Database Management Systems

SQL – Basic Features

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
Indian Statistical Institute, Kolkata

January, 2019

SQL

SQL or structured query language is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS). SQL uses a combination of relational algebra and relational calculus constructs.

SQL is not only for querying, rather it also helps in defining the structure of the data, modifying the data and specifying the security constraints.

SQL

SQL or structured query language is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS). SQL uses a combination of relational algebra and relational calculus constructs.

SQL is not only for querying, rather it also helps in defining the structure of the data, modifying the data and specifying the security constraints.

Note: The SQL keywords are case-insensitive, however, they are often written in uppercase. In some setups, table and column names are case-sensitive.

SQL functionalities

- **Data-definition language (DDL)** – provides commands for defining relation schemas, deleting relations, and modifying relation schemas.
- **Data-manipulation language (DML)** – includes commands to work on attributes, insert tuples into, delete tuples from, and modify tuples in the database.
- **View definition** – includes commands for defining views.
- **Transaction control** – includes commands for specifying the beginning and ending of transactions.
- **Embedded SQL and dynamic SQL** – embeds SQL statements into general-purpose programming languages.
- **Integrity** – includes commands for specifying integrity constraints that the data stored in the database must satisfy.
- **Authorization** – includes commands for specifying access rights to relations and views.

History

"An SQL query goes into a bar, walks up to two tables and asks, 'Can I join you?'." – Anonymous.

1970s: Original version called Sequel, developed as a part of the System R project, was first implemented by IBM.

History

"An SQL query goes into a bar, walks up to two tables and asks, 'Can I join you?'." – Anonymous.

1970s: Original version called Sequel, developed as a part of the System R project, was first implemented by IBM.
1986: American national Standards Institute (ANSI) and International Organization for Standardization (ISO) published an SQL standard SQL-86.

History

"An SQL query goes into a bar, walks up to two tables and asks, 'Can I join you?'." – Anonymous.

1970s: Original version called Sequel, developed as a part of the System R project, was first implemented by IBM.
1986: American national Standards Institute (ANSI) and International Organization for Standardization (ISO) published an SQL standard SQL-86.
1987: IBM published its own corporate SQL standard Systems Application Architecture Database Interface (SAA-SQL).

History

"An SQL query goes into a bar, walks up to two tables and asks, 'Can I join you?'." – Anonymous.

1970s: Original version called Sequel, developed as a part of the System R project, was first implemented by IBM.
1986: American national Standards Institute (ANSI) and International Organization for Standardization (ISO) published an SQL standard SQL-86.
1987: IBM published its own corporate SQL standard Systems Application Architecture Database Interface (SAA-SQL).
1989: ANSI published an extended version SQL-89.

History

"An SQL query goes into a bar, walks up to two tables and asks, 'Can I join you?'." – Anonymous.

1970s: Original version called Sequel, developed as a part of the System R project, was first implemented by IBM.
1986: American national Standards Institute (ANSI) and International Organization for Standardization (ISO) published an SQL standard SQL-86.
1987: IBM published its own corporate SQL standard Systems Application Architecture Database Interface (SAA-SQL).
1989: ANSI published an extended version SQL-89.
1992: A major extended version SQL-92 was published.

