Database Management Systems

MySQL - Advanced Features

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
and
Centre for Artificial Intelligence and Machine Learning
Indian Statistical Institute, Kolkata

June, 2021

Handling empty tuples

SQL can test whether a subquery has at all any tuple in its result or not. The `exists` construct returns the value true if the argument subquery is nonempty.

```
 select ...
from ...
where exists (
select ...
from ...
where ...);
```

Note: We can test for the nonexistence of tuples in a subquery by using the `not exists` construct.

Handling empty tuples – Practical applications

Consider the following table.

Table: BANK1

ID	CUSTOMER	ACCOUNT	YEAR
12340001	Sai Pallav	Savings	2018
12340002	Aarav Roy	Savings	2018
12340003	Vihaan Sen	Current	2018
12340004	Reyanshi Razdan	Current	2018
12340005	Balbindar Singh	Savings	2019
12340006	Aarohi Tendulkar	Current	2019
12340007	Karthikeyan Murugan	Current	2019
12340008	David Barik	Current	2019
12340009	Jagannath Mishra	Current	2019
12340010	Jahanara Begum	Savings	2020

Handling empty tuples – Practical applications

Consider another table as follows.

Table: BANK2

ID	CUSTOMER	ACCOUNT	YEAR
11110001	John Trevor	Current	2018
11110002	Anurak Eshin	Savings	2019
11110003	Somlata Sen	Current	2019
11110004	Ezhil Swami	Current	2019
11110005	Panna Singh	Savings	2019
11110006	Aarohi Tendulkar	Savings	2020
11110007	Aravind Babu	Current	2020
11110008	Jahanara Begum	Savings	2020
11110009	Aarohi Tendulkar	Current	2021

Handling empty tuples – Practical applications

Suppose the customer details of two banks are provided in the form of two separate tables (as shown in `BANK1` and `BANK2`). The names of all the customers, who have accounts in both the banks, can be retrieved with the following SQL query.

```
select BANK1.CUSTOMER from BANK1, BANK2 where
BANK1.CUSTOMER = BANK2.CUSTOMER;
```

This will yield the following result.

CUSTOMER
Aarohi Tendulkar
Jahanara Begum

Handling empty tuples – Practical applications

Suppose the customer details of two banks are provided in the form of two separate tables (as shown in `BANK1` and `BANK2`). The names of all the customers, who have accounts in both the banks, can be retrieved with the following SQL query.

```
select BANK1.CUSTOMER from BANK1, BANK2 where
BANK1.CUSTOMER = BANK2.CUSTOMER;
```

This will yield the following result.

CUSTOMER
Aarohi Tendulkar
Jahanara Begum

Note: The space requirement of this approach is huge (as Cartesian product is to be performed).

Handling empty tuples – Practical applications

This can be efficiently done with the following alternative query.

```
 select CUSTOMER
from BANK1
where exists (
select CUSTOMER
from BANK2
where BANK1.CUSTOMER = BANK2.CUSTOMER);
```

Note: Existence of tuples in the relation returned by the subquery is verified tuplewise from the main query (not as a whole).

Handling duplicate tuples

SQL can test whether a subquery has any duplicate tuples in its result or not. The `unique` construct returns the value true if the argument subquery contains no duplicate tuples.

```
 select ...
from ...
where unique (
select ...
from ...
where ...);
```

Note: We can test for the existence of duplicate tuples in a subquery by using the `not unique` construct.

Handling duplicate tuples – Practical applications

Suppose the customer details of two banks are provided in the form of two separate tables (as shown in BANK1 and BANK2). The names of all the customers, who have any arbitrary number of accounts in BANK1 but exactly one account in BANK2, can be retrieved with the following SQL query.

```
 select CUSTOMER
from BANK1
where unique (
select CUSTOMER
from BANK2
where BANK1.CUSTOMER = BANK2.CUSTOMER);
```

Note: Duplicity of tuples in the relation returned by the subquery is verified tuplewise from the main query (not as a whole).

Creating views

We can define a view in SQL by using the `create view` construct. View names may appear in any place that a relation name may appear. To define a view, we must give the view a name and must state the query that computes the view.

The form of the create view command is as follows:

```
create view <view_name> as <query_expression>;
```

where `query_expression` is any arbitrary query expression which is legal.

Creating views

```
create view BANK2_VIEW as
select * from BANK2 where YEAR = 2020;
select * from BANK2_VIEW;
```

This will yield the following result.

