JoinFKPK(garbage,trash\_type," = "," AND") \*/

garbage.trash\_type\_ID = trash\_type.trash\_type\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last garbage because trash\_type exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* location garbage on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="location"

CHILD\_OWNER="", CHILD\_TABLE="garbage"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_23", FK\_COLUMNS="location\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,location

WHERE

/\* %JoinFKPK(deleted,location," = "," AND") \*/

deleted.location\_ID = location.location\_ID AND

NOT EXISTS (

SELECT \* FROM garbage

WHERE

/\* %JoinFKPK(garbage,location," = "," AND") \*/

garbage.location\_ID = location.location\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last garbage because location exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_garbage ON garbage FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on garbage \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@insgarbage\_ID char(18),

@insgarbage\_type\_ID integer,

@instrash\_type\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* garbage garbage\_employee on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00058a1b", PARENT\_OWNER="", PARENT\_TABLE="garbage"

CHILD\_OWNER="", CHILD\_TABLE="garbage\_employee"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_28", FK\_COLUMNS="garbage\_ID""garbage\_type\_ID""trash\_type\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(garbage\_ID) OR

UPDATE(garbage\_type\_ID) OR

UPDATE(trash\_type\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,garbage\_employee

WHERE

/\* %JoinFKPK(garbage\_employee,deleted," = "," AND") \*/

garbage\_employee.garbage\_ID = deleted.garbage\_ID AND

garbage\_employee.garbage\_type\_ID = deleted.garbage\_type\_ID AND

garbage\_employee.trash\_type\_ID = deleted.trash\_type\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update garbage because garbage\_employee exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* garbage\_type garbage on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="garbage\_type"

CHILD\_OWNER="", CHILD\_TABLE="garbage"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_19", FK\_COLUMNS="garbage\_type\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(garbage\_type\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,garbage\_type

WHERE

/\* %JoinFKPK(inserted,garbage\_type) \*/

inserted.garbage\_type\_ID = garbage\_type.garbage\_type\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update garbage because garbage\_type does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* trash\_type garbage on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="trash\_type"

CHILD\_OWNER="", CHILD\_TABLE="garbage"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_20", FK\_COLUMNS="trash\_type\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(trash\_type\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,trash\_type

WHERE

/\* %JoinFKPK(inserted,trash\_type) \*/

inserted.trash\_type\_ID = trash\_type.trash\_type\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update garbage because trash\_type does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* location garbage on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="location"

CHILD\_OWNER="", CHILD\_TABLE="garbage"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_23", FK\_COLUMNS="location\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(location\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,location

WHERE

/\* %JoinFKPK(inserted,location) \*/

inserted.location\_ID = location.location\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

select @nullcnt = count(\*) from inserted where

inserted.location\_ID IS NULL

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update garbage because location does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_garbage\_employee ON garbage\_employee FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on garbage\_employee \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* garbage garbage\_employee on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0002e540", PARENT\_OWNER="", PARENT\_TABLE="garbage"

CHILD\_OWNER="", CHILD\_TABLE="garbage\_employee"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_28", FK\_COLUMNS="garbage\_ID""garbage\_type\_ID""trash\_type\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,garbage

WHERE

/\* %JoinFKPK(deleted,garbage," = "," AND") \*/

deleted.garbage\_ID = garbage.garbage\_ID AND

deleted.garbage\_type\_ID = garbage.garbage\_type\_ID AND

deleted.trash\_type\_ID = garbage.trash\_type\_ID AND

NOT EXISTS (

SELECT \* FROM garbage\_employee

WHERE

/\* %JoinFKPK(garbage\_employee,garbage," = "," AND") \*/

garbage\_employee.garbage\_ID = garbage.garbage\_ID AND

garbage\_employee.garbage\_type\_ID = garbage.garbage\_type\_ID AND

garbage\_employee.trash\_type\_ID = garbage.trash\_type\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last garbage\_employee because garbage exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* employee garbage\_employee on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="employee"

CHILD\_OWNER="", CHILD\_TABLE="garbage\_employee"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_30", FK\_COLUMNS="employee\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,employee

WHERE

/\* %JoinFKPK(deleted,employee," = "," AND") \*/

deleted.employee\_ID = employee.employee\_ID AND

NOT EXISTS (

SELECT \* FROM garbage\_employee

WHERE

/\* %JoinFKPK(garbage\_employee,employee," = "," AND") \*/

garbage\_employee.employee\_ID = employee.employee\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last garbage\_employee because employee exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_garbage\_employee ON garbage\_employee FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on garbage\_employee \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@insgarbage\_ID char(18),

@insgarbage\_type\_ID integer,

@instrash\_type\_ID char(18),

@insemployee\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* garbage garbage\_employee on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0002f95d", PARENT\_OWNER="", PARENT\_TABLE="garbage"

CHILD\_OWNER="", CHILD\_TABLE="garbage\_employee"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_28", FK\_COLUMNS="garbage\_ID""garbage\_type\_ID""trash\_type\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(garbage\_ID) OR

UPDATE(garbage\_type\_ID) OR

UPDATE(trash\_type\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,garbage

WHERE

/\* %JoinFKPK(inserted,garbage) \*/

inserted.garbage\_ID = garbage.garbage\_ID and

inserted.garbage\_type\_ID = garbage.garbage\_type\_ID and

inserted.trash\_type\_ID = garbage.trash\_type\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update garbage\_employee because garbage does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* employee garbage\_employee on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="employee"

CHILD\_OWNER="", CHILD\_TABLE="garbage\_employee"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_30", FK\_COLUMNS="employee\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(employee\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,employee

WHERE

/\* %JoinFKPK(inserted,employee) \*/

inserted.employee\_ID = employee.employee\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update garbage\_employee because employee does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_garbage\_type ON garbage\_type FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on garbage\_type \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* garbage\_type main\_discharge\_point on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="000216fa", PARENT\_OWNER="", PARENT\_TABLE="garbage\_type"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_1", FK\_COLUMNS="garbage\_type\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,main\_discharge\_point

WHERE

/\* %JoinFKPK(main\_discharge\_point,deleted," = "," AND") \*/

main\_discharge\_point.garbage\_type\_ID = deleted.garbage\_type\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete garbage\_type because main\_discharge\_point exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* garbage\_type garbage on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="garbage\_type"

CHILD\_OWNER="", CHILD\_TABLE="garbage"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_19", FK\_COLUMNS="garbage\_type\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,garbage

WHERE

/\* %JoinFKPK(garbage,deleted," = "," AND") \*/

garbage.garbage\_type\_ID = deleted.garbage\_type\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete garbage\_type because garbage exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_garbage\_type ON garbage\_type FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on garbage\_type \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@insgarbage\_type\_ID integer,

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* garbage\_type main\_discharge\_point on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00024799", PARENT\_OWNER="", PARENT\_TABLE="garbage\_type"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_1", FK\_COLUMNS="garbage\_type\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(garbage\_type\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,main\_discharge\_point

WHERE

/\* %JoinFKPK(main\_discharge\_point,deleted," = "," AND") \*/

main\_discharge\_point.garbage\_type\_ID = deleted.garbage\_type\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update garbage\_type because main\_discharge\_point exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* garbage\_type garbage on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="garbage\_type"

CHILD\_OWNER="", CHILD\_TABLE="garbage"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_19", FK\_COLUMNS="garbage\_type\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(garbage\_type\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,garbage

WHERE

/\* %JoinFKPK(garbage,deleted," = "," AND") \*/

garbage.garbage\_type\_ID = deleted.garbage\_type\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update garbage\_type because garbage exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_location ON location FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on location \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* location garbage on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0003295d", PARENT\_OWNER="", PARENT\_TABLE="location"

CHILD\_OWNER="", CHILD\_TABLE="garbage"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_23", FK\_COLUMNS="location\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,garbage

WHERE

/\* %JoinFKPK(garbage,deleted," = "," AND") \*/

garbage.location\_ID = deleted.location\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete location because garbage exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* region location on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="region"

CHILD\_OWNER="", CHILD\_TABLE="location"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_8", FK\_COLUMNS="region\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,region

WHERE

/\* %JoinFKPK(deleted,region," = "," AND") \*/

deleted.region\_ID = region.region\_ID AND

NOT EXISTS (

SELECT \* FROM location

WHERE

/\* %JoinFKPK(location,region," = "," AND") \*/

location.region\_ID = region.region\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last location because region exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* building location on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="building"

CHILD\_OWNER="", CHILD\_TABLE="location"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_9", FK\_COLUMNS="building\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,building

WHERE

/\* %JoinFKPK(deleted,building," = "," AND") \*/

deleted.building\_ID = building.building\_ID AND

NOT EXISTS (

SELECT \* FROM location

WHERE

/\* %JoinFKPK(location,building," = "," AND") \*/

location.building\_ID = building.building\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last location because building exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_location ON location FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on location \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@inslocation\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* location garbage on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0003908d", PARENT\_OWNER="", PARENT\_TABLE="location"

CHILD\_OWNER="", CHILD\_TABLE="garbage"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_23", FK\_COLUMNS="location\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(location\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,garbage

WHERE

/\* %JoinFKPK(garbage,deleted," = "," AND") \*/

garbage.location\_ID = deleted.location\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update location because garbage exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* region location on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="region"

CHILD\_OWNER="", CHILD\_TABLE="location"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_8", FK\_COLUMNS="region\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(region\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,region

WHERE

/\* %JoinFKPK(inserted,region) \*/

inserted.region\_ID = region.region\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update location because region does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* building location on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="building"

CHILD\_OWNER="", CHILD\_TABLE="location"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_9", FK\_COLUMNS="building\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(building\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,building

WHERE

/\* %JoinFKPK(inserted,building) \*/

inserted.building\_ID = building.building\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update location because building does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_main\_discharge\_point ON main\_discharge\_point FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on main\_discharge\_point \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* main\_discharge\_point main\_discharge\_point\_record on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0003e57e", PARENT\_OWNER="", PARENT\_TABLE="main\_discharge\_point"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point\_record"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_49", FK\_COLUMNS="main\_discharge\_point\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,main\_discharge\_point\_record

WHERE

/\* %JoinFKPK(main\_discharge\_point\_record,deleted," = "," AND") \*/

main\_discharge\_point\_record.main\_discharge\_point\_ID = deleted.main\_discharge\_point\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete main\_discharge\_point because main\_discharge\_point\_record exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* garbage\_type main\_discharge\_point on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="garbage\_type"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_1", FK\_COLUMNS="garbage\_type\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,garbage\_type

WHERE

/\* %JoinFKPK(deleted,garbage\_type," = "," AND") \*/

deleted.garbage\_type\_ID = garbage\_type.garbage\_type\_ID AND

NOT EXISTS (

SELECT \* FROM main\_discharge\_point

WHERE

/\* %JoinFKPK(main\_discharge\_point,garbage\_type," = "," AND") \*/

main\_discharge\_point.garbage\_type\_ID = garbage\_type.garbage\_type\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last main\_discharge\_point because garbage\_type exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* capacity main\_discharge\_point on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="capacity"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_17", FK\_COLUMNS="capacity\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,capacity

WHERE

/\* %JoinFKPK(deleted,capacity," = "," AND") \*/

deleted.capacity\_ID = capacity.capacity\_ID AND

NOT EXISTS (

SELECT \* FROM main\_discharge\_point

WHERE

/\* %JoinFKPK(main\_discharge\_point,capacity," = "," AND") \*/

main\_discharge\_point.capacity\_ID = capacity.capacity\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last main\_discharge\_point because capacity exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_main\_discharge\_point ON main\_discharge\_point FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on main\_discharge\_point \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@insmain\_discharge\_point\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* main\_discharge\_point main\_discharge\_point\_record on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="000437bb", PARENT\_OWNER="", PARENT\_TABLE="main\_discharge\_point"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point\_record"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_49", FK\_COLUMNS="main\_discharge\_point\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(main\_discharge\_point\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,main\_discharge\_point\_record

WHERE

/\* %JoinFKPK(main\_discharge\_point\_record,deleted," = "," AND") \*/

main\_discharge\_point\_record.main\_discharge\_point\_ID = deleted.main\_discharge\_point\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update main\_discharge\_point because main\_discharge\_point\_record exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* garbage\_type main\_discharge\_point on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="garbage\_type"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_1", FK\_COLUMNS="garbage\_type\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(garbage\_type\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,garbage\_type

WHERE

/\* %JoinFKPK(inserted,garbage\_type) \*/

inserted.garbage\_type\_ID = garbage\_type.garbage\_type\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update main\_discharge\_point because garbage\_type does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* capacity main\_discharge\_point on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="capacity"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_17", FK\_COLUMNS="capacity\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(capacity\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,capacity

WHERE

/\* %JoinFKPK(inserted,capacity) \*/

inserted.capacity\_ID = capacity.capacity\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update main\_discharge\_point because capacity does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_main\_discharge\_point\_record ON main\_discharge\_point\_record FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on main\_discharge\_point\_record \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* main\_discharge\_point main\_discharge\_point\_record on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0002f101", PARENT\_OWNER="", PARENT\_TABLE="main\_discharge\_point"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point\_record"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_49", FK\_COLUMNS="main\_discharge\_point\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,main\_discharge\_point

WHERE

/\* %JoinFKPK(deleted,main\_discharge\_point," = "," AND") \*/

deleted.main\_discharge\_point\_ID = main\_discharge\_point.main\_discharge\_point\_ID AND

NOT EXISTS (

SELECT \* FROM main\_discharge\_point\_record

WHERE

/\* %JoinFKPK(main\_discharge\_point\_record,main\_discharge\_point," = "," AND") \*/

main\_discharge\_point\_record.main\_discharge\_point\_ID = main\_discharge\_point.main\_discharge\_point\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last main\_discharge\_point\_record because main\_discharge\_point exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* record main\_discharge\_point\_record on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="record"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point\_record"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_51", FK\_COLUMNS="record\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,record

WHERE

/\* %JoinFKPK(deleted,record," = "," AND") \*/

deleted.record\_ID = record.record\_ID AND

NOT EXISTS (

SELECT \* FROM main\_discharge\_point\_record

WHERE

/\* %JoinFKPK(main\_discharge\_point\_record,record," = "," AND") \*/

main\_discharge\_point\_record.record\_ID = record.record\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last main\_discharge\_point\_record because record exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_main\_discharge\_point\_record ON main\_discharge\_point\_record FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on main\_discharge\_point\_record \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@insmain\_discharge\_point\_ID char(18),

@insrecord\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* main\_discharge\_point main\_discharge\_point\_record on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0002ed92", PARENT\_OWNER="", PARENT\_TABLE="main\_discharge\_point"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point\_record"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_49", FK\_COLUMNS="main\_discharge\_point\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(main\_discharge\_point\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,main\_discharge\_point

WHERE

/\* %JoinFKPK(inserted,main\_discharge\_point) \*/

inserted.main\_discharge\_point\_ID = main\_discharge\_point.main\_discharge\_point\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update main\_discharge\_point\_record because main\_discharge\_point does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* record main\_discharge\_point\_record on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="record"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point\_record"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_51", FK\_COLUMNS="record\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(record\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,record

WHERE

/\* %JoinFKPK(inserted,record) \*/

inserted.record\_ID = record.record\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update main\_discharge\_point\_record because record does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_record ON record FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on record \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* record main\_discharge\_point\_record on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00012189", PARENT\_OWNER="", PARENT\_TABLE="record"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point\_record"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_51", FK\_COLUMNS="record\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,main\_discharge\_point\_record

WHERE

/\* %JoinFKPK(main\_discharge\_point\_record,deleted," = "," AND") \*/

main\_discharge\_point\_record.record\_ID = deleted.record\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete record because main\_discharge\_point\_record exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_record ON record FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on record \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@insrecord\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* record main\_discharge\_point\_record on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00013aa2", PARENT\_OWNER="", PARENT\_TABLE="record"

CHILD\_OWNER="", CHILD\_TABLE="main\_discharge\_point\_record"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_51", FK\_COLUMNS="record\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(record\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,main\_discharge\_point\_record

WHERE

/\* %JoinFKPK(main\_discharge\_point\_record,deleted," = "," AND") \*/

main\_discharge\_point\_record.record\_ID = deleted.record\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update record because main\_discharge\_point\_record exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_region ON region FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on region \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* region location on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0000ead4", PARENT\_OWNER="", PARENT\_TABLE="region"

CHILD\_OWNER="", CHILD\_TABLE="location"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_8", FK\_COLUMNS="region\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,location

WHERE

/\* %JoinFKPK(location,deleted," = "," AND") \*/

location.region\_ID = deleted.region\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete region because location exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_region ON region FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on region \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@insregion\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* region location on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00010613", PARENT\_OWNER="", PARENT\_TABLE="region"

CHILD\_OWNER="", CHILD\_TABLE="location"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_8", FK\_COLUMNS="region\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(region\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,location

WHERE

/\* %JoinFKPK(location,deleted," = "," AND") \*/

location.region\_ID = deleted.region\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update region because location exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_storey ON storey FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on storey \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* storey building on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0000eb7d", PARENT\_OWNER="", PARENT\_TABLE="storey"

CHILD\_OWNER="", CHILD\_TABLE="building"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_6", FK\_COLUMNS="storey\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,building

WHERE

/\* %JoinFKPK(building,deleted," = "," AND") \*/

building.storey\_ID = deleted.storey\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete storey because building exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_storey ON storey FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on storey \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@insstorey\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* storey building on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00010a3d", PARENT\_OWNER="", PARENT\_TABLE="storey"

CHILD\_OWNER="", CHILD\_TABLE="building"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_6", FK\_COLUMNS="storey\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(storey\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,building

WHERE

/\* %JoinFKPK(building,deleted," = "," AND") \*/

building.storey\_ID = deleted.storey\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update storey because building exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_trash\_company ON trash\_company FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on trash\_company \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* trash\_company trash\_type on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00010675", PARENT\_OWNER="", PARENT\_TABLE="trash\_company"

CHILD\_OWNER="", CHILD\_TABLE="trash\_type"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_105", FK\_COLUMNS="trash\_company\_id" \*/

IF EXISTS (

SELECT \* FROM deleted,trash\_type

WHERE

/\* %JoinFKPK(trash\_type,deleted," = "," AND") \*/

trash\_type.trash\_company\_id = deleted.trash\_company\_id

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete trash\_company because trash\_type exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_trash\_company ON trash\_company FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on trash\_company \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@instrash\_company\_id char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* trash\_company trash\_type on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00011e69", PARENT\_OWNER="", PARENT\_TABLE="trash\_company"

CHILD\_OWNER="", CHILD\_TABLE="trash\_type"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_105", FK\_COLUMNS="trash\_company\_id" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(trash\_company\_id)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,trash\_type

WHERE

/\* %JoinFKPK(trash\_type,deleted," = "," AND") \*/

trash\_type.trash\_company\_id = deleted.trash\_company\_id

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update trash\_company because trash\_type exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tD\_trash\_type ON trash\_type FOR DELETE AS

/\* ERwin Builtin Trigger \*/

/\* DELETE trigger on trash\_type \*/

BEGIN

DECLARE @errno int,

@errmsg varchar(255)

/\* ERwin Builtin Trigger \*/

/\* trash\_type garbage on parent delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="000374a9", PARENT\_OWNER="", PARENT\_TABLE="trash\_type"

CHILD\_OWNER="", CHILD\_TABLE="garbage"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_20", FK\_COLUMNS="trash\_type\_ID" \*/

IF EXISTS (

SELECT \* FROM deleted,garbage

WHERE

/\* %JoinFKPK(garbage,deleted," = "," AND") \*/

garbage.trash\_type\_ID = deleted.trash\_type\_ID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete trash\_type because garbage exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* capacity trash\_type on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="capacity"

CHILD\_OWNER="", CHILD\_TABLE="trash\_type"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_18", FK\_COLUMNS="capacity\_ID" \*/

IF EXISTS (SELECT \* FROM deleted,capacity

WHERE

/\* %JoinFKPK(deleted,capacity," = "," AND") \*/

deleted.capacity\_ID = capacity.capacity\_ID AND

NOT EXISTS (

SELECT \* FROM trash\_type

WHERE

/\* %JoinFKPK(trash\_type,capacity," = "," AND") \*/

trash\_type.capacity\_ID = capacity.capacity\_ID

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last trash\_type because capacity exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

/\* trash\_company trash\_type on child delete no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="trash\_company"

CHILD\_OWNER="", CHILD\_TABLE="trash\_type"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_105", FK\_COLUMNS="trash\_company\_id" \*/

IF EXISTS (SELECT \* FROM deleted,trash\_company

WHERE

/\* %JoinFKPK(deleted,trash\_company," = "," AND") \*/

deleted.trash\_company\_id = trash\_company.trash\_company\_id AND

NOT EXISTS (

SELECT \* FROM trash\_type

WHERE

/\* %JoinFKPK(trash\_type,trash\_company," = "," AND") \*/

trash\_type.trash\_company\_id = trash\_company.trash\_company\_id

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last trash\_type because trash\_company exists.'

GOTO ERROR

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

CREATE TRIGGER tU\_trash\_type ON trash\_type FOR UPDATE AS

/\* ERwin Builtin Trigger \*/

/\* UPDATE trigger on trash\_type \*/

BEGIN

DECLARE @NUMROWS int,

@nullcnt int,

@validcnt int,

@instrash\_type\_ID char(18),

@errno int,

@errmsg varchar(255)

SELECT @NUMROWS = @@rowcount

/\* ERwin Builtin Trigger \*/

/\* trash\_type garbage on parent update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="0003f643", PARENT\_OWNER="", PARENT\_TABLE="trash\_type"

CHILD\_OWNER="", CHILD\_TABLE="garbage"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_20", FK\_COLUMNS="trash\_type\_ID" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(trash\_type\_ID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,garbage

WHERE

/\* %JoinFKPK(garbage,deleted," = "," AND") \*/

garbage.trash\_type\_ID = deleted.trash\_type\_ID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update trash\_type because garbage exists.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

/\* capacity trash\_type on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="capacity"

CHILD\_OWNER="", CHILD\_TABLE="trash\_type"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_18", FK\_COLUMNS="capacity\_ID" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(capacity\_ID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,capacity

WHERE

/\* %JoinFKPK(inserted,capacity) \*/

inserted.capacity\_ID = capacity.capacity\_ID

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

/\* ERwin Builtin Trigger \*/

/\* trash\_company trash\_type on child update no action \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="trash\_company"

CHILD\_OWNER="", CHILD\_TABLE="trash\_type"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="R\_105", FK\_COLUMNS="trash\_company\_id" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(trash\_company\_id)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,trash\_company

WHERE

/\* %JoinFKPK(inserted,trash\_company) \*/

inserted.trash\_company\_id = trash\_company.trash\_company\_id

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

select @nullcnt = count(\*) from inserted where

inserted.trash\_company\_id IS NULL

IF @validcnt + @nullcnt != @NUMROWS

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update trash\_type because trash\_company does not exist.'

GOTO ERROR

END

END

/\* ERwin Builtin Trigger \*/

RETURN

ERROR:

raiserror @errno @errmsg

rollback transaction

END

go

# **SQL EXPERIENCE TO THREE TABLE**

## **A)DDL STATEMENTS**

### **A.1)CREATE/ALTER VIEW EXAMPLES**

CREATE VIEW [dbo].[BIN\_GETIR]

AS

select BIN\_ID,Garbage\_Type\_Name,Garbage\_Type\_Definition,

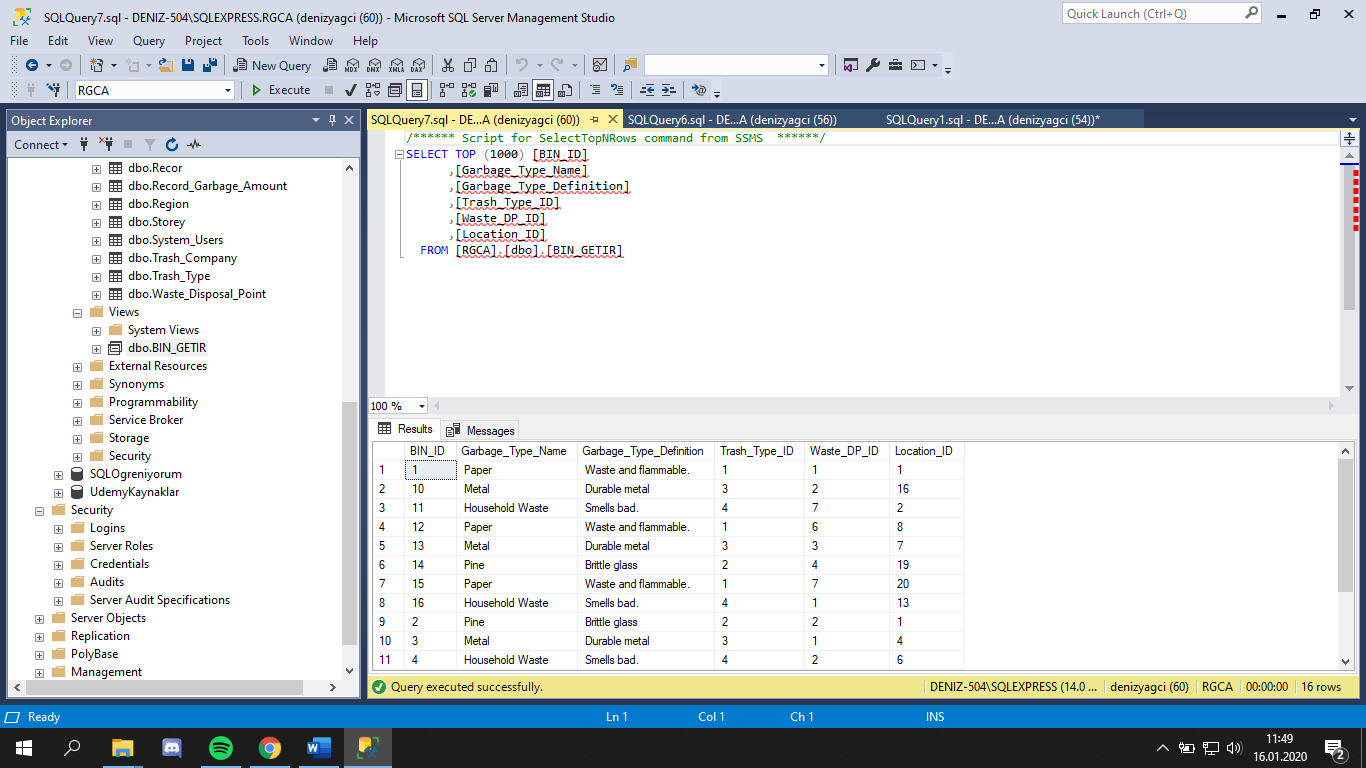
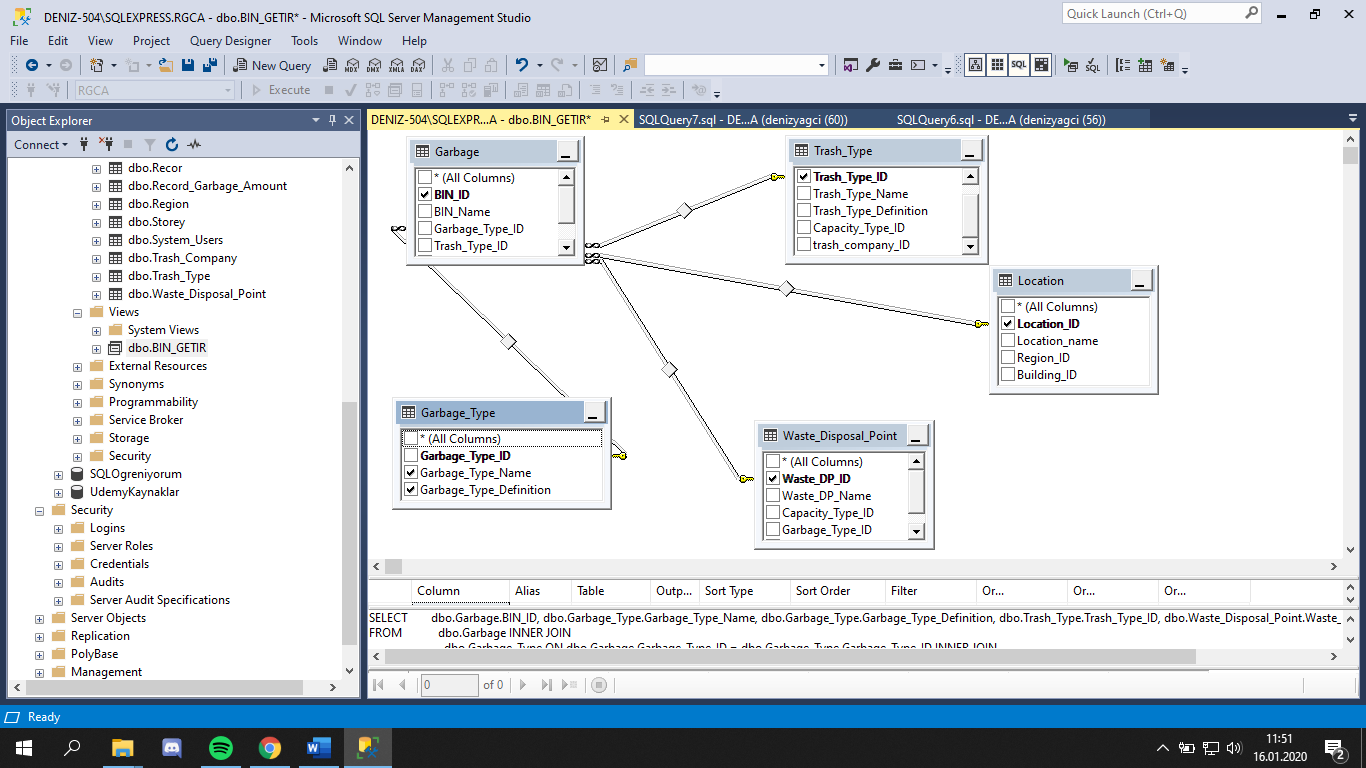
Trash\_Type.Trash\_Type\_ID,Waste\_Disposal\_Point.Waste\_DP\_ID,Location.Location\_ID from Garbage

inner join Garbage\_Type on Garbage.Garbage\_Type\_ID=Garbage\_Type.Garbage\_Type\_ID

inner join Trash\_Type on Garbage.Trash\_Type\_ID=Trash\_Type.Trash\_Type\_ID

inner join Location on Garbage.Location\_ID=Location.Location\_ID

inner join Waste\_Disposal\_Point on Garbage.Waste\_DP\_ID=Waste\_Disposal\_Point.Waste\_DP\_ID

