

Data Management and Business Intelligence

Assignment 1

ELENI RALLI (f2822312)

A telecom provider (TelcoX)

**🡪** develop a relational database to monitor customers, calls and plans.

**Customers**

1. a unique identifier,
2. first name
3. last name **a** customer has **one/more** contracts with
4. date of birth Telcox.
5. gender (‘male’ or ‘female’) and
6. live in a city.

**Cities**

1. a unique identifier,
2. name,
3. population
4. mean income.

**Contract**

1. a unique identifier,
2. phone number,
3. starting date, **a** contract is associated to **a** plan
4. ending date offered by Telcox.
5. a description.

**Plan**

1. a unique identifier,
2. name,
3. free-minutes,
4. free-sms
5. free-MB

**Calls**

1. a unique identifier,
2. the date/time of the call (hour, minute, day, month, year) calls made
3. the called phone number by a phone number
4. the duration of the call (in seconds).

**Deliverables:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

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.

According to the data of the assignment,

There are 5 **entities**, with the following **attributes**:

1. **Customer** (customer\_id, first\_name, last\_name, dateofbirth, gender)
2. **City** (city\_id, city\_name, population, mean\_income)
3. **Contract** (contract\_id, phone\_number, start\_date, end\_date, descrip)
4. **Plan** ( plan\_id, plan\_name, freemin, freesms, freemb )
5. **Call** ( call\_id, datetimeofcall, calledphonenumber, duration)

Νow, the relationship between the entities should be specified:

* what attributes will connect the entities
* how entities interact with each other (relationships),

mapping cardinalities to express the number of entities to which another entity can be associated via a relationship set (we have binary relationship sets so the mapping is useful),

participation

**Relationships:**

1. **Customer live in City (many to one)**

* A Customer could live in one City (the min number of Cities that a customer can lives is 1, and the max number of cities that a customer can live is also 1. So, “one and only one” relationship 1:1)
* In a city could live many Customers (the min number of Customers that can live a city is 0, and the max number of customers that can live a city is “many”. So, “zero or many” relationship)

So, the relationship between Customer and city is **“many to one “.**

A Customer could live in one and only one city and in a city could live zero or many customers.

1. **Customer signing Contract (one to many)**

* A Customer could sign one/more Contracts (the min number of Contracts that a customer could sign is 1- *otherwise he is not even considered a customer*, and the max number is many- *a customer can have many different numbers and so contracts if he wants. So*, “one or many” relationship)
* A Contract could be signed by one Customers (the min number of Customers that could sign a Contract is 1- *otherwise this contract does not even exist, a phone number needs to be given to a customer and start to exist*,

and the max is 1- *A telephone number always belongs to a specific customer*

. So, “one and only one” relationship)

So, the relationship between Customer and Contract is **“one to many “.**

A Customer could sign one/many Contracts and a Contract could be signed by one and only one Customers.

1. **Contract** **is associated to Plan (many to one)**

* A Contract could be associated to one Plan (the min number of Plans that a Contract could be associated is 1- *in order to make a contract a specific plan must be chosen*

and the max number of Plans that a Contract could be associated is also 1. So, “one and only one” relationship)

* A Plan could be associated to many Contracts (the min number of Contracts that could sign a Plan is 0*- e.g., no client wanted this specific plan, so it is not included in any contract*,

and the max is “many*”- e.g., it's a very good plan and everyone wants it.*

So, “zero or many” relationship)

So, the relationship between Contract and Plan is **“many to one “.**

A Contract could be associated to one and only one Plan and a Plan could be associated to zero or many Contracts.

1. **Contract associated to Call (one to many)**

* A Contract could be associated to many Calls*,*

*because a contract has a phone number and calls made by a phone number.*

(The min number of calls that a contract-phone number could make is 0 – e.g., he never made a single phone call and the max is many. So, “zero or many” relationship)

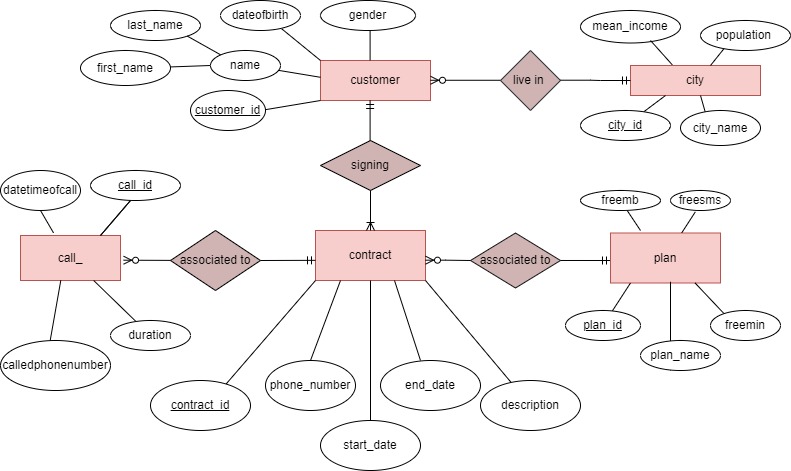
* A Call could be associated to one Contract. (The min number of Contracts that a call is associated with is 1 ant the max is also 1 - the call is made from a specific phone number. So, “one and only one” relationship)

So, the relationship between Contract and Call is **“one to many “.**

A Contract could be associated to zero or many Calls and a Call can be associated to one and only one Contract.

I will use the online tool draw.io <https://app.diagrams.net/>

**Entity-Relationship Diagrams (ERD) with the method of Graphical figures (shapes) :**



• Rectangles represent entity sets. (Customer, City, Call, Contract, Plan)

• Diamonds represent relationship sets.