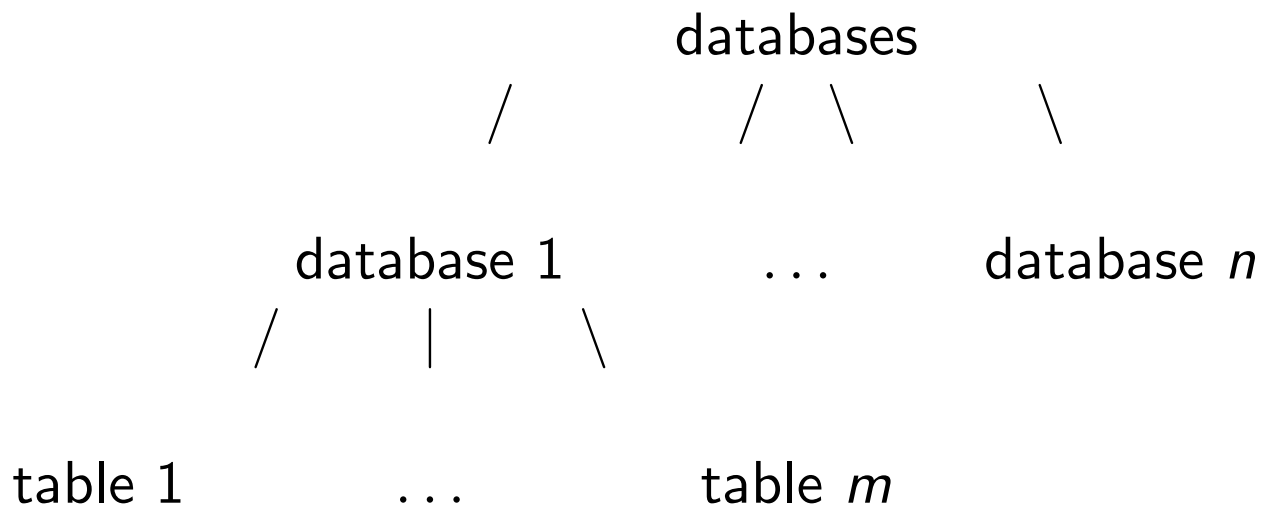
History

"An SQL query goes into a bar, walks up to two tables and asks, 'Can I join you?'." – Anonymous.

1970s: Original version called Sequel, developed as a part of the System R project, was first implemented by IBM.
1986: American national Standards Institute (ANSI) and International Organization for Standardization (ISO) published an SQL standard SQL-86.
1987: IBM published its own corporate SQL standard Systems Application Architecture Database Interface (SAA-SQL).
1989: ANSI published an extended version SQL-89.
1992: A major extended version SQL-92 was published.
1999-2016: The versions SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011 and SQL:2016 were published.

Data view through SQL

In practice, the databases (as a whole) comprises several separate database and each database consists of several tables.



Connecting with MySQL

```
 $ mysql -u <user_name> -p
Enter password:
mysql> _
mysql> show databases;
mysql> connect <database_name>;
mysql> show tables;
mysql> desc <table_name>;
mysql> exit
$ _
```

We will be using the accounts with username/password `root/mysql` (DBA) and `student/student123` (general user) on MySQL during the course.

Note: In Oracle SQL, `scott` and `tiger` are the general username and password, respectively. The default password for the DBA is `system`.

