**Project Report**

**ON**

**SIT SPORTS MANAGEMENT SYSTEM**

**BACHELOR OF TECHNOLOGY**

**IN**

**INFORMATION TECHNOLOGY**

SUBMITTED BY

**Sai Krishana P. Agrawal – 17070124006**

**Harsh Raj – 17070124024**

**Karan Gulati – 17070124032**

**Keshav Basandrai – 17070124033**

**Raunak Verma - 17070124056**

**Utkarsh Singh – 17070124068**

****

**Under the Guidance of**

**PROF. VIJAYSHRI KHEDKAR**

**SYMBIOSIS INSTITUTE OF TECHNOLOGY**

**(A CONSTITUTENT OF SYMBIOSIS INTERNATIONAL UNIVERSITY)**

**Pune - 412115**

**2019-20**

**Software Requirement Specification**

**Table of Content**

1. **Problem Statement**
2. **Functional Requirements**
3. **Entities and Relationships**
4. **Entity Relationship Diagram**
5. **Relational Schema**
6. **Anomalies**
7. **Functional Dependencies**
8. **Normalization**
9. **Implementation**
10. **Quries, Functions, Procedures and Trigger**
11. **Problem Statement**

The current sports oversight system is manual and we aim to automise it so that there is no ambiguity in respect to any tournament taking place and all the information is readily available.

1. **Functional Requirements**
2. **Software:**

* Browser required to access the internet (Chrome v8 and above, Firefox 43 and above)

1. **Admin:**

* Only admin is responsible for managing the databases extensively.
* He/she can create the database provided that the same database has not been created before and can also delete it whenever required.
* He/she can insert, modify or update the data in our database.
* The faculty heads of each sport only have certain admin privileges including the right to insert, modify or update the databases whereas they may not be able to create new databases or delete databases.

1. **User:**

* We are considering new users as players.
* User needs to login with validated credentials.
* Once a new user has signed in to his/her account, they get access to view the sports they have already opted for.
* Once the user selects a particular sport, he/she can view various details like sport captains of respective branches, past achievements and also the best players the institute has produced in that respective sport
* Players can search the tournaments that they have played or are interested in playing.
* Players can view match schedule with detailed information.
* Players do also get the right to register themselves for new sports after selecting their respective branch.

1. **Business requirements:**

* User data must be displayed only if user has logged into his account.
* Check if every login is valid and new accounts are validated.

1. **Security requirements:**

* User should not be able to create two accounts.
* No one should be able to access an account without prior authorization.
* Should not be able to access additional features if not logged in.

1. **Hardware:**

* Any computer having minimum 512 MB RAM, 30GB HDD
* Sufficient network bandwidth.

