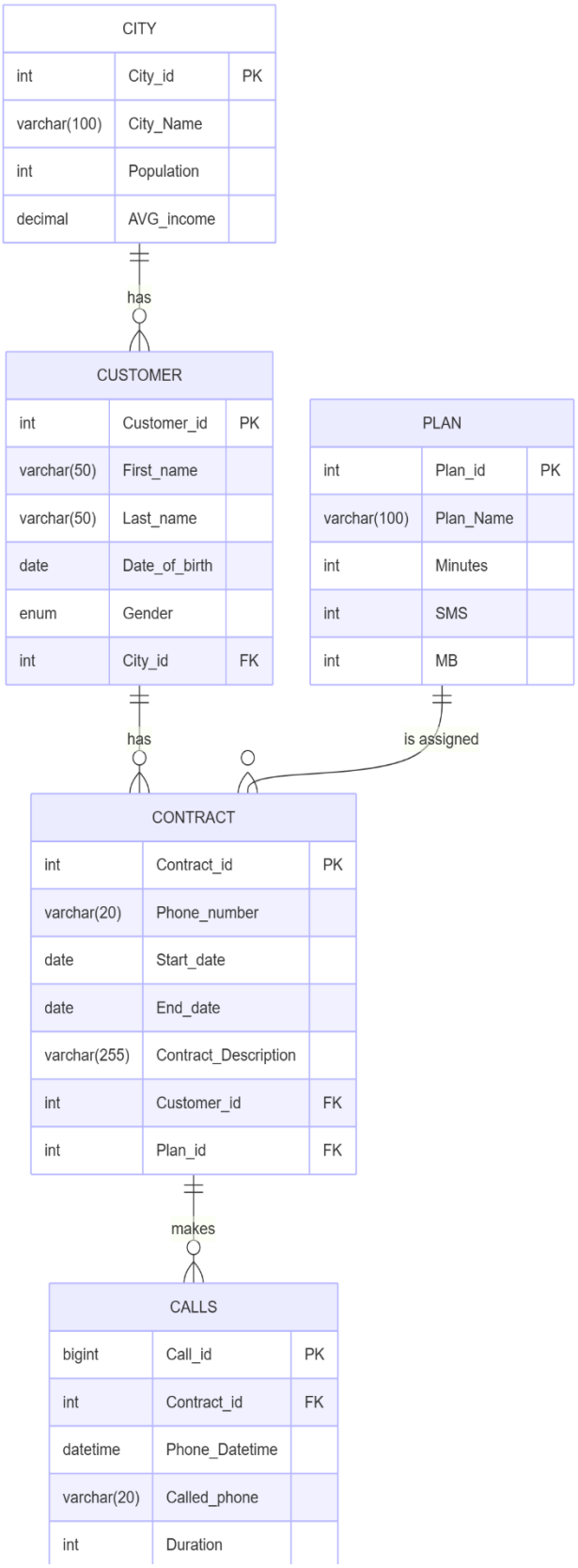
FUNDAMENTALS IN DATA MANAGEMENT

Assignment #1

T******* D*******
F**********

1. **(10%) Use the Entity-Relationship Diagram (ERD) to model entities, relationships, attributes, cardinalities, and all necessary constraints. Use any tool you like to draw the ERD.**



1. Entity-Relationship Diagram (ERD) - MermaidChart

This schema models a telecom setting where **Customers** living in **Cities** sign **Contracts** for mobile service **Plans** and then make **Calls** from those contracts

Tools:

i. Mermaid Chart
ii. Smartdraw

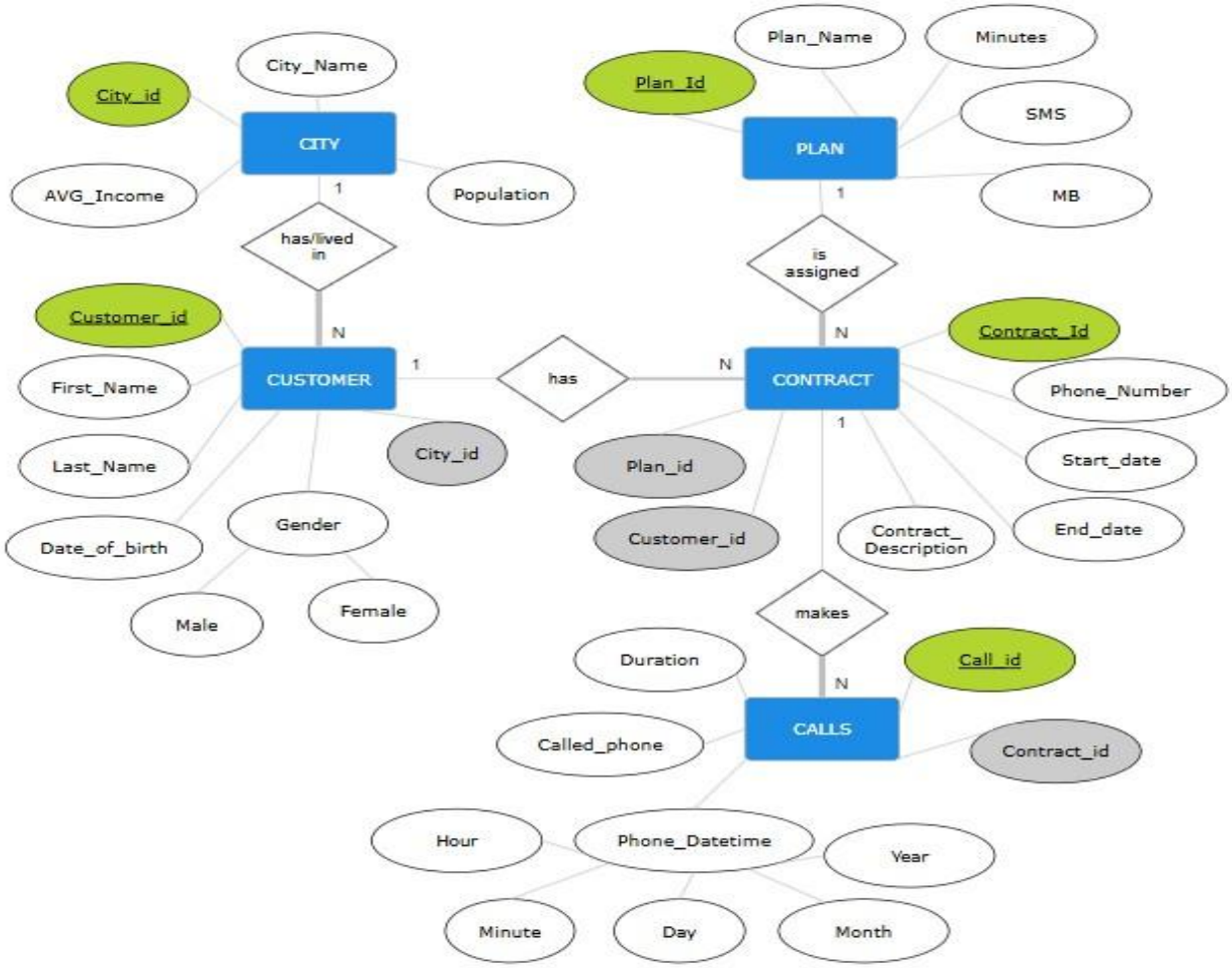
Relationships

- **CITY 1 ——< CUSTOMER**
One city can have many customers.
Each customer lives in one city.

- **CUSTOMER 1 ——< CONTRACT**
A customer can hold many contracts.
Each contract belongs to one customer.

- **PLAN 1 ——< CONTRACT**
A plan can be assigned to many contracts.
Each contract uses exactly one plan.

- **CONTRACT 1 ——< CALLS**
A contract can generate many calls.
Each call is made by exactly one contract.



2. Entity-Relationship Diagram (ERD) - Smartdraw

2. **(10%) Create the relational schema in MySQL/SQLServer and insert a few records into the tables to test your queries below. You will have to hand in the CREATE TABLE statements.**

The model consists of five tables using InnoDB and utf8mb4 for full Unicode support:

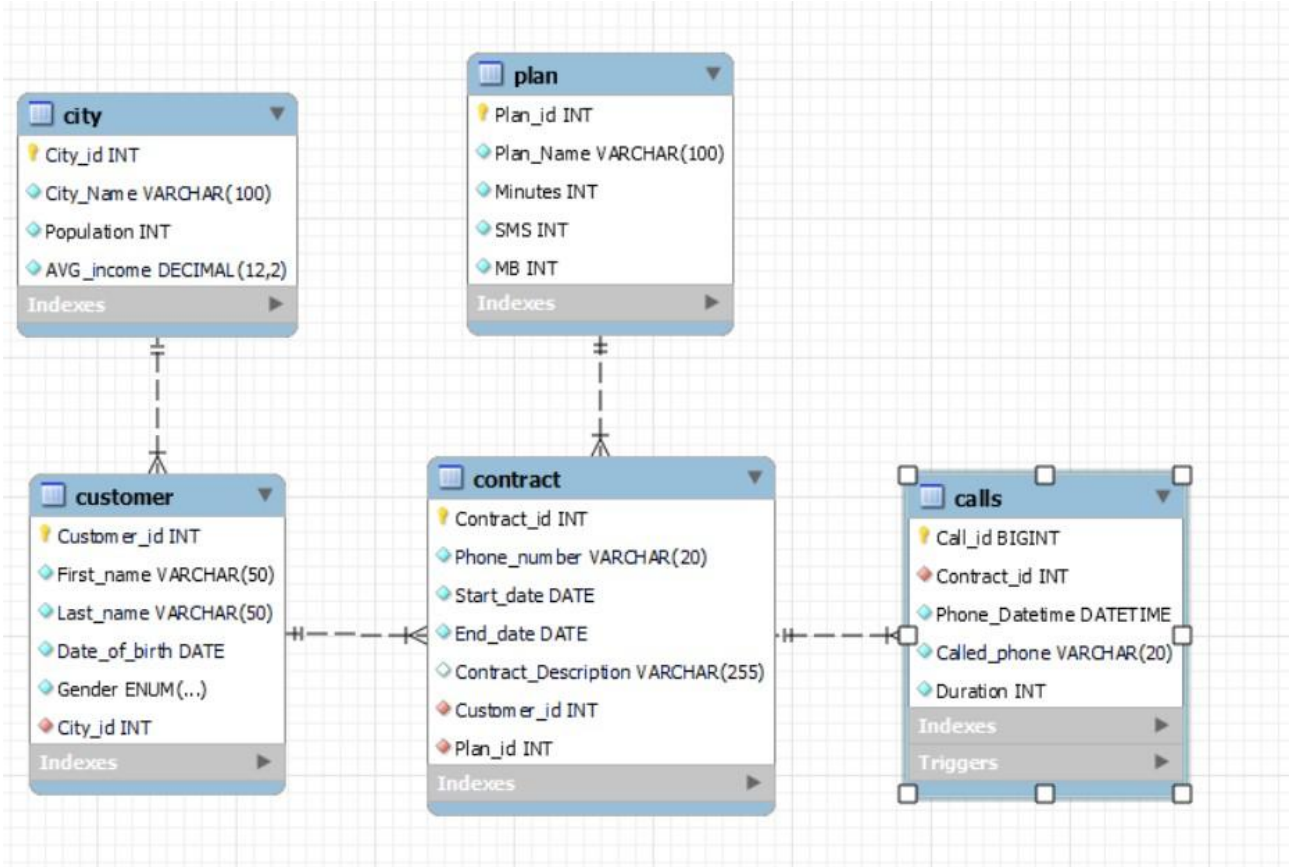
- **CITY**: reference data for where customers live (City_Name unique, Population > 0, AVG_income ≥ 0).
- **CUSTOMER**: people with basic demographics and a mandatory link to a city (Gender constrained by ENUM).
- **PLAN**: catalog of offerings with monthly allowances (Minutes, SMS, MB all ≥ 0).
- **CONTRACT**: each mobile line/SIM with a unique Phone_number, active window (Start_date–End_date) and links to one customer and one plan (date sanity via CHECK End_date ≥ Start_date).

