Databases 2020

Querying Relational Databases

Sławek Staworko

University of Lille

History of SQL

Relational Database Management Systems (RDBMSs)

- Relational Model proposed by Edgar F. Codd in 1969
- System R (IBM) and Ingres (UC Berkley), two first production RDBMS, 1974
- Nowadays, three major kinds of RDBMS's have emerged:

client-server	Multiple clients connect and interact with a database present on a single server `PostgreSQL`, `MySQL`, `SQLServer`, `DB2`, ...
embedded	Database used in isolation by a single application `SQLite`, `android.database`, ...
distributed	for high-performance, high throughput, big data `MySQL/PostgreSQL Cluster`, `MemSQL`, `F1`

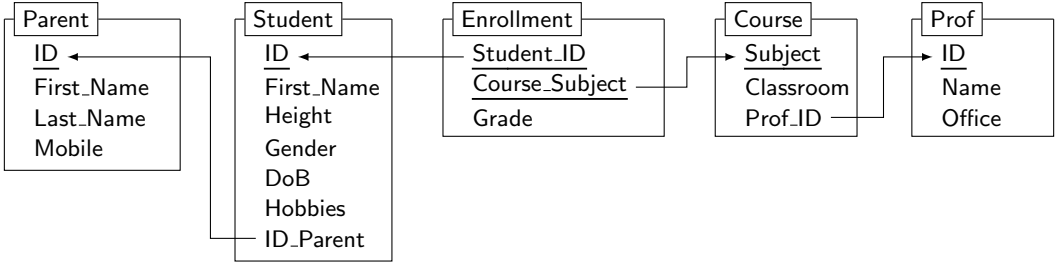
Structured Query Language (SQL)

- Foundation: Relational Algebra (RA) a query formalism proposed as a part of Relational Model.
- Declarative language: specify what you want, RDBMS figures out how to compute it
- Standardization bodies: ANSI 1986 (US) and ISO 1987 (Europe); standard revised every 4-5 years
- SQL is generally not portable between different RDBMS's without minor modifications
- What parts of the standard are supported depends largely on particular goals of a given platform

SQLite vs PostgreSQL vs F1

	SQLite	PostgreSQL	Google F1
Target	small DBs (<1GB)	large DBs (~1TB)	huge DBs (~1PB)
• architecture	single file	server/client	distributed
• applications	embedded	web	big data
example app	*address book*	*internet store*	*Google AdWords*
SQL Features	Basic	Rich	Standard
• data types	only `INT`, `REAL`, `TEXT`	..., `DATE`, `ARRAY`, `JSON`, ...	standard
• constraints	no `CHECK` or `FOREIGN KEY`	full enforcement	standard
• queries	some features missing	fully standard compliant	standard
• indexes	B+-trees	B+-trees, Hash, Geo,...	global and local
Concurrency	Limited	Sophisticated	State of the Art
Locking	global only	table and row	table, row, column
Isolation	none	serializable	optimistic trans.
Parallelism	None	Simple	Full
Replication	–	master-slave	shards (cluster)
Partitioning	–	declarative	automatic
Execution	–	limited distribution	fully distributed
Open Source	`sqlite.org`	`postgresql.org`	Nope ☹

Working Example: Schema




```sql
CREATE TABLE Parent (
  ID INT PRIMARY KEY,
  First_Name TEXT,
  Last_Name TEXT,
  Mobile TEXT
);

CREATE TABLE Prof (
  ID INT PRIMARY KEY,
  Name TEXT,
  Office TEXT
);
```

```sql
CREATE TABLE Course (
  Subject TEXT PRIMARY KEY,
  Classroom TEXT,
  Prof_ID INT REFERENCES Prof(ID)
);

CREATE TABLE Enrollement (
  Student_ID INT
    REFERENCES Student(ID),
  Course_Subject TEXT
    REFERENCES Course(Subject),
  Grade FLOAT,
  PRIMARY KEY (Student_ID, Course_Subject)
);
```

```sql
CREATE TABLE Student (
  ID INT PRIMARY KEY,
  First_Name TEXT,
  Height INT,
  Gender TEXT,
  Hobbies TEXT,
  DoB DATE,
  Parent_ID INT
    REFERENCES Parent(ID)
);
```