• Lines link attributes to entity sets and entity sets to relationship sets.

The notation on the lines is there to note the Cardinalities and Participation constraints

(also called minimum cardinality constraint) exactly as I have explained before.

• Ellipses represent attributes

• Double ellipses represent multivalued attributes.

• Underline indicates primary key attributes.

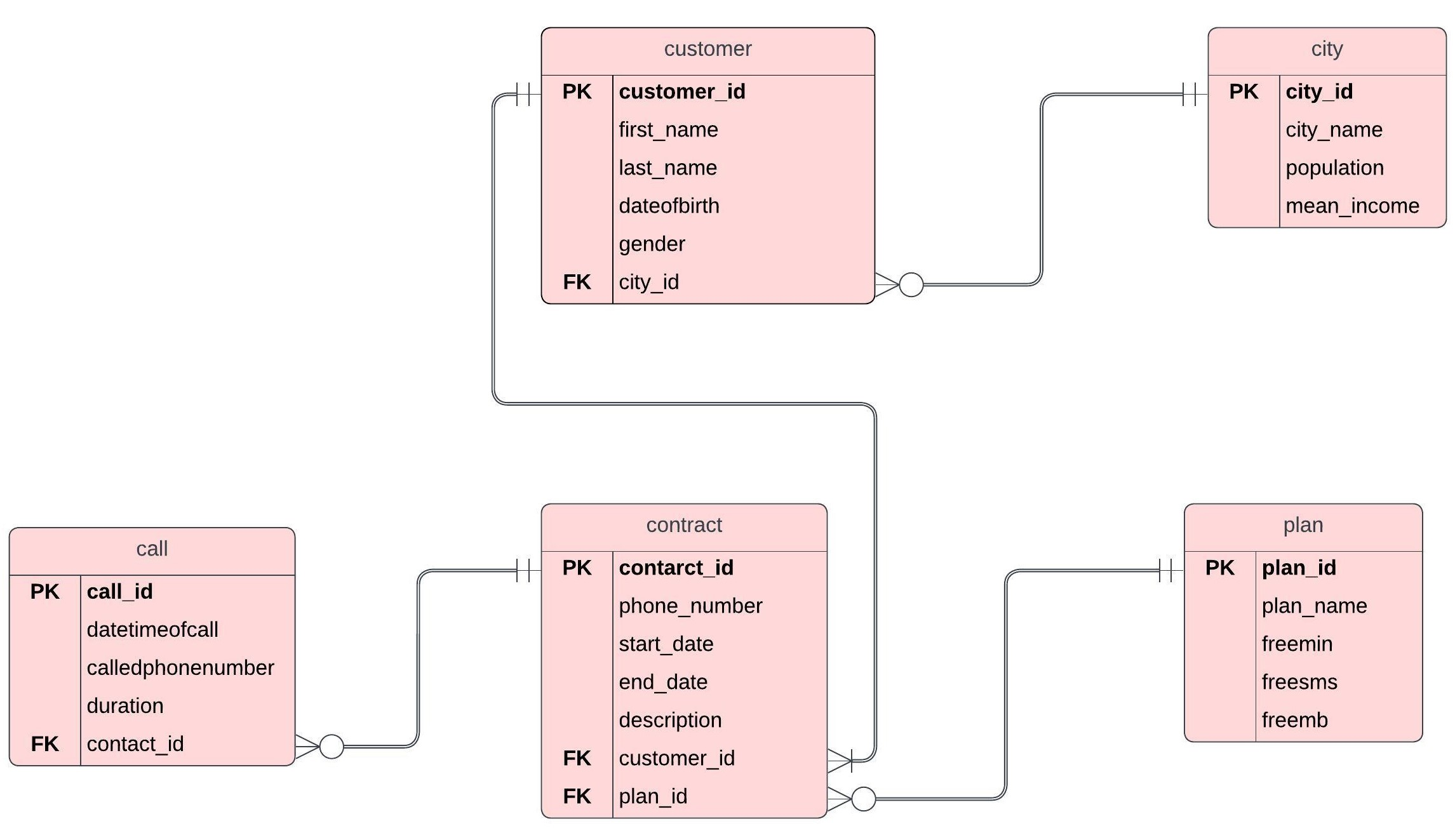
Note:

* The name attribute is composite multi-valued attribute ( be composed of first\_name , last\_name but there are people with 2 names , so “multi-valued” ) .
* All the relationship sets are binary.

In this kind of ERD, we **cannot** have keys from other entities as foreign keys on other entities, because that way the concept of association is lost.

For a better understanding, the ERD will also be made with tables (in this method we can see also the foreign keys inside the table-entity), as combined these 2 methods for constructing graphs give a better understanding of the database that we will create.

**Entity-Relationship Diagrams (ERD) with tables – “physical modeling”:**



Unique

PK = Primary key Never changing

Never NULL

the same as a Primary key, but just located in foreign table.

Deciding which table should contain the foreign key attribute FK = Foreign key depends on the nature of the relationship between the tables and

the direction of the relationship.

a primary key in a foreign key typically makes good column choices

for joining together two or more tables

• If one to many or many to one = on the «many» side we put another column that is the primary key of the other entity and make it a foreign key

e.g., I want to represent “A customer live in a city “and no “a city has these customers”, so the foreign key will be in the Customer table.

Also, I want to represent “A Contact could be signed by one Customers “, so the foreign key will be in the Contract table.

“A Call could be associated to one Contract “, so the foreign key will be in the Call table.

etc.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

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.

