**Assignment 1**

Students:

     Yara Yachnyk (4913329)

     Niels (3133998)

     Osaro (1168827)

     Máté (4395565)

     Fani (1240390))

**TASK 1: Database (re)design**

1. We added extra attributes to capture details that a trading company would realistically need for daily operations and analysis.  
   For bill we included **reference**, **holder**, and **description** so each invoice can be uniquely identified and clearly described.  
   For customer we added **age**, **salary**, and **nationality** to support demographic and financial analyses (for example, estimating purchasing power).  
   For partner we added **number of clients**, **annual revenue**, and **headquarters** to give a clearer picture of the size and reach of each partner company.
2. Values of Cardinalities are ChatGPT generated according to the assignment explanation. The foreign key should reference same unique values as the primary key in each relationship.
3. Schema chart (website to make this schema:

A screenshot of a computer

AI-generated content may be incorrect.

Schema of the Database (Website used for this schema: https://app.dbdesigner.net/designer/schema/new)

1. Cardinalities represent the number of rows (tuples) in each ratio, showing how many instances can exist for a given attribute. They also describe how certain columns (foreign keys) establish relationships between different tables. Thus, in segment – bill relation one segment can be responsible for many bills(giving one to many relationship), but each bill must belong to specific one segment; for customer – shareholder one customer can be linked with many shareholders (same one to many relationship); in customer—partner one partner can serve many customers, as well as one customer can work with many partners, in here we see many – to many relations cause customer-partner has two foreign keys and forms a composite primary key.

|  |  |  |
| --- | --- | --- |
| Relationship | Cardinality | Description |
| customer - shareholder | 1:N | a shareholder can have many customers |
| segment – bill | 1:N | a segment can have many bills |
| partner – customer\_partner | 1:N | one partner can have many customer\_partners |
| customer – customer\_partner | 1:N | one customer can have many customer\_partner entries |
| customer - partner | M:N | there is a many-to-many relation between the customer and partner tables, represented by the customer\_partner table in the design |

1. FK and PK are the foreign and primary keys:

1. FK bill.branch\_id – PK segment.branch\_id;

2. FK shareholder.customer\_id – PK customer.customer\_id;

3. FK customer\_partner.customer\_id – PK customer.customer\_id;

4. FK customer-partner.company\_name – PK partner.company\_name;

1. see attached SQL file
2. the chosen relation is the shareholder table:

shareholder(shareholder\_id, customer\_id, shareholder\_name, n\_of\_shares, amount)

shareholder\_id is the primary key, meaning every row is uniquely identified by shareholder\_id. Since no other attribute is a derived value, the only functional dependencies we have is shareholder\_id → customer\_id, shareholder\_name, n\_of\_shares, amount. Since the shareholder\_id is the only key candidate and it implies all other attributes, it is a superkey and thus the table shareholder is in BCNF.

**TASK  2: Querying the database**

1. Natural Language:

Retrieve the shareholder id, contract amount and contract description of each shareholder, along with the customer names and salary of customers these shareholders were working with.

Relational algebra:

πcustomerid, shareholderid, shareholdername,contractamount,contractdescription,salary((SH ××(σC.customerid=CP.customer\_id(C​ ×CP))𝜋customerid, shareholderid, shareholdername,contractamount,contractdescription,salary((SH ××(𝜎C.customerid=CP.customer\_id(C​ ×CP))

 SQL:

SELECT

c.customer\_id,

  sh.shareholder\_id,

  sh.shareholder\_name,

  cp.contract\_amount,

  cp.contract\_description,

  c.salary

FROM shareholder AS sh

JOIN customer AS c

  ON c.customer\_id = sh.customer\_id

LEFT JOIN customer\_partner AS cp

  ON cp.customer\_id = c.customer\_id;

1. Natural Language: Find the average salary of customers in each city.

Relational Algebra:

πcity, avg\_salary(γcity;AVG(salary)→avg\_salary(Customer))𝜋city, avg\_salary(𝛾cityAVGsalary→avg\_salaryCustomer)

SQL:

SELECT city, AVG(salary) AS avg\_salary

FROM customer

GROUP BY city;

1. Natural Language:

Find id’s of all shareholders who have at least one contract.

Relational Algebra:

πshareholderid(σsh.customerid=cp.customerid(SH ×CP))𝜋shareholderid(𝜎sh.customerid=cp.customeridSH ×CP)

SQL:

SELECT sh.shareholder\_id

FROM shareholder sh

WHERE EXISTS (

  SELECT \*

  FROM customer\_partner cp

  WHERE cp.customer\_id = sh.customer\_id );

**TASK 3: Data extraction and entity resolution using Python**

**3a**  
We first ensured the SQLite database file could be rebuilt and queried entirely from Python.  
Using sqlite3 we opened a connection and executed the SQL schema stored in Assignment\_\_1.sql to create the database file assignment1.db automatically:

See collab

After the build, SQL queries were executed from Python and results were fetched into pandas DataFrames.  
For example, to list the tables and preview customer data:

See collab

This demonstrates direct programmatic access to the database.

3b

To work with the data in Python, the complete customer table was imported into a pandas DataFrame:

See collab

To detect potentially duplicate customers we implemented the *overlap coefficient* between every pair of rows.  
The coefficient is defined as

#WRITE THE OVERLAP EQUATIONS

where AAA and BBB are the sets of attribute values (excluding the unique customer\_id).  
Records with an overlap greater than a chosen threshold (0.7) are flagged.