Advanced Databases
Coursework

- **Child Well-Being Monitor**
- **Education inequality in Vietnam**
- **Greater Manchester Crime Profiler**

Kamal Kiani
Student ID: @00588509

Contents

Task 1: Child Well-Being Monitor

1.1 Abstract

In this task, the child well-being will be monitored in 4 countries including Ethiopia, Peru, Vietnam, and India. As the analysis tool SQL server express is used to complete the task1 requirements. The dataset used for this task is the "Young Lives Dataset" which is an international study of childhood poverty in 5 different rounds. In the following sections of this part the detailed characteristics of this dataset will be discussed and different reports will be generated based on this data.

1.2 Introduction

SQL server express is a free edition DBMS provided by Microsoft team, to store and manage data in a relational model and provides a powerful querying language. In this study SQL server express edition v.13 (2016) and SQL management studio version v.18 are used to complete the task1.

The dataset given for task1, is the "Young Lives Dataset" which is an international study of childhood poverty in 5 different rounds.

The mentioned survey is a long-term study discovering the nature of childhood poverty in four different regions and countries. The project conducted in Ethiopia, India, Peru, and Vietnam and aims to track the lives of 12,000 children over a period of 15-year, surveyed once every 3-4 years. Therefore, the younger children are being tracked from infancy to their mid-teens and the older children through into adulthood, when some will become parents themselves (Boyden, 2021).

Table1.1 indicates some more information about this dataset.

Dates of fieldwork:	*2002 - 2016*
Country:	Ethiopia , India, Peru, Vietnam
Observation units:	Individuals, Families/households
Population:	Young Lives children and their households, in Ethiopia, India (Andhra Pradesh), Peru and Vietnam, in 2002-2016.
Number of units:	Ethiopia: 14,995 cases, India: 15,097 cases, Peru: 13,830 cases, Vietnam: 15,000 cases.
Kind of data:	Numeric
Weighting:	No weighting used

Table1.1

To complete this task, in coming sections a database including 15 tables will be designed and SQL query language will be used to generate reports in SQL server environment.

1.3 Relational Schema

1.3.1 Tables identification

As we wanted to keep all the data in database and do not lose any necessary data, 15 separated tables designed which they are shown in the Table1.2. The reason behind this design is discussed in section1.4. The input values for these 15 tables come from 4 main tables holding downloaded datasets for each country. Figure1.1 shows a view of tables in SQL server.

Table Name	**Table Description**
t_general_info	Child's General Info
t_cohabited	*Child married or cohabited*
t_Identification	*Identification and location*
t_panel_info	*Panel information*
t_antro_info	*Child's anthropometric information*
t_immunisation	*Birth and immunisation*
t_health	*Child's health and well-being*
t_habit	*Smoking and drinking habits and reproductive health*
t_time_use	*Time-use*
t_education	*Education and skills*
t_mfc_info	*Mother, father, and caregiver characteristics*
t_h_head	*Household head characteristics*
t_h_size	*Household size and composition*
t_ownership	*Livestock, land, and house ownership*
t_h_shocks	*Household shocks*

Table1.2

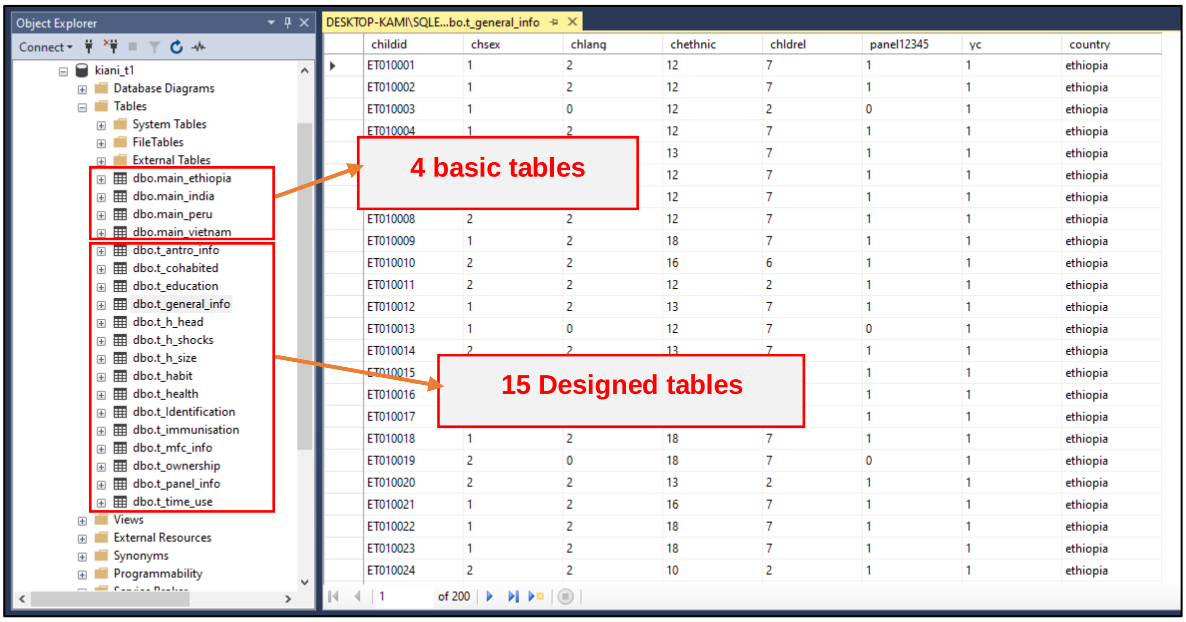


Figure1.1

1.3.2 Tables creation

To create the tables, first the 4 datasets which are in .tab format are loaded separately into SQL server using the import flat-file toolbox. Then, the desired tables have been created based on a design logic which will be explained in next sections. The tables creation and initial data insertion from 4 basic tables, is done by some SQL scripts. Figure1.2 shows a sample script to create "t_general_info" table and inserting initial values into it.

```sql
BEGIN TRY
    DROP TABLE IF EXISTS t_general_info;
    create table t_general_info (
        childid VARCHAR(15) , chsex int, chlang int, chethnic int,
        chldrel int, panel12345 int, yc int, country varchar(10),
        PRIMARY KEY (childid)
    );
    INSERT INTO t_general_info
        select distinct childid,chsex,chlang,chethnic,chldrel,panel12345,yc,'ethiopia'
        from main_ethiopia
    INSERT INTO t_general_info
        select distinct childid,chsex,chlang,chethnic,chldrel,panel12345,yc,'india'
        from main_india
    INSERT INTO t_general_info
        select distinct childid,chsex,chlang,chethnic,chldrel,panel12345,yc,'peru'
        from main_peru
    INSERT INTO t_general_info
        select distinct childid,chsex,chlang,chethnic,chldrel,panel12345,yc,'vietnam'
        from main_vietnam
END TRY
BEGIN CATCH
      SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_STATE() AS ErrorState,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

Table creation

Initial values from basic tables

Exception handling

Try-Catch block

Figure1.2

1.3.3 Tables relationship

The database diagram is shown figure1.3. where the "t_general_info" is the primary-key table and other 14 tables are foreign-key tables. The variable "childid" is the primary key in "t_general_info" and foreign key in the other tables to build the relationship between these tables. The relations between foreign-key tables and primary-key table, are in the kind of "one to many" or in other words "1-N".

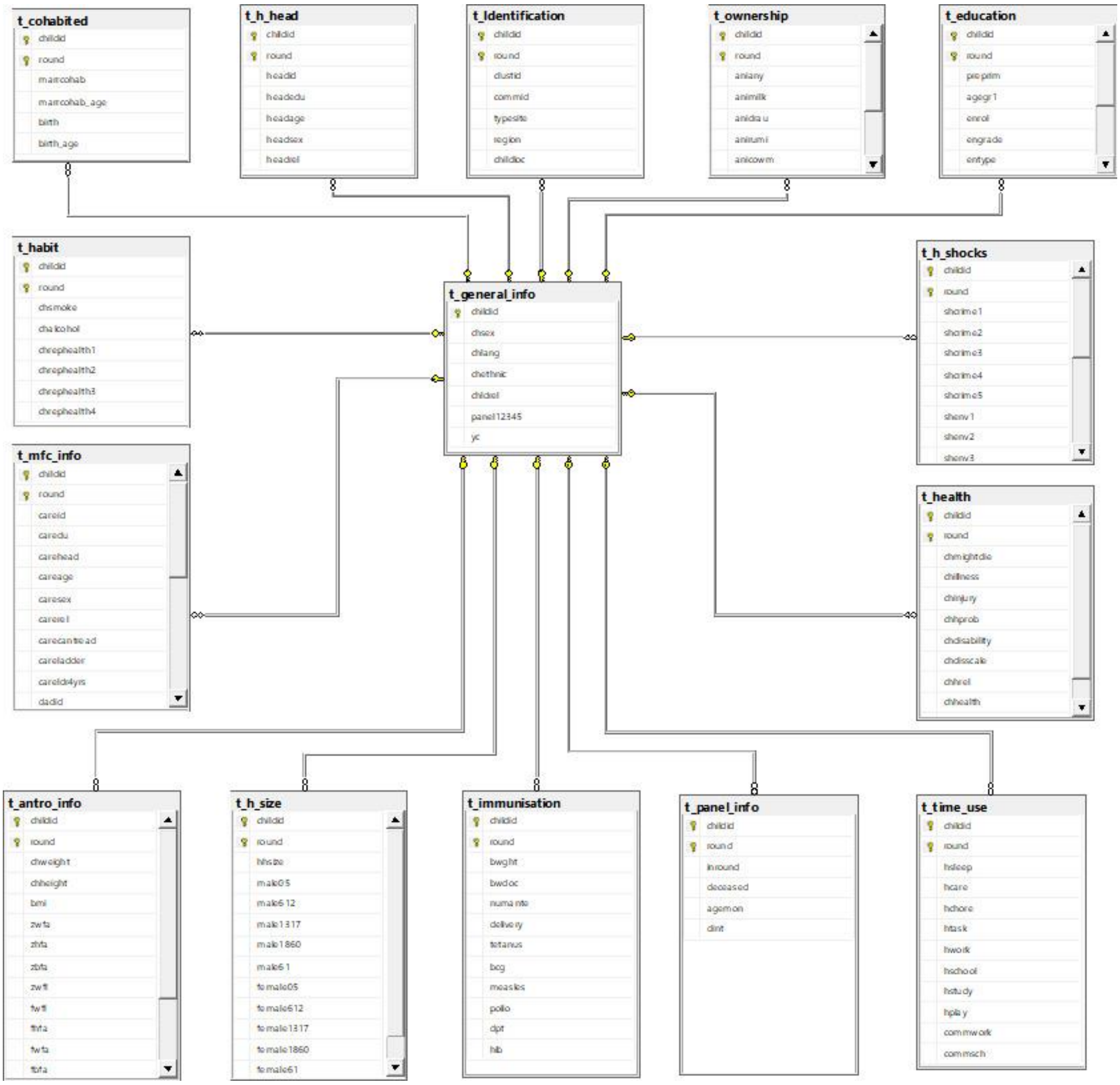


Figure1.3

Figure1.4 shows the created foreign keys and relationships. It is remarkable, this database is designed for a study purpose and a as there is no other users and there is no external transaction from this database, <u>the cascading rules based on these relationships are not necessary to be used.</u> In case we want to add cascading rules, we can define in the window shown in figure1.5 or even we can write a script when creating the table. A sample code to do this is the following code:

```
.
.
.
PRIMARY KEY (childid, [round]),
CONSTRAINT FK_t_antro_info FOREIGN KEY (childid) REFERENCES t_general_info(childid)
ON UPDATE CASCADE
ON DELETE CASCADE
```

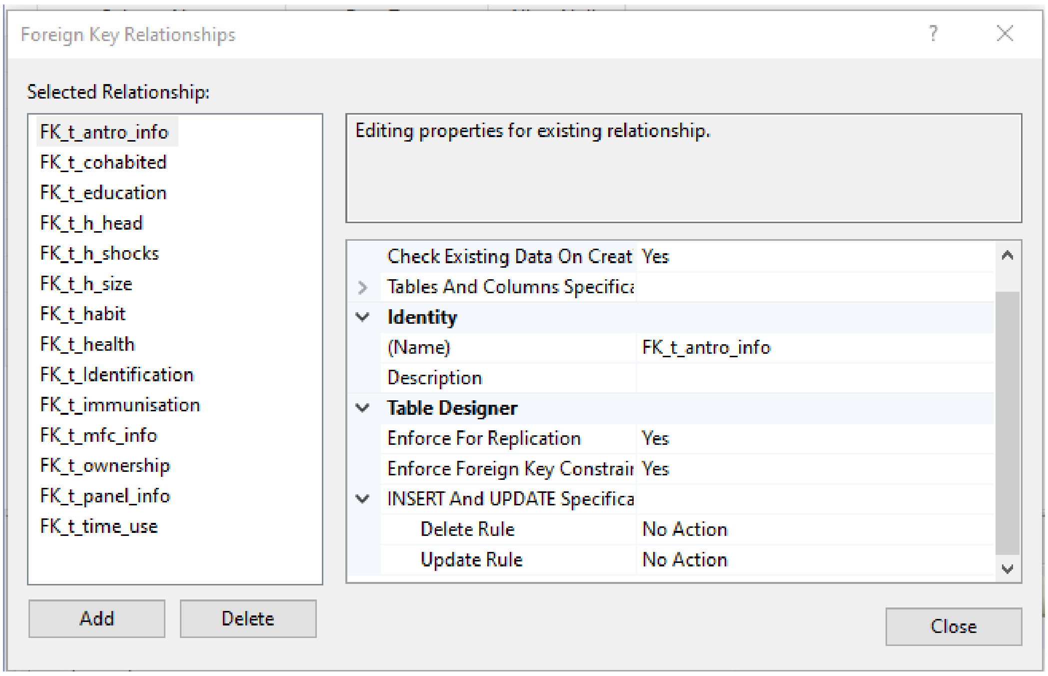


Figure1.4

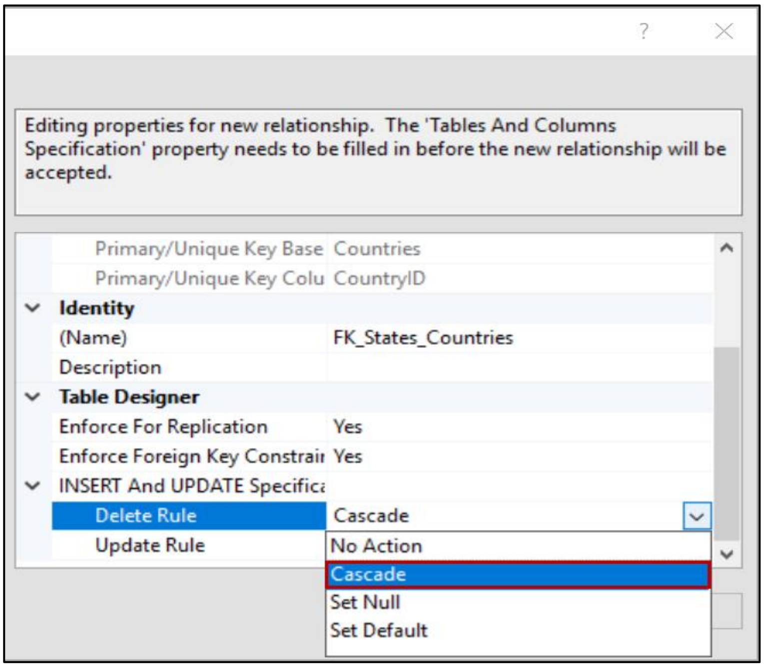


Figure1.5

1.4 Design Rationale

As mentioned before, the primary table is the "t_general_info" with the primary key "childid" which is a unique variable in this table. The child's general information will not change in any of the 5 rounds, thus, defining this table separately prevents static data repeating in 5 rounds.

Other 14 tables are divided based on the similarity of variables. As the value for these variables change in each round, the primary key for these tables is the combination of (childid, round).

According to the previous section, these 15 tables have relationships defined by foreign keys which enables us to join tables when we require more details from the tables.

1.5 Design Consideration

1.5.1. Database Normalization

According to Wikipedia, "*Database normalization is the process of restructuring a relational database in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity*". (wikipedia, n.d.)

Usually, if a database meets the 3 normal forms, it is acceptable as a proper normal database. Normalizing the database more than 3 levels, may increase the complexity in almost databases.

<u>The main 3 normal forms which is considered in our design</u> are as follows:

First Normal Form 1NF: (Jayaram, 2018)

- ✓ Rows and Columns are not ordered
- ✓ There may be duplicated data
- ✓ Row-and-column intersections always have a unique value
- ✓ All columns are "regular" with no hidden values

Second Normal Form 2NF: (Jayaram, 2018)

- ✓ The table must be already in 1 NF and all non-key columns of the tables must depend on the PRIMARY KEY
- ✓ The partial dependencies are removed and placed in a separate table

Third Normal Form 3NF: (Jayaram, 2018)

- ✓ A Table is already in 2 NF
- ✓ Non-Primary key columns shouldn't depend on the other non-Primary key columns
- ✓ There is no transitive functional dependency

The database <u>tables in task1 are in 3-NF normal</u> form as they are already in second normal form and there is no transitive dependency between columns and they are all directly related to the primary key.

1.5.2. Constraints and Data Validation

- ✓ Data types are properly selected based on variable types
- ✓ "Not Null" is used in table creation, when necessary
- ✓ Primary key: "childid" is the primary key in "t_general_info"
- ✓ Primary key: (childid, round) is the primary key in other tables

