Bay Area Bike Share Database Design, Normalization, and Performance Optimization

Pradhiksha Suresh Database Foundations Project Report

Dataset Description

The **Bay Area Bike Share dataset** contains operational data from a public bike-sharing system across the San Francisco Bay Area. It includes information on **stations** (locations and capacities), **trip records** (start/end stations, duration, user type), **station status** (bike and dock availability over time), and **weather data** (daily weather conditions by ZIP code). This multi-table dataset captures both transactional and contextual data, making it well-suited for relational database design, normalization, and performance analysis of real-world transportation systems.

Abstract

This project focuses on the design, implementation, and optimization of a relational database system using the Bay Area Bike Share dataset. The objective was to transform raw operational data into a structured, efficient, and scalable database that supports analytical queries and maintains data integrity. The project involved requirement analysis, ER modeling, functional dependency analysis, and normalization to Boyce–Codd Normal Form (BCNF), followed by SQL-based implementation with appropriate constraints. In the performance optimization phase, indexed and non-indexed schemas were compared using complex JOIN queries to evaluate execution efficiency, demonstrating the impact of indexing on query performance. The project highlights best practices in database modeling, normalization, and optimization for large-scale, real-world datasets.

STEP 1: DATABASE DESIGN

Requirement Analysis

Q1. Why has this data been gathered?
This data has been gathered to understand the usage patterns, customer behavior, and operational
efficiency of the bike-sharing service.
Q2. What insightful information can this data provide us that can be used to improve the business?
The dataset provides key insights such as demand patterns, popular routes, customer demographics, seasonal trends, user experience and revenue optimization. These insights are essential for improving business operations and optimizing services
Q3. Why are we studying this data?
By studying this data, we hope to gain insight into the trends and correlations between the usage rate of the bikes and other factors in the data such as temperature, that would be useful to make business decisions to further grow the business.
Q4. What are some of the data points that we can gather to help us optimize the business?
a. Assess the bike availability at each station during peak hours to determine if additional bikes are needed at specific stations.
b. Evaluate bike usage frequency to prompt alerts for necessary repairs and maintenance.
c. Identify frequent customers who have not yet subscribed, allowing us to create tailored special offers for them.
Q5. Can we find any solution by studying this data?
Yes, solutions could be found by taking averages of the docks available and bikes available to gain a better understanding of the demand by each location. Using this information business decisions such as building a new location or taking down a location could be justified using data from this dataset. Additionally, solutions such as providing discounts on hotter days to bring in more people could be justified by using the data to show that on higher mean temperature days the usage of bikes is less.

Data Understanding

Step 3 from Project 1

1. Schema Design

a. Find entities, their attributes, their primary keys from your dataset.

This dataset contains 4 entities namely – Station, Status, Trip and Weather

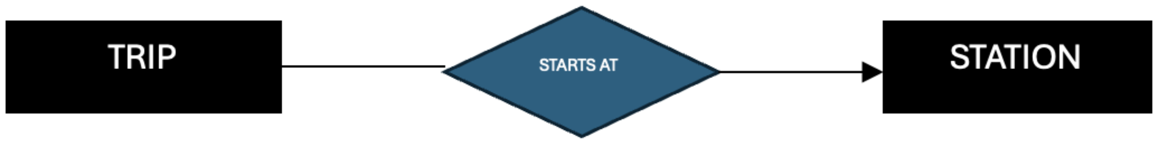
Entity	Attributes	Primary Key
Station	station_id	station_id
	Name	
	Lat	
	Long	
	Dock_count	
	City	
	Installation_date	
	Coordinates	
	Installation_year	
Trip	trip_id	trip_id
	Duration	
	Start_date	
	Start_station_name	
	Start_station_id	
	End_date	
	End_station_name	
	End_station_id	
	Bike_id	
	Subscription_type	
	Zip_code	
	Duration_in_hours	
	Start_day	
Weather	date (Primary Key)	date
	max_temperature_f	
	mean_temperature_f	
	min_temperature_f	
	Max_dew_point_f	
	Mean_dew_point_f	
	Min_dew_point_f	
	Max_humidity	
	Mean_humidity	
	Min_humidity	
	max_sea_level_pressure_inches	
	mean_sea_level_pressure_inches	
	min_sea_level_pressure_inches	
	Max_visibility_miles	
	Mean_visibility_miles	
	Min_visibility_miles	
	Max_wind_speed_mph	
	Mean_wind_speed_mph	
	Max_gust_speed_mph	

	Precipitation_inches	
	Cloud_cover	
	events	
	Win_dir_degrees	
	Zip_code	
	Temperature_range	
Status	Station_id	Time,station_id
	Bikes_available	
	Docks_available	
	time	

b. Find the relationships between entities you have.

1. Station and Trip:

The Station entity likely serves as the starting and ending points for bike trips recorded in the Trip entity. This relationship is established through the start_station_id and end_station_id attributes in the Trip entity, which are foreign keys referencing the station_id attribute in the Station entity.

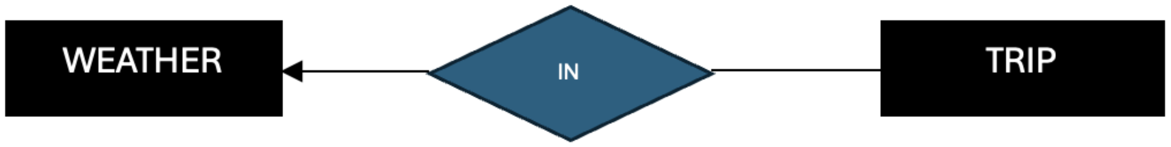


Relationship: Many to One Relationship

This relationship implies that one station may be the starting or ending point for multiple bike trips recorded in the Trip entity, but each trip is associated with only one starting and one ending station.

2. Trip and Weather:

The Trip entity has a relationship with the Weather entity based on the area of the trip. This relationship would be based on the zip_code, start_date, end_date attributes of the trip, with the date and zip_code attribute in the Weather entity serving as a point of connection.

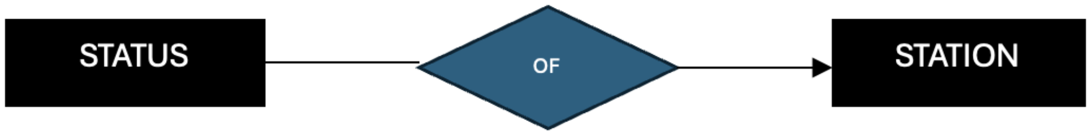


Relationship: One to Many Relationship

This relationship implies that multiple trip records in the Trip entity corresponds to one weather record in the Weather entity.

3. Station and Status:

The Station entity has a relationship with the Status entity based on the information about the availability of bikes and docks at each station. This relationship could be established through the station_id attribute, which serves as a foreign key in the Status entity.



Relationship: Many to One Relationship

This relationship implies that each station in the Station entity can have multiple corresponding status records in the Status entity.

c. Describe all the constraints you believe should be applied to the relationships in your schema.

Key Constraints:

Entity	Keys	Primary key	Foreign key
Station	Station_id Station_name	Station_id	
Trip	Trip_id	Trip_id	Start_station_id Start_station_name End_station_id End_station_name
Weather	date	date	Zip_code
Status	time	Time,station_id	Station_id

These single value constraints play a crucial role in maintaining the consistency of data within a database by ensuring that individual values meet specified criteria or conditions.

Single-Value Constraints:

Station:

- Each station in the Station entity has a unique station_id(PK)
- Each station in the Station entity can have at most one installation date and installation_year
- Each station in the Station entity can have at most one city
- Each station in the Station entity has a unique combination of latitude and longitude

Trip:

- Each trip in the Trip entity has a unique trip_id(PK)

Weather:

- Each weather record in the Weather entity has a unique date
- Each weather record in the Weather entity has a unique combination of date and zip code.

Status:

- Each status record in the Status entity has a unique combination of time and station_id(PK).

These single value constraints ensure that certain attributes or combinations of attributes have unique values or relationships within the dataset, maintaining data integrity and consistency.

Referential Constraints

- When a station is referenced as a start_station_id or end_station_id in the Trip entity, it must exist in the Station entity. This ensures that each trip starts and ends at a valid station.
- If trip information is recorded for a particular date and zip code in the Trip entity, there should be corresponding weather records in the Weather entity for the same dare and zip code. This ensures that trip data is associated with valid weather data.
- When a station's status is recorded in the Status entity, it must correspond to a valid station in the Stations entity. This ensures that status information is associated with existing station data.
- If any of the above referenced entity is deleted, then all entities that reference it should also be deleted. For example, if a station is deleted from the Stations entity, related status records in the Status entity that reference the deleted station should also be deleted.

These referential integrity constraints ensure that values referenced by attributes in one entity actually exist in the related entity, maintaining the integrity and consistency of the database.

Domain Constraints

1. Stations Entity:

- Latitude (lat) and Longitude (long) must be within valid ranges for geographic coordinates
- Dock count (dock_count) must be a non-negative integer, indicating the number of docks available at the station
- Installation date (installation_date) must be a valid date value

2. Trips Entity:

- Duration (duration_sec) must be a non-negative integer, representing the duration of the trip in seconds
- Start and end dates (start_time, end_time) must be valid date and time values
- Zip code (zip_code) must be a valid ZIP code within the SF Bay Area region

3. Weather Entity:

- Temperature values (max_temperature_f, mean_temperature_f, min_temperature_f) must fall within reasonable ranges for temperatures in Fahrenheit
- Dew point values (max_dew_point_f, mean_dew_point_f, min_dew_point_f) must fall within reasonable ranges for dew points in Fahrenheit
- Humidity values (max_humidity, mean_humidity, min_humidity) must be percentages between 0 and 100
- Wind speed values (max_wind_speed_mph, mean_wind_speed_mph) must be non-negative integers representing speeds in miles per hour

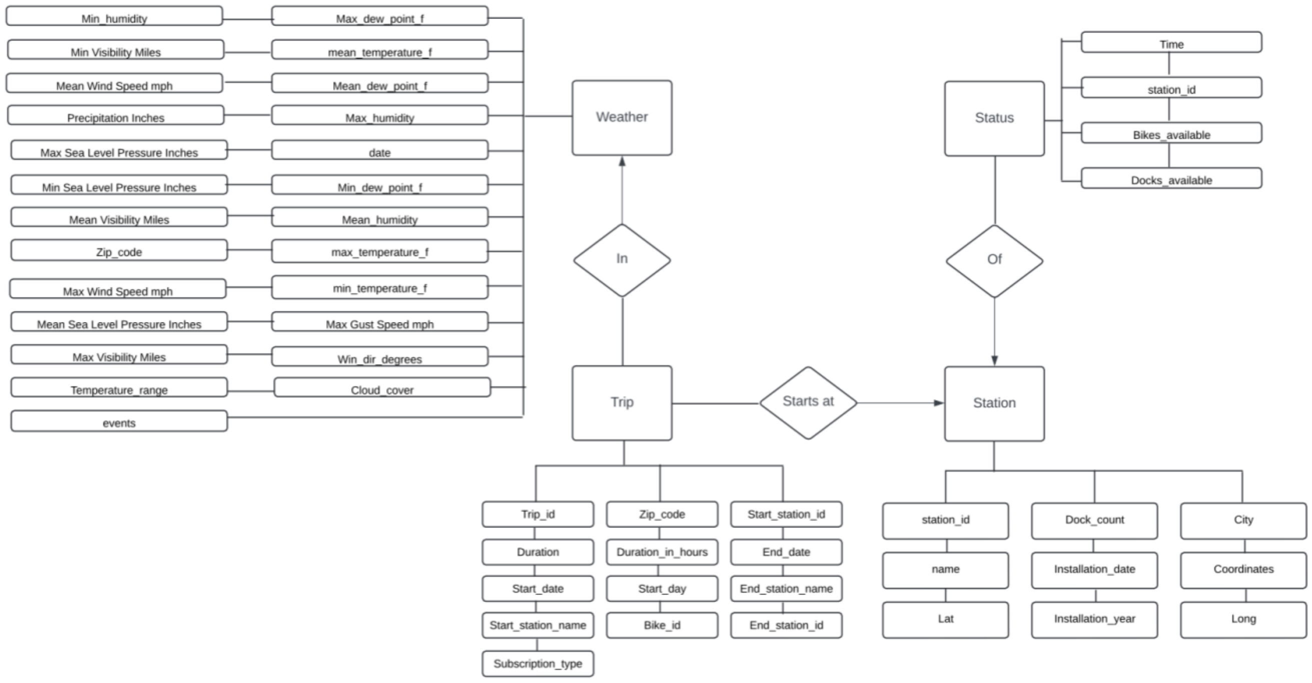
4. Status Entity:

- Bikes available (bikes_available) and docks available (docks_available) must be non-negative integers, indicating the number of bikes and docks available at each station
- Time (time) must be a valid date and time value

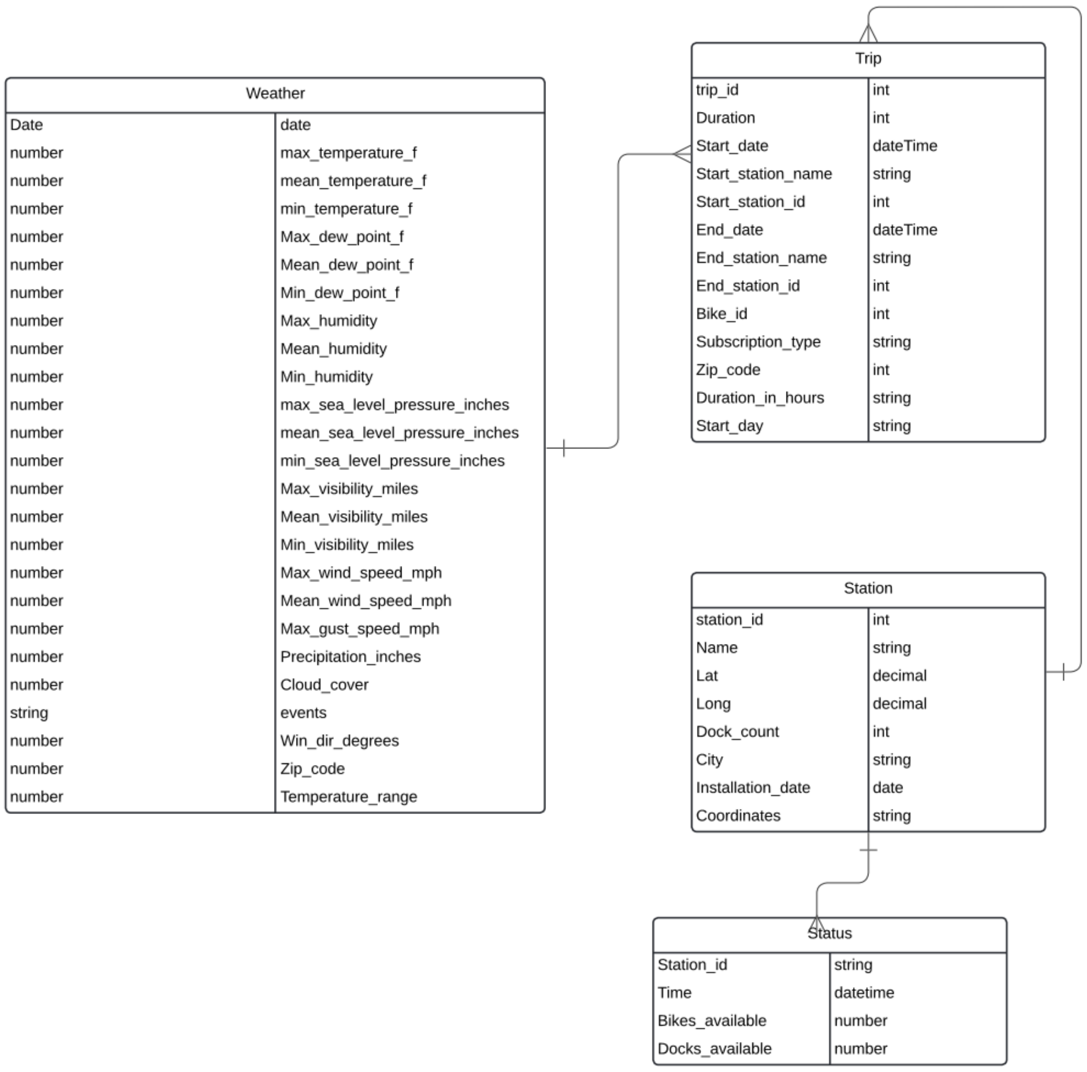
These domain constraints ensure that the values stored in the database adhere to reasonable and acceptable ranges or sets, maintaining data integrity and consistency.

d. Draw an Entity-Relationship (ER) diagram of your case

The following diagram represents the Entity-Relationship diagram for our SF Bay Area Bikeshare dataset.



e. Translate your ER diagram into relations.



Weather	
Date	date
number	max_temperature_f
number	mean_temperature_f
number	min_temperature_f
number	Max_dew_point_f
number	Mean_dew_point_f
number	Min_dew_point_f
number	Max_humidity
number	Mean_humidity
number	Min_humidity
number	max_sea_level_pressure_inches
number	mean_sea_level_pressure_inches
number	min_sea_level_pressure_inches
number	Max_visibility_miles
number	Mean_visibility_miles
number	Min_visibility_miles
number	Max_wind_speed_mph
number	Mean_wind_speed_mph
number	Max_gust_speed_mph
number	Precipitation_inches
number	Cloud_cover
string	events
number	Win_dir_degrees
number	Zip_code
number	Temperature_range

Trip	
trip_id	int
Duration	int
Start_date	dateTime
Start_station_name	string
Start_station_id	int
End_date	dateTime
End_station_name	string
End_station_id	int
Bike_id	int
Subscription_type	string
Zip_code	int
Duration_in_hours	string
Start_day	string

Station	
station_id	int
Name	string
Lat	decimal
Long	decimal
Dock_count	int
City	string
Installation_date	date
Coordinates	string

Status	
Station_id	string
Time	datetime
Bikes_available	number
Docks_available	number

2. Schema Normalization

a. Find all the functional dependencies you can from your schema and list them

1. Station Entity:

- {station_id} -> {name, lat, long, dock_count, city, installation_date, coordinates, installation_year}
- {installation_date} -> {installation_year}
- {lat, long} -> {city}
- {lat, long} -> {coordinates}

2. Trip Entity:

- {trip_id} -> {duration, start_date, start_station_id, start_station_name, end_date, end_station_id, end_station_name, bike_id, subscription_type, zip_code, duration_in_hours, start_day}
- {start_station_id} - > {start_station_name}
- {end_station_id} - > {end_station_name}
- {start_time, end_time} -> {duration, duration_in_hours}
- {start_date} - > {start_day}

3. Weather Entity:

- {date} -> {max_temperature_f, mean_temperature_f, min_temperature_f, max_dew_point_f, mean_dew_point_f, min_dew_point_f, max_humidity, mean_humidity, min_humidity, max_sea_level_pressure_inches, mean_sea_level_pressure_inches, min_sea_level_pressure_inches, max_visibility_miles, mean_visibility_miles, min_visibility_miles, max_wind_speed_mph, mean_wind_speed_mph, max_gust_speed_mph, precipitation_inches, cloud_cover, events, win_dir_degrees, zip_code, temperature_range}
- {max_temperature_f, min_temperature_f} - > {temperature_range}

4. Status Entity:

- {station_id, time} - > { bikes_available, docks_available}

b. Show that the attribute set you have chosen as the primary key for each relation does work as a key (prove)

To prove that the attribute set chosen as the primary key for each relation in our dataset functions as a key, we need to show two things:

1. The chosen attribute set functionally determines all other attributes of the relation.
2. No proper subset of the chosen attribute set functionally determines all other attributes of the relation.

Let's prove these 2 things for all the relations:

1. Stations Table:

- Chosen attribute set as the primary key: {station_id}
- Functional determination: The station_id uniquely identifies each station, and all other attributes (name, lat, long, dock_count, city, installation_date, installation_year) are fully dependent on station_id.

2. Trips Table:

- Chosen attribute set as the primary key: {trip_id}
- Functional determination: The trip_id uniquely identifies each trip, and all other attributes (duration_sec, start_time, start_station_id, end_time, end_station_id, bike_id, subscription_type, zip_code, duration_in_hours, start_day) are fully dependent on trip_id.

3. Weather Table:
    - Chosen attribute set as the primary key: {date}
    - Functional determination: The date uniquely identifies each weather observation, and all other attributes (max_temperature_f, mean_temperature_f, min_temperature_f, max_dew_point_f, mean_dew_point_f, min_dew_point_f, max_humidity, mean_humidity, min_humidity, max_sea_level_pressure_inches, mean_sea_level_pressure_inches, min_sea_level_pressure_inches, max_visibility_miles, mean_visibility_miles, min_visibility_miles, max_wind_speed_mph, mean_wind_speed_mph, max_gust_speed_mph, precipitation_inches, cloud_cover, events, win_dir_degrees, zip_code, temperature_range) are fully dependent on date.

4. Status Table:

- Chosen attribute set as the primary key: {station_id, time}
- Functional determination: The combination of station_id and time uniquely identifies each status record, and all other attributes (bikes_available, docks_available) are fully dependent on station_id and time.

Therefore, the chosen attribute sets for the primary keys in each relation fulfill the conditions of a key, as they functionally determine all other attributes of the relation.

c. Check if the keys you have chosen for your relations are minimal (prove)

1. Stations Table:

- Chosen attribute set as the primary key: {station_id}
- Minimality: Since station_id is the only attribute that uniquely identifies each station and no proper subset of {station_id} can functionally determine all other attributes, it is minimal.

2. Trip Table:
- Chosen attribute set as the primary key: {trip_id}
- Minimality: Since trip_id is the only attribute that uniquely identifies each trip and no proper subset of {trip_id} can functionally determine all other attributes, it is minimal.

3. Weather Table:
    - Chosen attribute set as the primary key: {date}
    - Minimality: Since date is the only attribute that uniquely identifies each weather observation and no proper subset of {date} can functionally determine all other attributes, it is minimal.

4. Status Table:
- Chosen attribute set as the primary key: {station_id, time}
- Minimality: Since the combination of station_id and time is the only set that uniquely identifies each status record and no proper subset of {station_id, time} can functionally determine all other attributes, it is minimal.

d. Check if your schema is in BCNF (Boyce-Codd Normal Form) (prove)

Station Entity:

Attributes:

A = { station_id, name, lat,long, dock_count, city, installation_date, coordinates, installation_year}

Station_id is the key

Functional Dependencies:

F = {

{station_id} -> {name, lat, long, dock_count, city, installation_date, coordinates, installation_year}, {installation_date} -> {installation_year} , {lat, long} -> {city}, {lat, long} -> {coordinates}
}

1. {station_id} -> {name, lat, long, dock_count, city, installation_date, coordinates, installation_year}

This dependency indicates that all attributes except station_id are functionally dependent on station_id. Since station_id is the key, this dependency does not violate BCNF.

