3.3 Session 3 reflection

Combining skills acquired in the first three sessions, write queries (on the data of your choice that comprises at least two tables) that contain the requested elements and make sense on your data.

Please show the exact SQL query together with its result in your response. Each step is worth one point for adequate completion. Additionally, please include in the first response the commands that create and populate the database as well as a brief description of what it represents. You are welcome to reuse parts of what you created for the previous reflections.

1. A query that contains a join of any kind.

SELECT \* FROM grades JOIN students USING (StudentID) ORDER BY firstname ASC;

Graphical user interface, application, Teams

Description automatically generated

1. A query that contains a join of any kind and filtering into two or more cases (for example students who pass or fail, from the example data or a variant thereof).

SELECT studentid, course, grade, CASE WHEN grade > 80 then 'Pass' WHEN grade < 80 THEN 'Fail' END as Final\_grade FROM grades;

Table

Description automatically generated

1. A query that contains a join of any kind, an ordering of some sort, and a cut-off to show only a part of the results (for example the names of the students who hold the best five grades in the example data).

SELECT \* FROM grades FULL OUTER JOIN students USING (StudentID) WHERE Grade > 80 ORDER BY grade DESC Limit 5;

A picture containing funnel chart

Description automatically generated

1. A query that groups the results of a join by one or more columns.

SELECT StudentID, FirstName, COUNT(Grade) AS total FROM students LEFT JOIN grades USING (StudentID) GROUP BY StudentID ORDER BY total ASC;

Graphical user interface, application, table, Teams

Description automatically generated

1. A query that counts matches, possibly by groups.

SELECT course, COUNT(Grade) AS Total\_Students FROM students FULL OUTER JOIN grades USING (StudentID) GROUP BY Course;

Application, Teams

Description automatically generated with medium confidence

1. A query that sums over a column, possibly by groups.

SELECT grades.course, SUM(grade) AS total FROM grades JOIN students USING (StudentID) GROUP BY ( grades.course);

Table

Description automatically generated

1. A query that averages over a column, possibly by groups.

/\* student GPA \*/

SELECT FirstName, AVG(Grade) FROM students JOIN grades USING (StudentID) GROUP BY StudentID;

Table

Description automatically generated

1. A query that uses a join and null fields to gain useful insights (such as how many students have not completed any courses, per program, in the example data).

SELECT \* FROM students FULL OUTER JOIN grades USING (StudentID) WHERE Grade IS NULL;

Background pattern

Description automatically generated

1. Describe a situation in which a cross join could be useful or, failing to come up with one, discuss why that tends to be an impractical thing to compute. (You are welcome to discuss the latter even if you come up with the former.)

SELECT firstname, programname, grade, CASE WHEN grade > 80 and programname='IT' then 'Pass' WHEN grade < 80 THEN 'Fail' END as Final\_grade FROM grades CROSS JOIN students ORDER BY programname DESC;

Graphical user interface, table, Teams

Description automatically generated

DATA BASE

CREATE TABLE students(StudentID integer, FirstName VARCHAR(20), ProgramName VARCHAR(30), primary key (StudentID));

INSERT INTO students(StudentID, FirstName, ProgramName) VALUES (123, 'Carmen', 'HR');

INSERT INTO students(StudentID, FirstName, ProgramName) VALUES (125, 'Emilia', 'Finance');

INSERT INTO students(StudentID, FirstName, ProgramName) VALUES (132, 'Sami', 'IT');

INSERT INTO students(StudentID, FirstName, ProgramName) VALUES (139, 'Carlo', 'HR');

INSERT INTO students(StudentID, FirstName, ProgramName) VALUES (141, 'Jules', 'Finance');

INSERT INTO students(StudentID, FirstName, ProgramName) VALUES (142, 'Elisa', 'IT');

INSERT INTO students(StudentID, FirstName, ProgramName) VALUES (147, 'James', 'IT');

INSERT INTO students(StudentID, FirstName, ProgramName) VALUES (151, 'Carmen', 'Finance');

INSERT INTO students(StudentID, FirstName, ProgramName) VALUES (152, 'Guadalupe', 'IT');

INSERT INTO students(StudentID, FirstName, ProgramName) VALUES (156, 'Ivan', 'HR');

INSERT INTO students(StudentID, FirstName, ProgramName) VALUES (163, 'Cecilia', 'IT');

CREATE TABLE grades(StudentID INTEGER, Course VARCHAR(20), Grade INTEGER, PRIMARY KEY (StudentID, Course));

INSERT INTO grades(StudentID, Course, Grade) VALUES (123, 'Stats', 87);

INSERT INTO grades(StudentID, Course, Grade) VALUES (139, 'Stats', 93);

INSERT INTO grades(StudentID, Course, Grade) VALUES (141, 'Stats', 78);

INSERT INTO grades(StudentID, Course, Grade) VALUES (163, 'Stats', 73);

INSERT INTO grades(StudentID, Course, Grade) VALUES (123, 'Programming', 82);

INSERT INTO grades(StudentID, Course, Grade) VALUES (132, 'Programming', 79);

INSERT INTO grades(StudentID, Course, Grade) VALUES (147, 'Programming', 96);

INSERT INTO grades(StudentID, Course, Grade) VALUES (151, 'Programming', 67);

INSERT INTO grades(StudentID, Course, Grade) VALUES (163, 'Programming', 85);

INSERT INTO grades(StudentID, Course, Grade) VALUES (123, 'Writing', 78);

INSERT INTO grades(StudentID, Course, Grade) VALUES (125, 'Writing', 93);

INSERT INTO grades(StudentID, Course, Grade) VALUES (141, 'Writing', 84);

INSERT INTO grades(StudentID, Course, Grade) VALUES (152, 'Writing', 64);

INSERT INTO grades(StudentID, Course, Grade) VALUES (156, 'Writing', 89);

INSERT INTO grades(StudentID, Course, Grade) VALUES (163, 'Writing', 92);

SELECT \* FROM students JOIN grades USING (StudentID) ORDER BY Grade;

SELECT \* FROM students LEFT JOIN grades USING (StudentID);

SELECT \* FROM students RIGHT JOIN grades USING (StudentID);

SELECT \* FROM students FULL OUTER JOIN grades USING (StudentID);

/\* Note the NULL values for Elisa \*/

SELECT \* FROM students FULL OUTER JOIN grades USING (StudentID) WHERE Grade IS NULL;

/\* just Elisa now \*/

SELECT StudentID, FirstName, COUNT(Grade) AS total FROM students JOIN grades USING (StudentID) GROUP BY StudentID ORDER BY total DESC;

/\* course completions per student \*/

SELECT ProgramName, COUNT(Grade) FROM students JOIN grades USING (StudentID) GROUP BY ProgramName; /\* course completions per program

\*/

SELECT FirstName, AVG(Grade) FROM students JOIN grades USING (StudentID) GROUP BY StudentID;

/\* student GPA \*/

SELECT ProgramName, AVG(Grade) FROM students JOIN grades USING (StudentID) GROUP BY ProgramName;

/\* program-level GPA \*/

SELECT \* FROM grades FULL OUTER JOIN students USING (StudentID) WHERE (course='Programming' OR course='Writing') AND grade > 80 ORDER BY course;

SELECT studentid, case WHEN grade > 80 then 'Pass' end as grade FROM grades;

SELECT \* FROM grades FULL OUTER JOIN students USING (StudentID) WHERE Grade > 60 ORDER BY grade;