Create the relational schema in MySQL:

* I will create first the tables that don’t have foreign keys, because we want to create the referenced tables(parent tables) before the referencing tables(child tables that contain the foreign Key). So, the referenced keys will already exist when defining foreign keys (If I want to delete tables I have to start from the last one I created).

CREATE SCHEMA `assignment1` ;

USE assignment1 ;

CREATE TABLE plan (

plan\_id INT PRIMARY KEY, -- no need to declare NOT NULL is included

plan\_name VARCHAR(100), -- it may not have a name, it may be customized, so I will not put a

-- NOT NULL constraint

freemin INT NOT NULL ,

freesms INT NOT NULL ,

freemb INT NOT NULL

);

CREATE TABLE city (

city\_id INT NOT NULL PRIMARY KEY,

city\_name VARCHAR(30) NOT NULL ,

population INT,

mean\_income DECIMAL(10,2) CHECK (mean\_income>0)

);

CREATE TABLE customer (

customer\_id INT NOT NULL PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

-- the format 'YYYY-MM-DD' I'll have to put a restriction over 18 BUT I'LL HAVE TO DO IT --OTHERWISE BECAUSE IT'S NOT RECEIVING THIS WAY

dateofbirth DATE NOT NULL ,

gender ENUM('male', 'female') CHECK (gender IN ('male','female')),

city\_id INT, FOREIGN KEY(city\_id) REFERENCES city(city\_id)

);

CREATE TABLE contract (

contract\_id INT NOT NULL PRIMARY KEY ,

phone\_number VARCHAR(20) UNIQUE ,

start\_date DATE,

end\_date DATE,

descrip TEXT,

customer\_id INT ,FOREIGN KEY (customer\_id) REFERENCES customer(customer\_id),

plan\_id INT ,FOREIGN KEY (plan\_id) REFERENCES plan(plan\_id)

);

CREATE TABLE call\_ (

call\_id INT NOT NULL PRIMARY KEY,

datetimeofcall DATETIME NOT NULL,

calledphonenumber VARCHAR(20) NOT NULL,

duration INT NOT NULL ,

contract\_id INT

);

ALTER TABLE call\_

ADD CONSTRAINT fk\_contract FOREIGN KEY(contract\_id) REFERENCES contract(contract\_id);

I want to add one more constraint to the customer table, customers must be over 18 to have a contract, so I want to limit the entries that are made - not to let people born after a certain date as customers.

We have the date of birth so, I will need a function where it will find the difference of today's date minus the date of birth it will return me the years of this difference, which I want to be >=18 so that the customer is an adult.

There is already such a function in mysql : TIMESTAMPDIFF(unit, datetime\_expr1, datetime\_expr2)

DELIMITER //

CREATE FUNCTION check\_age\_adult (dateofbirth DATE)

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE curent\_age INT;

SET curent\_age = TIMESTAMPDIFF(YEAR, dateofbirth, CURDATE());

IF curent\_age >= 18 THEN

RETURN 1; -- Indicates true

ELSE

RETURN 0; -- Indicates false

END IF;

END;

//

DELIMITER ;

and, now I'm going to create a trigger that will fire every time I go to put age values in the customers.

DELIMITER //

CREATE TRIGGER check\_age\_before\_insert BEFORE INSERT ON customer

FOR EACH ROW

BEGIN

IF check\_age\_adult(NEW.dateofbirth) = 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Age must be 18 or older.';

END IF;

END;

//

DELIMITER ;

I enter data into the tables I created:

INSERT INTO plan (plan\_id, plan\_name, freemin, freesms, freemb)

VALUES (1, 'Freedom', 100, 500, 9000),

(2, 'Studentfull', 150, 300, 3072),

(3, 'Businessto', 900, 300, 8072);

INSERT INTO city (city\_id, city\_name, population, mean\_income)

VALUES (1, 'Athens', 3154000, 35259),

(2, 'Patra', 173600 , 30884),

(3, 'Kalamata', 54567, 31000),

(4, 'Mesologi',12785,28000)

(5, 'Sparti', 18926, 29000);

INSERT INTO customer (customer\_id, first\_name, last\_name, dateofbirth, gender, city\_id)

VALUES (1, 'Nikos', 'Nikolaou', '2000-01-15', 'male', 2),

(2, 'Katerina', 'Romanou', '1995-03-10', 'female', 1),

(3, 'Petros', 'Nikiou', '2000-08-25', 'male', 5),

(4,'Eleni','Petrou','1977-08-19','female',4),

(5,'Simeon','Grigoriou','1964-03-29','male',1),

(6,'Panagiota','Panagiwtou','1996-02-16','female',2),

(7,'Alexia','Petrou','1967-12-19','female',3),

(8,'Eleanna','Floka','2001-12-09','female',4),

(9,'Basiliki','Stelounou','1977-03-14','female',5),

(10,'Dionisis','Alexiou','1967-07-04','male',2);

INSERT INTO contract (contract\_id, phone\_number, start\_date, end\_date, descrip, customer\_id, plan\_id)

VALUES (1, '6972515669', '2020-01-15', '2023-12-03', 'morefreemb', 1, 1),

(2, '6974455688' , '2023-02-20', '2025-02-19', 'pasofree', 2, 2),

(3, '6986988456', '2022-03-10', '2024-01-02', 'businesscontract', 3, 3),

(4, '6976900111', '2023-04-05', '2024-04-04', 'morefreemb', 4, 1),

(5, '6976944123', '2023-05-12', '2024-05-11', 'morefreemb', 5, 1),

(6, '6978544852', '2023-06-21', '2024-06-20', 'businesscontract', 6, 3),

(7, '6978455123', '2021-07-11', '2024-07-10', 'morefreemb', 7, 1),

(8, '6976988452', '2023-08-09', '2024-08-08', 'pasofree', 8, 2),

(9, '6977477741', '2023-09-15', '2024-09-14', 'businesscontract', 9, 3),

(10, '6976999992', '2023-10-25', '2024-10-24', 'morefreemb', 10, 1);