- **CALLS**: facts of individual calls (timestamp, called phone number, duration in seconds, Duration > 0) tied to the originating contract.

All identifiers are AUTO_INCREMENT primary keys, referential integrity is enforced with foreign keys. To keep common lookups fast, the script adds purposeful **indexes**: CONTRACT(End_date) for expiring lines, CONTRACT(Plan_id) for plan analytics, and CALLS(Contract_id, Phone_Datetime) plus CALLS(Called_phone, Phone_Datetime) for time-window and callee analyses.

Because cross-table CHECK constraints are not supported in MySQL, the script includes **two triggers** on CALLS that enforce the key business rule: a call's timestamp must fall **within its contract's active dates** (Start_date ≤ Phone_Datetime ≤ End_date) on both insert and update. This guarantees temporal consistency between usage and the governing contract.

Finally, the script loads a small but representative dataset (3 cities, 4 plans, 7 customers, 10 contracts and calls across 2021–2022).



3. EER Diagram

```
DROP SCHEMA IF EXISTS Assignment_1;
CREATE SCHEMA IF NOT EXISTS Assignment_1 DEFAULT CHARSET utf8mb4;
USE Assignment_1;
```

```
DROP TABLE IF EXISTS CALLS;
DROP TABLE IF EXISTS CONTRACT;
DROP TABLE IF EXISTS CUSTOMER;
DROP TABLE IF EXISTS PLAN;
DROP TABLE IF EXISTS CITY;

CREATE TABLE CITY (
City_id INT PRIMARY KEY AUTO_INCREMENT,
City_Name VARCHAR(100) NOT NULL UNIQUE,
Population INT NOT NULL CHECK (Population > 0),
AVG_income DECIMAL(12,2) NOT NULL CHECK (AVG_income >= 0)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE CUSTOMER (
Customer_id INT PRIMARY KEY AUTO_INCREMENT,
First_name VARCHAR(50) NOT NULL,
Last_name VARCHAR(50) NOT NULL,
Date_of_birth DATE NOT NULL,
Gender ENUM('Male','Female') NOT NULL,
City_id INT NOT NULL,
CONSTRAINT Fk_Customer_City
FOREIGN KEY (City_id) REFERENCES CITY(City_id)
ON UPDATE CASCADE ON DELETE RESTRICT
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE PLAN (
Plan_id INT PRIMARY KEY AUTO_INCREMENT,
Plan_Name VARCHAR(100) NOT NULL UNIQUE,
Minutes INT NOT NULL DEFAULT 0 CHECK (Minutes >= 0),
SMS INT NOT NULL DEFAULT 0 CHECK (SMS >= 0),
MB INT NOT NULL DEFAULT 0 CHECK (MB >= 0)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE CONTRACT (
Contract_id INT PRIMARY KEY AUTO_INCREMENT,
Phone_number VARCHAR(20) NOT NULL UNIQUE,
Start_date DATE NOT NULL,
End_date DATE NOT NULL,
Contract_Description VARCHAR(255),
Customer_id INT NOT NULL,
Plan_id INT NOT NULL,
CONSTRAINT chk_contract_dates CHECK (End_date >= Start_date),
CONSTRAINT Fk_Contract_Customer
FOREIGN KEY (Customer_id) REFERENCES CUSTOMER(Customer_id)
ON UPDATE CASCADE ON DELETE RESTRICT,
CONSTRAINT Fk_Contract_Plan
FOREIGN KEY (Plan_id) REFERENCES PLAN(Plan_id)
ON UPDATE CASCADE ON DELETE RESTRICT,
```

```
INDEX idx_contract_end_date (End_date),
INDEX idx_contract_plan (Plan_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE CALLS (
Call_id BIGINT PRIMARY KEY AUTO_INCREMENT,
Contract_id INT NOT NULL,
Phone_Datetime DATETIME NOT NULL,
Called_phone VARCHAR(20) NOT NULL,
Duration INT NOT NULL CHECK (Duration > 0),
CONSTRAINT Fk_Call_Contract
FOREIGN KEY (Contract_id) REFERENCES CONTRACT(Contract_id)
ON UPDATE CASCADE ON DELETE CASCADE,
INDEX idx_call_contract_datetime (Contract_id, Phone_Datetime),
INDEX idx_call_called_and_dt (Called_phone, Phone_Datetime)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

DELIMITER //
CREATE TRIGGER trg_calls_1
BEFORE INSERT ON CALLS
FOR EACH ROW
BEGIN
  DECLARE v_start DATE; DECLARE v_end DATE;
  SELECT Start_date, End_date INTO v_start, v_end
  FROM CONTRACT WHERE Contract_id = NEW.Contract_id;

  IF NEW.Phone_Datetime < v_start THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Call before contract Start_date';
  END IF;

  IF NEW.Phone_Datetime >= v_end + INTERVAL 1 DAY THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Call after contract End_date';
  END IF;
END//

CREATE TRIGGER trg_calls_2
BEFORE UPDATE ON CALLS
FOR EACH ROW
BEGIN
  DECLARE v_start DATE; DECLARE v_end DATE;
  SELECT Start_date, End_date INTO v_start, v_end
  FROM CONTRACT WHERE Contract_id = NEW.Contract_id;

  IF NEW.Phone_Datetime < v_start THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Call before contract Start_date';
  END IF;

  IF NEW.Phone_Datetime >= v_end + INTERVAL 1 DAY THEN
```

```
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Call after contract End_date';
  END IF;
END//
DELIMITER ;
```

```
INSERT INTO CITY (City_Name, Population, AVG_income) VALUES
('Athens',3000015,16000.00),
('Thessaloniki',1200000,14500.00),
('Patras',16446,12000.00);
```

```
INSERT INTO PLAN (Plan_Name, Minutes, SMS, MB) VALUES
('Red2',300,300,6000),
('Student',400,400,20000),
('Red1',150,150,3000),
('Freedom',1000,1000,50000);
```