2. {installation_date} -> {installation_year}

This dependency indicates that installation_year is functionally dependent on installation_date. Since installation_date is not a superkey on its own, this dependency violates BCNF.

3. {lat, long} -> {city}

This dependency indicates that city is functionally dependent on the combination of lat and long. Since lat and long together are not a superkey, this dependency violates BCNF.

4. {lat, long} -> {coordinates}

This dependency indicates that coordinates are functionally dependent on the combination of lat and long. Since lat and long together are not a superkey, this dependency violates BCNF.

Trip Entity

Attributes:

A = {trip_id, duration, start_date, start_station_id, start_station_name, end_date, end_station_id, end_station_name, bike_id, subscription_type, zip_code, duration_in_hours, start_day}

Trip_id is the key

Functional Dependencies:

F = {

{trip_id} -> {duration, start_date, start_station_id, start_station_name, end_date, end_station_id, end_station_name, bike_id, subscription_type, zip_code, duration_in_hours, start_day}
{start_station_id} - > {start_station_name}
{end_station_id} - > {end_station_name}
{start_time, end_time} -> {duration}

{duration} - > {duration_in_hours}
{start_date} - > {start_day}
}

1. {trip_id} -> {duration, start_date, start_station_id, start_station_name, end_date, end_station_id, end_station_name, bike_id, subscription_type, zip_code, duration_in_hours, start_day}

This dependency indicates that all attributes except trip_id are functionally dependent on trip_id. Since trip_id is the key, this dependency does not violate BCNF.

2. {start_station_id} -> {start_station_name}

This dependency indicates that start_station_name is functionally dependent on start_station_id. Since start_station_id is not a superkey on its own, this dependency violates BCNF.

3. {end_station_id} -> {end_station_name}

This dependency indicates that end_station_name is functionally dependent on end_station_id. Since end_station_id is not a superkey on its own, this dependency violates BCNF.

4. {start_time, end_time} -> {duration}

This dependency indicates that duration is functionally dependent on the combination of start_time and end_time. Since start_time and end_time together are not a superkey, this dependency violates BCNF.

5. {duration} -> {duration_in_hours}

This dependency indicates that duration_in_hours is functionally dependent on duration. Since duration is not a superkey on its own, this dependency violates BCNF.

6. {start_date} -> {start_day}

This dependency indicates that start_day is functionally dependent on start_date. Since start_date is not a superkey on its own, this dependency violates BCNF.

Weather Entity

Attributes:

A = { date, max_temperature_f, mean_temperature_f, min_temperature_f, max_dew_point_f, mean_dew_point_f, min_dew_point_f, max_humidity, mean_humidity, min_humidity, max_sea_level_pressure_inches, mean_sea_level_pressure_inches, min_sea_level_pressure_inches, max_visibility_miles, mean_visibility_miles, min_visibility_miles, max_wind_speed_mph, mean_wind_speed_mph, max_gust_speed_mph, precipitation_inches, cloud_cover, events, win_dir_degrees, zip_code, temperature_range}

Date is key

Functional Dependencies:

F = {

{date} -> {max_temperature_f, mean_temperature_f, min_temperature_f, max_dew_point_f, mean_dew_point_f, min_dew_point_f, max_humidity, mean_humidity, min_humidity, max_sea_level_pressure_inches, mean_sea_level_pressure_inches, min_sea_level_pressure_inches, max_visibility_miles, mean_visibility_miles, min_visibility_miles, max_wind_speed_mph, mean_wind_speed_mph, max_gust_speed_mph, precipitation_inches, cloud_cover, events, win_dir_degrees, zip_code, temperature_range},
{max_temperature_f, min_temperature_f} - > {temperature_range}
}

1. {date} -> {max_temperature_f, mean_temperature_f, min_temperature_f, max_dew_point_f, mean_dew_point_f, min_dew_point_f, max_humidity, mean_humidity, min_humidity, max_sea_level_pressure_inches, mean_sea_level_pressure_inches, min_sea_level_pressure_inches, max_visibility_miles, mean_visibility_miles, min_visibility_miles, max_wind_speed_mph, mean_wind_speed_mph, max_gust_speed_mph, precipitation_inches, cloud_cover, events, win_dir_degrees, zip_code, temperature_range}

This dependency indicates that all attributes except date are functionally dependent on date. Since date is the key, this dependency does not violate BCNF.

2. {max_temperature_f, min_temperature_f} -> {temperature_range}

This dependency indicates that temperature_range is functionally dependent on the combination of max_temperature_f and min_temperature_f. Since max_temperature_f and min_temperature_f together are not a superkey, this dependency violates BCNF.

Status Entity

Attributes :

A = {station_id, time, bike_available, docks_available}

{station_id, time} is key

Functional Dependencies:

F = {

{station_id, time} - > { bikes_available, docks_available}

}

1. {station_id, time} - > { bikes_available, docks_available}

Since the given functional dependency {station_id, time} -> {bikes_available, docks_available} already indicates that both bikes_available and docks_available are functionally dependent on the combination of station_id and time, and the key {station_id, time} is a superkey on its own, there are no violations of BCNF in this case.

f. If your schema violates BCNF, bring it to BCNF by decomposing it

Since our schema violates BCNF in several instances, we need to decompose it

Station

1. {installation_date} -> {installation_year}

Violates BCNF

To decompose this dependency, we can create a new relation with attributes {installation_date, installation_year}, making installation_date the key of this new relation.

2. {lat, long} -> {city}

Violates BCNF

To decompose this dependency, we can create a new relation with attributes {lat, long, city}, making {lat, long} the key of this new relation.

3. {lat, long} -> {coordinates}

Violates BCNF

To decompose this dependency, we can create a new relation with attributes {lat, long, coordinates}, making {lat, long} the key of this new relation.

Trip:

1. {start_station_id} -> {start_station_name}

Violates BCNF

To decompose this dependency, we can create a new relation with attributes {start_station_id, start_station_name}, making start_station_id the key of this new relation.

2. {end_station_id} -> {end_station_name}

Violates BCNF

To decompose this dependency, we can create a new relation with attributes {end_station_id, end_station_name}, making end_station_id the key of this new relation

3. {start_time, end_time} -> {duration}

Violates BCNF

To decompose this dependency, we can create a new relation with attributes {start_time, end_time, duration}, making {start_time, end_time} the key of this new relation

5. {duration} -> {duration_in_hours}

Violates BCNF

To decompose this dependency, we can create a new relation with attributes {duration, duration_in_hours}, making duration the key of this new relation.

6. {start_date} -> {start_day}

Violates BCNF

To decompose this dependency, we can create a new relation with attributes {start_date, start_day}, making start_date the key of this new relation

Weather:

1. {max_temperature_f, min_temperature_f} -> {temperature_range}

Violates BCNF

To decompose this dependency, we can create a new relation with attributes {max_temperature_f, min_temperature_f, temperature_range}, making {max_temperature_f, min_temperature_f} the key of this new relation.

Status:

none

After decomposing the violating dependencies, we have the following relations:

1. Stations(station_id, name, lat, long, dock_count, city,installation_date)

Key: station_id

2. Installation_dates(installation_date, installation_year)

Key: installation_date

3. Coordinates(lat, long, coordinates)

Key: {lat, long}

4. Trips(trip_id, start_date, start_station_id, end_date, end_station_id, bike_id, subscription_type, zip_code)

Key: trip_id

5. Start_stations(start_station_id, start_station_name)

Key: start_station_id

6. End_stations(end_station_id, end_station_name)

Key: end_station_id

7. Weather_data(date, max_temperature_f, mean_temperature_f, min_temperature_f, max_dew_point_f, mean_dew_point_f, min_dew_point_f, max_humidity, mean_humidity, min_humidity, max_sea_level_pressure_inches, mean_sea_level_pressure_inches, min_sea_level_pressure_inches, max_visibility_miles, mean_visibility_miles, min_visibility_miles, max_wind_speed_mph, mean_wind_speed_mph, max_gust_speed_mph, precipitation_inches, cloud_cover, events, win_dir_degrees, zip_code)

Key: date

8. Temperature_ranges(max_temperature_f, min_temperature_f, temperature_range)

Key: {max_temperature_f, min_temperature_f}

9. Status(station_id, time, bike_available, docks_available)

Key: {station_id, time}

Now, each relation satisfies BCNF, and the dependencies are properly represented without any violations.

f. Update your ER diagram with the latest schema

Appendix

Updated Relations Diagram

Errors while importing

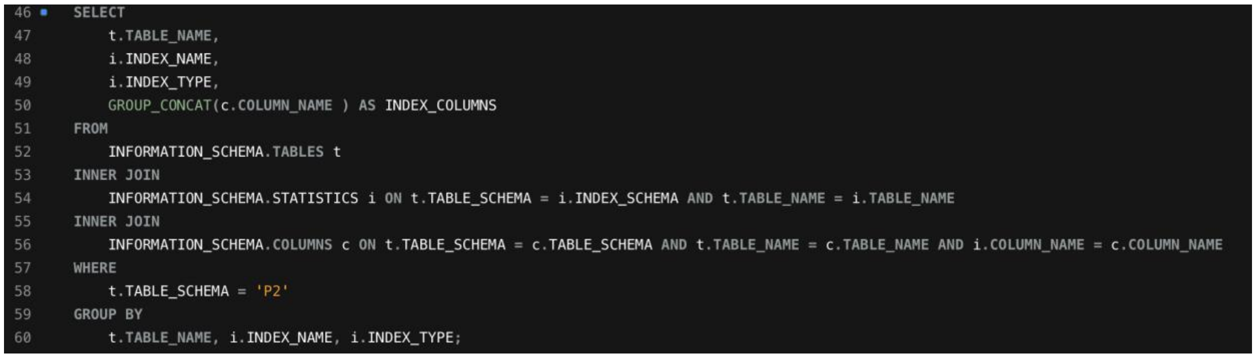
- Status table (originally 7m rows, reduced it to 1,048,575 rows)
- Changed the datatype of date columns from text to date
  Queries to change datatype:
  ```
  UPDATE status SET time = STR_TO_DATE(time, '%m/%d/%y %H:%i');
  ALTER TABLE weather MODIFY `date` date;
  ALTER TABLE status MODIFY `time` datetime;
  ALTER TABLE installation_dates MODIFY installation_date date;
  alter table stations modify installation_date date;
  UPDATE trips SET start_date = STR_TO_DATE(start_date, '%m/%d/%y %H:%i');
  UPDATE trips SET end_date = STR_TO_DATE(end_date, '%m/%d/%y %H:%i');
  ALTER TABLE trips MODIFY start_date datetime;
  ALTER TABLE trips MODIFY end_date datetime;
  ```
- Weather table had null values that were not detected when importing in Mysql, so did preprocessing in Excel to fill in the blank cellas with "NULL"
- Assigning primary keys for all 9 tables
  Queries:
  ```
  ALTER TABLE end_stations ADD PRIMARY KEY(end_station_id);
  ALTER TABLE installation_dates ADD PRIMARY KEY(installation_date);
  ALTER TABLE start_stations ADD PRIMARY KEY(start_station_id);
  ALTER TABLE stations ADD PRIMARY KEY(id);
  ALTER TABLE trips ADD PRIMARY KEY(id);
  ALTER TABLE weather ADD PRIMARY KEY(`date`);
  ```

ALTER TABLE status ADD PRIMARY KEY(station_id,`time`);
ALTER TABLE temperature_ranges ADD PRIMARY KEY(max_temperature_f,min_temperature_f);
ALTER TABLE coordinates ADD PRIMARY KEY(lat,`long`);

