Database Management Systems

MySQL - Data Manipulation

Malay Bhattacharyya

Associate Professor

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

February, 2025

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
2020	UAE	Mumbai Indians	Jofra Archer

An example

Given the IPL table, the SQL query "`select * from IPL where PoS = 'Shane Watson';`" will yield the following.

YEAR	VENUE	WINNER	PoS
2008	India	Rajasthan Royals	Shane Watson
2013	India	Mumbai Indians	Shane Watson

`select` clause in SQL versus projection operation in relational algebra

"`select VENUE from IPL;`" will yield

VENUE
India
South Africa
India
India
India
India
India, UAE
India
India
India
India
India
UAE

`select` clause in SQL versus projection operation in relational algebra

"`select distinct VENUE from IPL;`" will yield

VENUE
India
South Africa
India, UAE
UAE

Relational operations

The following relational operators are available in SQL.

Operator	Description
=	Equal
<> or !=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
`LIKE ...`	Search for a pattern
`BETWEEN ... AND ...`	Between an inclusive range
`IN (..., ..., ...)`	Verify multiple values for an attribute

Note: These operators are used in the `where` clause.

Relational operations – {=, <>/ !=, >, <, >=, <=}

These are the standard ones!!!

Relational operations – LIKE

Like helps to perform the pattern matching operation on strings. The '%' and '_' are used to match any substring (of length zero or more) and any single character, respectively.

Relational operations – LIKE

Like helps to perform the pattern matching operation on strings. The '%' and '_' are used to match any substring (of length zero or more) and any single character, respectively.

"`select * from IPL where PoS like '%ell';`" will yield

YEAR	VENUE	WINNER	PoS
2014	India, UAE	Kolkata Knight Riders	Glenn Maxwell
2015	India	Mumbai Indians	Andre Russell
2019	India	Mumbai Indians	Andre Russell

Relational operations – LIKE

Like helps to perform the pattern matching operation on strings. The '%' and '_' are used to match any substring (of length zero or more) and any single character, respectively.

"`select * from IPL where PoS like '%ell';`" will yield

YEAR	VENUE	WINNER	PoS
2014	India, UAE	Kolkata Knight Riders	Glenn Maxwell
2015	India	Mumbai Indians	Andre Russell
2019	India	Mumbai Indians	Andre Russell

"`select * from IPL where PoS like 'S%a___';`" will yield

YEAR	VENUE	WINNER	PoS
2008	India	Rajasthan Royals	Shane Watson
2012	India	Kolkata Knight Riders	Sunil Narine
2013	India	Mumbai Indians	Shane Watson
2018	India	Chennai Super Kings	Sunil Narine

Relational operations – LIKE

"`select * from IPL where VENUE like '%, %';`" will yield

YEAR	VENUE	WINNER	PoS
2014	India, UAE	Kolkata Knight Riders	Glenn Maxwell

Relational operations – LIKE

"`select * from IPL where VENUE like '%, %';`" will yield

YEAR	VENUE	WINNER	PoS
2014	India, UAE	Kolkata Knight Riders	Glenn Maxwell

"`select * from IPL where WINNER like '% % %';`" will yield

YEAR	VENUE	WINNER	PoS
2010	India	Chennai Super Kings	Sachin Tendulkar
2011	India	Chennai Super Kings	Chris Gayle
2012	India	Kolkata Knight Riders	Sunil Narine
2014	India, UAE	Kolkata Knight Riders	Glenn Maxwell
2018	India	Chennai Super Kings	Sunil Narine

Relational operations – BETWEEN-AND

"`select * 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
2015	India	Mumbai Indians	Andre Russell
2016	India	Sunrisers Hyderabad	Virat Kohli
2017	India	Mumbai Indians	Ben Stokes

Relational operations – IN

"`select * from IPL where VENUE in ('India', 'UAE');`"
will yield

YEAR	VENUE	WINNER	PoS
2008	India	Rajasthan Royals	Shane Watson
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
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
2020	UAE	Mumbai Indians	Jofra Archer

Logical operations – NOT

"`select * from IPL where YEAR not between 2013 and 2017;`" will yield

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
2018	India	Chennai Super Kings	Sunil Narine
2019	India	Mumbai Indians	Andre Russell
2020	UAE	Mumbai Indians	Jofra Archer

Logical operations – OR

"`select * from IPL where YEAR < 2013 or YEAR > 2017;`" will yield

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
2018	India	Chennai Super Kings	Sunil Narine
2019	India	Mumbai Indians	Andre Russell
2020	UAE	Mumbai Indians	Jofra Archer

Logical operations – AND

"`select * from IPL where WINNER = 'Chennai Super Kings' and PoS = 'Sachin Tendulkar';`" will yield

YEAR	VENUE	WINNER	PoS
2010	India	Chennai Super Kings	Sachin Tendulkar

Set operations – Difference

"`(select VENUE from IPL) except (select VENUE from IPL where VENUE = 'South Africa');`" will yield

VENUE
India
India, UAE
UAE

Note: The `except` operation automatically eliminates duplicates from the first query. Therefore, adding `distinct` (before `VENUE`) in the first query yields the same as the above result.

