KATHMANDU UNIVERSITY

SCHOOL OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MINI PROJECT

REPORT ON

Airport Management System

A second year mini project report submitted in partial fulfillment
of the requirements of COMP 232 by:

Sanjiv Gautam (17)
Pritam Shrestha (52)

CE 2nd year

Date of Submission : <Date> July 2018

Airport Management System

Description:

Objective:

1. To understand the concept and utilization of database management system.
2. To develop a database regarding airport management system
3. To create a simple web application to perform and demonstrate the basic operations on the created database including insert, delete, update and search.

System Design:

1. Programming Language: Python and PHP
2. Database Management System: MySQL

Scenario:

This project has been introduced with sole purpose of designing the database for airport. The system may come handy for government parties responsible for flight management. This project aims to help officers to acquire the statistics about the flight in quick succession. The system will be built on WEB UI Platform and have a user friendly interface that can be used and navigated without any problems. The user will be able to search for employees, flights, people abroad. Moreover, the data can also be used to see the brain drain of the country, which country is luring people from Nepal, which is useful information for government as well.

This report however is for employees working in the airport. So, we would be focusing on employees and their information.

Schemes:

The user will be able to search through the statistics of employees and retrieve the information they need. They can also insert new records into the database or delete a record from the database. The users also have the option to update the values of some of the attributes as per requirement.

Entities and their attributes

1. **Employees**: consists information about employees working in the airport, their section, designation, joining_date e.t.c

Attributes:

EMPID
Full_Name
Address
Branch
Designation
Email
Mobile_number
Salary

Functional dependencies(F) and Closure of F (F$_+$)

EMPID -> full_name
EMPID -> address
EMPID -> branch
EMPID -> designation
EMPID -> email
EMPID -> mobile_number
EMPID -> salary

mobile_number,email -> address
mobile_number,email -> branch
mobile_number,email -> designation
mobile_number,email -> salary
mobile_number,email -> EMPID
branch -> designation
Designation -> salary

Closure of mobile_number, email = { Full_Name Address Branch Designation Email , Mobile_number ,Salary, EMPID }

Since EMP_ID is primary key, its closure gives all values

So our super keys are EMPID and mobile_number, email and candidate key is EMPID.

Normalization

1NF

Since the attribute email in Employee table could be multi valued, we decompose the relation Employee into

Employee (EMPID, full_name, address, branch, designation, mobile_number, salary)
Email (EMPID, Email)

All other attributes are single valued. Hence the database is reduced to

1NF form.

2NF

From our functional dependency , we could see that there is no partial dependency as there is no such key alpha and beta in alpha-> beta , such that alpha is part of prime attribute and beta is non prime attribute.

3NF

For a functional dependency to be in 3NF alpha - > beta , alpha must be a non prime attribute, and beta must be non prime.

Since designation attribute is functionally dependent in branch, and both of them are non prime attribute, this is a clear cut case of transitive dependency. So, 3NF doesn't allow this. So, we make another table where branch acts as candidate key with another attribute of designation

Also, salary, being non prime attribute is entirely dependent on designation, and both are non prime, we make a separate table where designation is candidate key, and salary is its other attribute

Employee (EMPID, full_name, address, branch, designation, mobile_number, salary)
Email (EMPID, Email)
Branch (branch, designation)
Email (designation, salary)

BCNF form:

In the above schemas, all the determinants are candidate keys which uniquely identifies each record. So the above schema is already in BCNF form.
We might get confused at mobile_number,email -> EMPID , but mobile_number, email is a already a super key.

Primary Keys

The underlined attributes in each entity are the primary keys of the corresponding entities because it uniquely identifies each entity.

1. Employee (EMPID, full_name, address, branch, designation, mobile_number, salary)
2. Email (EMPID, Email)
3. Branch (branch, designation)
4. Email (designation, salary)

EMPID

The empid is generated by sequential adding of user i.e. the 1st user gets 1, 2nd gets 2 and so on.

Foreign Keys

The foreign keys used to link the parent and child tables. Some of the foreign keys are:

1. EMPID is a foreign key in table 'Email' and is a primary key in table Employee'. So Email is child table for parent table 'Employee' .

Referential Integrity

We used referential integrity constraints primary key and foreign key.

Referential Integrity

We used referential integrity constraints primary key and foreign key.