```
PRIMARY KEY (childid, [round]),
CONSTRAINT FK_t_habit FOREIGN KEY (childid) REFERENCES t_general_info(childid)
```

- ✓ Foreign key: "childid" is the foreign key in foreign-key tables

```
PRIMARY KEY (childid, [round]),
CONSTRAINT FK_t_habit FOREIGN KEY (childid) REFERENCES t_general_info(childid)
```

- ✓ Cascading rules are not necessary as the database in for study purpose
- ✓ Inserting restrictions are not necessary as the values are initially loading from a trusted source

1.5.3. Transaction and Concurrency Control

If locking and concurrency controlling is not used in a transactional database, and when no isolation exists between different transactions, some problems may occur:

1. Dirty reads
2. Nonrepeatable reads
3. Phantoms

As it is written in Microsoft website "A lower isolation level increases the ability of many users to access data at the same time. But it increases the number of concurrency effects. A higher isolation level reduces the types of concurrency effects that users might see. But it requires more system resources and increases the chances that one transaction will block another". (docs.microsoft.com, 2021)

The highest isolation level, serializable, guarantees that a transaction will retrieve exactly the same data every time it repeats a read operation. The lowest isolation level, read uncommitted, can retrieve data that has been modified but not committed by other transactions. All concurrency side effects can happen in read uncommitted, however there's no read locking or versioning, so overhead is minimized. (docs.microsoft.com, 2021).

Table1.3 shows the different Isolation levels recommended by Microsoft team.

Isolation Level	Dirty Read	Non-Repeatable Read	Phantom
Read uncommitted	Yes	Yes	Yes
Read committed	No	Yes	Yes
Repeatable read	No	No	Yes
Snapshot	No	No	No
Serializable	No	No	No

Table1.3

It is remarkeable that in our study, the used database is not a transactional database and it is designed and created for study porspuse. Thus, there is no need to use these isolations for our transactions and queries. If case it is needed, we can do so by a code like the following:

```
-- ensure we use SQL Server default isolation level (or we can change it!)
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;
    .
    .
    .
COMMIT TRANSACTION;
```

1.5.4. Error Handling

In this task scripts, the "Try-Catch" block is used when necessary to ensure we can handle any error. The syntax used is similar to the following code:

```
BEGIN TRY
   --code to try
END TRY
BEGIN CATCH
   --code to run if an error occurs
   --is generated in try
END CATCH
```

Anything between the "BEGIN TRY" and "END TRY" is the code that we want to monitor for an error. Inside the catch statement we have access to some special data as described below: (Petrovic, 2018)

- ERROR_NUMBER – Returns the internal number of the error
- ERROR_STATE – Returns the information about the source

- ERROR_SEVERITY – Returns the information about anything from informational errors to errors user of DBA can fix, etc.
- ERROR_LINE – Returns the line number at which an error happened on
- ERROR_PROCEDURE – Returns the name of the stored procedure or function
- ERROR_MESSAGE – Returns the most essential information and that is the message text of the error. (Petrovic, 2018)

These values are used in our code. A sample script for this is shown in figure1.6.

```
BEGIN TRY
    DROP TABLE IF EXISTS t_panel_info;
    create table t_panel_info (
        childid varchar(15) , [round] int, inround  int , deceased  int ,
        agemon  int , dint date ,
        PRIMARY KEY (childid, [round]),
        CONSTRAINT FK_t_panel_info FOREIGN KEY (childid) REFERENCES t_general_i
    );
    INSERT INTO t_panel_info
        select childid  ,[round]    ,inround    ,deceased   ,agemon ,dint
        from main_ethiopia
    INSERT INTO t_panel_info
        select childid  ,[round]    ,inround    ,deceased   ,agemon ,dint
        from main_india
    INSERT INTO t_panel_info
        select childid  ,[round]    ,inround    ,deceased   ,agemon ,dint
        from main_peru
    INSERT INTO t_panel_info
        select childid  ,[round]    ,inround    ,deceased   ,agemon ,dint
        from main_vietnam
END TRY
BEGIN CATCH
      SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_STATE() AS ErrorState,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

Try Block

Catch block

Figure1.6

1.5.5. Comments

Comments are used in the scripts when it was necessary to clearly define the syntax. Figure1.7 is a sample of commenting in the code while creating a view in task1.

```
View_Vaccinated.s...OP-KAMI\kami (53))*
/*
    This view counts the number of fully vaccinated childs in
    urban or rural regions separated by each country.

    The values for Vaccination variables are as follows:
    bcg = 1 :       Child have received BCG vaccination
    measles = 1 :   Child have received vaccination against measles
    dpt = 1 :       Child have received vaccination against DPT
    polio = 1 :     Child have received vaccination against polio
    hib = 1 :       Child have received vaccination against HIB
*/

CREATE or ALTER view View_Vaccinated AS
select  country,
    CASE
        WHEN typesite = 1 THEN 'urban'
        WHEN typesite = 2 THEN 'rural'
        ELSE 'N/A'
```

Figure1.7

1.6 T-SQL Statements

1.6.1 Tables

Tables creations are discussed in previous sections. Thus, here we just show a sample code in figure1.8.

```
BEGIN TRY
    DROP TABLE IF EXISTS t_general_info;
    create table t_general_info (
        childid VARCHAR(15) , chsex int, chlang int, chethnic int,
        chldrel int, panel12345 int, yc int, country varchar(10),
        PRIMARY KEY (childid)
    );
    INSERT INTO t_general_info
        select distinct childid,chsex,chlang,chethnic,chldrel,panel12345,yc,'ethiopia'
        from main_ethiopia
    INSERT INTO t_general_info
        select distinct childid,chsex,chlang,chethnic,chldrel,panel12345,yc,'india'
        from main_india
    INSERT INTO t_general_info
        select distinct childid,chsex,chlang,chethnic,chldrel,panel12345,yc,'peru'
        from main_peru
    INSERT INTO t_general_info
        select distinct childid,chsex,chlang,chethnic,chldrel,panel12345,yc,'vietnam'
        from main_vietnam
END TRY
BEGIN CATCH
      SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_STATE() AS ErrorState,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

Table creation

Initial values from basic tables

Exception handling

Try-Catch block

Figure1.8

1.6.2 Views

Three different views are created for task1, the first of which is a view named "**View_Smoking_Alcoholic**" to calculate the number of Childs who smoke or drink every day or at least once a week in each country. Figure1.9 is the code used to create this view. First, Q1 is done to find the number of alcoholic childs grouped by each country. Then, Q2 is done to calculate the number of smoking childs in each country group. Finally, these Q1 and Q2 are joined to show the result in one single table output that we save it a view. Figure1.10 is the output after running the code.

```
/*
    This code creates a view and calculates the number of childs
    who smoke or drink every day or at least once a week per each country
*/

CREATE or ALTER view View_Smoking_Alcoholic as

with
Q1 as (
    SELECT country , count(distinct t_habit.childid) as alcoholic_cnt
    from t_habit inner join t_general_info
    on t_habit.childid =  t_general_info.childid
    where chalcohol = 1
    group by country
),
Q2 as (
    SELECT country , count(distinct t_habit.childid) as smoking_cnt
    from t_habit inner join t_general_info
    on t_habit.childid =  t_general_info.childid
    where chsmoke    = 1 or chsmoke = 2
    group by country
)
select Q1.country , Q1.alcoholic_cnt, Q2.smoking_cnt
from Q1 inner join Q2
on Q1.country = Q2.country
```

Figure1.9

Results | Messages

	country	alcoholic_cnt	smoking_cnt
1	ethiopia	391	16
2	india	58	45
3	peru	96	139
4	vietnam	187	183

Figure1.10

Second View is the "**View_General_Health**" in figure1.11, that finds the child's health in general according to the final round data in each country. It uses a CASE syntax to map each "chhealth" value to its proper label. The grouping logic is based on 2 variables, "country" and "chhealth". The final output is shown in figure1.12.

```sql
/*
    This code creates a view to count the "Child's health in general" in the final round per each country
*/

CREATE or ALTER view View_General_Health AS
SELECT country,
    CASE
        WHEN chhealth = 1 THEN 'very poor'
        WHEN chhealth = 2 THEN 'poor'
        WHEN chhealth = 3 THEN 'average'
        WHEN chhealth = 4 THEN 'good'
        WHEN chhealth = 5 THEN 'very good'
        ELSE 'N/A'
    END AS child_health,
    count(t_health.childid) as cnt
FROM t_health inner join t_general_info on t_health.childid = t_general_info.childid
where round = 5 and chhealth != 0
group by country, chhealth
```

Figure1.11

Results | Messages

	country	child_health	cnt
1	ethiopia	very poor	27
2	ethiopia	poor	55
3	ethiopia	average	323
4	ethiopia	good	1112
5	ethiopia	very good	1107
6	india	very poor	11
7	india	poor	95
8	india	average	508
9	india	good	1849
10	india	very good	353
11	peru	very poor	5
12	peru	poor	31
13	peru	average	705
14	peru	good	1487
15	peru	very good	213
16	vietnam	very poor	5
17	vietnam	poor	77
18	vietnam	average	1814
19	vietnam	good	841
20	vietnam	very good	110

Figure1.12

The third view is "**View_Vaccinated**" and is shown in figure1.13. and it calculates the number of fully vaccinated childs in urban or rural regions in each country. Fully vaccinated means having 5 vaccines that are included in this study. These 5 vaccines are introduced below:

bcg	= 1:	Child have received BCG vaccination
measles	= 1:	Child have received vaccination against measles
dpt	= 1:	Child have received vaccination against DPT
polio	= 1:	Child have received vaccination against polio
hib	= 1:	Child have received vaccination against HIB

This view join 3 different tables "t_immunisation" for vaccine info, "t_general_info" for child's country data, and "t_Identification" for typesite (urban/rural) info. The output is illustrated in figure1.14.

```
/*
    This view counts the number of fully vaccinated childs in urban or rural regions per each country.
    bcg = 1 :       Child have received BCG vaccination
    measles = 1 :   Child have received vaccination against measles
    dpt = 1 :       Child have received vaccination against DPT
    polio = 1 :     Child have received vaccination against polio
    hib = 1 :       Child have received vaccination against HIB
*/

CREATE or ALTER view View_Vaccinated AS
select  country,
    CASE
        WHEN typesite = 1 THEN 'urban'
        WHEN typesite = 2 THEN 'rural'
        ELSE 'N/A'
    END AS type_site, count(*) as full_vaccinated
from t_immunisation inner join t_Identification
on t_immunisation.childid = t_Identification.childid
and t_immunisation.round = t_Identification.round
inner join t_general_info on t_general_info.childid = t_Identification.childid
where
    bcg = 1      and
    measles = 1 and
    dpt = 1      and
    polio = 1    and
    hib = 1      and
    typesite != 0
group by country , typesite
```

Figure1.13

	country	type_site	full_vaccinated
1	ethiopia	urban	216
2	ethiopia	rural	88
3	india	urban	316
4	india	rural	570
5	peru	urban	221
6	peru	rural	59
7	vietnam	urban	241
8	vietnam	rural	902

Figure1.14

1.6.3 Stored Procedure

"**SP_Mothers_Injections**" is a stored procedure defined to find the number mothers who received at least two injections for tetanus during pregnancy with YL chi. This procedure returns an integer value as the result of calling this procedure. The script and output of calling this procedure is shown in figure1.15.

```
/*
    This procedure returns the number mothers received at least
    two injections for tetanus during pregnancy with YL chi
*/

CREATE or ALTER PROCEDURE SP_Mothers_Injections
@cnt int OUTPUT
AS
    SET NOCOUNT ON;
    SELECT  @cnt = count( DISTINCT childid )
    from t_immunisation
    where tetanus = 1
RETURN @cnt
GO

DECLARE @ret nvarchar(15);
EXEC @ret = SP_Mothers_Injections @cnt = 0;
SELECT N'Result: ' + @ret;
```

100 %

Results | Messages

	(No column name)
1	Result: 4894

Figure1.15

1.6.4 User Defined Function

"**Func_UnderWeight**" is the function defined to calculate the number of childs with the given underweight stage (0/1/2). The input parameter for this function is the underweight stage which the value label information for "underweight" are as follows:

Value = 0.0 Label = not underweight

Value = 1.0 Label = moderately underweight

Value = 2.0 Label = severely underweight

Figure1.16 indicates the code and resulting output after calling the function.

```sql
/*
    The Function calculates number of childs with the given underweight stage (0/1/2)

    Value label information for underweight:
    Value = 0.0 Label = not underweight
    Value = 1.0 Label = moderately underweight
    Value = 2.0 Label = severely underweight
*/

CREATE or ALTER FUNCTION Func_UnderWeight(@UnderWeight_stage int)
RETURNS int
AS
BEGIN
    DECLARE @child_cnt int;
    SELECT @child_cnt = count(distinct childid)
    FROM t_antro_info
    WHERE underweight = @UnderWeight_stage
    RETURN @child_cnt;
END;
Go

DECLARE @ret nvarchar(15);
EXEC @ret = dbo.Func_UnderWeight @UnderWeight_stage = 2;
SELECT N'Result: ' + @ret;
```

100 %

Results | Messages

	(No column name)
1	Result: 1412

Figure1.16

1.7 Database Security

As the database security plan, we can define users and control the permissions for them. In this task, a user named "kamal" is defined and for a sample permission control, "insert" and "select" are granted for this user while "update" and "delete" are not allowed for the user in table "t_general_info". Figures1.17 shows the script for applying this control, and figure1.18 is the result of running code in the visual window in security tab.

