Team4-DummyHead-Database Design

1. Main Tables

Our team chooses Table User, Student, Teacher, Reservation, Message, Court, Sport, Hobby as the main tables.

```
mysql> SHOW TABLES;
+-----------------------+
| Tables_in_sportgather |
+-----------------------+
| Appointment           |
| Course                |
| Court                 |
| Enrollment            |
| Hobby                 |
| Message               |
| Rate                  |
| Reservation           |
| Sport                 |
| Student               |
| Teacher               |
| User                  |
+-----------------------+
12 rows in set (0.00 sec)
```

2.DDL Commands for Tables

CREATE TABLE User (

UserId VARCHAR(50) PRIMARY KEY,

FirstName VARCHAR(20) NOT NULL,

LastName VARCHAR(20),

Gender VARCHAR(20),

Age REAL,

Email VARCHAR(50) NOT NULL,

Password VARCHAR(100) NOT NULL,

Phone VARCHAR(20),

Location VARCHAR(200),

Type ENUM('S', 'T')

);

CREATE TABLE Student (

StudentId VARCHAR(50) REFERENCES User ON DELETE CASCADE,

Major VARCHAR(20),

PRIMARY KEY (StudentId)

);

CREATE TABLE Teacher (

TeacherId VARCHAR(50) REFERENCES User ON DELETE CASCADE,

Rating REAL,

YearofTeaching REAL NOT NULL,

```
PRIMARY KEY (TeacherId)
);

CREATE TABLE Sport(
SportId VARCHAR(50) PRIMARY KEY,
SportName VARCHAR(50) NOT NULL,
SportIntroduction VARCHAR(1000) NOT NULL,
SportImage VARCHAR(500)
);

CREATE TABLE Course (
CourseId VARCHAR(50) PRIMARY KEY,
Name VARCHAR(20) NOT NULL,
Location VARCHAR(200),
StartDate DATE NOT NULL,
EndDate DATE NOT NULL,
CourseType ENUM('Online', 'In-person'),
SportId VARCHAR(50) REFERENCES Sport ON DELETE CASCADE,
TeacherId VARCHAR(50) REFERENCES Teacher ON DELETE CASCADE
);

CREATE TABLE Message(
MessageId VARCHAR(50),
LaunchTime DATETIME,
Title VARCHAR(50) NOT NULL,
Content VARCHAR(2000) NOT NULL,
StudentId VARCHAR(50)  REFERENCES Student ON DELETE CASCADE,
PRIMARY KEY (MessageId, StudentId)
);

CREATE TABLE Court(
CourtId VARCHAR(50) PRIMARY KEY,
Name VARCHAR(50),
Location VARCHAR(200) NOT NULL
SportId VARCHAR(50) REFERENCES Sport ON DELETE CASCADE
);

CREATE TABLE Reservation(
ReservationId VARCHAR(50) PRIMARY KEY,
CourtId VARCHAR(50) REFERENCES Court ON DELETE CASCADE,
```

```
StudentId VARCHAR(50) REFERENCES Student ON DELETE CASCADE,
BeginTime DATETIME NOT NULL,
EndTime DATETIME NOT NULL
);

CREATE TABLE Appointment (
AppointmentId VARCHAR(50) PRIMARY KEY,
StudentId VARCHAR(50) REFERENCES Student ON DELETE CASCADE,
TeacherId VARCHAR(50) REFERENCES Teacher ON DELETE CASCADE,
Link VARCHAR(200),
ReservationId VARCHAR(50) REFERENCES Reservation ON DELETE CASCADE,
AppointmentType ENUM('Online', 'In-person'),
Comment VARCHAR(2000)
);

CREATE TABLE Enrollment (
StudentId VARCHAR(50) REFERENCES Student ON DELETE CASCADE,
CourseId VARCHAR(50) REFERENCES Course ON DELETE CASCADE,
PRIMARY KEY (StudentId, CourseId)
);

CREATE TABLE Rate (
StudentId VARCHAR(50) REFERENCES Student ON DELETE CASCADE,
CourseId VARCHAR(50) REFERENCES Course ON DELETE CASCADE,
Rating REAL,
Review VARCHAR(2000),
PRIMARY KEY (StudentId, CourseId)
);

CREATE TABLE Hobby (
StudentId VARCHAR(50) REFERENCES Student ON DELETE CASCADE,
SportId VARCHAR(50) REFERENCES Sport ON DELETE CASCADE,
Year REAL,
PRIMARY KEY (StudentId, SportId)
);
```