-- Insert data into the `call\_` table

INSERT INTO call\_ (call\_id, datetimeofcall, calledphonenumber, duration, contract\_id)

VALUES (1, '2022-01-15 09:30:00', '6974455688', 4050, 1),

(2, '2023-08-16 15:45:00', '6998847456', 960, 2),

(3, '2022-02-10 14:20:00', '6974455688', 360, 3),

(4, '2022-06-20 09:10:00', '6944887744', 20, 4),

(5, '2023-06-05 17:55:00', '6774488774', 1500, 10),

(6, '2021-03-10 08:30:00', '6112255663', 620, 6),

(7, '2023-04-12 12:15:00', '6774411223', 10, 7),

(8, '2023-05-05 20:40:00', '6112288995', 9001, 8),

(9, '2022-05-25 10:05:00', '6976900841', 540, 9),

(10, '2023-06-18 14:25:00', '6976922471', 805, 10),

(11, '2023-06-19 14:25:00', '6976922472', 909, 10),

(12, '2023-06-21 14:25:00', '6976922473', 999, 10),

(13, '2023-06-25 16:25:00', '6976922473', 2000, 10),

(14, '2021-01-18 08:30:00', '6112255663',3000, 6),

(15, '2021-02-12 08:30:00', '6112255663', 6200, 6),

(16, '2021-06-13 08:30:00', '6112255663', 15, 6),

(17, '2021-05-17 08:30:00', '6112255663', 150, 6)

;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

I'll need the table:

* call\_ ( call\_id, datetimeofcall, calledphonenumber, duration,contract\_id)

I will use WHERE where with this the rows are essentially filtered according to the conditions that are requested .

The datetimeofcall attribute has the format 'Year-Month-Day Hour:Minute:Second'.

To be able to select individual specific elements (Hour, Minutes AND YEAR, MONTH) of this attribute I will use the ready function

DATE\_FORMAT .

General syntax : DATE\_FORMAT(date\_value, format)

and in this function I will use the following symbols which mean:

%H: Hour (00-23)

%i: Minutes (00-59)

%Y: Year with century as a decimal number (e.g., 2022)

%m: Month as a zero-padded decimal number (e.g., 05 for May)

I will also need BETWEEN where it is the same as saying

DATE\_FORMAT(datetimeofcall, '%H:%i') >= '08:00'

AND DATE\_FORMAT(datetimeofcall, '%H:%i') <= '10:00'

AND stands for intersection of sets – they must apply at the same time.

**SELECT** call\_id

**FROM** call\_

**WHERE** **DATE\_FORMAT (**datetimeofcall, '%H : %i'**)** **BETWEEN** '08:00' **AND** '10:00'

**AND DATE\_FORMAT** (datetimeofcall, '%Y-%m') = '2022-06'

**AND** duration < 30;

It gives me :

|  |  |
| --- | --- |
|  | call\_id |
|  | 4 |

I can check that what I have done is correct:

|  |  |  |
| --- | --- | --- |
| call\_id | datetimeofcall | duration |
| 1 | "2022-01-15 09:30:00" | 450 |
| 2 | "2023-08-16 15:45:00" | 560 |
| 3 | "2020-02-10 14:20:00" | 360 |
| 4 | "2022-06-20 09:10:00" | 20 |
| 5 | "2023-03-05 17:55:00" | 390 |
| 6 | "2021-03-10 08:30:00" | 620 |
| 7 | "2023-04-12 12:15:00" | 10 |
| 8 | "2023-05-05 20:40:00" | 480 |
| 9 | "2022-05-25 10:05:00" | 540 |
| 10 | "2023-06-18 14:25:00" | 490 |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Show the **first and last name of customers** that live in a city with **population greater than 20000**

* Customer (customer\_id, first\_name, last\_name, dateofbirth, gender, city\_id)
* City (city\_id, city\_name, population, mean\_income)

I will use an explicit join, specifically the INNER JOIN to combine information from these 2 tables. This type of join returns only the rows where there is a match in both of the joined tables.

**SELECT** c**.**first\_name, c.last\_name

**FROM** customer AS c

**INNER JOIN** city AS ci -- I could have just written JOIN

**ON** c.city\_id=ci.city\_id

**WHERE**  ci.population >20000 ;

I can get the same result with implicit join:

**SELECT** c.first\_name, c.last\_name

**FROM** customer AS c , city AS ci

**WHERE**  c.city\_id=ci.city\_id AND ci.population >20000 ;

|  |  |
| --- | --- |
| first\_name | last\_name |
| Katerina | Romanou |
| Simeon | Grigoriou |
| Nikos | Nikolaou |
| Panagiota | Panagiwtou |
| Dionisis | Alexiou |
| Alexia | Petrou |

To check that is right :

**SELECT c.**first\_name, c.last\_name, ci.population,ci.city\_name

**FROM** customer c

**INNER JOIN** city ci

**ON** c.city\_id=ci.city\_id

**WHERE**  ci.population >20000 ;

We see that the cities of Mesologi and Sparti are not included:

|  |  |  |  |
| --- | --- | --- | --- |
| first\_name | last\_name | population | city\_name |
| Katerina | Romanou | 3154000 | Athens |
| Simeon | Grigoriou | 3154000 | Athens |
| Nikos | Nikolaou | 173600 | Patra |
| Panagiota | Panagiwtou | 173600 | Patra |
| Dionisis | Alexiou | 173600 | Patra |
| Alexia | Petrou | 54567 | Kalamata |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Show the **customer id** that have **a contract in the plan with name LIKE ‘Freedom’** (use nested queries).