```sql
IF  EXISTS (SELECT * FROM sys.database_principals WHERE name = N'kamal') DROP USER kamal

 /* new user */
CREATE USER kamal
FOR LOGIN myNewLogin;
GO

/* allowed for user */
GRANT SELECT, INSERT
ON t_general_info
TO kamal;

/* not allowed for user */
REVOKE DELETE, UPDATE
ON t_general_info
FROM kamal;
```

Figure1.17

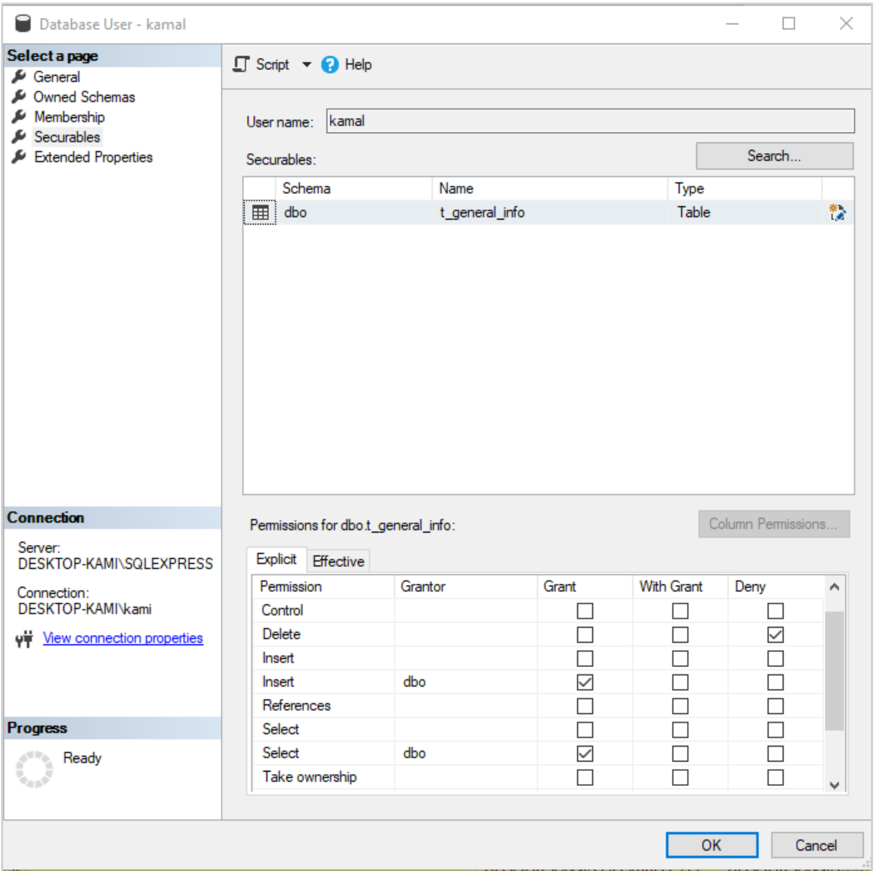


Figure1.18

1.8 Database Backup and Restore

Using the visual tools in SQL management studio or by using the SQL syntax, it is possible to get a full backup from the database and also restore the backup file whenever it is needed. Following figures1.19 and 1.20 show the SQL syntax for backup and restore operation in task1 database. The SQL server visual tool to take backup or restore the database are shown in figure1.21 and 1.22.

```
BACKUP DATABASE [kiani_t1]
TO  DISK = N'C:\Program Files\Microsoft SQL Server\MSSQL13.SQLEXPRESS\MSSQL\Backup\kiani_t1'
WITH NOFORMAT, NOINIT,  NAME = N'kiani_t1-Full Database Backup',
SKIP, NOREWIND, NOUNLOAD,  STATS = 10
GO
```

Figure1.19

```
USE [master]
RESTORE DATABASE [kiani_t1]
FROM  DISK = N'C:\Program Files\Microsoft SQL Server\MSSQL13.SQLEXPRESS\MSSQL\Backup\kiani_t1'
WITH  FILE = 1,  NOUNLOAD,  STATS = 5
GO
```

Figure1.20

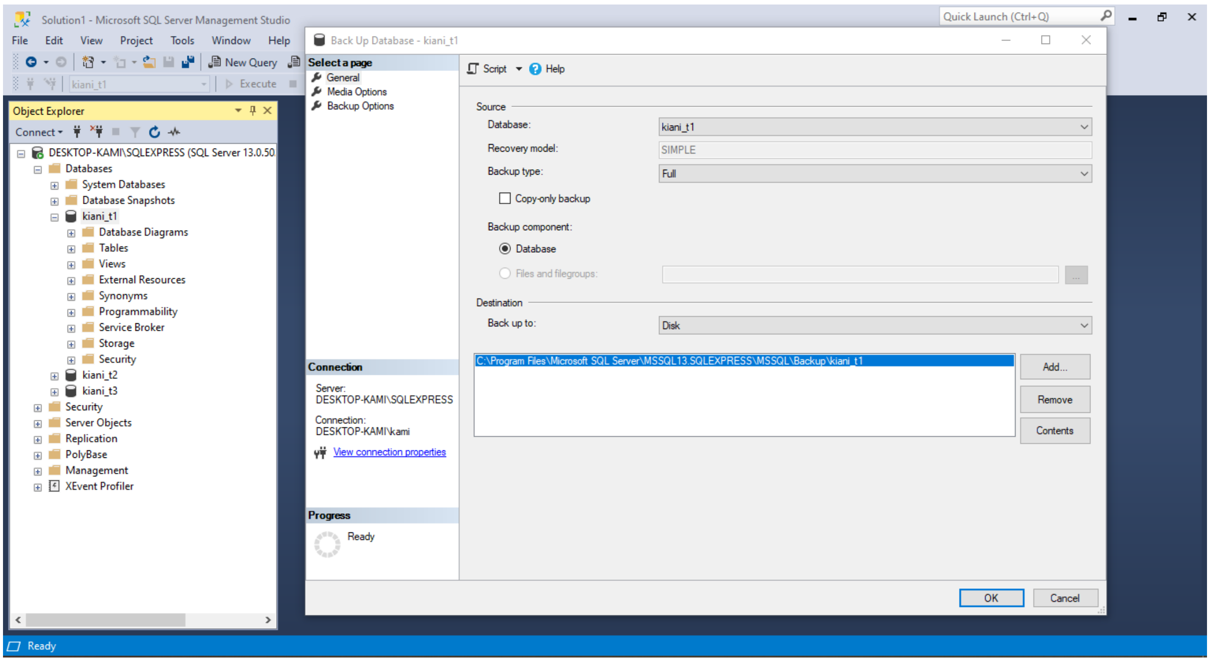


Figure1.21

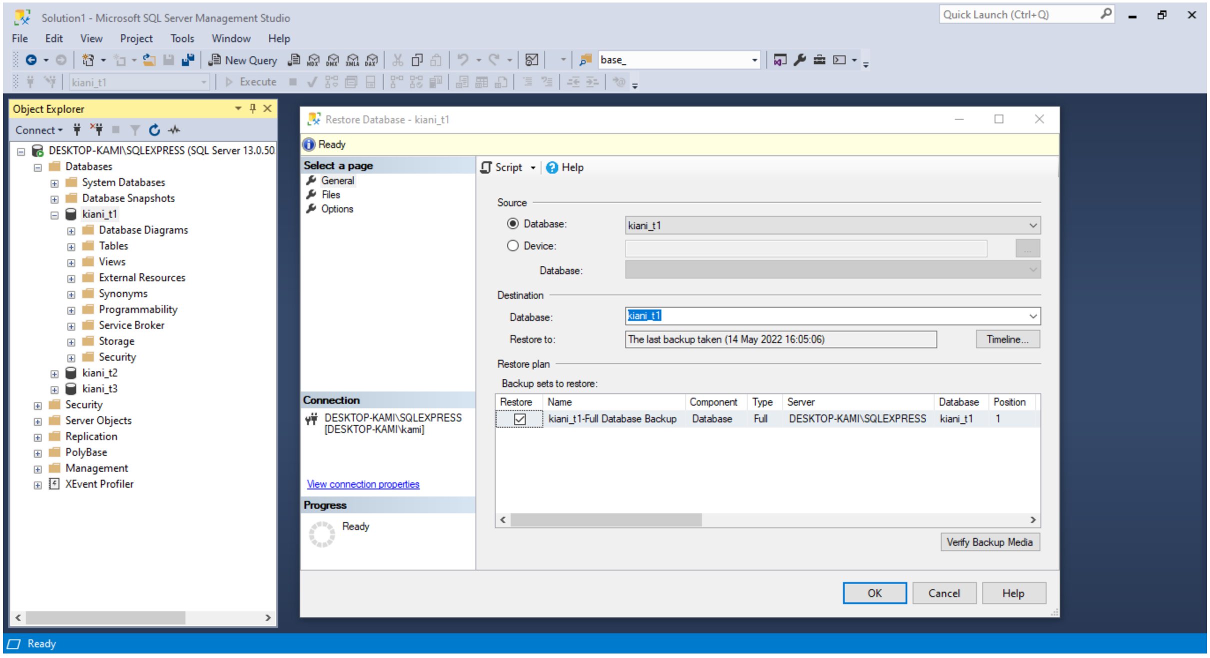


Figure1.22

1.9 Data Privacy, Ethical and Legal Issues

In this task we used a dataset which there are some information that are personal and sensitive. According to "Data Privacy, Ethical and Legal" rules protecting the personal information is vital and we can use these data just for the scientific and analytical purposes.

In the engineering level, we must make sure that there is no illegal access to the data. This can be made by defining security rules and plans.

In the analytical level, we must make sure that we do not focus on finding any personal information which leads us to find a clear individual person.

In the gathering information and preparing dataset level, we must make sure that sensitive data are not included before publishing. Due to the dataset used in this task, is downloaded from a legal source "**UK Data Services**" and according to detailed information they have provided in their website, all necessary protection rules and acts are applied on this dataset to ensure about the privacy and information protection against any harmful purposes.

As some examples from this dataset, we can see that all the values are coded, there is no name, contact information or exact location for childs and their families.

1.10 Conclusion

In this task, the child well-being monitored in 4 countries including Ethiopia, Peru, Vietnam, and India by SQL server as the analytical tool. As mentioned before, the dataset used for this task was the "Young Lives Dataset" which contains childhood poverty information in 5 different rounds. In sections of this part some queries were done to study the child-well being status in these countries. As a better look on what these queries told us, some diagrams are provided which are shown in figure1.23, figure1.24, and figure1.25.

Looking deep to these diagrams it shows us that the rural regions in Vietnam have the most value for fully vaccinated while the least value of fully vaccinated belongs to rural regions in Peru.

The second diagram identifies that, Ethiopia has the most alcoholic value between these countries and this value in significantly high in comparison to the other countries.

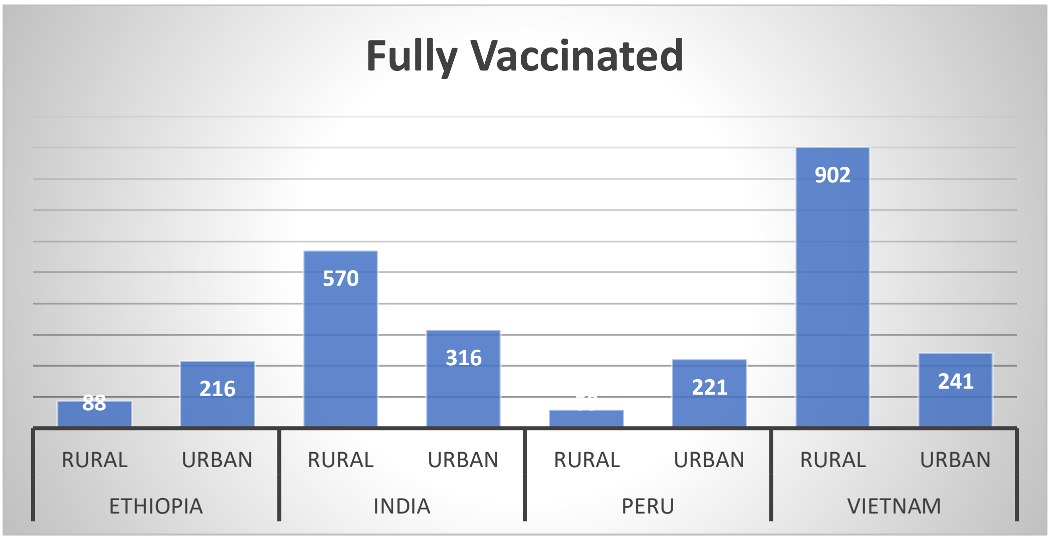


Figure1.23

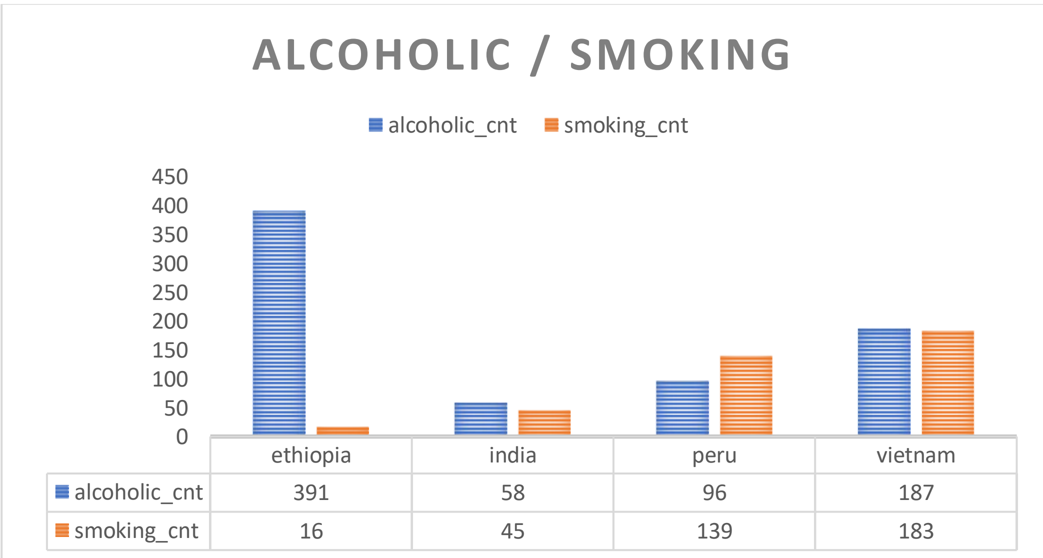


Figure1.24

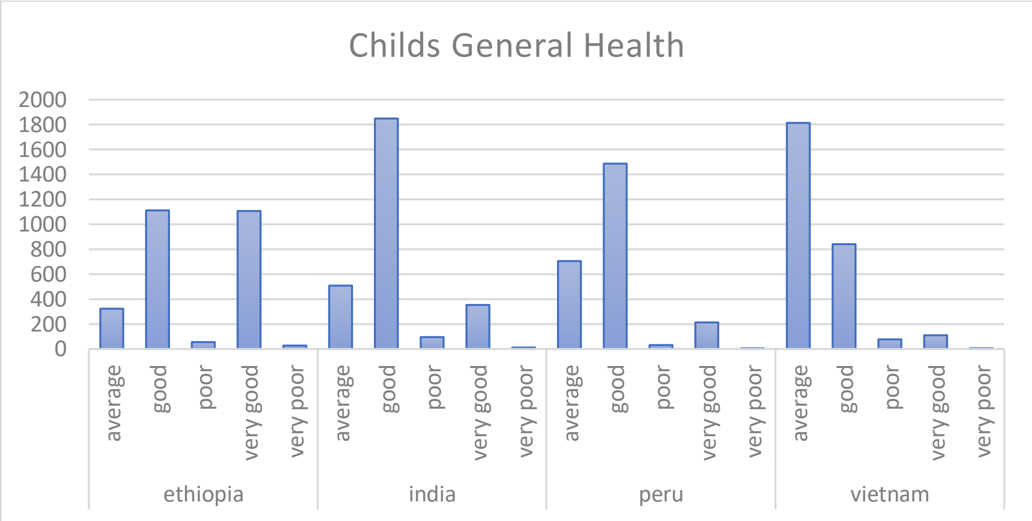

Childs General Health
2000
1800
1600
1400
1200
1000
800
600
400
200
0
average
good
poor
very good
very poor
ethiopia
india
peru
vietnam

Figure1.25

Task 2: Education inequality in Vietnam

2.1 Abstract

In the task2, the " Education inequality in Vietnam " will be studied. As the analysis tool SQL server express is used to complete the task2 requirements. The dataset used for this task is "**Young Lives: School Survey, Vietnam**" which has two separated files "vietnam_wave_1" and "vietnam_wave_2". The school survey focuses on the level of schooling accessed by 15-year-olds, so including Grade 10 students in Vietnam (upper secondary level). In the following sections of this part the detailed characteristics of this dataset will be discussed and different reports will be generated based on this data.

2.2 Introduction

In this study SQL server express edition v.13 (2016) and SQL management studio version v.18 are used to complete the task2. SQL server express is a free edition DBMS provided by Microsoft team, to store and manage data in a relational model and provides a powerful querying language.

The dataset given for task2, is the " **Young Lives: School Survey, Vietnam** " which focuses on the level of schooling accessed by 15-year-olds, including Grade 10 students in Vietnam (upper secondary level).

The survey allows researchers to link longitudinal information on household and child characteristics from the household survey with data on the schools attended by the Young Lives children and children's achievements inside and outside the school. It provides policy-relevant information on the relationship between child development (and its determinants) and children's experience of school, including access, quality, and progression. This combination of household, child and school-level data over time constitutes the comparative advantage of Young Lives (University of Oxford, 2018).

The Vietnam survey included data collection at the school, class, and pupil level, and involved the Principal / Head teacher, the Math's and English teachers, and the Young Lives child (University of Oxford, 2018). Table2.1 indicates some information about this dataset.

Dates of fieldwork:	**September 2016 - April 2017** Wave 1 between October and November 2016. Wave 2 between March and April 2017.
Country:	Vietnam
Spatial units:	No spatial unit
Observation units:	Individuals and Institutions/organisations
Observation unit location:	National
Population:	Schools in Young Lives survey sites, the pupils in Grade 10, the head teachers and class teachers, in Vietnam, 2016-2017.
Number of units:	8,740 Grade 10 students in 220 classes in 52 schools in five provinces of Vietnam
Method of data collection:	Face-to-face interview/ Self-completion/ Educational measurements/ Observation
Kind of data:	Numeric
Weighting:	No weighting used

Table2.1

To complete this task, in coming sections a database including 5 tables will be designed and SQL query language will be used to generate reports in SQL server environment.

2.3 Design Rationale

Task 2 database consists of 5 separated tables introduced in the Table2.2. The input values for these 5 tables come from 2 main tables holding downloaded datasets for each wave. Figure2.1 shows a view of tables in SQL server.

The primary table is the "t_student" with the primary key "UNIQUEID" which is a unique variable in this table. Other 4 tables are divided *based on the similarity of variables* and the primary key for these tables is " UNIQUEID ".

It is remarkable that UNIQUEID is the key in all tables and we can join tables when we require more details from the tables.

Table Name	Fields	Fields Description
t_student	UNIQUEID	
	SCHOOLID	
	CLASSID	
	STUDENTID	
	YLCHILDID	
	PROVINCE	
	DISTRICTCODE	
	LOCALITY	
	GENDER	
	AGE	
	ABSENT_DAYS	
	ETHNICITY	
t_educatin_fac	UNIQUEID	
	STNMBOOK	books
	STPLSTDY	place to study
	STHVDESK	Study desk
	STHVCHR	Study Chair
	STHVLAMP	Study lamp
t_comunication_fac	UNIQUEID	
	STHVCOMP	Computer/laptop
	STHVINTR	Internet
	STHVDVD	DVD player
	STHVCBLE	Cable TV box
	STHVMTEL	Mobile telephone
	STHVRADO	Radio
	STHVTELE	Television
t_extra_fac	UNIQUEID	
	STHVEFAN	Electric fan
	STHVAIRC	Air conditioning
	STHVFRDG	Fridge
	STHVMCRO	Microwave
	STHVBIKE	Bicycle
	STHVCAR	Car
	STHVMTBK	Motorbike
t_w2	UNIQUEID	
	STCMPSCH	computer for schoolwork -At school
	STCMPHME	computer for schoolwork -At home
	STPRVMT	private classes outside school in Maths
	STPRVEN	private classes outside school in English
	STPRVOT	private classes outside school in Other subjects

Table2.2

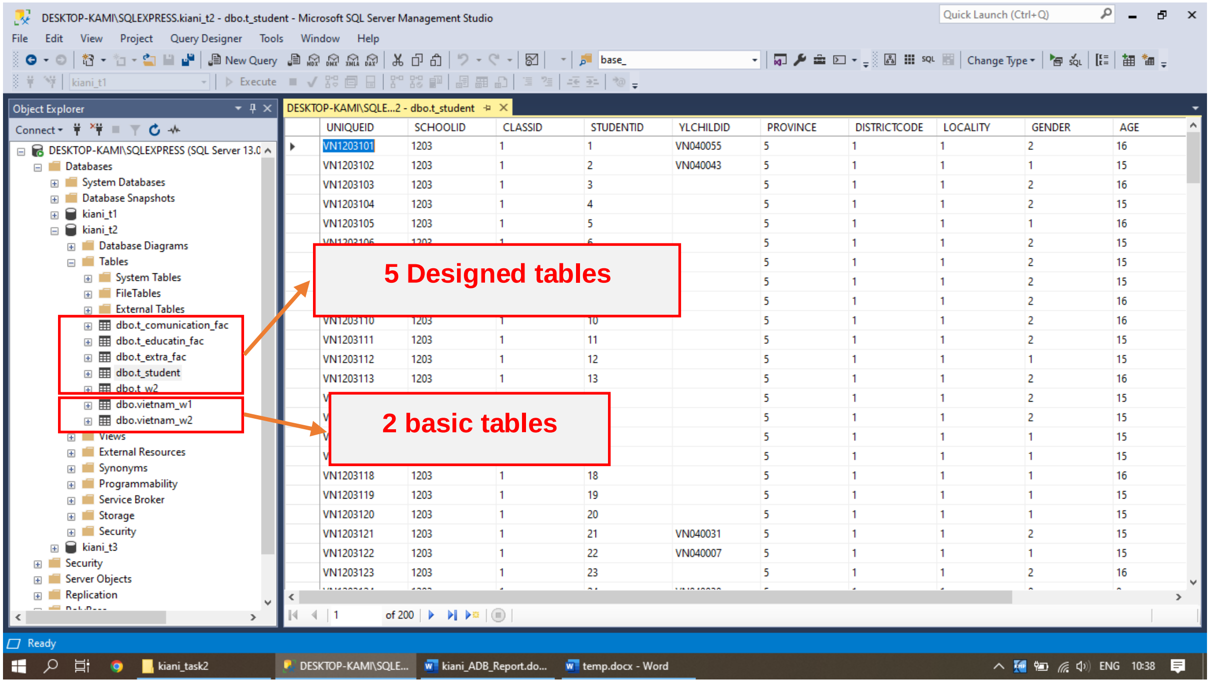


Figure2.1

2.4 Design Considerations

2.4.1 Database Normalization

Task2 database tables are in 3-NF normal form as they are already in second normal form and there is no transitive dependency between columns and they are all directly related to the primary key "UNIQUEID".

As discussed before in section1.5.1 about the database normalization, the detailed information about normalization is not included in this section and can be referred to section1.5.1.

2.4.2 Constraints and Data Validation

- ✓ Primary key: UNIQUEID is a primary key in "t_student" table and also primary key in all other tables.
- ✓ Some values like the primary keys are set to "Not Null"
- ✓ UNIQUEID could be the foreign key to tables but as the relation between tables are 1:1 (one to one) it is not defined explicitly.
- ✓ Data types are defined based on variable types
- ✓ Cascading rules are not necessary as the database in for study purpose
- ✓ Inserting restrictions are not necessary as the values are initially loading from a trusted source

Some of these items are clear in figure2.2.

Column Name	Data Type	Allow Nulls
UNIQUEID	varchar(50)	☐
SCHOOLID	int	☑
CLASSID	int	☑
STUDENTID	int	☑
YLCHILDID	varchar(50)	☑
PROVINCE	int	☑
DISTRICTCODE	int	☑
LOCALITY	int	☑
GENDER	int	☑
AGE	int	☑
ABSENT_DAYS	int	☑
ETHNICITY	int	☑
		☐

Figure2.2

2.4.3 Transaction and Concurrency Control

As discussed before in section1.5.3 about the Transaction concurrency and isolations, the detailed information about this topic is not included in this section and therefore can be referred to section1.5.3.

It is remarkeable that in our study, the used database is not a transactional database and it is designed and created for study porspuse. Thus, there is no need to use the isolation techniques for queries. If case it is needed, we can do so by a code like the following:

```
-- ensure we use SQL Server default isolation level (or we can change it!)
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;
    .
    .
    .
COMMIT TRANSACTION;
```

2.4.4 Error Handling

The "Try-Catch" block is used when necessary to ensure we can handle any error. The syntax used is similar to the following code:

```
BEGIN TRY
   --code to try
END TRY
```

```
BEGIN CATCH
    --code to run if an error occurs
    --is generated in try
END CATCH
```

Figure2.3 shows a sample script for error handling using Try-Catch block in task2 database while creating "t_education_fac" table.

```
BEGIN TRY
    DROP TABLE IF EXISTS t_educatin_fac;

      Create Table t_educatin_fac (
        UNIQUEID     varchar(50) ,
        STNMBOOK     int ,
        STPLSTDY     int ,
        STHVDESK     int ,
        STHVCHR      int ,
        STHVLAMP     int
        PRIMARY KEY (UNIQUEID));

    INSERT INTO t_educatin_fac
        SELECT   UNIQUEID, STNMBOOK,    STPLSTDY,    STHVDESK,    STHVCHR,    STHVLAMP
        FROM vietnam_w1;
END TRY
BEGIN CATCH
      SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_STATE() AS ErrorState,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

Try Block

Catch block

Figure2.3

2.4.5 Comments

Comments are used in the scripts when it was necessary to clearly define the syntax. Figure2.4 is a sample of commenting in the code while creating a view in task2.

```
/*
    This code, creates a view to find the number of students who do not
    attend any private classes in English or Math, seprated by provience

    Value label information for STPRVMT :
    Value = 88.0    Label = NA
    Value = 1.0 Label = Yes, to excel in class
    Value = 2.0 Label = Yes, to keep up in class
    Value = 3.0 Label = No
    Value = 79.0    Label = Missing

    Value label information for STPRVEN :
    Value = 88.0    Label = NA
    Value = 1.0 Label = Yes, to excel in class
    Value = 2.0 Label = Yes, to keep up in class
    Value = 3.0 Label = No
    Value = 79.0    Label = Missing
*/

CREATE or ALTER VIEW View_PrivateClasses AS

SELECT
    Case
        WHEN PROVINCE =1 THEN 'Ben Tre'
```

Figure2.4

2.5 T-SQL Statements

2.5.1 Tables

As mentioned before 5 tables are created by SQL scripts and 2 basic tables are loaded from the raw dataset "wave1" and "wave2". As two samples of how tables are created, 2 following figures show the syntax used as this purpose.

```sql
BEGIN TRY
    DROP TABLE IF EXISTS t_student;

    Create Table t_student (
        UNIQUEID    varchar(50) ,
        SCHOOLID    int ,
        CLASSID int ,
        STUDENTID   int ,
        YLCHILDID   varchar(50) ,
        PROVINCE    int ,
        DISTRICTCODE    int ,
        LOCALITY    int ,
        GENDER  int ,
        AGE int ,
        ABSENT_DAYS int ,
        ETHNICITY   int ,
        PRIMARY KEY (UNIQUEID));

    INSERT INTO t_student
        SELECT UNIQUEID,SCHOOLID,CLASSID,STUDENTID,YLCHILDID,PROVINCE,DISTRI
        FROM vietnam_w1;
END TRY
BEGIN CATCH
      SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_STATE() AS ErrorState,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

Table creation

Initial values from basic table

Exception handling

Try-Catch block

Figure2.5

```sql
BEGIN TRY
    DROP TABLE IF EXISTS t_w2;

    Create Table t_w2 (
        UNIQUEID    varchar(50) ,
        STCMPSCH    int ,
        STCMPHME    int ,
        STPRVMT     int ,
        STPRVEN     int ,
        STPRVOT     int ,
        PRIMARY KEY (UNIQUEID));

    INSERT INTO t_w2
        SELECT UNIQUEID,    STCMPSCH,    STCMPHME,    STPRVMT,
        FROM vietnam_w2;
END TRY
BEGIN CATCH
      SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_STATE() AS ErrorState,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

Table creation

Initial values from basic table

Exception handling

Try-Catch block

Figure2.6

2.5.2 Views

In this part for task2, three different views are generated to monitor the inequality state in Vietnam education system.

The first view is called "View_UsingComp" which counts number of males and females based on how often they use a computer for schoolwork at home. The value label information for "STCMPHME" variable is as follows:

- ✓ Value = 1.0 Label = Never or almost never
- ✓ Value = 2.0 Label = Once or twice a month
- ✓ Value = 3.0 Label = Once or twice a week
- ✓ Value = 4.0 Label = Every day or almost every day

Figure2.7 is the code to run this query and build the view and figure2.8 is the resulting output table view. In this script CASE statement is used to map each coded value to its equivalent label. Grouping technique is used based on Gender and STCMPHME variables.

```
/*
This code creates a view and counts number of males and females based on
how often do they use a computer for schoolwork -At home

    Value label information for STCMPHME
    Value = 1.0 Label = Never or almost never
    Value = 2.0 Label = Once or twice a month
    Value = 3.0 Label = Once or twice a week
    Value = 4.0 Label = Every day or almost every day
*/

CREATE or ALTER view View_UsingComp as
select
    Case
        WHEN STCMPHME =1 THEN 'Never or almost never'
        WHEN STCMPHME =2 THEN 'Once or twice a month'
        WHEN STCMPHME =3 THEN 'Once or twice a week'
        WHEN STCMPHME =4 THEN 'Every day or almost every day'
        ELSE 'NA'
    END as Type,
    Case
        WHEN GENDER =1 THEN 'Male'
        WHEN GENDER =2 THEN 'Female'
        ELSE 'NA'
    END as Gender,
    count(*) as 'Computer for schoolwork at home'
from t_w2 inner join t_student on t_w2.UNIQUEID =  t_student.UNIQUEID
where Gender in (1,2) and STCMPHME in(1,2,3,4)
group by STCMPHME , GENDER
```

Figure2.7

Results | Messages

	Type	Gender	Computer for schoolwork at home
1	Never or almost never	Female	1122
2	Never or almost never	Male	1325
3	Once or twice a month	Female	547
4	Once or twice a month	Male	490
5	Once or twice a week	Female	1109
6	Once or twice a week	Male	965
7	Every day or almost every day	Female	1164
8	Every day or almost every day	Male	933

Figure2.8

The second view is called " View_PrivateClasses " and finds the number of students who do not attend any private classes in English or Math, separated by providence. The code joins two tables t_w2 and t_student to find the result. Figure2.9 and figure2.10 are the code and output for this view.

```
/*
    This code, creates a view to find the number of students who do not
    attend any private classes in English or Math, seprated by provience

    Value label information for STPRVMT :
    Value = 88.0    Label = NA
    Value = 1.0 Label = Yes, to excel in class
    Value = 2.0 Label = Yes, to keep up in class
    Value = 3.0 Label = No
    Value = 79.0    Label = Missing

    Value label information for STPRVEN :
    Value = 88.0    Label = NA
    Value = 1.0 Label = Yes, to excel in class
    Value = 2.0 Label = Yes, to keep up in class
    Value = 3.0 Label = No
    Value = 79.0    Label = Missing
*/
CREATE or ALTER VIEW View_PrivateClasses AS
SELECT
    Case
        WHEN PROVINCE =1 THEN 'Ben Tre'
        WHEN PROVINCE =2 THEN 'Da Nang'
        WHEN PROVINCE =3 THEN 'Hung Yen'
        WHEN PROVINCE =4 THEN 'Lao Cai'
        WHEN PROVINCE =5 THEN 'Phu Yen'
        ELSE 'NA'
    END as PROVINCE, count(distinct t_student.UNIQUEID) as No_Private_Classes
FROM t_student inner join t_w2
on t_student.UNIQUEID = t_w2.UNIQUEID
where STPRVMT = 3 or STPRVEN = 3
group by PROVINCE
```

Figure2.9

Results | Messages

	PROVINCE	No_Private_Classes
1	Hung Yen	912
2	Ben Tre	1150
3	Lao Cai	1347
4	Phu Yen	1567
5	Da Nang	568

Figure2.10

The third view is " View_StudyPlace " which calculates the number and percentage of students in each ETHNICITY who have study place and basic study facilities.

Figure2.11 shows the used script to create this view. As it is shown, "WITH" statement is used in this query to find the total number of students in each ETHNICITY as Q1 and number of students who have study place and basic facilities in each ETHNICITY as Q2. Then, based on Q1 and Q2 the final result and percentage in each group is calculated. To calculate the percentage, we have to "CAST" the integer values to DECIMAL to get a precise percentage.