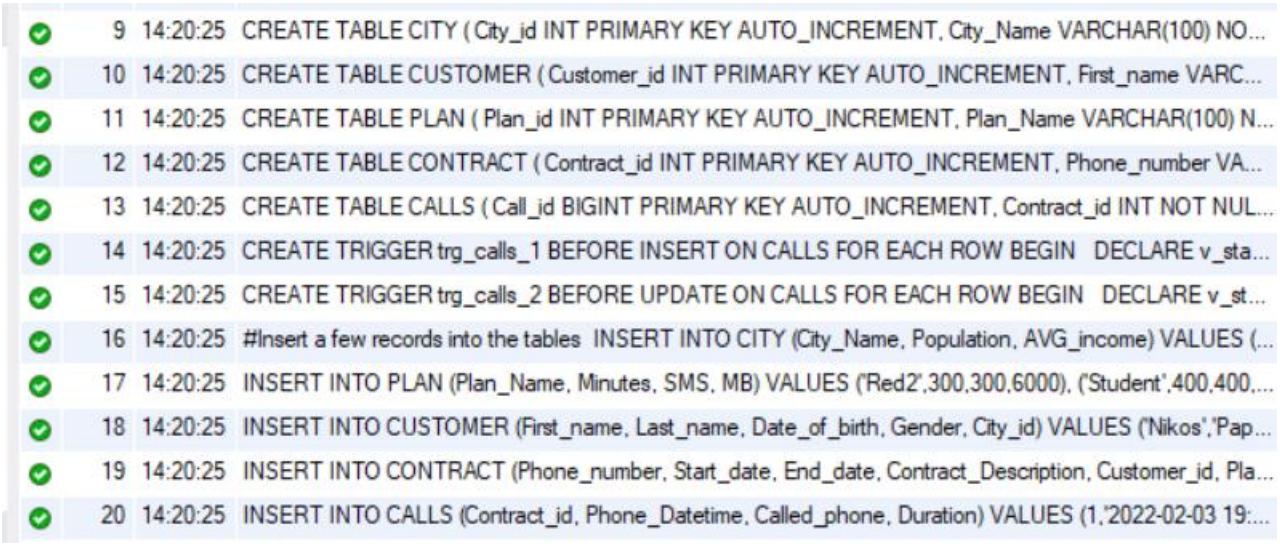
```
INSERT INTO CUSTOMER (First_name, Last_name, Date_of_birth, Gender, City_id) VALUES
('Nikos','Papadopoulos','1988-03-12','Male',1),
('Maria','Ioannou','1992-07-25','Female',2),
('Giorgos','Konstantinou','1985-11-05','Male',2),
('Eleni','Christou','1990-02-18','Female',1),
('Dimitrios','Oikonomou','1995-09-30','Male',1),
('Katerina','Georgiou','1987-01-14','Female',2),
('Panagiotis','Nikolaou','1993-04-22','Male',3);
```

```
INSERT INTO CONTRACT (Phone_number, Start_date, End_date, Contract_Description,
Customer_id, Plan_id) VALUES
('6940000001','2020-01-01','2025-11-20','Main line',1,1),
('6940000002','2019-05-15','2026-05-14','Student package',2,2),
('6940000003','2019-09-01','2025-12-10','Student package',3,2),
('6940000004','2020-02-10','2026-02-09','Main line',4,4),
('6940000005','2021-07-20','2025-12-25','Main line',5,4),
('6940000006','2020-11-11','2027-11-10','Main line',6,3),
('6940000007','2019-03-05','2026-03-04','Main line',7,1),
('6940000008','2020-12-01','2026-11-30','Student package',1,2),
('6940000009','2021-08-18','2026-08-17','Student package',2,2),
('6940000010','2020-06-22','2026-06-21','Main line',3,4);
```

```
INSERT INTO CALLS (Contract_id, Phone_Datetime, Called_phone, Duration) VALUES
(1,'2022-02-03 19:05:00','6940000009',4500),
(1,'2022-02-04 19:05:00','6940000009',5500),
(1,'2022-02-05 19:05:00','6940000009',1500),
(1,'2022-02-06 19:05:00','6940000009',11550),
(1,'2022-02-10 19:05:00','6940000009',11650),
```