1. **Entities and Relationships**
2. **Entities:**

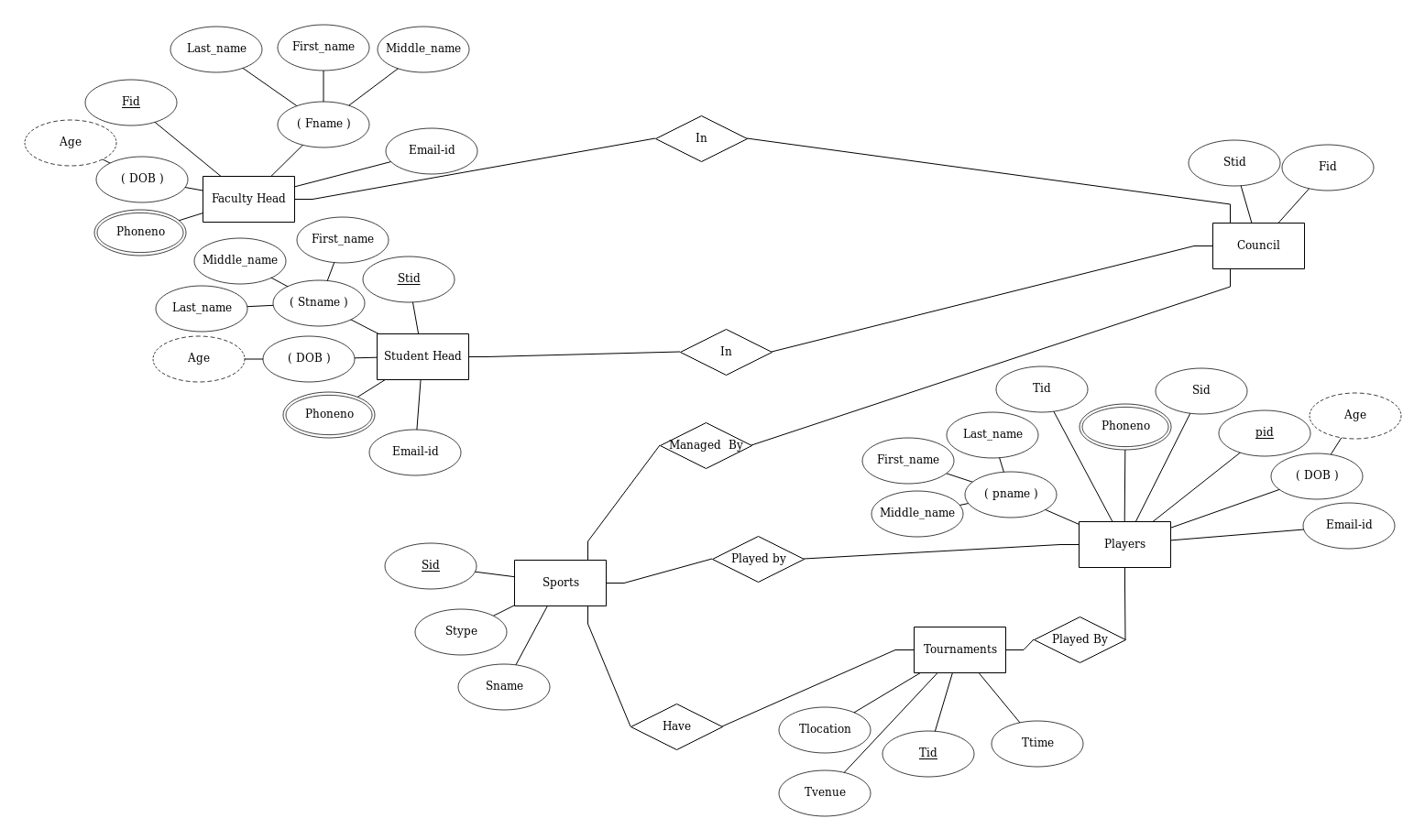
Student Head, Faculty Head, Council, Players, Tournaments and Sports

1. **Relation:**

In, Managed By, Played By, Have

* A council has Student Head details and Faculty Head details.
* Various Sports are played by the different players.
* Sports could have the tournaments.
* Tournaments have been played by the players.
* Sports are managed by the council.

1. **Entity Relationship Diagram**

****

1. **Relational Schema**

Student Head

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stid | Stname | Stadd | Email-id | DOB |

Faculty Head

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fid | Fname | Phoneno | Email-id | DOB |

Sports

|  |  |  |  |
| --- | --- | --- | --- |
| Sid | Sname | Stype | Pid |

Players

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pid | Phoneno | pname | DOB | Email-id |

Tournaments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tid | Tdate | Tvenue | Tlocation | sid |

Council

|  |  |
| --- | --- |
| Fid | Stid |

1. **Anomolies**

## Anomalies in the database can occur depending on the data inside the tables and the relationship between the tables. The anomalies can be update anomalies, deletion anomalies and insertion anomalies.

For managing the database perfectly, there should be no anomalies in the tables.

The three types of anomalies are following:

**Insert anomalies** − The nature of a database may be such that it is not possible to add a required piece of data unless another piece of unavailable data is also added. E.g. A library database that cannot store the details of a new member until that member has taken out a book.

**Deletion anomalies** − A record of data can legitimately be deleted from a database, and the deletion can result in the deletion of the only instance of other, required data, E.g. Deleting a book loan from a library member can remove all details of the particular book from the database such as the author, book title etc.

**Update anomalies** − Incorrect data may have to be changed, which could involve many records having to be changed, leading to the possibility of some changes being made incorrectly.

**The anomalies which can occur in our relational schema are:**

Insertion Anomaly: - Suppose a new player is added, until and unless he/she chooses the sports, their data cannot be inserted, otherwise their Sports information will be NULL.

Deletion Anomaly: For example, in the Sports table, two different information are kept together, Sports information and Player information. So, if sports records are deleted, we will also lose the player information of the related sports.

Updating Anomaly: What if some player leaves the tournament? In that case all the tournament records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency.

1. **Functional Dependencies**

Functional dependency is a relationship that exists when one attribute uniquely determines another attribute.

