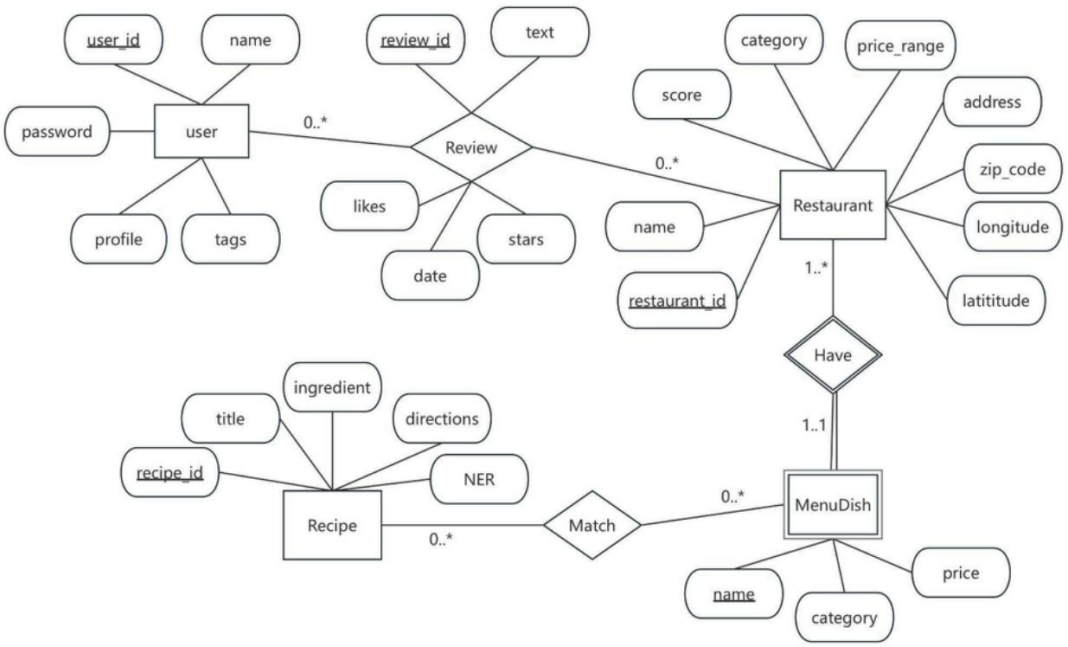
Milestone 3

Credentials:


```
{
"rds_host": "database-1.cfcenfnbcvhy.us-east-
1.rds.amazonaws.com",
"rds_port": "5432",
"rds_user": "cis5500",
"rds_password": "kUWDP0g66ONQPkaiMKMo",
"rds_db": "postgres"
}
```

Note that:

1. All queries below are based on the above tables in the ER diagram.
2. Most of the queries are complex and optimized, while the remaining ones are relatively simple and do not require optimization.
3. The final delivery of the project may differ from this version.

Query 1 - Most Liked Review Per Active User

Find each active user who reviewed at least 3 different restaurants in the last 90 days and return their most liked review during that period. For each review, include the review text, number of likes, score, date, the restaurant's name and location (latitude and longitude), and the user's name.

Unoptimized Version:

```
SELECT
  u.name AS user_name,
  rev.text,
  rev.likes,
  rev.score,
  rev.date,
  r.name AS restaurant_name,
  r.latitude,
  r.longitude
FROM Review rev
JOIN User u ON rev.user_id = u.user_id
JOIN Restaurant r ON rev.restaurant_id = r.restaurant_id
WHERE rev.date >= NOW() - INTERVAL 90 DAY
  AND rev.user_id IN (
    SELECT user_id
    FROM Review
    WHERE date >= NOW() - INTERVAL 90 DAY
    GROUP BY user_id
    HAVING COUNT(DISTINCT restaurant_id) >= 3
  )
  AND rev.likes = (
    SELECT MAX(r2.likes)
    FROM Review r2
    WHERE r2.user_id = rev.user_id
      AND r2.date >= NOW() - INTERVAL 90 DAY
  );
```