(2,'2022-06-15 09:50:00','6940000005',25),
(3,'2022-01-12 14:10:00','2101234567',190),
(3,'2022-08-12 14:10:00','2101234567',190),
(1,'2022-02-03 09:05:00','6940000009',440),
(5,'2022-03-21 10:15:00','6940000001',310),
(5,'2022-09-21 10:15:00','6940000001',310),
(6,'2022-04-09 22:45:00','6940000005',60),
(7,'2022-07-30 11:00:00','2105558888',90),
(9,'2022-10-10 16:20:00','6940000005',450),
(9,'2022-12-25 09:05:00','6940000003',45),
(10,'2021-05-18 13:40:00','2109990000',175),
(5,'2021-07-20 14:10:00','2101234567',140),
(2,'2021-02-03 19:05:00','6940000009',490),
(6,'2021-03-21 10:15:00','6940000001',360),
(7,'2021-04-09 22:45:00','6940000005',10),
(8,'2021-07-30 11:00:00','2105558888',100),
(9,'2021-10-10 16:20:00','6940000005',470),
(9,'2021-08-18 16:20:00','6940000003',170),
(9,'2021-12-25 09:05:00','6940000003',15),
(10,'2021-05-18 13:40:00','2109990000',75);

Output:



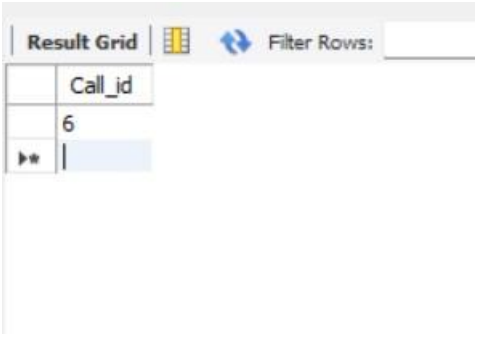
✓	9	14:20:25	CREATE TABLE CITY (City_id INT PRIMARY KEY AUTO_INCREMENT, City_Name VARCHAR(100) NO...
✓	10	14:20:25	CREATE TABLE CUSTOMER (Customer_id INT PRIMARY KEY AUTO_INCREMENT, First_name VARC...
✓	11	14:20:25	CREATE TABLE PLAN (Plan_id INT PRIMARY KEY AUTO_INCREMENT, Plan_Name VARCHAR(100) N...
✓	12	14:20:25	CREATE TABLE CONTRACT (Contract_id INT PRIMARY KEY AUTO_INCREMENT, Phone_number VA...
✓	13	14:20:25	CREATE TABLE CALLS (Call_id BIGINT PRIMARY KEY AUTO_INCREMENT, Contract_id INT NOT NUL...
✓	14	14:20:25	CREATE TRIGGER trg_calls_1 BEFORE INSERT ON CALLS FOR EACH ROW BEGIN DECLARE v_sta...
✓	15	14:20:25	CREATE TRIGGER trg_calls_2 BEFORE UPDATE ON CALLS FOR EACH ROW BEGIN DECLARE v_st...
✓	16	14:20:25	#Insert a few records into the tables INSERT INTO CITY (City_Name, Population, AVG_income) VALUES (...
✓	17	14:20:25	INSERT INTO PLAN (Plan_Name, Minutes, SMS, MB) VALUES ('Red2',300,300,6000), ('Student',400,400,...
✓	18	14:20:25	INSERT INTO CUSTOMER (First_name, Last_name, Date_of_birth, Gender, City_id) VALUES ('Nikos','Pap...
✓	19	14:20:25	INSERT INTO CONTRACT (Phone_number, Start_date, End_date, Contract_Description, Customer_id, Pla...
✓	20	14:20:25	INSERT INTO CALLS (Contract_id, Phone_Datetime, Called_phone, Duration) VALUES (1,'2022-02-03 19:...

3. **(60%) Write SQL code and test it to your data for the following queries**

#a. Show the call id of all calls that were made between 8am and 10am on June 2022 having duration < 30.

```
SELECT Call_id
FROM CALLS
WHERE Phone_Datetime >= '2022-06-01' AND Phone_Datetime < '2022-07-01'
  AND TIME(Phone_Datetime) BETWEEN '08:00:00' AND '10:00:00'
  AND Duration < 30;
```

Output:

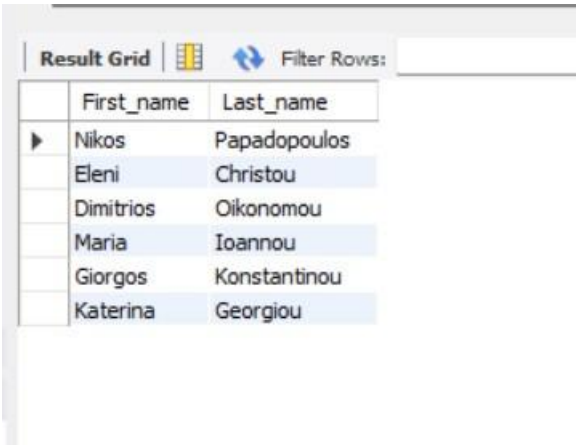


Call_id
6

#b. Show the first and last name of customers that live in a city with population greater than 20.000

```
SELECT c.First_name,c.Last_name
FROM CUSTOMER c join CITY ct on c.City_id = ct.City_id
WHERE ct.Population>20000;
```

