PES UNIVERSITY
100 feet Ring Road, BSK 3rd Stage, Bengaluru 560085

Department of Computer Science and Engineering
Jan – May 2020

UE18CS252
Database Management Systems

Project Report

Natural History Museum

PES1201800722 Sanjna Chaturvedi
4th Sem Sec.A Roll No. 24

PROJECT SUMMARY

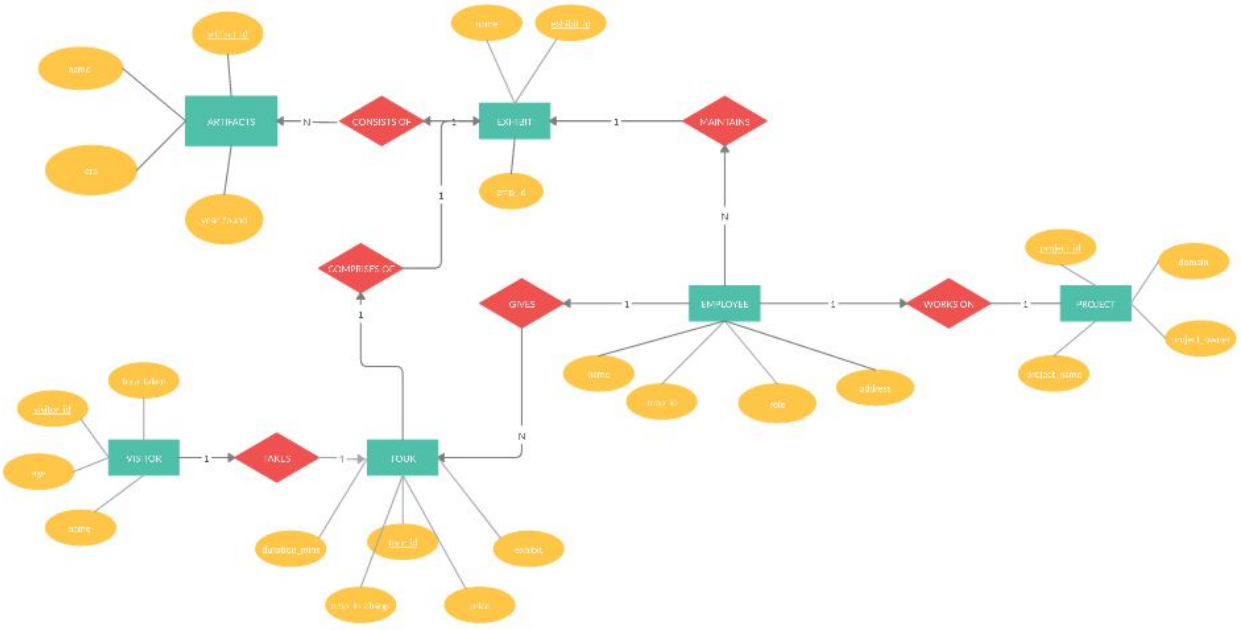
This Project covers the various aspects that went into making a database for a Natural History Museum. Right from the design using ER diagrams and relational schemas ensuing that the database was normalized to the implementation using data definition language. All the way to writing queries to test the validity of our database.

Introduction

The daily operations of a natural history museum has several moving parts. On one side, there is the maintenance and record keeping of the various artifacts across the different exhibits. Then we have the coming and going of tourists taking the various torus the museum has to offer. And finally, we have the museums independent projects to help expand on the research of the sciences. All three of these aspects have to be managed and executed by the employees of the museum.
The following database aims at capturing all of these details in a concise and executable maner, ensuring that every employee knows where to go and what to do, all artifacts, visitors and research projects are well documented and accounted for.

Data Model

ER diagram.



