Database assignment

This assignment is composed out of four parts:

1. MS Access: SQL visual interface (10%);
2. SQL programming part A (30%)
3. SQL programming part B (40%)
4. Data modeling with MS Visio/DrawIO (20%).

You have to work together with the same group for all assignments.

With your solutions you have to include a (brief) "Who did what?" section, explaining who did what for the completion of the assignment. Please also indicate for which parts you used ChatGPT (or similar software) to support you in the coding. Feel free to use ChatGPT in a sensible way for all parts to improve your solutions. Part 3 contains a more unstructured question for which you can apply ChatGPT only indirectly.

For parts 2-4 of the assignment, the name of the database server you have to connect with via MS SQL Server Management Studio is ‘*uvtsql.database.windows.net*’. Your login and password can be found on the Canvas. You only have access to your group’s database.

The deadline for the assignment is *Monday, October 9, 23:59h*. You have to upload your solutions via Canvas. Be sure to collect all your files in a single *.zip or *.rar file.

Have fun,

Emiel

Part 1. MS Access

Movie case
The owner of a (very) classic movie rental company which rents older movies is concerned about the returns of rented movie. The owner believes that the late return of movies is creating a negative effect on the business. Since movies not being returned on the expected date affects their availability for future rentals, he would like you to investigate the rental and return records for the last three months and report what you have discovered. When a movie is not returned at the appropriate time, a late fee is assessed for each day late. The late fees will vary with the popularity of the movie. The owner wants you to determine the movies that are being returned late, the number of days late, and the late fees charged. He has an Access database that tracks store rentals. You will modify the database to answer the owner's questions. Do not worry about any possible errors in the data.

Database File Setup

1. Download the file named **Movie.accdb** and **save it.**

2. Rename the file as **Movie_Solution.accdb**.

3. Open the **Movie_Solution.accdb** file.

4. Open the **Movie table**.

5. Make a query that makes a list of all the unique movie titles with the word 'the' in the title. The list should be sorted alphabetically. Save the query as **UniqueMovieTitles**.

Rentals Report
You need to create a detailed query to calculate a couple of things, like the due date. You also need the query to determine the number of days late and the late fee.

1. Create a detail query named **Rentals Report** that you will use to calculate the date the movie is due to be returned, the number of days late and the rental late fee. Set a criterion that limits the query to only the most recent three months of rentals. You will need the following fields: **CustomerName** (from the Customer table), **MovieName** and **DailyLateFee**, **DateRented** and **DateReturned**.

2. Create a calculated field named **DateDue** to show the date that the movie should be returned. A movie may be rented for 7 days.

3. Create a calculated field named **DaysLate** to indicate the number of days the movie was late in being returned. If that value is negative, calculate **DaysLate** as zero, since the movie was returned before the **DueDate**.

4. Create a calculated field named **RentalLateFee** to calculate the late fee for each movie rented.

5. Sort by rental date and format the **RentalLateFee** as Currency. Save the query.

Totals Query

You need to create a totals query based on the **Rentals Report** query. You will group the totals query by the date rented to provide aggregate statistics in order to summarize the number of days late and the rental late fee.

1. Create a **Totals** query based on the **Rentals Report** Query. You are to summarize **DaysLate** and the **RentalLateFee** by **DateRented**. Save this query as **Rentals Summary**.

Part 2. SQL programming: MS SQL Server

In this part you have to write sql-code for Sales and Patstat database. A detailed description of the tables in this database is found in the document 'Sales_database.pdf', available on the Canvas.

General remarks

- First study the slides of *Sales_database.pdf*.
- Collect your queries in a single file labelled solution_groupX.sql.
- All queries should be written in lowercase.
- Each query starts with a comment, like '--Q1' or '--Q2a'.
- After each new query, use an empty line.
- Additional (useful) comments are appreciated.
- Use only one sql clause per line, e.g. 'select' on line 1, 'from' on line 2, etc.
- Try to use aliases for tables a much as possible. Use 'a' for the first table, 'b' for the second, and so on.
- Typically, use the 'join' clause for joining tables together.
- In the case of subqueries nested inside a larger query use indentation.
- Queries need to be developed for the general case of question.
- It is possible that some queries do not give any records back for the underlying data set.
- *Do not change or drop the original tables in your group's database*, because also your group members are working on the same database. Always work on temporary tables (starting with the "#" (hash) symbol) or create a new permanent table starting with your lastname. Temporary tables are similar to permanent tables, except temporary tables are stored in tempdb and are deleted automatically when they are no longer used (see https://technet.microsoft.com/en-us/library/ms177399(v=sql.105).aspx).