Output:

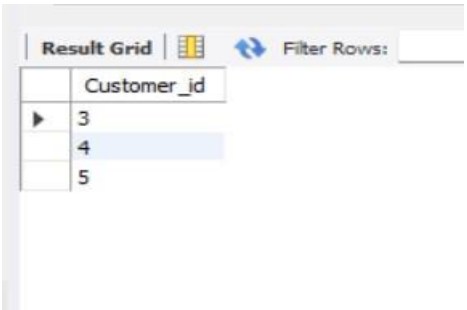


First_name	Last_name
Nikos	Papadopoulos
Eleni	Christou
Dimitrios	Oikonomou
Maria	Ioannou
Giorgos	Konstantinou
Katerina	Georgiou

#c. Show the customer id that have a contract in the plan with name LIKE 'Freedom' (use nested queries).

```
SELECT DISTINCT Customer_id
FROM CONTRACT
WHERE Plan_id IN (SELECT Plan_id FROM PLAN WHERE Plan_Name LIKE 'Freedom%')
ORDER BY Customer_id;
```

Output:



Customer_id
3
4
5

#d. For each contract that ends in less than sixty days from today, show the contract id, the phone number, the customer's id, his/her first name and his/her last name.

```
SELECT ct.Contract_id,ct.Phone_number,c.Customer_id,c.First_name,c.Last_name
FROM CUSTOMER c JOIN CONTRACT ct ON ct.Customer_id = c.Customer_id
WHERE  datediff(ct.End_date,CURDATE())BETWEEN 0 AND 59;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Contract_id	Phone_number	Customer_id	First_name	Last_name
1	6940000001	1	Nikos	Papadopoulos
3	6940000003	3	Giorgos	Konstantinou
5	6940000005	5	Dimitrios	Oikonomou

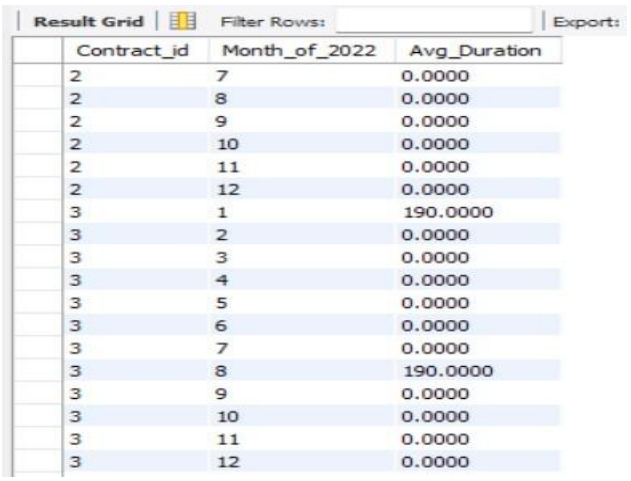
#e. For each contract id and each month of 2022, show the average duration of calls
(Note: we want all the contract ids and all the months of 2022 to be appear even if the duration is **zero**.)

```
WITH RECURSIVE months(mo) AS (
  SELECT 1
  UNION ALL
  SELECT mo + 1 FROM months WHERE mo < 12
)
SELECT
  ct.Contract_id,
  m.mo AS Month_of_2022,
  COALESCE(AVG(cl.Duration), 0) AS Avg_Duration
FROM CONTRACT ct
CROSS JOIN months m
LEFT JOIN CALLS cl
  ON cl.Contract_id = ct.Contract_id
 AND cl.Phone_Datetime >= '2022-01-01'
 AND cl.Phone_Datetime <  '2023-01-01'
 AND MONTH(cl.Phone_Datetime) = m.mo
GROUP BY ct.Contract_id, m.mo
ORDER BY ct.Contract_id, m.mo;
```

Sample **Outputs:**

Contract_id	Month_of_2022	Avg_Duration
1	1	0.0000
1	2	5856.6667
1	3	0.0000
1	4	0.0000
1	5	0.0000
1	6	0.0000
1	7	0.0000
1	8	0.0000
1	9	0.0000
1	10	0.0000
1	11	0.0000
1	12	0.0000
2	1	0.0000
2	2	0.0000
2	3	0.0000
2	4	0.0000
2	5	0.0000
2	6	25.0000

:

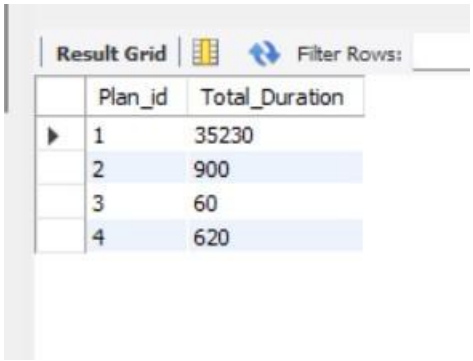


Contract_id	Month_of_2022	Avg_Duration
2	7	0.0000
2	8	0.0000
2	9	0.0000
2	10	0.0000
2	11	0.0000
2	12	0.0000
3	1	190.0000
3	2	0.0000
3	3	0.0000
3	4	0.0000
3	5	0.0000
3	6	0.0000
3	7	0.0000
3	8	190.0000
3	9	0.0000
3	10	0.0000
3	11	0.0000
3	12	0.0000

#f. Show the total duration of calls in 2022 per plan id