3.Screenshots of Data Counts in Main Tables

```
mysql> SELECT COUNT(*) FROM User;
+----------+
| COUNT(*) |
+----------+
|     3000 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Student;
+----------+
| COUNT(*) |
+----------+
|     1500 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Teacher;
+----------+
| COUNT(*) |
+----------+
|     1500 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Hobby;
+----------+
| COUNT(*) |
+----------+
|     1007 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Reservation;
+----------+
| COUNT(*) |
+----------+
|     2000 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Message;
+----------+
| COUNT(*) |
+----------+
|     2000 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Sport;
+----------+
| COUNT(*) |
+----------+
|       46 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Court;
+----------+
| COUNT(*) |
+----------+
|       38 |
+----------+
1 row in set (0.00 sec)
```

4.Advanced SQL Queries

(1) Find top 15 users who share most number of sport hobbies with the user whose id is 1039.

```
SELECT h.StudentId, u.FirstName, u.LastName, COUNT(*) AS MatchNum
FROM Hobby h JOIN User  u on h.StudentId = u.UserId
WHERE h.StudentId <> '1039'
AND h.SportId IN (SELECT SportId
            FROM Hobby
            WHERE StudentId = '1039')
GROUP BY h.StudentId,  u.FirstName, u.LastName
ORDER BY COUNT(h.SportId) DESC
LIMIT 15
```

(2) For each sport, find top 2 users who reserve the court related to this sport most frequently.

```
With newTable AS
(SELECT UserId, SportId,
row_number() OVER (PARTITION BY SportId ORDER BY cnt DESC) AS rk
FROM
(SELECT r.UserId, c.SportId, COUNT(r.ReservationId) AS cnt
FROM Reservation AS r NATURAL JOIN Court AS c
GROUP BY r.UserId, c.SportId) AS a)
SELECT s.SportName, u.FirstName, u.LastName
FROM newTable AS n NATURAL JOIN User AS u NATURAL JOIN Sport AS s
WHERE n.rk <= 2
ORDER BY s.SportName ASC
LIMIT 15;
```

5.Screenshots of Top 15 Rows

(1) Top 15 users who share most number of sport hobbies with user whose id is 1039.

StudentId	FirstName	LastName	MatchNum
454	Dorothy	Betancourt	4
801	Olivia	Campbell	2
531	Diane	Campbell	2
433	Danielle	Snider	2
773	Gene	Nichols	2
312	Julie	Davis	2
1428	George	Kujawa	2
1300	Terry	Franklin	2
1027	Jan	Jackson	1
1058	George	Gable	1
1060	Anthony	Bethune	1
997	Charles	Dean	1
1089	Linda	Gay	1
1093	Thuy	Long	1
1113	Sharon	Johnson	1

(2) Top 2 users who reserve the court related to each sport most frequently.