* contract (contract\_id, phone\_number, start\_date, end\_date, descrip,customer\_id,plan\_id)
* plan ( plan\_id, plan\_name, freemin, freesms, freemb )

Essentially, we select in the WHERE through more complex conditions, that is, not a simple condition but involve some set of values within the condition. For this reason we use the IN operator, because the subquery returns back multiple values.

**SELECT** customer\_id

**FROM** contract

**WHERE** plan\_id **IN** ( **SELECT** plan\_id

**FROM** plan

**WHERE** plan\_name **LIKE** 'Freedom');

|  |  |
| --- | --- |
|  | customer\_id |
|  | 1 |
|  | 4 |
|  | 5 |
|  | 7 |
|  | 10 |

To check if I have it right, I should do a JOIN and not nested queries, because with the subqueries in the external SELECT I can't also request the plan\_names:

**SELECT** c.customer\_id, p.plan\_name,p.plan\_id

**FROM** contract c

**JOIN**  plan p

**ON** c.plan\_id = p.plan\_id

**WHERE**  p.plan\_name **LIKE** 'Freedom';

|  |  |  |
| --- | --- | --- |
| customer\_id | plan\_name | plan\_id |
| 1 | Freedom | 1 |
| 4 | Freedom | 1 |
| 5 | Freedom | 1 |
| 7 | Freedom | 1 |
| 10 | Freedom | 1 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. 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.

* contract (contract\_id, phone\_number, start\_date, end\_date, descrip,customer\_id,plan\_id)
* customer (customer\_id, first\_name, last\_name, dateofbirth, gender, city\_id)

I will also use the ready-made DATEDIFF function, calculates the difference

between two dates and returns the result in terms of a specified date part (such as days, months, years, etc.).

The general syntax is : DATEDIFF(date\_part, start\_date, end\_date).

And I will also use the ready-made function CURDATE(), which return the current date in the 'YYYY-MM-DD' format.

**SELECT** co.contract\_id, co.phone\_number, cu.customer\_id, cu.first\_name, cu.last\_name

**FROM** contract co

**JOIN** customer cu

**ON** co.customer\_id = cu.customer\_id

**WHERE DATEDIFF(**co.end\_date, **CURDATE())** < 60;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| contract\_id | phone\_number | customer\_id | first\_name | last\_name |
| 1 | 6972515669 | 1 | Nikos | Nikolaou |
| 3 | 6986988456 | 3 | Petros | Nikiou |

We can also do the same with implicit join:

**SELECT** co.contract\_id, co.phone\_number, cu.customer\_id, cu.first\_name, cu.last\_name

**FROM** contract co, customer cu

**WHERE** co.customer\_id = cu.customer\_id **AND DATEDIFF(**co.end\_date, **CURDATE())** < 60;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. For each contract id and **each month of 2022,** show the average duration of calls

* contract (contract\_id, phone\_number, start\_date, end\_date, descrip, customer\_id,plan\_id)
* call\_ ( call\_id, datetimeofcall, calledphonenumber, duration,contract\_id)

"For each contract id and each month of 2022" From this expression we understand

that we are talking about grouping, by contract id and by each month of 2022. So I do GROUP BY co.contract\_id, month .

In each of these groups, I calculate AVG(duration).

**SELECT** co.contract\_id,

**DATE\_FORMAT**(datetimeofcall, '%Y-%m') **AS** month,

**AVG(**duration) **AS** 'average duration'

**FROM** call\_ ca

**JOIN** contract co

**ON** ca.contract\_id = co.contract\_id

**WHERE** **DATE\_FORMAT(**datetimeofcall, '%Y') = '2022'

**GROUP BY** co.contract\_id, month;

|  |  |  |
| --- | --- | --- |
| contract\_id | month | "average duration" |
| 1 | 2022-01 | 4050.0000 |
| 3 | 2022-02 | 360.0000 |
| 4 | 2022-06 | 20.0000 |
| 9 | 2022-05 | 540.0000 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

* contract (contract\_id, phone\_number, start\_date, end\_date, descrip, customer\_id,plan\_id)
* call\_ ( call\_id, datetimeofcall, calledphonenumber, duration,contract\_id)

The SUM function calculates a sum. In each of these groups, I calculate SUM (ca.duration).

**SELECT** co.plan\_id, **SUM(**ca.duration) **AS** total\_duration

**FROM** call\_ ca

**JOIN** contract co

**ON** ca.contract\_id = co.contract\_id

**WHERE DATE\_FORMAT**(ca.datetimeofcall, '%Y') = '2022' -- **YEAR(**datetimeofcall**)** =

**GROUP BY** co.plan\_id;

|  |  |
| --- | --- |
| plan\_id | total\_duration |
| 1 | 4070 |
| 3 | 900 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Show the top called number among TP’s customers in 2022

* contract (contract\_id, phone\_number, start\_date, end\_date, descrip, customer\_id,plan\_id)
* call\_ ( call\_id, datetimeofcall, calledphonenumber, duration,contract\_id)

Each customer has a contract, each contract corresponds to a phone\_number.