Figure2.12 is the result of running this code.

```
/*  This code creates a view to calculate the number and percentage of students
    in each ETHNICITY who have study place and basic study facilities    */
CREATE or ALTER VIEW View_StudyPlace AS
with Q1 as (
SELECT
    Case
        WHEN ETHNICITY =1 THEN 'Kinh'
        WHEN ETHNICITY =2 THEN 'H Mong'
        WHEN ETHNICITY =3 THEN 'Cham-HRoi'
        WHEN ETHNICITY =4 THEN 'Ede'
        WHEN ETHNICITY =5 THEN 'Ba Na'
        WHEN ETHNICITY =6 THEN 'Nung'
        WHEN ETHNICITY =7 THEN 'Tay'
        WHEN ETHNICITY =8 THEN 'Dao'
        WHEN ETHNICITY =9 THEN 'Giay'
        WHEN ETHNICITY =10 THEN 'Other'
        WHEN ETHNICITY =99 THEN 'Missing'
        WHEN ETHNICITY =88 THEN 'NA'
        ELSE 'NA'
    END as ETHNICITY_Label, ETHNICITY, count(*) as Total
FROM t_student
group by ETHNICITY
),
Q2 as (
    SELECT ETHNICITY, count(*) as StudyPlace_and_Facilities
    FROM t_student inner join t_educatin_fac
    on t_student.UNIQUEID = t_educatin_fac.UNIQUEID
    where STPLSTDY=1 and STHVDESK=1 and STHVCHR=1
    group by ETHNICITY
)
select ETHNICITY_Label, Total, StudyPlace_and_Facilities,
        Round (CAST(StudyPlace_and_Facilities AS DECIMAL(7,2)) / CAST(Total AS DECIMAL(7,2)) *100 , 2 ) as [percent]
from Q1 inner join Q2 on Q1.ETHNICITY = Q2.ETHNICITY
```

Figure2.11

Results | Messages

	ETHNICITY_Label	Total	StudyPlace_and_Facilities	percent
1	NA	327	6	1.8300000000
2	Ede	22	5	22.7300000000
3	H Mong	376	115	30.5900000000
4	Other	52	19	36.5400000000
5	Dao	235	114	48.5100000000
6	Ba Na	4	2	50.0000000000
7	Cham-HRoi	37	21	56.7600000000
8	Giay	141	103	73.0500000000
9	Nung	80	66	82.5000000000
10	Tay	98	84	85.7100000000
11	Kinh	7368	6689	90.7800000000

Figure2.12

2.6 Report Design

To have a better sight on results and deep into the output values, and in the other hand have a report panel to monitor future changes in data, some diagrams are created using Microsoft Excel reporting tools. To do this, 2 ways and approaches are available, the first one is the import data menu in excel software that connect to the SQL server database in a local machine and imports the data into excel sheets to generate any reports and plot any diagrams. (figure2.14)

The second approach is to use " ODBC connection" to connect Excel directly to SQL server using windows ODBC tool.(figure2.13)

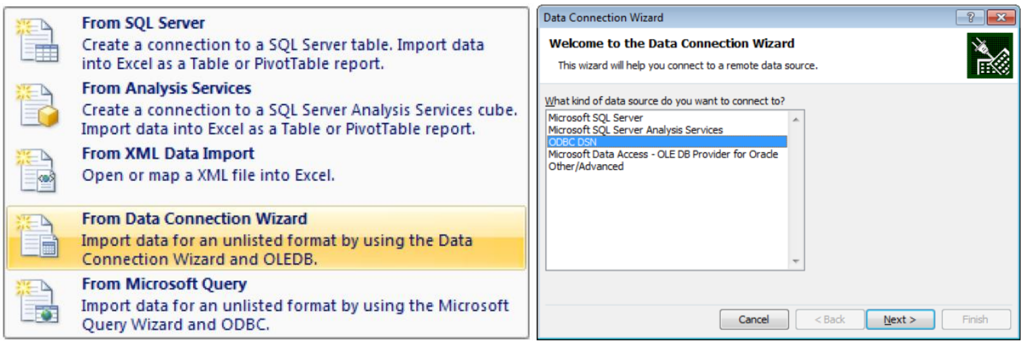


Figure2.13

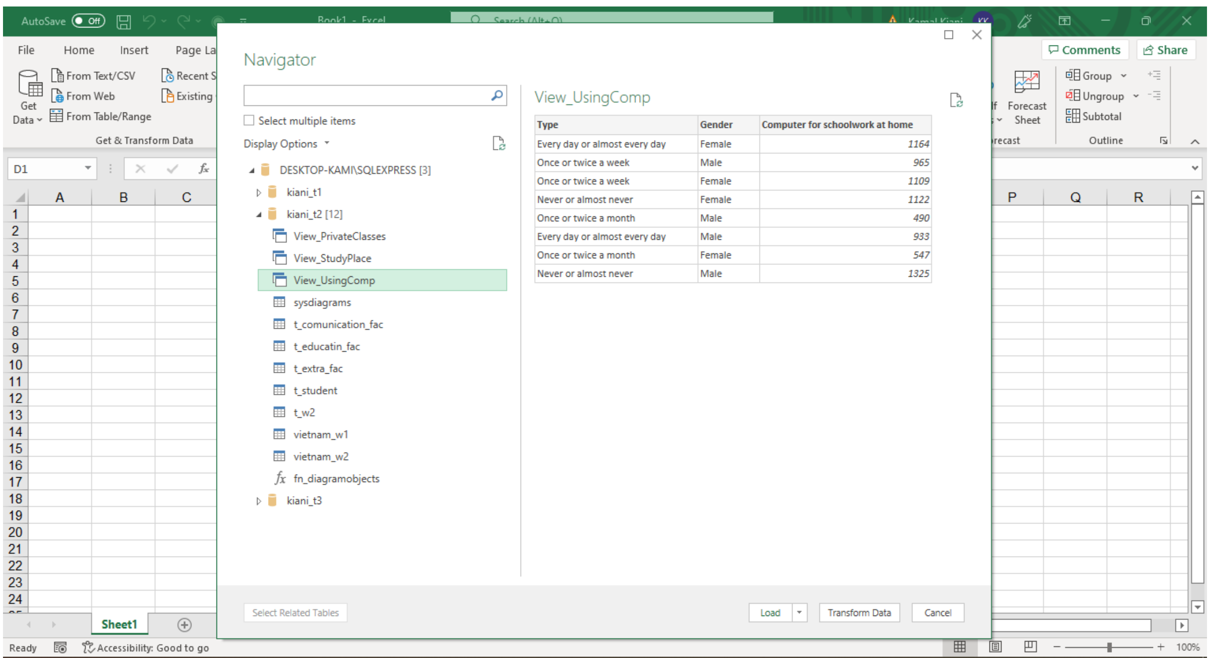


Figure2.14

After importing the 3 views from SQL server into Excel, some different diagrams are plotted to have a better understanding about data. Figure2.15 indicates that, most of the student in Vietnam "Never or almost never" use computer at home for their homework and "male" value is higher than "female" value for student in this group.

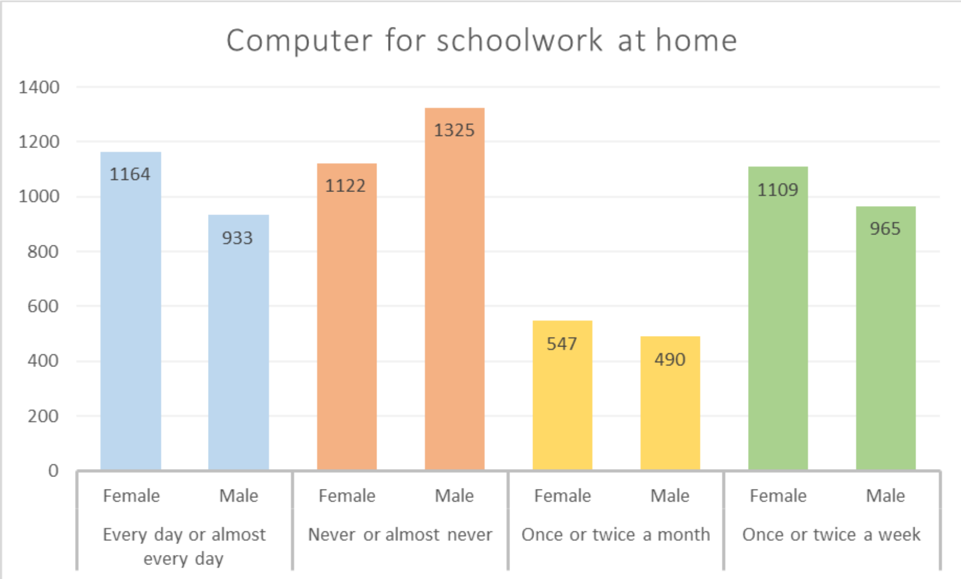
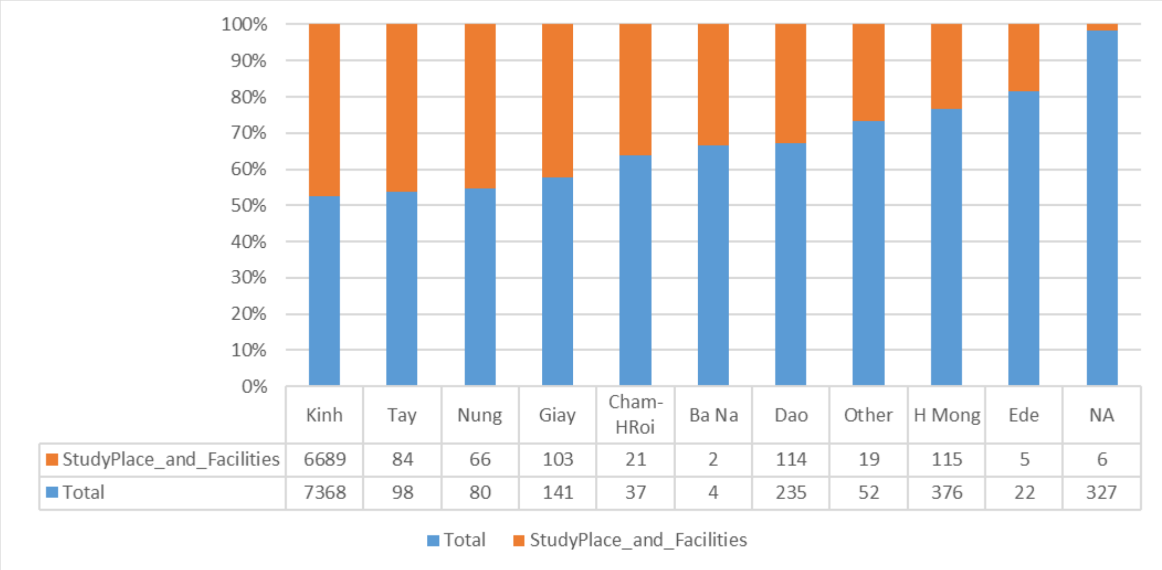


Figure2.15

Figure2.16 shows us that in "Da Nang" students are less motivated to go in private classes for Math or English, while students in "Phu Yen" has the most participation in English or Math private classes.

Figure2.16

Figure2.17 and figure2.18 shows the percentage of students who has basic study facilities and study place. It is clear in these diagrams that in "Kinh" group most of the students have these facilities and in "Ede" group the least students have facilities and a place to study.



	Kinh	Tay	Nung	Giay	Cham-HRoi	Ba Na	Dao	Other	H Mong	Ede	NA
StudyPlace_and_Facilities	6689	84	66	103	21	2	114	19	115	5	6
Total	7368	98	80	141	37	4	235	52	376	22	327

Figure2.17

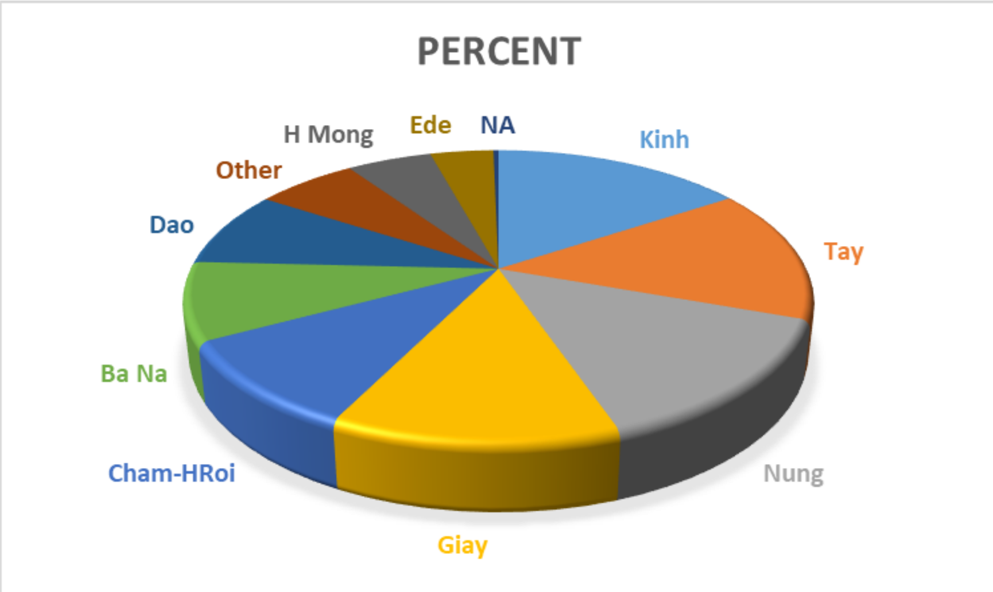


Figure2.18

2.7 Database Security

As the database security plan, we can define users and control the permissions for them. In this task, a user named "kamal" is defined and for a sample permission control, "insert" and "select" are granted for this user while "update" and "delete" are not allowed for the user in table "t_student". Figure2.19 shows the script for applying this control, and figure2.20 is the result of running code in the visual window in security tab.

```sql
IF  EXISTS (SELECT * FROM sys.database_principals WHERE name = N'kamal') DROP USER kamal

 /* new user */
CREATE USER kamal
FOR LOGIN myNewLogin;
GO

/* allowed for user */
GRANT SELECT, INSERT
ON t_student
TO kamal;

/* not allowed for user */
REVOKE DELETE, UPDATE
ON t_student
FROM kamal;
```

Figure2.19

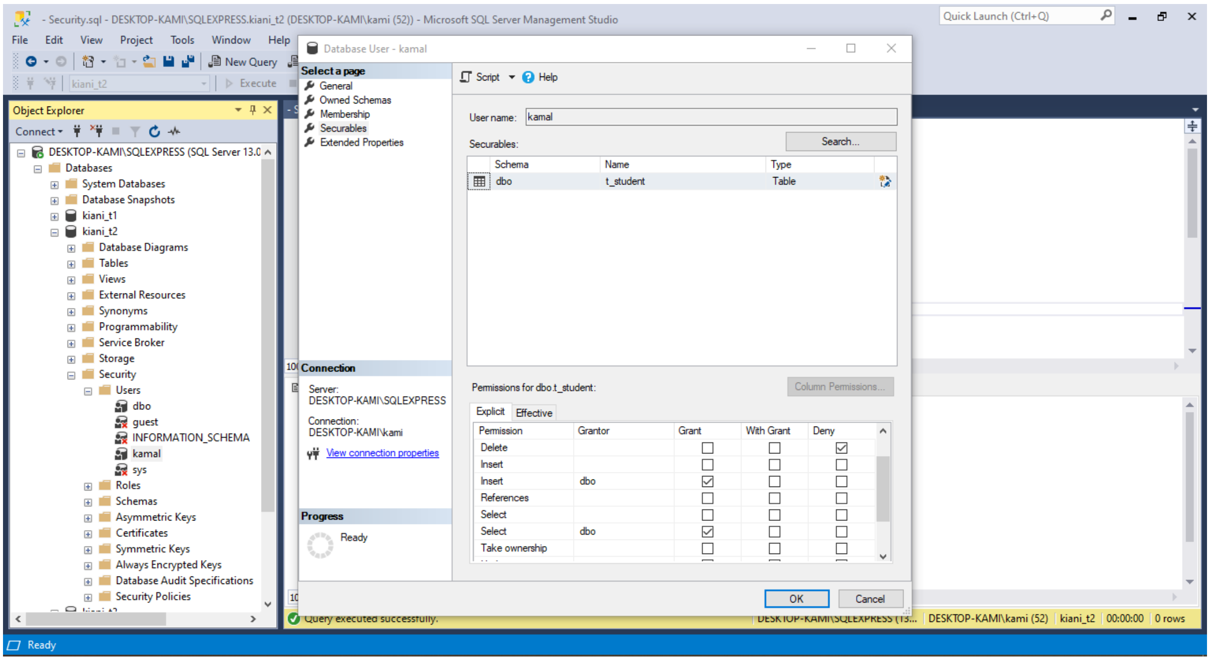


Figure2.20

2.8 Database Backup and Restore

Using the visual tools in SQL management studio or by using the SQL syntax, it is possible to get a full back up from the database and also restore the backup file whenever it is needed. Following figures2.21 and 2.22 show the SQL syntax for backup and restore operation in task2 database.