	SportName	FirstName	LastName
▶	American Football	Hector	Mares
	American Football	Cathy	King
	Badminton	Kimberly	Crabtree
	Badminton	Bertha	Black
	Basketball	Antonio	Jensen
	Basketball	Jerome	McCoy
	Boxing	Susan	Helmer
	Boxing	Michael	Dobson
	Cycling	Hillary	Johnson
	Cycling	Daniel	Price
	Figure Skating	Larry	Ashley
	Figure Skating	Joe	Ward
	Football	Melvin	Crosby
	Football	Ernesto	Bolden
	Squash	Larry	Williams

6.Trying at Least 3 Indices on Each Advanced Query

Query 1 - No index

```
| -> Limit: 15 row(s)  (actual time=9.379..9.382 rows=15 loops=1)
    -> Sort: `count(h.SportId)` DESC, limit input to 15 row(s) per chunk  (actual time=9.378..9.380 rows=15 loops=1)
        -> Stream results  (cost=1005.83 rows=1005) (actual time=0.256..9.227 rows=233 loops=1)
            -> Group aggregate: count(h.SportId), count(0)  (cost=1005.83 rows=1005) (actual time=0.248..8.996 rows=233 loops=1)
                -> Nested loop inner join  (cost=905.33 rows=1005) (actual time=0.110..8.687 rows=243 loops=1)
                    -> Nested loop inner join  (cost=553.58 rows=1005) (actual time=0.098..6.291 rows=243 loops=1)
                        -> Filter: (h.StudentId <> '1039')  (cost=201.83 rows=1005) (actual time=0.060..1.125 rows=1005 loops=1)
                            -> Index range scan on h using PRIMARY  (cost=201.83 rows=1005) (actual time=0.057..0.800 rows=1005 loops=1)
                        -> Single-row index lookup on Hobby using PRIMARY (StudentId='1039', SportId=h.SportId)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=0 loops=1005)
                    -> Single-row index lookup on u using PRIMARY (UserId=h.StudentId)  (cost=0.25 rows=1) (actual time=0.009..0.009 rows=1 loops=243)
 |
```

Query 1 - Index On User.FirstName

```
| -> Limit: 15 row(s)  (actual time=7.669..7.672 rows=15 loops=1)
    -> Sort: `count(h.SportId)` DESC, limit input to 15 row(s) per chunk  (actual time=7.668..7.670 rows=15 loops=1)
        -> Stream results  (cost=1005.83 rows=1005) (actual time=0.253..7.545 rows=233 loops=1)
            -> Group aggregate: count(h.SportId), count(0)  (cost=1005.83 rows=1005) (actual time=0.247..7.356 rows=233 loops=1)
                -> Nested loop inner join  (cost=905.33 rows=1005) (actual time=0.113..7.098 rows=243 loops=1)
                    -> Nested loop inner join  (cost=553.58 rows=1005) (actual time=0.101..5.319 rows=243 loops=1)
                        -> Filter: (h.StudentId <> '1039')  (cost=201.83 rows=1005) (actual time=0.061..0.983 rows=1005 loops=1)
                            -> Index range scan on h using PRIMARY  (cost=201.83 rows=1005) (actual time=0.058..0.706 rows=1005 loops=1)
                        -> Single-row index lookup on Hobby using PRIMARY (StudentId='1039', SportId=h.SportId)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=0 loops=1005)
                    -> Single-row index lookup on u using PRIMARY (UserId=h.StudentId)  (cost=0.25 rows=1) (actual time=0.007..0.007 rows=1 loops=243)
 |
```

Query 1 - Index On User.LastName

```
| -> Limit: 15 row(s)  (actual time=7.773..7.776 rows=15 loops=1)
    -> Sort: `count(h.SportId)` DESC, limit input to 15 row(s) per chunk  (actual time=7.772..7.774 rows=15 loops=1)
        -> Stream results  (cost=1005.83 rows=1005) (actual time=0.329..7.646 rows=233 loops=1)
            -> Group aggregate: count(h.SportId), count(0)  (cost=1005.83 rows=1005) (actual time=0.323..7.457 rows=233 loops=1)
                -> Nested loop inner join  (cost=905.33 rows=1005) (actual time=0.186..7.200 rows=243 loops=1)
                    -> Nested loop inner join  (cost=553.58 rows=1005) (actual time=0.173..5.454 rows=243 loops=1)
                        -> Filter: (h.StudentId <> '1039')  (cost=201.83 rows=1005) (actual time=0.057..0.994 rows=1005 loops=1)
                            -> Index range scan on h using PRIMARY  (cost=201.83 rows=1005) (actual time=0.055..0.717 rows=1005 loops=1)
                        -> Single-row index lookup on Hobby using PRIMARY (StudentId='1039', SportId=h.SportId)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=0 loops=1005)
                    -> Single-row index lookup on u using PRIMARY (UserId=h.StudentId)  (cost=0.25 rows=1) (actual time=0.007..0.007 rows=1 loops=243)
 |
```