```
SELECT ct.Plan_id, SUM(cl.Duration) AS Total_Duration
FROM CALLS cl
JOIN CONTRACT ct ON cl.Contract_id = ct.Contract_id
WHERE cl.Phone_Datetime >= '2022-01-01'
  AND cl.Phone_Datetime <  '2023-01-01'
GROUP BY ct.Plan_id
ORDER BY ct.Plan_id;
```

Output:

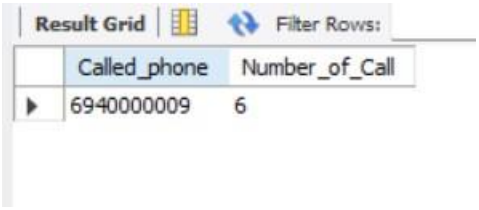


Plan_id	Total_Duration
1	35230
2	900
3	60
4	620

#g. Show the top called number among TP's customers in 2022

```
SELECT cl.Called_phone, COUNT(*) AS Number_of_Call
FROM CALLS cl
JOIN CONTRACT ct ON ct.Phone_number = cl.Called_phone
WHERE cl.Phone_Datetime >= '2022-01-01' AND cl.Phone_Datetime <  '2023-01-01'
GROUP BY cl.Called_phone
ORDER BY Number_of_Call DESC
LIMIT 1;
```

Output:

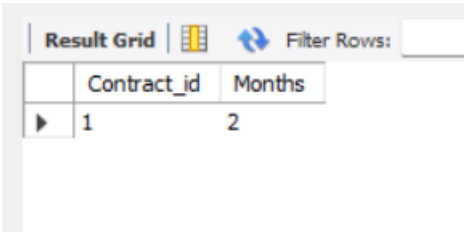


Called_phone	Number_of_Call
6940000009	6

#h. Show the contract ids and the months where the total duration of the calls was greater than the free minutes offered by the plan of the contract.
(Note: we interest for the months as total Jan,Feb etc. and we don't separate them by year)

```
SELECT TEMP1.Contract_id, TEMP1.Months
FROM (
  SELECT
    ct.Contract_id,
    MONTH(cl.Phone_Datetime) AS Months,
    SUM(cl.Duration) AS Total_Duration,
    MAX(p.Minutes) * 60 AS Seconds
  FROM CONTRACT ct
  JOIN CALLS cl ON cl.Contract_id = ct.Contract_id
  JOIN PLAN  p  ON p.Plan_id      = ct.Plan_id
  GROUP BY ct.Contract_id, MONTH(cl.Phone_Datetime)
) AS TEMP1
WHERE TEMP1.Total_Duration > TEMP1.Seconds
ORDER BY TEMP1.Contract_id, TEMP1.Months;
```

Output:



Contract_id	Months
1	2

#i. For each month of 2022, show the percentage change of the total duration of calls compared to the same month of 2021.
(Note: we want all months of 2022 to be appear even if there is no call.**)**

```
WITH RECURSIVE months(mo) AS (
  SELECT 1
  UNION ALL
```

```
  SELECT mo + 1 FROM months WHERE mo < 12
),
m21 AS (
  SELECT MONTH(Phone_Datetime) AS Mo, SUM(Duration) AS Total_2021
  FROM CALLS
  WHERE Phone_Datetime >= '2021-01-01' AND Phone_Datetime < '2022-01-01'
  GROUP BY MONTH(Phone_Datetime)
),
m22 AS (
  SELECT MONTH(Phone_Datetime) AS Mo, SUM(Duration) AS Total_2022
  FROM CALLS
  WHERE Phone_Datetime >= '2022-01-01' AND Phone_Datetime < '2023-01-01'
  GROUP BY MONTH(Phone_Datetime)
)
SELECT
  m.mo AS Months,
  COALESCE(m22.Total_2022, 0) AS Total_2022,
  COALESCE(m21.Total_2021, 0) AS Total_2021,
  CASE
    WHEN COALESCE(m21.Total_2021,0) = 0 AND COALESCE(m22.Total_2022,0) > 0 THEN
100.00
    WHEN COALESCE(m21.Total_2021,0) > 0 AND COALESCE(m22.Total_2022,0) = 0 THEN -
100.00
    WHEN COALESCE(m21.Total_2021,0) = 0 AND COALESCE(m22.Total_2022,0) = 0 THEN
0.00
    ELSE ROUND(100.0 * (m22.Total_2022 - m21.Total_2021) / NULLIF(m21.Total_2021,0), 2)
  END AS Percentage_Change
FROM months m
LEFT JOIN m22 ON m22.Mo = m.mo
LEFT JOIN m21 ON m21.Mo = m.mo
ORDER BY Months;
```

Output:

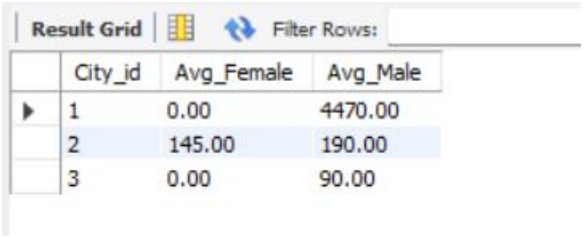
Result Grid | Filter Rows: | Export:

Months	Total_2022	Total_2021	Percentage_Change
1	190	0	100.00
2	35140	490	7071.43
3	310	360	-13.89
4	60	10	500.00
5	0	250	-100.00
6	25	0	100.00
7	90	240	-62.50
8	190	170	11.76
9	310	0	100.00
10	450	470	-4.26
11	0	0	0.00
12	45	15	200.00

#j. For each city id and calls made in 2022, show the average call duration by females and the average call duration by males (i.e. three columns)