If R is a relation with attributes X and Y, a functional dependency between the attributes is represented as X->Y, which specifies Y is functionally dependent on X. Here X is a determinant set and Y is a dependent attribute. Each value of X is associated with precisely one Y value.

## Multi-valued dependency:

Multi-valued dependency occurs in the situation where there are multiple independent multi valued attributes in a single table. A multi-valued dependency is a complete constraint between two sets of attributes in a relation. It requires that certain tuples be present in a relation.

**Trivial Dependency:**

The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute.

So, X -> Y is a trivial functional dependency if Y is a subset of X.

In Player table, two columns pid and pname.

{pid, pname} -> pid is a trivial functional dependency as pid is a subset of {pid,pname}.

## Non trivial functional dependency:

Functional dependency which also known as a nontrivial dependency occurs when A->B holds true where B is not a subset of A. In a relationship, if attribute B is not a subset of attribute A, then it is considered as a non-trivial dependency.

{email} -> {pname} (if we know the email, we know the pname)

But pname is not a subset of email, and hence it's non-trivial functional dependency.

## Transitive Dependency:

A transitive is a type of functional dependency which happens when it is indirectly formed by two functional dependencies.

{email} -> {pname} (if we know the email, we know the player\_name)

{pname} -> {sports\_type} If we know the player\_name, we know the sports\_type

Therefore, according to the rule of rule of transitive dependency:

{email} -> {sports\_type} should hold, that makes sense because if we know the email name, we can know their sports.

1. **Normalization**

Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization rules are divided into the following normal forms:

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. BCNF
5. Fourth Normal Form

**First normal form:** A table is in the first normal form if it contains no repeating columns.

**Second normal form:** A table is in the second normal form if it is in the first normal form and contains only columns that are dependent on the whole (primary) key.

For a table to be in the Second Normal Form, it must satisfy two conditions:

1. The table should be in the First Normal Form.
2. There should be no Partial Dependency.

This is **Partial Dependency**, where an attribute in a table depends on only a part of the primary key and not on the whole key.

**Third normal form:** A table is in the third normal form if it is in the second normal form and all the non-key columns are dependent only on the primary key. If the value of a non-key column is dependent on the value of another non-key column, we have a situation known as transitive dependency. This can be resolved by removing the columns dependent on non-key items to another table.

### Advantage of removing Transitive Dependency

The advantage of removing transitive dependency is,

* Amount of data duplication is reduced.
* Data integrity achieved.

**Boyce-Codd Normal Form:**

For a table to satisfy the Boyce-Codd Normal Form, it should satisfy the following two conditions:

1. It should be in the **Third Normal Form**.
2. And, for any dependency A → B, A should be a **super key**.

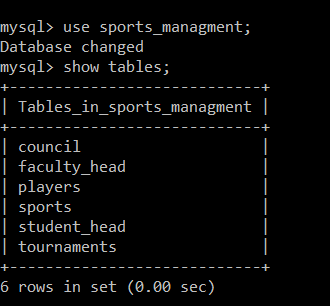
In simple words, it means, that for a dependency A → B, A cannot be a **non-prime attribute**, if B is a **prime attribute**.

## 4th Normal Form:

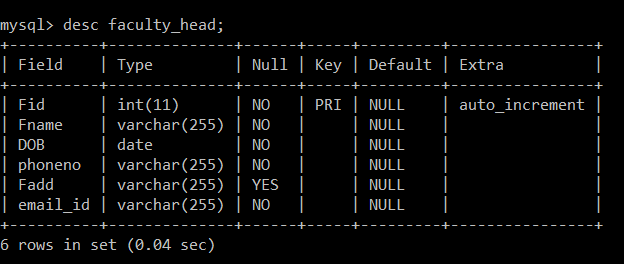
For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:

1. It should be in the **Boyce-Codd Normal Form**.
2. And, the table should not have any **Multi-valued Dependency**.
3. **Implementation**

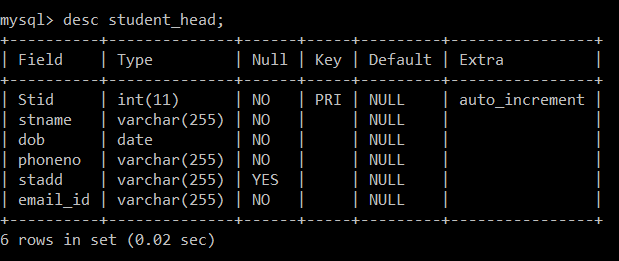
Create Tables:



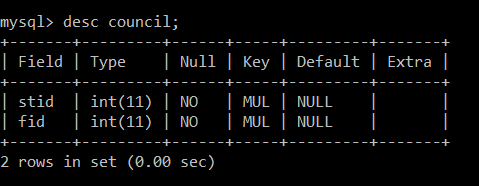
Faculty\_Head



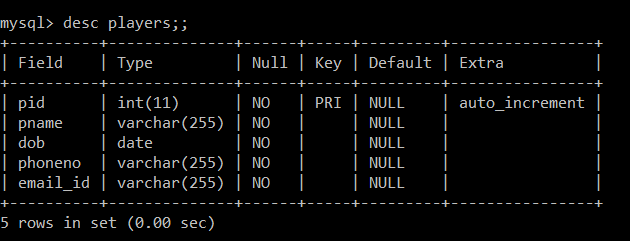
Student\_Head



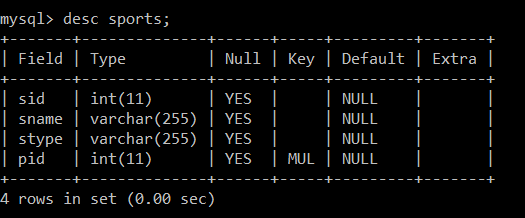
Council



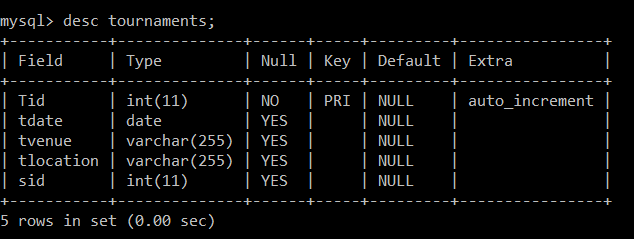
Players



Sports

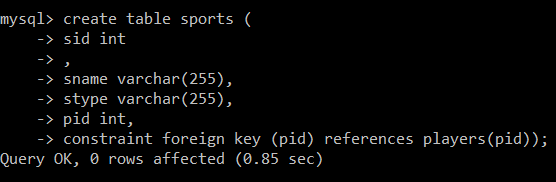


Tournaments

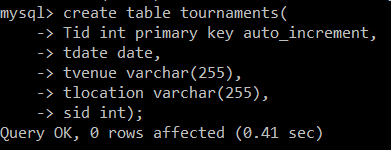


Create Statements:

Sports:

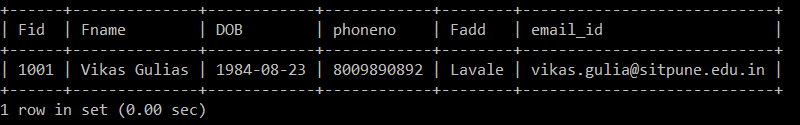


Tournaments:

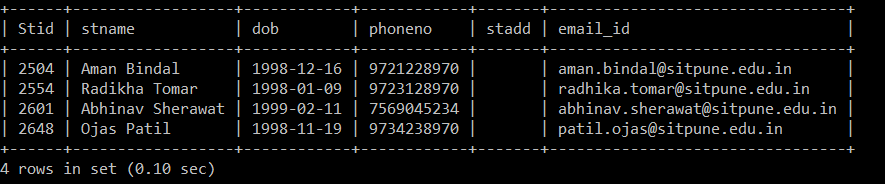


Insertion:

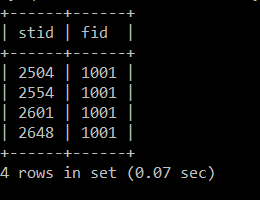
Faculty\_Head



Student\_Head

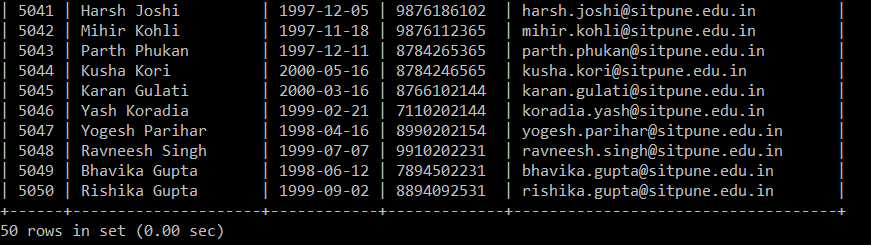


Council

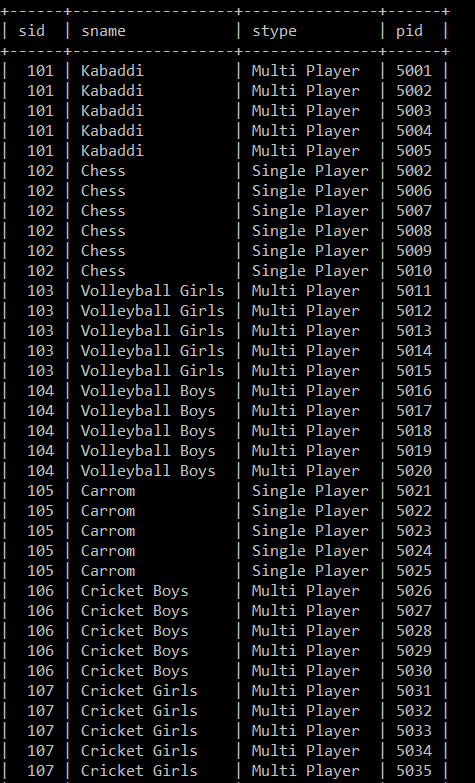


Players



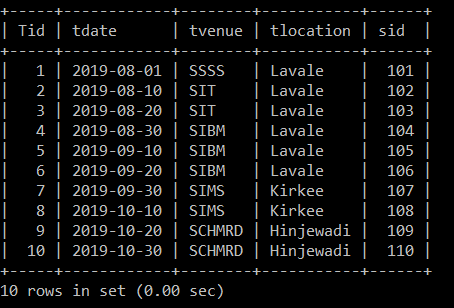


Sports



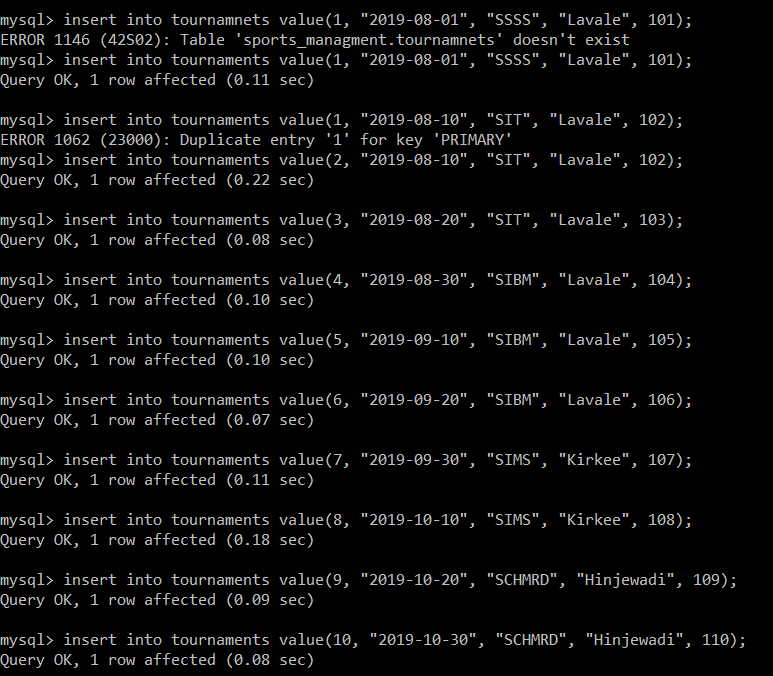


Tournaments

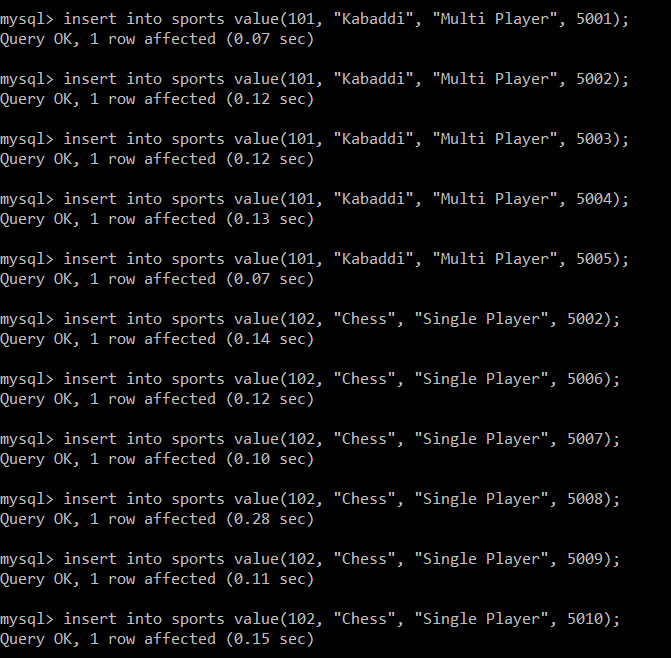


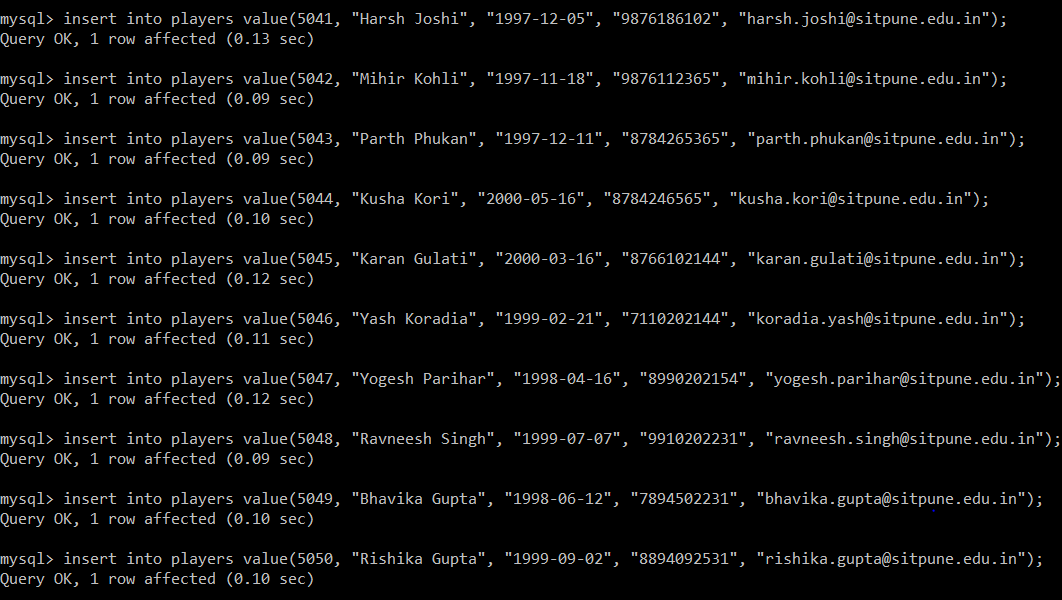
Insert Statements:

Tournaments:



Sports:



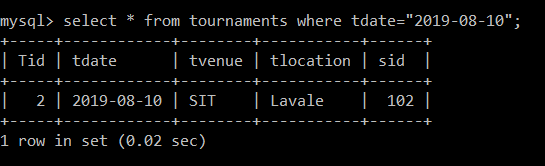
Players: 

1. **Queries, Functions, Procedures and Triggers**