ID	CUSTOMER	ACCOUNT	YEAR
11110006	Aarohi Tendulkar	Savings	2020
11110007	Aravind Babu	Current	2020
11110008	Jahanara Begum	Savings	2020

Creating views

```
create view BANK2_VIEW as
select * from BANK2 where YEAR = 2020;
select * from BANK2_VIEW;
```

This will yield the following result.

ID	CUSTOMER	ACCOUNT	YEAR
11110006	Aarohi Tendulkar	Savings	2020
11110007	Aravind Babu	Current	2020
11110008	Jahanara Begum	Savings	2020

Note: A view is a logical representation of another table (termed as base table). Any change in the view is also reflected (if permissible) on the base table. However, this change is temporary.

Updating tables using view

```
insert into BANK2_VIEW values (0, null, null, 2021);
```

"select * from BANK2_VIEW;" will yield

ID	CUSTOMER	ACCOUNT	YEAR
11110006	Aarohi Tendulkar	Savings	2020
11110007	Aravind Babu	Current	2020
11110008	Jahanara Begum	Savings	2020

Updating tables using view

```
insert into BANK2_VIEW values (0, null, null, 2021);
```

"`select * from BANK2_VIEW;`" will yield

ID	CUSTOMER	ACCOUNT	YEAR
11110006	Aarohi Tendulkar	Savings	2020
11110007	Aravind Babu	Current	2020
11110008	Jahanara Begum	Savings	2020

"`select * from BANK2;`" will yield

ID	CUSTOMER	ACCOUNT	YEAR
11110001	John Trevor	Current	2018
11110002	Anurak Eshin	Savings	2019
11110003	Somlata Sen	Current	2019
11110004	Ezhil Swami	Current	2019
11110005	Panna Singh	Savings	2019
11110006	Aarohi Tendulkar	Savings	2020
11110007	Aravind Babu	Current	2020
11110008	Jahanara Begum	Savings	2020
0	NULL	NULL	2021

Updating views

SQL allows a view name to appear wherever a relation may appear. Hence, we can use the following constructs.

```
select * from <view_name> where ...;
```

or

```
insert into <view_name> values (...);
```

etc.

Dropping views

We can drop a view in SQL by using the `drop view` construct as follows.

```
drop view <view_name>;
```

Note: The view definition stays in the database until it is dropped.

Problems

1. Consider the following schema of an online code repository system like GitHub:
   - CONTRIBUTOR = $\langle \underline{contributor\_id} : integer, contributor\_name : string \rangle$
   - CODE-GROUP = $\langle contributor\_id : integer, code\_group : string, count\_submissions : integer \rangle$

   Write the following queries in SQL.
   i) Find all the contributors who have made no submissions within the Java code group.
   ii) Find all the contributors who have made at most one submission within the R code group.
   iii) Find all the contributors who have made at least three submissions within the Scala code group.

Problems

3 Consider the following schema of a bank:

- BRANCH = $\langle \underline{branch\_id} : integer, branch\_name : string, branch\_city : string, assets : real \rangle$
- CUSTOMER = $\langle \underline{customer\_id} : integer, customer\_name : string, customer\_street : string, customer\_city : string \rangle$
- ACCOUNT = $\langle \underline{account\_number} : integer, customer\_id : integer, branch\_name : string, balance : real \rangle$
- DEPOSITOR = $\langle \underline{customer\_id} : integer, customer\_name : string, account\_number : integer \rangle$
- BORROWER = $\langle customer\_name : string, loan\_number : integer \rangle$

Write the following queries in SQL.

i) Find all customers who have both an account and a loan at the bank.

ii) Find all customers who have an account at all the branches located in Kolkata.

Problems

4. Consider the following schema.
   - EMPLOYEE = $\langle \underline{empid\_id} : integer, fullname : string, managerid : integer, dataofjoining : date \rangle$
   - EMPSALARY = $\langle \underline{empid\_id} : integer, project : string, salary : integer \rangle$

   Write the following queries in SQL.
   i) Fetch employee names having salary no lesser than 5000 and no higher than 10000.
   ii) Fetch only the first name (string before space) from the *fullname* column of *EMPLOYEE* table.
   iii) Fetch all employees from the *EMPLOYEE* table who are also managers.
   iv) Fetch only odd rows from the table *EMPSALARY*.
   v) Find the third highest salary from the table *EMPSALARY*.