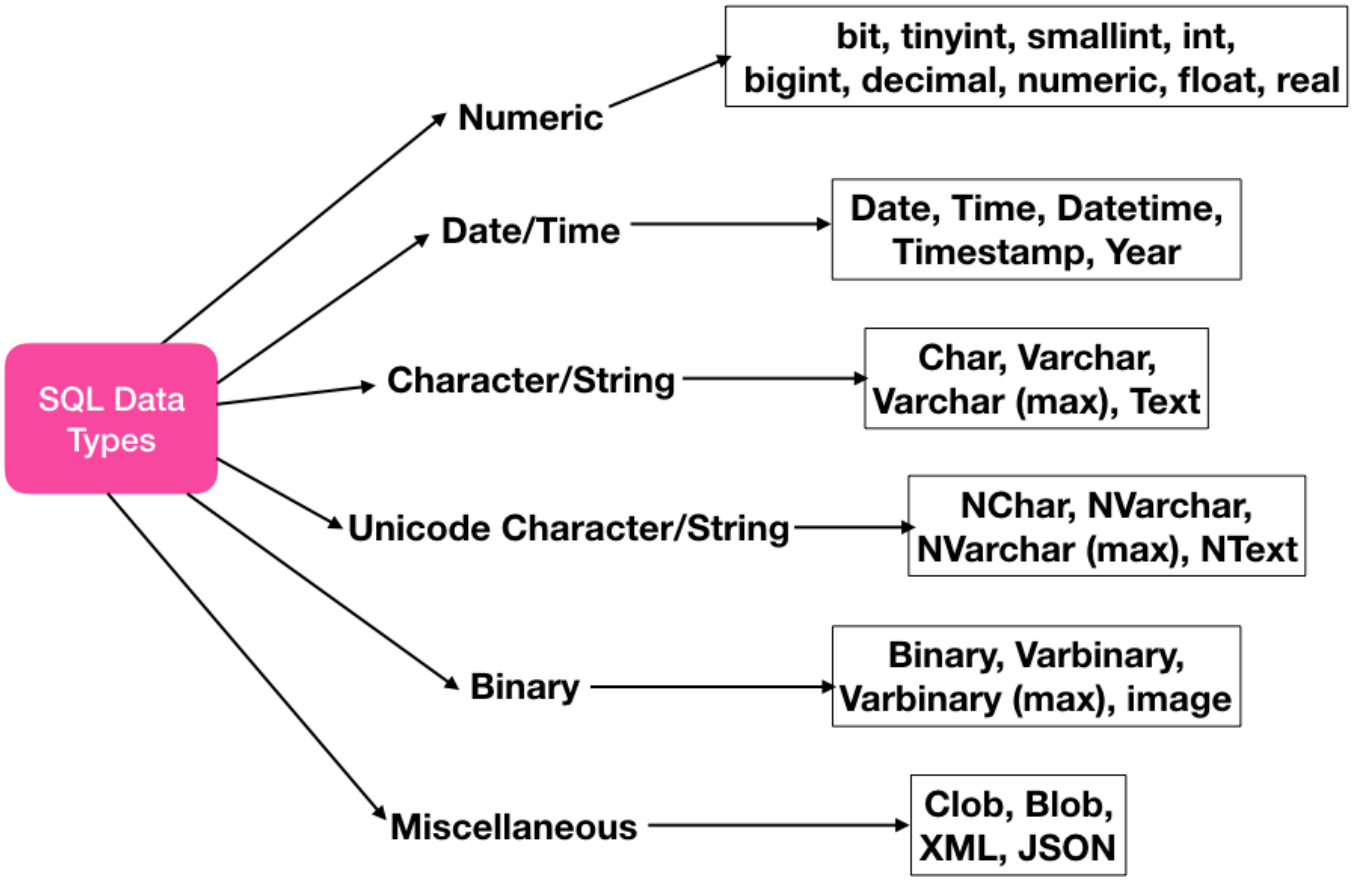
Principle structure

A typical SQL query for data definition appears as follows:

```
create table R(
A1 D1, A2 D2, ..., Ak Dk ,
(IC1), ..., (ICl)
);
```

Here, each A_i represents an attribute in the schema of relation R, each D_i denotes the data type of values in the domain of the corresponding attribute A_i, and IC_i symbolizes an integrity-constraint.

The data types in SQL



The data types in MySQL – Numerics

Type	Size (bits)	Minimum	Maximum
tinyint(dig)	8	-2^7	$2^7 - 1$
unsigned tinyint(dig)	8	0	$2^8 - 1$
smallint(dig)	16	-2^{15}	$2^{15} - 1$
unsigned smallint(dig)	16	0	$2^{16} - 1$
mediumint(dig)	24	-2^{23}	$2^{23} - 1$
unsigned mediumint(dig)	24	0	$2^{24} - 1$
int(dig)	32	-2^{31}	$2^{31} - 1$
unsigned int(dig)	32	0	$2^{32} - 1$
bigint(dig)	64	-2^{63}	$2^{63} - 1$
unsigned bigint(dig)	64	0	$2^{64} - 1$
real(dig, dec)	32	$-3.40E + 38$	$3.40E + 38$
float(dig, dec)	64	$-1.79E + 308$	$1.79E + 308$
decimal(dig, dec)	136	$-10^{38} + 1$	$10^{38} - 1$
numeric(dig, dec)	136	$-10^{38} + 1$	$10^{38} - 1$

Note: The maximum number of digits before and after the decimal point can be specified with *dig* and *dec*, respectively.

The data types in MySQL – Text

Type	Storage	Maximum
char(size)	Fixed length string	255 characters
varchar(size)	Variable length string	255 characters
tinytext(size)	Character large object/CLOB	255 characters
text(size)	Character large object/CLOB	65,535 characters
mediumtext(size)	Character large object/CLOB	16,777,215 characters
longtext(size)	Character large object/CLOB	4,294,967,295 characters
blob(size)	Binary large object/BLOB	65,535 bytes
mediumblob(size)	Binary large object/BLOB	16,777,215 bytes
longblob(size)	Binary large object/BLOB	4,294,967,295 bytes
enum(x,y,z,etc.)	A list of possible values	65535 values

Note: A string can contain letters, numbers, and special characters. The maximum number of characters in a string can be specified with *size*.