Part 1: Indexing and Query Timing

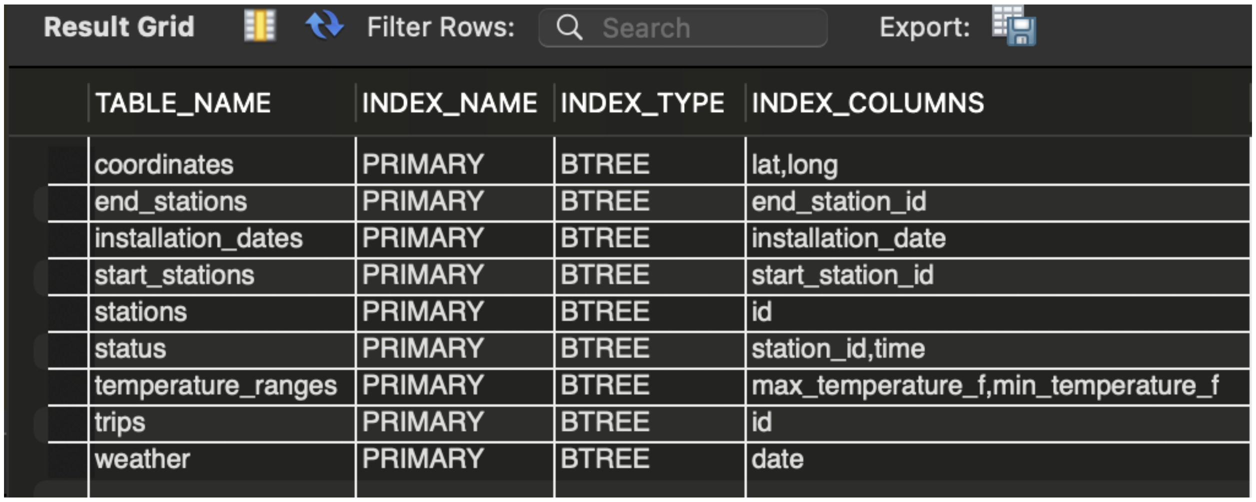
1.1. List all the current indexes in your database and the columns they are associated with along with the index type

We have created a database with the new schema as per 'Project 2' and we have named the database as 'P2'. To view all the current implicit indexes and the columns they are associated with along with their index type in P2, we executed the following query.



```
SELECT
    t.TABLE_NAME,
    i.INDEX_NAME,
    i.INDEX_TYPE,
    GROUP_CONCAT(c.COLUMN_NAME ) AS INDEX_COLUMNS
FROM
    INFORMATION_SCHEMA.TABLES t
INNER JOIN
    INFORMATION_SCHEMA.STATISTICS i ON t.TABLE_SCHEMA = i.INDEX_SCHEMA AND t.TABLE_NAME = i.TABLE_NAME
INNER JOIN
    INFORMATION_SCHEMA.COLUMNS c ON t.TABLE_SCHEMA = c.TABLE_SCHEMA AND t.TABLE_NAME = c.TABLE_NAME AND i.COLUMN_NAME = c.COLUMN_NAME
WHERE
    t.TABLE_SCHEMA = 'P2'
GROUP BY
    t.TABLE_NAME, i.INDEX_NAME, i.INDEX_TYPE;
```

Here is the output containing a table that displays the 'Table_name', 'Index_name', 'Index_type' and 'Index_columns:



Result Grid Filter Rows: Search Export:

TABLE_NAME	INDEX_NAME	INDEX_TYPE	INDEX_COLUMNS
coordinates	PRIMARY	BTREE	lat,long
end_stations	PRIMARY	BTREE	end_station_id
installation_dates	PRIMARY	BTREE	installation_date
start_stations	PRIMARY	BTREE	start_station_id
stations	PRIMARY	BTREE	id
status	PRIMARY	BTREE	station_id,time
temperature_ranges	PRIMARY	BTREE	max_temperature_f,min_temperature_f
trips	PRIMARY	BTREE	id
weather	PRIMARY	BTREE	date

1.2. Explain what is in common between these columns (why these columns are indexed automatically by the database management system)

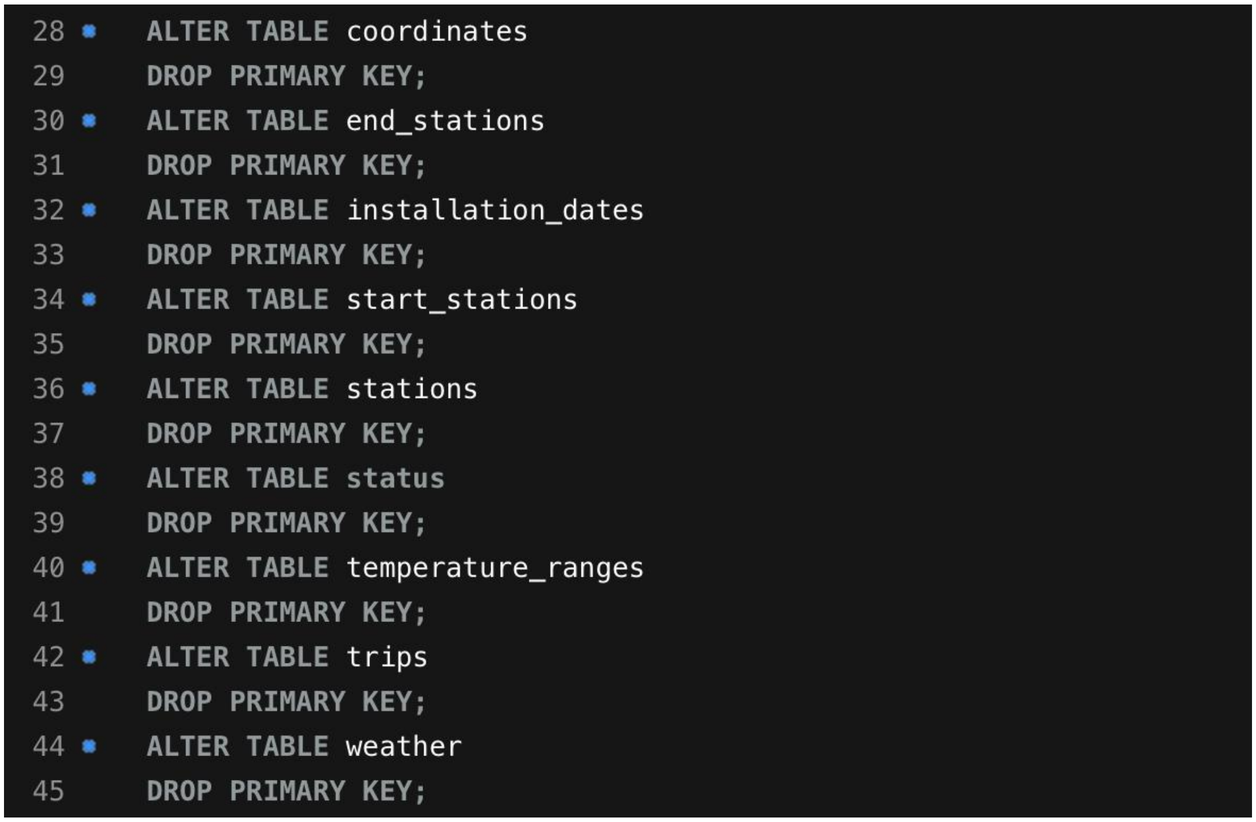
The columns that are automatically indexed by the database management system (DBMS) in our database are all primary keys for the 9 tables. These primary keys are indexed automatically by MySQL for efficient data retrieval and to enforce data integrity.

When a column is designated as a primary key in a table, it serves as a unique identifier for each record in that table. Indexing these primary key columns allows the DBMS to quickly locate specific rows based on their unique identifiers. This indexing mechanism significantly speeds up data retrieval operations.

In summary, MySQL automatically indexes primary key columns in our database to optimize query performance by facilitating fast data access and to enforce the uniqueness constraint.

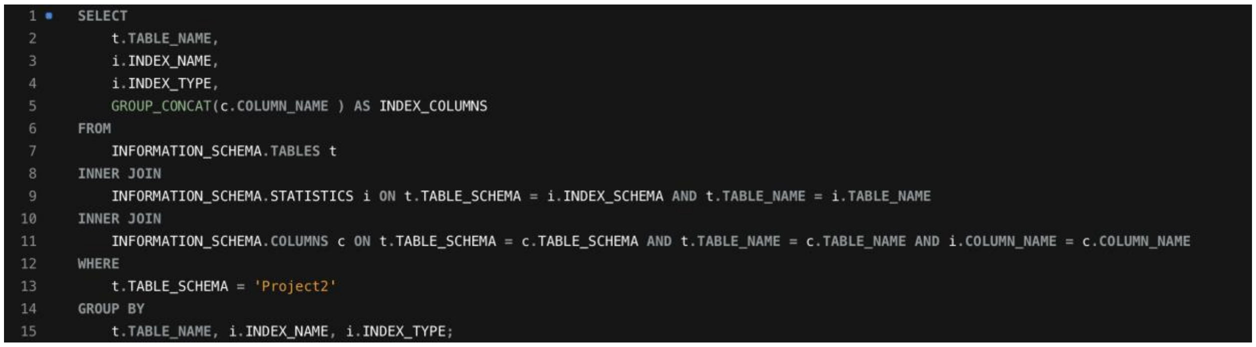
1.3. Make a copy of your database and delete all the indexes there– now you have two databases: database A with indexes and database B without any indexes.

After duplicating our database into a new database named ‘Project 2’ , we deleted all the primary key constraints to remove the implicit indexes by executing the following queries in ‘Project2’ :

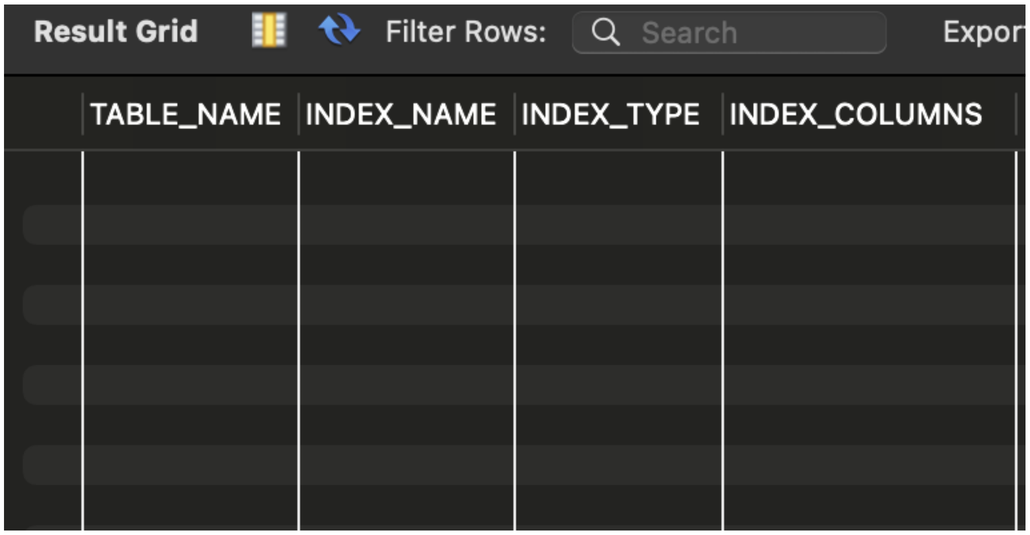


```sql
ALTER TABLE coordinates
DROP PRIMARY KEY;
ALTER TABLE end_stations
DROP PRIMARY KEY;
ALTER TABLE installation_dates
DROP PRIMARY KEY;
ALTER TABLE start_stations
DROP PRIMARY KEY;
ALTER TABLE stations
DROP PRIMARY KEY;
ALTER TABLE status
DROP PRIMARY KEY;
ALTER TABLE temperature_ranges
DROP PRIMARY KEY;
ALTER TABLE trips
DROP PRIMARY KEY;
ALTER TABLE weather
DROP PRIMARY KEY;
```