Query 1 - Index On User.FirstName & User.LastName

```
| -> Limit: 15 row(s)  (actual time=5.170..5.173 rows=15 loops=1)
    -> Sort: `count(h.SportId)` DESC, limit input to 15 row(s) per chunk  (actual time=5.169..5.171 rows=15 loops=1)
        -> Stream results  (cost=1005.83 rows=1005) (actual time=0.168..5.081 rows=233 loops=1)
            -> Group aggregate: count(h.SportId), count(0)  (cost=1005.83 rows=1005) (actual time=0.162..4.949 rows=233 loops=1)
                -> Nested loop inner join  (cost=905.33 rows=1005) (actual time=0.082..4.773 rows=243 loops=1)
                    -> Nested loop inner join  (cost=553.58 rows=1005) (actual time=0.073..3.534 rows=243 loops=1)
                        -> Filter: (h.StudentId <> '1039')  (cost=201.83 rows=1005) (actual time=0.049..0.650 rows=1005 loops=1)
                            -> Index range scan on h using PRIMARY  (cost=201.83 rows=1005) (actual time=0.047..0.469 rows=1005 loops=1)
                        -> Single-row index lookup on Hobby using PRIMARY (StudentId='1039', SportId=h.SportId)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=0 loops=1005)
                    -> Single-row index lookup on u using PRIMARY (UserId=h.StudentId)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=243)
 |
```

Query 2 - No index

```
| -> Limit: 15 row(s)  (cost=653.81 rows=15) (actual time=14.457..14.815 rows=15 loops=1)
    -> Nested loop inner join  (cost=653.81 rows=463) (actual time=14.455..14.812 rows=15 loops=1)
        -> Nested loop inner join  (cost=491.70 rows=463) (actual time=14.429..14.756 rows=15 loops=1)
            -> Sort: s.SportName  (cost=5.35 rows=46) (actual time=0.172..0.177 rows=38 loops=1)
                -> Table scan on s  (cost=5.35 rows=46) (actual time=0.087..0.110 rows=46 loops=1)
            -> Filter: ((n.rk <= 2) and (n.UserId is not null))  (cost=0.75..7.57 rows=10) (actual time=0.376..0.383 rows=0 loops=38)
                -> Index lookup on n using <auto_key2> (SportId=s.SportId)  (actual time=0.001..0.005 rows=26 loops=38)
                    -> Materialize CTE newtable  (cost=0.00..0.00 rows=0) (actual time=14.289..14.496 rows=1820 loops=1)
                        -> Window aggregate: row_number() OVER (PARTITION BY a.SportId ORDER BY a.cnt desc )   (actual time=11.306..12.282 rows=1820 loops=1)
                            -> Sort: a.SportId, a.cnt DESC  (actual time=1.556..1.704 rows=1820 loops=1)
                                -> Table scan on a  (cost=227.61 rows=2001) (actual time=0.001..0.181 rows=1820 loops=1)
                                    -> Materialize  (cost=0.00..0.00 rows=0) (actual time=11.296..11.564 rows=1820 loops=1)
                                        -> Table scan on <temporary>  (actual time=0.001..0.154 rows=1820 loops=1)
                                            -> Aggregate using temporary table  (actual time=8.883..9.154 rows=1820 loops=1)
                                                -> Nested loop inner join  (cost=902.70 rows=2001) (actual time=0.085..4.933 rows=2000 loops=1)
                                                    -> Filter: (r.CourtId is not null)  (cost=202.35 rows=2001) (actual time=0.057..1.520 rows=2000 loops=1)
                                                        -> Table scan on r  (cost=202.35 rows=2001) (actual time=0.056..1.298 rows=2000 loops=1)
                                                    -> Single-row index lookup on c using PRIMARY (CourtId=r.CourtId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=2000)
        -> Single-row index lookup on u using PRIMARY (UserId=n.UserId)  (cost=0.25 rows=1) (actual time=0.003..0.004 rows=1 loops=15)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Query 2 - Index on Reservation.UserId