We want to see out of these mobile phones, which was the one on which there were the most calls (so I will have to count calledphonenumber – count( \* )).

So, I say, from the table call\_

Do an inner join with table contract in order to match with «customers» (each contract = a customer), (grab only the rows that happen to be in both table).

Filter the rows for 2022.

And now group by calledphonenumber (because calls with a different id, they may call the same number).

after that, go and count how many values are in each group

sort them from the one with the most values to the one with the least

and show me the ca.calledphonenumber, call\_count “table” but only the first row.

**SELECT** ca.calledphonenumber, **COUNT(\*)** AS call\_count

**FROM** call\_ ca

**INNER JOIN** contract co

**ON** ca.contract\_id = co.contract\_id

**WHERE YEAR(**datetimeofcall**)** = 2022

**GROUP BY** ca.calledphonenumber

**ORDER BY** call\_count **DESC**

**LIMIT** 1;

|  |  |
| --- | --- |
| calledphonenumber | call\_count |
| 6974455688 | 2 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. 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

* call\_ ( call\_id, datetimeofcall, calledphonenumber, duration,contract\_id)
* contract (contract\_id, phone\_number, start\_date, end\_date, descrip, customer\_id,plan\_id)
* plan ( plan\_id, plan\_name, freemin, freesms, freemb )

I put my solution in an internal query so that the only two columns that are requested are returned to the screen.

The question asks me to apply an aggregate function (SUM) on the duration of the calls and after the result that this grouping will give me to filter it (HAVING)

Essentially, what is done here is:

I join with Inner Join 3 tables and group by cal.contract\_id and then by month. Then I filter according to "where the total duration of the calls was greater than the free minutes offered by the plan of the contract".

SELECT sub.contract\_id,sub.months

FROM (

**SELECT** cal.contract\_id,

EXTRACT(**MONTH FROM** cal.datetimeofcall) **as** months,

**SUM**(duration**)** **as** totalduration,

pl.freemin a**s** freeminutes

**FROM** call\_ **as** cal

**INNER JOIN** contract **as** con

**ON** cal.contract\_id=con.contract\_id

**INNER JOIN** plan **as** pl

**ON** con.plan\_id=pl.plan\_id

**GROUP BY** cal.contract\_id, months

**HAVING**  totalduration > freeminutes \*60 -- sec=min\*60

) as sub;

|  |  |
| --- | --- |
| contract\_id | months |
| 10 | 6 |
| 8 | 5 |

Inner query table:

|  |  |  |  |
| --- | --- | --- | --- |
| contract\_id | months | totalduration | freeminutes |
| 10 | 6 | 6213 | 100 |
| 8 | 5 | 9001 | 150 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. For each month of 2022, show the percentage change of the total duration of calls compared to the same month of 2021

* call\_ ( call\_id, datetimeofcall, calledphonenumber, duration, contract\_id)

From the question I understand that a grouping should be done by month.

But we will also group by call\_id as:

1) When you use GROUP BY, you need to include all non-aggregated columns from the SELECT clause in the GROUP BY clause.

2) I would lose the distinction between different cal.call\_id values within the same month.

We also understand that we will have to filter to keep only the years 2021 and 2022.

The percentage he requests is given by the general relationship:

**percentage\_change = ((new\_value - old\_value) / old\_value ) \*100**

I will initially make the following table to better understand what I am going to use:

Here from the call\_ table, I filter to get only the 2 years I want and then group by call\_id and month. Finally I ask to see the following columns shown in the table.

**SELECT** cal.call\_id,

**DATE\_FORMAT**(datetimeofcall, '%Y') **AS** years,

**DATE\_FORMAT**(datetimeofcall, '%m') **AS** months,

SUM(cal.duration) as total\_duration

**FROM** call\_ as cal

**WHERE DATE\_FORMAT(**datetimeofcall, '%Y'**) IN ('**2021', '2022'**)** -- in WHERE it don't let

-- me use alias years, and in having

**GROUP BY** cal.call\_id, months

This gives me the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| call\_id | years | months | total\_duration |
| 1 | 2022 | 01 | 4050 |
| 3 | 2022 | 02 | 360 |
| 4 | 2022 | 06 | 20 |
| 6 | 2021 | 03 | 620 |
| 9 | 2022 | 05 | 540 |
| 14 | 2021 | 01 | 3000 |
| 15 | 2021 | 02 | 6200 |
| 16 | 2021 | 06 | 15 |
| 17 | 2021 | 05 | 150 |

So now I basically want to create 2 "SUB-tables" of this and calculate the percentage\_change (one for 2002 and the other for 2021).

**CREATE VIEW** year\_2022 **AS**

**SELECT** cal.call\_id,

**DATE\_FORMAT**(datetimeofcall, '%Y') **AS** years,

**DATE\_FORMAT**(datetimeofcall, '%m') **AS** months,

SUM(cal.duration) as total\_duration

**FROM** call\_ as cal

**WHERE DATE\_FORMAT(**datetimeofcall, '%Y'**) =**2022

**GROUP BY** cal.call\_id, months ;

-- -----------------------------------------------------------------------------------------------------------

**CREATE VIEW** year\_2021 **AS**

**SELECT** cal.call\_id,

**DATE\_FORMAT**(datetimeofcall, '%Y') **AS** years,

**DATE\_FORMAT**(datetimeofcall, '%m') **AS** months,

SUM(cal.duration) as total\_duration

**FROM** call\_ as cal

**WHERE DATE\_FORMAT(**datetimeofcall, '%Y'**) =**2021

**GROUP BY** cal.call\_id, months ;

Now that I have 2 different "sub-arrays" I should obviously combine with join.

