Fundamentals of Database Systems

[SQL – III]

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
Indian Statistical Institute, Kolkata

August, 2019

Handling empty tuples

SQL can test whether a subquery has any tuples in its result. 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

Suppose the customer details of two banks are provided in the form of two separate tables (say `bank1` and `bank2`). The names of all the customers who have an account in both the banks can be retrieved with the following SQL query.

```
 select customer_name
from bank1
where exists (
select *
from bank2
where bank1.customer_name = bank2.customer_name);
```

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. 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.

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.

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.

Problems

1. 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

2. 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.