Consider a table

Table: IPL

YEAR	VENUE	WINNER	PoS
8	India	Rajasthan Royals	Shane Watson
9	South Africa	Deccan Chargers	Adam Gilchrist
10	India	Chennai Super Kings	Sachin Tendulkar
11	India	Chennai Super Kings	Chris Gayle
12	India	Kolkata Knight Riders	Sunil Narine
13	India	Mumbai Indians	Shane Watson
14	India, UAE	Kolkata Knight Riders	Glenn Maxwell
15	India	Mumbai Indians	Andre Russell
16	India	Sunrisers Hyderabad	Virat Kohli
17	India	Mumbai Indians	Ben Stokes
18	India	Chennai Super Kings	Sunil Narine
19	South Africa	null	null

Creating a table

The IPL table, on which we will be working on further, can be created with the following SQL query.

```
 create table IPL
(YEAR tinyint(4) not null,
VENUE char(50),
WINNER char(30),
PoS char(30),
primary key (YEAR));
```

Note: The attribute `YEAR`, which cannot be `null`, is defined as the primary key of IPL table.

Deleting a table

The IPL table can be deleted from database using the following SQL query.

```
drop table IPL;
```

Altering a table

The IPL table can be altered by adding a new attribute A_p and mentioning its domain D_p (data type) as follows. All the existing tuples will be assigned a `null` value for the new attribute A_p.

`alter table IPL add` $A_p D_p$;

Altering a table

The IPL table can be altered by adding a new attribute A_p and mentioning its domain D_p (data type) as follows. All the existing tuples will be assigned a `null` value for the new attribute A_p.

`alter table IPL add` $A_p D_p$;

The IPL table can be altered by dropping an existing attribute A_p as follows.

`alter table IPL drop` A_q;

Altering a table

The IPL table can be altered by adding a new attribute A_p and mentioning its domain D_p (data type) as follows. All the existing tuples will be assigned a `null` value for the new attribute A_p.

`alter table IPL add` $A_p D_p$;

The IPL table can be altered by dropping an existing attribute A_p as follows.

`alter table IPL drop` A_q;

Note: New primary keys can also be added/dropped in a similar way.

Renaming a table and its attributes

The IPL table and its attributes can be renamed and reused, as and when required, within an SQL query as follows:

```
 select IPL1.PoS from IPL as IPL1, IPL as IPL2 where
IPL1.PoS = IPL2.PoS and IPL1.YEAR > IPL2.YEAR and
IPL2.WINNER = ‘Rajasthan Royals’;
```

Renaming a table and its attributes

The IPL table and its attributes can be renamed and reused, as and when required, within an SQL query as follows:

```
 select IPL1.PoS from IPL as IPL1, IPL as IPL2 where
IPL1.PoS = IPL2.PoS and IPL1.YEAR > IPL2.YEAR and
IPL2.WINNER = 'Rajasthan Royals';
```

This will yield the names of Player of Series (PoS) winners while playing for Rajasthan Royals earlier and then winning again at a later time.

Delete operation on the tuples

```
 delete from IPL
where YEAR < 10;
```

Note: It works on the entire tuple and can not delete values on arbitrary attributes.

Delete operation on the tuples

Table: IPL

YEAR	VENUE	WINNER	PoS
10	India	Chennai Super Kings	Sachin Tendulkar
11	India	Chennai Super Kings	Chris Gayle
12	India	Kolkata Knight Riders	Sunil Narine
13	India	Mumbai Indians	Shane Watson
14	India, UAE	Kolkata Knight Riders	Glenn Maxwell
15	India	Mumbai Indians	Andre Russell
16	India	Sunrisers Hyderabad	Virat Kohli
17	India	Mumbai Indians	Ben Stokes
18	India	Chennai Super Kings	Sunil Narine
19	South Africa	null	null

Insert operation

```
 insert into IPL
values (8, 'India', 'Rajasthan Royals','Shane
Waton');

insert into IPL values (9, 'South Africa', 'Deccan
Chargers','Adam Gilchrist');
```

Note: You can optionally mention the attributes as well as follows "`insert into IPL (YEAR, VENUE, WINNER, PoS) values (9, 'South Africa', 'Deccan Chargers','Adam Gilchrist');`".