Part 2. Queries related to the sales database (tables start with x)

1. Find the names and salaries of employees with the letter 'a' in their name, or, end with the letter 'd' in their name, from the Marketing, Management, and Accounting department. Order the names alphabetically.
    a. Store the result in a temporary table called #xemp.
    b. Employees in the table #xemp with a salary above > 25,000 get an increase in salary of 10%. Update the table with a query to reflect this change.
    c. Write a query to see all records of the updated table.
    d. Finally, drop the temporary table you have created.

2. Find the Cartesian product of xsale and xitem table.
    a. Store the result in a tempory table #cartesian_temp.
    b. Remove all records from this temporary table, where the item_color is 'bamboo', or where the deptname is 'books'.
    c. After that, show all the remaining records in this table.
    d. Finally, drop the temporary table you have created.

3. List the names of green items sold by no department on the first floor. Do not show duplicates. Solve this question with three different queries:
    a. Create a query with an 'in' clause and a subquery.
    b. Create a query with an 'exists' clause and subquery.
    c. Create a query with a 'join' clause.

4. Do the following:
    a. Find the average salary of employees in the 'marketing' department. Store the results in a temporary table #avg_marketing_table with two columns 'avg_salary' and 'id' with value 1.
    b. Find the average salary of employees in the 'purchasing' department. Store the results in a temporary table #avg_purchasing_table with two columns 'avg_salary' and 'id' with value 1.
    c. Find the absolute difference between the average salaries between the marketing and the purchasing department by joining the two temporary tables of a) and b). The name of the result column is 'dif_salary'.
    d. Store the result of c) in a temporary table #dif_salary_table.
    e. Drop all the temporary tables created in 4).

5. Create queries that answer the following questions (do not use temporary tables):
    a. Whom does Alice manage? Order the result alphabetically by employee name.
    b. What is the salary of the manager of Nancy's manager?
    c. Write a query that answers the question: '*Are there employees that are in a different department compared to their manager's department*'? The query gives a list of employees, their department, their manager, and the manager's department.

   d. Find numbers and names of those employees, who make more money than their manager. Include, the difference in salary in the query.
   e. List the name, salary, and manager of the employees of the Accounting department who have a salary over €20,000.

6. List the suppliers that deliver at least 6 unique items. Include a calculated column ‘total’ to show the total of (unique) items. Order the result set on the field ‘total’ in an ascending order.

7. Make a single query that (inner) joins all 6 tables together starting with an x, based on the provided data model (see Sales database schema). Only show the columns from the tables xemp and xitem for the department 'Marketing' and the itemname 'Compass' and do not show duplicate records.

8. Count the number of direct employees of each manager. Include the employee number, employee name, and the calculated column ‘direct_employees’ in the query. Now determine the manager with the lowest number of direct employees, with a second query.

9. List the names of the employees who earn more than any employee in the Books department, order the result by salary from low to high
   a. Store the result in a temporary table.
   b. Add a salary ranking to a copy of the temporary table, where 1 is the highest salary. When salaries are the same, they get the same rank number.
   c. Create a query on the previous query, where you only show the top 5 salaries.
   d. Drop all the tables you have created.

10. Among all the departments with an average salary greater than €15,000, find the departments that sell ‘Pith helmet’ or ‘sextant’. Solve this question with two different solutions:
    a. Create a query with an ‘intersect’ clause.
    b. Create a query with an ‘in’ clause and a subquery.

11. Find the suppliers that do not deliver ‘Tent - 2 person’.
    a. Create a query with a subquery.
    b. Create a query with a join clause.

12. Write a query that lists all columns from the xitem table, together with a new colunn ‘itemname_firstword’, that gives only the first word in the itemname column.
    a. You have to create a function ‘`getFirstWord`’ for this.
    b. Now use ChatGPT to create the function `getFirstWord` for you.
    c. What is the question you have to use in ChatGPT for this purpose?

13. Which department(s) has (have) the lowest average salary? Report in your query the deptname and the average salary. Hint. Use a temporary table in your solution.

14. List the items delivered by ‘All sports manufacturing’ *or* sold in the Navigation department. Solve this question with two different solutions:
    a. Use a ‘union’ clause in your query.
    b. Do not use a union clause but a logical operator in your (sub)query.

15. List the unique items delivered by ‘All sports manufacturing’ that are not sold in the Navigation department. Solve this question with two different solutions:
    a. Use an ‘except’ clause in your query.
    b. Do not use an except clause but a logical operator in your (sub)query.

16. Is it true that all the departments that sell items of type C are located on the third floor? The result of this query can be a ‘1’ or ‘0’ simulated with a count, meaning ‘yes’ or ‘no’. In addition, the count can be translated to a ‘yes’ or a ‘no’ with a ‘case’ statement. Preferably, create a single query.

17. Write a query that finds the items delivered to all departments. Exclude the administrative departments from your query.

18. Suppose that the table xsale_copy is a table with the same structure as table xsale. However, its data (i.e., its records) might be different compared to the table xsale. Write a query that checks if these two tables have identical data (i.e., records) or not. Your query should return all rows that are in one table but not in the other, and 0 rows if the tables have the same data.

Part 3. Queries related to tables of the Patstat tables

19. In the table 'patstat_golden_set', publications are listed and assigned to clusters (1-100).
    a. Write a query that counts the number of publications per cluster_id and stores the result as the calculated column ‘n_pubs’. List both the cluster_id and the total number of publications in your result. Order the query on n_pubs in a descending way.
    b. Now extend the query in 20a in such a way, that it also shows the count per cluster_id as a percentage from the total number of publications in the table, with a new calculated column ‘probability’. Do not use a temporary table.
    c. Now store the result of the query of 20b in a temporary table named ‘#result’.
    d. Now write a query that adds a new column ‘nomalized_n_pubs’ to the table #result, where n_pubs is *normalized* by the z-score, z = (x-μ)/σ. So, in the column ‘nomalized_n_pubs’ the n_pubs values are expressed in terms of standard deviations from the mean.
    e. Drop the temporary table.

20. The table ‘patstat_golden_set contains name variants of scientific publications from the Patstat database. In addition, it contains 100 perfect clusters of name variants, which are the result of an unknown clustering algorithm, For example, cluster_id ‘100’ contains 1,590 name variants of the publication: ‘*KOHLER G., and MILSTEIN C., Nature, Vol. 256, August 7, 1975, pp. 495-497, Continuous Cultures of Fused Cells Secreting Antibody of Predefined Specificity’*.

    In this part of the assignment, *you have to create your own sql-script that can cluster name variants of scientific publications, when the cluster_id’s are not given*. The input of the script is a set of unclustered name variants of scientific publications. And the output of the procedure is a table ‘groupx_patstat_clusters’ with *clusters of name variants* (records) for each, unique scientific entity. Of course, the golden cluster_id’s could be used as a benchmark to access the quality of your script. Your script should focus on providing ‘the best possible results’ for the clusters, because perfect results might be difficult to obtain. In addition, you can validate the quality of your clusters by comparing the results with the goldens clusters with *precision-recall-f1 analysis*.

    To solve this question, you have quite some degrees of freedom. You use all the knowledge obtained from the classes and additionally you can find relevant sources of knowledge on the internet.

    Hints (for Question 20)

    You are free to come up with your own algorithmic design, but you can also work via these hints:

1. Work on a sample (say 1000 records or so) and later on the complete set.
2. Data pre-processing (to harmonize the data).

   a. Pre-clean the data by removing leading and trailing spaces.
   b. Remove multiple white spaces.
   c. Remove diacritics, remove (irrelevant) interpunction.
   d. Etc.

3. Use of helper functions.
   a. Write a function that removes multiple spaces from a string, which is a common procedure in data cleaning. This means that all multiple spaces are replaced by a single space. The name of the function should be funcDeleteMultipleSpaces. The first line of the function might be:
      `Create function funcDeleteMultipleSpaces (@inputstring nvarchar(1024))`.
      It might be convenient to use a *while* construct in your solution.
      The result of your function should be tested on the table Patstat, field 'npl_biblio'.
   b. Write a function cleanIt to remove unwanted characters from a string. This function could be considered to be a generalization of the function developed in the previous question to other characters. The syntax for the cleanIt function is: `cleanIt (StringInput, InvalidCharacters)`. StringInput is the string to be clean up from invalid characters and InvalidCharacters are the range of characters which you don't want to be included in your StringInput.
      Examples are:

```
cleanIt('3567576Q7XDEWC6A871','%[^1-3A-D]%'); --would return '3DCA1'
cleanIt('3567576Q7XDEWC6A871','%[A-Z]%');  --would return '356757676871'
cleanIt('3567576Q7XDEWC6A871','%[0-9]%');  --would return 'QXDEWCA'
cleanIt ('64@#7*&*^6^$%Q7C6A871','%[^0-9A-Z]%'); --would return '6476Q7C6A871'
cleanIt('64@#7*&*^6^$%Q7C6A871','%[0-9A-Z]%'); --would return '@#*&*^^$%'
cleanIt('the  fox   ran into the forest   following         other   foxes!','% %'); --
would return 'thefoxranintotheforestfollowingotherfoxes!' [note: remove single space]
cleanIt('the  fox   ran into the forest   following         other   foxes!','%  %'); --
would return 'the fox ran into the forest following other foxes!'  [note: remove doubles
spaces]
```

      The result of your function should be tested on the table Patstat, field 'npl_biblio', where you remove all interpunction.
   c. Etc.