```
| -> Limit: 15 row(s)  (cost=653.81 rows=15) (actual time=14.082..14.514 rows=15 loops=1)
    -> Nested loop inner join  (cost=653.81 rows=463) (actual time=14.080..14.511 rows=15 loops=1)
        -> Nested loop inner join  (cost=491.70 rows=463) (actual time=14.065..14.463 rows=15 loops=1)
            -> Sort: s.SportName  (cost=5.35 rows=46) (actual time=0.198..0.204 rows=38 loops=1)
                -> Table scan on s  (cost=5.35 rows=46) (actual time=0.087..0.124 rows=46 loops=1)
            -> Filter: ((n.rk <= 2) and (n.UserId is not null))  (cost=0.75..7.57 rows=10) (actual time=0.366..0.375 rows=0 loops=38)
                -> Index lookup on n using <auto_key2> (SportId=s.SportId)  (actual time=0.001..0.006 rows=26 loops=38)
                    -> Materialize CTE newtable  (cost=0.00..0.00 rows=0) (actual time=13.908..14.159 rows=1820 loops=1)
                        -> Window aggregate: row_number() OVER (PARTITION BY a.SportId ORDER BY a.cnt desc )   (actual time=10.675..11.743 rows=1820 loops=1)
                            -> Sort: a.SportId, a.cnt DESC  (actual time=1.513..1.680 rows=1820 loops=1)
                                -> Table scan on a  (cost=227.61 rows=2001) (actual time=0.001..0.157 rows=1820 loops=1)
                                    -> Materialize  (cost=0.00..0.00 rows=0) (actual time=10.668..10.964 rows=1820 loops=1)
                                        -> Table scan on <temporary>  (actual time=0.001..0.155 rows=1820 loops=1)
                                            -> Aggregate using temporary table  (actual time=8.328..8.601 rows=1820 loops=1)
                                                -> Nested loop inner join  (cost=902.70 rows=2001) (actual time=0.089..4.621 rows=2000 loops=1)
                                                    -> Filter: (r.CourtId is not null)  (cost=202.35 rows=2001) (actual time=0.063..1.425 rows=2000 loops=1)
                                                        -> Table scan on r  (cost=202.35 rows=2001) (actual time=0.061..1.218 rows=2000 loops=1)
                                                    -> Single-row index lookup on c using PRIMARY (CourtId=r.CourtId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=2000)
        -> Single-row index lookup on u using PRIMARY (UserId=n.UserId)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=15)
 |
```

Query 2 - Index on Court.SportId

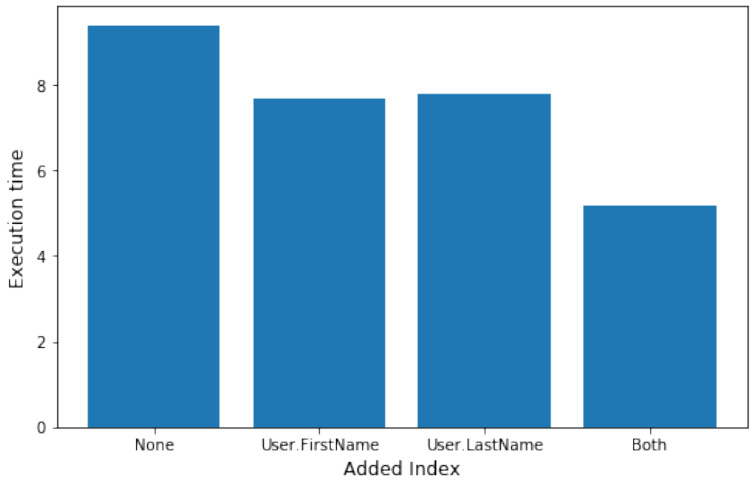