Then, we executed the the previous query to display all the implicit indexes:


```
SELECT
    t.TABLE_NAME,
    i.INDEX_NAME,
    i.INDEX_TYPE,
    GROUP_CONCAT(c.COLUMN_NAME ) AS INDEX_COLUMNS
FROM
    INFORMATION_SCHEMA.TABLES t
INNER JOIN
    INFORMATION_SCHEMA.STATISTICS i ON t.TABLE_SCHEMA = i.INDEX_SCHEMA AND t.TABLE_NAME = i.TABLE_NAME
INNER JOIN
    INFORMATION_SCHEMA.COLUMNS c ON t.TABLE_SCHEMA = c.TABLE_SCHEMA AND t.TABLE_NAME = c.TABLE_NAME AND i.COLUMN_NAME = c.COLUMN_NAME
WHERE
    t.TABLE_SCHEMA = 'Project2'
GROUP BY
    t.TABLE_NAME, i.INDEX_NAME, i.INDEX_TYPE;
```

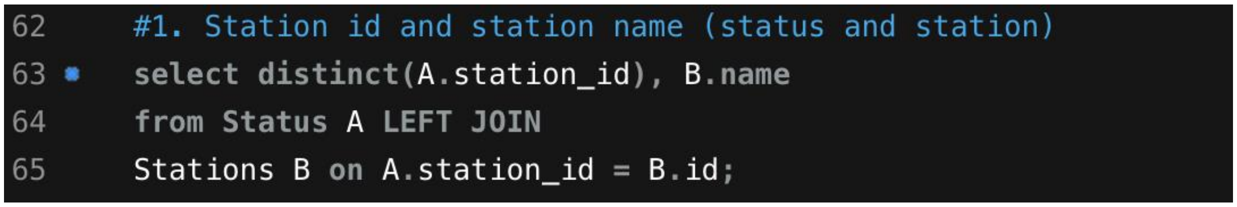
The output below displays that all the indexes have been removed from the database :

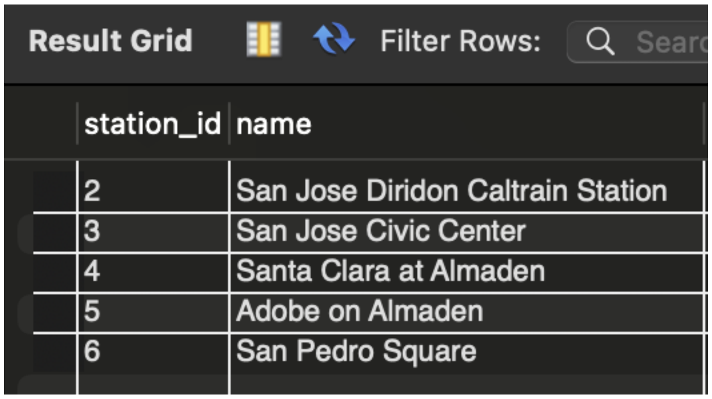


TABLE_NAME	INDEX_NAME	INDEX_TYPE	INDEX_COLUMNS

1.4. Write at least 5 queries (with JOINs between your tables)

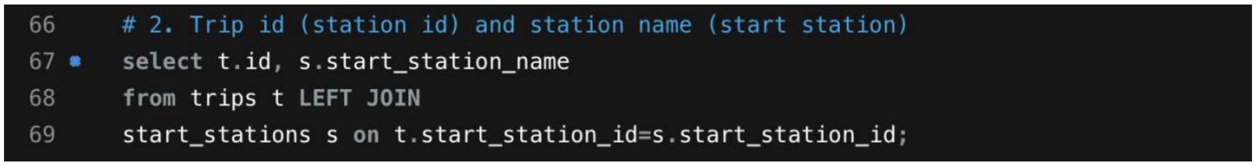
Queries:


```
#1. Station id and station name (status and station)
select distinct(A.station_id), B.name
from Status A LEFT JOIN
Stations B on A.station_id = B.id;
```

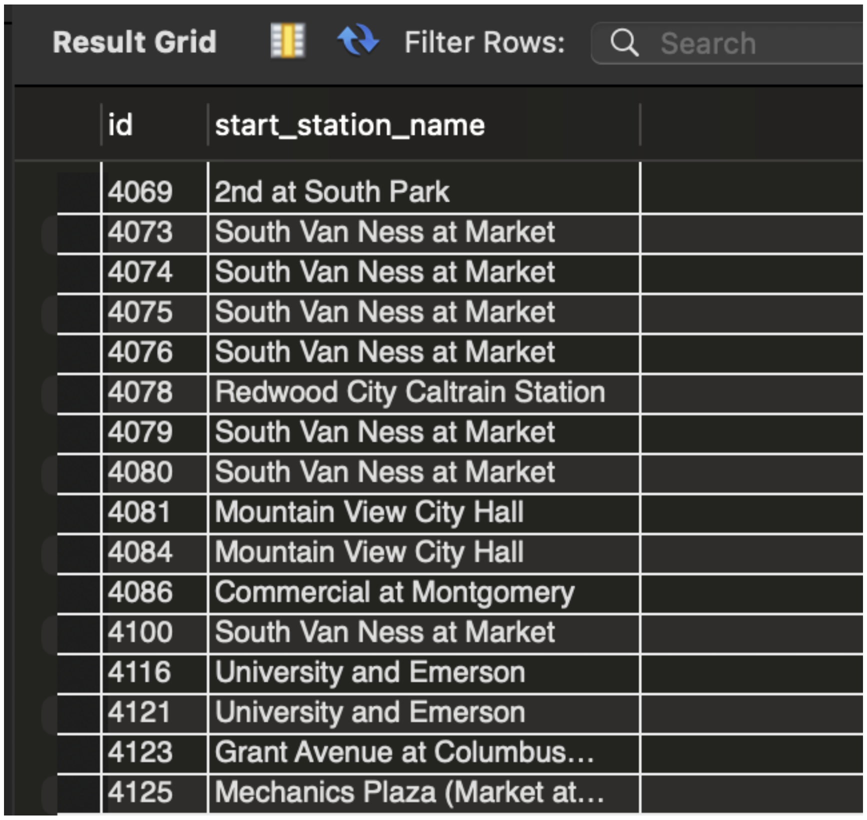


station_id	name
2	San Jose Diridon Caltrain Station
3	San Jose Civic Center
4	Santa Clara at Almaden
5	Adobe on Almaden
6	San Pedro Square

This query retrieves a unique list of station IDs from the "Status" table, along with their corresponding names from the "Stations" table.

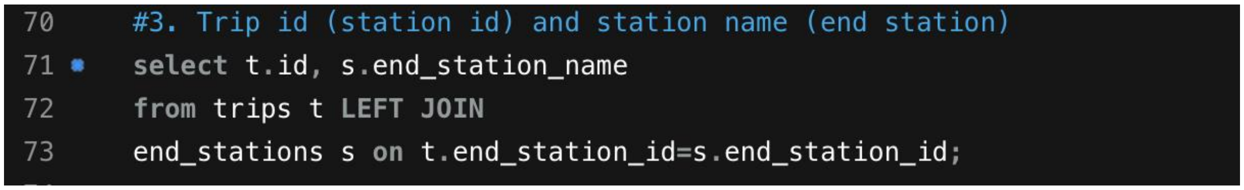


```
# 2. Trip id (station id) and station name (start station)
select t.id, s.start_station_name
from trips t LEFT JOIN
start_stations s on t.start_station_id=s.start_station_id;
```

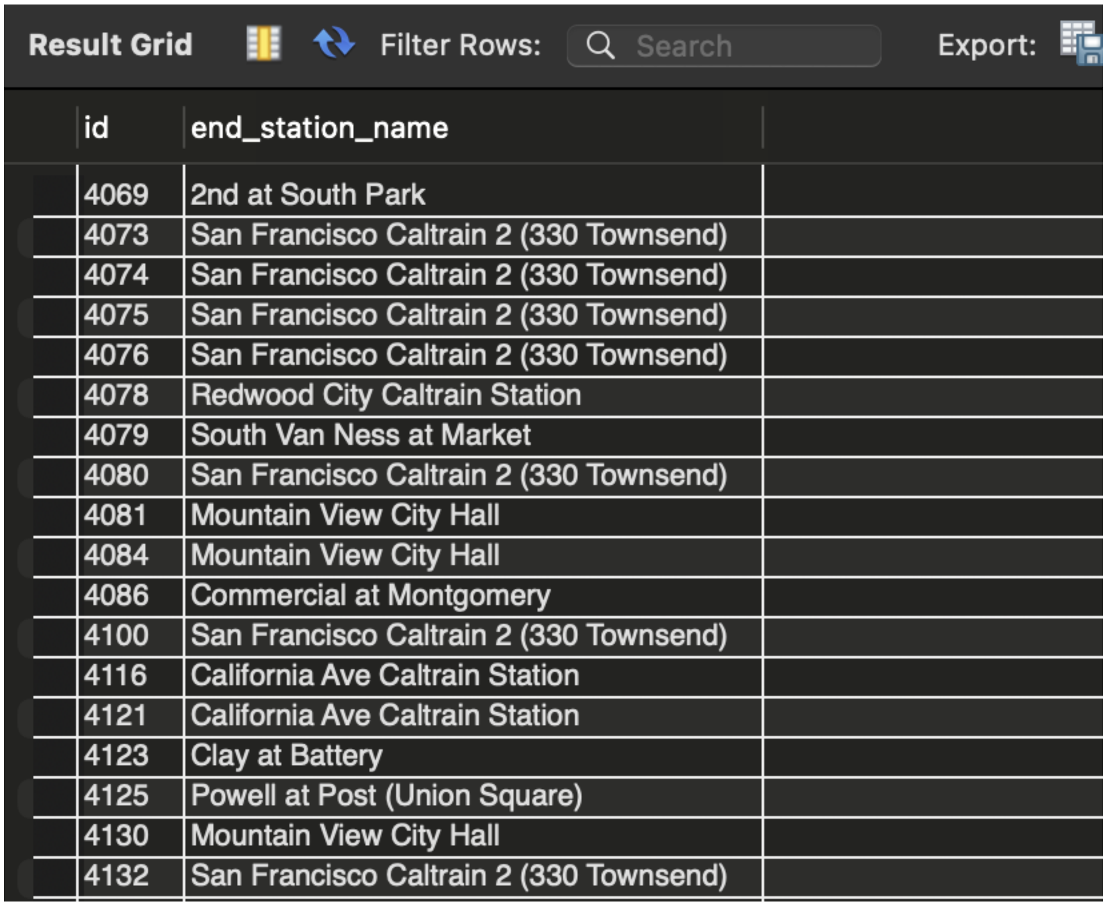


id	start_station_name
4069	2nd at South Park
4073	South Van Ness at Market
4074	South Van Ness at Market
4075	South Van Ness at Market
4076	South Van Ness at Market
4078	Redwood City Caltrain Station
4079	South Van Ness at Market
4080	South Van Ness at Market
4081	Mountain View City Hall
4084	Mountain View City Hall
4086	Commercial at Montgomery
4100	South Van Ness at Market
4116	University and Emerson
4121	University and Emerson
4123	Grant Avenue at Columbus…
4125	Mechanics Plaza (Market at…

This query retrieves the ID of trips along with the corresponding start station names from two tables, "trips" (referenced as alias 't') and "start_stations" (referenced as alias 's').