EMPLOYEE

emp_id	role	address	name

TOUR

tour_id	duration_mins	exhibit	emp_in_charge	price

VISITOR

visitor_id	tour_taken	name	age

EXHIBIT

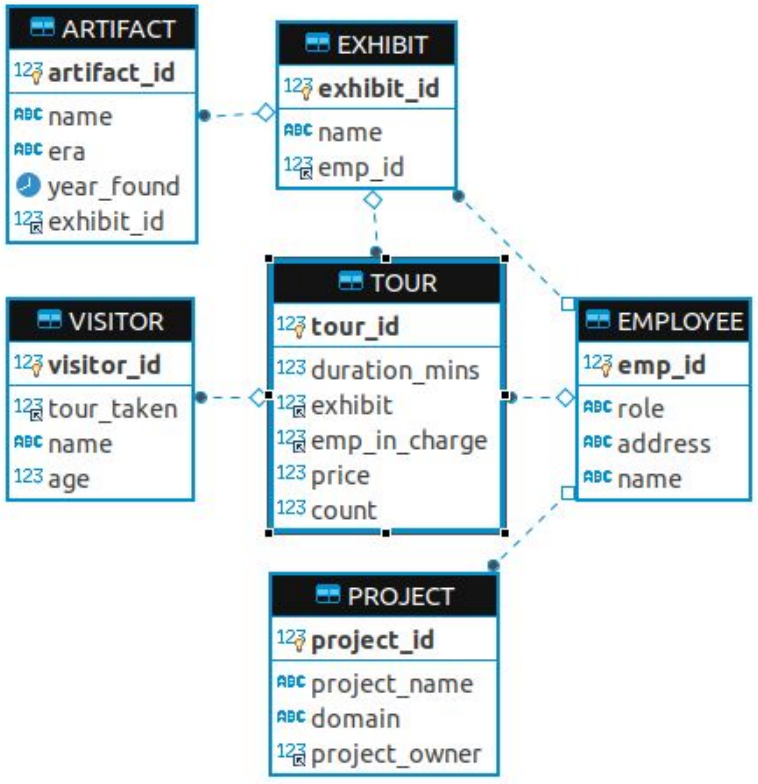
name	exhibit_id	emp_id

ARTIFACT

artifact_id	name	era	year_found	exhibit_id

PROJECT

project_id	project_name	domain	project_owner



As can be seen, there are 6 tables that help keep track of the various aspects of the museum. This can be divided into 3 different parts to understand the various functionalities better.

Part 1: The Artifacts

- Each artifact has a unique id associated with it, so that one can identify the name, era, year it was discovered and find the exhibit it belongs to using the exhibit id which is its forign key. The exhibit helps keep track of the various employees that are responsible for taking care of the exhibit.

Part 2: The visitors

- Each visitor has a unique id that helps keep track of what tour they have taken, this will automatically trigger the count column in the tour table to keep track of how many visitors took each given tour. Which also ties back into the employee table that keeps track of which employee is responsible for conducting a given tour.

Part 3: The projects.

- Each project has a specific domain and links back to the researchers who project belongs to as the foreign key.

FD and Normalization

What are the functional dependencies in this database?
Functional dependencies are the set of constraints between two attributes in a relation.
EMPLOYEE
{emp_id -> role, emp_id->address, emp_id->name}
PROJECT
{project_id->project_name, project_id->domain->project_id->project_owner}
TOUR
{tour_id->duration_mins,tour_id->exhibit, tour_id->emp_in_charge, tour_id->price, tour_id->count}
VISITOR
{visitor_id->tour_taken,visitor_id->name, visitor_id->age}
EXHIBIT
{exhiit_id -> name, exhibit_id->emp_id}
ARTIFACT
{artifact_id->name, artifact_id->era, artifact_id->year_found, artifact_id->exhibit_id}

When is the second normal form violated?
Second normal form is violated when there are attributes that are partially dependent on each other. For example, if we add employee_name to the exhibit table, that would violate the second normal form. Another way to violate the second normal form is to include the exhibit name with the artifact name, thus making a partially dependent entry.

When is the third normal form violated?
Third normal form is violated when we have transitive dependencies. This means that a non primary attribute depends on another non primary attribute instead of the primary attribute. For example, a third normal form is violated when we have a subject specification for the domain in the project id, or if we were to add a relationship between the visitor and the employee table.