Optimized Version:

```
CREATE INDEX idx_review_user_rest_date_likes
ON Review(user_id, restaurant_id, date, likes);

CREATE INDEX idx_review_user_rest
ON Review(user_id, restaurant_id);

WITH recent_reviews AS (
  SELECT *
  FROM Review
  WHERE date >= NOW() - INTERVAL 90 DAY
),
active_users AS (
  SELECT user_id
  FROM recent_reviews
  GROUP BY user_id
  HAVING COUNT(DISTINCT restaurant_id) >= 3
),
ranked_reviews AS (
  SELECT *,
         ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY likes DESC) AS rk
  FROM recent_reviews
)
SELECT
  u.name AS user_name,
  rr.text,
  rr.likes,
  rr.score,
  rr.date,
  r.name AS restaurant_name,
  r.latitude,
  r.longitude
FROM ranked_reviews rr
JOIN active_users au ON rr.user_id = au.user_id
JOIN User u ON rr.user_id = u.user_id
JOIN Restaurant r ON rr.restaurant_id = r.restaurant_id
WHERE rr.rk = 1;
```

Query 2 - Dish Category Averages and Price Outliers

For each dish category, calculate the average score and the price range (minimum and maximum) of all dishes in that category. Then, identify specific dishes whose price is at least 20% higher than the category's average price and whose associated reviews have a score greater than 4. For each of these dishes, include the dish name, price, score, the name of the restaurant offering it, and the restaurant's geographic location (latitude and longitude).

Unoptimized Version:

```
SELECT
  md.name AS dish_name,
  md.category,
  md.price,
  rev.score,
  r.name AS restaurant_name,
  r.latitude,
  r.longitude,
  cs.avg_price,
  cs.min_price,
  cs.max_price
FROM MenuDish md
JOIN Have h ON md.name = h.name
JOIN Restaurant r ON h.restaurant_id = r.restaurant_id
JOIN Review rev ON rev.restaurant_id = r.restaurant_id
JOIN (
    SELECT
      category,
      AVG(price) AS avg_price,
      MIN(price) AS min_price,
      MAX(price) AS max_price
    FROM MenuDish
    GROUP BY category
) cs ON md.category = cs.category
WHERE md.price > cs.avg_price * 1.2
  AND rev.score > 4;
```

Optimized Version:

```
CREATE INDEX idx_menu_category_price ON MenuDish(category, price);
CREATE INDEX idx_review_restaurant_score_date ON Review(restaurant_id, score, date);
CREATE INDEX idx_have_name_restaurant ON Have(name, restaurant_id);

WITH category_stats AS (
  SELECT
    category,
    AVG(price) AS avg_price,
    MIN(price) AS min_price,
    MAX(price) AS max_price
  FROM MenuDish
  GROUP BY category
),
recent_reviews AS (
  SELECT *
  FROM Review
  WHERE date >= NOW() - INTERVAL 90 DAY
    AND score > 4
),
dish_reviews AS (
  SELECT
    md.name AS dish_name,
    md.category,
    md.price,
    rev.score,
    r.name AS restaurant_name,
    r.latitude,
    r.longitude
  FROM MenuDish md
  JOIN Have h ON md.name = h.name
  JOIN Restaurant r ON r.restaurant_id = h.restaurant_id
  JOIN recent_reviews rev ON rev.restaurant_id = r.restaurant_id
)
SELECT
  dr.*,
  cs.avg_price,
  cs.min_price,
  cs.max_price
FROM dish_reviews dr
JOIN category_stats cs ON dr.category = cs.category
WHERE dr.price > cs.avg_price * 1.2;
```

Query 3 - Top Restaurants with Review Counts

Find the top 5 restaurants with the highest number of reviews in the past 90 days, considering only those that serve at least 20 different dishes. For each restaurant, return its name, geographic location (latitude and longitude), the total number of reviews during that period, the average review score, the number of unique users who reviewed it, and the number of distinct dishes offered.

Unoptimized Version:

```
SELECT
  r.name AS restaurant_name,
  r.latitude,
  r.longitude,
  COUNT(rev.review_id) AS review_count,
  AVG(rev.score) AS avg_score,
  COUNT(DISTINCT rev.user_id) AS unique_users,
  COUNT(DISTINCT md.name) AS dish_count
FROM Restaurant r
JOIN Review rev ON r.restaurant_id = rev.restaurant_id
JOIN Have h ON r.restaurant_id = h.restaurant_id
JOIN MenuDish md ON h.name = md.name
WHERE rev.date >= NOW() - INTERVAL 90 DAY
GROUP BY r.restaurant_id, r.name, r.latitude, r.longitude
HAVING COUNT(DISTINCT md.name) >= 20
ORDER BY review_count DESC
LIMIT 5;
```