Set operations – Difference

"`(select VENUE from IPL) except all (select VENUE from IPL where VENUE = 'South Africa');`" will yield

VENUE
India
India
India
India
India
India, UAE
India
India
India
India
India
UAE

Note: To retain all duplicates from the first query, `except all` is to be used.

Set operations – Difference (in MySQL)

As there is no except operation in MySQL, we can write the following equivalent query:

```
select VENUE from IPL where VENUE not in (select
VENUE from IPL where VENUE = 'South Africa');
```

VENUE
India
India
India
India
India
India, UAE
India
India
India
India
India
UAE

Set operations – Union

"(select YEAR, WINNER
from IPL
where VENUE = 'India, UAE')
union
(select YEAR, WINNER
from IPL
where VENUE = 'South Africa'); " will yield

YEAR	WINNER
2014	Kolkata Knight Riders
2009	Deccan Chargers
2019	Mumbai Indians

Note: The `union` operation automatically eliminates duplicates. To retain all duplicates, `union all` is to be used.

Set operations – Intersection

```
“(select *
from IPL
where VENUE = ‘India’)
intersect
(select *
from IPL
where WINNER = ‘Chennai Super Kings’); ” will yield
```

YEAR	VENUE	WINNER	PoS
2010	India	Chennai Super Kings	Sachin Tendulkar
2011	India	Chennai Super Kings	Chris Gayle
2018	India	Chennai Super Kings	Sunil Narine

Note: The `intersect` operation automatically eliminates duplicates from the first query. To retain all duplicates, `intersect all` is to be used.

Set operations – Intersection (in MySQL)

As there is no `intersect` operation in MySQL, we can write the following equivalent query:

```
select * from IPL where VENUE = 'India' and WINNER =
'Chennai Super Kings';
```

YEAR	VENUE	WINNER	PoS
2010	India	Chennai Super Kings	Sachin Tendulkar
2011	India	Chennai Super Kings	Chris Gayle
2018	India	Chennai Super Kings	Sunil Narine

Sorting tuples in ascending order – asc

"`select * from IPL where YEAR <= 2013 order by WINNER asc;`" will yield

YEAR	VENUE	WINNER	PoS
2010	India	Chennai Super Kings	Sachin Tendulkar
2011	India	Chennai Super Kings	Chris Gayle
2009	South Africa	Deccan Chargers	Adam Gilchrist
2012	India	Kolkata Knight Riders	Sunil Narine
2013	India	Mumbai Indians	Shane Watson
2008	India	Rajasthan Royals	Shane Watson

Note: The ascending order is the default option. Hence, including the keyword `asc` is optional.

Sorting tuples in descending order – `desc`

"`select * from IPL where YEAR <= 2013 order by WINNER desc;`" will yield

YEAR	VENUE	WINNER	PoS
2008	India	Rajasthan Royals	Shane Watson
2013	India	Mumbai Indians	Shane Watson
2012	India	Kolkata Knight Riders	Sunil Narine
2009	South Africa	Deccan Chargers	Adam Gilchrist
2011	India	Chennai Super Kings	Chris Gayle
2010	India	Chennai Super Kings	Sachin Tendulkar

Limiting the number of tuples returned (in Oracle)

"`select * from IPL where YEAR <= 2013 order by YEAR desc rownum <= 2;`" will yield

YEAR	VENUE	WINNER	PoS
2013	India	Mumbai Indians	Shane Watson
2012	India	Kolkata Knight Riders	Sunil Narine

Limiting the number of tuples returned (in Oracle)

"`select * from IPL where YEAR <= 2013 order by YEAR desc rownum <= 2;`" will yield

YEAR	VENUE	WINNER	PoS
2013	India	Mumbai Indians	Shane Watson
2012	India	Kolkata Knight Riders	Sunil Narine

Note: "`rownum <= n`" will return n tuples starting from the top.

Limiting the number of tuples returned (in MySQL)

"`select * from IPL where YEAR <= 2013 order by YEAR desc limit 2;`" will yield

YEAR	VENUE	WINNER	PoS
2013	India	Mumbai Indians	Shane Watson
2012	India	Kolkata Knight Riders	Sunil Narine

Limiting the number of tuples returned (in MySQL)

"`select * from IPL where YEAR <= 2013 order by YEAR desc limit 2;`" will yield

YEAR	VENUE	WINNER	PoS
2013	India	Mumbai Indians	Shane Watson
2012	India	Kolkata Knight Riders	Sunil Narine

Note: "`limit n`" will return n tuples starting from the top.