Working Example: Database Instance

Parent

ID	First_Name	Last_Name	Mobile
1	Bruno	Dubois	06.14.21.56.34
2	Constance	Dupont	06.41.21.32.14
3	Adèle	Martin	06.84.81.96.12

Course

Subject	Classroom	Prof_ID
SQL	B2.461	11
HTML	A2.061	46
IA	A1.423	11

Prof

ID	Name	Office
11	Sławek	D.42
46	Fabien	C.21
57	Marc	D.42

Student

ID	First_Name	Height	Gender	Hobbies	DoB	Parent_ID
1	Jean	178	M	Reading, Skateboarding	1999-04-13	1
3	Paul	162	M	Sleeping	2000-09-29	3
4	Marie	159	F	Music, Reading, Partying	1998-10-03	1
5	Paul	161	M	NULL	2001-01-07	2
6	Luc	161	M	Reading	2000-10-11	NULL
8	Marion	164	F	Music	1998-04-23	4

Enrollment

Student_ID	Course_Subject	Grade
1	SQL	12.0
1	HTML	12.0
1	IA	NULL
3	SQL	15.0
4	SQL	16.0
4	HTML	17.0
4	IA	12.0
6	SQL	11.0
6	IA	NULL
8	HTML	16.0
8	IA	NULL

SQL Queries

SELECT queries

```
 SELECT   output column list
[FROM     input table expression list]
[WHERE    row filtering conditions]
[GROUP BY grouping expression list
[HAVING   group filtering conditions]]
[ORDER BY ordering specification]
[LIMIT    upper bound on number of rows returned
[OFFSET   number of first rows omitted]]
```

Compound queries

- `JOIN` operations (`INNER`, `OUTER`, `FULL`, `LEFT`, `CROSS`)
- `UNION` [`ALL`] union with duplicates removed (or kept)
- `INTERSECT` [`ALL`] intersection with duplicates removed (or kept)
- `EXCEPT` [`ALL`] set difference (or a bag one)
- `AND`, `OR`, `NOT` Boolean combinations in filter expressions
- `AS`, `EXISTS`, `CREATE VIEW` nested queries

SQL: Selection-Projection Queries

Q1 Display the full table `Student`

```
SELECT * FROM Student;
```

ID	First_Name	Height	Gender	Hobbies	DoB	Parent_ID
1	Jean	178	M	Reading, Skateboarding	1999-04-13	1
3	Paul	162	M	Sleeping	2000-09-29	3
4	Marie	159	F	Music, Reading, Partying	1998-10-03	1
5	Paul	161	M	NULL	2001-01-07	2
6	Luc	161	M	Reading	2000-10-11	NULL
8	Marion	164	F	Music	1998-04-23	4

Notes on SQL Syntax

- SQL syntax (keywords) is not case-sensitive i.e., `SELECT` = `SeLeCt` = `select`
- It is common practice to use all-caps for SQL keywords for better readability
- Table and column names are not case-sensitive either unless surrounded by double quotes " i.e., `Student` = `student` ≠ `"sTuDeNt"`

SQL: Selection-Projection Queries

Q2 Display the first 4 rows in table `Student`

```
SELECT * FROM Student LIMIT 4;
```

ID	First_Name	Height	Gender	Hobbies	DoB	Parent_ID
1	Jean	178	M	Reading, Skateboarding	1999-04-13	1
3	Paul	162	M	Sleeping	2000-09-29	3
4	Marie	159	F	Music, Reading, Partying	1998-10-03	1
5	Paul	161	M	NULL	2001-01-07	2

SQL Standard (and the "organic" way in which it evolves)

- `LIMIT` is not a standard SQL keyword but it has been implemented by most systems since 1990's
- The 2008 revision of SQL has introduced `FETCH`:

```
SELECT * FROM Student FETCH FIRST 4 ROWS;
```

- `FETCH` is supported by PostgreSQL since version 8.3 but it is not supported by SQLite

SQL: Selection-Projection Queries

Q3 Return id, first name, and the height of male students

```
SELECT ID, First_Name, Height
  FROM Student
 WHERE Gender = 'M';
```

ID	First_Name	Height
1	Jean	178
3	Paul	162
5	Paul	161
6	Luc	161

SQL: Selection-Projection Queries

Q4 Find students whose name is Luc or Paul and are born in year 2000

```
SELECT *
  FROM Student
 WHERE (First_Name = 'Luc' OR First_Name = 'Paul')
   AND DoB >= '2000-01-01' AND DoB <= '2000-12-31';
```

ID	First_Name	Height	Gender	Hobbies	DoB	Parent_ID
3	Paul	162	M	Sleeping	2000-09-29	3
6	Luc	161	M	Reading	2000-10-11	NULL

```
SELECT *
  FROM Student
 WHERE First_Name IN ('Luc','Paul')
   AND DoB BETWEEN '2000-01-01' AND '2000-12-31';
```

SQL: Selection-Projection Queries

Q5 Find students whose hobbies include reading

```
SELECT ID, First_Name, DoB
  FROM Student
 WHERE Hobbies LIKE '%Reading%';
```

ID	First_Name	DoB
1	Jean	1999-04-13
4	Marie	1998-10-03
6	Luc	2000-10-11

Pattern matching in SQL

- % matches any sequence of characters (even empty sequence) [PostgreSQL, SQLite]
- _ matches a single character [PostgreSQL]
- in SQLite matching is case insensitive but case-sensitive in PostgreSQL (c.f., `ILIKE`)

SQL: Selection-Projection Queries

Q6 List the height in meters of every male student

```
SELECT ID, First_Name, Height/100.0 AS "Height in Meters"
  FROM Student
 WHERE Gender = 'M';
```

ID	First_Name	Height in Meters
1	Jean	1.78
3	Paul	1.62
5	Paul	1.61
6	Luc	1.61

Spaces in table and column names

- Table and column names may use spaces but then they have to be delimited with double quotes "
- The name inside double quotes is interpreted in a case-sensitive fashion, which can lead to errors
- Spaces in table and column names are best to be avoided

SQL: Sorting

Q7 List all female students in the order of their height, from shortest to highest

```
   SELECT *
     FROM Student
    WHERE Gender = 'F'
ORDER BY Height;
```

ID	First_Name	Height	Gender	Hobbies	DoB	Parent_ID
4	Marie	159	F	Music, Reading, Partying	1998-10-03	1
8	Marion	164	F	Music	1998-04-23	4

SQL: Sorting (cont'd.)

Q8 List all male students and order them by their height, from highest to shortest, and among the students of the same height use the lexicographical order

```
   SELECT *
     FROM Student
    WHERE Gender = 'M'
ORDER BY Height DESC, First_Name;
```

ID	First_Name	Height	Gender	Hobbies	DoB	Parent_ID
1	Jean	178	M	Reading, Skateboarding	1999-04-13	1
3	Paul	162	M	Sleeping	2000-09-29	3
6	Luc	161	M	Reading	2000-10-11	NULL
5	Paul	161	M	NULL	2001-01-07	2

SQL: Duplicate Removal

Q9 List all first names of students without repetitions

```
SELECT DISTINCT First_Name FROM Student;
```

First_Name
Jean
Paul
Marie
Luc
Marion

SQL: Aggregates

Q10 Find the minimum, the average, the maximum, and the sum of heights of all students

```
SELECT MIN(Height), AVG(Height), MAX(Height), SUM(Height) FROM Student;
```

MIN(Height)	AVG(Height)	MAX(Height)	SUM(Height)
159	164.166666666667	178	985

Q11 Find the number of all students born in the 90s

```
SELECT COUNT(*) FROM Student WHERE SUBSTR(DoB,1,3) = '199';
```

COUNT(*)
3

Q12 Find the number of all different (first) names used among students

```
SELECT COUNT(DISTINCT First_Name) AS "Diff Names Count" FROM Student;
```

Diff Names Count
5

SQL: Grouping

Q13 Calculate height average of male students

```
SELECT Gender, AVG(Height) FROM Student WHERE Gender = 'F';
```

Gender	AVG(Height)
F	161.5

Q14 Calculate height average of female students

```
SELECT Gender, AVG(Height) FROM Student WHERE Gender = 'M';
```

Gender	AVG(Height)
M	165.5

Q15 Calculate height average for every gender

```
SELECT Gender, AVG(Height) FROM Student GROUP BY Gender;
```

Gender	AVG(Height)
F	161.5
M	165.5

SQL: Grouping (imperative interpretation)

Q15 Calculate height average for every gender

```
SELECT Gender,     Height
  FROM Student
```

Gender	Height
M	178
M	162
F	159
M	161
M	161
F	164

SQL: Grouping (imperative interpretation)

Q15 Calculate height average for every gender

```
SELECT Gender,     Height
  FROM Student GROUP BY Gender
```

Gender	Height
M	178
M	162
F	159
M	161
M	161
F	164

sort →

Gender	Height
F	159
F	164
M	178
M	162
M	161
M	161

SQL: Grouping (imperative interpretation)

Q15 Calculate height average for every gender

```
SELECT Gender, AVG(Height)
  FROM Student GROUP BY Gender
```

Gender	Height
M	178
M	162
F	159
M	161
M	161
F	164

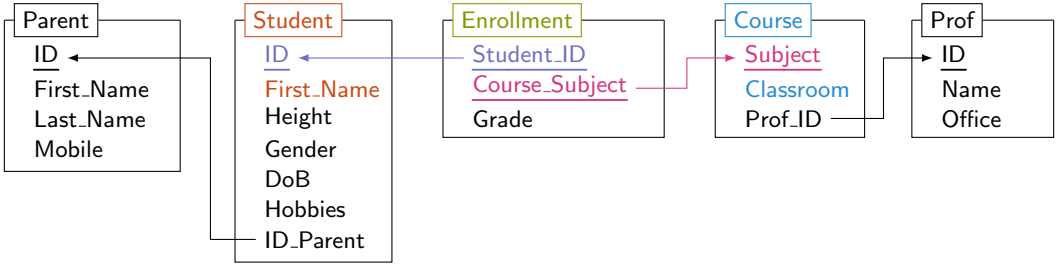
sort →

Gender	Height
F	159
F	164
M	178
M	162
M	161
M	161

aggr. →

Gender	AVG(Height)
F	161.5
M	165.5

SQL: Join Queries



Q16 List the first name of all student that attends a class in the classroom A1.423

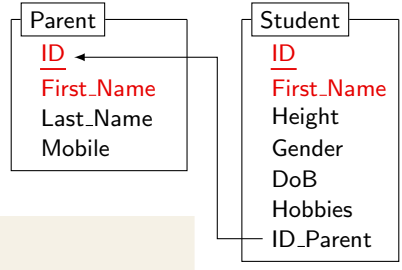
```
SELECT First_Name
  FROM Student
  JOIN Enrollment ON (ID = Student_ID)
  JOIN Course ON (Course_Subject = Subject)
 WHERE Classroom = 'A1.423';
```

First_Name
Jean
Marie
Luc
Marion

SQL: Qualified names

Q17 Display first and last name of every student

```
SELECT First_Name, Last_Name
  FROM Student
  JOIN Parent ON (Parent_ID = ID);
```



Error Ambiguous column names: ID, First_Name

```
SELECT Student.First_Name, Last_Name
  FROM Student
  JOIN Parent ON (Parent_ID = Parent.ID);
```

First_Name	Last_Name
Jean	Dubois
Paul	Martin
Marie	Dubois
Paul	Dupont

SQL: Qualified names and aliases

1. You can use the least amount of qualification in names

```
SELECT Student.First_Name, Last_Name
  FROM Student
  JOIN Parent ON (Parent_ID = Parent.ID);
```

2. But it's a good practice to qualify names of all attributes

```
SELECT Student.First_Name, Parent.Last_Name
  FROM Student
  JOIN Parent ON (Student.Parent_ID = Parent.ID);
```

3. Use aliases if you want to make the query more compact

```
SELECT S.First_Name, P.Last_Name
  FROM Student AS S
  JOIN Parent AS P ON (S.Parent_ID = P.ID);
```

```
SELECT S.First_Name, P.Last_Name
  FROM Student S
  JOIN Parent P ON (S.Parent_ID = P.ID);
```

SQL: Qualified *

Q18 Display all course together with the information on the professor who teaches it

```
SELECT *
  FROM Course
  JOIN Prof ON (Prof_ID = ID);
```

Subject	Classroom	Prof_ID	ID	Name	Office
SQL	B2.461	11	11	Sławek	D.42
HTML	A2.061	46	46	Fabien	C.21
IA	A1.423	11	11	Sławek	D.42

Q19 Display all course together with the name on the professor who teaches it

```
SELECT Course.*, Prof.Name
  FROM Course
  JOIN Prof ON (Prof_ID = ID);
```

Subject	Classroom	Prof_ID	Name
SQL	B2.461	11	Sławek
HTML	A2.061	46	Fabien
IA	A1.423	11	Sławek

SQL: Outer joins

Q20 For every student display their first name and if provided, their last name

```
          SELECT Student.First_Name, Parent.Last_Name
            FROM Student
LEFT OUTER JOIN Parent ON (Student.Parent_ID = Parent.ID);
```

First_Name	Last_Name
Jean	Dubois
Paul	Martin
Marie	Dubois
Paul	Dupont
Luc	NULL
Marion	NULL

SQL: Handling NULL values

Q21 For every student display their first name, their last name, and their full name

```
SELECT Student.First_Name, Parent.Last_Name,
       Student.First_Name || ' ' || Parent.Last_Name AS Full_Name
  FROM Student
LEFT OUTER JOIN Parent ON (Student.Parent_ID = Parent.ID);
```

First_Name	Last_Name	Full_Name
Jean	Dubois	Jean Dubois
Paul	Martin	Paul Martin
Marie	Dubois	Marie Dubois
Paul	Dupont	Paul Dupont
Luc	NULL	NULL
Marion	NULL	NULL

NULL is an absorbing element of practically any function and any operator $x \oplus \texttt{NULL} \mapsto \texttt{NULL}$

'a' || NULL $\mapsto$ NULL 1 * NULL $\mapsto$ NULL NULL/0 $\mapsto$ NULL TRUE AND NULL $\mapsto$ NULL

There are exceptions but also special functions and operators designed to handle NULL values

FALSE AND NULL $\mapsto$ FALSE 'Steve' = NULL $\mapsto$ UNKNOWN NULL IS NULL $\mapsto$ TRUE

IFNULL('a','b') $\mapsto$ 'a' IFNULL(NULL,'b') $\mapsto$ 'b' COALESCE(NULL,'a','b') $\mapsto$ 'a'

SQL: Handling NULL values (cont'd)

Q22 For every student display their first name, their last name, and their full name; if their last name is not provided, their full name consists of their first name alone

```
SELECT Student.First_Name, Parent.Last_Name,
       COALESCE(Student.First_Name || ' ' || Parent.Last_Name,
                Student.First_Name) AS Full_Name
  FROM Student LEFT OUTER JOIN Parent ON (Student.Parent_ID = Parent.ID);
```

```
SELECT Student.First_Name, Parent.Last_Name,
       CASE
            WHEN Parent.Last_Name IS NULL THEN Student.First_Name
            ELSE Student.First_Name || ' ' || Parent.Last_Name
       END AS Full_Name
  FROM Student LEFT OUTER JOIN Parent ON (Student.Parent_ID = Parent.ID);
```

First_Name	Last_Name	Full_Name
Jean	Dubois	Jean Dubois
Paul	Martin	Paul Martin
Marie	Dubois	Marie Dubois
Paul	Dupont	Paul Dupont
Luc	NULL	Luc
Marion	NULL	Marion

SQL: Subqueries, and how to get rid of them

Q23 List professors who teach at least one course

```
SELECT *
  FROM Prof
 WHERE ID IN (SELECT Prof_ID FROM Course);
```

ID	Name	Office
11	Sławek	D.42
46	Fabien	C.21

```
SELECT *
  FROM Prof
 WHERE EXISTS (SELECT * FROM Course WHERE Course.Prof_ID = Prof.ID);
```

```
SELECT DISTINCT Prof.*
  FROM Course JOIN Prof ON (Course.Prof_ID = Prof.ID);
```

SQL: Subqueries, and how to get rid of them (cont'd.)

Q24 List professors who don't teach any course

```
SELECT *
  FROM Prof
 WHERE ID NOT IN (SELECT Prof_ID FROM Course);
```

ID	Name	Office
57	Marc	D.42

```
SELECT *
  FROM Prof
 WHERE NOT EXISTS (SELECT * FROM Course WHERE Course.Prof_ID = Prof.ID);
```

```
SELECT DISTINCT Prof.*
  FROM Prof LEFT OUTER JOIN Course ON (Prof.ID = Course.Prof_ID)
 WHERE Course.Subject IS NULL;
```

SQL: Correlated ANY and ALL Subqueries

Q25 Find the shortest male student(s)

```
SELECT * FROM Student WHERE Gender = 'M' ORDER BY Height ASC LIMIT 1;
```

ID	First_Name	Height	Gender	Hobbies	DoB	Parent_ID
5	Paul	161	M	NULL	2001-01-07	2

```
SELECT * FROM Student
 WHERE Gender = 'M'
   AND Height <= ALL (SELECT Height FROM Student WHERE Gender = 'M');
```

```
SELECT S.*
  FROM Student AS S JOIN Student AS T
 WHERE S.Gender = 'M' AND T.Gender = 'M'
GROUP BY S.ID
HAVING MIN(S.Height <= T.Height) = TRUE;
```

ID	First_Name	Height	Gender	Hobbies	DoB	Parent_ID
5	Paul	161	M	NULL	2001-01-07	2
6	Luc	161	M	Reading	2000-10-11	NULL

SQL: Cross product

```
SELECT * FROM Course, Prof;
```

Subject	Classroom	Prof_ID	ID	Name	Office
SQL	B2.461	11	11	Sławek	D.42
SQL	B2.461	11	46	Fabien	C.21
SQL	B2.461	11	57	Marc	D.42
HTML	A2.061	46	11	Sławek	D.42
HTML	A2.061	46	46	Fabien	C.21
HTML	A2.061	46	57	Marc	D.42
IA	A1.423	11	11	Sławek	D.42
IA	A1.423	11	46	Fabien	C.21
IA	A1.423	11	57	Marc	D.42

```
SELECT * FROM Course, Prof WHERE Course.Prof_ID = Prof.ID;
```

Subject	Classroom	Prof_ID	ID	Name	Office
SQL	B2.461	11	11	Sławek	D.42
HTML	A2.061	46	46	Fabien	C.21
IA	A1.423	11	11	Sławek	D.42

SQL: Cross product (cont'd.)

Q26 List professors who share an office

```
SELECT P1.Name, P2.Name
  FROM Prof AS P1, Prof AS P2
 WHERE P1.Office = P2.Office
   AND P1.ID <> P2.ID;
```

Name	Name
Sławek	Marc
Marc	Sławek

Q27 List professors who share an office without repetition

```
SELECT P1.Name, P2.Name
  FROM Prof AS P1, Prof AS P2
 WHERE P1.Office = P2.Office
   AND P1.ID < P2.ID;
```

Name	Name
Sławek	Marc

SQL: Views

```
CREATE VIEW StudentN AS
SELECT Student.ID, Student.Height, Student.Gender, Student.DoB,
       Student.First_Name || IFNULL(' ' || Parent.Last_Name,'') AS Full_Name
  FROM Student LEFT OUTER JOIN Parent ON (Student.Parent_ID = Parent.ID);
```

Views are virtual tables

- defined by SQL queries
- can be used in queries as standard tables
- are not materialized but can be treated as if their contents changed dynamically

```
SELECT * FROM StudentN;
```

ID	Height	Gender	DoB	Full_Name
1	178	M	1999-04-13	Jean Dubois
3	162	M	2000-09-29	Paul Martin
4	159	F	1998-10-03	Marie Dubois
5	161	M	2001-01-07	Paul Dupont
6	161	M	2000-10-11	Luc
8	164	F	1998-04-23	Marion

SQL: Unions

Q28 Display full prefixed names of all students

```
SELECT 'M. ' || Full_Name AS Prefixed_Name FROM StudentN WHERE Gender = 'M'
UNION
SELECT 'Mlle. ' || Full_Name FROM StudentN WHERE Gender = 'F';
```

Prefixed_Name
M. Jean Dubois
M. Luc
M. Paul Dupont
M. Paul Martin
Mlle. Marie Dubois
Mlle. Marion

Q29 For every student calculate their GPA and the number of classes they attend; Display the list in the order of GPA

```
  SELECT S.Full_Name, AVG(E.Grade) AS GPA, COUNT(*) AS Class_Count
    FROM StudentN AS S JOIN Enrollment E ON (S.ID = E.Student_ID)
GROUP BY S.Full_Name
ORDER BY GPA DESC;
```

Full_Name	GPA	Class_Count
Marion	16.0	2
Marie Dubois	15.0	3
Paul Martin	15.0	1
Jean Dubois	12.0	3
Luc	11.0	2

SQL: Complex queries

Q30 Find students with GPA at least 15

```
  SELECT S.Full_Name
    FROM StudentN AS S JOIN Enrollment E ON (S.ID = E.Student_ID)
GROUP BY S.Full_Name
  HAVING AVG(E.Grade) >= 15;
```

Full_Name
Marie Dubois
Marion
Paul Martin

SQL: Complex queries (cont'd.)

Q31 Find GPA of every student whose all notes are in (the database)

```
  SELECT S.Full_Name, AVG(Grade) AS GPA
    FROM StudentN AS S JOIN Enrollment E ON (S.ID = E.Student_ID)
GROUP BY S.Full_Name
  HAVING COUNT(*) = COUNT(Grade);
```

Full_Name	GPA
Marie Dubois	15.0
Paul Martin	15.0

SQL: Complex queries (cont'd.)

Q32 Calculate for every professor the number of courses they teach

```
  SELECT Prof.Name, COUNT(Course.Subject) AS Courses
    FROM Prof LEFT OUTER JOIN Course ON (Prof.ID = Course.Prof_ID)
GROUP BY Prof.Name;
```

Name	Courses
Fabien	1
Marc	0
Sławek	2

SQL: Complex queries (cont'd.)