Optimized Version:

```
CREATE INDEX idx_review_restaurant_date_user ON Review(restaurant_id, date, user_id);
CREATE INDEX idx_have_rest_dish ON Have(restaurant_id, name);
CREATE INDEX idx_menu_name ON MenuDish(name);

CREATE VIEW restaurant_90day_review_summary AS
SELECT
  restaurant_id,
  COUNT(*) AS review_count,
  AVG(score) AS avg_score,
  COUNT(DISTINCT user_id) AS unique_users
FROM Review
WHERE date >= NOW() - INTERVAL 90 DAY
GROUP BY restaurant_id;

SELECT
  r.name AS restaurant_name,
  r.latitude,
  r.longitude,
  rs.review_count,
  rs.avg_score,
  rs.unique_users,
  COUNT(DISTINCT md.name) AS dish_count
FROM restaurant_90day_review_summary rs
JOIN Restaurant r ON rs.restaurant_id = r.restaurant_id
JOIN Have h ON r.restaurant_id = h.restaurant_id
JOIN MenuDish md ON h.name = md.name
GROUP BY r.restaurant_id, r.name, r.latitude, r.longitude,
rs.review_count, rs.avg_score, rs.unique_users
HAVING COUNT(DISTINCT md.name) >= 20
ORDER BY rs.review_count DESC
LIMIT 5;
```

Query 4 - Highest-Rated Restaurant and Most Common Dish Category

For each user, find their highest-rated restaurant. Then, among the dishes they have reviewed at that restaurant, return those that belong to the most common dish category served by that restaurant.

Unoptimized Version:

```
SELECT
  u.user_id,
  u.name AS user_name,
  r.restaurant_id,
  r.name AS restaurant_name,
  md.name AS dish_name,
  md.category
FROM User u
JOIN Review rev ON u.user_id = rev.user_id
JOIN Restaurant r ON r.restaurant_id = rev.restaurant_id
JOIN Have h ON r.restaurant_id = h.restaurant_id
JOIN MenuDish md ON h.name = md.name
WHERE rev.score = (
  SELECT MAX(r2.score)
  FROM Review r2
  WHERE r2.user_id = u.user_id
)
  AND md.name IN (
    SELECT h2.name
    FROM Review rev2
    JOIN Have h2 ON rev2.restaurant_id = h2.restaurant_id
    WHERE rev2.user_id = u.user_id
      AND rev2.restaurant_id = r.restaurant_id
  )
  AND md.category = (
    SELECT md3.category
    FROM Have h3
    JOIN MenuDish md3 ON h3.name = md3.name
    WHERE h3.restaurant_id = r.restaurant_id
    GROUP BY md3.category
    ORDER BY COUNT(*) DESC
    LIMIT 1
  );
```

Optimized Version:

```
CREATE INDEX idx_review_user_rest_score ON Review(user_id, restaurant_id, score);
CREATE INDEX idx_have_rest_dish ON Have(restaurant_id, name);
CREATE INDEX idx_menu_name_category ON MenuDish(name, category);

WITH user_top_restaurant AS (
  SELECT user_id, restaurant_id, MAX(score) AS top_score
  FROM Review
  GROUP BY user_id
),
user_restaurant_selected AS (
  SELECT r.user_id, r.restaurant_id
  FROM Review r
  JOIN user_top_restaurant ut
    ON r.user_id = ut.user_id AND r.restaurant_id = ut.restaurant_id AND r.score = ut.top_score
),
restaurant_main_category AS (
  SELECT h.restaurant_id, md.category
  FROM Have h
  JOIN MenuDish md ON h.name = md.name
  GROUP BY h.restaurant_id, md.category
  HAVING COUNT(*) = (
    SELECT MAX(cat_count)
    FROM (
      SELECT COUNT(*) AS cat_count
      FROM Have h2
      JOIN MenuDish md2 ON h2.name = md2.name
      WHERE h2.restaurant_id = h.restaurant_id
      GROUP BY md2.category
    ) AS counts
  )
),

user_reviewed_dishes AS (
  SELECT rev.user_id, rev.restaurant_id, md.name AS dish_name, md.category
  FROM Review rev
  JOIN Have h ON rev.restaurant_id = h.restaurant_id
  JOIN MenuDish md ON h.name = md.name
  WHERE rev.user_id IS NOT NULL
)
SELECT
  u.name AS user_name,
  r.name AS restaurant_name,
  urd.dish_name,
  urd.category
FROM user_restaurant_selected ur
JOIN User u ON ur.user_id = u.user_id
JOIN Restaurant r ON ur.restaurant_id = r.restaurant_id
JOIN user_reviewed_dishes urd ON ur.user_id = urd.user_id AND ur.restaurant_id = urd.restaurant_id
JOIN restaurant_main_category rc ON ur.restaurant_id = rc.restaurant_id AND urd.category = rc.category;
```