GO

ALTER VIEW [dbo].[BIN\_GETIR]

AS

select BIN\_ID,Garbage\_Type\_Name,Garbage\_Type\_Definition,

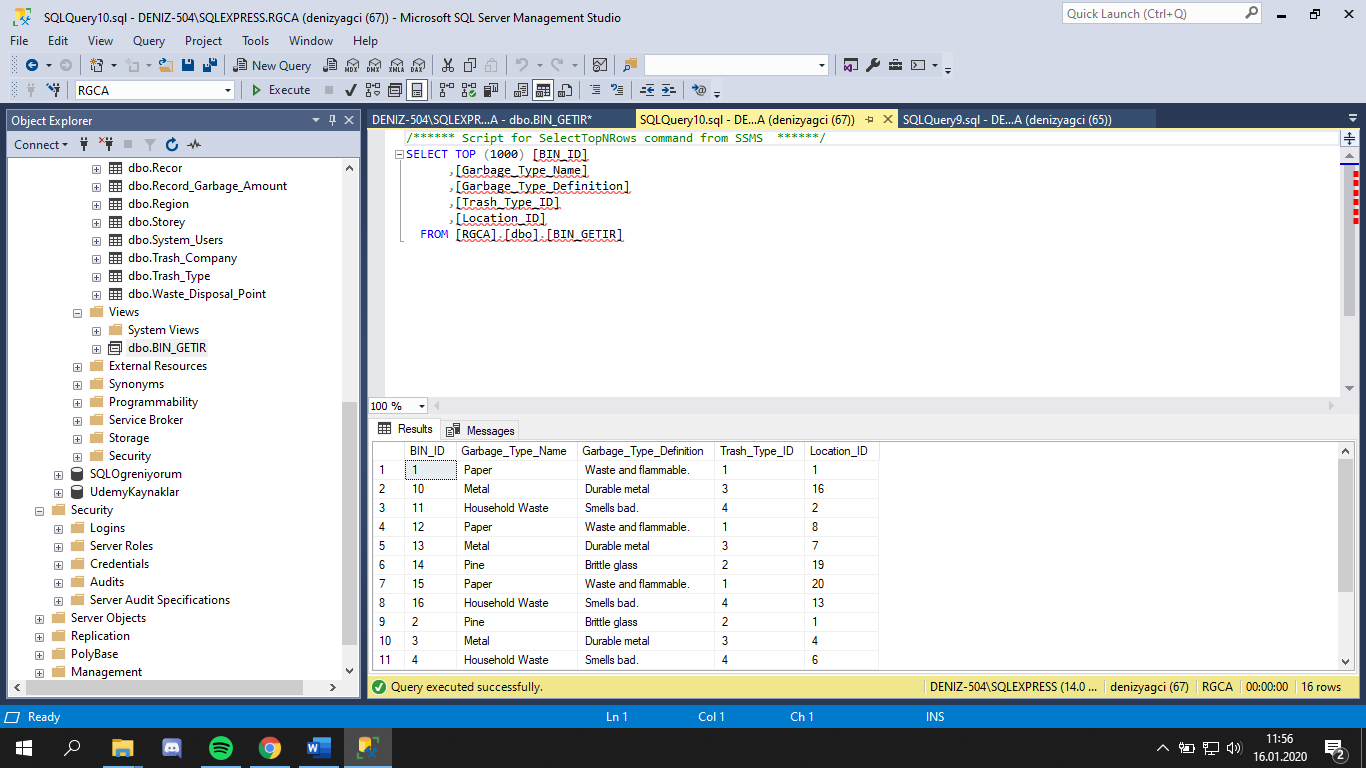
Trash\_Type.Trash\_Type\_ID,Location.Location\_ID from Garbage

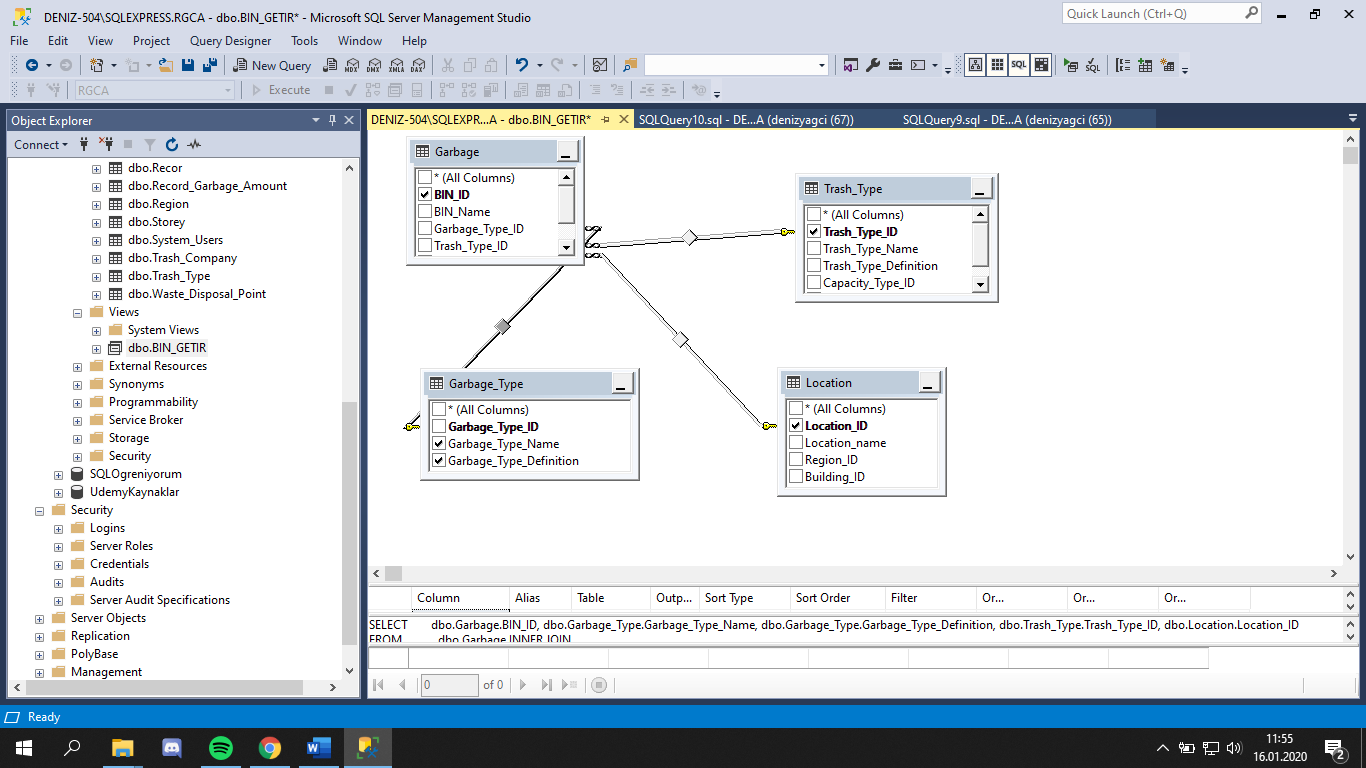
inner join Garbage\_Type on Garbage.Garbage\_Type\_ID=Garbage\_Type.Garbage\_Type\_ID

inner join Trash\_Type on Garbage.Trash\_Type\_ID=Trash\_Type.Trash\_Type\_ID

inner join Location on Garbage.Location\_ID=Location.Location\_ID

GO





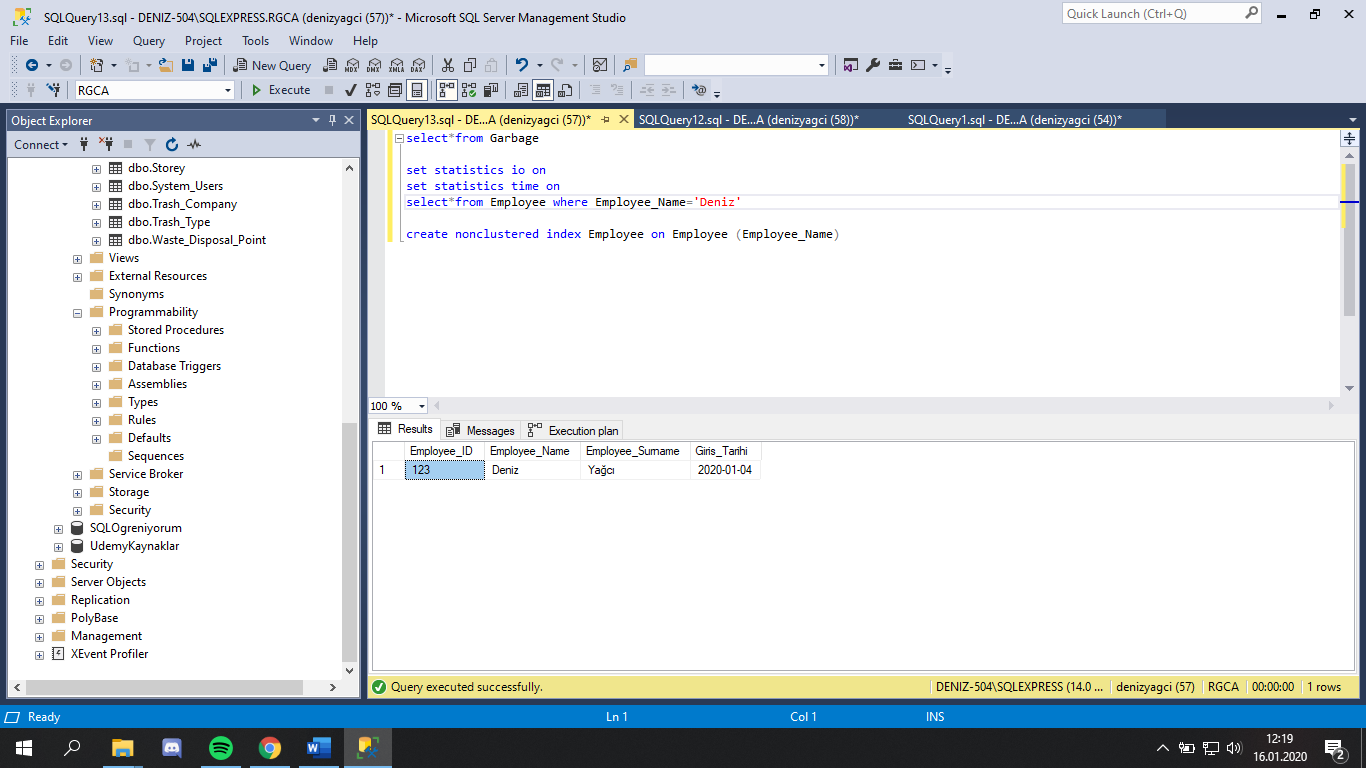
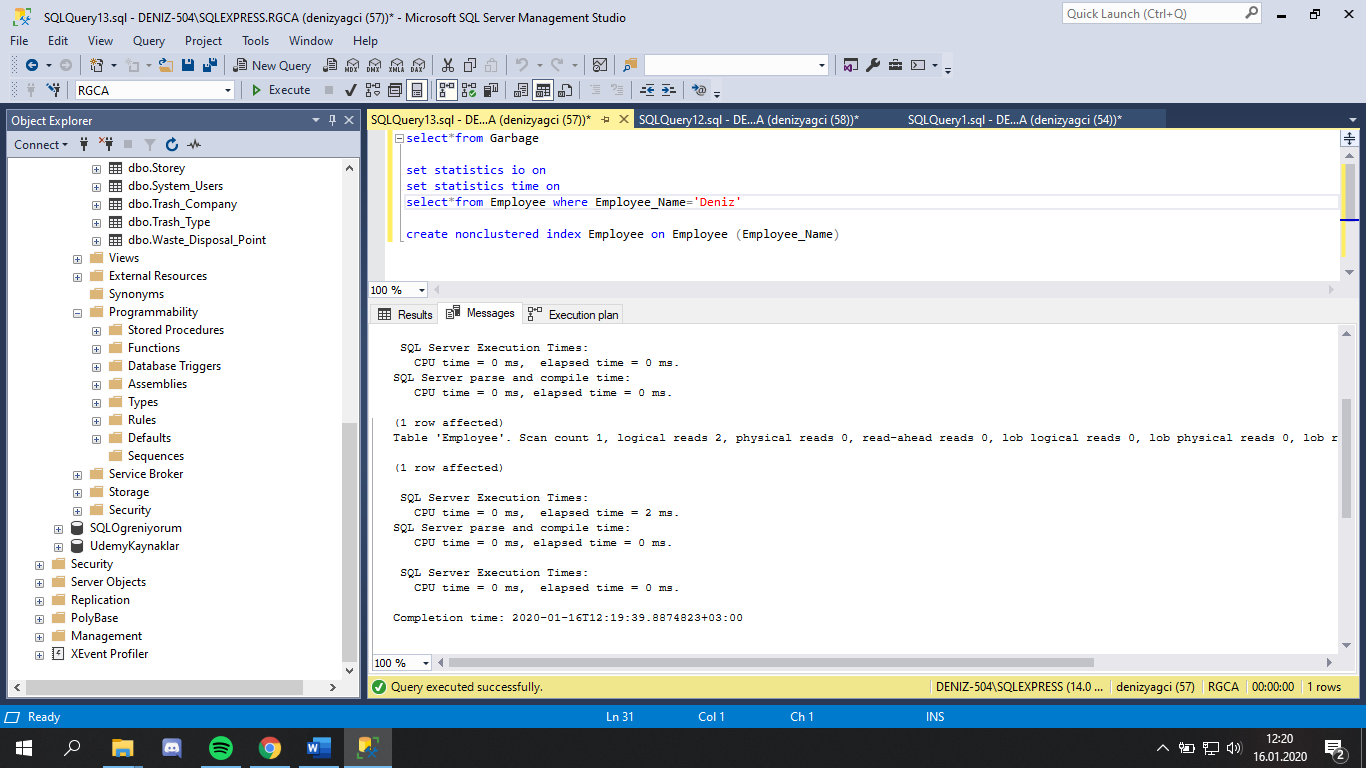
### **A.2)CREATE/ALTER INDEX EXAMPLES**

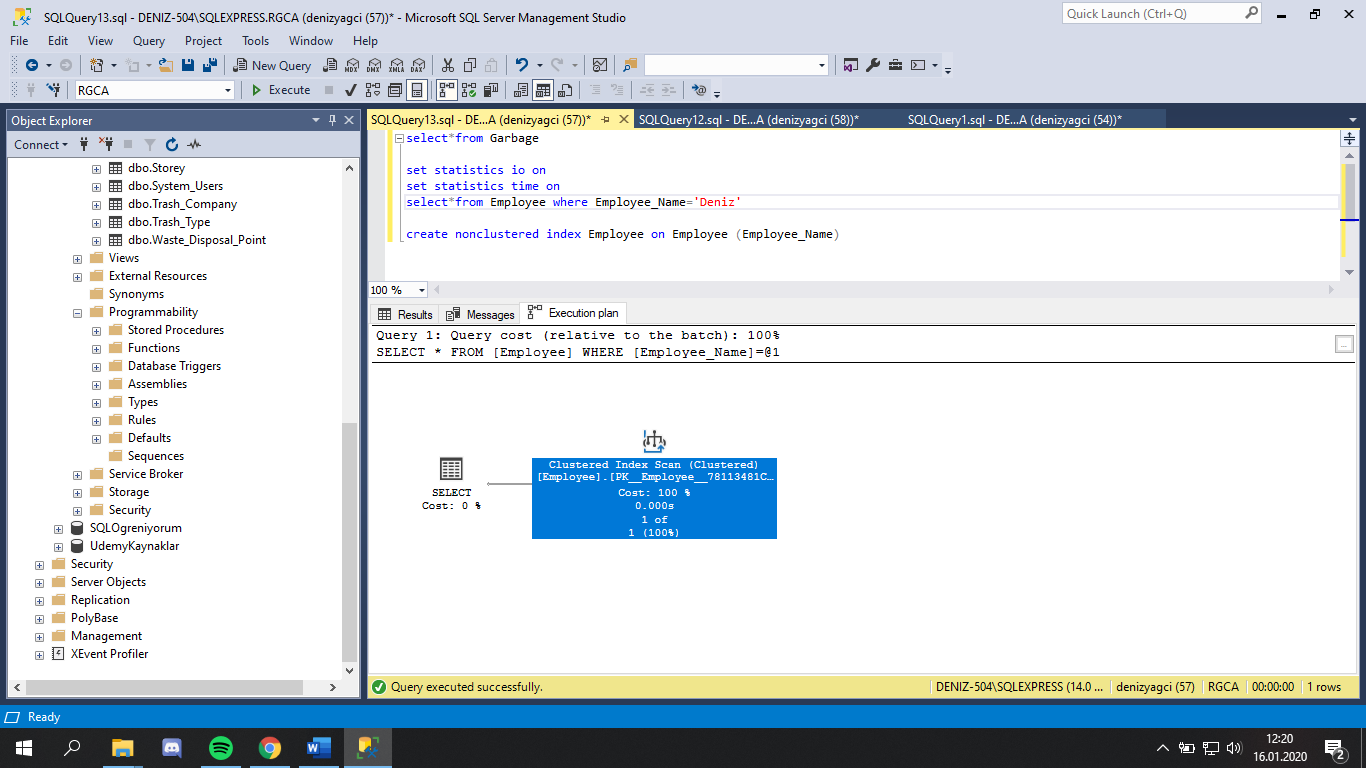
set statistics io on

set statistics time on

select\*from Employee where Employee\_Name='Deniz'

create nonclustered index Employee on Employee (Employee\_Name)



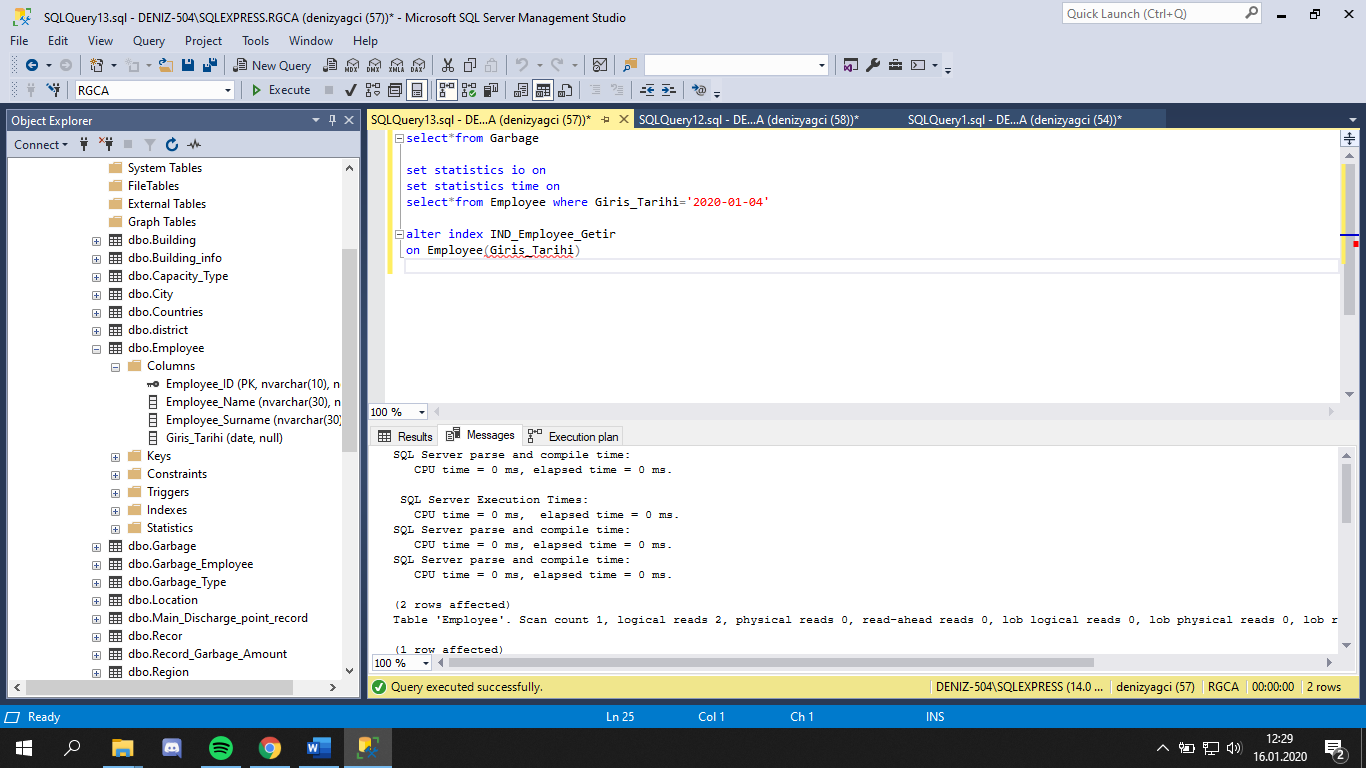
statistics io on

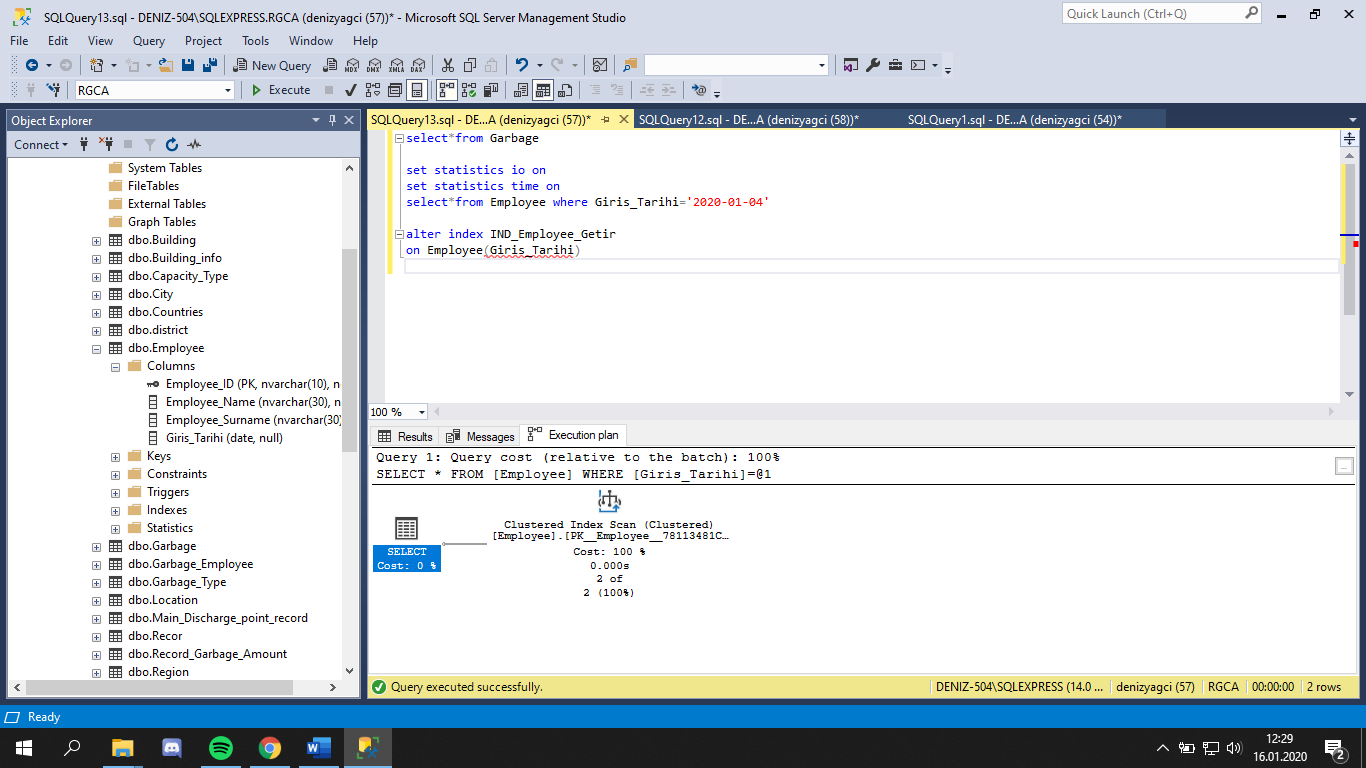
set statistics time on

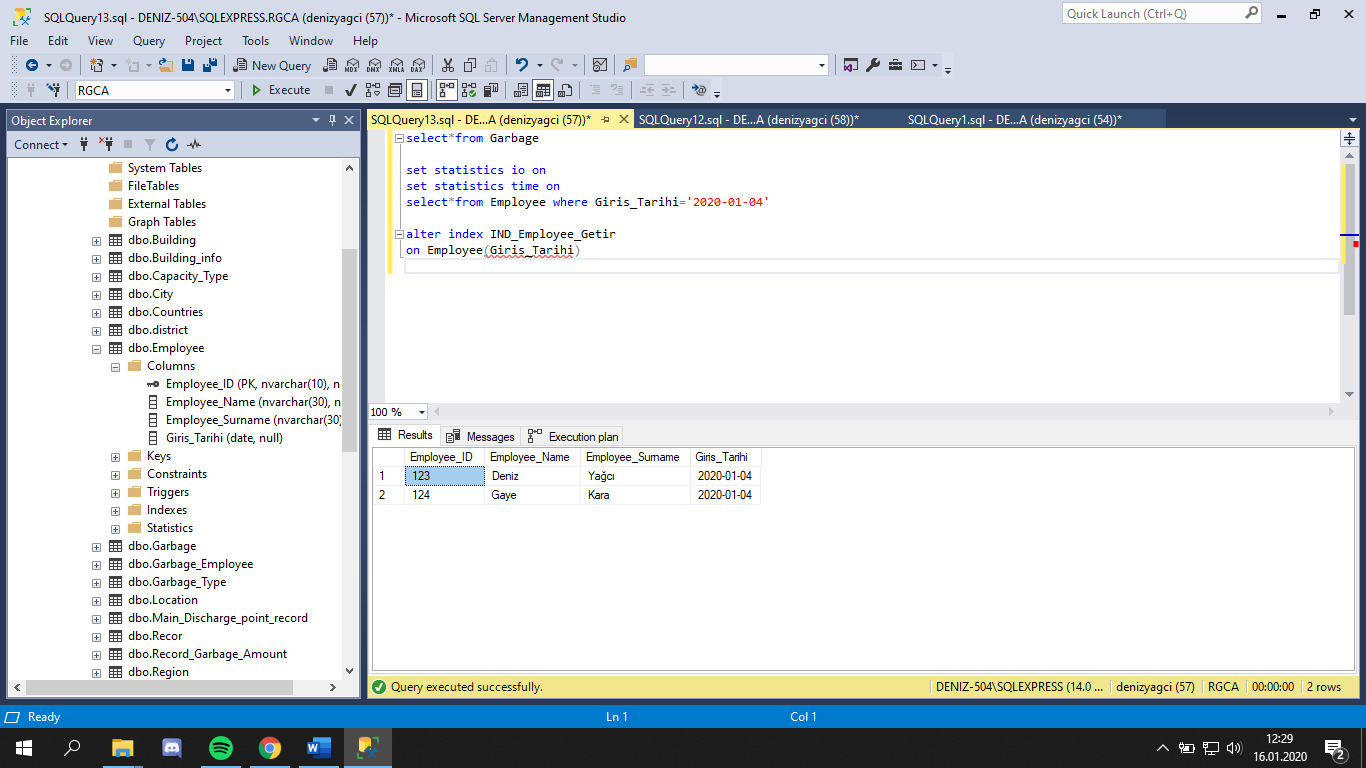
select\*from Employee where Giris\_Tarihi='2020-01-04'

alter index IND\_Employee\_Getir

on Employee(Giris\_Tarihi)