```
| -> Limit: 15 row(s)  (cost=653.81 rows=15) (actual time=13.880..14.236 rows=15 loops=1)
    -> Nested loop inner join  (cost=653.81 rows=463) (actual time=13.878..14.233 rows=15 loops=1)
        -> Nested loop inner join  (cost=491.70 rows=463) (actual time=13.870..14.198 rows=15 loops=1)
            -> Sort: s.SportName  (cost=5.35 rows=46) (actual time=0.207..0.211 rows=38 loops=1)
                -> Table scan on s  (cost=5.35 rows=46) (actual time=0.091..0.115 rows=46 loops=1)
            -> Filter: ((n.rk <= 2) and (n.UserId is not null))  (cost=0.75..7.57 rows=10) (actual time=0.361..0.368 rows=0 loops=38)
                -> Index lookup on n using <auto_key2> (SportId=s.SportId)  (actual time=0.001..0.005 rows=26 loops=38)
                    -> Materialize CTE newtable  (cost=0.00..0.00 rows=0) (actual time=13.698..13.905 rows=1820 loops=1)
                        -> Window aggregate: row_number() OVER (PARTITION BY a.SportId ORDER BY a.cnt desc )   (actual time=10.836..11.788 rows=1820 loops=1)
                            -> Sort: a.SportId, a.cnt DESC  (actual time=1.464..1.608 rows=1820 loops=1)
                                -> Table scan on a  (cost=227.61 rows=2001) (actual time=0.000..0.151 rows=1820 loops=1)
                                    -> Materialize  (cost=0.00..0.00 rows=0) (actual time=10.832..11.093 rows=1820 loops=1)
                                        -> Table scan on <temporary>  (actual time=0.001..0.156 rows=1820 loops=1)
                                            -> Aggregate using temporary table  (actual time=8.556..8.826 rows=1820 loops=1)
                                                -> Nested loop inner join  (cost=902.70 rows=2001) (actual time=0.087..4.584 rows=2000 loops=1)
                                                    -> Filter: (r.CourtId is not null)  (cost=202.35 rows=2001) (actual time=0.059..1.467 rows=2000 loops=1)
                                                        -> Table scan on r  (cost=202.35 rows=2001) (actual time=0.058..1.240 rows=2000 loops=1)
                                                    -> Single-row index lookup on c using PRIMARY (CourtId=r.CourtId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=2000)
        -> Single-row index lookup on u using PRIMARY (UserId=n.UserId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)
 |
```

Query 2 - Index on Reservation.UserId and Court.SportId

