**DBM1: SEOUL Public Bicycle**

Haneul KIM, Hayeong LEE, Joohwan HAN

1. **Introduction**

Currently, many countries use the public bicycle system. Since the first public bicycle was introduced in Amsterdam in the 1960s, the public bicycle system has been promoted in urban cities around the world, such as Paris, Barcelona, Berlin, Montreal, and Salt Lake. It was created out of the necessity of the consumer class that a bus cannot go to the desired destination, and that the bus is more expensive than a bicycle. In addition, it has the advantage of reducing traffic congestion and reducing noise and air pollution by reducing the use of automobiles (Lin, J. R., & Yang, T. H. 2011). The public bicycle system has already been established, and the interaction between storage and bicycles is also automated. Using the data generated here, our purpose is to find the questions we are curious about from the point of view of companies and users with SQL.



figure 1 Public bicycle in Seoul

This report contains the process of using data from public bicycles operated by the Seoul city, South Korea, to answer questions from the perspective of users and companies through queries. We used data provided by Seoul, Korea, and completed the query using SQL and python.

1. **Dataset and Preprocessing**

We make use of database from “서울열린데이터광장” which is data resource from capital of South of Korea. (<https://data.seoul.go.kr/>) We selected some data in a lot of databases to figure out our purpose. Therefore we used “Data of station”, “History of broken”, “History of usage”, “User information”, “Usage” and also determined to use data of June 2022. For history of bicycle, there is so many data which is around 1 million. We extracted some data randomly to handle using python. It is hard to find information which is the final station of each bicycle from original data. Accordingly, we traced where the bicycle was going and found the final destination, besides we also found total used time and total distance as python.

Finally, we made new dataset which we can use easily through merging and subtracting. There are “Bicycle\_final”(34892 columns), “Broken”(74924 columns), “Employee”(2627 columns), “History\_random”(10486 columns), “Locations”(8438 columns), “Station”(2623 columns) and “Users\_final”(82 columns).

1. **Database Schema and ER Diagram**

We considered that someone rides the bicycle at the station at the time. In this situation, we can find three strong entities which are ‘*Users’*, *‘Station’,* and *‘Bicycle’*. These three entities are related each other’s. Therefore, we made new weak entity, *‘History’*, which makes strong entities connect. Also, the station is managed by employee which is another weak entity. Finally, we have three strong entities and two weak entities.

We started from ‘Users’ entity. There is no one attribute to distinguish users but three attributes represent to user. Thus, we selected ‘Age’, ’Sex’, and ‘Type’ to primary key. The primary key of other entities would be each ID obviously. ‘Users’ can access to ‘History’ as relationship set which is ‘Usage’. User can use bike as he or she want. And one history has only one user. The relation between ‘History’ and ‘Users’ is one to many. In the same way, relation between ‘Bicycle’ and ‘History’ is one to many as ‘Details’. Each station can have zero to many histories, and history has departed station and arrival station. Therefore, the relation between ‘History’ and ‘Station’ is many to many. The station is managed by department form ‘Employee’. Accordingly, the relation between ‘Employee’ and ‘Station’ is one to many as ‘Manage’. Likewise, broken bicycle is fixed by employee as many to one. Finlay, we made E-R diagram structure.