The SQL server visual tool to take backup or restore the database are shown previous part (task1) and because it is the same visual window, it is not included in this section and can be referred to previous sections.

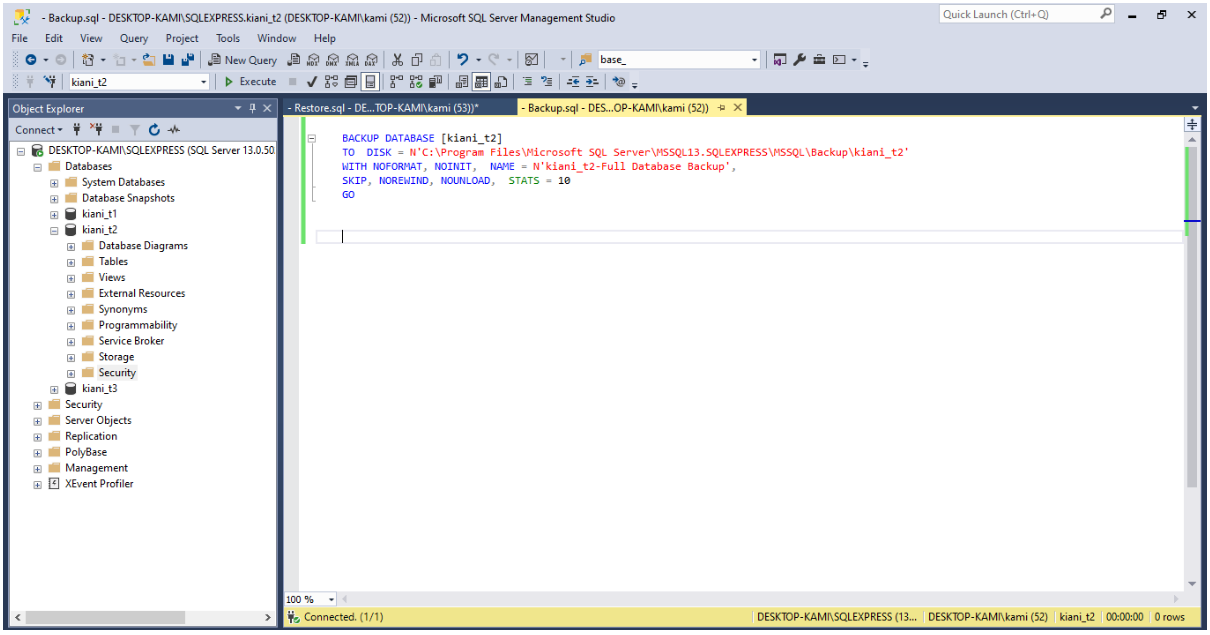


Figure2.21

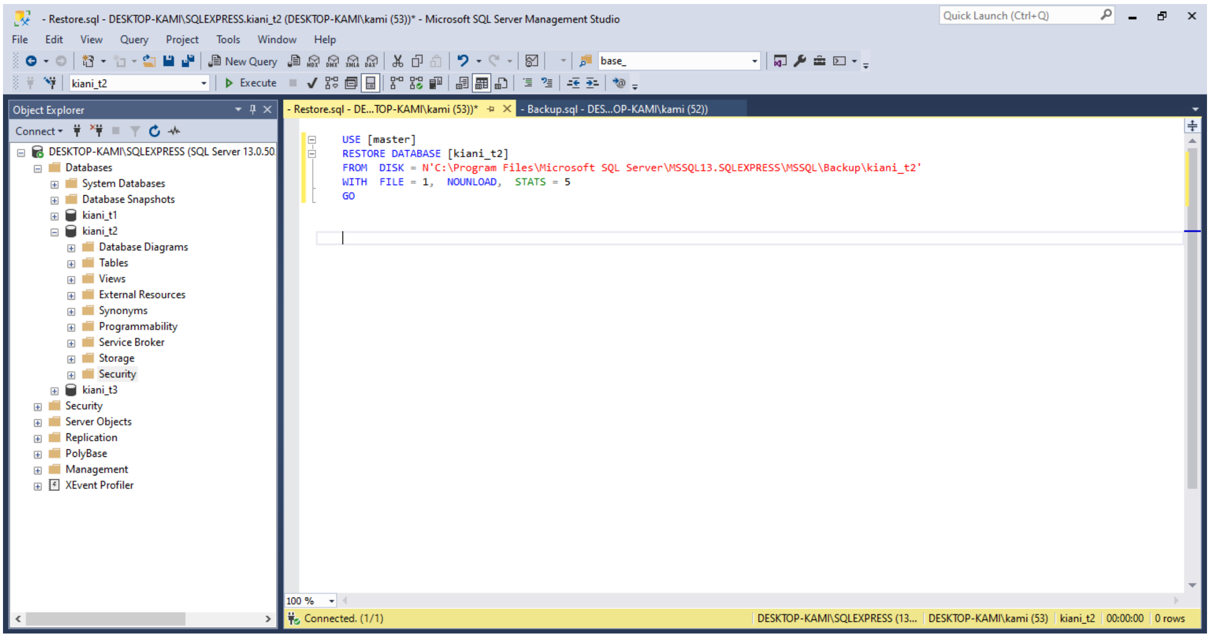


Figure2.22

2.9 Data Privacy, Ethical and Legal Issues

In this task we used a dataset which there are some information that are personal and sensitive. According to "Data Privacy, Ethical and Legal" rules protecting the personal information is vital and we can use these data just for the scientific and analytical purposes.

In the engineering level, we must make sure that there is no illegal access to the data. This can be made by defining security rules and plans.

In the analytical level, we must make sure that we do not focus on finding any personal information which leads us to find a clear individual person.

In the gathering information and preparing dataset level, we must make sure that sensitive data are not included before publishing. Due to the dataset used in this task, is downloaded from a legal source "**UK Data Services**" and according to detailed information they have provided in their website, all necessary protection rules and acts are applied on this dataset to ensure about the privacy and information protection against any harmful purposes.

As some examples from this dataset, we can see that all the values are coded, there is no name, contact information or exact location for students and schools.

2.10 Conclusion

In the task2, the "Education inequality in Vietnam" was studied with SQL server express as the analysis tool. The dataset used for this task was "Young Lives: School Survey, Vietnam" which was a survey on the level of schooling accessed by 15-year-olds, so including Grade 10 students in Vietnam. 3 different views and queries were done by the researcher to discover detailed information about this dataset and report the findings. As discussed in report section of this part, it seems some ENTHICITIES and PROVIENCIES need more focus to prepare study places, educational facilities, and private classes for students. Also, there should be some plans to motivate students specially "male" students to use computer at home for their homeworks.

Task 3: Greater Manchester Crime Profiler

3.1 Abstract

In this task, the "Greater Manchester Crime Profiler" will be done with SQL server, Excel and QGIS as the analysis and visualization tools. The dataset used for this task is the "Greater Manchester Crimes" in years between 2017 and 2018 available in https://data.police.uk. In this task we are going to build a report containing Lower Layer Super Output Areas (LSOAs) wise crime, with local population data in Greater Manchester between Jan 2017 and Dec 2018. In the following sections of this part the detailed characteristics of this dataset will be discussed and different reports will be generated based on this data.

3.2 Introduction

In this task, we aim to build reports containing Lower Layer Super Output Areas (LSOAs) wise crime, with local population data in Greater Manchester between Jan 2017 and Dec 2018. To do this, SQL server, QGIS, and Microsoft Excel will be used by the researcher. SQL server express is a free edition DBMS provided by Microsoft team, to store and manage data in a relational model and provides a powerful querying language. In this study SQL server express edition v.13 (2016) and SQL management studio version v.18 are used to complete the task3.

The version of QGIS used by the researcher is 3.12. After connecting to SQL server, this tool provides data and map layers for us to find the exact location of crimes. To complete task3, in coming sections a database including 4 tables will be created and SQL query language will be used to generate queries in SQL server environment.

3.3 Design Rationale

This database is designed with 4 tables as follows:

a) "**Manchester_Crimes**": Holds the records of crimes in greater Manchester between years 2017 and 2018.
b) "**Geo_Points**": Holds geometrical points of each crime record.
c) "**LSOA2018_Code**": The population of each LSOA code in 2018.
d) "**LSOA2018_District**": The population of each LSOA district in 2018.

Figure3.1 shows a view of these 4 tables in SQL server.

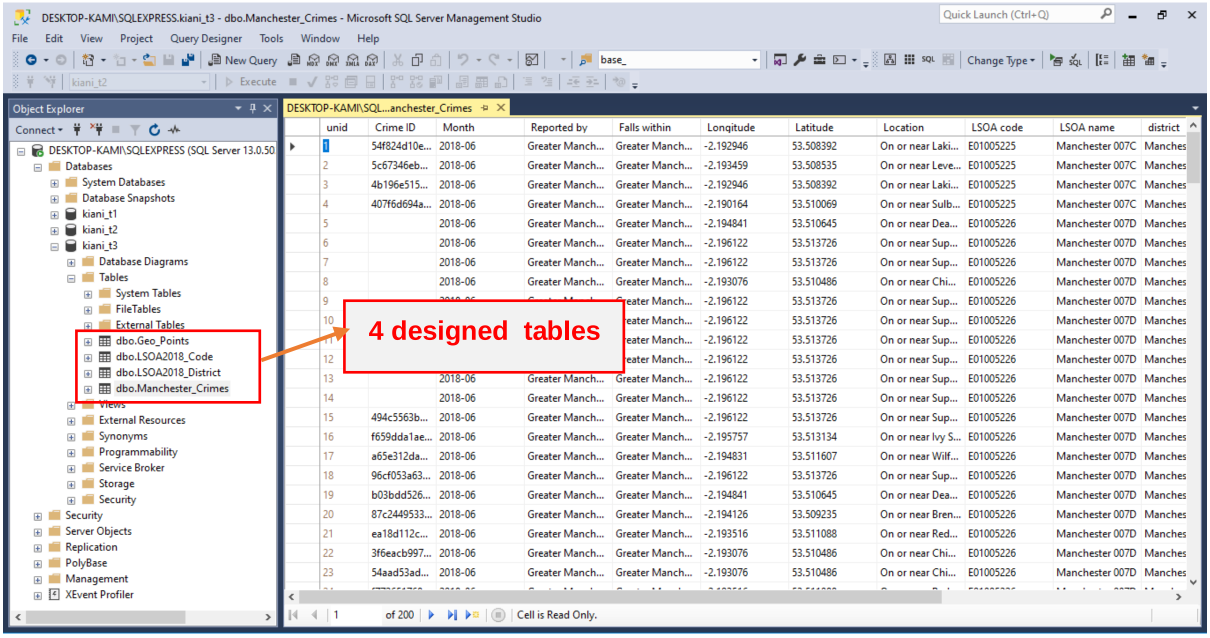


Figure3.1

The primary key for " Manchester_Crimes " and " Geo_Points " tables is "unid" which indicates each committed crime individually. As the relationship between these 2 tables is (1:1) and in the other hand, we do not want to define any cascading rules on delete and update, there is no need to explicitly define a foreign key between them and we can join these two tables as necessary. For the "LSOA2018_District" and "LSOA2018_Code" tables we have defined primary keys that are visible in figure3.2. In this database diagram, we can find that, there are two foreign keys defined between foreign-key table "Manchester_Crimes" and two LSOA tables as primary-key tables. Figures 3.3 and 3.4 indicate the foreign key definitions between tables.

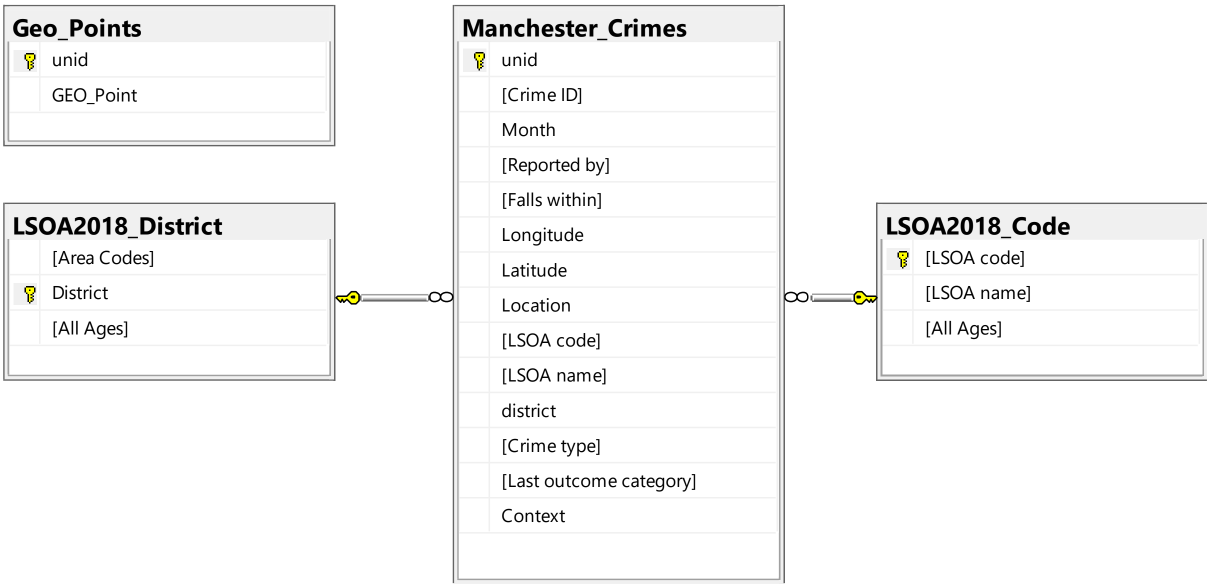


Figure3.2

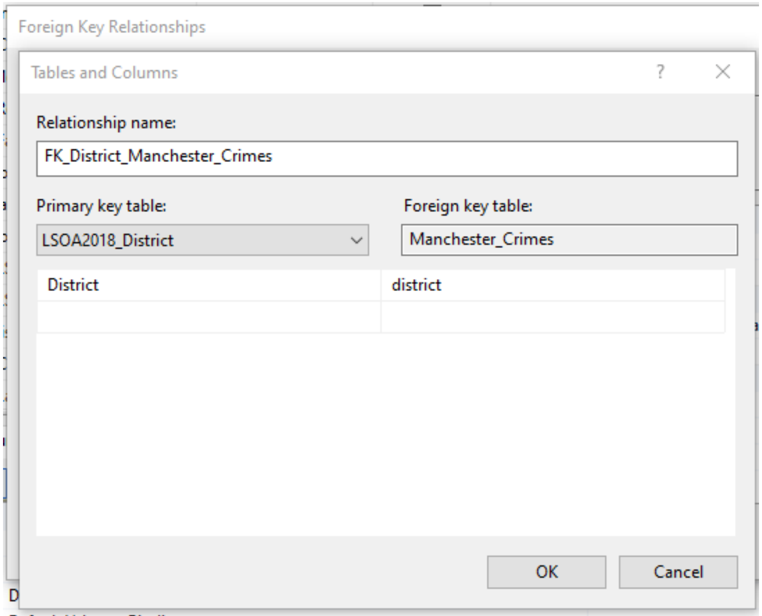


Figure3.3

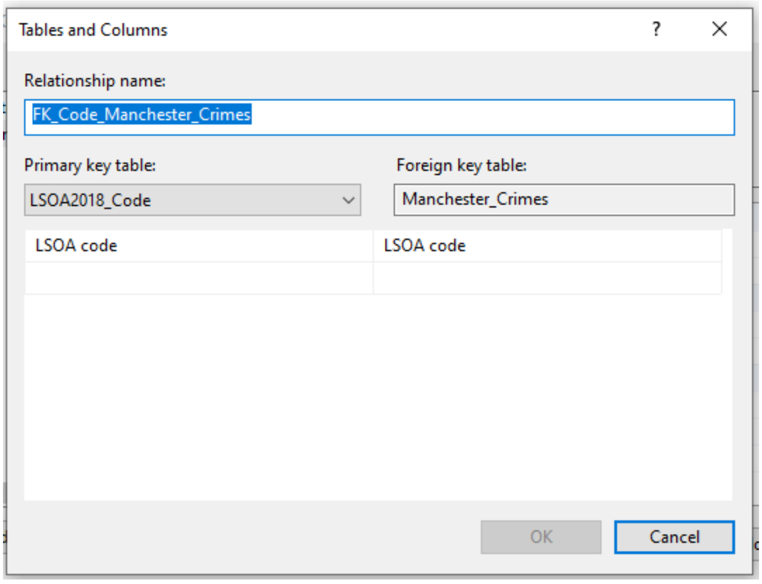


Figure3.4

3.4 Design Considerations

3.4.1 Constraints and Data Validation

- ✓ Primary key: 'unid' is the primary key in "Manchester_Crimes"
- ✓ Primary key: 'unid' is the primary key in "GEO_Points"
- ✓ Primary key: "LSOA Code" is the primary key in "LSOA2018_code"
- ✓ Primary key: "district" is the primary key in "LSOA2018_district"
- ✓ Required foreign keys are defined between tables
- ✓ Data types are defined based on variable types

3.4.2 Transaction and Concurrency Control

As discussed before in section1.5.3 about the Transaction concurrency and isolations, the detailed information about this topic is not included in this section and therefore can be referred to section1.5.3.