## **B)DML STATEMENTS-DATA MANIPULATION LANGUAGE**

### **B.1)INSERT INTO**

INSERT INTO [dbo].[Garbage]

([BIN\_ID]

,[BIN\_Name]

,[Garbage\_Type\_ID]

,[Trash\_Type\_ID]

,[Location\_ID]

,[Waste\_DP\_ID])

VALUES

(

'17',

'BIN Paper',

'1',

'1',

'15',

'2'

)

GO

## 

### **B.2)UPDATE STATEMENTS**

UPDATE [dbo].[Garbage]

SET [BIN\_ID] = '18'

,[BIN\_Name] = 'BIN Metal'

,[Garbage\_Type\_ID] = '3'

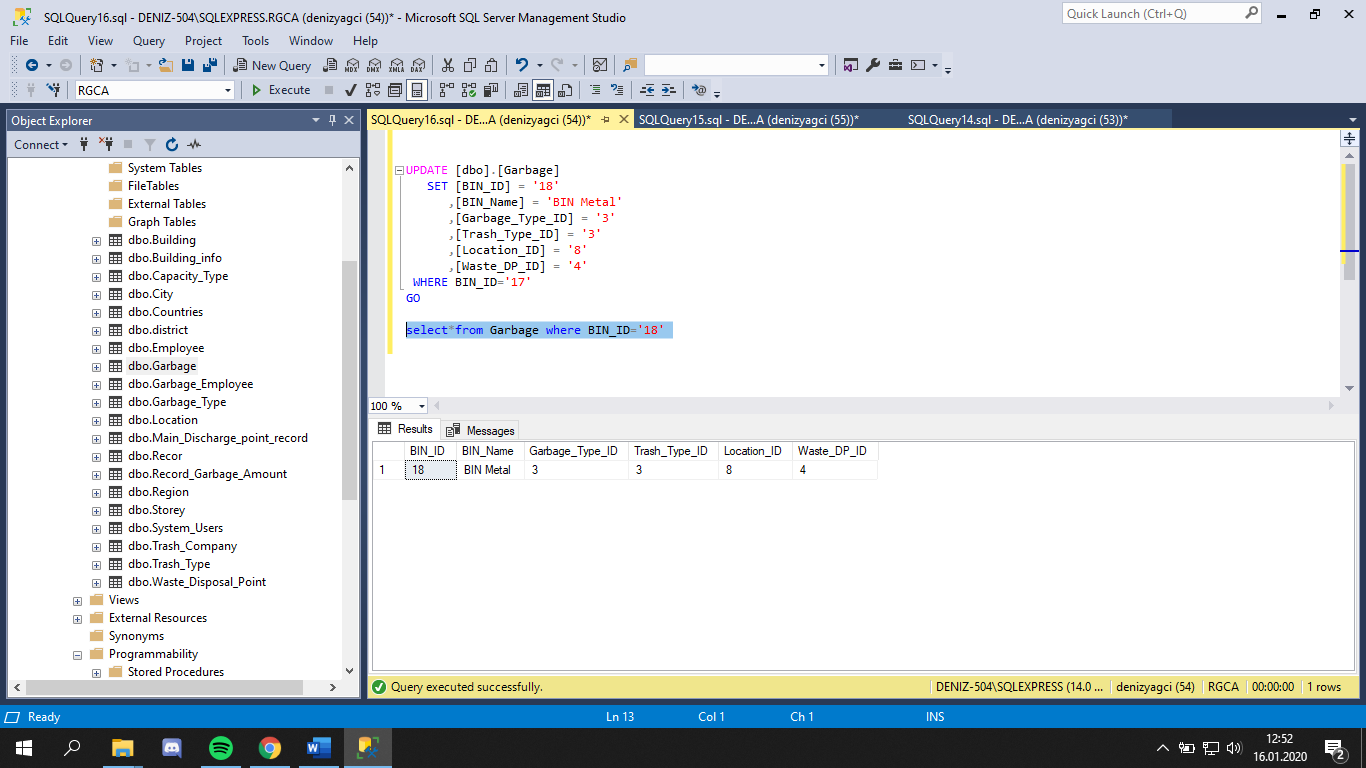
,[Trash\_Type\_ID] = '3'

,[Location\_ID] = '8'

,[Waste\_DP\_ID] = '4'

WHERE BIN\_ID='17'

GO

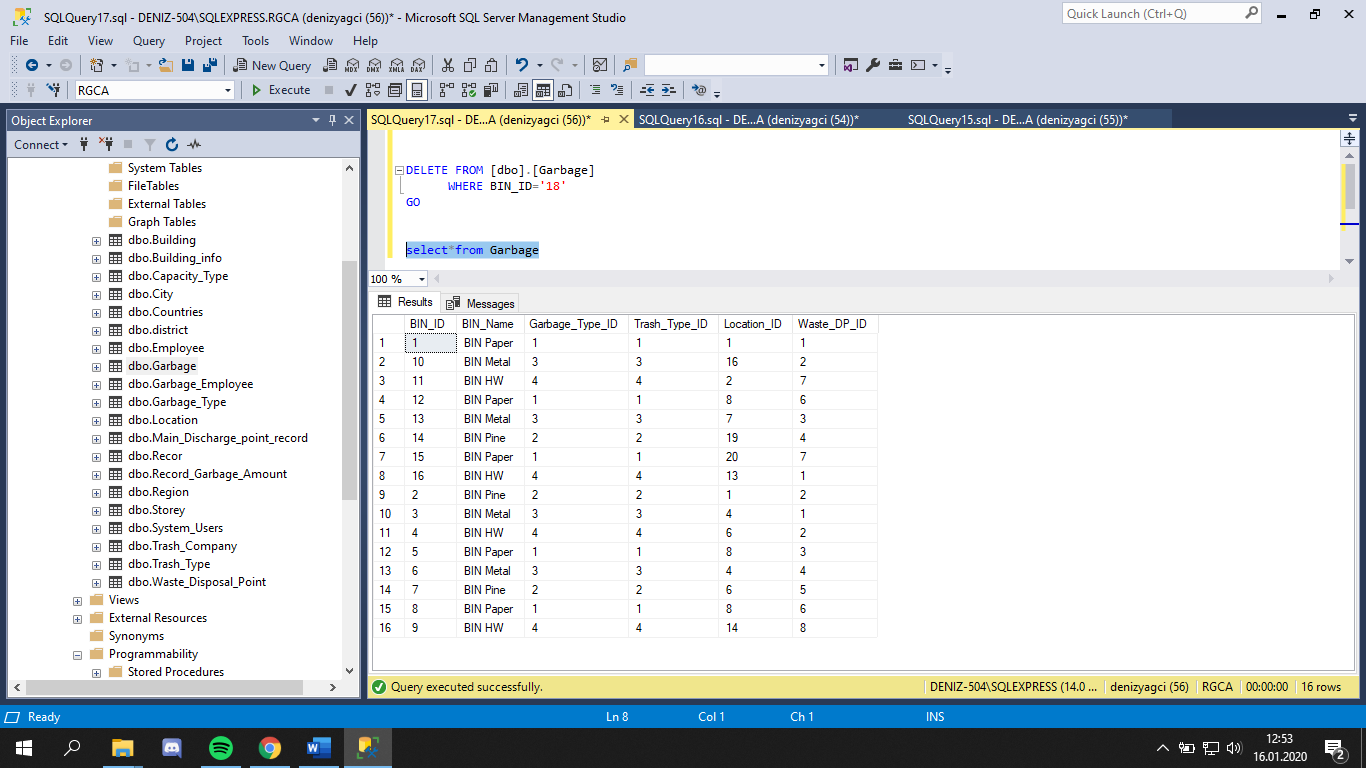


### **B.3)DELETE STATEMENTS**

DELETE FROM [dbo].[Garbage]

WHERE BIN\_ID='18'

GO



### **B.4)STORE PROCEDURE**

create proc [dbo].[SP\_Location\_Getir]

as

begin

select Location.Location\_ID, Location\_name,Region\_Name,Building\_Name, Storey\_Name from Location

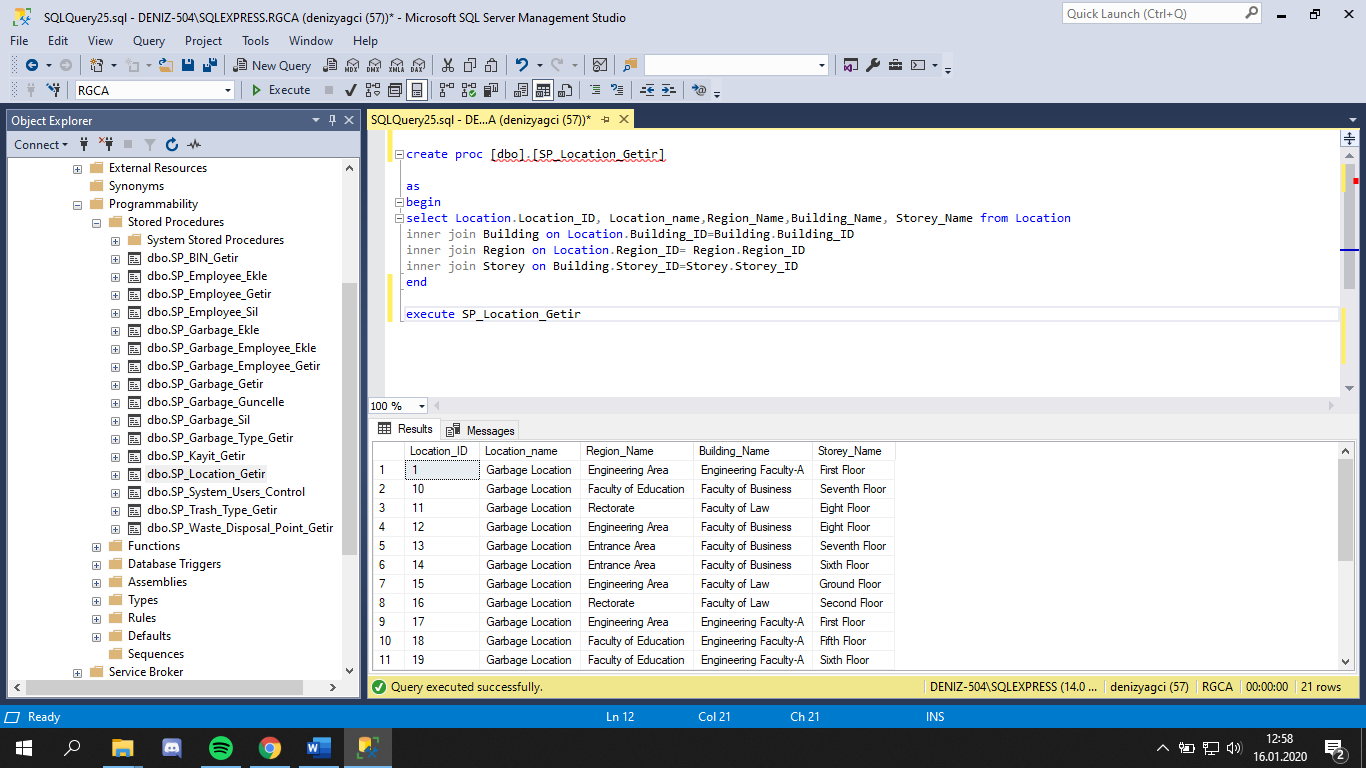
inner join Building on Location.Building\_ID=Building.Building\_ID

inner join Region on Location.Region\_ID= Region.Region\_ID

inner join Storey on Building.Storey\_ID=Storey.Storey\_ID

end

execute SP\_Location\_Getir



## **C)DQL STATEMENTS-DATA QUERY LANGUAGE**

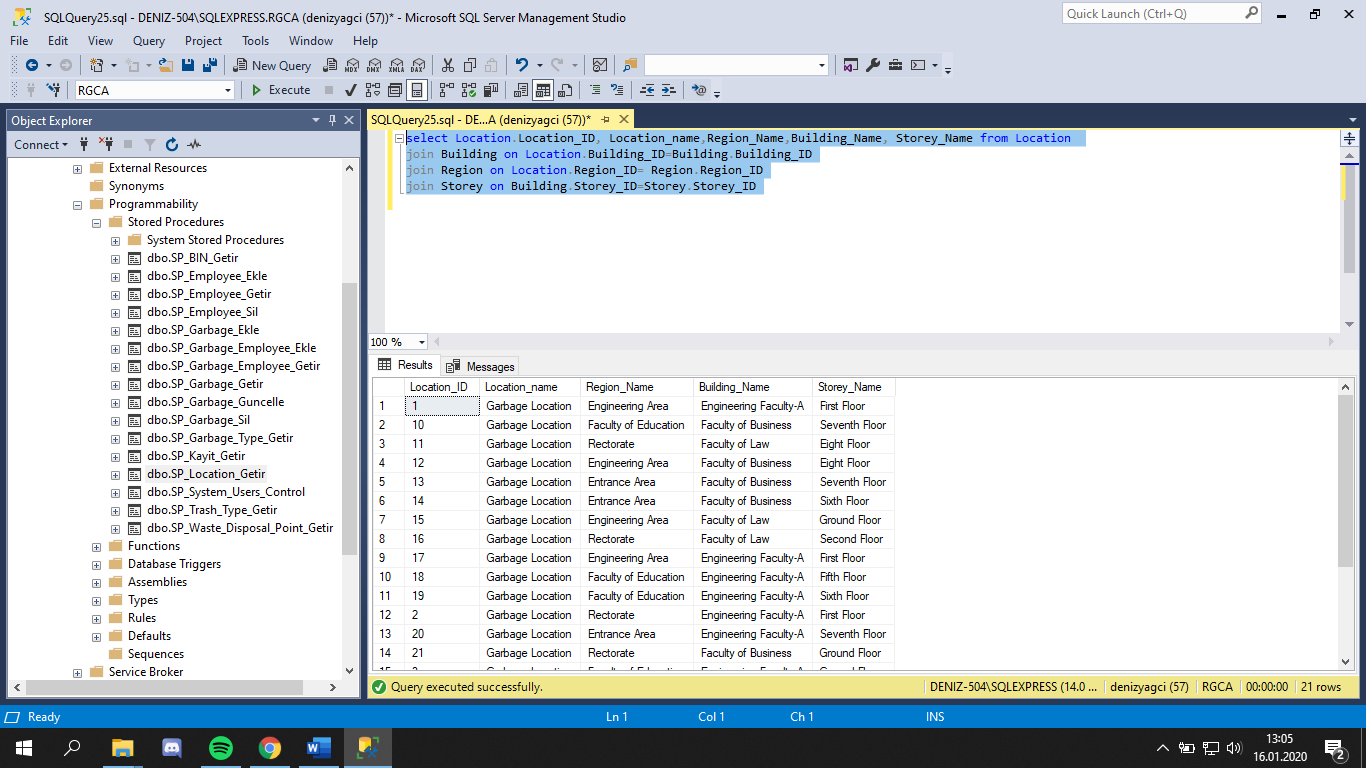
### **C.1)SIMPLE JOINS-JOINING TWO TABLES**

select Location.Location\_ID, Location\_name,Region\_Name,Building\_Name, Storey\_Name from Location

join Building on Location.Building\_ID=Building.Building\_ID

join Region on Location.Region\_ID= Region.Region\_ID

join Storey on Building.Storey\_ID=Storey.Storey\_ID

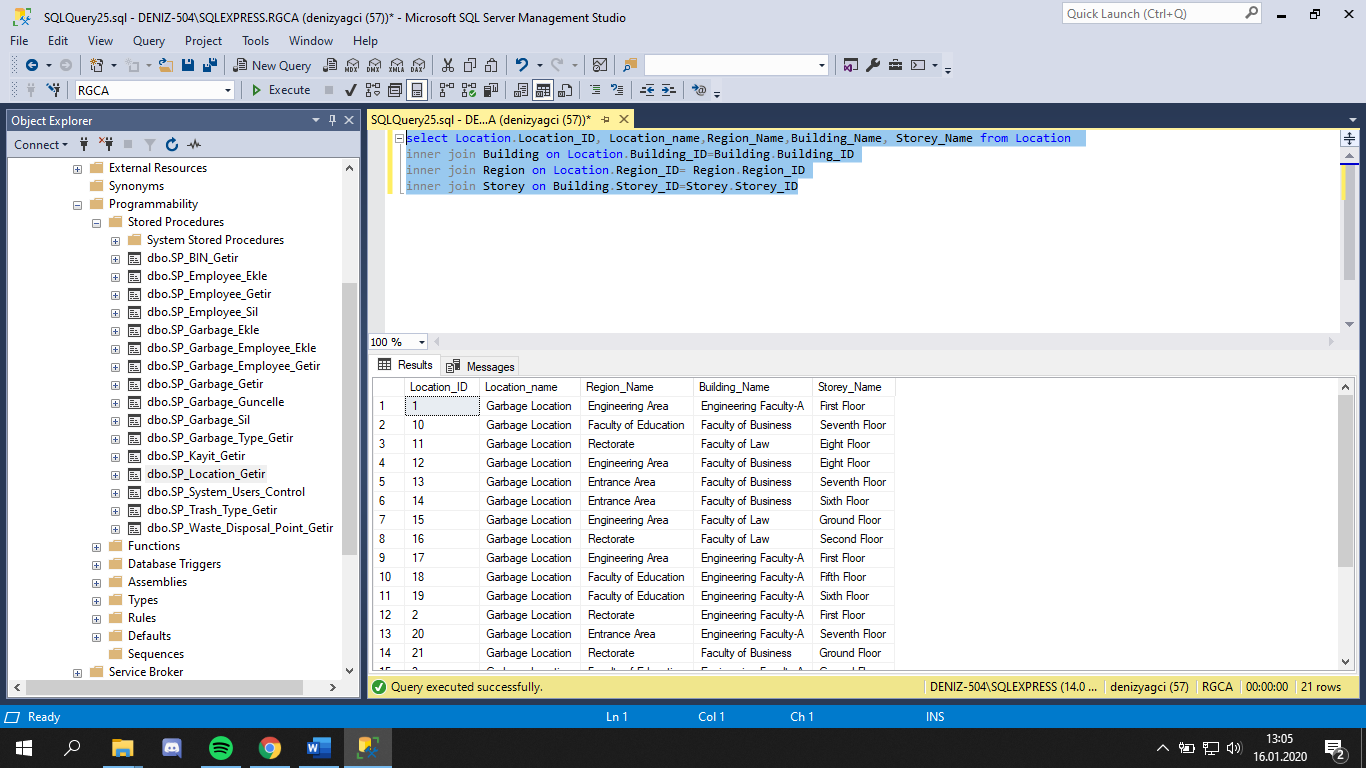


### **C.2)INNER/OUTER JOINS**

select Location.Location\_ID, Location\_name,Region\_Name,Building\_Name, Storey\_Name from Location

inner join Building on Location.Building\_ID=Building.Building\_ID

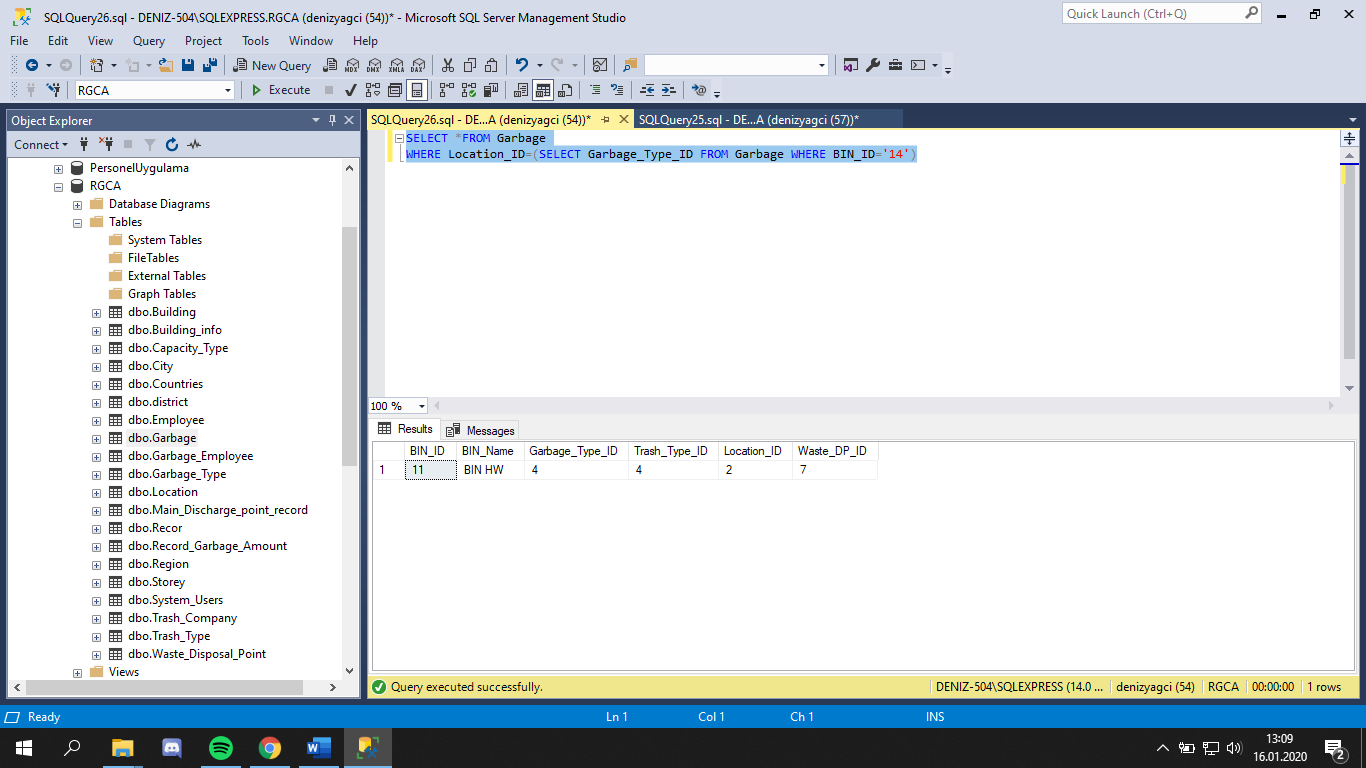
inner join Region on Location.Region\_ID= Region.Region\_ID

inner join Storey on Building.Storey\_ID=Storey.Storey\_ID 

### **C.3)NESTED QUERIES**

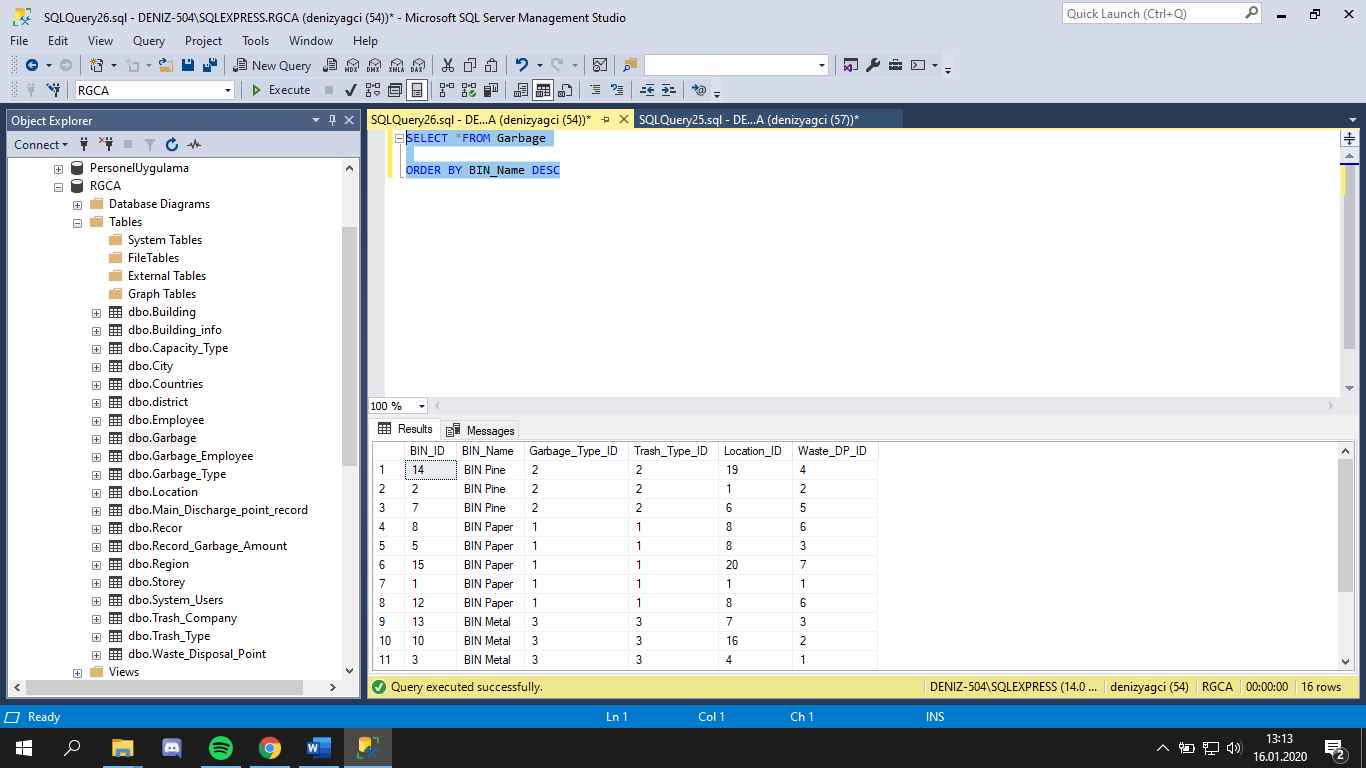
SELECT \*FROM Garbage

WHERE Location\_ID=(SELECT Garbage\_Type\_ID FROM Garbage WHERE BIN\_ID='14')



### **C.4)SORTING**

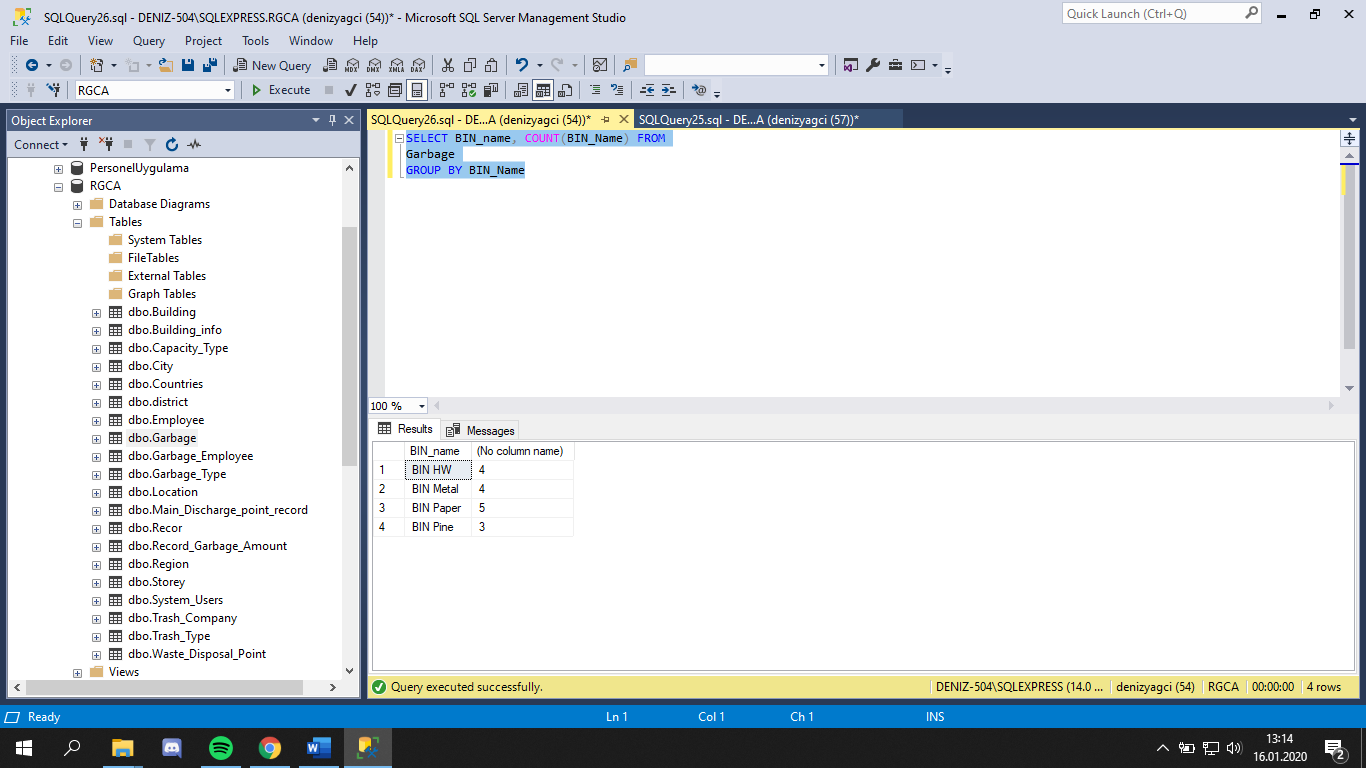
SELECT \*FROM Garbage

ORDER BY BIN\_Name DESC 

### **C.5)GROUP BY**

SELECT BIN\_name, COUNT(BIN\_Name) FROM

Garbage

GROUP BY BIN\_Name 

### **C.6)USING SELECT STATEMENTS WITH STORED PROCEDURE**

CREATE proc [dbo].[SP\_BIN\_Getir]

as

begin

select BIN\_ID,Garbage\_Type\_Name,Garbage\_Type\_Definition,

Trash\_Type.Trash\_Type\_ID,Waste\_Disposal\_Point.Waste\_DP\_ID,Location.Location\_ID from Garbage

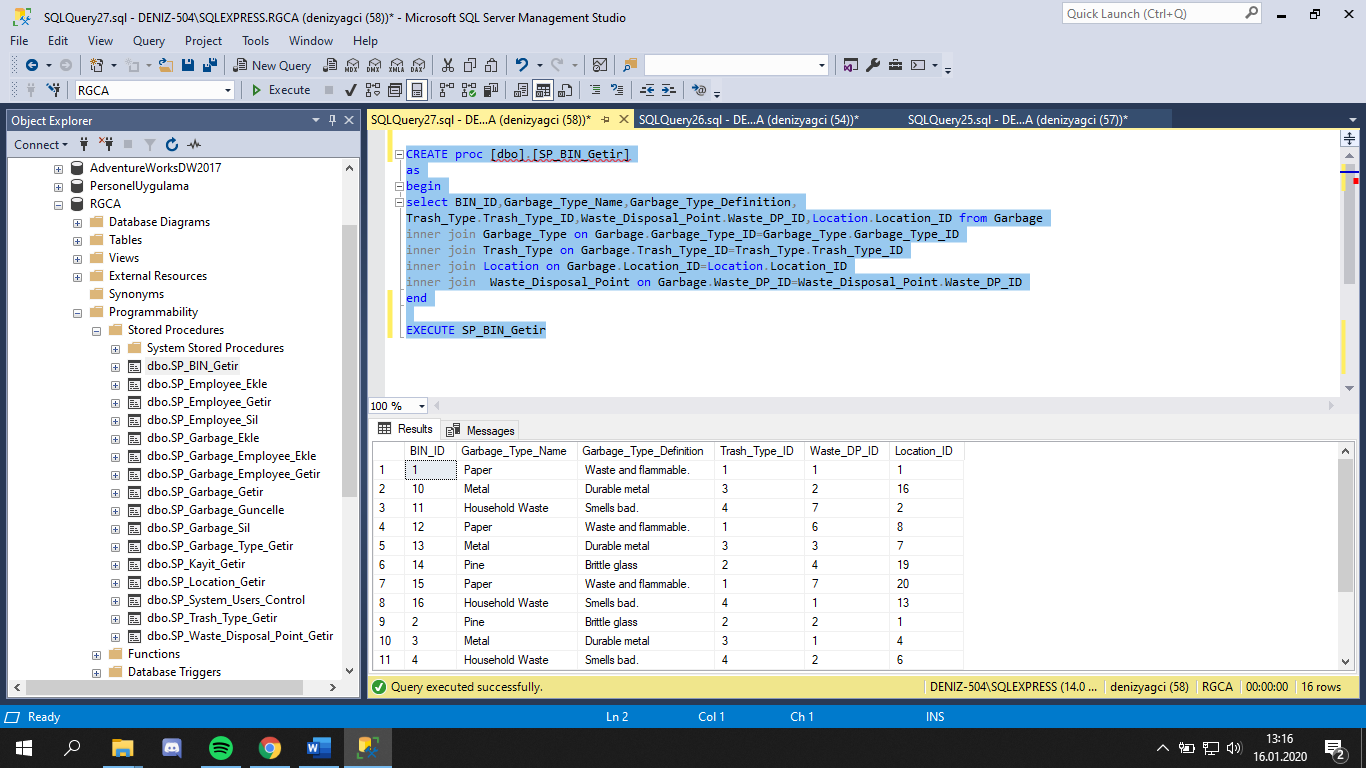
inner join Garbage\_Type on Garbage.Garbage\_Type\_ID=Garbage\_Type.Garbage\_Type\_ID

inner join Trash\_Type on Garbage.Trash\_Type\_ID=Trash\_Type.Trash\_Type\_ID

inner join Location on Garbage.Location\_ID=Location.Location\_ID

inner join Waste\_Disposal\_Point on Garbage.Waste\_DP\_ID=Waste\_Disposal\_Point.Waste\_DP\_ID

end

EXECUTE SP\_BIN\_Getir 

## **D)DCL STATEMENTS-DATA CONTROL LANGUAGE**

CREATE TRANSACTION

UPDATE System\_Users SET User\_Sifre='2'

WHERE User\_ID='1'

COMMIT

BEGIN CATCH

ROLLBACK

END CATCH



# **INTEGRETY TEST**

## **1)TEST 1**

CREATE TABLE [dbo].[Garbage\_Employee](

[Garbage\_Employee\_ID] [nvarchar](10) NOT NULL,

[BIN\_ID] [nvarchar](10) NULL,

[Employee\_ID] [nvarchar](10) NULL,

PRIMARY KEY CLUSTERED

(

[Garbage\_Employee\_ID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

ALTER TABLE [dbo].[Garbage\_Employee] WITH CHECK ADD FOREIGN KEY([Employee\_ID])

REFERENCES [dbo].[Employee] ([Employee\_ID])

GO

ALTER TABLE [dbo].[Garbage\_Employee] WITH CHECK ADD FOREIGN KEY([BIN\_ID])

REFERENCES [dbo].[Garbage] ([BIN\_ID])

GO

INSERT INTO [dbo].[Garbage\_Employee]

([Garbage\_Employee\_ID]

,[BIN\_ID]

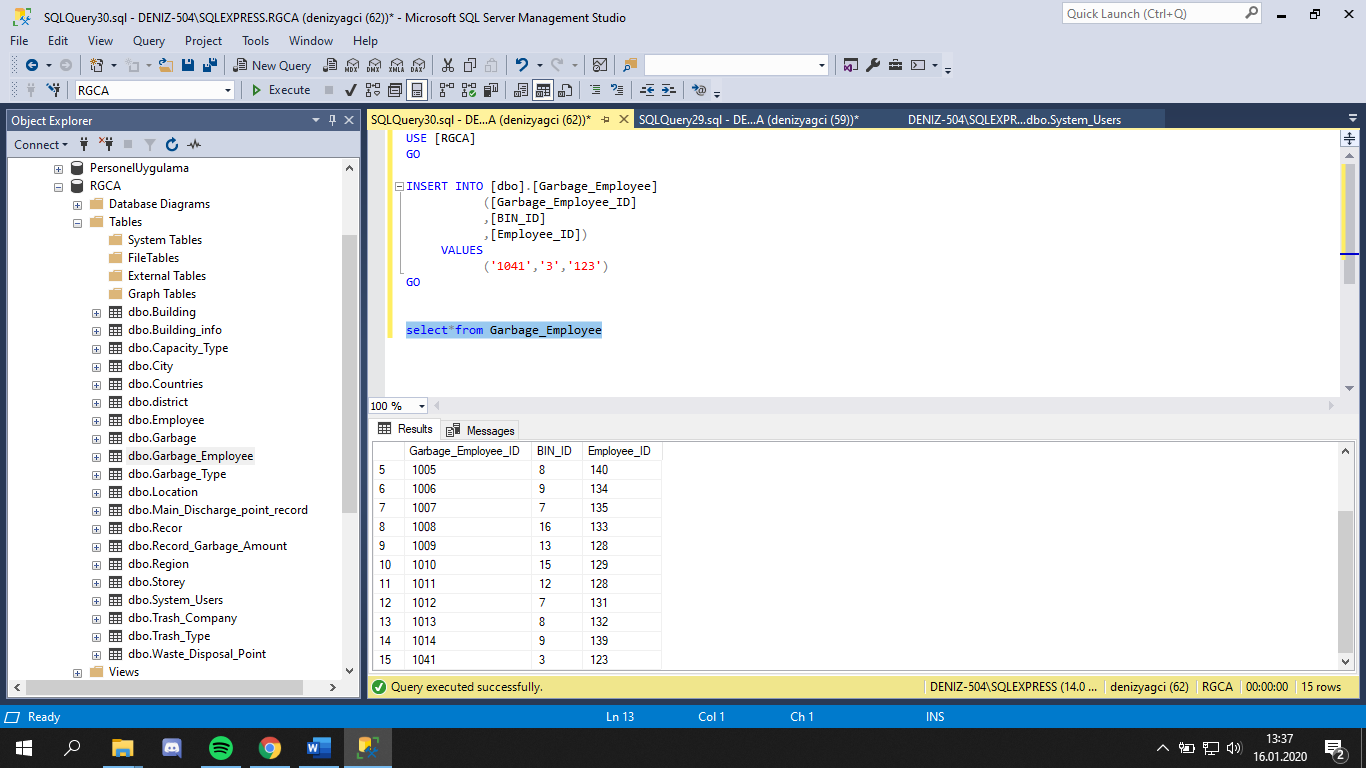
,[Employee\_ID])

VALUES

('1041','3','123')

GO

**It run because, Garbage and Employee tables(Master tables) have this data BIN\_ID=’3’ and Employee\_ID=’123’**



## **2)TEST 2**

UPDATE [dbo].[Building]

SET [Building\_ID] = '23'

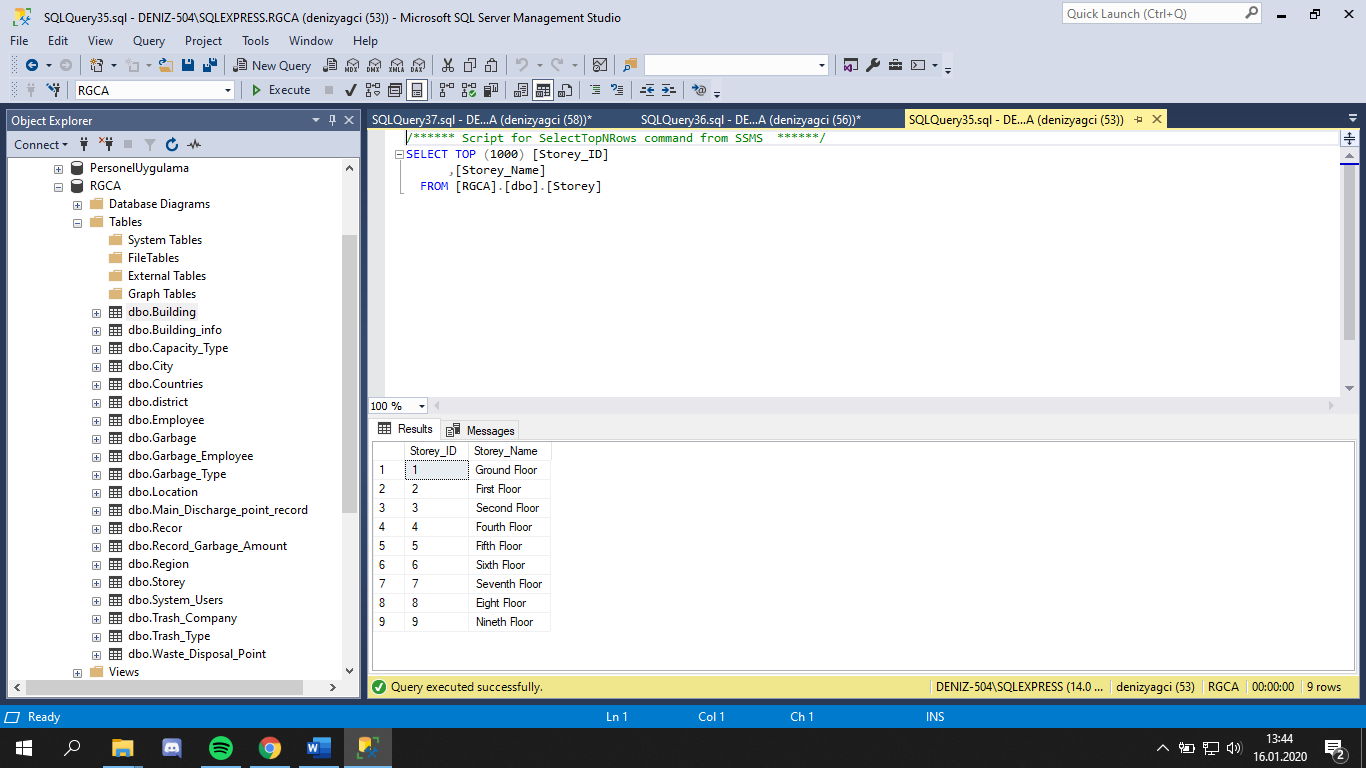
,[Building\_Name] = 'xxxxxx'

,[Storey\_ID] = '10'

WHERE Building='10'

GO

**It doesn’t run because Storey table(Child table) don’t contain data is storey\_ıd=’10’**



**3)TEST 3**

DELETE FROM [dbo].[Storey]

WHERE Storey\_Name='First Floor'

GO

**It doesn’t run because this data is used by Building table(master table).**



## **4)TEST 4**

select BIN\_ID,Garbage\_Type\_Name,Garbage\_Type\_Definition,

Trash\_Type.Trash\_Type\_ID,Waste\_Disposal\_Point.Waste\_DP\_ID,Location.Location\_ID from Garbage

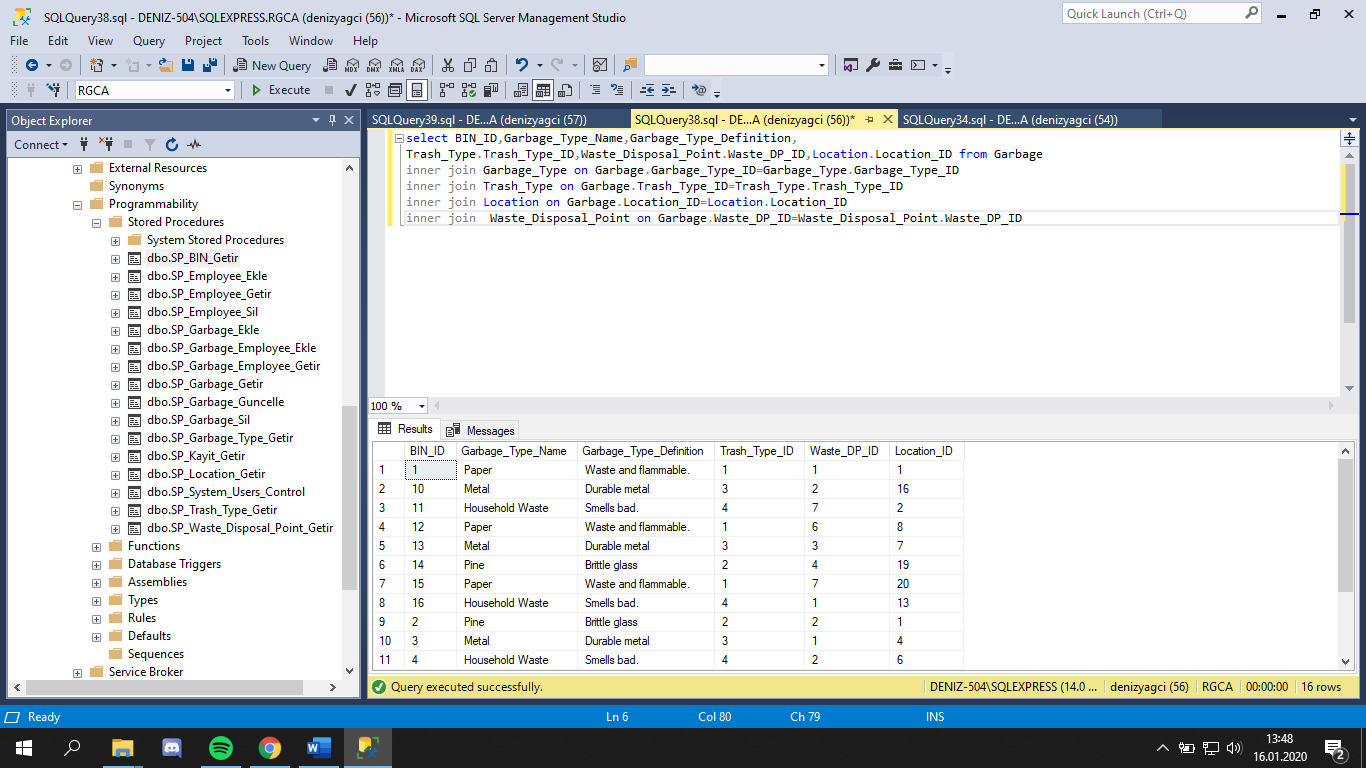
inner join Garbage\_Type on Garbage.Garbage\_Type\_ID=Garbage\_Type.Garbage\_Type\_ID

inner join Trash\_Type on Garbage.Trash\_Type\_ID=Trash\_Type.Trash\_Type\_ID

inner join Location on Garbage.Location\_ID=Location.Location\_ID

inner join Waste\_Disposal\_Point on Garbage.Waste\_DP\_ID=Waste\_Disposal\_Point.Waste\_DP\_ID

**It run because these tables have relations with PK and FK consraints.**



## **5)TEST 5**

INSERT INTO [dbo].[System\_Users]

([User\_ID]

,[User\_Name]

,[User\_Sifre])

VALUES

('1223456'

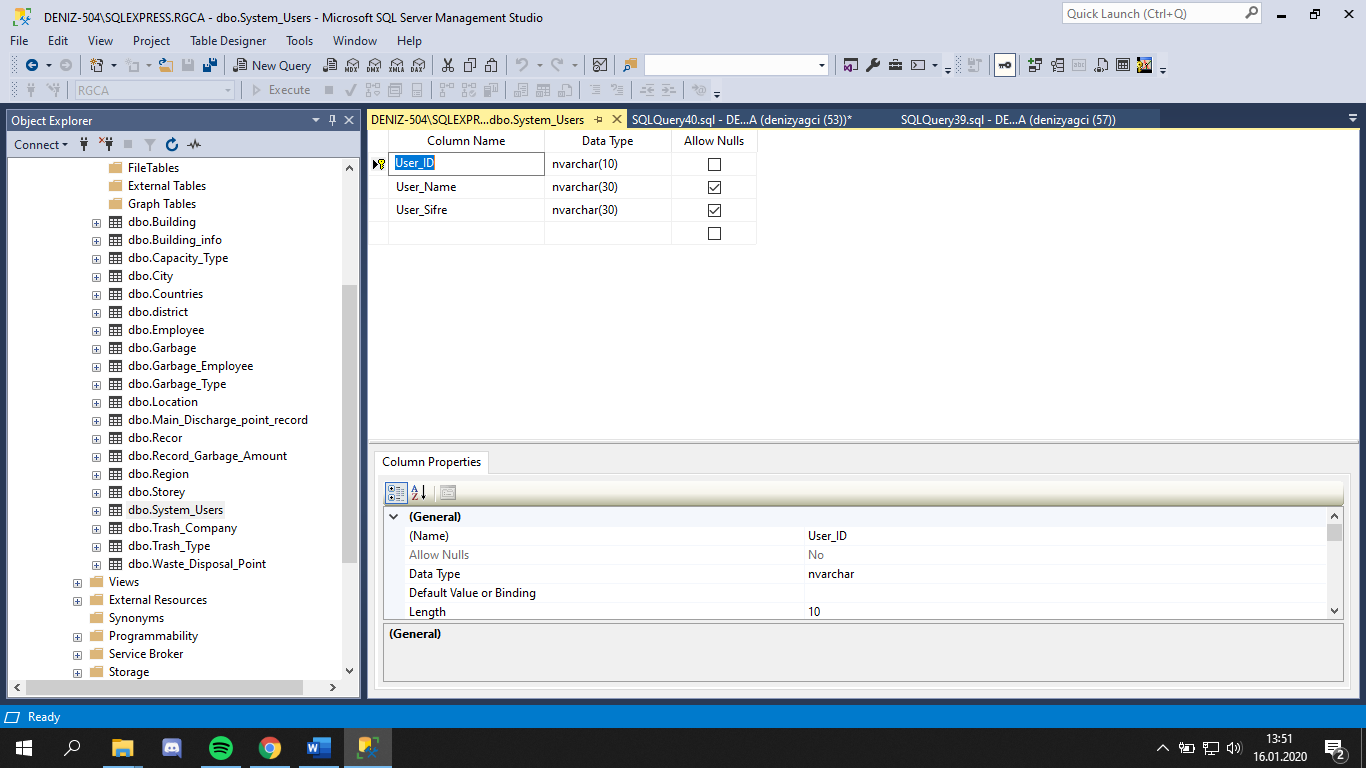
,'ddedededededededeedddddddddddedededede'

,'dededededededededededededdddddddddd')

GO

**It doesn’t run because, User([User\_ID] ,[User\_Name] ,[User\_Sifre])**

**Data type is different.**



**6)TEST 6**

INSERT INTO [dbo].[Garbage]

([BIN\_ID]

,[BIN\_Name]

,[Garbage\_Type\_ID]

,[Trash\_Type\_ID]

,[Location\_ID]

,[Waste\_DP\_ID])

VALUES

('1'

,'BIN paper'

,'1'

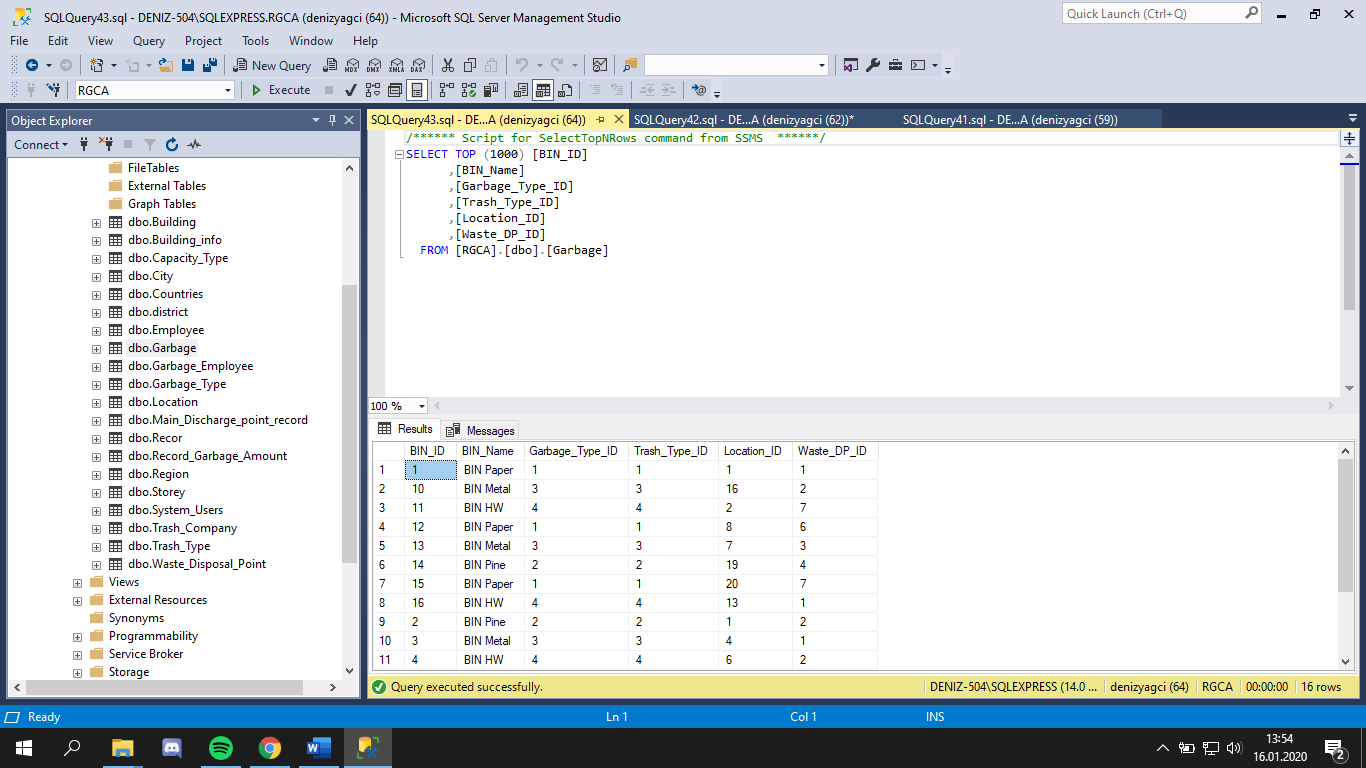
,'2'