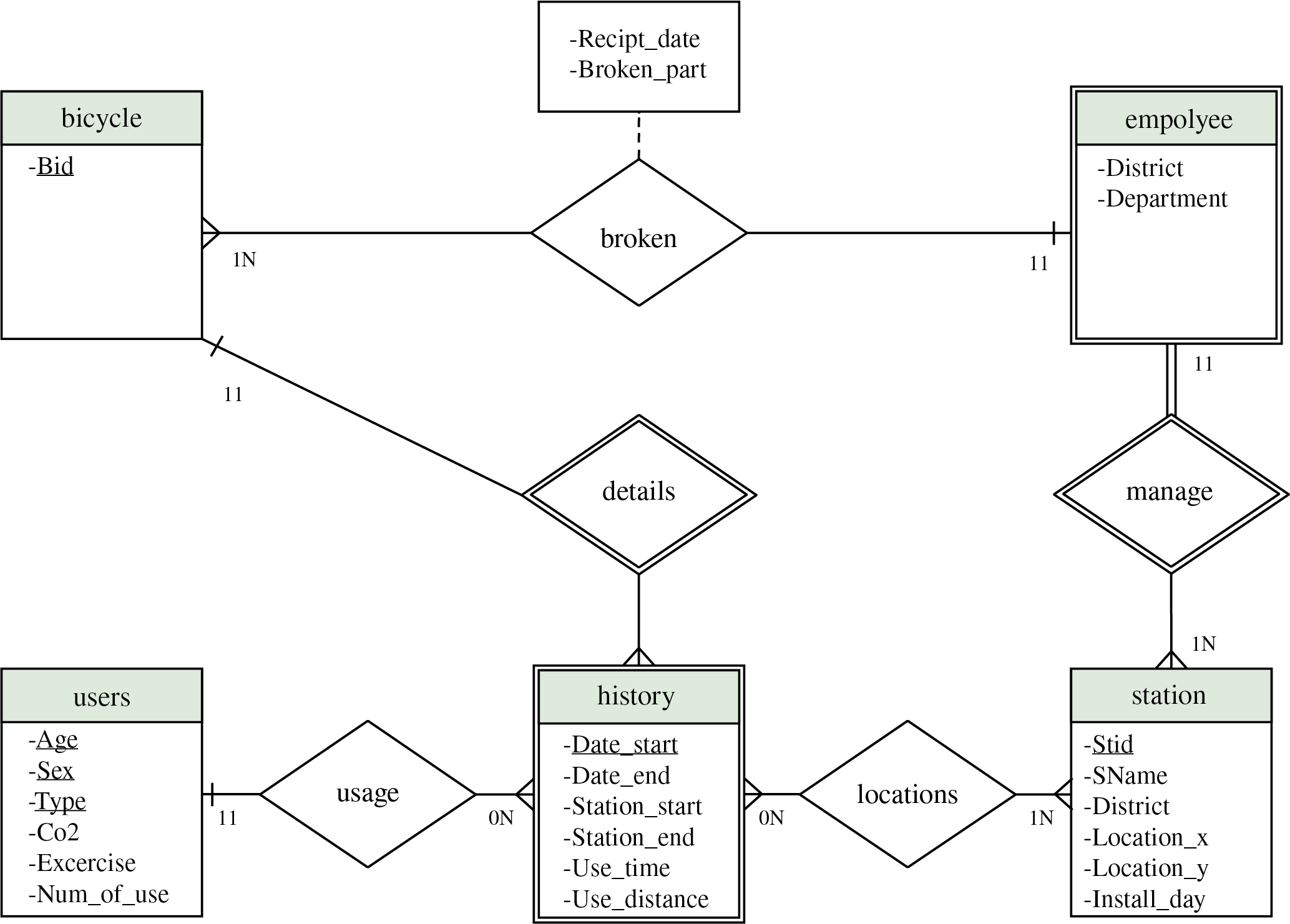


figure ER Diagram

1. **Implementation of the Database**

We implemented our table in the PostgreSQL database using python based on the preprocessed .csv files and ER diagram. In the process that implementing the database, we needed to modify some data using python and query, because we had to put a foreign key attribute in many entities.

Firstly, in the ‘Bicycle’ entity, we had to include the ‘Department’ attribute. However, we only had data about the department that manages a specific station and the bicycle ids stored in the station. The department and bicycle are not directly connected. However, we can find common characteristics that both data had, which is station id. So, to solve this problem, we decided to create the table we wanted by using the query statement.

Initially, we write the query statement, “SELECT Bid, Department FROM Employee, Locations WHERE Employee.Stid = Locations.Stid”. We could successfully select the bicycle id and department in the same station using this statement and make ‘Bicycle\_management.csv’ file using this new table. Then, in ‘Bicycle\_management.py’ file, we can create a ‘Bicycle\_management’ table containing the bicycle ids and department in charge of the specific bicycle. Also, in ‘Bicycle\_temp.py’ file, a ‘Bicycle\_temp’ table containing all bicycle ids was created. Lastly, the two tables were joined using the query statement, “SELECT Bid, Department FROM Bicycle natural left outer join Temp” and the final ‘Bicycle\_final.csv’ file was created. And, in the bicycle.py file, a ‘Bicycle’ table with all bicycle id and department attributes was created.

Secondly, the ‘Station’ entity also had to include the ‘Department’ attribute, which is a foreign key. Similarly, there was no department data in our station data, and we had to retrieve station id data from the employee table.

Initially, in ‘Station\_temp.py’, a ‘Station\_temp’ table containing all properties of ‘Station\_temp.csv’ was created. And, the query statement, “SELECT Station\_temp.Stid, Station\_temp.Sname, Station\_temp.Addr, Station\_temp.District, Station\_temp.Location\_x, Station\_temp.Location\_y, Station\_temp.Install\_day, Employee.Department INTO new\_Station FROM Station\_temp JOIN Employee ON Station\_temp.Stid = Employee. Stid”, created a table called ‘new\_station’ by joining employee's Department with the ‘Station\_temp’ table under the same Station id condition, and saved this table as ‘station.csv’ file. Using this csv file, in the ‘station.py’ file, the ‘Station’ table was finally created.