Limiting the number of tuples returned (in MySQL)

"`select * from IPL where YEAR <= 2013 order by YEAR desc limit 2,3;`" will yield

YEAR	VENUE	WINNER	PoS
2011	India	Chennai Super Kings	Chris Gayle
2010	India	Chennai Super Kings	Sachin Tendulkar
2009	South Africa	Deccan Chargers	Adam Gilchrist

Limiting the number of tuples returned (in MySQL)

"`select * from IPL where YEAR <= 2013 order by YEAR desc limit 2,3;`" will yield

YEAR	VENUE	WINNER	PoS
2011	India	Chennai Super Kings	Chris Gayle
2010	India	Chennai Super Kings	Sachin Tendulkar
2009	South Africa	Deccan Chargers	Adam Gilchrist

Note: "`limit m, n`" will return n tuples after skipping m tuples from the top.

Grouping tuples

To group the tuples based on same values on the attribute VENUE, we write the following.

```
select VENUE from IPL group by VENUE;
```

VENUE
India
South Africa
India, UAE
UAE

Grouping tuples – More features

We can group the tuples and count based on same values of an attribute as follows.

```
select VENUE, count(WINNER) from IPL group by VENUE;
```

VENUE	count(WINNER)
India	10
South Africa	1
India, UAE	1
UAE	1

Grouping tuples – with `having` clause

We can group the tuples and set conditions on attribute as follows.

```
 select VENUE, count(WINNER) from IPL group by VENUE
having count(WINNER) > 1;
```

VENUE	count(WINNER)
India	10

Join operations

Consider another relation as follows.

Table: WC

YEAR	VENUE	WINNER	PoS
2003	South Africa, Zimbabwe, Kenya	Australia	Sachin Tendulkar
2007	West Indies	Australia	Glenn McGrath
2011	India, Sri Lanka, Bangladesh	India	Yuvraj Singh
2015	Australia, New Zealand	Australia	Mitchell Starc
2019	England, Wales	England	Kane Williamson

Join operations

Inner join: `select * from IPL inner join WC;`

Natural inner join: `select * from IPL natural inner join WC;`

Left outer join: `select * from IPL left outer join WC;`
Natural left outer join: `select * from IPL natural left outer join WC;`

Right outer join: `select * from IPL right outer inner join WC;`

Natural right outer join: `select * from IPL natural right outer inner join WC;`

Full outer join: `select * from IPL full outer join WC;`

Cartesian product

The Cartesian product of the two relations IPL and WC can be obtained as follows.

```
select * from IPL, WC;
```

Aggregate functions

The following functions take a collection of values (generally through attribute names) as input and return a single value.

Function	Description
`count()`	Number of items
`sum()`	Summation
`avg()`	Average value
`max()`	Maximum value
`min()`	Minimum value

Note: The aggregate functions work at the `select` line and takes attributes (not relations) as the arguments.

Aggregate functions – sum() and avg()

We can compute average of the values selected over a particular attribute as follows.

```
select avg(YEAR) from IPL where YEAR < 2011;
```

avg(YEAR)
2009

Note: sum() and avg() work only on numeric data.

Aggregate functions – count(), max() and min()

We can compute the minimum of the values selected over a particular attribute as follows.

```
select WINNER, PoS from IPL where YEAR = (select
max(YEAR) from IPL);
```

WINNER	PoS
Mumbai Indians	Jofra Archer

Note: count(), max() and min() can work on both numeric and nonnumeric data.

Nested structure

```
 select VENUE, PoS
from IPL
where WINNER not in
(select WINNER
from IPL
where YEAR >= 2010);
```

VENUE	PoS
India	Shane Watson
South Africa	Adam Gilchrist

Problems

1. Consider the following schema representing a train reservation database:
   - PASSENGER = $\langle \underline{pid} : integer, pname : string, age : integer \rangle$
   - RESERVATION = $\langle pid : integer, class : string, tid : integer, tamount : number \rangle$

   Note that, a single transaction (through *tid*) can include multiple reservations of passengers travelling in a group. Write the following queries in SQL.
   i) Find the *pname*s (names of passengers) that comprise firstname and surname both.
   ii) Find the *pid*s of passengers who are not adults and have a reservation in the 'Sleeper' class.
   iii) Calculate the total amount paid by all the senior citizens (age more than 60) together through the system.