Query 5 - Users Reviewing Every Restaurant Serving a Specific Dish

Find all users who have reviewed every restaurant where a specific dish (e.g., "Spicy Ramen") is served. Return the user ID and name of all users.

Unoptimized Version:

```
SELECT u.user_id, u.name
FROM User u
WHERE NOT EXISTS (
  SELECT h.restaurant_id
  FROM Have h
  JOIN MenuDish md ON h.name = md.name
  WHERE md.name = 'Spicy Ramen'
    AND h.restaurant_id NOT IN (
      SELECT r.restaurant_id
      FROM Review r
      WHERE r.user_id = u.user_id
    )
);
```

Optimized Version:

```
CREATE INDEX idx_review_user_rest ON Review(user_id, restaurant_id);
CREATE INDEX idx_have_name_rest ON Have(name, restaurant_id);
CREATE INDEX idx_menu_name ON MenuDish(name);

WITH ramen_restaurants AS (
  SELECT DISTINCT h.restaurant_id
  FROM Have h
  JOIN MenuDish md ON h.name = md.name
  WHERE md.name = 'Spicy Ramen'
),
user_ramen_reviews AS (
  SELECT r.user_id, COUNT(DISTINCT r.restaurant_id) AS reviewed_count
  FROM Review r
  WHERE r.restaurant_id IN (SELECT restaurant_id FROM ramen_restaurants)
  GROUP BY r.user_id
),
ramen_total AS (
  SELECT COUNT(*) AS total_required
  FROM ramen_restaurants
)
SELECT u.user_id, u.name
FROM user_ramen_reviews urr
JOIN User u ON u.user_id = urr.user_id
JOIN ramen_total rt ON 1 = 1
WHERE urr.reviewed_count = rt.total_required;
```

Query 6 - Users Reviewing at Least One Dish from Every Category

Find users who have reviewed at least one dish from every category.

Unoptimized Version:

```
SELECT u.user_id, u.name
FROM User u
WHERE NOT EXISTS (
  SELECT DISTINCT md.category
  FROM MenuDish md
  WHERE NOT EXISTS (
    SELECT 1
    FROM Review rev
    JOIN Have h ON rev.restaurant_id = h.restaurant_id
    WHERE rev.user_id = u.user_id AND h.name IN (
      SELECT name FROM MenuDish WHERE category = md.category
    )
  )
);
```

Optimized Version:

```
WITH all_categories AS (
  SELECT DISTINCT category FROM MenuDish
),
user_categories AS (
  SELECT rev.user_id, md.category
  FROM Review rev
  JOIN Have h ON rev.restaurant_id = h.restaurant_id
  JOIN MenuDish md ON h.name = md.name
  GROUP BY rev.user_id, md.category
),
user_category_counts AS (
  SELECT user_id, COUNT(DISTINCT category) AS user_cat_count
  FROM user_categories
  GROUP BY user_id
),
total_categories AS (
  SELECT COUNT(DISTINCT category) AS total_cat FROM MenuDish
)
SELECT u.user_id, u.name
FROM user_category_counts uc
JOIN total_categories tc ON 1=1
JOIN User u ON u.user_id = uc.user_id
WHERE uc.user_cat_count = tc.total_cat;
```

Query 7 - Top Recipes Matching High-Rated Dishes

Identify the top 5 recipes that match the highest number of dishes, with an average review score of at least 4. For each recipe, return the recipe title, the number of such high-rated dishes it is matched to, and the names of those dishes.

