DBMS-1

AI20BTECH11006

Assignment-2

Question -1)

Find all the artists who have albums to their name, including those artists that do not. Report for each first alphabet of the artist name, the count of artists per alphabet.
This count should account for: (i) repetitions caused due to the same artist having multiple albums and (ii) artists having no albums.
For e.g., AC/DC is counted twice because of their two albums ...
1 | AC/DC | | For Those About To Rock We Salute You |
1 | AC/DC | | Let There Be Rock |
Your final output must look something like the following:
A | 200
B | 2
...
Z | 56

Answer-1)

```
select substr(Name,1,1) as Alphabet, count(*) as num from artists natural left join albums group by Alphabet;
```

Explanation: substr(Name,1,1) will extract the first letter from the Name column, here all the artists names begin with a block letter, so we need not convert it to uppercase, otherwise, it is better to convert to uppercase. Hence a better query would be

```
select upper(substr(Name,1,1)) as Alphabet, count(*) as num from artists natural left join albums group by Alphabet;
```

Output of query is given below

Alphabet	num
A	32
B	27
C	25
D	19
E	14
F	15
G	15
H	6
I	24
J	19
K	5
L	31
M	33
N	8
O	18
P	15
Q	3
R	16
S	32
T	29
U	11
V	14
W	2
X	1
Y	3
Z	1

Question-2)

2 Full Outer Join (10 marks)

• Using the following DDL statements create the instructor and student tables and populate them with values.

-- create and insert data into the instructor table
CREATE TABLE instructor (
name TEXT,
course_id TEXT
);
INSERT INTO instructor (name, course_id)
VALUES
('Amy','CS1000'),
('Aaron','CS700'),
('Anne','CS400');
-- create and insert data into the student table
CREATE TABLE student (
name TEXT,
course_id TEXT
);
INSERT INTO student (name, course_id)
VALUES
('Jack','CS800'),
('Jones','CS1000'),
('Jason', 'CS450');

• Verify you have the tables setup correctly and that the data is populated correctly.

• Write a query that "emulates" a FULL OUTER JOIN on the student and instructor tables listing the instructor's name, the course they teach, the student's name, and the course they take.

But there is a caveat, you are ONLY allowed to use a LEFT OUTER JOIN clause to achieve the output one would get from a FULL OUTER JOIN!

Answer-2)

Given commands

```
create table instructor ( name TEXT, courseID TEXT);
insert into instructor (name, courseID) values ( 'Amy','CS1000'),(
'Aaron','CS700'),( 'Anne','CS400');
```

Checking if the table is set correctly

```
PRAGMA table_info('instructor');
```

output

cid	name	type	notnull	dflt_value	pk
0	name	TEXT	0	NULL	0
1	courseID	TEXT	0	NULL	0

Checking if the values are entered correctly

```
select * from instructor;
```

Output

name	courseID
Amy	CS1000
Aaron	CS700
Anne	CS400

Now for student table

Creating table, inserting values

```
create table student ( name TEXT, courseID TEXT);
insert into student (name, courseID) values ( 'Jack','CS800'),(
'Jones','CS1000'), ( 'Jason','CS450');
```

Checking if table is set correctly

```
PRAGMA table_info('student');
```

Output

cid	name	type	notnull	dflt_value	pk
0	name	TEXT	0	NULL	0
1	courseID	TEXT	0	NULL	0

Checking if values are correctly populated

```
select * from student;
```

Output

name	courseID
Jack	CS800
Jones	CS1000
Jason	CS450

The query that imitates full outer join is

```
select instructor.name as Instructor_name, Instructor.courseID as
instructor_courseID, student.name as Student_name, student.courseID as
student_courseID from instructor left outer join student on
student.courseID = instructor.courseID
```

```
union
select instructor.name as Instructor_name, Instructor.courseID as
instructor_courseID, student.name as Student_name, student.courseID as
student_courseID from student left outer join instructor on
student.courseID = instructor.courseID;
```

Here, we take left outer join two times once instructor with student and then student with instructor and then take a union of the two, this would account for the inner join, the null values that would come from the left join of instructor with student, and right join of instructor with student which is necessarily the same as left join of student with instructor. The output of the query is given below.

Instructor_name	instructor_courseID	Student_name	student_courseID
NULL	NULL	Jack	CS800
NULL	NULL	Jason	CS450
Aaron	CS700	NULL	NULL
Amy	CS1000	Jones	CS1000
Anne	CS400	NULL	NULL

Question-3

3 Track counts (10 marks)
Classify the tracks based on its play length.
a. “short”: less than a minute,
b. “medium”: between 1 and 5 minutes,
c. “long”: greater than 5 minutes.
Report the count of tracks per play length classification.

Answer-3)

Here, we will convert milliseconds to minutes, and then make cases and classify based on that, here’s the query

```
select classification, count(*) as count from (select *, case when minutes
<1 then 'short' when minutes between 1 and 5 then 'medium' else 'long' end
as classification from (select *, Milliseconds/60000.0 as minutes from
tracks)) t group by t.classification;
```

The output of they query is given below

classification	count
long	1069
medium	2407
short	27

Question-4

4 Generate some combinations! (10 marks)
• Using the following DDL statements, create table X and populate it with values.
create table X (
id_num integer,
id_str text
);
insert into X (id_num, id_str)
values
(1,’A’),
(2,’B’),
(3,’C’),
(4,’D’),
(5,’E’);

• Verify you have the tables setup correctly and that the data is populated correctly.
• For all entries in table X, generate all possible combinations of size 3 for both the numeric IDs (id num) and corresponding string IDs (id str).
Note that a valid row should look like:
1,2,3 | A,B,C
but
3,2,1 | C,B,A
is NOT valid. Order does matter!

Answer-4)

Creating the table

```
create table X( id_num integer, id_str TEXT);
```

Inserting the values

```
insert into X (id_num,id_str) values
(1,'A'),(2,'B'),(3,'C'),(4,'D'),(5,'E');
```

To check if the table is correct

```
PRAGMA table_info('X');
```

Output of the query

cid	name	type	notnull	dflt_value	pk
0	id_num	integer	0	NULL	0
1	id_str	TEXT	0	NULL	0

To check is the data is correctly entered

```
select * from X;
```

id_num	id_str
1	A
2	B
3	C
4	D
5	E

Now the query to generate combinations

```
Select distinct T1.id_num || ',' || T2.id_num || ',' || T3.id_num as
id_num_combinations, T1.id_str || ',' || T2.id_str || ',' || T3.id_str as
id_str_combinations from X as T1, X as T2, X as T3 where T1.id_num <
T2.id_num and T2.id_num < T3.id_num;
```

Explanation:

We will form a cartesian product and then concatenate the relevant columns, after removing the rows in which the id_num is smaller for succeeding columns compared to the preceding ones. Here, distinct doesn't really change anything, but if the same sql file is run over and over, it would still output the same thing for this question.

Output of the query

id_num_combinations	id_str_combinations
1,2,3	A,B,C
1,2,4	A,B,D
1,2,5	A,B,E
1,3,4	A,C,D
1,3,5	A,C,E
1,4,5	A,D,E
2,3,4	B,C,D
2,3,5	B,C,E
2,4,5	B,D,E
3,4,5	C,D,E

---------- END OF ASSIGNMENT ----------

Resources:

VS code SQLite extension used to display the output of queries : alexcvzz.vscode-sqlite