But what join? and which table do I use as a foreign key?

I will use the months as the reference and link column, which is the same in both tables so that the rows are matched correctly.

I also want an inner join because if there is a month that doesn't exist in the 2 years, I don't want to get a price

**SELECT** year\_2022.months,

((year\_2022.total\_duration - year\_2021.total\_duration) / year\_2021.total\_duration )\* 100 **AS**  total\_duration\_percentage\_change

**FROM** year\_2022

**INNER JOIN** year\_2021

**ON** year\_2022.months = year\_2021.months ;

|  |  |
| --- | --- |
| months | total\_duration\_percentage\_change |
| 01 | 35.0000 |
| 02 | -94.1935 |
| 06 | 33.3333 |
| 05 | 260.0000 |

The percentages come out a bit strange because of the values I have put in the tables, but computationally they are fine.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. 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)

From the question I understand that a grouping by city\_id and gender should be done. Nevertheless, it asks me for 3 columns where one will have citry \_id and the other 2 the average call duration by females and the average call duration by males. But GROUP BY allows us to aggregate **columns** per some category. Here I will essentially ask for Group By **by lines**. I can achieve this through a conditional expression (CASE) where it will examine whether each line belongs to female or male.

The logic is presented in more detail below:

I will get the city\_id through the customer table so that I can connect it through the contract table to the call table.

* call\_ ( call\_id, datetimeofcall, calledphonenumber, duration,contract\_id)
* contract (contract\_id, phone\_number, start\_date, end\_date, descrip, customer\_id,plan\_id)
* customer (customer\_id, first\_name, last\_name, dateofbirth, gender, city\_id)

**SELECT** cus.city\_id, cus.gender  **,**

AVG(cal.duration)

**FROM** contract as con

**INNER JOIN** customer as cus

**ON** con.customer\_id=cus.customer\_id

**INNER JOIN** call\_ as cal

**ON** con.contract\_id=cal.contract\_id

**WHERE DATE\_FORMAT**(cal.datetimeofcall, '%Y')=2022

**GROUP BY** cus.city\_id, cus.gender

|  |  |  |
| --- | --- | --- |
| city\_id | gender | AVG(cal.duration) |
| 2 | male | 4050.0000 |
| 5 | male | 360.0000 |
| 4 | female | 20.0000 |
| 5 | female | 540.0000 |

But we want three columns:

So I will have to find a way in the above query to do a grouping based on gender**, but after** the grouping is done based on city\_id.

Below I basically say:

1) Join the tables I will need (FROM, INNER JOIN, ON)

2) do a filtering on the rows, based on the date – I only want 2022(WHERE)

3) group me based on cus.city\_id only ( GROUP BY )

4) and now essentially through SELECT I will do another grouping on gender as well,

What you achieve this way is that actually since the SELECT is executed "after" the GROUP BY, I can group-break the AVG(cal.duration) column by gender .

Inside AVG I will put:

CASE Expression: The CASE expression is a conditional expression in SQL that allows you to evaluate conditions and return different values based on those conditions. It's often used for conditional logic within SQL queries.

This part of the query calculates the average call duration for "female" customers. It evaluates each row and includes the call duration in the calculation only if the customer's gender is "female." If the gender is not "female," it treats the value as NULL, effectively excluding it from the calculation.

**SELECT**

cus.city\_id,

**AVG(CASE WHEN** cus.gender = 'female' **THEN** cal.duration **END**) **AS** avgduration\_female,

**AVG(CASE WHEN** cus.gender = 'male' **THEN** cal.duration **END**) **AS** avg\_duration\_male

**FROM** contract as con

**INNER JOIN** customer as cus

**ON** con.customer\_id=cus.customer\_id

**INNER JOIN** call\_ as cal

**ON** con.contract\_id=cal.contract\_id

**WHERE DATE\_FORMAT**(cal.datetimeofcall, '%Y')=2022

**GROUP BY** cus.city\_id

|  |  |  |
| --- | --- | --- |
| city\_id | avg\_duration\_female | avg\_duration\_male |
| 2 | NULL | 4050.0000 |
| 5 | 540.0000 | 360.0000 |
| 4 | 20.0000 | NULL |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. 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)

* call\_ ( call\_id, datetimeofcall, calledphonenumber, duration,contract\_id)
* contract (contract\_id, phone\_number, start\_date, end\_date, descrip, customer\_id,plan\_id)
* customer (customer\_id, first\_name, last\_name, dateofbirth, gender, city\_id)
* City (city\_id, city\_name, population, mean\_income)

Essentially I want to calculate a table that looks something like this:

|  |  |  |
| --- | --- | --- |
| city\_id | duration\_ratio | population\_ratio |
| 1 | city\_total\_duration / total\_duration\_2022 | city\_population /  total\_population |
| 2 | … | … |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |

First I'm going to calculate the numerators of duration\_ratio and population\_ratio :

To do this I will create a subquery in which I will match the duration with the city\_id . So, basically I will create a table that will have city\_id and duration together, so that I can then easily join using city\_id as a key with the city table, which contains population inside.

I join these 2 tables with LEFT JOIN (LEFT JOIN and not INNER JOIN to include all cities regardless of whether they have data in the other table, i.e. I want all city\_ids to appear regardless of whether there were calls with duration in 2022 for them the cities) with the external query.