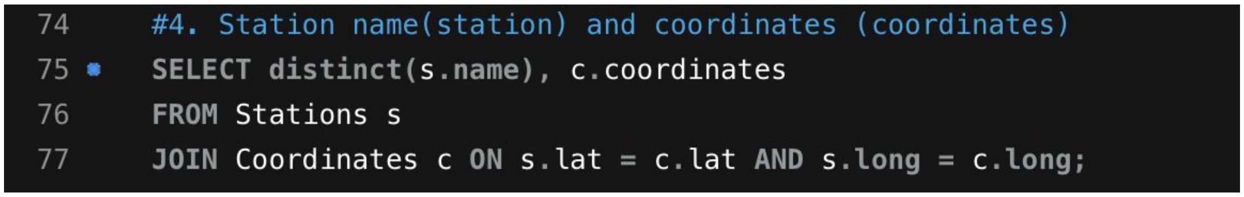


```sql
#3. Trip id (station id) and station name (end station)
select t.id, s.end_station_name
from trips t LEFT JOIN
end_stations s on t.end_station_id=s.end_station_id;
```

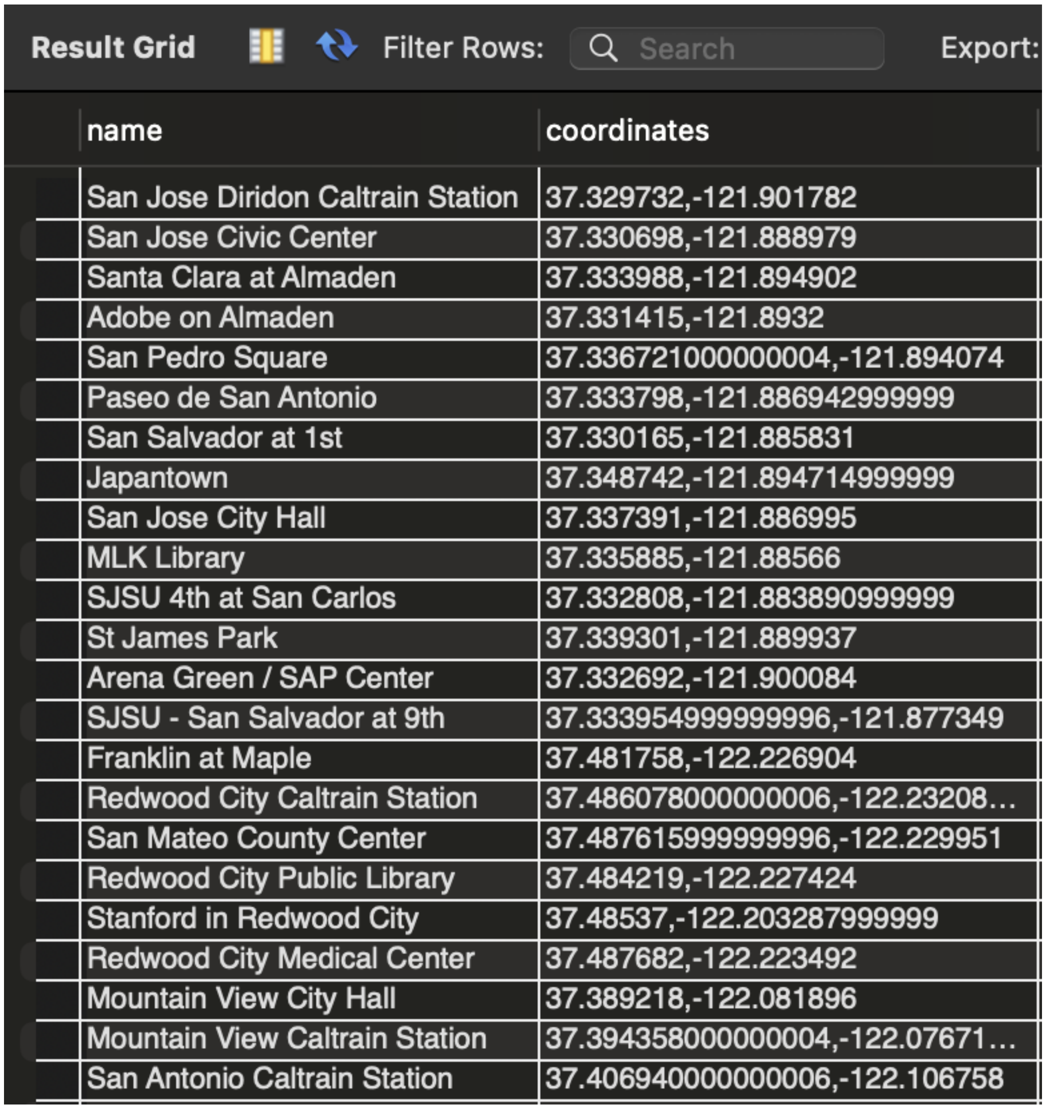


id	end_station_name
4069	2nd at South Park
4073	San Francisco Caltrain 2 (330 Townsend)
4074	San Francisco Caltrain 2 (330 Townsend)
4075	San Francisco Caltrain 2 (330 Townsend)
4076	San Francisco Caltrain 2 (330 Townsend)
4078	Redwood City Caltrain Station
4079	South Van Ness at Market
4080	San Francisco Caltrain 2 (330 Townsend)
4081	Mountain View City Hall
4084	Mountain View City Hall
4086	Commercial at Montgomery
4100	San Francisco Caltrain 2 (330 Townsend)
4116	California Ave Caltrain Station
4121	California Ave Caltrain Station
4123	Clay at Battery
4125	Powell at Post (Union Square)
4130	Mountain View City Hall
4132	San Francisco Caltrain 2 (330 Townsend)

This query retrieves the ID of trips along with the corresponding end station names from two tables, "trips" (referenced as alias 't') and "end_stations" (referenced as alias 's').



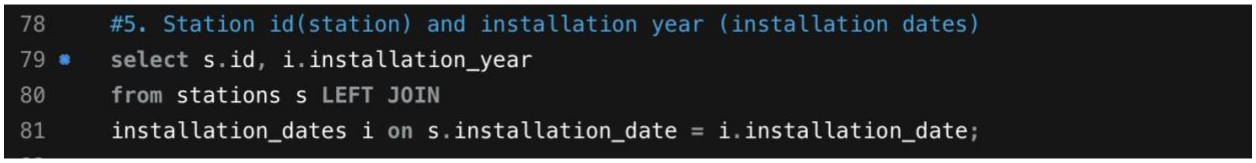
```sql
#4. Station name(station) and coordinates (coordinates)
SELECT distinct(s.name), c.coordinates
FROM Stations s
JOIN Coordinates c ON s.lat = c.lat AND s.long = c.long;
```



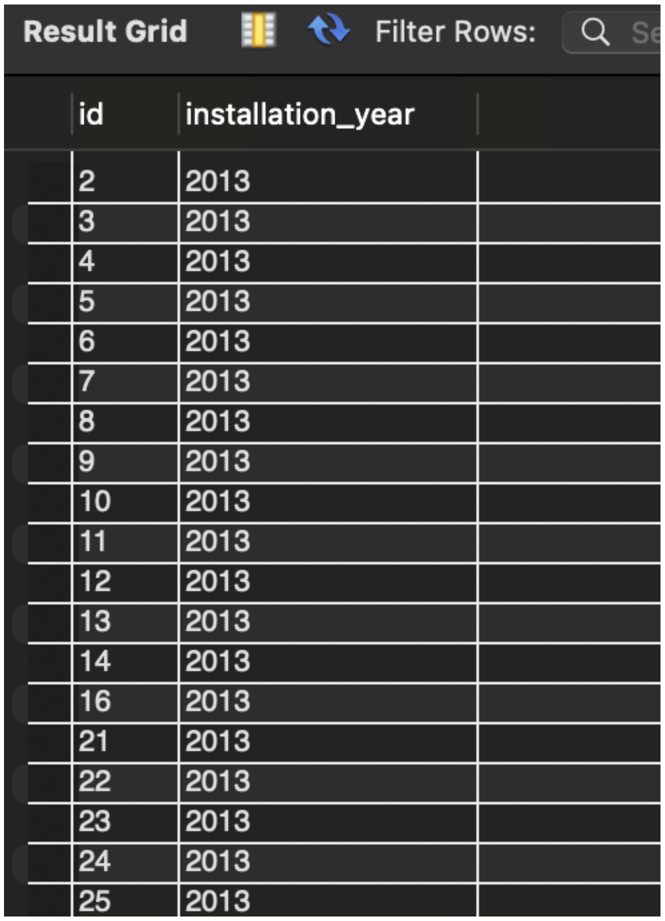
Result Grid Filter Rows: Search Export:

name	coordinates
San Jose Diridon Caltrain Station	37.329732,-121.901782
San Jose Civic Center	37.330698,-121.888979
Santa Clara at Almaden	37.333988,-121.894902
Adobe on Almaden	37.331415,-121.8932
San Pedro Square	37.336721000000004,-121.894074
Paseo de San Antonio	37.333798,-121.886942999999
San Salvador at 1st	37.330165,-121.885831
Japantown	37.348742,-121.894714999999
San Jose City Hall	37.337391,-121.886995
MLK Library	37.335885,-121.88566
SJSU 4th at San Carlos	37.332808,-121.883890999999
St James Park	37.339301,-121.889937
Arena Green / SAP Center	37.332692,-121.900084
SJSU - San Salvador at 9th	37.333954999999996,-121.877349
Franklin at Maple	37.481758,-122.226904
Redwood City Caltrain Station	37.486078000000006,-122.23208…
San Mateo County Center	37.487615999999996,-122.229951
Redwood City Public Library	37.484219,-122.227424
Stanford in Redwood City	37.48537,-122.203287999999
Redwood City Medical Center	37.487682,-122.223492
Mountain View City Hall	37.389218,-122.081896
Mountain View Caltrain Station	37.394358000000004,-122.07671…
San Antonio Caltrain Station	37.406940000000006,-122.106758

This query selects distinct station names along with their corresponding coordinates from two tables, "Stations" (aliased as 's') and "Coordinates" (aliased as 'c').



```
#5. Station id(station) and installation year (installation dates)
select s.id, i.installation_year
from stations s LEFT JOIN
installation_dates i on s.installation_date = i.installation_date;
```



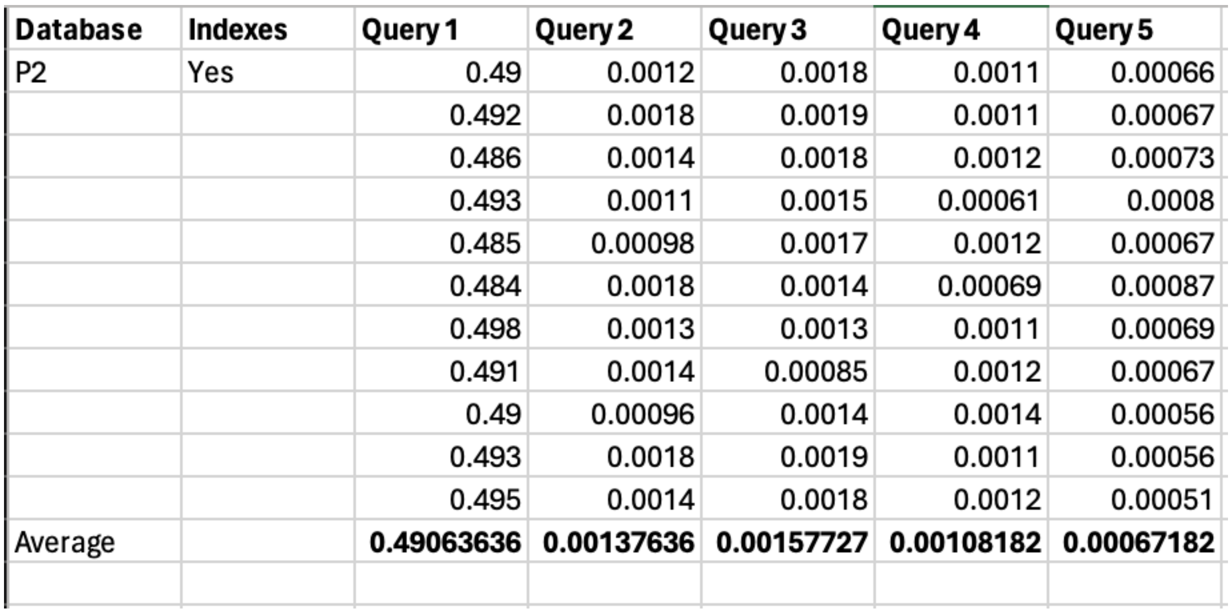
Result Grid Filter Rows:

id	installation_year
2	2013
3	2013
4	2013
5	2013
6	2013
7	2013
8	2013
9	2013
10	2013
11	2013
12	2013
13	2013
14	2013
16	2013
21	2013
22	2013
23	2013
24	2013
25	2013

This query retrieves the IDs of stations along with their corresponding installation years from two tables, "stations" (aliased as 's') and "installation_dates" (aliased as 'i').

