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

February, 2020

1 Limiting the Tuples

2 Dealing with Nullity and Duplicity

3 View

4 Problems

Consider a table

Table: IPL

YEAR	VENUE	WINNER	PoS
2008	India	Rajasthan Royals	Shane Watson
2009	South Africa	Deccan Chargers	Adam Gilchrist
2010	India	Chennai Super Kings	Sachin Tendulkar
2011	India	Chennai Super Kings	Chris Gayle
2012	India	Kolkata Knight Riders	Sunil Narine
2013	India	Mumbai Indians	Shane Watson
2014	India, UAE	Kolkata Knight Riders	Glenn Maxwell
2015	India	Mumbai Indians	Andre Russell
2016	India	Sunrisers Hyderabad	Virat Kohli
2017	India	Mumbai Indians	Ben Stokes
2018	India	Chennai Super Kings	Sunil Narine
2019	India	Mumbai Indians	Andre Russell

Returning limited number of tuples (in MySQL)

"`select * from IPL where YEAR between 2013 and 2017 limit 2;`" will yield

YEAR	VENUE	WINNER	PoS
2013	India	Mumbai Indians	Shane Watson
2014	India, UAE	Kolkata Knight Riders	Glenn Maxwell

Returning limited number of tuples (in SQL Server)

"`select top 2 * from IPL where YEAR between 2013 and 2017;`" will yield

YEAR	VENUE	WINNER	PoS
2013	India	Mumbai Indians	Shane Watson
2014	India, UAE	Kolkata Knight Riders	Glenn Maxwell

Returning limited number of tuples (in Oracle)

"`select * from IPL where YEAR between 2013 and 2017 and rownum <= 2;`" will yield

YEAR	VENUE	WINNER	PoS
2013	India	Mumbai Indians	Shane Watson
2014	India, UAE	Kolkata Knight Riders	Glenn Maxwell

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. Let there be a relation R $= \langle A, B, C \rangle$, in which the attribute B is numeric. Write an SQL query that can find out the maximum value of the attribute B without using the max() function.
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.

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.