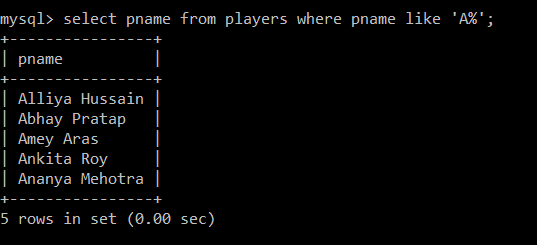
A **query** is a request for data or information from a database table or combination of tables. This data may be generated as results returned by Structured **Query** Language (**SQL**) or as pictorials, graphs or complex results, e.g., trend analyses from data-mining tools.

In our project we created this query generated using MySQL:

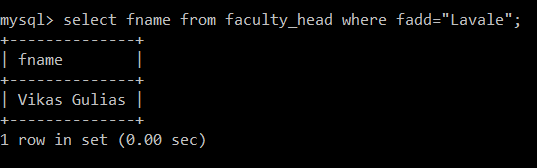
1)List the tournaments played on 10-08-2019?



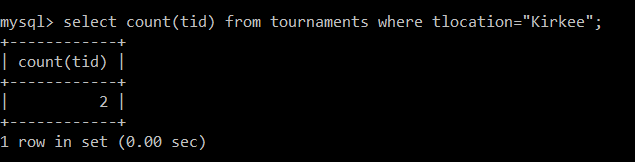
2)Find out players whose name start from “A”.