Thirdly, for user data, the primary key was designated with three attributes, ‘Age’, ‘Sex’, and ‘Types’. However, there was a problem that there was duplicate information among the three data. To resolve this problem, we used query statement again. First, in the ‘Users\_before.py’ file, we created the ‘Users\_before’ table from ‘Users.csv’ file. And we wrote a query statement, “SELECT Age, Sex, Types, SUM(Co2) AS Co2, SUM(Exercise) AS Exercise, SUM(Num\_of\_use) AS Num\_of\_use FROM Users\_before GROUP BY Age, Sex, Types ORDER BY Age, Sex”, to delete duplicate data, and saved the result as ‘Users\_final.csv’. Finally, we created the ‘Users’ table in the ‘users\_final.py’ file.

Except for these three entities, each table was created in each Python file and filled with data from the preprocessed .csv files. Through this process, we can successfully implement our database.

1. **SQL Queries**

The questions we wanted to answer using our database are as follows. We made two basic queries and other seven queries using query operations.

|  |  |
| --- | --- |
| **1. Which bicycle has more than 60 minutes and has 4 uses?** | |
| **SELECT** F\_bid  **FROM** Locations  **WHERE** (T\_time > 60) AND (Num\_of\_used = 4) | 텍스트이(가) 표시된 사진  자동 생성된 설명 |
| **2. What is the total CO2 reduction and exercise for women in their 30s with season tickets?** | |
| **SELECT** Co2, Exercise  **FROM** Users  **WHERE** Age = ‘30대’ AND Sex = ‘F’ AND Types = ‘정기’ |  |
| **3. Which departure and arrival station had the most number of usage in a month?** | |
| **SELECT** Station\_start, COUNT (Bid) AS Num\_of\_bicycle  **FROM** History  **GROUP BY** Station\_start  **HAVING** COUNT (Bid) >= ALL (**SELECT** COUNT(Bid) **FROM** History **GROUP BY** Station\_start)  **SELECT** Station\_end, COUNT (Bid) AS Num\_of\_bicycle  **FROM** History  **GROUP BY** Station\_end  **HAVING** COUNT(Bid) >= ALL (**SELECT** COUNT(Bid) **FROM** History **GROUP BY** Station\_end) | 텍스트이(가) 표시된 사진  자동 생성된 설명 |
| **4. What is the biggest number of usage of bicycles, and what is the total time?** | |
| **SELECT** F\_bid, Num\_of\_used,T\_time, T\_distance  **FROM** Locations  **WHERE** Num\_of\_used = (**SELECT** MAX(Num\_of\_used) **FROM** Locations) | 텍스트, 측정기, 장치이(가) 표시된 사진  자동 생성된 설명 |
| **5. A station with the most bicycles, and the number of bicycles** | |
| **SELECT** F\_sid, COUNT(F\_bid) AS Num\_of\_bicycle  **FROM** Locations  **GROUP BY** F\_sid  **HAVING** COUNT (F\_bid) >= ALL (**SELECT** COUNT (F\_bid) **FROM** Locations **GROUP BY** F\_sid) |  |
| **6. Which department is in charge of the station the most? And the number of stations?** | |
| **SELECT** Department, COUNT(Stid) AS Num\_of\_station  **FROM** Employee  **GROUP BY** Department  **HAVING** COUNT (Stid) >= ALL (**SELECT** COUNT (Stid) **FROM** Employee **GROUP BY** Department) |  |
| **7. Which district has the most station? And the number of stations?** | |
| **SELECT** District, COUNT(Stid) AS Num\_of\_station  **FROM** Employee  **GROUP BY** District  **HAVING** COUNT (Stid) >= ALL (**SELECT** COUNT(Stid) **FROM** Employee **GROUP BY** District) |  |
| **8. What was the travel distance and usage time of the broken bicycle?** | |
| **SELECT** Bid, AVG(T\_time) AS Moving\_time, AVG(T\_distance) AS Moving\_distance  **FROM** Locations  **JOIN** Broken on Locations.F\_bid = Broken.Bid  **GROUP BY** F\_bid | 텍스트, 장치, 측정기이(가) 표시된 사진  자동 생성된 설명 |
| **9. What was the fastest bicycle ID and its speed?** | |
| **SELECT** F\_bid, Avg\_speed  **FROM** (**SELECT** F\_bid, (T\_distance/T\_time)/60 AS Avg\_speed **FROM** Locations **WHERE** T\_time != 0)  **ORDER BY** Avg\_speed **DESC**  **LIMIT 1** |  |

Table SQL Queries and results

For the query 1 and 2, we can represent these questions in relation algebra.