,'4'

,'5')

GO

**It doesn’t run because, BIN\_ID=’1’ data already been in Garbage table.**



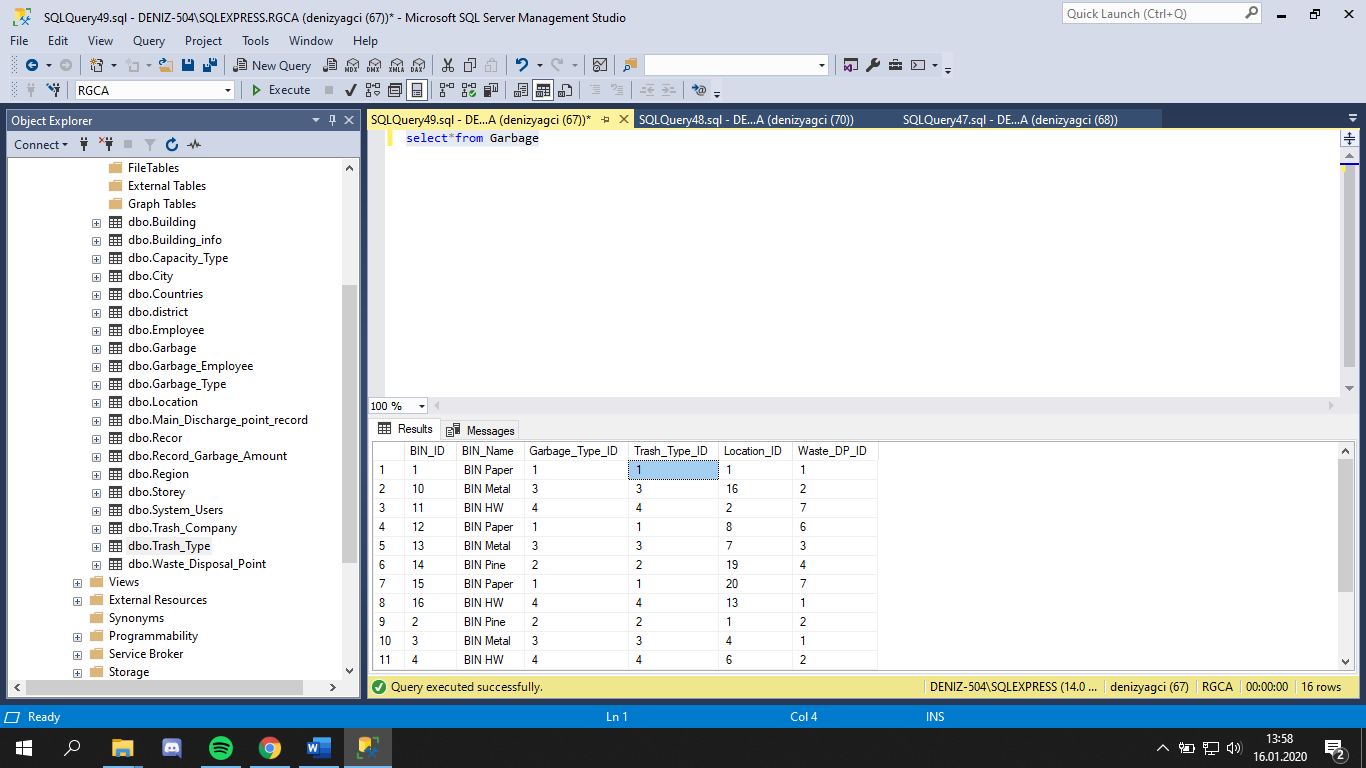
## **7)TEST 7**

DELETE FROM [dbo].[Trash\_Type]

WHERE Trash\_Type\_ID='1'

GO

**It doesn’t run because this data is used by Garbage table(master table).**

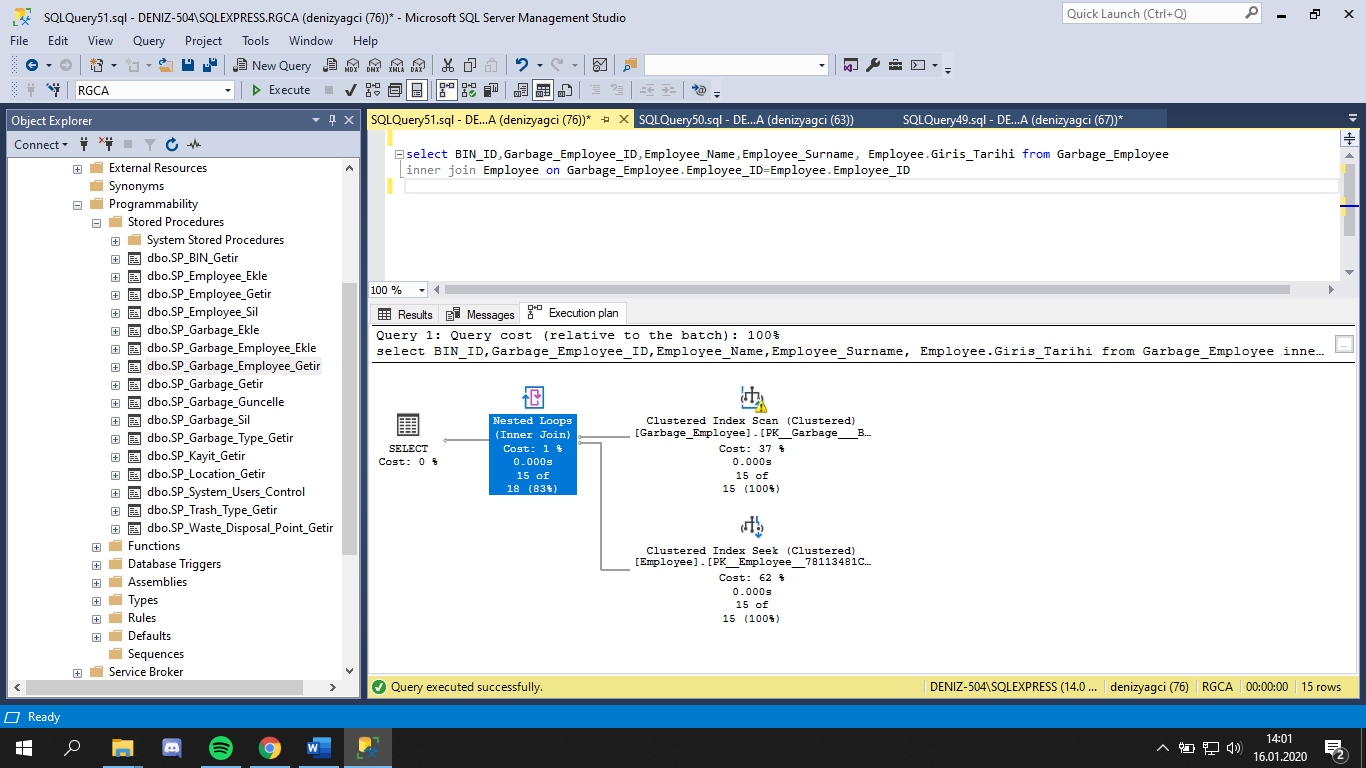


## **8)TEST 8**

select BIN\_ID,Garbage\_Employee\_ID,Employee\_Name,Employee\_Surname, Employee.Giris\_Tarihi from Garbage\_Employee

inner join Employee on Garbage\_Employee.Employee\_ID=Employee.Employee\_ID

**It run because, Employee table(master) and Garbage\_Employee table(child) have a relation with Employee\_ID** .



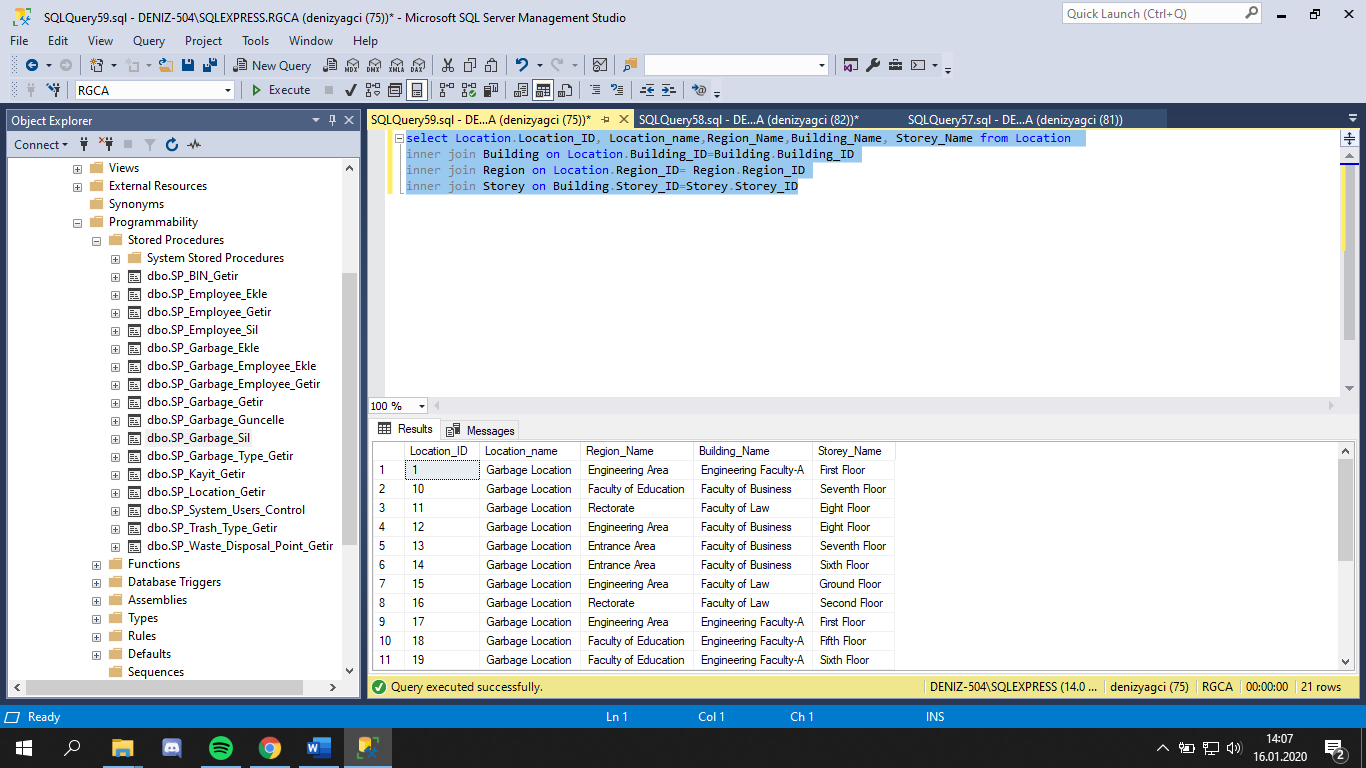
## **9)TEST 9**

select Location.Location\_ID, Location\_name,Region\_Name,Building\_Name, Storey\_Name from Location

inner join Building on Location.Building\_ID=Building.Building\_ID

inner join Region on Location.Region\_ID= Region.Region\_ID

inner join Storey on Building.Storey\_ID=Storey.Storey\_ID



# **RGC Application(Visual Studio/C#)**

N-tier architecture was used to create this application.

* DataAccessLayer
* BusinessLogicLayer
* UserInterFace

Object-oriented programming is done in a certain order and order when dealing with the database. applications; how to access the data, what applications will be made on it and how to present it to the user side are the most important problems for a programmer. The three-tier architecture helps programmers at this point. Three-tier architecture is an important programming technique that has gained importance with object-oriented programming and is used by many programmers today. It is easier and faster to manage programs that are prepared using three-tier architecture. Because in programs prepared according to three separate layers, each action is designed separately and it is clear what is where; thus, in cases where intervention is required, this process is performed easily and without wasting time.

Layered architecture consists of 3 separate layers. These are: Data Layer, Business Layer and Presentation Layer.

In the three-tier architecture, the lowest layer is the data layer (Data Layer) where the data is withdrawn from the database and the incoming data is added to the database. With the data layer, the existing data in the database is drawn to the application. In the same way, the data layer required for the user-entered data to be saved in the application environment is saved in the existing database.

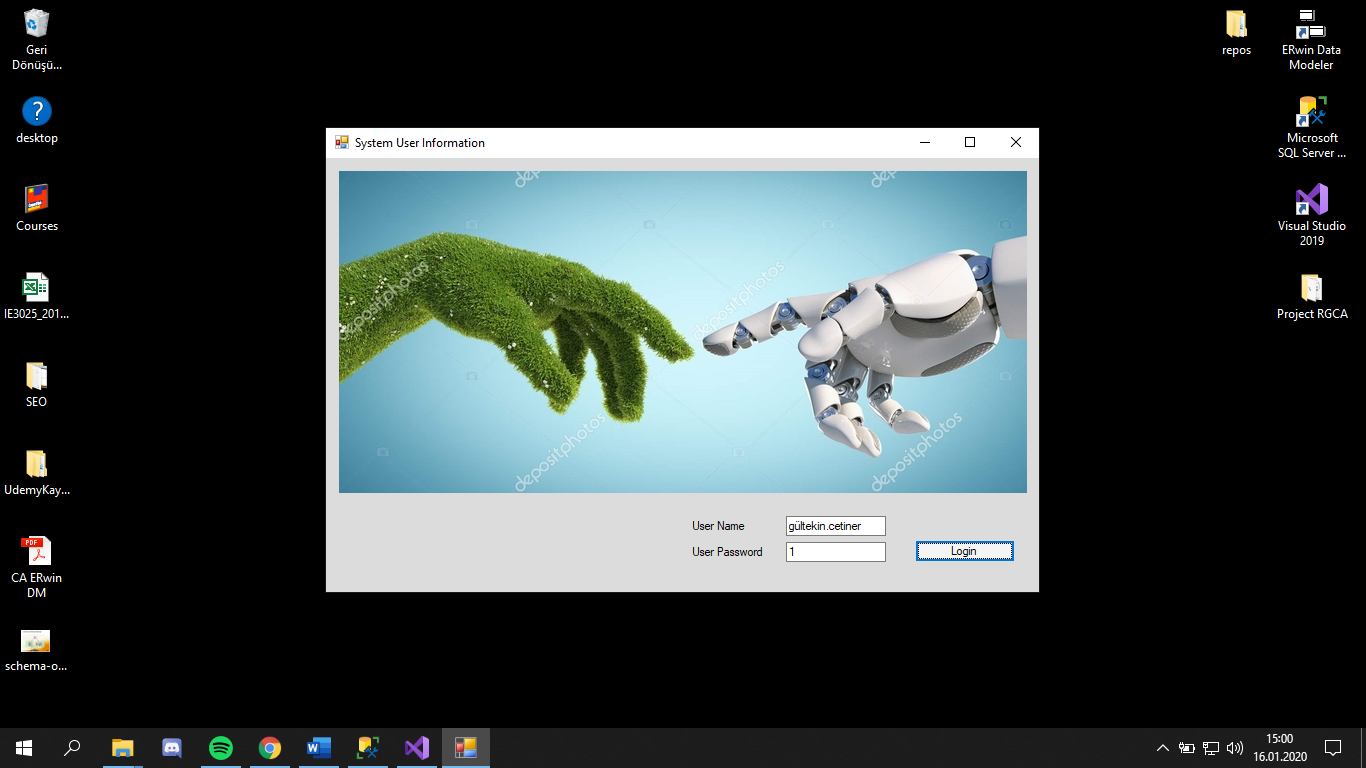
The data extracted via the data layer cannot be directly integrated into the created application. Because with the data layer, only the data is drawn to the application environment, but the operations to be performed on it are still unclear. For this purpose, adaptation of the extracted data to the application must be performed. This is provided by the Business Layer. Data adapted to the program created with the work layer is now ready.

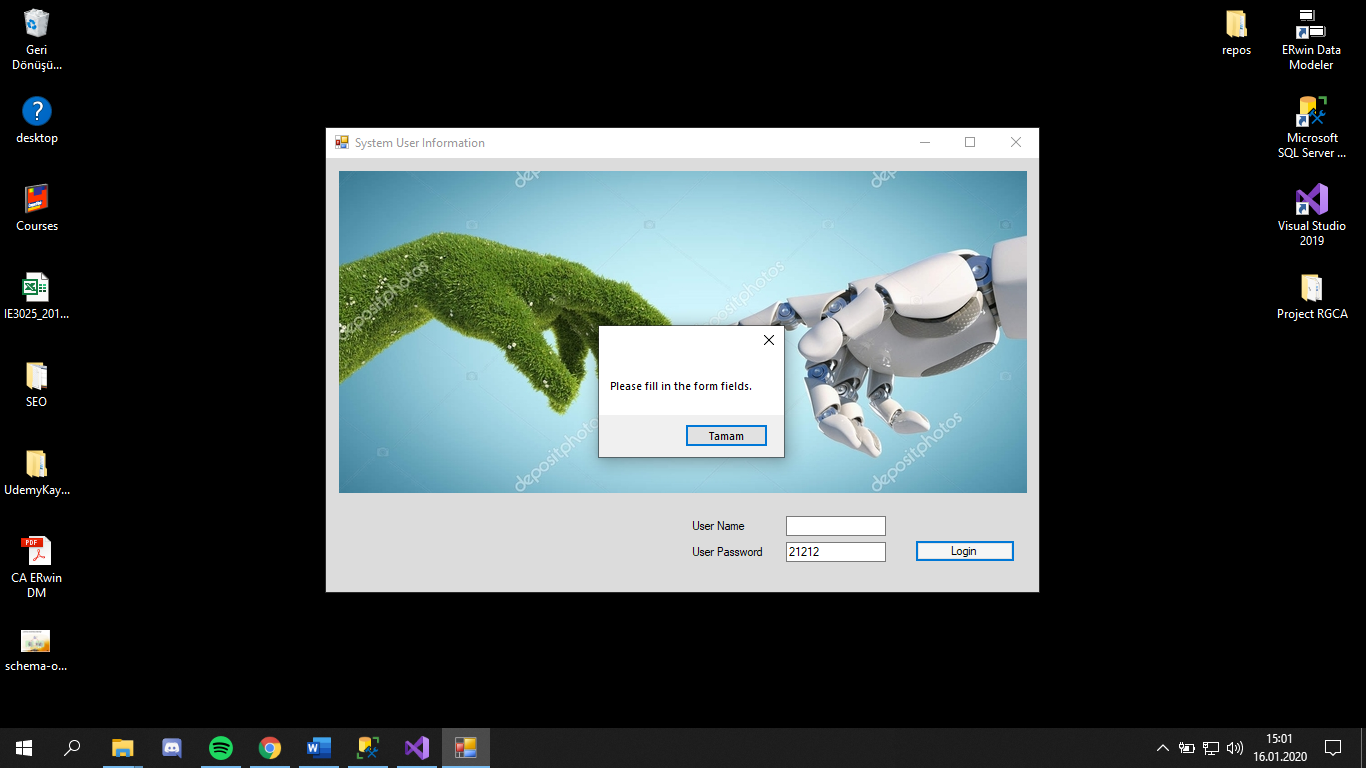
The last one is the Presentation Layer, where the user interface of the application is configured. With the presentation layer, the view of the data that we have made available in the business layer, which will now go to the user, is determined and the application we have created is now complete.

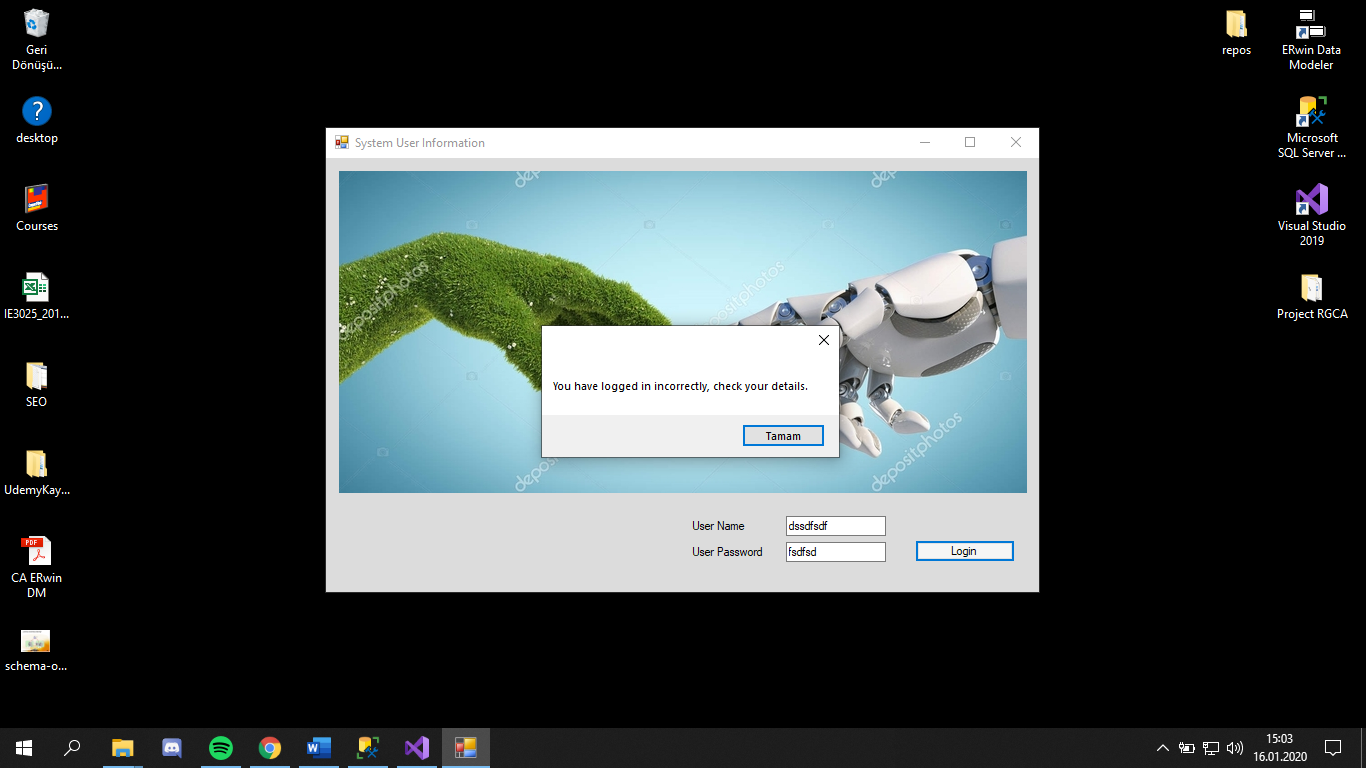
As described above, the three-tier architecture is implemented interactively. First, the data layer (Data Layer) is drawn from the database, the business layer (Business Layer) to ensure the compatibility of this data and the presentation layer (Presentation Layer) to create an interface with the user data is shown. This can be done in the opposite direction, ie the user enters the data via the user interface (presentation layer), the entered data is adapted to the database (business layer) and finally transferred to the database (data layer). As can be seen, the three-tier architecture is implemented interactively. Below is an application for a clear understanding of this structure.

## **A)Photos of App**

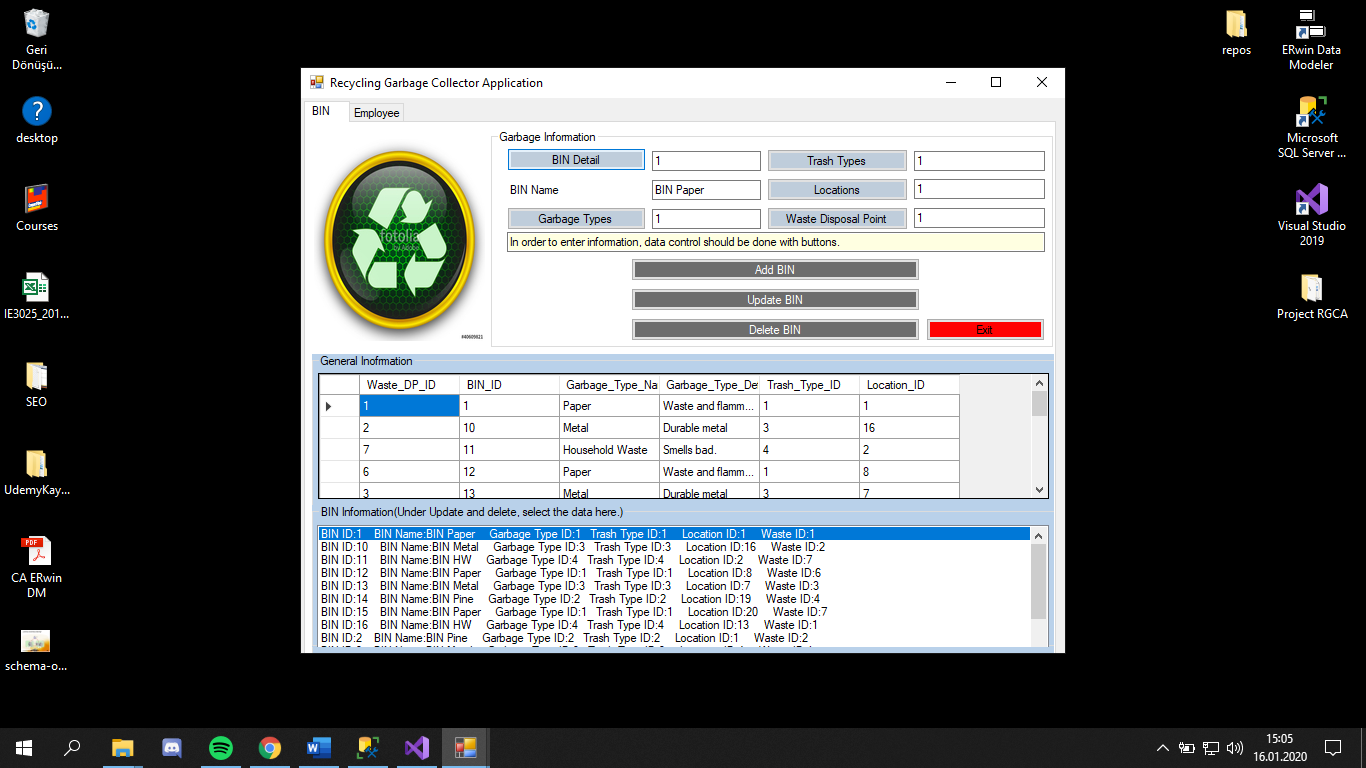
* The system user is the login page.