It is remarkeable that in our study, the used database is not a transactional database and it is designed and created for study porspuse. Thus, there is no need to use the isolation techniques for queries. If case it is needed, we can do so by a code like the following:

```
-- ensure we use SQL Server default isolation level (or we can change it!)
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;
    .
    .
    .
COMMIT TRANSACTION;
```

3.4.3 Error Handling

The "Try-Catch" block is used when necessary to ensure we can handle any error. The syntax used is similar to the following code:

```
BEGIN TRY
    --code to try
END TRY
BEGIN CATCH
    --code to run if an error occurs
    --is generated in try
END CATCH
```

Figure3.5 shows a sample script for error handling using Try-Catch block in task3 database while creating "GEO_Points" table.

```
BEGIN TRY
    DROP TABLE IF EXISTS Geo_Points;
    create table Geo_Points (
        unid bigint,
        GEO_Point geography,
        PRIMARY KEY (unid)
    );
    INSERT INTO Geo_Points (unid,GEO_Point)
        SELECT      unid, geography::Point(Latitude, Longitude, 43.
        FROM        dbo.Manchester_Crimes
        WHERE       (Latitude <> 0) AND (Longitude <> 0)
END TRY
BEGIN CATCH
      SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_STATE() AS ErrorState,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

Try Block

Catch block

Figure3.5

3.4.4 Comments

Comments are used in the scripts when it was necessary to clearly define the syntax. Figure3.6 is a sample of commenting in the code while creating a view in task3.

```sql
/*
    this code creates a view and finds the top 10
    commiting crimes locations in Salford along
    with the population of each location
*/
CREATE or ALTER view View_Salford AS
    select TOP 10 Manchester_Crimes.[LSOA name],
    count(*) as crimes_cnt,
    avg([All Ages]) as population
    from Manchester_Crimes inner join LSOA2018_Code
    on Manchester_Crimes.[LSOA code] = LSOA2018_Code.[LSOA code]
    where Manchester_Crimes.[LSOA name] like '%Salford%'
    group by Manchester_Crimes.[LSOA name]
    order by crimes_cnt DESC
```

Figure3.6

3.5 T-SQL Statements

3.5.1 Tables

As mentioned, 4 tables are defined in task3 database. The code below creates a table called "GEO_Points" which converts the Latitude and Longitude to a GEO POINT with the geography data type. This table will be used by QGIS software to show the location of each crime.

```sql
BEGIN TRY
    DROP TABLE IF EXISTS Geo_Points;
    create table Geo_Points (
        unid bigint,
        GEO_Point geography,
        PRIMARY KEY (unid)
    );
    INSERT INTO Geo_Points (unid,GEO_Point)
        SELECT      unid, geography::Point(Latitude, Longitude, 4326)
        FROM        dbo.Manchester_Crimes
        WHERE       (Latitude <> 0) AND (Longitude <> 0)
END TRY
BEGIN CATCH
      SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_STATE() AS ErrorState,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

Figure3.7

Other 3 tables are loaded and created directly via import data menu from raw dataset files into SQL server. Figure3.8 shows the menu and window used in SQL server for importing and creating these tables from a "FlatFile".

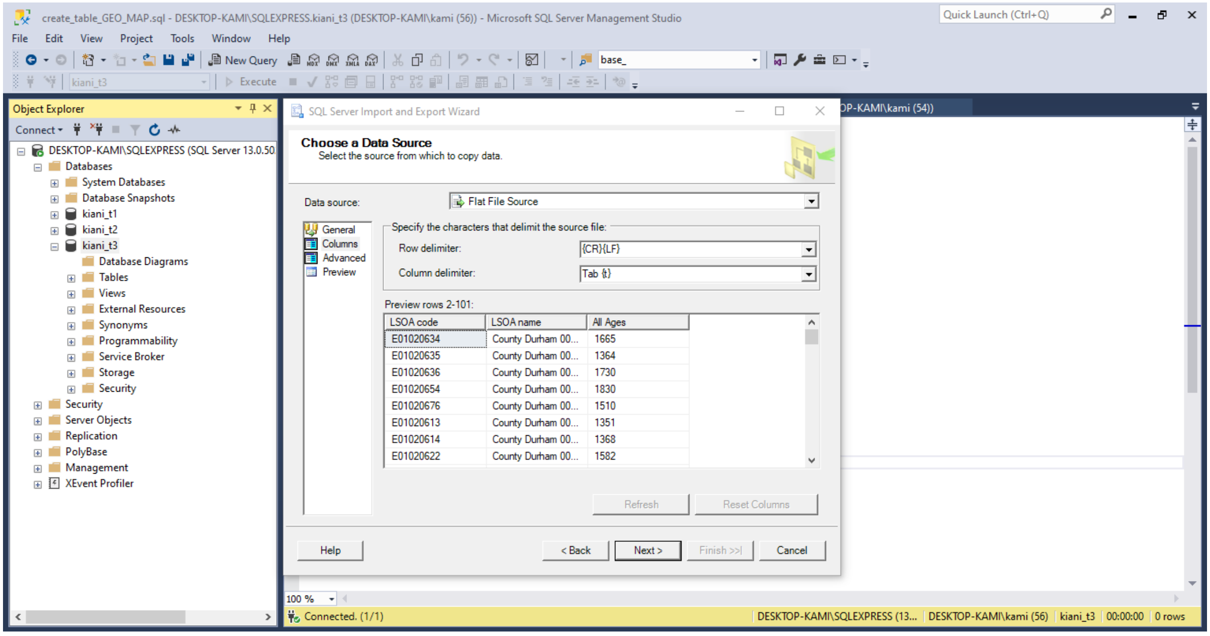


Figure3.8

3.5.2 Views

In this task 5 different views are generated. 2 of which are for using in the QGIS and 3 of them for exporting to Excel for plotting diagrams.

The first view is "View_Salford", which finds the top 10 committing crimes locations in Salford along with the population of each location. In this query two tables are joined and the data is grouped by LSOA names. The code for this view is shown in figure3.9 and figure3.10 is the output.

```
/*
    this code creates a view and finds the top 10
    commiting crimes locations in Salford along
    with the population of each location
*/
CREATE or ALTER view View_Salford AS
    select TOP 10 Manchester_Crimes.[LSOA name],
    count(*) as crimes_cnt,
    avg([All Ages]) as population
    from Manchester_Crimes inner join LSOA2018_Code
    on Manchester_Crimes.[LSOA code] = LSOA2018_Code.[LSOA code]
    where Manchester_Crimes.[LSOA name] like '%Salford%'
    group by Manchester_Crimes.[LSOA name]
    order by crimes_cnt DESC
```

Figure3.9

	LSOA name	crimes_cnt	population
1	Salford 017B	1606	1516
2	Salford 002D	1349	1584
3	Salford 016B	1344	2172
4	Salford 020A	1295	1568
5	Salford 026A	1268	1636
6	Salford 021B	1171	1512
7	Salford 024D	1144	1641
8	Salford 017C	1125	1744
9	Salford 017F	1108	2814
10	Salford 024C	984	1904

Figure3.10

The second view is named "View_Dsitricts" which counts the crimes in each district of greater Manchester along with the population of each district as shown in figure3.11 and figure3.12.

```
/*
    this code creates a view and counts the crimes
    in each district of greater manchester along
    with the population of each district
*/
CREATE or ALTER view View_Dsitricts AS
    select  Manchester_Crimes.district, count(*) as crimes_cnt,
            LSOA2018_District.[All Ages] as population
    from Manchester_Crimes inner join LSOA2018_District
    on Manchester_Crimes.district = LSOA2018_District.district
    group by Manchester_Crimes.district , LSOA2018_District.[All Ages]
```

Figure3.11

	district	crimes_cnt	population
1	High Peak	1	92221
2	calderdale	5	210082
3	Chorley	5	116821
4	West Lancashire	13	113949
5	St. Helens	22	180049
6	Rossendale	46	70895
7	Blackburn with Darwen	46	148942
8	Warrington	78	209547
9	Cheshire East	93	380790
10	Trafford	44096	236370
11	Bury	45615	190108
12	Stockport	61900	291775
13	Tameside	62302	225197
14	Rochdale	63762	220001
15	Salford	69470	254408
16	Oldham	70822	235623
17	Wigan	73998	326088
18	Bolton	82047	285372
19	Manchester	230734	547627

Figure3.12

The third created view is "View_Month", to count the Greater Manchester crimes, in each month from 2017 to 2018. Figure3.13 shows that "WITH" statement is used here to split the month of each crime by calling "substring()" function in SQL. Then, from the Q1 query, we can count the number of crimes in each month. Figure3.14 is the output after running the query.

```
/*
    this code creates a view and counts the Greater Manchester
    crimes, in each month from 2017 to 2018
*/
CREATE or ALTER view View_Month AS
    with Q1 as(
        select unid, SUBSTRING(Month,6,2) as crime_month
        from Manchester_Crimes
    )
    select crime_month,
        CASE
            WHEN crime_month =  '01' THEN 'Jan'
            WHEN crime_month =  '02' THEN 'Feb'
            WHEN crime_month =  '03' THEN 'Mar'
            WHEN crime_month =  '04' THEN 'Apr'
            WHEN crime_month =  '05' THEN 'May'
            WHEN crime_month =  '06' THEN 'Jun'
            WHEN crime_month =  '07' THEN 'Jul'
            WHEN crime_month =  '08' THEN 'Aug'
            WHEN crime_month =  '09' THEN 'Sep'
            WHEN crime_month =  '10' THEN 'Oct'
            WHEN crime_month =  '11' THEN 'Nov'
            WHEN crime_month =  '12' THEN 'Dec'
        END AS month_name,
    count(unid) as crimes_cnt
    from Q1
    group By crime_month
```

Figure3.13

Results | Messages

	crime_month	month_name	crimes_cnt
1	01	Jan	63444
2	02	Feb	60393
3	03	Mar	68528
4	04	Apr	68998
5	05	May	71368
6	06	Jun	70783
7	07	Jul	72813
8	08	Aug	67842
9	09	Sep	63724
10	10	Oct	68828
11	11	Nov	66724
12	12	Dec	61610

Figure3.14

Another view created in this task is the "QGIS_VehicleCrimes" which is the vehicle crimes in Manchester and their GEO location. This view will be used in QGIS to map the location of crimes. In figure3.15 the code for creating this view is illustrated.

```sql
/*
    Vehicle crimes in manchester and related GEO location
*/

CREATE or ALTER VIEW QGIS_VehicleCrimes as
    select Manchester_Crimes.unid, [Crime type], Latitude, Longitude, GEO_Point
    from Manchester_Crimes inner join Geo_Points
    on Manchester_Crimes.unid = Geo_Points.unid
    where [Crime type]='Vehicle crime'
```

Figure3.15

And finally, the last created view "QGIS_AntiSocialCrimes" is indicated in figure3.16 which finds the anti-social behavior crimes in Salford with the related GEO location of crimes. This view will be used in QGIS to map the location of crimes, as well.

```sql
/*
    Anti-social behaviour crimes in Salford and related GEO location
*/

CREATE or ALTER VIEW QGIS_AntiSocialCrimes as
    select Manchester_Crimes.unid, [Crime type], district, Latitude, Longitude, GEO_Point
    from Manchester_Crimes inner join Geo_Points
    on Manchester_Crimes.unid = Geo_Points.unid
    where [Crime type]= 'Anti-social behaviour' and district LIKE '%Salford%'
```

Figure3.16

3.6 Report Design

The two ways of exporting data to Microsoft Excel were discussed in section 2.6. After importing the values of views to Excel worksheets, some different diagrams were plotted that can be seen in following figures.

Figure3.17 indicates that the most crimes committed in "Jul" while the least number of crimes is in "Feb". Figure3.18 indicates that the most crimes were committed in "017B" LSOA code and figure3.19 shows the number of crimes in each district and in comparison with the population of that district.