```
| -> Limit: 15 row(s)  (cost=653.81 rows=15) (actual time=13.264..13.633 rows=15 loops=1)
    -> Nested loop inner join  (cost=653.81 rows=463) (actual time=13.263..13.631 rows=15 loops=1)
        -> Nested loop inner join  (cost=491.70 rows=463) (actual time=13.242..13.572 rows=15 loops=1)
            -> Sort: s.SportName  (cost=5.35 rows=46) (actual time=0.233..0.238 rows=38 loops=1)
                -> Table scan on s  (cost=5.35 rows=46) (actual time=0.082..0.106 rows=46 loops=1)
            -> Filter: ((n.rk <= 2) and (n.UserId is not null))  (cost=0.75..7.57 rows=10) (actual time=0.343..0.351 rows=0 loops=38)
                -> Index lookup on n using <auto_key2> (SportId=s.SportId)  (actual time=0.001..0.005 rows=26 loops=38)
                    -> Materialize CTE newtable  (cost=0.00..0.00 rows=0) (actual time=13.042..13.253 rows=1820 loops=1)
                        -> Window aggregate: row_number() OVER (PARTITION BY a.SportId ORDER BY a.cnt desc )   (actual time=10.115..11.107 rows=1820 loops=1)
                            -> Sort: a.SportId, a.cnt DESC  (actual time=1.537..1.681 rows=1820 loops=1)
                                -> Table scan on a  (cost=227.61 rows=2001) (actual time=0.001..0.155 rows=1820 loops=1)
                                    -> Materialize  (cost=0.00..0.00 rows=0) (actual time=10.106..10.369 rows=1820 loops=1)
                                        -> Table scan on <temporary>  (actual time=0.002..0.169 rows=1820 loops=1)
                                            -> Aggregate using temporary table  (actual time=7.743..8.027 rows=1820 loops=1)
                                                -> Nested loop inner join  (cost=902.70 rows=2001) (actual time=0.092..4.296 rows=2000 loops=1)
                                                    -> Filter: (r.CourtId is not null)  (cost=202.35 rows=2001) (actual time=0.063..1.363 rows=2000 loops=1)
                                                        -> Table scan on r  (cost=202.35 rows=2001) (actual time=0.062..1.171 rows=2000 loops=1)
                                                    -> Single-row index lookup on c using PRIMARY (CourtId=r.CourtId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=2000)
        -> Single-row index lookup on u using PRIMARY (UserId=n.UserId)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=15)
 |
```

7.Indexing Analysis Report

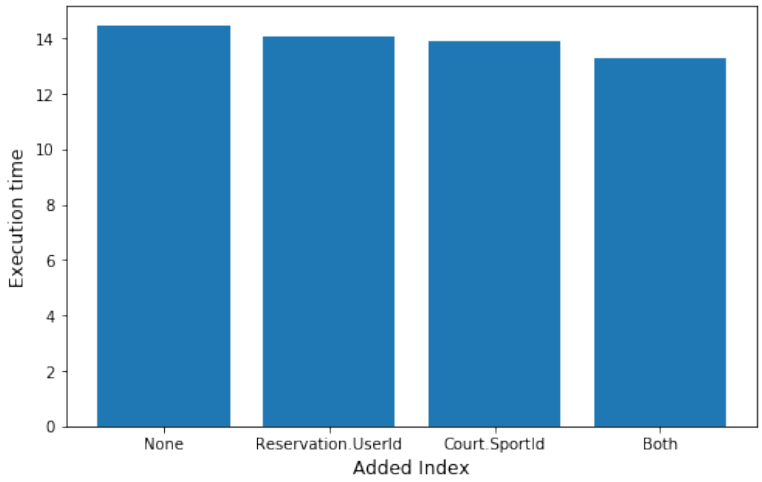
Query 1



Index candidates for this advanced query are set to be User.FirstName, User.LastName and both. We set these indices because our advanced query uses these two attributes to aggregate results and they might cost much time.

By observing execution time shown above, we find that adding more indices makes the whole query run more efficiently, and the improvement in time is significant. Execution time for using only User.FirstName or User.LastName are similar and it drops by 2 seconds from not using any index. Using both indices makes query execute most efficiently and its running time drops by 4 seconds from not using any index. Therefore, adding indices are good practices for this advanced query. We believe if we insert more data into schemas, adding indices of User.FirstName and User.LastName would improve execution time more significantly.

Query 2



In query 2, the UserId column of Reservation table and the SportId column of Court table are used in the groupby statement, which means they need to be compared with each other, so we decide to use them two as the indexes. The bar chart above clearly shows that adding any index shortens the runtime than using the original primary key index. As a result, both of the indexes can be used to minimize the runtime and gain performance improvements.