DDL

```
CREATE TABLE `ARTIFACT` (
  `artifact_id` int(5) NOT NULL,
  `name` varchar(30) NOT NULL,
  `era` varchar(30) DEFAULT NULL,
  `year_found` date DEFAULT NULL,
  `exhibit_id` int(5) NOT NULL,
  PRIMARY KEY (`artifact_id`),
  KEY `exhibit_id` (`exhibit_id`),
  CONSTRAINT `ARTIFACT_ibfk_1` FOREIGN KEY (`exhibit_id`) REFERENCES `EXHIBIT` (`exhibit_id`)
)

CREATE TABLE `EMPLOYEE` (
  `emp_id` int(5) NOT NULL,
  `role` varchar(25) NOT NULL,
  `address` varchar(50) NOT NULL,
  `name` varchar(50) NOT NULL,
  PRIMARY KEY (`emp_id`)
)

CREATE TABLE `EXHIBIT` (
  `exhibit_id` int(5) NOT NULL,
  `name` varchar(50) NOT NULL,
  `emp_id` int(5) NOT NULL,
  PRIMARY KEY (`exhibit_id`),
  KEY `emp_id` (`emp_id`),
  CONSTRAINT `EXHIBIT_ibfk_1` FOREIGN KEY (`emp_id`) REFERENCES `EMPLOYEE` (`emp_id`) ON
UPDATE CASCADE
)
CREATE TABLE `VISITOR` (
  `visitor_id` int(5) NOT NULL,
  `tour_taken` int(5) NOT NULL,
  `name` varchar(50) DEFAULT NULL,
  `age` int(3) DEFAULT NULL,
  PRIMARY KEY (`visitor_id`),
  KEY `tour_taken` (`tour_taken`),
  CONSTRAINT `VISITOR_ibfk_1` FOREIGN KEY (`tour_taken`) REFERENCES `TOUR` (`tour_id`)
)

CREATE TABLE `TOUR` (
  `tour_id` int(5) NOT NULL,
  `duration_mins` int(5) DEFAULT NULL,
  `exhibit` int(5) DEFAULT NULL,
  `emp_in_charge` int(5) DEFAULT NULL,
  `price` int(5) DEFAULT NULL,
  `count` int(5) DEFAULT NULL,
  PRIMARY KEY (`tour_id`),
```

```
  KEY `emp_in_charge` (`emp_in_charge`),
  KEY `exhibit` (`exhibit`),
  CONSTRAINT `TOUR_ibfk_1` FOREIGN KEY (`emp_in_charge`) REFERENCES `EMPLOYEE`
(`emp_id`),
  CONSTRAINT `TOUR_ibfk_2` FOREIGN KEY (`exhibit`) REFERENCES `EXHIBIT` (`exhibit_id`)
)
CREATE TABLE `PROJECT` (
  `project_id` int(5) NOT NULL,
  `project_name` varchar(50) DEFAULT NULL,
  `domain` varchar(50) NOT NULL,
  `project_owner` int(5) NOT NULL,
  PRIMARY KEY (`project_id`),
  KEY `project_owner` (`project_owner`),
  CONSTRAINT `PROJECT_ibfk_1` FOREIGN KEY (`project_owner`) REFERENCES `EMPLOYEE`
(`emp_id`) ON UPDATE CASCADE
)
```

Triggers

The trigger created for this database counts the number of visitors taking each tour. Everytime we add a new visitor to our database, we keep track of what tour they took. And we increment the number of visitors taking that tour in the tour column.

```
CREATE TRIGGER VIS_UPDATE
    -> AFTER INSERT ON VISITOR
    -> FOR EACH ROW
    -> UPDATE TOUR, VISITOR
    -> SET TOUR.count = TOUR.count + 1
    -> WHERE TOUR.tour_id = VISITOR.tour_taken;
```

	visitor_id	tour_taken	name	age
1	10,001	1	James	30
2	10,002	2	Susan	28
3	10,003	2	Mary	28
4	10,004	4	Rick	43
5	10,005	4	Jacky	43
6	10,006	4	Phil	23
7	10,007	4	Victor	23
8	10,008	2	Arpit	23
9	10,009	2	Markus	24
10	10,010	2	Zack	24
11	10,011	3	Manny	18
12	10,012	3	Jithin	21
13	10,013	10	Adi	21

	tour_id	duration_mins	exhibit	emp_in_charge	price	count
1	1	60	101	1	20	1
2	2	60	102	2	20	5
3	3	60	103	3	20	2
4	4	120	104	4	40	4
5	5	120	101	5	40	0
6	6	120	102	6	40	0
7	7	120	108	1	40	0
8	8	120	109	2	40	0
9	9	120	110	2	40	0
10	10	120	110	3	40	1

SQL Queries

Using nested queries.

1. List all exhibits that have tours associated with them.

```
mysql> SELECT name FROM EXHIBIT WHERE EXHIBIT.exhibit_id IN (SELECT exhibit FROM
 TOUR);
+---------------------+
| name                |
+---------------------+
| Ancient history     |
| Edwardian era       |
| Greek Dark Ages     |
| Classical antiquity |
| Middle Ages         |
| Victorian era       |
| Viking Age          |
+---------------------+
7 rows in set (0.00 sec)
```

The same question can be answered using a full outer join as well.

```
mysql> SELECT DISTINCT(name) from EXHIBIT FULL JOIN TOUR ON exhibit_id= exhibit;

+---------------------+
| name                |
+---------------------+
| Ancient history     |
| Edwardian era       |
| Greek Dark Ages     |
| Classical antiquity |
| Middle Ages         |
| Victorian era       |
| Viking Age          |
+---------------------+
7 rows in set (0.00 sec)
```

2. Which employees work on projects under the Mesoproterozic domain?

```
mysql> SELECT e.name
    -> FROM EMPLOYEE e
    -> WHERE e.emp_id IN (SELECT p.project_owner FROM PROJECT p WHERE p.`domain`
='Mesoproterozic');
+------+
| name |
+------+
| Alex |
+------+
1 row in set (0.00 sec)
```