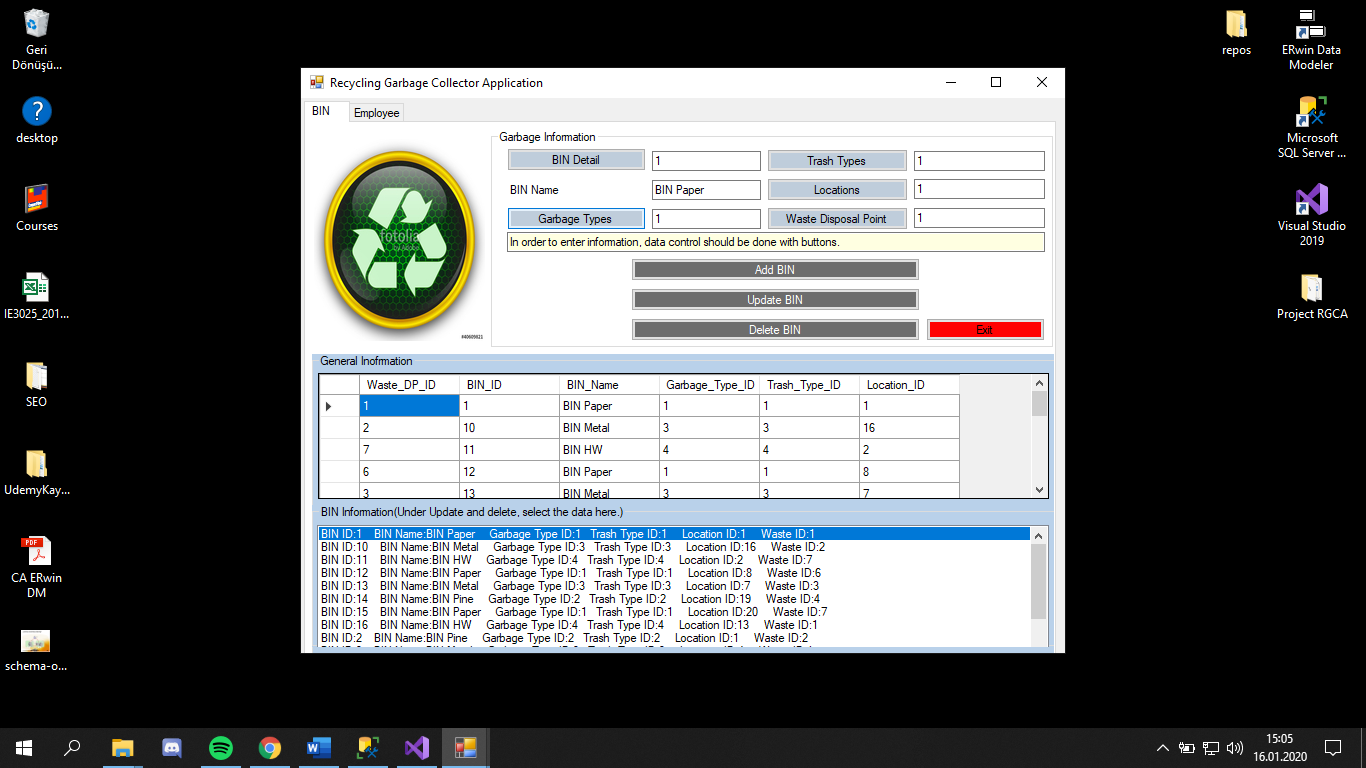
You can login using your username and password. 

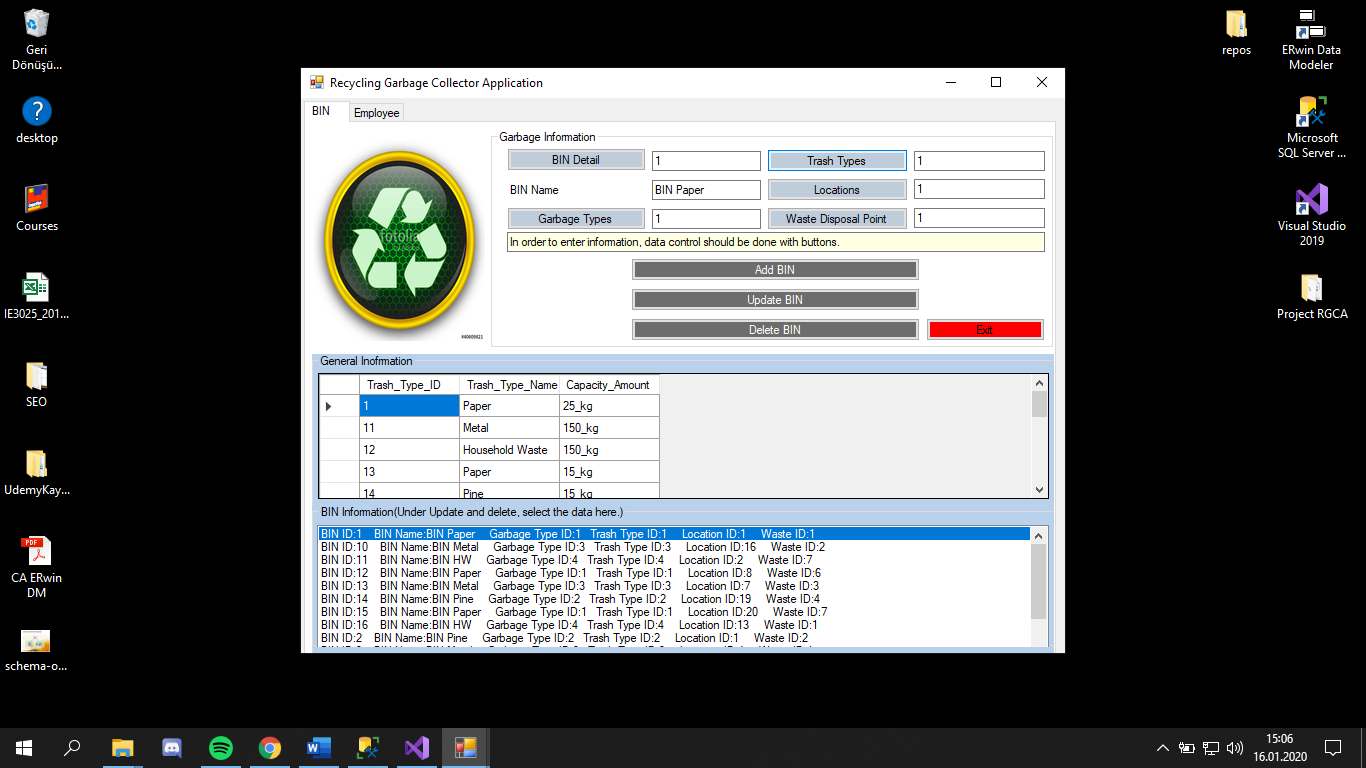
* It will give warning when missing information is entered.
* It will give warning when wrong information is entered.

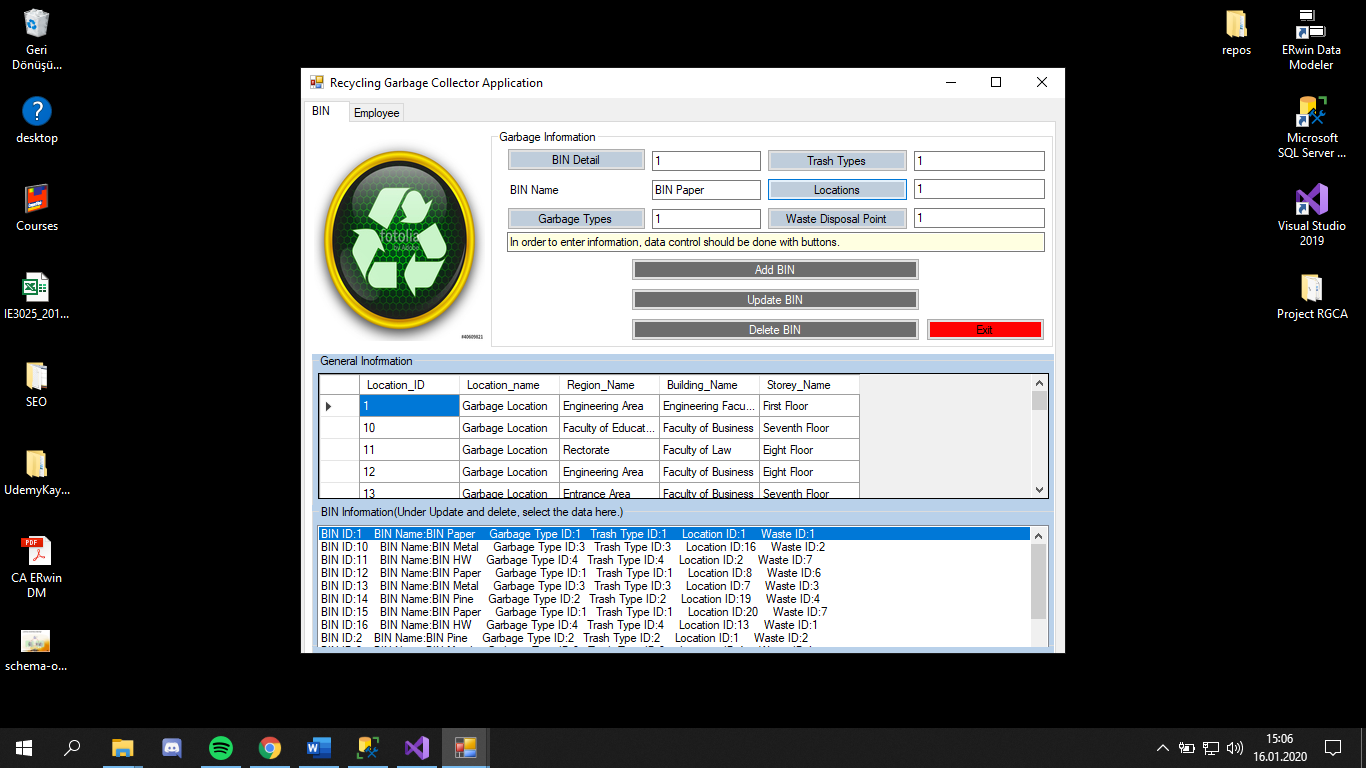


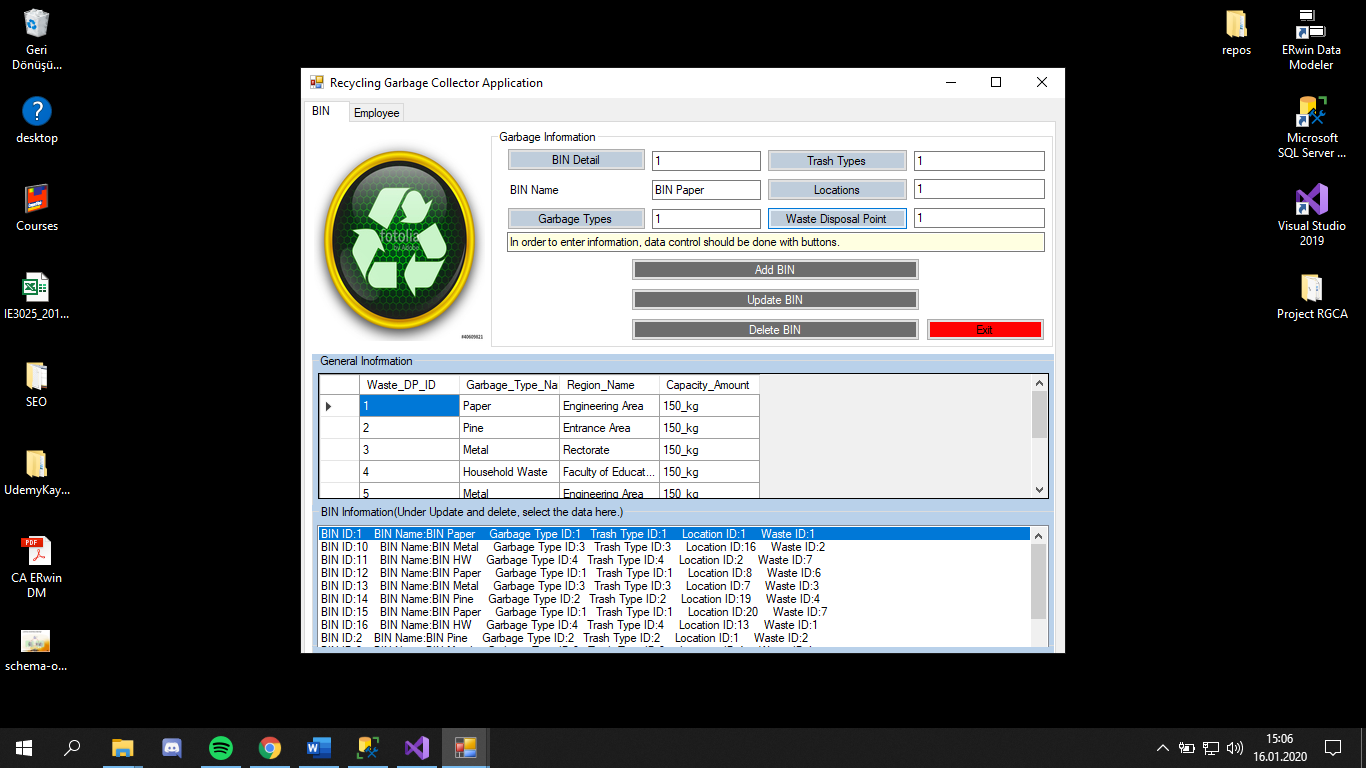
* By clicking on the buttons, the data of the relevant button will appear at the bottom. According to the information 'add' button you can add records.



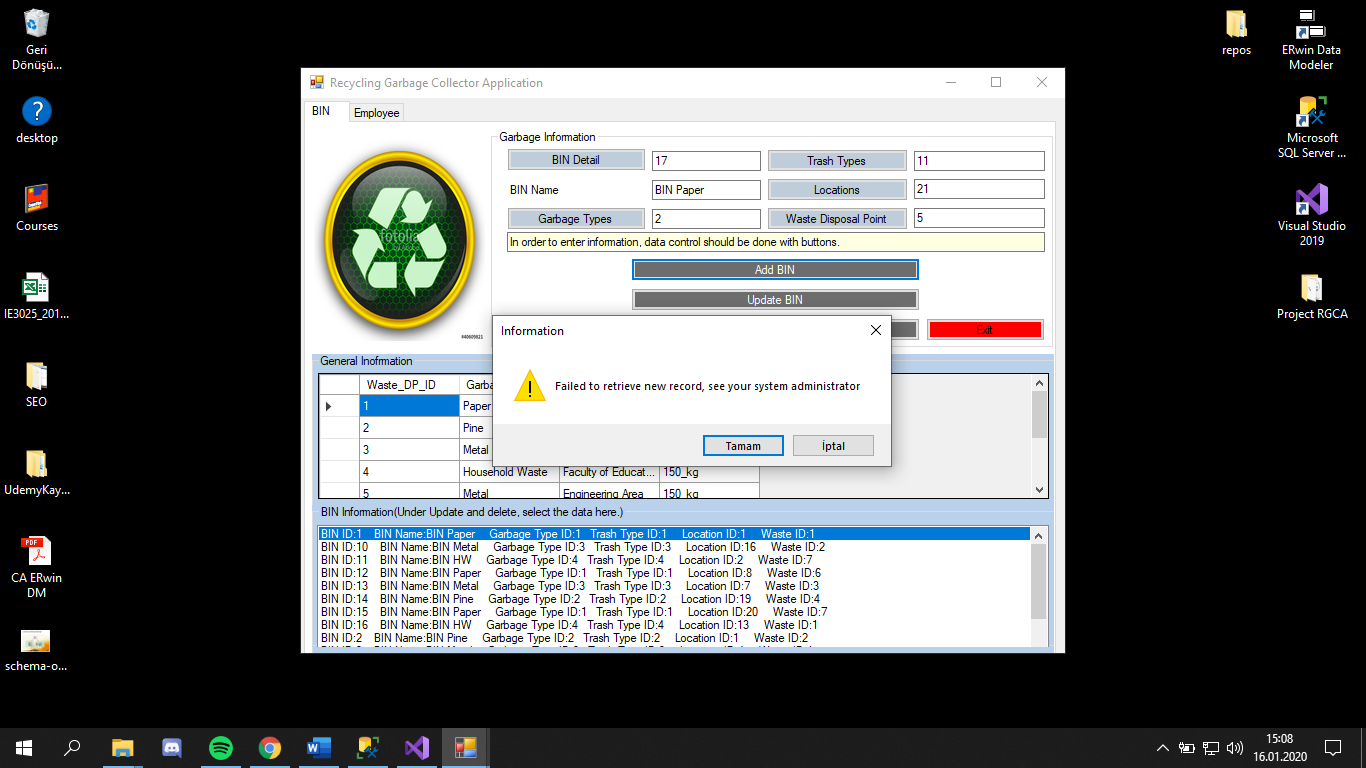




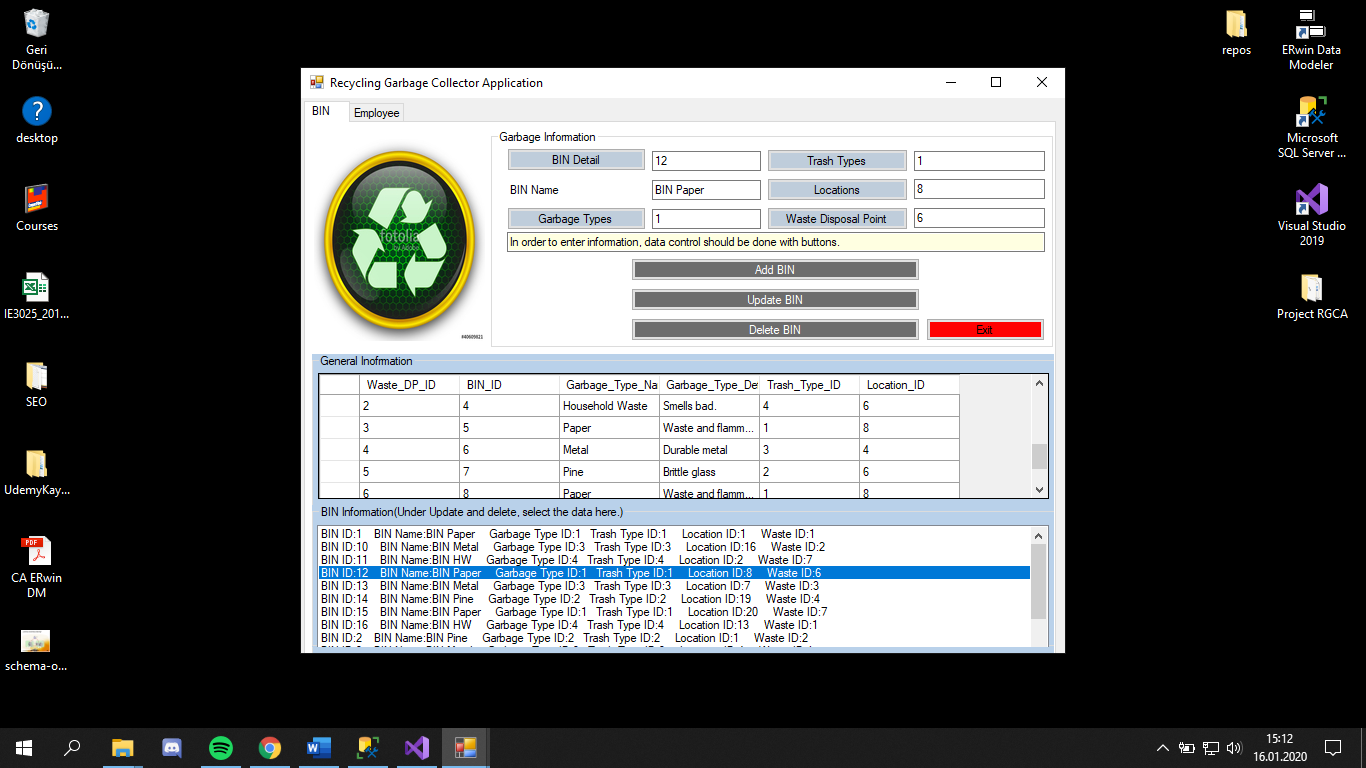


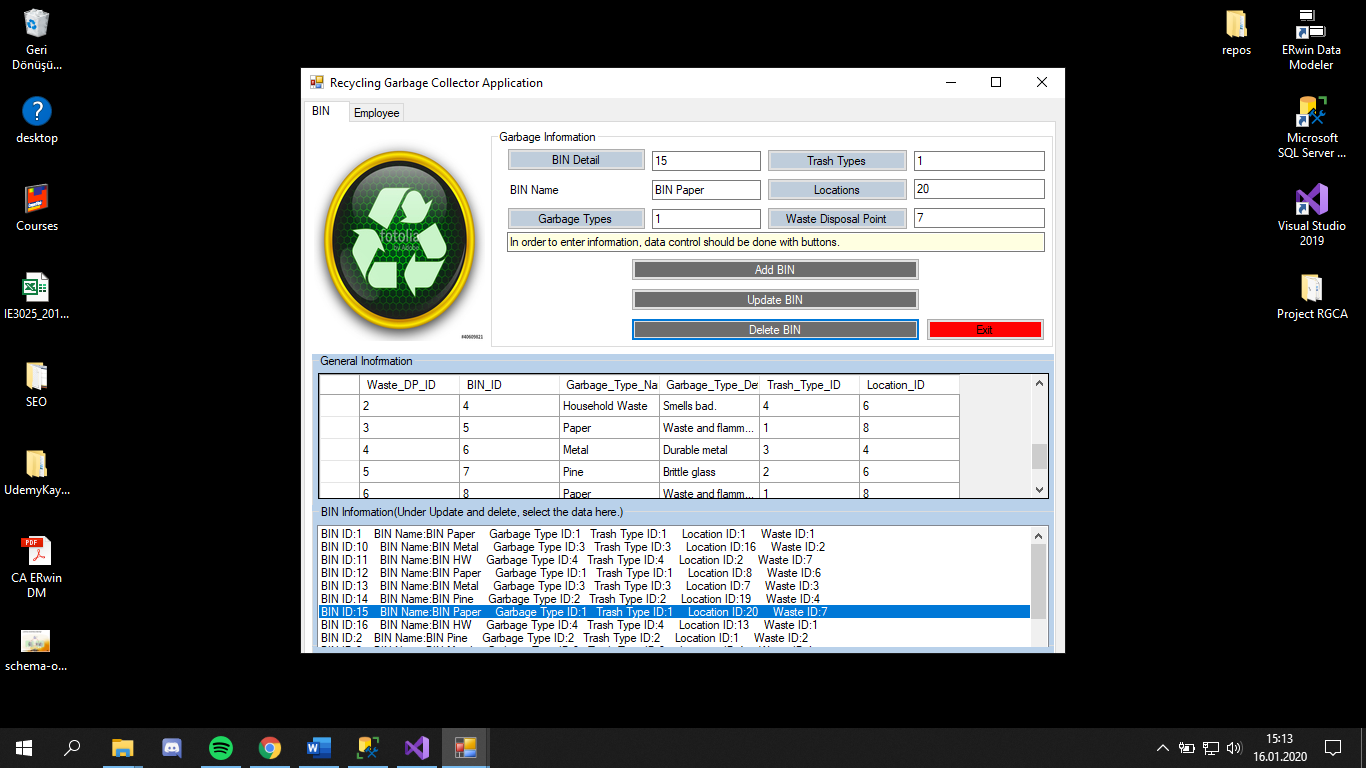


* Clicking the 'add' button will ask you if you are sure to register.

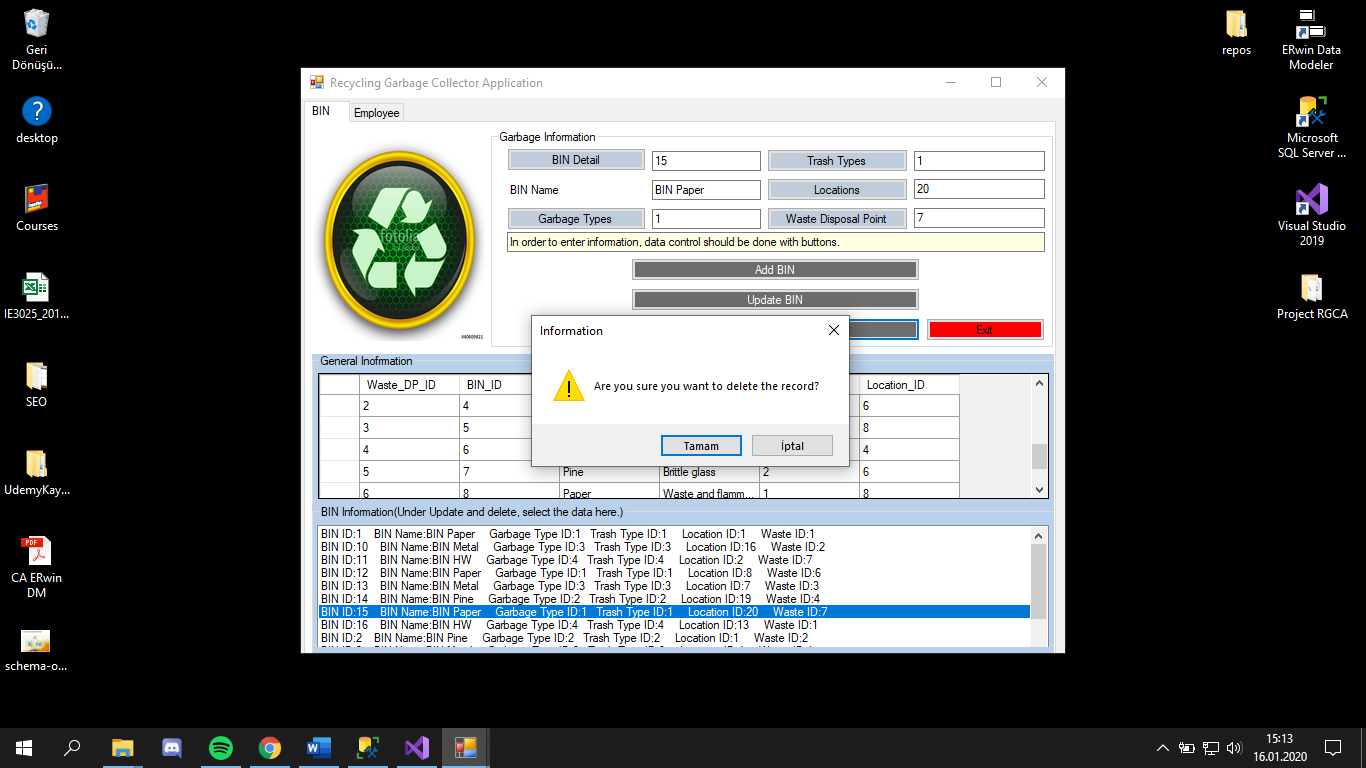


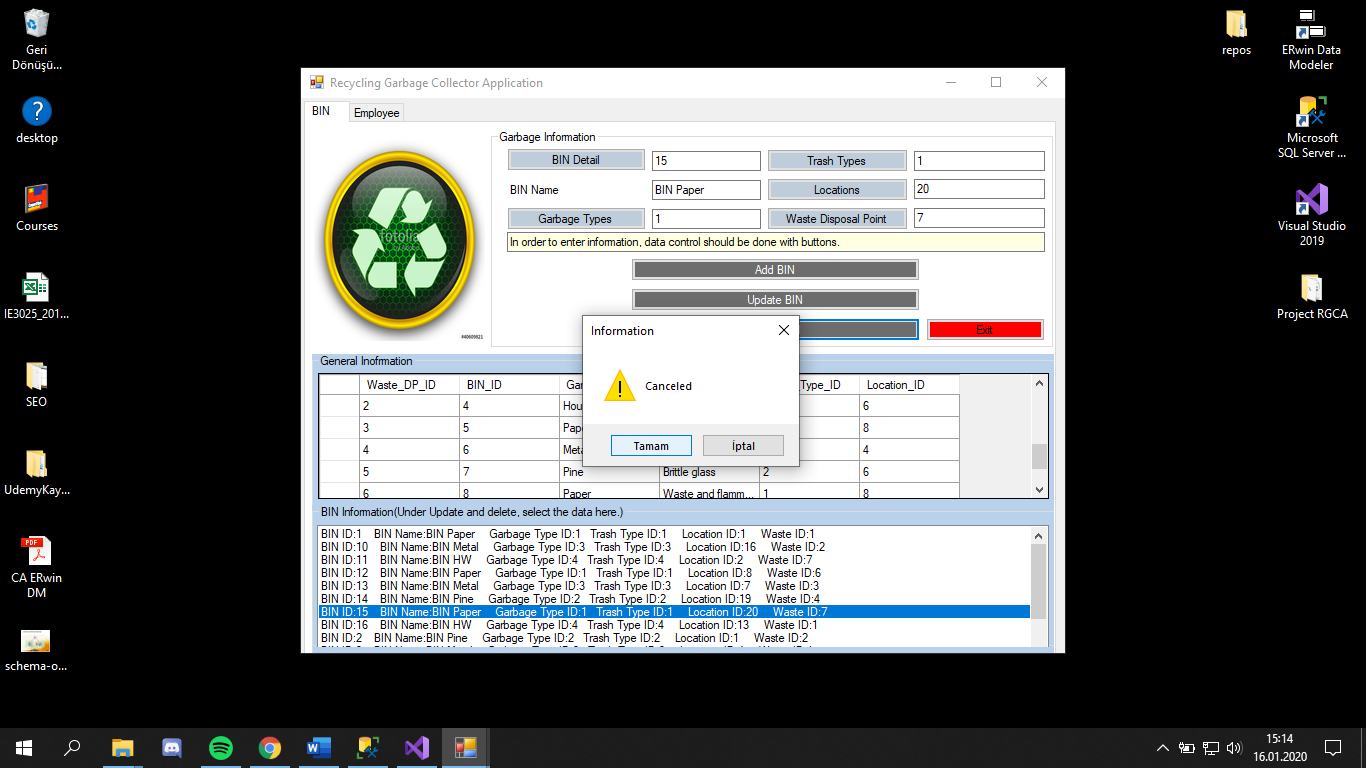
* By selecting the data from the 'text\_file' section, the boxes match the data. As a result of pairing, 'delete' and 'update' button is done with the necessary actions.

