3. Introductory SQL

The shown tables contain the query answers for the database instance you can create with the script **UniversityDB.**

3.2.1 SELECT with MAX aggregation

Find the highest salary of any instructor:

```
SELECT MAX(Salary) FROM Instructor;
```

95000.00

3.2.2 Scalar Subquery

Find ID and Name of all instructors earning the highest salary. There might be more than one with the same salary:

```
SELECT InstID, InstName FROM Instructor
WHERE Salary =
(SELECT MAX(Salary) FROM Instructor);
```

InstID	InstName
22222	Einstein

3.2.3 DELETE using IN

Delete courses BIO-101 and BIO-301 in the Takes table:

```
DELETE FROM Takes WHERE CourseID IN ('BIO-101',
'BIO-301');
```

After this question run the UniversityDB script to restore tables to initial instances.

3.2.4 Advanced WHERE condition using NOT IN

Find those students who have not taken a course.

```
SELECT StudID FROM Student WHERE StudID
NOT IN (SELECT StudID FROM Takes);
```

70557

3.2.5 Advanced WHERE condition using ALL

Find the name(s) of those department(s) which have the highest budget, i.e. a budget which is higher than or equal to those for all other departments:

```
SELECT DeptName FROM Department
WHERE Budget >= ALL (SELECT Budget FROM
Department);
```

Finance

3.2.6 Advanced WHERE condition using SOME

Find the names of those students who have the same name as some instructor. Use the SOME operator for this:

```
SELECT DISTINCT StudName
FROM Student
WHERE StudName = SOME (SELECT InstName
FROM Instructor);
```

Brandt

Make another statement querying the same, but without using SOME:

```
SELECT DISTINCT StudName
FROM Student, Instructor
WHERE StudName = InstName;
```

3.2.7 Create and populate table GradePoints(Grade, Points) to provide a conversion from letter grades to numeric scores such that

SELECT * FROM GradePoints;

gives:

Grade	Points
A	4.0
A-	3.7
B	3.0
B+	3.3
B-	2.7
C	2.0
C+	2.3
C-	1.7
D	1.0
D+	1.3
D-	0.7
F	0.0

This shows that an "A" corresponds to 4 points, an "A-" to 3.7 points, and so on.

```
CREATE TABLE GradePoints (
Grade VARCHAR(2) PRIMARY KEY,
Points DECIMAL(3,1));


INSERT INTO GradePoints VALUES
('A', 4.0),  ('A-', 3.7),  ('B+', 3.3), ('B', 3.0), ('B-', 2.7),
('C+', 2.3), ('C', 2.0), ('C-', 1.7), ('D+', 1.3), ('D', 1.0),
('D-', 0.7), ('F', 0.0);
```

The Grade-Points earned by a student for a course offering (section) is the number of credits for the course multiplied by the points for the grade that the student received.

The Total Grade-Points earned by a student is the sum of grade points for all courses taken by the student.

3.2.8 Find the Total Grade-Points earned by each student who has taken a course. Hint: use GROUP BY.

```
SELECT StudID, SUM(Credits * Points)
FROM (Takes NATURAL JOIN Course)
NATURAL JOIN GradePoints
GROUP BY StudID;
```

StudID	TotalGradePoints
00128	27.1
12345	48.0
19991	9.0
23121	6.9
44553	10.8
45678	22.2
54321	28.0
55739	11.1
76543	28.0
76653	6.0
98765	15.8
98988	16.0

3.2.9 Find the Total Grade-Points Average (GPA) for each student who has taken a course, that is, the total grade-points divided by the total credits for the associated courses. Order the students by falling averages. Hint: use **GROUP BY.**

```
SELECT StudID, SUM(Credits*Points) / SUM(Credits)
AS GPA
FROM (Takes NATURAL JOIN Course)
        NATURAL JOIN GradePoints
GROUP BY StudID
ORDER BY GPA DESC;
```

StudID	GPA
76543	4.00000
98988	4.00000
00128	3.87143
55739	3.70000
54321	3.50000
12345	3.42857
19991	3.00000
44553	2.70000
23121	2.30000
98765	2.25714
45678	2.01818
76653	2.00000

3.2.10 Query using UNION

Now modify the queries from the two previous questions such that students who have not taken a course are also included.

Hint: use the UNION operator.

```
(SELECT StudID, SUM(Credits * Points)
FROM (Takes NATURAL JOIN Course)
NATURAL JOIN GradePoints
GROUP BY StudID
)

UNION

(SELECT StudID, 0 AS GPA
FROM Student
WHERE StudID NOT IN  (SELECT StudID FROM Takes)
) ;
```

```
(SELECT StudID, SUM(Credits*Points) / SUM(Credits)
AS GPA
FROM (Takes NATURAL JOIN Course)
        NATURAL JOIN GradePoints
GROUP BY StudID)

UNION

(SELECT StudID, NULL AS GPA
FROM Student
WHERE StudID NOT IN  (SELECT StudID FROM Takes)
)

ORDER BY GPA DESC ;
```

3.2.11 Testscores example.

Create a relation schema Testscores (by a table declaration) and insert values such that

SELECT * FROM Testscores;

gives:

Student	Test	Score
Brandt	A	47
Brandt	B	50
Brandt	C	NULL
Brandt	D	NULL
Chavez	A	52
Chavez	B	45
Chavez	C	53
Chavez	D	NULL

```
DROP TABLE IF EXISTS Testscores;

CREATE TABLE Testscores (
Student VARCHAR(20) NOT NULL,
Test VARCHAR(20) NOT NULL,
Score INT,
PRIMARY KEY (Student, test) );

INSERT INTO Testscores values
('Brandt', 'A', 47), ('Brandt', 'B', 50),
('Brandt', 'C', NULL), ('Brandt', 'D', NULL),
('Chavez', 'A', 52), ('Chavez', 'B', 45),
('Chavez', 'C', 53), ('Chavez', 'D', NULL);

SELECT * FROM Testscores;
```

Then find the maximal score for each student who has an average larger than 49.

```
SELECT Student, MAX(Score)
FROM Testscores
GROUP BY Student HAVING AVG(Score) > 49;
```

Then find those students for whom some score is unknown.

```
SELECT DISTINCT Student FROM Testscores
WHERE Score IS NULL;
```

Student
Brandt
Chavez