Insert operation

Table: IPL

YEAR	VENUE	WINNER	PoS
8	India	Rajasthan Royals	Shane Watson
9	South Africa	Deccan Chargers	Adam Gilchrist
10	India	Chennai Super Kings	Sachin Tendulkar
11	India	Chennai Super Kings	Chris Gayle
12	India	Kolkata Knight Riders	Sunil Narine
13	India	Mumbai Indians	Shane Watson
14	India, UAE	Kolkata Knight Riders	Glenn Maxwell
15	India	Mumbai Indians	Andre Russell
16	India	Sunrisers Hyderabad	Virat Kohli
17	India	Mumbai Indians	Ben Stokes
18	India	Chennai Super Kings	Sunil Narine
19	South Africa	null	null

Note: The inserted tuples are added as per the order of primary key value, otherwise (no primary key) at the bottom.

Update operation

```
 update IPL
set YEAR = YEAR + 2000
where YEAR < 2000;
```

Update operation

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	South Africa	null	null

Principle structure

A typical SQL query for data manipulation appears as follows:

```
 select A_1, A_2, ..., A_m
from R_1, R_2, ..., R_n
where P;
```

Here, each A_i represents an attribute, each R_i denotes a relation and P is a predicate.

Principle structure

A typical SQL query for data manipulation appears as follows:

```
 select A_1, A_2, ..., A_m
from R_1, R_2, ..., R_n
where P;
```

Here, each A_i represents an attribute, each R_i denotes a relation and P is a predicate.

- The `select` clause corresponds to the projection operation of the relational algebra.
- The `from` clause corresponds to the Cartesian-product operation of the relational algebra.
- The `where` clause corresponds to the selection predicate of the relational algebra.

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

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 – Like

Like helps to perform the pattern matching operation on strings. The '%' and '_' are used to match any substring and any character, respectively.

Relational operations – Like

Like helps to perform the pattern matching operation on strings. The '%' and '_' are used to match any substring and any 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

Relational operations – Like

Like helps to perform the pattern matching operation on strings. The '%' and '_' are used to match any substring and any 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

"`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 – 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

Logical operations – Not

"`select * from IPL where not YEAR 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	South Africa	null	null

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	South Africa	null	null

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 distinct VENUE from IPL) except
(select VENUE from IPL where VENUE = 'South Africa');
```

VENUE
India
India, UAE

Set operations – Difference

```
(select distinct VENUE from IPL) except
(select VENUE from IPL where VENUE = 'South Africa');
```

VENUE
India
India, UAE

```
(select VENUE from IPL) except all
(select VENUE from IPL where VENUE = 'India');
```

VENUE
South Africa
India, UAE
South Africa

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

Set operations – Union

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

YEAR	WINNER
2014	Kolkata Knight Riders
2009	Deccan Chargers
2019	null

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');
```

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. To retain all duplicates, `intersect all` is to be used.

Ordering tuples

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

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

Grouping by

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

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

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	null	null

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: `sum()` and `avg()` work only on numeric data, however, `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 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 *iid*s of instructors who offer either part-time or full-time course (there can be other course types too).

   (ii) Find pairs of *iids* such that the instructor with the first *iid* charges more for some course than the instructor with the second *iid*.

   (iii) Find the *cid*s of courses offered by at least two different instructors.

   (iv) Find the *cid*s of the most expensive course(s) offered by the instructor named H. F. Korth.

Problems

2. 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.
3. 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

4. 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 *pid*s of passengers who are more than 65 years old and have a reservation in the 'AC' class.
   (ii) Find the *pname*s (names of passengers) that are involved in the reservation through a common transaction.
   (iii) Calculate the total amount paid by all the senior citizens (age more than 60) together through the system.