For the query 3, we set the “month” as the most recent data, the month of June 2022. In order to find the station, the number of uses by station was counted, and then only data greater than or equal to the number of uses by all stations was selected.

For the query 8, we created this question because we thought there would be a correlation between the travel distance and usage time of the broken bicycle. After joining the ‘Locations’ and ‘Broken’ tables, only rows with a broken bicycle ID were selected. When we printed out the travel distance and usage time, an error occurred. The reason was that there was no guarantee that the bicycle would break down only once. Since the bicycle can fail several times, we grouped the data by bicycle ID and calculated average distance and time of the bicycle. Although the exact correlation was not identified, it was found that the broken bicycle belongs to the high travel distance and usage time.

For the query 9, to determine the speed of the fastest bicycle, the distance traveled by the bicycle must be divided by the travel time. At this time, if the travel time is 0, the speed cannot be obtained, so the data is excluded. After dividing the travel distance by the travel time, we divide the minutes by to change to seconds to indicate the speed at m/s.

1. **Advanced Features and Fullstack Development**

We added indexes to our database to verify that answers to frequently asked questions can be processed faster. The indexes we used were the B-Tree index (‘Num\_of\_used’ on ‘Locations’ table) and the Hash index (‘Station\_start’, ‘Station\_end’ on ‘History’ table). Queries using new indexed tables are numbered 1, 3, and 4.

To test the performance improvement, we ran queries with and without indexes 10 times each and computed average running time. The results are shown in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Query** | **Avg Time Without Index [ms]** | **Avg Time With Index [ms]** |
| B-Tree | 1 | 140,4 | 120,6 |
| B-Tree | 4 | 68,3 | 70,5 |
| Hash | 3 | 65,7 | 70,1 |

Table 2 Comparison running time using Indexing

The results were slightly different from what we expected. For query 1, we can see that the time is reduced about 20ms, which can be inferred because there were many outputs using indexing. However, even the addition of indexes, queries 3 and 4 increased response time. Probably, if we had used all existing data instead of selecting random data, indexing would have had a distinct speed-up effect due to the increased amount of data.

Regarding Transaction, we used them a lot in the data insertion process. We had to be careful not to insert meaningless data. In addition, in order to remove the possibility of duplication, if the table already exists, the table was always dropped and data was newly inserted.

1. **Conclusion and Limitation**

We could design a database for the public bicycle system and implement queries in SQL to answer the questions we made. We collected public data from the Seoul public data portal and preprocessed the dataset into the form we will use. With this data, we made database schema and ER Diagram. We had three strong entities, two weak entities and five relationship sets in total. Based on this ER diagram and preprocessed data, we implemented our database using python and query. Then, we could answer our questions using SQL queries based on the database.

However, we had some limitations during the project.

Firstly, we randomly extracted data from ‘History’ by 1/100, resulting in data loss. As a result, it was not possible to track the exact path of the bicycle. Because, we used an algorithm to check the location of a bicycle, the starting point, and to change the arrival point of the bicycle to the final arrival point when using the bicycle again. The problem is that if the data is lost in the middle, the bicycle actually moved, but it is judged that it does not move on record. Query 3 shows that the station has a large number of bicycles at 152, which has occurred from the aforementioned limitations.

Secondly, user information is not specific. In the provided user information, users were classified into ‘Sex’, ‘Ages’, and ‘Types’. For this reason, we used three attributes as primary key. However, since bicycles are actually used by individuals, they are not one-to-one. So, we couldn't track one person's use.

Lastly, the effect of indexing was not great. We expected that the database with index is much faster to retrieve and find specific rows than that without index. However, there was an effect of speed improvement for one query, but not much effect for the other two queries. The reason is assumed to be due to the difference in the number of data. When there is a large number of data, the speed is faster with indexing, but it does not have a significant effect when the number of data is small.

Although we had some limitations, we could understand deeply what we learned in theory and we were able to handle data from the point of view of users and companies by constructing our database by ourselves. Also, we realized that it is important to make queries based on the users’ needs and build a database with it. Based on our shortcomings and lessons learned, we will be able to do a more developed project.

***[References]***

*Lin, J. R., & Yang, T. H. (2011). Strategic design of public bicycle sharing systems with service level constraints. Transportation research part E: logistics and transportation review, 47(2), 284-294*