Q33 Find professors who have not turned in all their notes

```
SELECT Prof.Name
  FROM Prof
 WHERE EXISTS (
    SELECT * FROM Course
             JOIN Enrollment ON (Course.Subject = Enrollment.Course_Subject)
     WHERE Course.Prof_ID = Prof.ID
       AND Enrollment.Grade IS NULL);
```

Name
Sławek

```
SELECT DISTINCT Prof.Name
  FROM Prof
  JOIN Course ON (Prof.ID = Course.Prof_ID)
  JOIN Enrollment ON (Course.Subject = Enrollment.Course_Subject)
 WHERE Enrollment.Grade IS NULL;
```

SQL: Complex queries (cont'd.)

Q34 Find professors who have not turned in all their notes, together with the number of grades missing

```
SELECT Prof.Name,
       (SELECT COUNT(*)
          FROM Course
          JOIN Enrollment ON (Course.Subject = Enrollment.Course_Subject)
         WHERE Course.Prof_ID = Prof.ID
           AND Enrollment.Grade IS NULL) AS Grades_Missing
  FROM Prof
 WHERE Grades_Missing > 0;
```

Name	Grades_Missing
Sławek	3

```
SELECT Prof.Name, COUNT(*) AS Grades_Missing
  FROM Prof
  JOIN Course ON (Prof.ID = Course.Prof_ID)
  JOIN Enrollment ON (Course.Subject = Enrollment.Course_Subject)
 WHERE Enrollment.Grade IS NULL;
```

SQL: Complex queries (cont'd.)

Q35 For every professor find the number of students they teach; student that attend multiple courses of a professor, should be counted only once for that professor

```
SELECT DISTINCT Course.Prof_ID, Enrollment.Student_ID
  FROM Course
  JOIN Enrollment ON (Course.Subject = Enrollment.Course_Subject);
```

Prof_ID	Student_ID
46	1
11	1
11	3
46	4
11	4
11	6
46	8
11	8

SQL: Complex queries (cont'd.)

Q35 For every professor find the number of students they teach; student that attend multiple courses of a professor, should be counted only once for that professor

```
  SELECT Prof.Name, COUNT(S.Student_ID) AS Student_Count
    FROM Prof LEFT OUTER JOIN (
       SELECT DISTINCT Course.Prof_ID, Enrollment.Student_ID
         FROM Course
         JOIN Enrollment ON (Course.Subject = Enrollment.Course_Subject)
       ) AS S ON (Prof.ID = S.Prof_ID)
GROUP BY Prof.Name;
```

Name	Student_Count
Fabien	3
Marc	0
Sławek	5

```
         SELECT P.Name, COUNT(DISTINCT E.Student_ID) AS Student_Count
           FROM Prof P
LEFT OUTER JOIN Course C ON (P.ID = C.Prof_ID)
LEFT OUTER JOIN Enrollment E ON (C.Subject = E.Course_Subject)
       GROUP BY P.Name;
```

SQL: Data Manipulation Language

Q36 Insert a missing parent

```
INSERT INTO Parent VALUES(4,'Constance','Shariff',NULL);
```

Q37 Delete orphan students

```
DELETE FROM Student WHERE Parent_ID NOT IN (SELECT ID FROM Parent);
```

Q38 Add 2 points to the grade of every student in any of Fabien's classes

```
 UPDATE Enrollment
    SET Grade = Grade + 2
  WHERE Course_Subject IN (
           SELECT Subject
             FROM Course
             JOIN Prof ON (Course.Prof_ID = Prof.ID)
            WHERE Prof.Name = 'Fabien'
        );
```

SQL: Data Definition Language

Q39 Create a table

```
CREATE TABLE Hobbies (ID INT PRIMARY KEY, Name TEXT);
```

Q40 Delete a table

```
DROP TABLE Hobbies;
```

Q41 Remove a view

```
DROP VIEW StudentN;
```

Q42 Extend a table horizontally (change its definition)

```
ALTER TABLE Student ADD COLUMN Phone CHAR(15);
```