Examples in our database:

```
ALTER TABLE `Goals_Team` ADD CONSTRAINT `Goals_id`
FOREIGN KEY (`Goals_id`) REFERENCES `Goals`(`Goals_id`)

ALTER TABLE `Fouls_Player` ADD CONSTRAINT `Fouls_id`
FOREIGN KEY (`Fouls_id`) REFERENCES `Fouls`(`Fouls_id`)
```

```
ALTER TABLE `Team` ADD CONSTRAINT
`Club_id` PRIMARY KEY (`Club_id`)
```

```
ALTER TABLE `Player` ADD CONSTRAINT
`Player_id` PRIMARY KEY (`Player_id`)
```

Cascading actions in foreign key constraint

We used on delete cascade and on update cascade in all our foreign key constraints.

Examples in our database:

```
ALTER TABLE `team` ADD CONSTRAINT `Team_teamname`
FOREIGN KEY (`Club_name`) REFERENCES
`team_info`(`Club_name`) ON DELETE CASCADE
ON UPDATE CASCADE;
```

```
ALTER TABLE `player_info` ADD CONSTRAINT
`Playername_team` FOREIGN KEY (`Club_id`) REFERENCES
`team`(`Club_id`) ON DELETE CASCADE
ON UPDATE CASCADE;
```

Check Constraints

The check constraints used in our database are:

```
alter table points add CONSTRAINT
Percentage check (Percentage<100);
```

```
alter table standingspoints add CONSTRAINT
standings check (standings BETWEEN 1 and 10);
```

However, MySQL does not support SQL **check constraints**. We can define them in your DDL query for compatibility reasons but they are just ignored.

Weak Entity:

In our database, the entity **Position** is a weak entity set since it doesn't have any primary key but has Player_id as a discriminating key. The identifying entity seat for this weak entity set is entity **Player**. There is many to one relationship from the weak entity set Position to identifying entity set Player since one player can be associated to many positions and Position has total participation since each position must be associated to a player.

Indexing:

In our database, we have implemented indexing on various attributes for faster operations especially for searching of records using the SQL queries:

```
create index player_name on player_info(player_name);
create index Club_name on Team(Club_name);
```

We created an index on each of the foreign keys as well as on Club_name of the table Team and on Player_name of the table Player because we search by name of players and teams in our database.

Join

We used natural join and inner join because each record in a table had an association to the table to be joined. So the multiple tables had only the common records.

Examples used in our database:
Natural join:

1. Team (<u>Club_id, Club_name</u>)
2. Goals_Team (<u>Goals_Team_id, Club_id, Goals_id</u>)
3. Goals (<u>Goals_id</u>, Goals_for, Goals_conceded, Goal_difference,Home_Goals, Away_Goals)

```
SELECT Club_name,Goals_for,Goals_conceeded,Goal_difference,Home_Goals,Away_Goals

FROM team NATURAL join goals_team
NATURAL join goals
order by Goal_difference desc;
```

1. Points_Away (<u>Points_Away_id, Club_id</u>, <u>Points_id)</u>
2. Points (<u>Points_id</u>, Points_won, Points_dropped, Percentage)
3. Team (<u>Club_id, Club_name</u>)

```
SELECT Club_name,Points_won,Points_dropped,Percentage
FROM points natural join points_away natural join team order
by Percentage desc;
```

Inner join:

1. Player (Player_id, Player_name)
2. Attacking_Output_Player (Attacking_Player_id, Attack_id, Player_id)
3. Attacking_Output (Attack_id, Goals, Assists, Attacking_Output)

```
Select
player.Player_name,attacking_output.Goals,attacking_output.Assists,attacking_output.Attacking
_Output

FROM attacking_output inner join attacking_output_player
ON attacking_output.Attack_id=attacking_output_player.Attack_id
 inner join player

on attacking_output_player.Player_id=player.Player_id
order by Attacking_Output desc;
```

1. Standings (Standings_id, Club_id, Standings)
2. StandingsPoints (Points, Standings)
3. Team (Club_id, Club_name)

```
SELECT team.Club_Name,standingspoints.Points,standingspoints.Standings
FROM standings inner join standingspoints

ON standings.Standings=standingspoints.Standings
inner join team

on standings.Club_id=team.Club_id
order by standings;
```