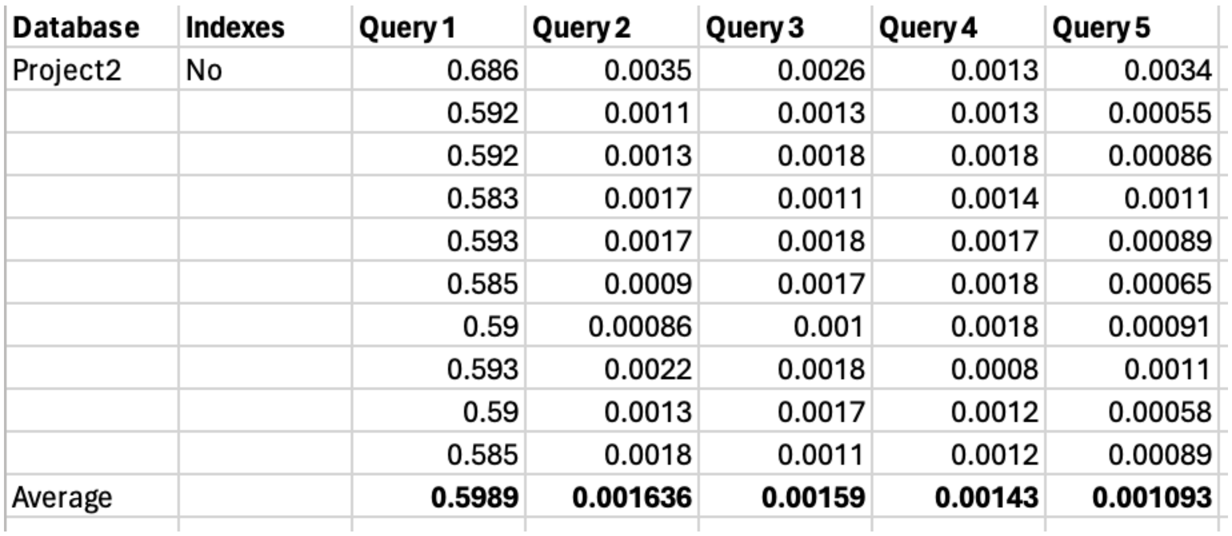
1.5. Execute and time these queries on both databases and report your findings (repeat timing for each query at least 10 times and average the times)

When we executed these 5 JOIN queries on the database “P2” that has indexes, we got the following results for the time it took to run the queries :



Database	Indexes	Query 1	Query 2	Query 3	Query 4	Query 5
P2	Yes	0.49	0.0012	0.0018	0.0011	0.00066
		0.492	0.0018	0.0019	0.0011	0.00067
		0.486	0.0014	0.0018	0.0012	0.00073
		0.493	0.0011	0.0015	0.00061	0.0008
		0.485	0.00098	0.0017	0.0012	0.00067
		0.484	0.0018	0.0014	0.00069	0.00087
		0.498	0.0013	0.0013	0.0011	0.00069
		0.491	0.0014	0.00085	0.0012	0.00067
		0.49	0.00096	0.0014	0.0014	0.00056
		0.493	0.0018	0.0019	0.0011	0.00056
		0.495	0.0014	0.0018	0.0012	0.00051
Average		**0.49063636**	**0.00137636**	**0.00157727**	**0.00108182**	**0.00067182**

We repeated the same process for the database "Project2" with no indexes and recorded the following results:

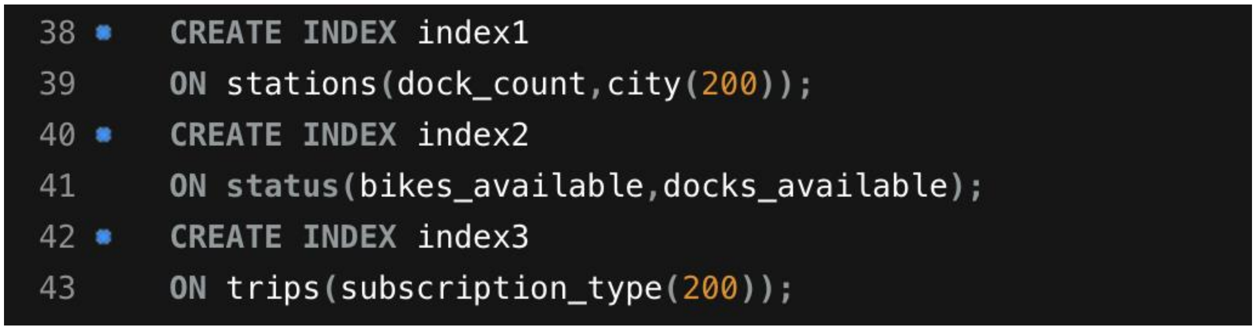


Database	Indexes	Query 1	Query 2	Query 3	Query 4	Query 5
Project2	No	0.686	0.0035	0.0026	0.0013	0.0034
		0.592	0.0011	0.0013	0.0013	0.00055
		0.592	0.0013	0.0018	0.0018	0.00086
		0.583	0.0017	0.0011	0.0014	0.0011
		0.593	0.0017	0.0018	0.0017	0.00089
		0.585	0.0009	0.0017	0.0018	0.00065
		0.59	0.00086	0.001	0.0018	0.00091
		0.593	0.0022	0.0018	0.0008	0.0011
		0.59	0.0013	0.0017	0.0012	0.00058
		0.585	0.0018	0.0011	0.0012	0.00089
Average		**0.5989**	**0.001636**	**0.00159**	**0.00143**	**0.001093**

In the absence of indexes, Query 1 experiences a 0.108-second delay in execution compared to its counterpart with indexes. Similarly, Query 2 encounters a slight 0.00026-second lag, while Query 3, Query 4, and Query 5 face delays of 0.0002, 0.0032, and 0.00042 seconds, respectively. These results strongly suggest that the removal of indexes has adversely impacted the execution speed of these queries. The inference drawn from the provided data is that indexes play a crucial role in enhancing query performance.

1.6. Select some columns from database A (columns that are not already indexed) and create index on them.

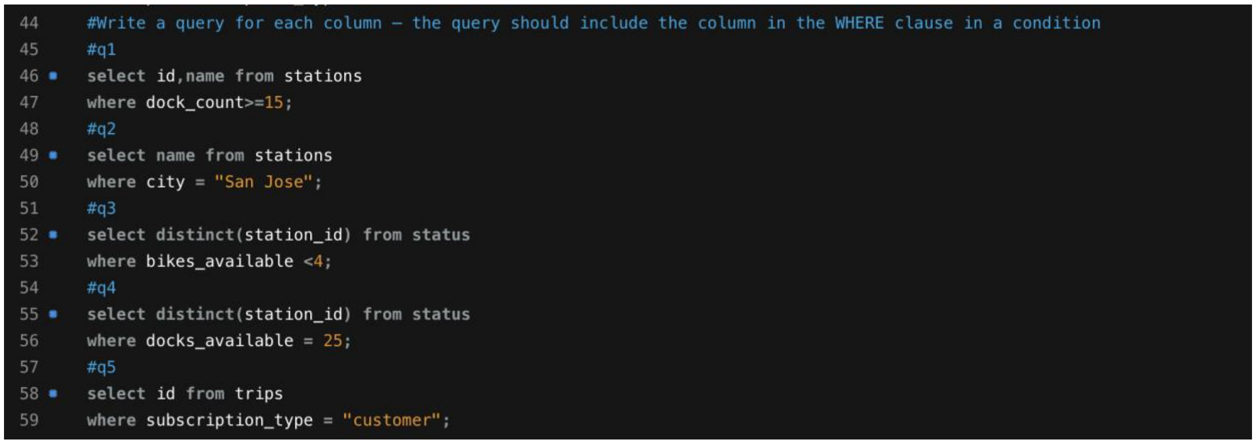
We have selected 5 columns namely dock_count, city, bikes_available, docks_available and subscription_type and created indexes on them using the following queries:



```sql
CREATE INDEX index1
ON stations(dock_count,city(200));
CREATE INDEX index2
ON status(bikes_available,docks_available);
CREATE INDEX index3
ON trips(subscription_type(200));
```

1.7. Write a query for each column – the query should include the column in the WHERE clause in a condition

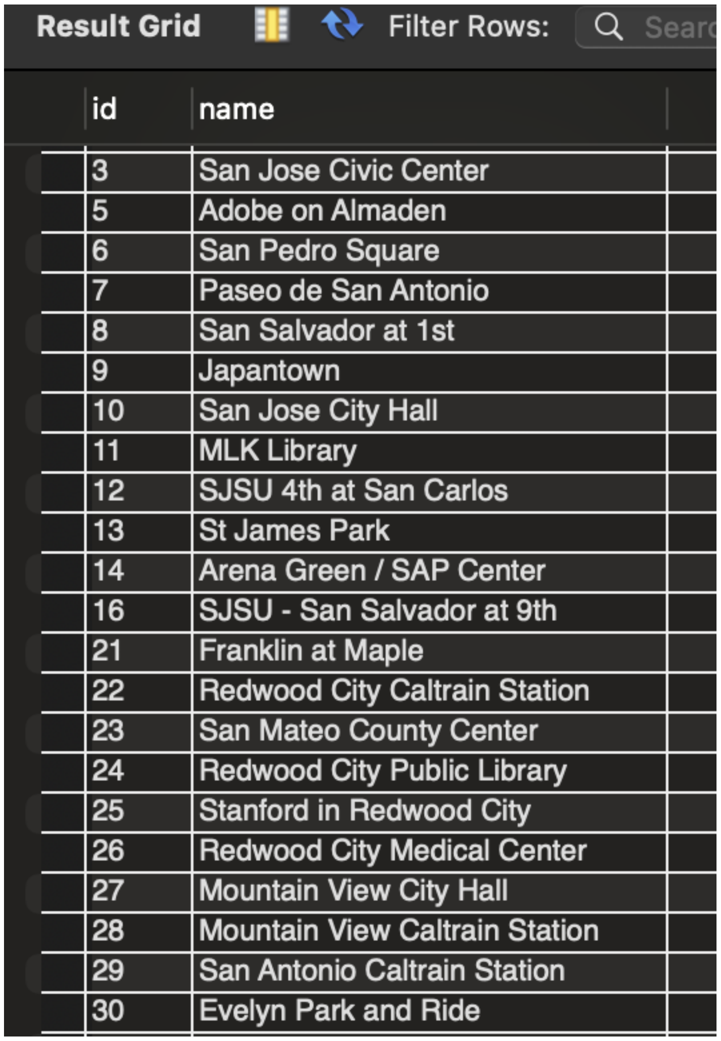
We have written the following queries for the 5 columns mentioned above:



```sql
#Write a query for each column — the query should include the column in the WHERE clause in a condition
#q1
select id,name from stations
where dock_count>=15;
#q2
select name from stations
where city = "San Jose";
#q3
select distinct(station_id) from status
where bikes_available <4;
#q4
select distinct(station_id) from status
where docks_available = 25;
#q5
select id from trips
where subscription_type = "customer";
```

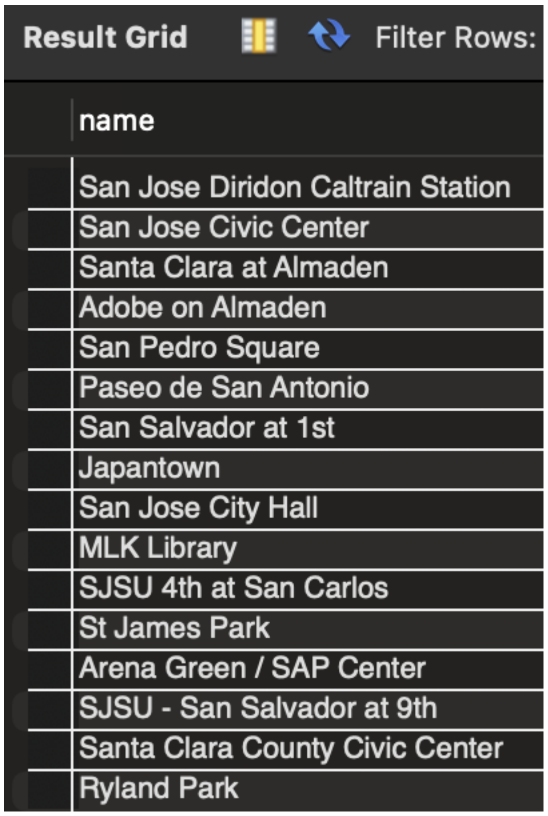
Here are the outputs for the above queries

Query 1



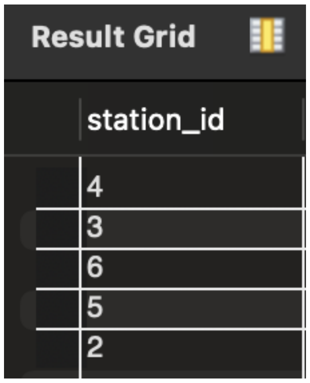
id	name
3	San Jose Civic Center
5	Adobe on Almaden
6	San Pedro Square
7	Paseo de San Antonio
8	San Salvador at 1st
9	Japantown
10	San Jose City Hall
11	MLK Library
12	SJSU 4th at San Carlos
13	St James Park
14	Arena Green / SAP Center
16	SJSU - San Salvador at 9th
21	Franklin at Maple
22	Redwood City Caltrain Station
23	San Mateo County Center
24	Redwood City Public Library
25	Stanford in Redwood City
26	Redwood City Medical Center
27	Mountain View City Hall
28	Mountain View Caltrain Station
29	San Antonio Caltrain Station
30	Evelyn Park and Ride

Query 2


Result Grid Filter Rows:

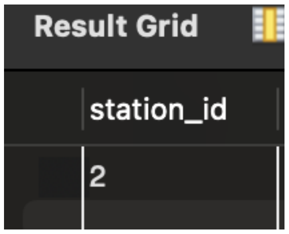
```
name
San Jose Diridon Caltrain Station
San Jose Civic Center
Santa Clara at Almaden
Adobe on Almaden
San Pedro Square
Paseo de San Antonio
San Salvador at 1st
Japantown
San Jose City Hall
MLK Library
SJSU 4th at San Carlos
St James Park
Arena Green / SAP Center
SJSU - San Salvador at 9th
Santa Clara County Civic Center
Ryland Park
```

Query 3


Result Grid

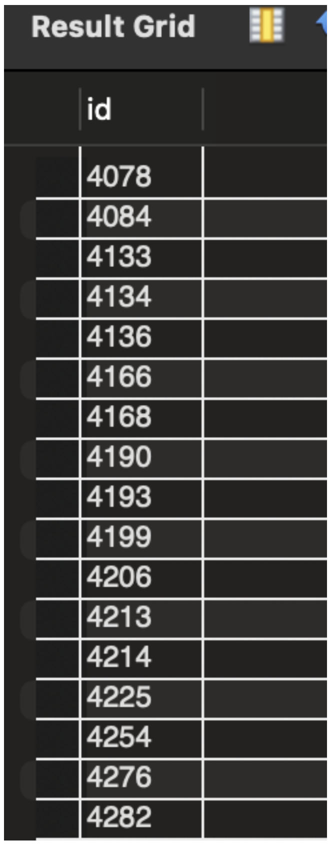
```
station_id
4
3
6
5
2
```

Query 4


Result Grid

```
station_id
2
```

Query 5

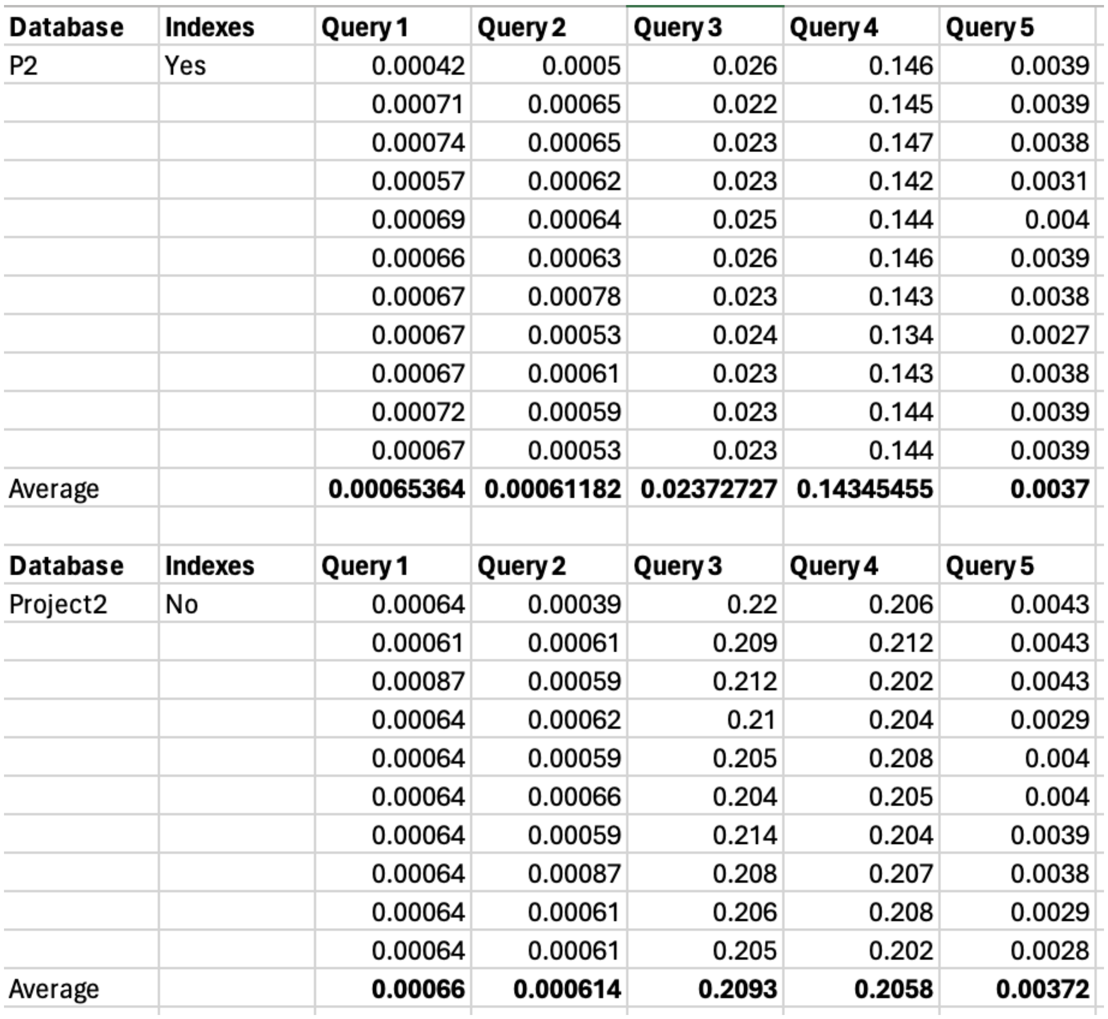


Result Grid

id
4078
4084
4133
4134
4136
4166
4168
4190
4193
4199
4206
4213
4214
4225
4254
4276
4282

1.8. Execute and time these queries on both databases and report your findings (repeat timing for each query at least 10 times and average the times)

After executing and timing the queries on both databases, here are our results:



Database	Indexes	Query 1	Query 2	Query 3	Query 4	Query 5
P2	Yes	0.00042	0.0005	0.026	0.146	0.0039
		0.00071	0.00065	0.022	0.145	0.0039
		0.00074	0.00065	0.023	0.147	0.0038
		0.00057	0.00062	0.023	0.142	0.0031
		0.00069	0.00064	0.025	0.144	0.004
		0.00066	0.00063	0.026	0.146	0.0039
		0.00067	0.00078	0.023	0.143	0.0038
		0.00067	0.00053	0.024	0.134	0.0027
		0.00067	0.00061	0.023	0.143	0.0038
		0.00072	0.00059	0.023	0.144	0.0039
		0.00067	0.00053	0.023	0.144	0.0039
Average		**0.00065364**	**0.00061182**	**0.02372727**	**0.14345455**	**0.0037**

Database	Indexes	Query 1	Query 2	Query 3	Query 4	Query 5
Project2	No	0.00064	0.00039	0.22	0.206	0.0043
		0.00061	0.00061	0.209	0.212	0.0043
		0.00087	0.00059	0.212	0.202	0.0043
		0.00064	0.00062	0.21	0.204	0.0029
		0.00064	0.00059	0.205	0.208	0.004
		0.00064	0.00066	0.204	0.205	0.004
		0.00064	0.00059	0.214	0.204	0.0039
		0.00064	0.00087	0.208	0.207	0.0038
		0.00064	0.00061	0.206	0.208	0.0029
		0.00064	0.00061	0.205	0.202	0.0028
Average		**0.00066**	**0.000614**	**0.2093**	**0.2058**	**0.00372**

Based on the provided results, we can infer that having indexes in the database improves query performance compared to when indexes are not present. In the case of the "P2" database with indexes, the average query execution time across all queries is consistently lower than that of the "Project2" database without indexes. 1.9. Make a conclusion based on your findings in this part

1.9. Make a conclusion based on your findings in this part

The analysis underscores the pivotal role of indexes in optimizing database performance. Queries executed on the "P2" database, which incorporates indexes, consistently demonstrated improved execution times compared to the "Project2" database, where indexes were absent. This disparity highlights the significance of index implementation in enhancing query efficiency and overall system responsiveness. The findings underscore the importance of strategic database design, emphasizing the necessity of incorporating indexes to expedite data retrieval processes and maximize system performance.