**SQL TRAINING**

[SQL Training - Steps to create your own project space - Overview (azure.com)](https://dev.azure.com/goazrsm01/data_engineering_SQL_training_2023/_wiki/wikis/data_engineering_SQL_training_2023.wiki/7/SQL-Training-Steps-to-create-your-own-project-space)

**Instructions**

**Azure Portal**

* Deploy Custom Template
* Template Spec
  + Sub: GO-SM-Data Platforms Engineering
  + RG: data\_engineering\_SQL\_training\_2023
  + Region: North Europe
  + SQL Server Name: sqlserversimon
  + SQL Admin Login: adminsqlsimon
  + Password: <Set Password>
  + Database Name: sqltrainingsimon
* Go to Resource Groups
* SQL Server
  + Networking
  + + Add client IP address (to add your IP)
  + Save
* Home (portal.azure.com)
* Azure SQL
  + Search for the Database and click it (sqltrainingsimon (sqlserver/sqltrainingsimon))
* Query Editor (Preview)
  + Type Password and Click OK
  + Check tables have populated

**Azure Boards**

* Open Template [link](https://dev.azure.com/goazrsm01/data_engineering_SQL_training_2023/_workitems/create/Epic?templateId=a2317bc3-d868-47d6-b7fc-fc2c42d7b72c&ownerId=7e5f5de2-2681-42eb-a1c0-aacc58ef474e) for new Work Item
* Enter Title and save
* To create your own Feature, Click on "Work items" and search "SQL Questions".
* Ensure that new Feature popup window opens. Click on three dots and choose "Create copy of work item"
* Copy Work item window pops up. Check the button to include child work items and click on Ok.
* A new copy of Feature is created. Rename the Feature title in the format "SQL Questions - <yourname>"
* To add this feature to your epic, click on "Add an existing work item as a parent" under Related Work section.
* Search for your Epic name and choose from the drop down list. Click on OK.
* Check if your Epic has been added and click on Save & Close.

**Repos**

* Create Branch on DevOps
  + Link to user\_story #
* Open branch and click Clone, copy URL
* Create new folder on local device
* Open Visual Studio Code, open folder, open terminal
* Type: git clone <URL>
* Type: cd data\_engineering\_SQL\_training\_2023
* Type: git checkout user\_story
* Type: git add file.sql
* Type: git commit -m “message”
* Type: git push
* Go to Branch in Repos, click Create Pull Request

**Question 1**

TABLE departments

id INTEGER PRIMARY KEY NOT NULL,

name VARCHAR(50) NOT NULL

TABLE employees

id INTEGER PRIMARY KEY NOT NULL,

departmentId INTEGER NOT NULL REFERENCES departments(id),

name VARCHAR(50) NOT NULL

Write a sql query to populate empty table departmentReport, defined below, with the total number of employees in each department.

INSERT INTO departmentReport (departmentName, employeeCount)

SELECT d.name, COUNT(e.id) AS employeeCount FROM departments d LEFT JOIN employees e on d.id = e.departmentID

GROUP BY d.name;

**Question 2**

Customers and their orders are stored in the following two tables:

TABLE customers

id INTEGER PRIMARY KEY NOT NULL,

name VARCHAR(50),

balance DECIMAL(10,2)

TABLE orders

id INTEGER PRIMARY KEY NOT NULL,

customerId INTEGER NOT NULL REFERENCES customers(id),

product VARCHAR(100)

Using a subquery, delete the orders of any customer whose balance is negative.

DELETE FROM orders WHERE (SELECT id FROM customers WHERE balance < 0);

**Question 3**

Given the following definition, write a query that selects all customers together with the total number of transactions that they made. Customers without transactions should be included as customers with 0 transactions.

TABLE customer\_tr

id INTEGER NOT NULL PRIMARY KEY,

name VARCHAR(30) NOT NULL

TABLE transactions  
id INTEGER NOT NULL PRIMARY KEY,

customerID INTEGER REFERENCES customer\_tr(id),

amount DECIMAL(15,2) NOT NULL

SELECT c.name AS "Name", COUNT(t.id) AS "Number of Transactions" FROM customer\_tr c LEFT JOIN transactions t ON c.id = t.customerID

GROUP BY c.name;

**Question 4**

Below are the tables of colleges, students and their rankings:

TABLE colleges

id INTEGER PRIMARY KEY,

name VARCHAR(50) NOT NULL

TABLE students

id INTEGER PRIMARY KEY,

name VARCHAR(50) NOT NULL,

collegeId INTEGER REFERENCES colleges(id)

TABLE rankings

studentId INTEGER REFERENCES students(id),

ranking INTEGER NOT NULL,

year INTEGER NOT NULL

Write a query that lists all colleges that have at least one student with a ranking between 1 and 3 (both inclusive), for the year 2015. The query should return:

* The college name
* The rank of their best ranking student for 2015
* The number of students who had rankings between 1 and 3 (both inclusive) for the year 2015.

Rank 1 is the best rank, rank 2 the second-best and so on. More than one student can tie for a rank in a year.

SELECT c.name, MIN(r.ranking) AS "Best Ranking 2015", COUNT(\*) AS "Number of Students Ranked 1 to 3" FROM colleges c INNER JOIN students s ON c.id = s.collegeId INNER JOIN rankings r ON s.id = r.studentId WHERE r.year = 2015 AND r.ranking BETWEEN 1 AND 3

GROUP BY c.name

HAVING COUNT(\*) > 0;

**Question 5**

Car companies are displaying their cars at an auto show. Each company displays at at least one car. The auto show also holds various events on which different cars are compared according to the selected features.

Given below is the data definition of tables that hold data for companies, their cars, events and participation:

TABLE companies

id INTEGER PRIMARY KEY,

name VARCHAR(40) NOT NULL

TABLE cars

id INTEGER PRIMARY KEY,

name VARCHAR(40) NOT NULL,

companyId INTEGER NOT NULL REFERENCES companies(id)

TABLE events  
id INTEGER PRIMARY KEY,

name VARCHAR(40) NOT NULL

TABLE eventParticipants  
carId INTEGER NOT NULL REFERENCES cars(id),

eventId INTEGER NOT NULL REFERENCES events(id),

PRIMARY KEY(carId, eventId)

Write a query that returns:

* The names of car companies.
* Total number of their cars that will **not** be displayed at any event.

SELECT c.name AS "Company Name", COUNT(cars.id) AS "Cars Not Displayed" FROM companies c LEFT JOIN cars ON c.id = cars.companyId LEFT JOIN eventParticipants ep ON cars.id = ep.carId WHERE ep.eventId IS NULL

GROUP BY c.name;

**Question 6**

The following two tables are used to define departments and their employees:

TABLE departments\_sal

id INTEGER PRIMARY KEY,

name VARCHAR(50) NOT NULL

TABLE employees\_sal

id INTEGER PRIMARY KEY,

name VARCHAR(50) NOT NULL,

salary INTEGER NOT NULL,

departmentId INTEGER REFERENCES departments\_sal(id)

The human resource department of the company needs a report of the average salary per department.

A department's average salary is equal to the total salaries of all the employees in the department divided by the total number of employees in that department. The company's average salary is calculated for all employees irrespective of their department.

Write a query that returns the following three columns:

* The department's id
* The average salary of the department. Departments with no employees should return the average salary as 0.
* The column 'status' that can have one of the following values:
  + 'Above' when a department's average salary is above the company's average salary.
  + 'Below' when a department's average salary is below company's average salary.
  + 'Equal' when a department's average salary is equal to company's average salary.

AVERAGES:

SELECT es.departmentId, SUM(es.salary)/COUNT(es.id) AS "Average Salary" FROM employees\_sal as es

GROUP BY es.departmentId;

SELECT es.departmentId, AVG(es.salary) AS "Average Salary" FROM employees\_sal as es

GROUP BY es.departmentId;

FINAL VERSION:

WITH DepartmentAverages AS (

  SELECT

    d.id AS departmentId,

    COALESCE(SUM(e.salary)/COUNT(e.id), 0) AS departmentAvgSalary

  FROM departments\_sal d

  LEFT JOIN employees\_sal e ON d.id = e.departmentId

  GROUP BY d.id

),

CompanyAverage AS (

  SELECT COALESCE(SUM(e.salary)/COUNT(e.id), 0) AS companyAvgSalary

  FROM employees\_sal e

)

SELECT

  d.id AS departmentId,

  da.departmentAvgSalary AS departmentAvgSalary,

  CASE

    WHEN da.departmentAvgSalary > ca.companyAvgSalary THEN 'Above'

    WHEN da.departmentAvgSalary < ca.companyAvgSalary THEN 'Below'

    ELSE 'Equal'

  END AS status

FROM departments\_sal d

LEFT JOIN DepartmentAverages da ON d.id = da.departmentId CROSS JOIN CompanyAverage ca;

**Question 7**

Employee data of an organization is defined below:

TABLE employees\_hier

id INTEGER PRIMARY KEY,

name VARCHAR(255) NOT NULL,

managerId VARCHAR(255)

Each employee has either one manager or none. Managers can also have managers and the hierarchy can extend multiple levels. Top-level managers have no managers above them (managerId is *null*). All employees whose managerId is *null*are top-levelmanagers regardless of whether they have employees under them or not.

Write a query that returns:

* The top-level manager's id.
* Total number of employees under them.

Any top-level managers who have no employees under them should not be returned.

SELECT sup.id, COUNT(sub.id) AS "Number of Employees" FROM employees\_hier sup JOIN employees\_hier sub ON sub.managerId = sup.id

GROUP BY sup.id;

**Question 8**

Information about people and their parents are stored in the following table:

TABLE people

id INTEGER NOT NULL PRIMARY KEY,

motherId INTEGER REFERENCES people(id),

fatherId INTEGER REFERENCES people(id),

name VARCHAR(30) NOT NULL,

age INTEGER NOT NULL

Write a query that selects the names of all parents together with the age of their youngest child.

SELECT p1.name as "Parent Name", MIN(p2.age) AS "Age of Youngest"

FROM people as p1 LEFT JOIN people as p2 ON p1.id = p2.motherId OR p1.id = p2.fatherId

WHERE p2.age IS NOT NULL

GROUP BY p1.id, p1.name;

**Question 9**

A movie streaming app is currently being developed by a start-up. A movie can belong to one or more genres. Given below is the table definitions for movies and genres:

TABLE genres

id INTEGER PRIMARY KEY,

name VARCHAR(50) NOT NULL

TABLE movies  
id INTEGER PRIMARY KEY,

name VARCHAR(50) NOT NULL

TABLE moviesGenres  
movieId INTEGER REFERENCES movies(id),

genreId INTEGER REFERENCES genres(id),

PRIMARY KEY(movieId, genreId)

A particular search parameter requires all movies which belong to more than two genres and none of the genres are 'Comedy'.

Write a query for the above criteria that returns:

* The movie's name.
* The number of genres the movie belongs to.

SELECT m.name AS "Movie Name", COUNT(DISTINCT mg.genreId) AS Number\_of\_Genres

FROM movies m

JOIN moviesGenres mg ON m.id = mg.movieId

WHERE mg.genreId <> (SELECT id FROM genres WHERE name = 'Comedy')

GROUP BY m.id, m.name

HAVING COUNT(DISTINCT mg.genreId) > 2;

**Question 10**

Consider the following table definition:

TABLE students\_scr

id INTEGER PRIMARY KEY,

name VARCHAR(255) NOT NULL,

score INTEGER NOT NULL,

class INTEGER NOT NULL

Write a query that **efficiently**with respect to time used, returns the result with the name, score and class of all the students who scored highest in their respective classes.

CREATE INDEX idx\_class ON students\_scr(class);

CREATE INDEX idx\_score ON students\_scr(score);

WITH RankedStudents AS (SELECT id, name, score, class, RANK() OVER (PARTITION BY class ORDER BY score DESC) AS rank FROM students\_scr)

SELECT name, score, class FROM RankedStudents

WHERE rank = 1;