Solution

(i) `select pname from PASSENGER;`

(ii) `select PASSENGER.pid from PASSENGER as p, RESERVATION as r where p.pid=r.pid and p.age<18 and r.class like "Sleeper";`

(ii) `create view T as (selct PASSENGER.pid, RESERVATION.tamount from PASSENGER as p, RESERVATION as r where p.pid=r.pid and p.age>60); select sum(T.tamount) from T;`

Problems

2. Consider the following schema representing the population of some cities in United States along with the names states to which they belong to:
   - CENSUS = $\langle \underline{id} : integer, city : string, state : string, population : number \rangle$

   Write queries in SQL that will return the names of least and most populous cities included in *CENSUS*. If there are more then return all.
3. Write an SQL query that performs a division operation on a pair of relations without using the division operator (i.e., $\div$).
   Hint: Use the Cartesain product and other operations.
4. Write an SQL query that performs a natural join without using any one of the available joining operations.
   Hint: Use the Cartesain product and other operations.

Problems

5. Consider the following schema representing the costs charged by the instructors for the courses on a MOOC platform:
   - COURSES = $\langle \underline{cid} : integer, cname : string, ctype : string \rangle$
   - INSTRUCTORS = $\langle \underline{iid} : integer, iname : string, affiliation : string \rangle$
   - CATALOG = $\langle \underline{cid} : integer, \underline{iid} : integer, cost : real \rangle$

   The *Catalog* relation lists the costs charged for courses by the Instructors. Write the following queries in SQL.
   i) Find the *cid*s of free courses offered from ISI, Kolkata.
   ii) Find the *cid*s of the most expensive course(s) offered by the instructor named H. F. Korth.
   iii) Find the *iid*s of instructors who offer either part-time or full-time course (there can be other course types too).
   iv) Find the *iid*s of instructors who offer only part-time courses.
   v) Find the *cid*s of courses offered by multiple instructors.
   vi) Find pairs of *iids* such that the instructor with the first *iid* charges more for some course than the instructor with the second *iid*.

Problems

6. Consider the following schema representing latitude and longitude of some cities along with their states in a weather observation station:
   - STATION $= \langle \underline{id} : integer, city : string, state : string, latitude : number, longitude : number \rangle$

   Write queries in SQL that will return the pair of cities in *Station* with the shortest and longest *city* names, as well as their respective lengths (i.e., number of characters in the name). If there is more than one smallest or largest *city* names, choose the one that is alphabetically ahead.

Solution

(i) `select top 1 city, len(city) from STATION order by len(city) ASC, city ASC;`

(ii) `select top 1 city, len(city) from STATION order by len(city) DESC, city ASC;`

Problems

7. Consider the following schema representing the record low and record high temperatures observed for various cities in India over a period of 100 years:
   - TEMPERATURE $= \langle \underline{cityid} : integer, cityname : string, recordlow : real, recordhigh : real, year : integer \rangle$

   The key corresponding to this relation is underlined. Write the following queries in SQL.
   i) Find the names of all cities with the highest high temperature.
   ii) Find the names of all cities with the highest high temperature and/or lowest low temperature.
   iii) Find the names of all cities that has never observed a freezing temperature (i.e., zero degree Celsius).

Problems

8. The following schema covers details of the online customers who place orders in different outlets of Domino's Pizza.
   - CUSTOMER = $\langle \underline{name} : string, age : integer, gender : string, \underline{road - address} : string \rangle$
   - ORDERS = $\langle \underline{name} : string, \underline{pizza} : string \rangle$
   - DELIVERS = $\langle \underline{outlet} : string, \underline{pizza} : string, cost : real \rangle$

   The keys corresponding to each relation are underlined. Write the following queries in SQL.
   i) Find the names of all teenager customers whose name consist of at least 5 characters and start with "A". Assume that a teenager is less than 18 years of age.
   ii) Find the names of all males who live at "203 B. T. Road" and orders both "Margherita" and "Chicken Tikka" pizza.
   iii) Find the names of all customers who have never placed an order. Note that, this might be a null value.
   iv) Find the outlets delivering the cheapest "Capsicum" pizza.
   v) Find all the outlets that deliver at least one pizza that can be ordered by "Malay" in less than Rs. 300.

Problems

9. Consider the following schema representing annual salaries of M.Tech (CS) students from the batch of 2019-21 offered by different companies. Let there be multiple job offers to the same student (denoted by fullname) in the said batch.
   - SALARY $= \langle \underline{jobid} : integer, fullname : string, company : string, salary : real \rangle$

   The primary key is underlined in the above schema and it gets automatically incremented. Write the following queries in SQL.
   (i) Delete the newest tuples from the SALARY table for the records that have duplicate entries in fullname.
   (ii) Delete the tuples from the SALARY table with lower salaries for the records that have multiple job offers (may be due to different positions) to the same person by the same company.

Solution

(i)
```
create table TEMP (select * from SALARY group by
fullname);
drop table SALARY;
create table SALARY select * from TEMP;
drop table TEMP;
```

(ii)
```
create table TEMP (select * from salary order by
salary desc);
drop table SALARY;
create table SALARY (select * from TEMP group by
fullname, company order by salary desc);
drop table TEMP;
```