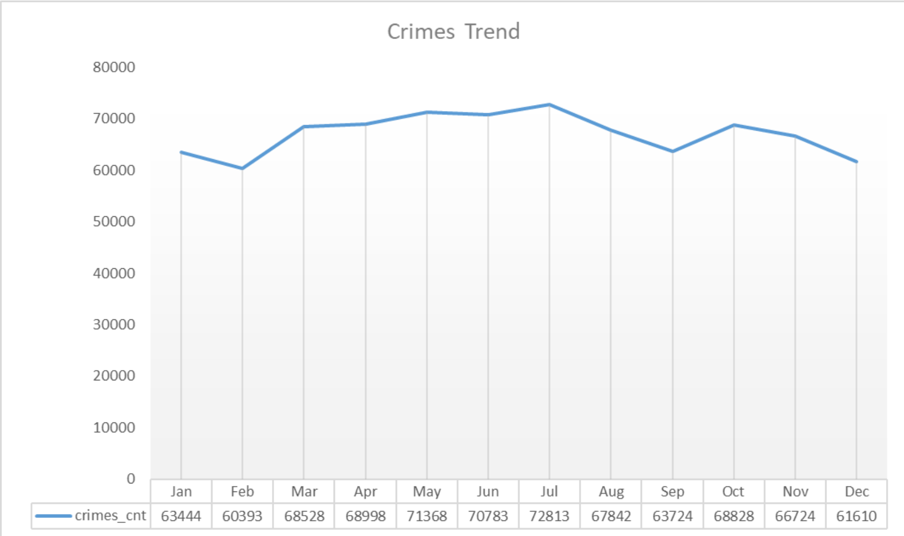


Figure3.17

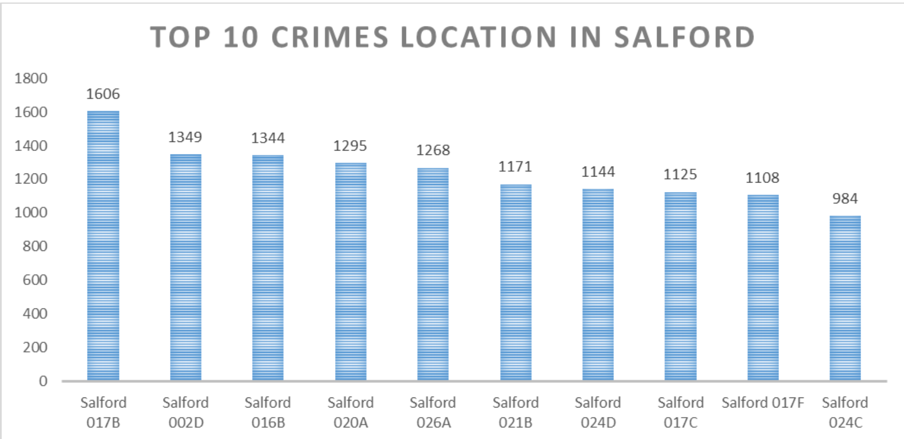


Figure3.18

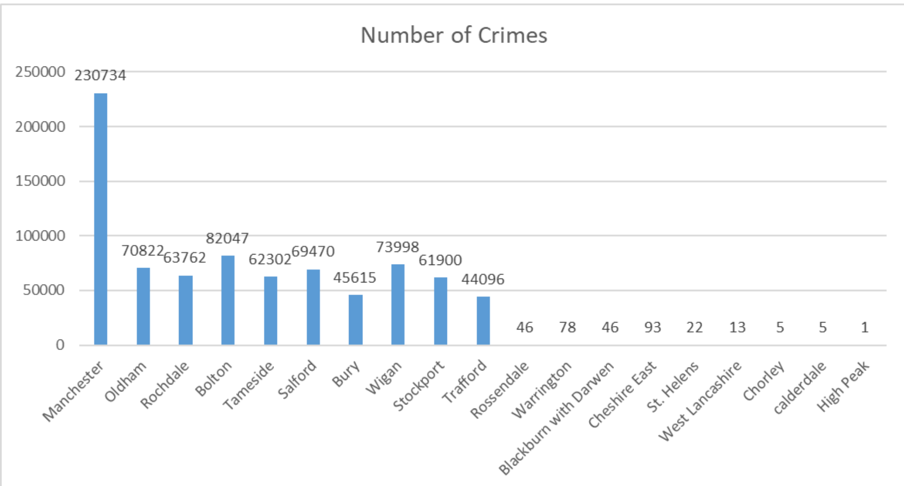


Figure3.19-a

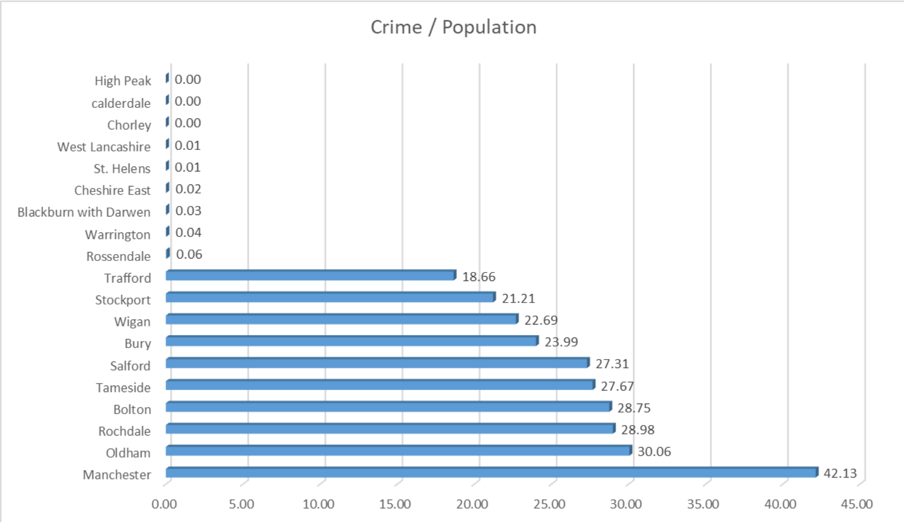


Figure3.19-b

In this part QGIS is used to map the GEO locations of crimes. Figure3.20 shows the data source connection between QGIS and SQL server in the QGIS software. After connecting the two "QGIS_VehicleCrimes" and "QGIS_AntiSocialCrimes" views to the QGIS and importing the location based data as new layers, the outputs are generated on the map layer. Figure3.21(a) is the output of Manchester vehicle crimes on the map and figure3.21(b) is the output of mapping Salford anti-social crimes.

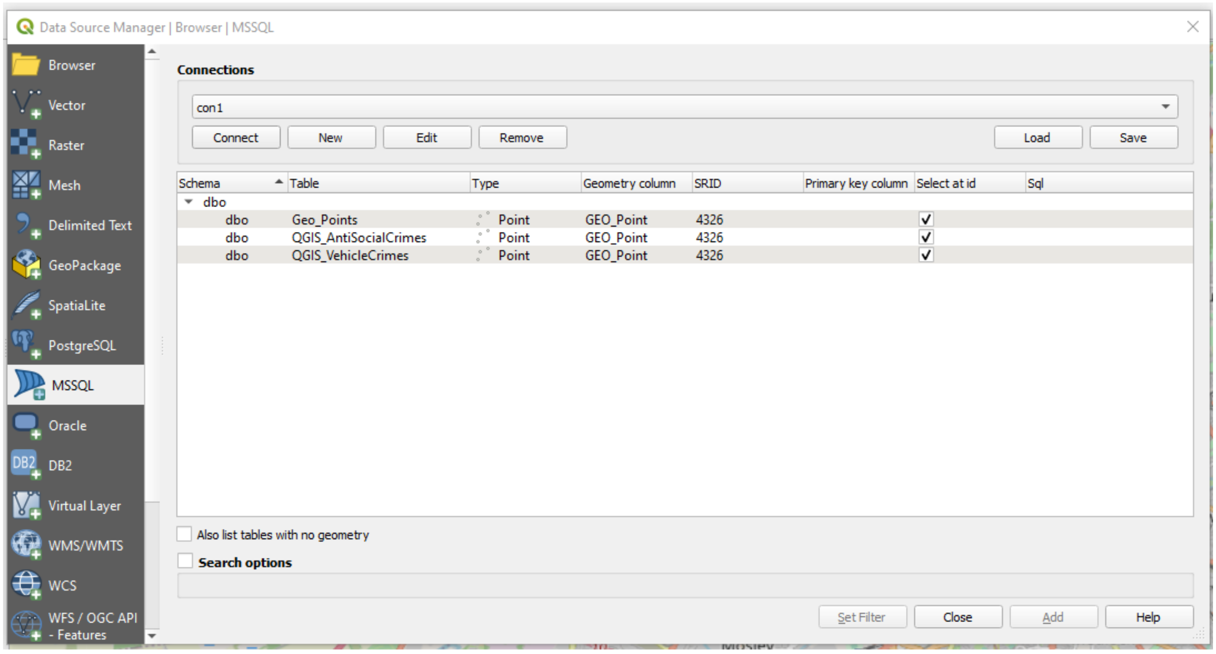


Figure3.20

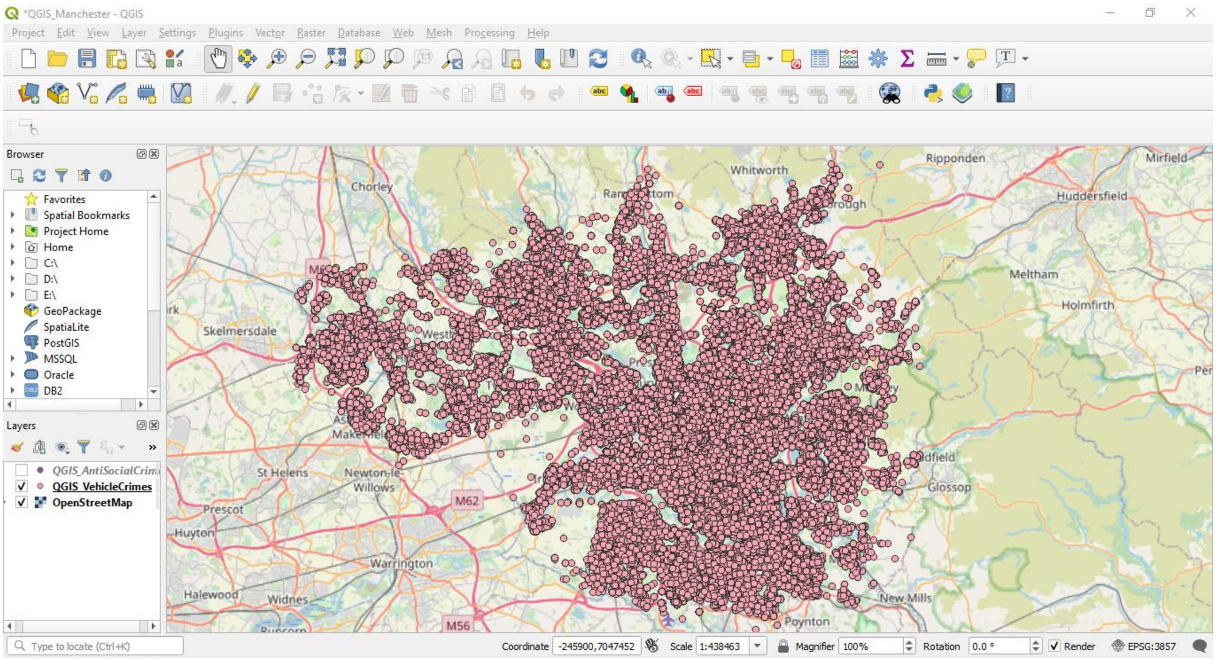


Figure3.21(a)

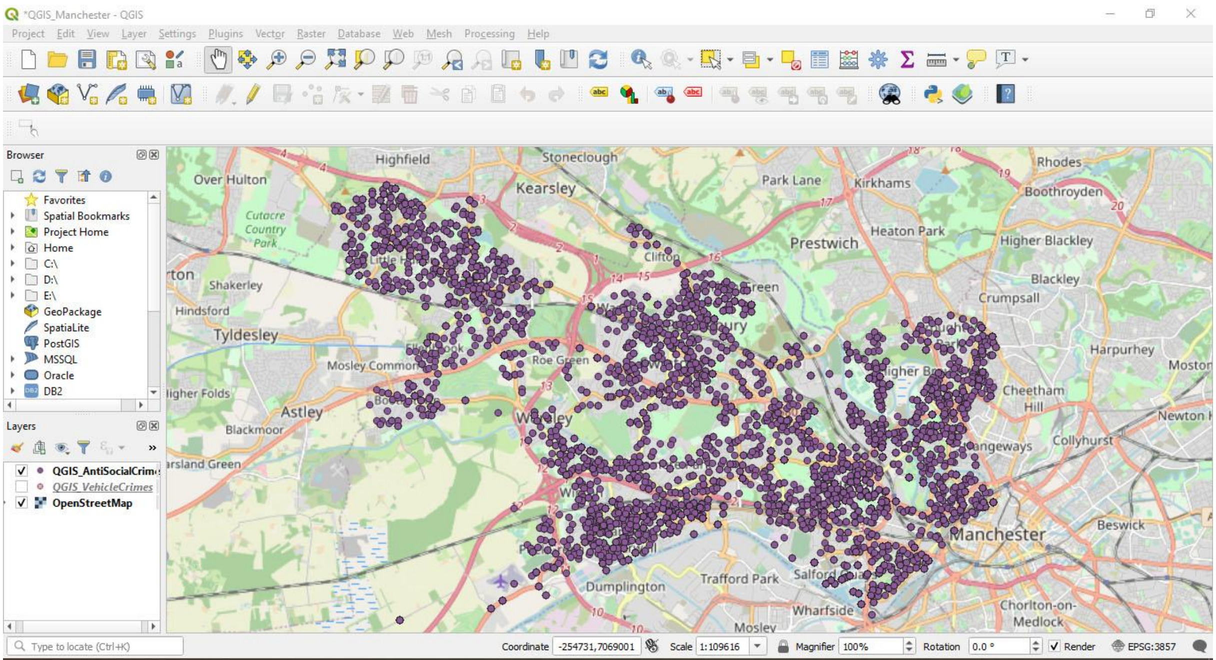


Figure3.21(b)

3.7 Database Security

As the database security plan, we can define users and control the permissions for them. In this task, a user named "kamal" is defined and for a sample permission control, "insert" and "select" are granted for this user while "update" and "delete" are not allowed for the user in table "Manchester_Crimes". Figure3.22 shows the script for applying this control.

```sql
IF  EXISTS (SELECT * FROM sys.database_principals WHERE name = N'kamal') DROP USER kamal

 /* new user */
CREATE USER kamal
FOR LOGIN myNewLogin;
GO

/* allowed for user */
GRANT SELECT, INSERT
ON Manchester_Crimes
TO kamal;

/* not allowed for user */
REVOKE DELETE, UPDATE
ON Manchester_Crimes
FROM kamal;
```

Figure3.22

3.8 Database Backup and Restore

Using the visual tools in SQL management studio or by using the SQL syntax, it is possible to get a full back up from the database and also restore the backup file whenever it is needed. Following figures3.23 and 3.24 show the SQL syntax for backup and restore operation in task3 database.

The SQL server visual tool to take backup or restore the database are shown previous part (task1) and because it is the same visual window, it is not included in this section and can be referred to previous sections.

```sql
BACKUP DATABASE [kiani_t3]
TO  DISK = N'C:\Program Files\Microsoft SQL Server\MSSQL13.SQLEXPRESS\MSSQL\Backup\kiani_t3'
WITH NOFORMAT, NOINIT,  NAME = N'kiani_t3-Full Database Backup',
SKIP, NOREWIND, NOUNLOAD,  STATS = 10
GO
```

Figure3.23

```sql
USE [master]
RESTORE DATABASE [kiani_t3]
FROM  DISK = N'C:\Program Files\Microsoft SQL Server\MSSQL13.SQLEXPRESS\MSSQL\Backup\kiani_t3'
WITH  FILE = 1,  NOUNLOAD,  STATS = 5
GO
```

Figure3.24

3.9 Data Privacy, Ethical and Legal Issues

In this task we used a dataset which there are some information that are personal and sensitive. According to "Data Privacy, Ethical and Legal" rules protecting the personal information is vital and we can use these data just for the scientific and analytical purposes.

In the engineering level, we must make sure that there is no illegal access to the data. This can be made by defining security rules and plans.

In the analytical level, we must make sure that we do not focus on finding any personal information which leads us to find a clear individual person.

In the gathering information and preparing dataset level, we must make sure that sensitive data are not included before publishing. Due to the dataset used in this task, is downloaded from a legal source "**UK Data Services**" and according to detailed information they have provided in their website, all necessary protection rules and acts are applied on this dataset to ensure about the privacy and information protection against any harmful purposes.

As some examples from this dataset, we can see that all the values are coded, there is no name, contact information or exact location for students and schools.

3.10 Conclusion

In this task, the "Greater Manchester Crime Profiler" was made with SQL server, Excel and QGIS tools. The dataset used for this task was the "Greater Manchester Crimes" in years between 2017 and 2018 and the report were built containing Lower Layer Super Output Areas (LSOAs) wise crime, with local population data in Greater Manchester. Results in section 3.6 show the crimes statistics and location in Manchester during study years.

References

Boyden, J. (2021). *Young Lives: an International Study of Childhood Poverty.* UK Data Service.

docs.microsoft.com. (2021). Retrieved from https://docs.microsoft.com/en-us/sql/connect/jdbc/understanding-isolation-levels?view=sql-server-ver15

Jayaram, P. (2018). *sqlshack.com*. Retrieved from https://www.sqlshack.com/what-is-database-normalization-in-sql-server/

Petrovic, B. (2018). *sqlshack-error-handling*. Retrieved from https://www.sqlshack.com/how-to-implement-error-handling-in-sql-server/

University of Oxford, D. o. (2018). *Young Lives: School Survey, Vietnam, 2016-2017.* UK Data Service .

wikipedia. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Database_normalization