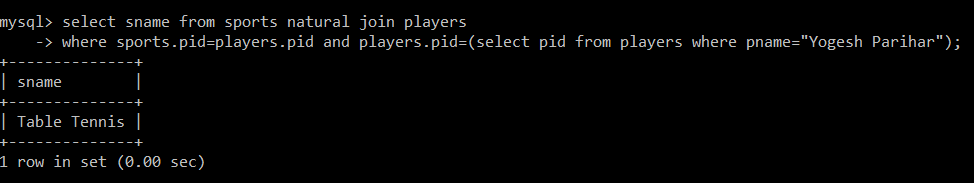
3)Give details of faculties who are part of sports council and are living in Pashan area



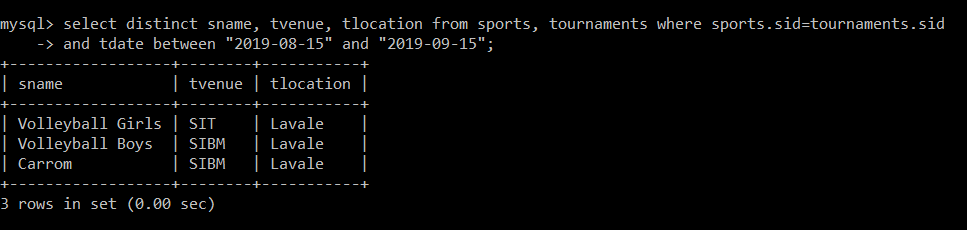
4)Find out number of SIU tournaments played in Symbiosis Kirkee Campus



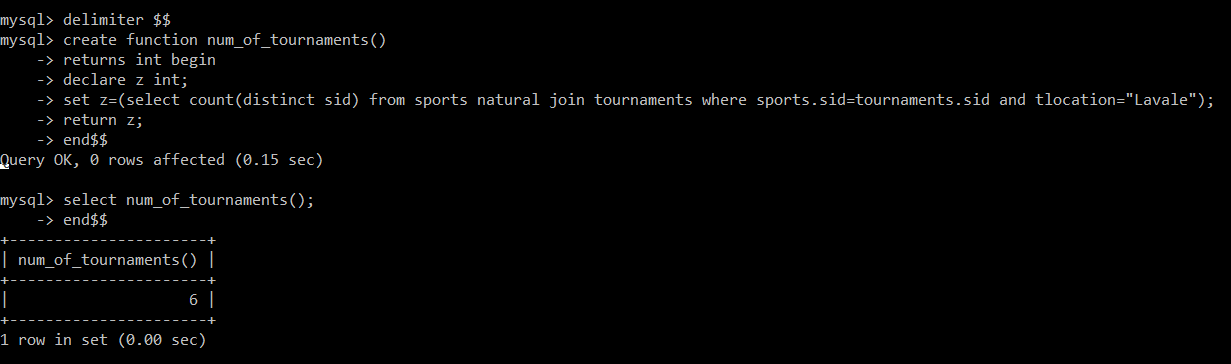
5)Give the name of sport played my ‘Yogesh Parihar’



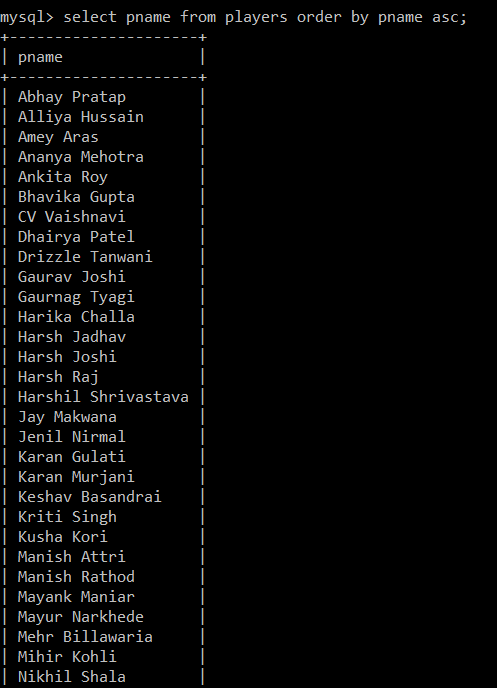
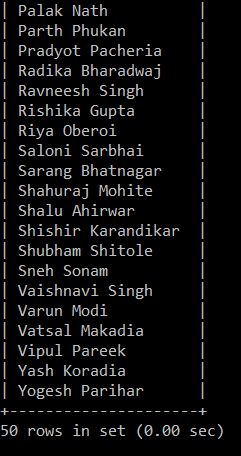
6)Find out tournament names and its location which are played between 15-08-2019 to 15-09-2019?