4. Extract some bibliographic meta data from scientific references and store the result in helper table:
   a. Journal name;
   b. Volume;
   c. Issue;
   d. Edition;
   e. Pages (start and end);
   f. Publication year;
   g. Publication month (date);
   h. ISSN and ISBN;
   i. DOI;

   j. XP number (a unique number used by Patstat sometimes) *
   k. …;
5. Make rules and clusters to identify duplicates.
   a. In this step, you have to determine the name variants and put them into unique clusters.
   b. A possible approach is to create *pairs of records* that are likely to be name variants based on simple rules. Those records are in some aspect(s) alike. Pairs are found by comparing records to each other according to some similarity measure defined by the rule. The most basic similarity measure is a comparison of one of the bibliographic meta data labels. For example, all pairs of records that match on their ISBN, pages_start, pages_end, volume, issue, d_year, and d_month, attributes may be considered as a useful pair. Such pairs can be found by joining the table (with the bibliographic meta data) with itself.
   c. You could make a number of simple rules to detect similarity between pairs and give some score value to the rules. Later you could add up the scores for the pairs and consider pairs for clustering which are *above some threshold value*, i.e. the combination of rules could function as a distance measure.
6. Data post-processing
   a. In the post-processing, records for which no duplicates are detected can be assigned to new, single record clusters.
   b. Finally, you store your results in a table with the name *'groupx_patstat_clusters*', with a *unique cluster number* for groups of name variants and the original identifiers.
7. Validation
   a. You validate the correctness of your clusters by comparing the results with the goldens clusters with *precision-recall-f1 analysis* (see https://en.wikipedia.org/wiki/Precision_and_recall). These values can be computed per cluster, but also as an average value for all clusters.
   b. Optional, you can repeat the steps, make changes to settings, create new clusters, and subsequently improve (optimize) the precision-recall-f1 values.

Part 4. Data modelling: MS Visio/Draw.IO & SQL server

The picturesque city of Tilburg, wants to maintain information about its extensive system of high schools, including its teachers and their (university) degrees, its students, administrators, and the subjects that it teaches.

Each school has a unique name, plus an address, telephone number, year built, and size in square meter. Students have a student number, name, home address, home telephone number, email, current grade, and age. Regarding a student's school assignment, the school system is only interested in keeping track of which school a student currently attends. Each school has several administrators, such as the principal and assistant principals. Administrators are identified by an employee number and also have a name, telephone number, and office number.

Teachers are also identified by an employee number and each has a name, age, subject specialty such as English (assume only one per teacher), and the year that they entered the school system. Teachers tend to periodically move from school to school and the school system wants to keep track of the history of which schools the teacher has taught in, including the current school. Included will be the year in which the teacher entered the school and the highest pay rate that the teacher attained at the school. The school system wants to keep track of the universities or other higher education that each teacher attended, including the degrees earned and the years in which they were earned. The school system wants to record each university's name, address, year founded, and Internet URL (address.) Some teachers, as department heads, supervise other teachers. The school system wants to keep track of these supervisory relationships but only for teachers' current supervisors.

The school system also wants to keep track of the subjects that it offers (e.g., English, Dutch, Mathematics, etc.). Each subject has a unique subject number, a subject name, the grade level in which it is normally taught, and the year in which it was introduced in the school system. The school system wants to keep track of which teacher taught which student which subject, including the year this happened and the grade received.

Remarks

- You can use MS Visio or another tool like Draw.io, but the result should look more or less like the examples in the sheets.
- Attributes in the Chen model are represented outside the entity to which they belong. We don't specify any connectivity constraint between entity and attribute. In the Crow's feet model, the attribute is represented inside the entity box.
- Although we did not talk about that in class, attributes can also be attached to relationships. For example, in the hypothetical case that a 'customer' wants to rent a 'car', then the 'rental date' and 'return date' can be modeled as attributes of the relationship 'rented' (in some ER dialects, the relationship must be turned into a special relationship called "association" before you can attach attributes).
- The connectivity in Chen models is typically represented by a 'N', 'M', or '1' symbol connected to the link between the relationship and the participating entity. Instead of 'N', 'M', you may also use a Crow Foot symbol.
- In the Crow's Feet model, specify not only the connectivity of a relationship but also the lower boundary (0 or 1), like in the examples in the sheets.

- Many relational database systems do not support generalization. You can still represent the situation by having, for example, a table for Student and a table for Person. Within the Student table, you have the student-specific attributes, plus a reference to the Person table, where her other attributes are specified. If your drawing tool does not support it, then make a relationship between the subtype and the supertype and call it "isa".

Questions

1. Draw an Entity-Relationship diagram using *Chen's Notation*.

2. Draw an Entity-Relationship diagram using *Crow's Foot Notation*.

3. Implement the tables in MS SQL Server in your group's database with create table statements. Be sure to use the appropriate data types and keys.

4. Insert some records for all tables, and

    a. implement a query which runs over multiple tables that answers a (business/organization) question of your choice. Store the query result in a temporary table with the name '#business_question'.
    b. after that drop *all* the tables you have created.