This is how the union is done inside the inner query:

* customer (customer\_id, first\_name, last\_name, dateofbirth, gender, city\_id)
* contract (contract\_id, phone\_number, start\_date, end\_date, descrip, customer\_id,plan\_id)
* call\_ ( call\_id, datetimeofcall, calledphonenumber, duration,contract\_id)

and through the internal query this essentially results:

|  |  |
| --- | --- |
| city\_id | duration |
| 2 | 4050 |
| 4 | 20 |
| 5 | 360 |
| 5 | 540 |

**CREATE VIEW** aritmites **AS**

**SELECT**

cit.city\_id,

**SUM(**citydur.duration) **AS** city\_total\_duration,

cit.population **AS** city\_population

**FROM** city as cit

**LEFT JOIN (**

**SELECT** cus.city\_id, cal.duration

**FROM** customer cus

**INNER** **JOIN** call\_ cal

**ON** cus.customer\_id = cal.contract\_id

**WHERE DATE\_FORMAT**(cal.datetimeofcall, '%Y') = '2022'

**)** **AS** citydur

**ON** cit.city\_id =citydur.city\_id

**GROUP BY** cit.city\_id, cit.population;

|  |  |  |
| --- | --- | --- |
| city\_id | city\_total\_duration | city\_population |
| 1 | NULL | 3154000 |
| 2 | 4050 | 173600 |
| 3 | NULL | 54567 |
| 4 | 20 | 12785 |
| 5 | 900 | 18926 |

Now I go and calculate the denominators total\_duration\_2022 and total\_population:

**create view** total\_duration\_2022 **as**

**SELECT SUM(**duration) **AS** total\_duration\_2022

**FROM** call\_

**WHERE DATE\_FORMAT(**datetimeofcall, '%Y') = '2022';

-- \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**create view** total\_population **as**

**SELECT SUM(**population) **AS** total\_population

**FROM** city;

And now I will join them:

I will use the IFNULL Function:

IFNULL(expr1, expr2): This function is used to handle null values in expressions. If expr1 is not NULL, it returns expr1; otherwise, it returns expr2.

it's used to prevent division by zero or null values. If the numerator or denominator is NULL, it substitutes it with 0 to avoid errors.

CROSS JOIN is used to combine each row from the first table (total\_duration\_2022) with each row from the second table (total\_population), resulting in a Cartesian product of the two tables.

I am creating a Cartesian product of the result sets from aritmites, total\_duration\_2022, and total\_population. This is a way to calculate ratios across all combinations of ar.city\_id, total\_duration\_2022, and total\_population.

The CROSS JOIN creates a result set with all combinations of rows from aritmites, total\_duration\_2022, and total\_population.

**SELECT**

ar.city\_id,

**IFNULL**(ar.city\_total\_duration /td. total\_duration\_2022, 0) **AS** duration\_ratio,

**IFNULL**(ar.city\_population / tp.total\_population, 0) **AS** population\_ratio

**FROM** aritmites **AS** ar

**CROSS JOIN** total\_duration\_2022 **as** td

**CROSS JOIN** total\_population **as**  tp ;

|  |  |  |
| --- | --- | --- |
| city\_id | duration\_ratio | population\_ratio |
| 1 | 0.0000 | 0.9239 |
| 2 | 0.8149 | 0.0509 |
| 3 | 0.0000 | 0.0160 |
| 4 | 0.0040 | 0.0037 |
| 5 | 0.1811 | 0.0055 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

import mysql.connector #database connection   
  
connection = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Er1997!",  
 database="assignment1"  
)

# a cursor is an object that allows you to interact with the database by executing SQL queries and fetching results

# it will return query results as dictionaries, making it easier to work with the data in Python.

if connection.is\_connected():  
 cursor = connection.cursor(dictionary=True)   
  
#the query without group by as the (k) query   
 query = """  
SELECT  
 cit.city\_id,  
 cus.city\_id AS customer\_city,  
 cal.duration,  
 cit.population  
FROM city AS cit  
LEFT JOIN customer AS cus ON cit.city\_id = cus.city\_id  
LEFT JOIN call\_ AS cal ON cus.customer\_id = cal.contract\_id  
WHERE DATE\_FORMAT(cal.datetimeofcall, '%Y') = '2022'  
 """

# the table that I get if I run this query in mysql

|  |  |  |
| --- | --- | --- |
| city\_id | duration | population |
| 2 | 4050 | 173600 |
| 4 | 20 | 12785 |
| 5 | 360 | 18926 |
| 5 | 540 | 18926 |

#executes the SQL query and retrieves the data into the rows variable as a #list of dictionaries.

cursor.execute(query)  
rows = cursor.fetchall()

# Initialize dictionaries for each city  
city\_data = {}  
total\_duration\_2022 = 0  
total\_population = 0  
  
for row in rows:  
 city\_id = row["city\_id"]  
 duration = row["duration"]  
 population = row["population"]  
  
 if city\_id not in city\_data:  
 city\_data[city\_id] = {"total\_duration": 0, "population": 0}  
  
 city\_data[city\_id]["total\_duration"] += duration  
 city\_data[city\_id]["population"] = population  
  
 if row["customer\_city"] is not None:  
 total\_duration\_2022 += duration  
 total\_population += population  
  
# Calculate and print the ratios  
for city\_id, data in city\_data.items():  
 city\_total\_duration = data["total\_duration"]  
 city\_population = data["population"]  
 duration\_ratio = city\_total\_duration / total\_duration\_2022  
 population\_ratio = city\_population / total\_population  
  
 print(f"City ID: {city\_id}, Duration Ratio: {duration\_ratio:.4f}, Population Ratio: {population\_ratio:.4f}")  
  
# Close the database connection  
cursor.close()  
connection.close()