Using outer joins

1. List out all exhibits that have tours and provide a count.

This question can be answered using a RIGHT OUTER

```
mysql> SELECT name, COUNT(name)
    -> FROM EXHIBIT e RIGHT OUTER JOIN TOUR t2
    -> ON e.exhibit_id = t2.exhibit
    -> GROUP BY name;
+---------------------+-------------+
| name                | COUNT(name) |
+---------------------+-------------+
| Ancient history     |           2 |
| Classical antiquity |           1 |
| Edwardian era       |           2 |
| Greek Dark Ages     |           1 |
| Middle Ages         |           1 |
| Victorian era       |           1 |
| Viking Age          |           2 |
+---------------------+-------------+
7 rows in set (0.00 sec)
```

2. Count the total number of visitors taking a certain tour.

This can be achieved using a LEFT OUTER JOIN

```
mysql> SELECT t.tour_id, t.total_num, e.name
    -> FROM  TOUR t
    -> LEFT OUTER JOIN EXHIBIT e
    -> ON e.exhibit_id =t.exhibit
    -> ORDER by t.total_num DESC ;
+---------+-----------+---------------------+
| tour_id | total_num | name                |
+---------+-----------+---------------------+
|       2 | 5         | Edwardian era       |
|       4 | 4         | Classical antiquity |
|       3 | 2         | Greek Dark Ages     |
|       1 | 1         | Ancient history     |
|      10 | 1         | Viking Age          |
|       5 | 0         | Ancient history     |
|       6 | 0         | Edwardian era       |
|       7 | 0         | Middle Ages         |
|       8 | 0         | Victorian era       |
|       9 | 0         | Viking Age          |
+---------+-----------+---------------------+
10 rows in set (0.00 sec)
```

Had we not included the trigger to automatically count the number of visitors in each tour, we would have had to join that table as well to get the total count of each visitor. here, we have used 3 inner joins.

```
mysql> SELECT  COUNT(e.name),e.name, t.tour_id
    -> FROM VISITOR v
    -> JOIN TOUR t
    -> ON v.tour_taken = t.tour_id
    -> JOIN EXHIBIT e
    -> ON t.exhibit = e.exhibit_id
    -> GROUP BY e.name
    -> ORDER BY COUNT(e.name) DESC;
+---------------+---------------------+---------+
| COUNT(e.name) | name                | tour_id |
+---------------+---------------------+---------+
|             5 | Edwardian era       |       2 |
|             4 | Classical antiquity |       4 |
|             2 | Greek Dark Ages     |       3 |
|             1 | Viking Age          |      10 |
|             1 | Ancient history     |       1 |
+---------------+---------------------+---------+
5 rows in set (0.00 sec)
```

Using aggregated queries

1. What are the various jobs an employee can hold and how many employees are present?

```
mysql> SELECT e.job, COUNT(e.emp_id )
    -> FROM EMPLOYEE e
    -> GROUP BY e.job;
+------------+------------------+
| job        | COUNT(e.emp_id ) |
+------------+------------------+
| help       |                4 |
| researcher |                4 |
| tour_guide |                6 |
+------------+------------------+
3 rows in set (0.00 sec)
```

2. Which exhibit has the most number of artifacts?

```
mysql> SELECT e.name, COUNT(a.artifact_id)
    -> FROM  EXHIBIT e right outer Join ARTIFACT a
    -> ON e.exhibit_id = a.exhibit_id
    -> GROUP BY e.name
    -> ORDER BY COUNT(a.artifact_id) DESC
    -> limit 1;
+-----------------+----------------------+
| name            | COUNT(a.artifact_id) |
+-----------------+----------------------+
| Ancient history |                    9 |
+-----------------+----------------------+
1 row in set (0.00 sec)
```

3. Which tour has the most visitors?

```
mysql> SELECT t.tour_id, MAX(t.total_num) , e.name
    ->  FROM  TOUR t
    ->  LEFT OUTER JOIN EXHIBIT e
    ->  ON e.exhibit_id =t.exhibit
    ->  ORDER by t.total_num DESC ;
+---------+------------------+-----------------+
| tour_id | MAX(t.total_num) | name            |
+---------+------------------+-----------------+
|       1 | 5                | Ancient history |
+---------+------------------+-----------------+
1 row in set (0.00 sec)
```

Conclusion

Capabilities

In conclusion, this database is capable of the following.

1. Effectively keeping track of all the artifacts and in various exhibits.
2. Maintaining records of which employee is responsible for a given exhibit.
3. Keeping tabs on the visitors coming in along with the tours they have decided to take.
4. Having details on which employee is responsible for conducting a given tour.
5. Managing the various details regarding projects researchers working at the museum have taken

Limitations

The database has the following limitations

1. It does not keep track of the date's a visitor came. This means that we need to reset this table along with the count on the total_num of tour's taken every day.

Future enhancements

1. Including the date functionality.