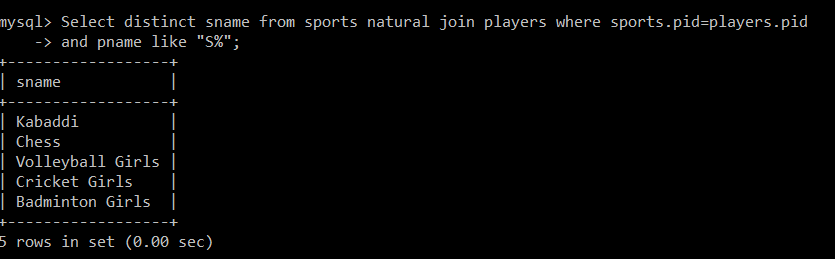
7) Write pl/sql function to find total number of tournaments played in SIU, Lavale campus



8)Give player names in increasing alphabetic order

9)Give name of sports having players whose name starts with ‘s’



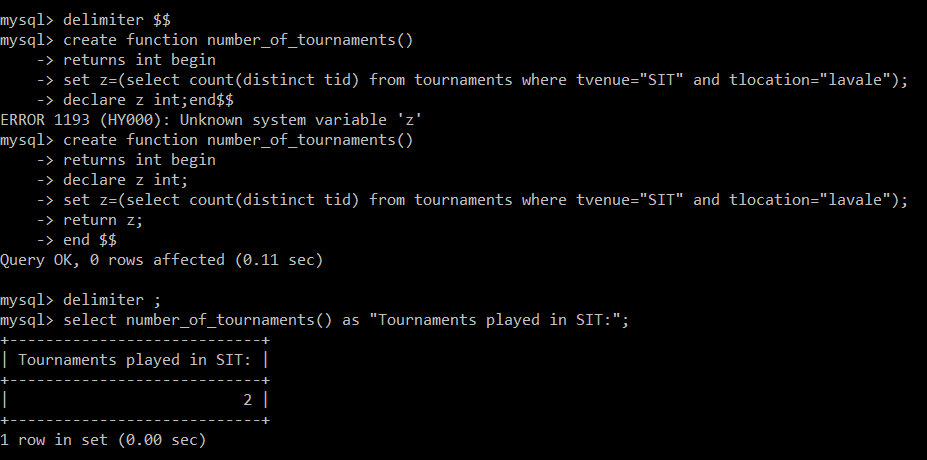
10) Give name of players whose phone no ends with ‘9’.



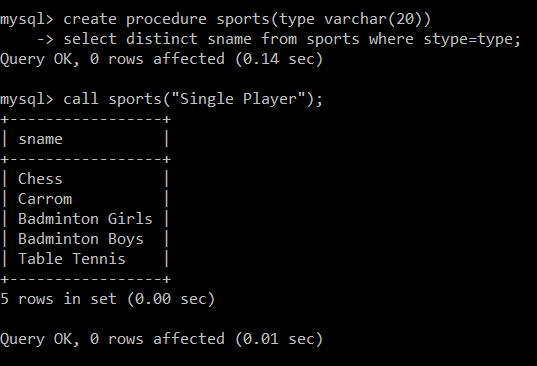
11) Give namewise details of sports



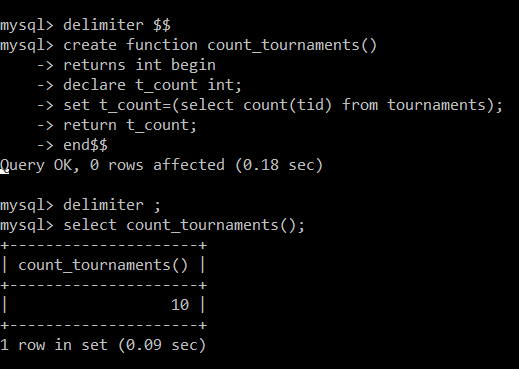
12) Write pl/sql function to find total number of tournaments played in SIT, Lavale campus



13) Using procedure list down the list of single player sports.



14) Using PL/SQL commands to count the total number of tournaments.



15) Trigger to store old value of tournaments in oldtournaments.