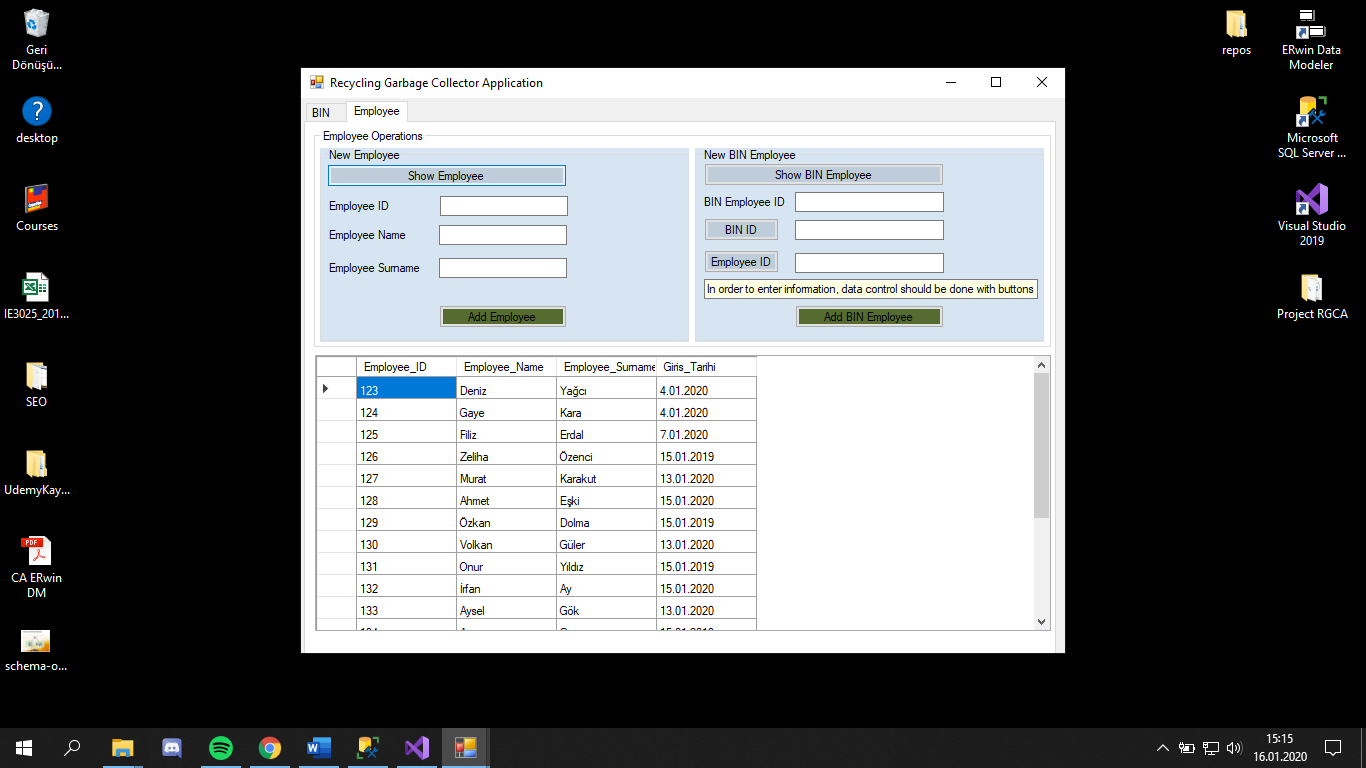


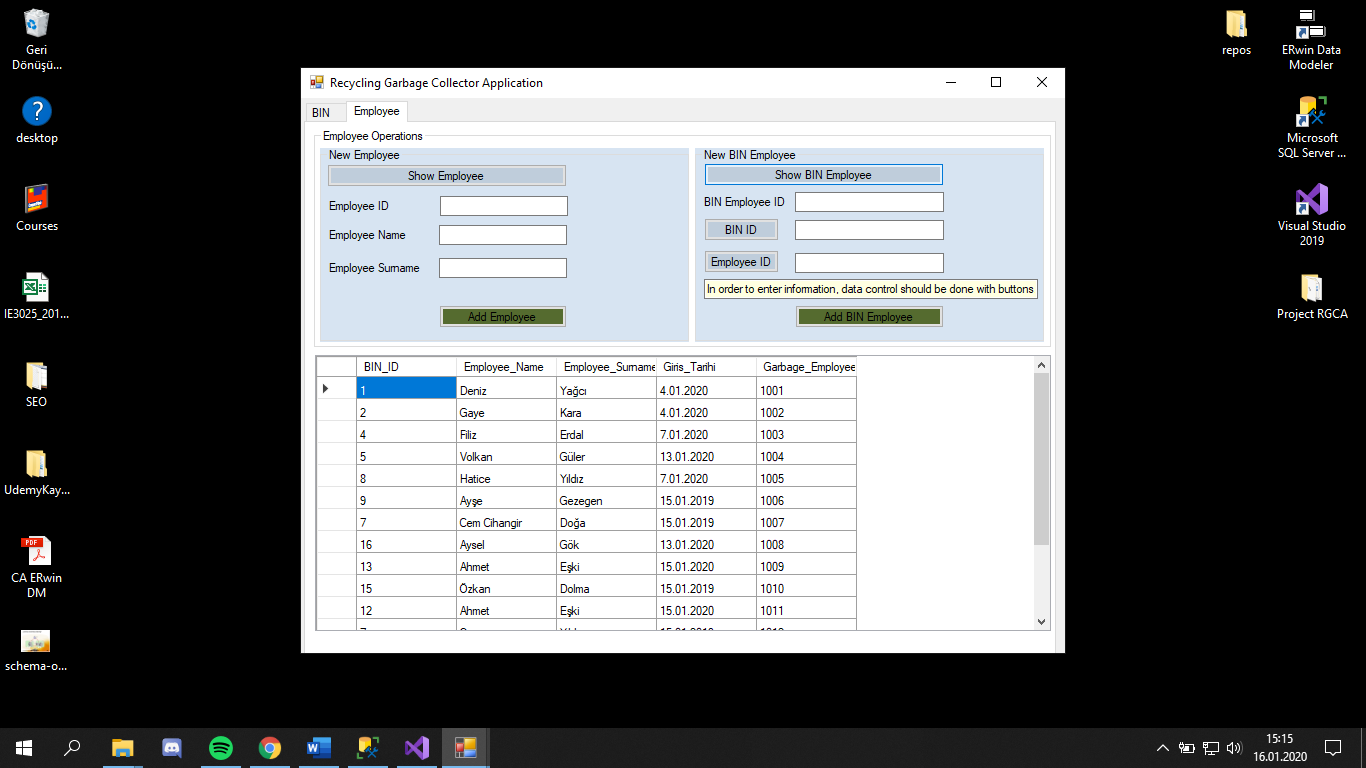


* Deletion can be dispensed with.

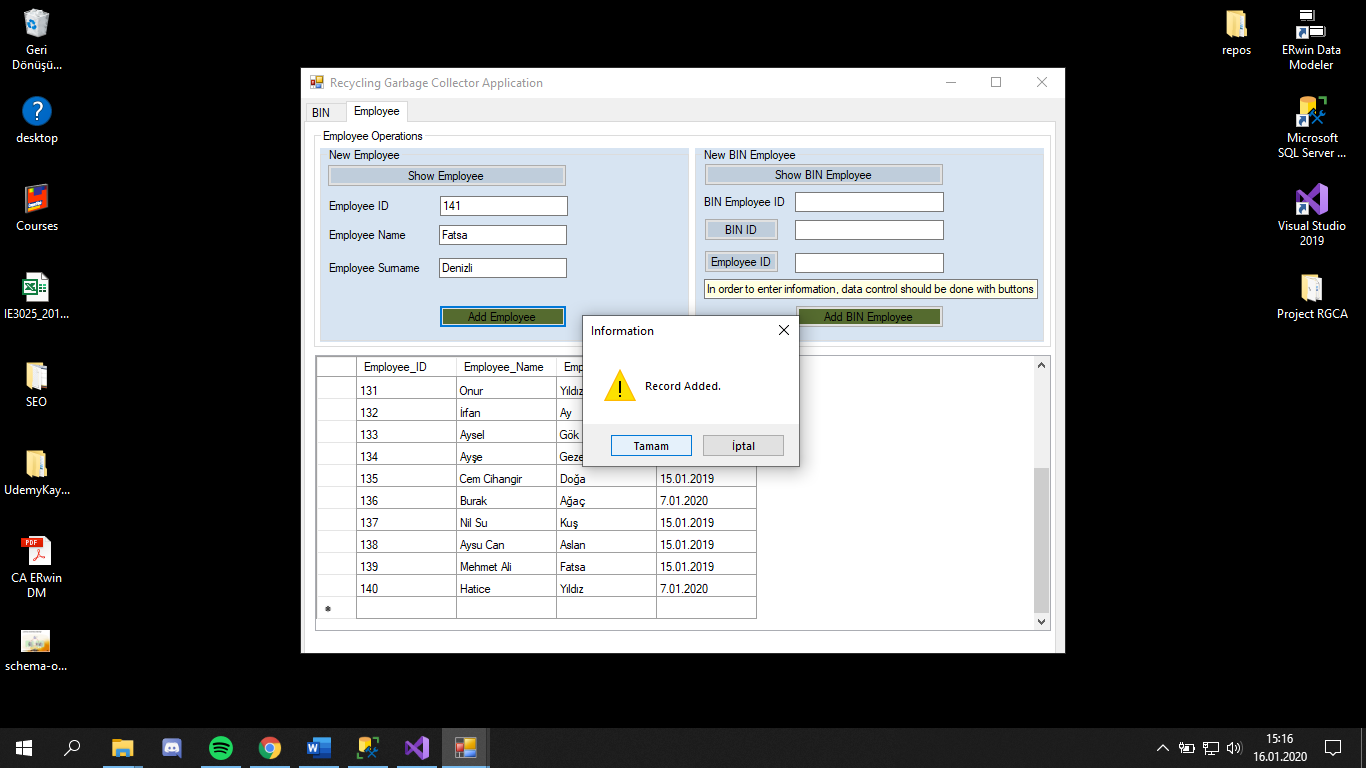


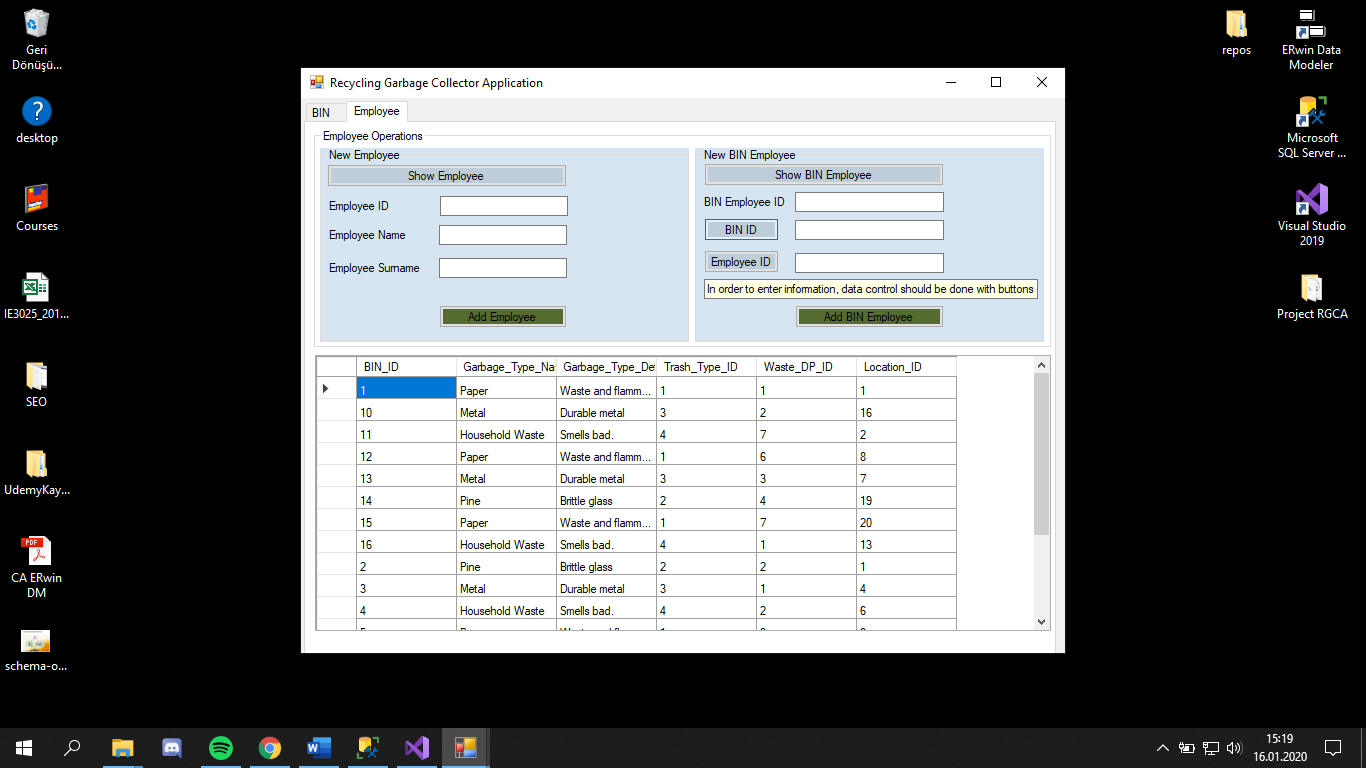


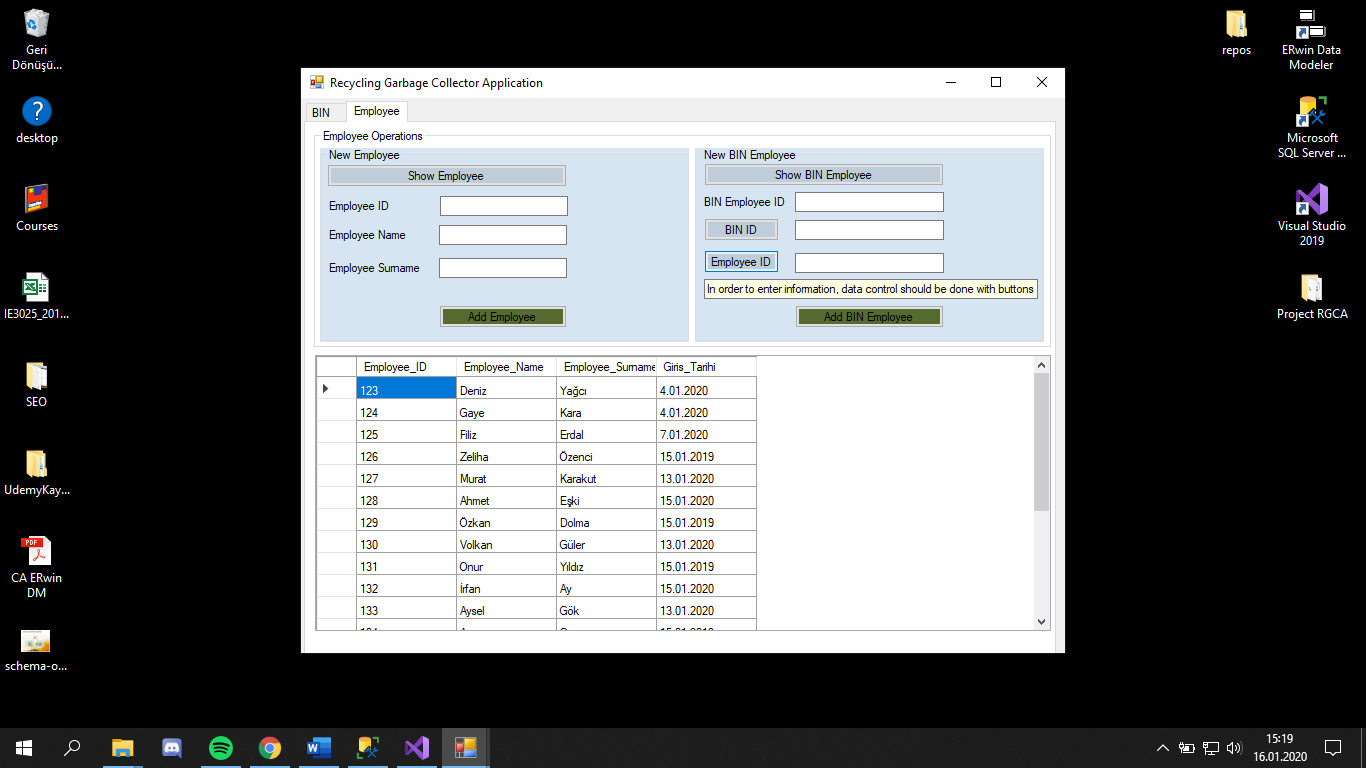
* In the 'Employee' section, general workers can be checked and added.

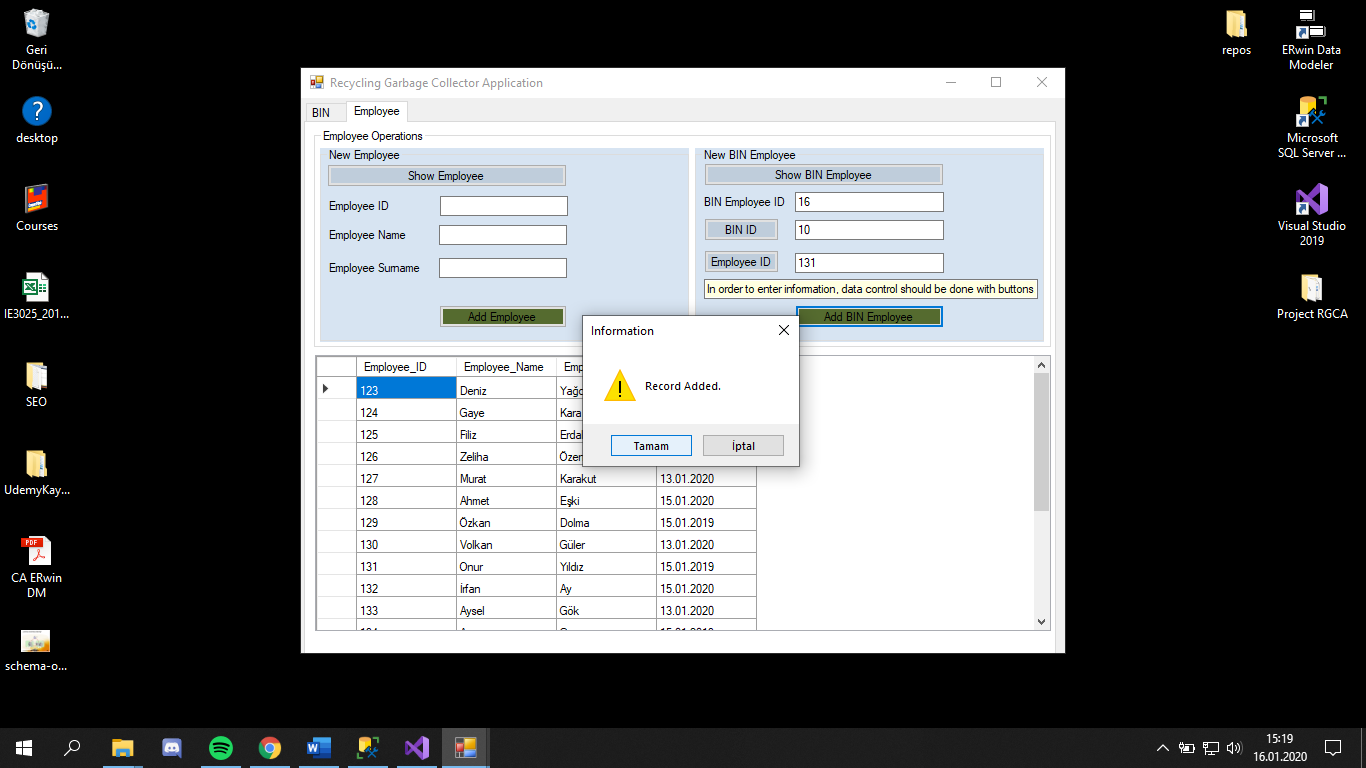


* After the required data is checked and entered, add can be done with 'add' buttons.









## **B)Codes of App**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data.SqlClient;

using System.Data;

using System.Configuration;

namespace RGCA

{

internal class DataAccessLayer

{

SqlConnection connect;

SqlCommand command;

SqlDataReader reader;

int sonuc = 0;

public DataAccessLayer()

{

connect = new SqlConnection();

connect.ConnectionString = ConfigurationManager.ConnectionStrings["SQL"].ConnectionString;

}

internal void BaglantiAyarla()

{

if (connect.State == System.Data.ConnectionState.Closed)

{

connect.Open();

}

else

{

connect.Close();

}

}

internal int SystemEntryControl(SystemUsers S)

{

try

{

command = new SqlCommand("SP\_System\_Users\_Control", connect);

command.CommandType = CommandType.StoredProcedure;

command.Parameters.Add("@User\_Name", SqlDbType.NVarChar).Value = S.User\_Name;

command.Parameters.Add("@User\_Sifre", SqlDbType.NVarChar).Value = S.User\_Sifre;

BaglantiAyarla();

sonuc = (int)command.ExecuteScalar();

}

catch (Exception ex)

{

}

finally

{

BaglantiAyarla();

}

return sonuc;

}

internal SqlDataReader Garbage\_Type\_Getir()

{

command = new SqlCommand("SP\_Garbage\_Type\_Getir", connect);

command.CommandType = CommandType.StoredProcedure;

BaglantiAyarla();

return command.ExecuteReader();

}

internal SqlDataReader Garbage\_Getir()

{

command = new SqlCommand("SP\_Garbage\_Getir", connect);

command.CommandType = CommandType.StoredProcedure;

BaglantiAyarla();

return command.ExecuteReader();

}

internal SqlDataReader Trash\_Type\_Getir()

{

command = new SqlCommand("SP\_Trash\_Type\_Getir", connect);

command.CommandType = CommandType.StoredProcedure;

BaglantiAyarla();

return command.ExecuteReader();

}

internal SqlDataReader Locations\_Getir()

{

command = new SqlCommand("SP\_Location\_Getir", connect);

command.CommandType = CommandType.StoredProcedure;

BaglantiAyarla();

return command.ExecuteReader();

}

internal SqlDataReader Waste\_Disposal\_Point\_Getir()

{

command = new SqlCommand("SP\_Waste\_Disposal\_Point\_Getir", connect);

command.CommandType = CommandType.StoredProcedure;

BaglantiAyarla();

return command.ExecuteReader();

}

internal SqlDataReader BIN\_Getir()

{

command = new SqlCommand("SP\_BIN\_Getir", connect);

command.CommandType = CommandType.StoredProcedure;

BaglantiAyarla();

return command.ExecuteReader();

}

internal int New\_Bin(Garbage G)

{

try

{

command = new SqlCommand("SP\_Garbage\_Ekle", connect);

command.CommandType = CommandType.StoredProcedure;

command.Parameters.Add("@BIN\_ID", SqlDbType.NVarChar).Value = G.BIN\_ID;

command.Parameters.Add("@BIN\_Name", SqlDbType.NVarChar).Value = G.BIN\_Name;

command.Parameters.Add("@Garbage\_Type\_ID", SqlDbType.NVarChar).Value = G.Garbage\_Type\_ID;

command.Parameters.Add("@Trash\_Type\_ID", SqlDbType.NVarChar).Value = G.Trash\_Type\_ID;

command.Parameters.Add("@Location\_ID", SqlDbType.NVarChar).Value = G.Location\_ID;

command.Parameters.Add("@Waste\_DP\_ID", SqlDbType.NVarChar).Value = G.Waste\_DP\_ID;

BaglantiAyarla();

sonuc = (int)command.ExecuteNonQuery();

}

catch (Exception ex)

{

}

finally

{

BaglantiAyarla();

}

return sonuc;

}

internal SqlDataReader Garbage\_Employee\_Getir()

{

command = new SqlCommand("SP\_Garbage\_Employee\_Getir", connect);

command.CommandType = CommandType.StoredProcedure;

BaglantiAyarla();

return command.ExecuteReader();

}

internal SqlDataReader Employee\_Getir()

{

command = new SqlCommand("SP\_Employee\_Getir", connect);

command.CommandType = CommandType.StoredProcedure;

BaglantiAyarla();

return command.ExecuteReader();

}

internal int New\_Employee(Employee E)

{

try

{

command = new SqlCommand("SP\_Employee\_Ekle", connect);

command.CommandType = CommandType.StoredProcedure;

command.Parameters.Add("@Employee\_ID", SqlDbType.NVarChar).Value = E.Employee\_ID;

command.Parameters.Add("@Employee\_Name", SqlDbType.NVarChar).Value = E.Employee\_Name;

command.Parameters.Add("@Employee\_Surname", SqlDbType.NVarChar).Value = E.Employee\_Surname;

command.Parameters.Add("@Giris\_Tarihi", SqlDbType.Date).Value = E.Giris\_Tarihi;