```
SELECT ci.City_id,
      ROUND(COALESCE(AVG(CASE WHEN c.Gender='Female' THEN cl.Duration END),0), 2) AS
Avg_Female,
      ROUND(COALESCE(AVG(CASE WHEN c.Gender='Male'   THEN cl.Duration END),0), 2) AS
Avg_Male
FROM CALLS cl
JOIN CONTRACT ct ON ct.Contract_id = cl.Contract_id
JOIN CUSTOMER c  ON c.Customer_id  = ct.Customer_id
JOIN CITY ci     ON ci.City_id     = c.City_id
WHERE cl.Phone_Datetime >= '2022-01-01' AND cl.Phone_Datetime < '2023-01-01'
GROUP BY ci.City_id
ORDER BY ci.City_id;
```

Output:


Result Grid | Filter Rows:

City_id	Avg_Female	Avg_Male
1	0.00	4470.00
2	145.00	190.00
3	0.00	90.00

#k. For each city id, show the city id, the ratio of the total duration of the calls made from customers staying in that city in 2022 over the total duration of all calls made in 2022, and the ratio of the city's population over the total population of all cities (i.e three columns)

```
WITH TEMP4 AS (
  SELECT SUM(cl.Duration) AS Total_Duration_2022
  FROM CALLS cl
  WHERE cl.Phone_Datetime >= '2022-01-01'
    AND cl.Phone_Datetime <  '2023-01-01'),
TEMP5 AS (
  SELECT
    ci.City_id,
    SUM(cl.Duration) AS CitySec,
    MAX(ci.Population) AS Population
  FROM CITY ci
  LEFT JOIN CUSTOMER cu ON cu.City_id     = ci.City_id
  LEFT JOIN CONTRACT ct ON ct.Customer_id = cu.Customer_id
  LEFT JOIN CALLS    cl ON cl.Contract_id  = ct.Contract_id
                       AND cl.Phone_Datetime >= '2022-01-01'
                       AND cl.Phone_Datetime <  '2023-01-01'
  GROUP BY ci.City_id),
TEMP6 AS (
  SELECT SUM(Population) AS Total_Population
  FROM CITY)
SELECT
  TEMP5.City_id,
  COALESCE(TEMP5.CitySec,0) / NULLIF(TEMP4.Total_Duration_2022,0) AS
City_Duration_Ratio,
  TEMP5.Population / NULLIF(TEMP6.Total_Population,0)  AS Population_Ratio
FROM TEMP5
CROSS JOIN TEMP4
CROSS JOIN TEMP6
ORDER BY TEMP5.City_id;
```

Output:

Result Grid | Filter Rows: | Export:

City_id	City_Duration_Ratio	Population_Ratio
1	0.9715	0.7115
2	0.0261	0.2846
3	0.0024	0.0039

4. **(20%) Using the programming language of your choice, connect to the database and implement query (k) above – without using GROUP BY SQL statements.**

```
5. import pymysql
   from collections import OrderedDict, defaultdict

   DB = dict(host="localhost", user="root", password="*******",
   database="Assignment_1", charset="utf8mb4")

   def safe_ratio(n, d): return 0.0 if not d else n / d

   conn = pymysql.connect(**DB, cursorclass=pymysql.cursors.DictCursor)

   city_population = OrderedDict()
   city_seconds    = defaultdict(int)
   total_pop = 0
   total_sec = 0

   with conn.cursor() as cur:
       cur.execute("SELECT City_id, Population FROM CITY ORDER BY City_id")
       for row in cur:
           cid = int(row["City_id"]); pop = int(row["Population"])
           city_population[cid] = pop
           total_pop += pop
           city_seconds[cid] += 0

   with conn.cursor() as cur:
       cur.execute("""
           SELECT ci.City_id, cl.Duration
           FROM CITY ci
           LEFT JOIN CUSTOMER cu ON cu.City_id = ci.City_id
           LEFT JOIN CONTRACT ct ON ct.Customer_id = cu.Customer_id
           LEFT JOIN CALLS   cl ON cl.Contract_id = ct.Contract_id
                                AND cl.Phone_Datetime >= '2022-01-01'
                                AND cl.Phone_Datetime <  '2023-01-01'
       """)
       for row in cur:
           dur = row["Duration"]
           if dur is not None:
               cid = int(row["City_id"]); d = int(dur)
               city_seconds[cid] += d
               total_sec += d

   conn.close()

   print(f"{'City_id':>7} | {'Duration_ratio':>15} |
   {'Population_ratio':>17}")
   for cid, pop in city_population.items():
       dur = city_seconds.get(cid, 0)
       print(f"{cid:7d} | {safe_ratio(dur, total_sec):15.6f} |
   {safe_ratio(pop, total_pop):17.6f}")
```

Output:

```
C:\Users\jimts\anaconda3\python.exe C:\Users\jimts\PycharmProjects\sql\4k.py
City_id |  Duration_ratio |  Population_ratio
      1 |        0.971475 |          0.711501
      2 |        0.026080 |          0.284599
      3 |        0.002445 |          0.003900

Process finished with exit code 0
```