Unoptimized Version:

```
SELECT
  r.recipe_id,
  r.title,
  COUNT(DISTINCT m.name) AS high_score_dish_count,
  GROUP_CONCAT(DISTINCT m.name) AS matched_dishes
FROM Recipe r
JOIN Match m ON r.recipe_id = m.recipe_id
WHERE m.name IN (
  SELECT h.name
  FROM Have h
  JOIN Review rev ON h.restaurant_id = rev.restaurant_id
  GROUP BY h.name
  HAVING AVG(rev.score) >= 4
)
GROUP BY r.recipe_id, r.title
ORDER BY high_score_dish_count DESC
LIMIT 5;
```

Optimized Version:

```
WITH dish_avg_score AS (
  SELECT h.name, AVG(r.score) AS avg_score
  FROM Review r
  JOIN Have h ON r.restaurant_id = h.restaurant_id
  GROUP BY h.name
  HAVING AVG(r.score) >= 4
),
recipe_dish_match AS (
  SELECT m.recipe_id, m.name AS dish_name
  FROM Match m
  JOIN dish_avg_score ds ON m.name = ds.name
),
recipe_dish_count AS (
  SELECT recipe_id, COUNT(*) AS high_score_dish_count
  FROM recipe_dish_match
  GROUP BY recipe_id
)
SELECT
  r.recipe_id,
  r.title,
  rc.high_score_dish_count,
  GROUP_CONCAT(rd.dish_name) AS matched_dishes
FROM Recipe r
JOIN recipe_dish_count rc ON r.recipe_id = rc.recipe_id
JOIN recipe_dish_match rd ON r.recipe_id = rd.recipe_id
GROUP BY r.recipe_id, r.title, rc.high_score_dish_count
ORDER BY rc.high_score_dish_count DESC
LIMIT 5;
```

Query 8 - Recipes Used by Restaurants in Multiple Locations

Find all recipes that are used by restaurants located in at least two different geographic locations. A restaurant is considered to be in a different location if it has a unique combination of latitude and longitude.

```
WITH dish_count AS (
  SELECT restaurant_id, COUNT(DISTINCT name) AS dish_total
  FROM Have
  GROUP BY restaurant_id
),
review_score AS (
  SELECT restaurant_id, AVG(score) AS avg_score
  FROM Review
  GROUP BY restaurant_id
)
SELECT r.restaurant_id, r.name, d.dish_total, s.avg_score
FROM Restaurant r
JOIN dish_count d ON r.restaurant_id = d.restaurant_id
JOIN review_score s ON r.restaurant_id = s.restaurant_id
WHERE d.dish_total >= 5 AND s.avg_score > 4;
```

Query 9 - Recipes with Distinct Geographic Usage

Find all recipes that are used by restaurants located in at least two different geographic coordinates. For each qualifying recipe, return its recipe ID, title, and the number of distinct geographic locations.

```
SELECT r.recipe_id, r.title, COUNT(DISTINCT CONCAT(res.latitude, ',', res.longitude)) AS location_count
FROM Recipe r
JOIN Match m ON r.recipe_id = m.recipe_id
JOIN Have h ON m.name = h.name
JOIN Restaurant res ON h.restaurant_id = res.restaurant_id
GROUP BY r.recipe_id, r.title
HAVING COUNT(DISTINCT CONCAT(res.latitude, ',', res.longitude)) >= 2;
```

Query 10 - User–Restaurant Pairs with Matching Categories

Find all user–restaurant pairs where the user has at least 2 tags that match the categories of dishes served by the restaurant. For each pair, return the user's ID and name, the restaurant's ID and name, and the number of matching categories.

```
WITH restaurant_categories AS (
  SELECT h.restaurant_id, md.category
  FROM Have h
  JOIN MenuDish md ON h.name = md.name
  GROUP BY h.restaurant_id, md.category
),
user_category_match AS (
  SELECT DISTINCT ut.user_id, rc.restaurant_id, rc.category
  FROM UserTag ut
  JOIN restaurant_categories rc ON ut.tag = rc.category
),
user_restaurant_match_count AS (
  SELECT user_id, restaurant_id, COUNT(DISTINCT category) AS matched_category_count
  FROM user_category_match
  GROUP BY user_id, restaurant_id
  HAVING COUNT(DISTINCT category) >= 2
)
SELECT
  urm.user_id,
  u.name AS user_name,
  urm.restaurant_id,
  r.name AS restaurant_name,
  urm.matched_category_count
FROM user_restaurant_match_count urm
JOIN User u ON urm.user_id = u.user_id
JOIN Restaurant r ON urm.restaurant_id = r.restaurant_id;
```