BaglantiAyarla();

sonuc = (int)command.ExecuteNonQuery();

}

catch (Exception ex)

{

}

finally

{

BaglantiAyarla();

}

return sonuc;

}

internal int New\_Bin\_Employee(BINEmployee BE)

{

try

{

command = new SqlCommand("SP\_Garbage\_Employee\_Ekle", connect);

command.CommandType = CommandType.StoredProcedure;

command.Parameters.Add("@Garbage\_Employee\_ID", SqlDbType.NVarChar).Value = BE.Garbage\_Employee\_ID;

command.Parameters.Add("@BIN\_ID", SqlDbType.NVarChar).Value = BE.BIN\_ID;

command.Parameters.Add("@Employee\_ID", SqlDbType.NVarChar).Value = BE.Employee\_ID;

BaglantiAyarla();

sonuc = (int)command.ExecuteNonQuery();

}

catch (Exception ex)

{

}

finally

{

BaglantiAyarla();

}

return sonuc;

}

internal int Kayit\_Guncelle(Garbage G)

{

try

{

command = new SqlCommand("SP\_Garbage\_Guncelle", connect);

command.CommandType = CommandType.StoredProcedure;

command.Parameters.Add("@BIN\_ID", SqlDbType.NVarChar).Value = G.BIN\_ID;

command.Parameters.Add("@BIN\_Name", SqlDbType.NVarChar).Value = G.BIN\_Name;

command.Parameters.Add("@Garbage\_Type\_ID", SqlDbType.NVarChar).Value = G.Garbage\_Type\_ID;

command.Parameters.Add("@Trash\_Type\_ID", SqlDbType.NVarChar).Value = G.Trash\_Type\_ID;

command.Parameters.Add("@Location\_ID", SqlDbType.NVarChar).Value = G.Location\_ID;

command.Parameters.Add("@Waste\_DP\_ID", SqlDbType.NVarChar).Value = G.Waste\_DP\_ID;

BaglantiAyarla();

sonuc = (int)command.ExecuteNonQuery();

}

catch (Exception ex)

{

}

finally

{

BaglantiAyarla();

}

return sonuc;

}

internal SqlDataReader Kayit\_Getir(string ID)

{

command = new SqlCommand("SP\_Kayit\_Getir", connect);

command.CommandType = CommandType.StoredProcedure;

command.Parameters.Add("@BIN\_ID", SqlDbType.NVarChar).Value = ID;

BaglantiAyarla();

return command.ExecuteReader();

}

internal int Delete\_BIN(string ID)

{

try

{

command = new SqlCommand("SP\_Garbage\_Sil", connect);

command.CommandType = CommandType.StoredProcedure;

command.Parameters.Add("@BIN\_ID", SqlDbType.NVarChar).Value = ID;

BaglantiAyarla();

sonuc = command.ExecuteNonQuery();

}

catch

{

}

finally

{

BaglantiAyarla();

}

return sonuc;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data;

using System.Data.SqlClient;

namespace RGCA

{

internal class BusinessLogicLayer

{

DataAccessLayer DAL;

SqlDataReader reader;

public BusinessLogicLayer()

{

DAL = new DataAccessLayer();

}

internal int SystemUserSystemEntryControl(string user\_name, string user\_sifre)

{

if (!string.IsNullOrEmpty(user\_name) && !string.IsNullOrEmpty(user\_sifre))

{

SystemUsers S = new SystemUsers();

S.User\_Name = user\_name;

S.User\_Sifre = user\_sifre;

return DAL.SystemEntryControl(S);

}

else

return -100;

}

internal List<GarbageType> Garbage\_Type\_Getir()

{

List<GarbageType> garbageType = new List<GarbageType>();

try

{

reader = DAL.Garbage\_Type\_Getir();

while (reader.Read())

{

garbageType.Add(new GarbageType()

{

Garbage\_Type\_ID = reader.IsDBNull(0) ? string.Empty : reader.GetString(0),

Garbage\_Type\_Name = reader.IsDBNull(1) ? string.Empty : reader.GetString(1),

Garbage\_Type\_Definition = reader.IsDBNull(2) ? string.Empty : reader.GetString(2),

});

}

reader.Close();

}

catch (Exception ex)

{

}

finally

{

DAL.BaglantiAyarla();

}

return garbageType;

}

internal List<Garbage> Garbage\_Getir()

{

List<Garbage> garbage1 = new List<Garbage>();

try

{

reader = DAL.Garbage\_Getir();

while (reader.Read())

{

garbage1.Add(new Garbage()

{

BIN\_ID = reader.IsDBNull(0) ? string.Empty : reader.GetString(0),

BIN\_Name=reader.IsDBNull(1)? string.Empty: reader.GetString(1),

Garbage\_Type\_ID= reader.IsDBNull(2) ? string.Empty : reader.GetString(2),

Trash\_Type\_ID = reader.IsDBNull(3) ? string.Empty : reader.GetString(3),

Location\_ID = reader.IsDBNull(4) ? string.Empty : reader.GetString(4),

Waste\_DP\_ID = reader.IsDBNull(5) ? string.Empty : reader.GetString(5)

});

}

reader.Close();

}

catch (Exception ex)

{

}

finally

{

DAL.BaglantiAyarla();

}

return garbage1;

}

internal List<TrashType> Trash\_Type\_Getir()

{

List<TrashType> trashType = new List<TrashType>();

try

{

reader = DAL.Trash\_Type\_Getir();

while (reader.Read())

{

trashType.Add(new TrashType()

{

Trash\_Type\_ID = reader.IsDBNull(0) ? string.Empty : reader.GetString(0),

Trash\_Type\_Name = reader.IsDBNull(1) ? string.Empty : reader.GetString(1),

Capacity\_Amount = reader.IsDBNull(2) ? string.Empty : reader.GetString(2),

});

}

reader.Close();

}

catch (Exception ex)

{

}

finally

{

DAL.BaglantiAyarla();

}

return trashType;

}

internal List<Locations> Locations\_Getir()

{

List<Locations> locationType = new List<Locations>();

try

{

reader = DAL.Locations\_Getir();

while (reader.Read())

{

locationType.Add(new Locations()

{

Location\_ID = reader.IsDBNull(0) ? string.Empty : reader.GetString(0),

Location\_name = reader.IsDBNull(1) ? string.Empty : reader.GetString(1),

Region\_Name = reader.IsDBNull(2) ? string.Empty : reader.GetString(2),

Building\_Name = reader.IsDBNull(3) ? string.Empty : reader.GetString(3),

Storey\_Name = reader.IsDBNull(4) ? string.Empty : reader.GetString(4)

});

}

reader.Close();

}

catch (Exception ex)

{

}

finally

{

DAL.BaglantiAyarla();

}

return locationType;

}

internal List<WasteDisposalPointType> Waste\_Disposal\_Point\_Getir()

{

List<WasteDisposalPointType> wasteType = new List<WasteDisposalPointType>();

try

{

reader = DAL.Waste\_Disposal\_Point\_Getir();

while (reader.Read())

{

wasteType.Add(new WasteDisposalPointType()

{

Waste\_DP\_ID = reader.IsDBNull(0) ? string.Empty : reader.GetString(0),

Garbage\_Type\_Name = reader.IsDBNull(1) ? string.Empty : reader.GetString(1),

Capacity\_Amount = reader.IsDBNull(2) ? string.Empty : reader.GetString(2),

Region\_Name = reader.IsDBNull(3) ? string.Empty : reader.GetString(3),

});

}

reader.Close();

}

catch (Exception ex)

{

}

finally

{

DAL.BaglantiAyarla();

}

return wasteType;

}

internal List<BINGetir> BIN\_Getir()

{

List<BINGetir> binType = new List<BINGetir>();

try

{

reader = DAL.BIN\_Getir();

while (reader.Read())

{

binType.Add(new BINGetir()

{

BIN\_ID = reader.IsDBNull(0) ? string.Empty : reader.GetString(0),

Garbage\_Type\_Name = reader.IsDBNull(1) ? string.Empty : reader.GetString(1),

Garbage\_Type\_Definition = reader.IsDBNull(2) ? string.Empty : reader.GetString(2),

Trash\_Type\_ID = reader.IsDBNull(3) ? string.Empty : reader.GetString(3),

Waste\_DP\_ID = reader.IsDBNull(4) ? string.Empty : reader.GetString(4),

Location\_ID = reader.IsDBNull(5) ? string.Empty : reader.GetString(5)

});

}

reader.Close();

}

catch (Exception ex)

{

}

finally

{

DAL.BaglantiAyarla();

}

return binType;

}

internal int New\_Bin(string binId, string binName, string garbageTypeId, string

trashTypeId, string locationId, string wasteDPId)

{

if (!string.IsNullOrEmpty(binId) && !string.IsNullOrEmpty(binName) &&

!string.IsNullOrEmpty(garbageTypeId)

&& !string.IsNullOrEmpty(trashTypeId) && !string.IsNullOrEmpty(locationId) &&

!string.IsNullOrEmpty(wasteDPId))

{

Garbage G1 = new Garbage();

G1.BIN\_ID = binId;

G1.BIN\_Name = binName;

G1.Garbage\_Type\_ID = garbageTypeId;

G1.Trash\_Type\_ID = trashTypeId;

G1.Location\_ID = locationId;

G1.Waste\_DP\_ID = wasteDPId;

return DAL.New\_Bin(G1);

}

else

{

return -100;

}

}

internal List<Employee> Employee\_Getir()

{

List<Employee> employeType = new List<Employee>();

try

{

reader = DAL.Employee\_Getir();

while (reader.Read())

{

employeType.Add(new Employee()

{

Employee\_ID = reader.IsDBNull(0) ? string.Empty : reader.GetString(0),

Employee\_Name = reader.IsDBNull(1) ? string.Empty : reader.GetString(1),

Employee\_Surname = reader.IsDBNull(2) ? string.Empty : reader.GetString(2),

Giris\_Tarihi = reader.GetDateTime(3)

});

}

reader.Close();

}

catch (Exception ex)

{

}

finally

{

DAL.BaglantiAyarla();

}

return employeType;

}

internal List<BIMemployeGetir> Garbage\_Employee\_Getir()

{

List<BIMemployeGetir> binEmployeeType = new List<BIMemployeGetir>();

try

{

reader = DAL.Garbage\_Employee\_Getir();

while (reader.Read())

{

binEmployeeType.Add(new BIMemployeGetir()

{

BIN\_ID = reader.IsDBNull(0) ? string.Empty : reader.GetString(0),

Garbage\_Employee\_ID = reader.IsDBNull(1) ? string.Empty : reader.GetString(1),

Employee\_Name = reader.IsDBNull(2) ? string.Empty : reader.GetString(2),

Employee\_Surname = reader.IsDBNull(3) ? string.Empty : reader.GetString(3),

});

}

reader.Close();

}

catch (Exception ex)

{

}

finally

{

DAL.BaglantiAyarla();

}

return binEmployeeType;

}

internal int New\_Employee(string employeeID, string employeeName, string employeeSurname, DateTime giristarihi)

{

if (!string.IsNullOrEmpty(employeeID) && !string.IsNullOrEmpty(employeeID) &&

!string.IsNullOrEmpty(employeeSurname))

{

Employee E1 = new Employee();

E1.Employee\_ID = employeeID;

E1.Employee\_Name = employeeName;

E1.Employee\_Surname = employeeSurname;

E1.Giris\_Tarihi = giristarihi;

return DAL.New\_Employee(E1);

}

else

{

return -100;

}

}

internal int New\_Bin\_Employee(string binEmployeeId, string binID, string employeeId)

{

if (!string.IsNullOrEmpty(binEmployeeId) && !string.IsNullOrEmpty(binID) &&

!string.IsNullOrEmpty(employeeId))

{

BINEmployee BE1 = new BINEmployee();

BE1.Garbage\_Employee\_ID = binEmployeeId;

BE1.BIN\_ID = binID;

BE1.Employee\_ID= employeeId;

return DAL.New\_Bin\_Employee(BE1);

}

else

{

return -100;

}

}

internal int Kayit\_Guncelle(string binId, string binName, string garbageTypeId, string

trashTypeId, string locationId, string wasteDPId)

{

Garbage G2 = new Garbage()

{ BIN\_ID = binId,

BIN\_Name = binName,

Garbage\_Type\_ID = garbageTypeId,

Trash\_Type\_ID = trashTypeId,

Location\_ID = locationId,

Waste\_DP\_ID = wasteDPId

};

return DAL.Kayit\_Guncelle(G2);

}

internal Garbage Kayit\_Getir(string ID)

{

Garbage G = new Garbage();

try

{

reader = DAL.Kayit\_Getir(ID);

while (reader.Read())

{

G.BIN\_ID = reader.IsDBNull(0) ? string.Empty : reader.GetString(0);

G.BIN\_Name = reader.IsDBNull(1) ? string.Empty : reader.GetString(1);

G.Garbage\_Type\_ID = reader.IsDBNull(2) ? string.Empty : reader.GetString(2);

G.Trash\_Type\_ID = reader.IsDBNull(3) ? string.Empty : reader.GetString(3);

G.Location\_ID = reader.IsDBNull(4) ? string.Empty : reader.GetString(4);

G.Waste\_DP\_ID = reader.IsDBNull(5) ? string.Empty : reader.GetString(5);

}

}

catch

{

}

finally

{

DAL.BaglantiAyarla();

}

return G;

}

internal int Delete\_BIN(String ID)

{

return DAL.Delete\_BIN(ID);

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Data.SqlClient;

using System.Configuration;

namespace RGCA

{

public partial class AnaForm : Form

{

BusinessLogicLayer BLL;

SqlConnection connect = new SqlConnection();

public AnaForm()

{

InitializeComponent();

BLL = new BusinessLogicLayer();

GarbageDoldur();

datagridviewgöster11("execute SP\_BIN\_Getir");

datagridviewgöster("execute SP\_Employee\_Getir");

SqlConnection connect = new SqlConnection();

connect.ConnectionString = ConfigurationManager.ConnectionStrings["SQL"].ConnectionString;

}

private void GarbageTypeDoldur()

{

List<GarbageType> garbageTypeList = BLL.Garbage\_Type\_Getir();

lst\_display.DataSource = garbageTypeList;

}

private void GarbageDoldur()

{

List<Garbage> garbageTypeList = BLL.Garbage\_Getir();

lst\_displayBın.DataSource = garbageTypeList;

}

private void TrashTypeDoldur()

{

List<TrashType> trashTypeList = BLL.Trash\_Type\_Getir();

lst\_display.DataSource = trashTypeList;

}

private void LocationTypeDoldur()

{

List<Locations> locationTypeList = BLL.Locations\_Getir();

lst\_display.DataSource = locationTypeList;

}

private void WasteTypeDoldur()

{

List<WasteDisposalPointType> wasteTypeList = BLL.Waste\_Disposal\_Point\_Getir();

lst\_display.DataSource = wasteTypeList;

}

public void BinTypeDoldur()

{

List<BINGetir> binTypeList = BLL.BIN\_Getir();

lst\_display.DataSource = binTypeList;

}

private void button3\_Click(object sender, EventArgs e)

{

GarbageTypeDoldur();

datagridviewgöster11("Execute SP\_Garbage\_Getir");

}

private void button4\_Click(object sender, EventArgs e)

{

TrashTypeDoldur();

datagridviewgöster11("Execute SP\_Trash\_Type\_Getir");

}

private void button5\_Click(object sender, EventArgs e)

{

LocationTypeDoldur();

datagridviewgöster11("Execute SP\_Location\_Getir");

}

private void button6\_Click(object sender, EventArgs e)

{

WasteTypeDoldur();

datagridviewgöster11("Execute SP\_Waste\_Disposal\_Point\_Getir");

}

private void tabPage2\_Click(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

GarbageDoldur();

}

private void button7\_Click(object sender, EventArgs e)

{

int EtkilenenKayitSayisi = BLL.New\_Bin(txt\_binId11.Text,txt\_binName11.Text,

txt\_garbageType11.Text,txt\_location11.Text,txt\_trashType11.Text,txt\_wasteDPId11.Text);

if (EtkilenenKayitSayisi > 0)

{

MessageBox.Show("Record Added.", "Information",

MessageBoxButtons.OKCancel,

MessageBoxIcon.Warning);

}

else if (EtkilenenKayitSayisi == -100)

{

MessageBox.Show("All Gaps are required fields.", "Information",

MessageBoxButtons.OKCancel,

MessageBoxIcon.Warning);

}

else

{

MessageBox.Show("Failed to retrieve new record, see your system administrator", "Information",

MessageBoxButtons.OKCancel,

MessageBoxIcon.Warning);

}

}

private void groupBox4\_Enter(object sender, EventArgs e)

{

}

private void EmployeeGetir()

{

List<Employee> employeeList = BLL.Employee\_Getir();

lst\_display.DataSource = employeeList;

}

private void BinEmployeeGetir()

{

List<BIMemployeGetir> binEmployeeList = BLL.Garbage\_Employee\_Getir();

lst\_display.DataSource = binEmployeeList;

}

private void button13\_Click(object sender, EventArgs e)

{

EmployeeGetir();

datagridviewgöster("execute SP\_Employee\_Getir");

}

private void button14\_Click(object sender, EventArgs e)

{

BinEmployeeGetir();

datagridviewgöster("execute SP\_Garbage\_Employee\_Getir");

}

private void button2\_Click(object sender, EventArgs e)

{

int EtkilenenKayitSayisi = BLL.New\_Employee(txt\_employeeId11.Text, txt\_empName11.Text, txt\_empSurname11.Text, DateTime.Now);

if (EtkilenenKayitSayisi > 0)

{

MessageBox.Show("Record Added.", "Information",

MessageBoxButtons.OKCancel,

MessageBoxIcon.Warning);

}

else if (EtkilenenKayitSayisi == -100)

{

MessageBox.Show("All Gaps are required fields.", "Information",

MessageBoxButtons.OKCancel,

MessageBoxIcon.Warning);

}

else

{

MessageBox.Show("Failed to retrieve new record, see your system administrator", "Information",

MessageBoxButtons.OKCancel,

MessageBoxIcon.Warning);

}

}

private void button10\_Click(object sender, EventArgs e)

{

int EtkilenenKayitSayisi = BLL.New\_Bin\_Employee(txt\_Bin\_emp\_ID.Text, txt\_binID\_employee.Text, txt\_binEmpId.Text);

if (EtkilenenKayitSayisi > 0)

{

MessageBox.Show("Record Added.", "Information",

MessageBoxButtons.OKCancel,

MessageBoxIcon.Warning);

}

else if (EtkilenenKayitSayisi == -100)

{

MessageBox.Show("All Gaps are required fields.", "Information",

MessageBoxButtons.OKCancel,

MessageBoxIcon.Warning);

}

else

{

MessageBox.Show("Failed to retrieve new record, see your system administrator", "Information",

MessageBoxButtons.OKCancel,

MessageBoxIcon.Warning);

}

}

private void button11\_Click(object sender, EventArgs e)

{

BinTypeDoldur();

}

private void button12\_Click(object sender, EventArgs e)

{

EmployeeGetir();

}

private void AnaForm\_Load\_1(object sender, EventArgs e)

{

}

private void button8\_Click(object sender, EventArgs e)

{

string ID = ((Garbage)lst\_displayBın.SelectedItem).BIN\_ID;

int etiklenSayi = BLL.Kayit\_Guncelle

(txt\_binId11.Text,

txt\_binName11.Text,

txt\_garbageType11.Text,

txt\_trashType11.Text,

txt\_location11.Text,

txt\_wasteDPId11.Text);

if (etiklenSayi >0)

{

GarbageDoldur();

}

else

{

MessageBox.Show("Please try again");

}

}

private void lst\_display\_SelectedIndexChanged\_1(object sender, EventArgs e)

{

}

private void groupBox1\_Enter(object sender, EventArgs e)

{

}

private void button15\_Click(object sender, EventArgs e)

{

BinTypeDoldur();

datagridviewgöster11("execute SP\_BIN\_Getir");

}

private void button16\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void lst\_displayBın\_SelectedIndexChanged(object sender, EventArgs e)

{

ListBox lst = (ListBox)sender;

Garbage lstgarbage = (Garbage)lst.SelectedItem;

Garbage dbgarbage = BLL.Kayit\_Getir(lstgarbage.BIN\_ID);

if (dbgarbage != null)

{

txt\_binId11.Text = dbgarbage.BIN\_ID;

txt\_binName11.Text = dbgarbage.BIN\_Name;

txt\_garbageType11.Text = dbgarbage.Garbage\_Type\_ID;

txt\_trashType11.Text = dbgarbage.Trash\_Type\_ID;

txt\_location11.Text = dbgarbage.Location\_ID;

txt\_wasteDPId11.Text = dbgarbage.Waste\_DP\_ID;

}

}

private void button9\_Click(object sender, EventArgs e)

{

DialogResult result= MessageBox.Show("Are you sure you want to delete the record?", "Information",

MessageBoxButtons.OKCancel,

MessageBoxIcon.Warning);

if (result== DialogResult.OK)

{

Garbage garbage2 = (Garbage)lst\_displayBın.SelectedItem;

int etkilenenSayi = BLL.Delete\_BIN(garbage2.BIN\_ID);

if (etkilenenSayi > 0)

{

GarbageDoldur();

}

}

else

{

MessageBox.Show("Canceled", "Information",

MessageBoxButtons.OKCancel,

MessageBoxIcon.Warning);

}

}

private void maskedTextBox1\_MaskInputRejected(object sender, MaskInputRejectedEventArgs e)

{

}

public void datagridviewgöster(string veriler)

{

SqlConnection connect = new SqlConnection();

connect.ConnectionString = ConfigurationManager.ConnectionStrings["SQL"].ConnectionString;

SqlDataAdapter da = new SqlDataAdapter(veriler,connect);

DataSet ds = new DataSet();

da.Fill(ds);

dataGridView2.DataSource = ds.Tables[0];

}

public void datagridviewgöster11(string veriler)

{

connect.ConnectionString = ConfigurationManager.ConnectionStrings["SQL"].ConnectionString;

SqlDataAdapter da = new SqlDataAdapter(veriler, connect);

DataSet ds = new DataSet();

da.Fill(ds);

dataGridView1.DataSource = ds.Tables[0];

}

private void button1\_Click\_1(object sender, EventArgs e)

{

datagridviewgöster("Select\*from Employee");

}

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

private void button1\_Click\_2(object sender, EventArgs e)

{

}

private void dataGridView2\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

private void dataGridView1\_CellContentClick\_1(object sender, DataGridViewCellEventArgs e)

{

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RGCA

{

class BIMemployeGetir

{

public string BIN\_ID{ get; set; }

public string Garbage\_Employee\_ID { get; set; }

public string Employee\_Name { get; set; }

public string Employee\_Surname { get; set; }

public override string ToString()

{

return $"BIN ID:{BIN\_ID} " +

$"BIN Employee ID:{Garbage\_Employee\_ID} " +

$"Employee Name:{Employee\_Name} " +

$"Employee Surname:{ Employee\_Surname}";

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RGCA

{

class BINEmployee

{

public string Garbage\_Employee\_ID { get; set; }

public string BIN\_ID { get; set; }

public string Employee\_ID { get; set; }

public DateTime Star\_date { get; set; }

public override string ToString()

{

return $"BIN Employee ID:{Garbage\_Employee\_ID} " +

$"BIN ID:{BIN\_ID} " +

$"Employee ID:{Employee\_ID} " +

$"Starting Date:{Star\_date}";

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RGCA

{

class BINGetir

{

public string BIN\_ID { get; set; }

public string Garbage\_Type\_Name { get; set; }

public string Garbage\_Type\_Definition { get; set; }

public string Trash\_Type\_ID { get; set; }

public string Waste\_DP\_ID { get; set; }

public string Location\_ID { get; set; }

public override string ToString()

{

return $"BIN ID:{BIN\_ID} Garbage Type:{Garbage\_Type\_Name} " +

$" Definition:{Garbage\_Type\_Definition} Trash Type:{Trash\_Type\_ID} " +

$" Waste ID:{Waste\_DP\_ID} LOcation ID:{Location\_ID} ";

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RGCA

{

class Employee

{

public string Employee\_ID { get; set; }

public string Employee\_Name { get; set; }

public string Employee\_Surname { get; set; }

public DateTime Giris\_Tarihi { get; set; }

public override string ToString()

{

return $"Employee ID:{Employee\_ID} " +

$"Employee Name:{Employee\_Name} " +

$"Employee\_Surname:{Employee\_Surname} " +

$"Employee Starting Date:{Giris\_Tarihi}";

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace RGCA

{

public partial class Form1 : Form

{

BusinessLogicLayer BLL;

public Form1()

{

InitializeComponent();

BLL = new BusinessLogicLayer();

}

private void btn\_login\_Click(object sender, EventArgs e)

{

int EtkilenenKayitSayisi= BLL.SystemUserSystemEntryControl(txt\_userName.Text, txt\_userSifre.Text);

if (EtkilenenKayitSayisi>0)

{

AnaForm AnaForm1 = new AnaForm();

AnaForm1.Show();

}

else if (EtkilenenKayitSayisi == -100)

{

MessageBox.Show("Please fill in the form fields.");

}

else

{

MessageBox.Show("You have logged in incorrectly, check your details.");

}

}

private void Form1\_Load(object sender, EventArgs e)

{

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RGCA

{

internal class Garbage

{

public string BIN\_ID {get; set;}

public string BIN\_Name {get; set;}

public string Garbage\_Type\_ID {get; set;}

public string Trash\_Type\_ID{get; set;}

public string Location\_ID{get; set;}

public string Waste\_DP\_ID { get; set; }

public override string ToString()

{

return $"BIN ID:{BIN\_ID} BIN Name:{BIN\_Name} Garbage Type ID:{Garbage\_Type\_ID} " +

$"Trash Type ID:{Trash\_Type\_ID} Location ID:{Location\_ID} Waste ID:{Waste\_DP\_ID} ";

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RGCA

{

internal class GarbageType

{

public string Garbage\_Type\_ID { get; set; }

public string Garbage\_Type\_Name { get; set; }

public string Garbage\_Type\_Definition { get; set; }

public override string ToString()

{

return $"Garbage Type:{Garbage\_Type\_ID} Garbage Name:{Garbage\_Type\_Name} Garbage Definition:{Garbage\_Type\_Definition}";

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RGCA

{

class Locations

{

public string Location\_ID { get; set; }

public string Location\_name { get; set; }

public string Region\_Name { get; set; }

public string Building\_Name { get; set; }

public string Storey\_Name { get; set; }

public override string ToString()

{

return $"Location:{Location\_ID} Location Name:{Location\_name} Region Name:{Region\_Name}" +

$" Building Name:{Building\_Name} Storey:{Storey\_Name}";

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace RGCA

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RGCA

{

internal class SystemUsers

{

public string User\_Name { get; set; }

public string User\_Sifre { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RGCA

{

class TrashType

{

public string Trash\_Type\_ID { get; set; }

public string Trash\_Type\_Name { get; set; }

public string Capacity\_Amount { get; set; }

public override string ToString()

{

return $"Trash Type:{Trash\_Type\_ID} Trash Type{Trash\_Type\_Name} Capacity:{Capacity\_Amount}";

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RGCA

{

class WasteDisposalPointType

{

public string Waste\_DP\_ID { get; set; }

public string Garbage\_Type\_Name { get; set; }

public string Capacity\_Amount { get; set; }

public string Region\_Name { get; set; }

public override string ToString()

{

return $"Waste ID:{Waste\_DP\_ID} Garbage Type:{Garbage\_Type\_Name}" +

$" Capacity:{Capacity\_Amount} Region Name:{Region\_Name}";

}

}

}