Agenda: Best Practices of Combining table/relations using JOIN SQL statement.

Scenario: This database includes three tables: city, weather, and forecast. The city table stores information about the cities, including the city's unique id, name, and state. The weather table stores the daily weather data for each city, including temperature, humidity, precipitation, and the city_id foreign key referencing the city table.

The forecast table stores the weather forecast data for each city, including the forecast date, high temperature, low temperature, and the city_id foreign key referencing the city table.

The four records inserted into each table provide some sample data for New York, Los Angeles, Chicago, and Houston. The city table has one record for each city. The weather table has one record for each city with daily weather data. The forecast table has one record for each city with the forecast data for a specific date.

Starter Code :

```
CREATE DATABASE weatherdb;
USE weatherdb;

-- create the tables
CREATE TABLE city (
    id INT PRIMARY KEY,
    name VARCHAR(255),
    state VARCHAR(255)
);

CREATE TABLE weather (
    id INT PRIMARY KEY,
    temperature FLOAT,
    humidity FLOAT,
    precipitation FLOAT,
    city_id INT,
    FOREIGN KEY (city_id) REFERENCES city(id)
);
CREATE TABLE forecast (
    id INT PRIMARY KEY,
    date DATE,
    high FLOAT,
    low FLOAT,
    city_id INT,
    FOREIGN KEY (city_id) REFERENCES city(id)
);

-- insert data into tables
INSERT INTO city (id, name, state) VALUES
```

(1, 'New York', 'New York'),
(2, 'Los Angeles', 'California'),
(3, 'Chicago', 'Illinois'),
(4, 'Houston', 'Texas');

INSERT INTO weather (id, temperature, humidity, precipitation, city_id) VALUES
(1, 64.2, 60.1, 0.0, 1),
(2, 75.6, 45.3, 0.0, 2),
(3, 55.3, 80.2, 0.3, 3),
(4, 88.9, 35.6, 0.0, 4);

INSERT INTO forecast (id, date, high, low, city_id) VALUES
(1, '2023-04-06', 73.2, 57.1, 1),
(2, '2023-04-06', 81.2, 64.4, 2),
(3, '2023-04-06', 60.1, 45.3, 3),
(4, '2023-04-06', 92.1, 75.6, 4);

SHOW TABLES;
SELECT * FROM city;
SELECT * FROM weather;
SELECT * FROM forecast;
Following Queries to Solve with MySQL :

1. Retrieve all weather data and city names for each record, using natural join.
   Sample Answer :

   ```
   SELECT *
   FROM weather
   NATURAL JOIN city;
   ```

2. Retrieve all forecast data and city names for each record, using a natural join 3. Retrieve all weather data, forecast data, and city names for each record, using a theta join on city_id
4. Retrieve all weather data for New York city, using a left outer join
5. Retrieve all forecast data for Los Angeles city, using a left outer join 6. Retrieve all weather and forecast data for Chicago city, using a left outer join 7. Retrieve all weather data and city names for each record, using an inner join 8. Retrieve all forecast data and city names for each record, using an inner join 9. Retrieve all weather data, forecast data, and city names for each record, using a cross join
10. Retrieve all weather data and city names for each record, using a right outer

join: **Extended Queries tasks [Aggregation, Having, and Group BY]**

1. Retrieve the average temperature for each city, with city names:
2. Retrieve the maximum temperature for each city, with city names, only for cities where the maximum temperature is greater than 80

3. Retrieve the sum of precipitation for each city, with city names:
4. Retrieve the average temperature for each month, with month names: