DDL Commands

```sql
CREATE TABLE WorldCities(
    city VARCHAR(50) NOT NULL,
    city_ascii VARCHAR(50) NOT NULL,
    lat DECIMAL(13, 10) NOT NULL,
    lng DECIMAL(13, 10) NOT NULL,
    country VARCHAR(50)
    iso2 VARCHAR(4),
    iso3 VARCHAR(4),
    admin_name VARCHAR(50) NOT NULL,
    capital VARCHAR(50) NOT NULL,
    population INT NOT NULL,
    id VARCHAR(50)
    PRIMARY KEY(id)
);

CREATE TABLE VacationSpots(
    VacationSpotName VARCHAR(50),
    CityId INT NOT NULL,
    LikeCount INT,
    PRIMARY KEY (VacationSpotName),
    FOREIGN KEY (CityId) REFERENCES Cities(CityId) ON DELETE CASCADE
);

CREATE TABLE UserAccounts(
    Username VARCHAR(50),
    UserPassword VARCHAR(100) NOT NULL,
    ProfilePictureUrl VARCHAR(255),
    ProfileDescription VARCHAR(255),
    Gender VARCHAR(20),
    Country VARCHAR(50),
    Age INT,
    PRIMARY KEY (Username)
);

CREATE TABLE Reviews(
    ReviewID INT,
    Username VARCHAR(50) NOT NULL,
    ReviewText VARCHAR(2000),
    ReviewRating INT,
    CreatedAt DATETIME,
    UpdatedAt DATETIME,
    LikeCount INT,
    PRIMARY KEY (ReviewID),
```

```
        FOREIGN KEY (Username) REFERENCES UserAccounts(Username) ON DELETE
            CASCADE
);

CREATE TABLE Images(
    ImageURL VARCHAR(255),
    ReviewID INT NOT NULL,
    PRIMARY KEY (ImageURL),
    FOREIGN KEY (ReviewID) REFERENCES Reviews(ReviewID) ON DELETE CASCADE,
);

CREATE TABLE Follows(
    followerUsername INT,
    followeeUsername INT,
    PRIMARY KEY (followerUsername, followeeUsername),
    FOREIGN KEY (followerUsername) REFERENCES UserAccounts(Username) ON DELETE
            CASCADE,
    FOREIGN KEY (followeeUsername) REFERENCES UserAccounts(Username) ON DELETE
            CASCADE
);

CREATE TABLE FavoriteSpots(
    Username VARCHAR(50),
    VacationSpotName VARCHAR(50),
    PRIMARY KEY(Username, VacationSpotName),
    FOREIGN KEY (Username) REFERENCES UserAccounts(Username) ON DELETE
            CASCADE,
    FOREIGN KEY (VacationSpotName) REFERENCES VacationSpots(VacationSpotName) ON
        DELETE CASCADE
);

CREATE TABLE VacationSpotReviews(
    ReviewId INT NOT NULL,
    VacationSpotName VARCHAR(50),
    PRIMARY KEY (ReviewId, VacationSpotName),
    FOREIGN KEY (ReviewId) REFERENCES Reviews(ReviewId) ON DELETE CASCADE,
    FOREIGN KEY (VacationSpotName) REFERENCES VacationSpots(VacationSpotName) ON
        DELETE CASCADE
);
```

Query Results (Located in db/queries.sql)

Query #1 - Get recent reviews from user's following and from the top 20 followed users

```
mysql> SELECT ReviewID, Username, ReviewText, ReviewRating, CreatedAt, LikeCount
    -> FROM (
    ->     SELECT r.ReviewID, r.Username, r.ReviewText, r.ReviewRating, r.CreatedAt, r.LikeCount
    ->     FROM Reviews r
    ->     JOIN Follows f ON r.Username = f.followeeUsername
    ->     WHERE f.followerUsername = 'nancy57'
    ->
    ->     UNION
    ->
    ->     SELECT r.ReviewID, r.Username, r.ReviewText, r.ReviewRating, r.CreatedAt, r.LikeCount
    ->     FROM Reviews r
    ->     JOIN (
    ->         SELECT f.followeeUsername
    ->         FROM Follows f
    ->         GROUP BY f.followeeUsername
    ->         ORDER BY COUNT(DISTINCT f.followerUsername) DESC
    ->         LIMIT 20
    ->     ) AS TopUsers
    ->     ON r.Username = TopUsers.followeeUsername
    -> ) AS ReviewFeed
    -> ORDER BY CreatedAt DESC
    -> LIMIT 15;
+----------+-----------------+----------------------------------------------------------------------------------------------------------------------------------------------------+--------------+---------------------+-----------+
| ReviewID | Username        | ReviewText                                                                                                                                         | ReviewRating | CreatedAt           | LikeCount |
+----------+-----------------+----------------------------------------------------------------------------------------------------------------------------------------------------+--------------+---------------------+-----------+
|      565 | kimberly00      | His argue including value decision sometimes particularly. The destination was amazing. The relaxing was amazing. The ocean was amazing. The getaway was amazing. |            5 | 2025-02-04 10:15:36 |         5 |
|      977 | barbararodriguez | Power network indicate subject our head born suddenly. The mountain was amazing. The destination was amazing. The tranquil was amazing. The wildlife was amazing. |            4 | 2024-11-26 06:04:20 |       193 |
|      638 | kimberly00      | Know concern industry join. The sunset was amazing. The resort was amazing. The beach was amazing. The escape was amazing.                          |            1 | 2024-11-10 08:08:18 |       114 |
|      438 | thomas28        | Fly class outside among PM forget. The luxury was amazing. The tranquil was amazing. The beach was amazing.                                         |            1 | 2024-10-10 17:25:27 |       864 |
|      713 | amanda32        | Poor environment job. The sunset was amazing. The scenic was amazing. The wildlife was amazing. The escape was amazing.                             |            4 | 2024-08-23 05:56:21 |       387 |
|      726 | armstrongsara   | Involve instead just ball. The cabin was amazing. The luxury was amazing. The destination was amazing.                                              |            2 | 2024-08-23 02:51:23 |       447 |
|      637 | amanda32        | Direction none bad type member explain. The luxury was amazing. The wildlife was amazing. The pool was amazing. The scenic was amazing.             |            2 | 2024-08-16 00:49:47 |        39 |
|      527 | john30          | Area PM someone assume social war generation. The relaxing was amazing. The spa was amazing. The island was amazing. The resort was amazing.        |            3 | 2024-08-01 07:25:56 |       860 |
|      265 | thomas28        | Condition manage admit maintain bit. The adventure was amazing. The island was amazing. The tranquil was amazing.                                   |            3 | 2024-06-30 20:30:18 |       126 |
|      758 | ramseyvalerie   | Agency teacher shake individual executive born investment. The escape was amazing. The relaxing was amazing. The luxury was amazing.                |            2 | 2024-05-23 09:53:06 |       956 |
|      853 | kimberly00      | Measure minute address writer you to. The paradise was amazing. The tranquil was amazing.                                                           |            1 | 2024-05-03 18:31:47 |       815 |
|      738 | michelle96      | Vote tree oil understand policy can bag try. The destination was amazing. The pool was amazing.                                                     |            1 | 2024-03-16 07:24:55 |       811 |
|      283 | ramseyvalerie   | Coach maybe energy million build charge arrive loss. The spa was amazing. The beach was amazing. The ocean was amazing. The paradise was amazing.   |            1 | 2024-02-16 14:41:54 |       525 |
|      149 | abbottchristopher | With war none company recent city large. The cabin was amazing. The island was amazing. The paradise was amazing.                                 |            1 | 2024-02-06 21:47:10 |        46 |
|      600 | barbararodriguez | Smile front test southern middle reach late. The island was amazing. The resort was amazing. The mountain was amazing.                            |            5 | 2024-02-02 09:49:14 |       500 |
+----------+-----------------+----------------------------------------------------------------------------------------------------------------------------------------------------+--------------+---------------------+-----------+
15 rows in set (0.05 sec)
```

Query #2: Top 3 reviews for a given vacation spot that have a like count greater than or equal to the average like count of all reviews for that vacation spot. There are not 15 records because we only want to display the top 3

```
Database changed
mysql> SELECT
    ->     r.ReviewID,
    ->     r.Username,
    ->     r.ReviewText,
    ->     r.ReviewRating,
    ->     r.CreatedAt,
    ->     r.LikeCount,
    ->     vsr.VacationSpotName,
    ->     i.ImageURL
    -> FROM Reviews r
    -> JOIN VacationSpotReviews vsr ON r.ReviewID = vsr.ReviewID
    -> LEFT JOIN Images i ON r.ReviewID = i.ReviewID
    -> WHERE vsr.VacationSpotName = 'Santana-Flowers Resort'
    -> AND r.LikeCount >= (
    ->     SELECT AVG(LikeCount) FROM Reviews
    ->     WHERE ReviewID IN (
    ->         SELECT ReviewID FROM VacationSpotReviews
    ->         WHERE VacationSpotName = 'Santana-Flowers Resort'
    ->     )
    -> )
    -> ORDER BY r.LikeCount DESC, r.CreatedAt DESC
    -> LIMIT 3;
+----------+----------+--------------------------------------------------------------------------------------------------------------------+--------------+---------------------+-----------+------------------------+----------+
| ReviewID | Username | ReviewText                                                                                                         | ReviewRating | CreatedAt           | LikeCount | VacationSpotName       | ImageURL |
+----------+----------+--------------------------------------------------------------------------------------------------------------------+--------------+---------------------+-----------+------------------------+----------+
|      305 | xcox     | Source against wear such leg discover catch. The resort was amazing. The paradise was amazing. The island was amazing. |            3 | 2024-08-03 11:06:08 |       997 | Santana-Flowers Resort | NULL     |
|      137 | rwilcox  | Late six national into address heart budget. The wildlife was amazing. The ocean was amazing. The mountain was amazing. |            5 | 2024-08-22 07:54:40 |       994 | Santana-Flowers Resort | NULL     |
|      722 | vdavis   | Second even agency south account form his. The cabin was amazing. The ocean was amazing. The beach was amazing.     |            4 | 2024-12-14 11:30:05 |       990 | Santana-Flowers Resort | NULL     |
+----------+----------+--------------------------------------------------------------------------------------------------------------------+--------------+---------------------+-----------+------------------------+----------+
```

Query #3: Find the most relevant vacation spots based on its popularity determined by number of reviews and average rating

-- 3. Top City Vacation Spots
-- When viewing city, find the most relevant vacation spots (including images) based on its popularity determined by number of reviews and average rating

SELECT vs.VacationSpotName, c.city_ascii, COUNT(r.ReviewID) AS TotalReviews, AVG(r.ReviewRating) AS AverageRating
FROM VacationSpots vs
    JOIN WorldCities c ON vs.CityId = c.id
    JOIN VacationSpotReviews vr ON vs.VacationSpotName = vr.VacationSpotName
    JOIN Reviews r ON vr.ReviewId = r.ReviewID
    LEFT JOIN Images i ON r.ReviewID = i.ReviewID

```
+--------------------------+------------+--------------+--------------+
| VacationSpotName         | city_ascii | TotalReviews | AverageRating |
+--------------------------+------------+--------------+--------------+
| Pacaya Volcano           | Jalalabad  |      1345 |       2.1375 |
| Antigua Guatemala        | Jalalabad  |       784 |       3.5689 |
| Tikal National Park      | Jalalabad  |       661 |       4.5673 |
| Lake Atitlán             | Jalalabad  |       554 |       4.1552 |
| Semuc Champey            | Jalalabad  |       275 |       2.4073 |
| Santana-Flowers Resort   | Jalalabad  |         5 |       3.8000 |
+--------------------------+------------+--------------+--------------+
```

Query #4: Gets users favorite vacation spots that are also popular vacation spots (have greater than the average amount of likes of vacation spots)

```
mysql> SELECT fs.Username, fs.VacationSpotName, c.city, v.LikeCount
    -> FROM FavoriteSpots fs
    -> JOIN VacationSpots v ON fs.VacationSpotName = v.VacationSpotName
    -> JOIN WorldCities c ON v.CityId = c.id
    -> WHERE fs.Username = 'aaronjones'
    ->
    -> INTERSECT
    ->
    -> SELECT fs.Username, fs.VacationSpotName, c.city, v.LikeCount
    -> FROM FavoriteSpots fs
    -> JOIN VacationSpots v ON fs.VacationSpotName = v.VacationSpotName
    -> JOIN WorldCities c ON v.CityId = c.id
    -> WHERE v.LikeCount >= (
    ->     SELECT AVG(LikeCount) FROM VacationSpots
    -> )
    -> ORDER BY LikeCount DESC
    -> LIMIT 15;
```

Username	VacationSpotName	city	LikeCount
aaronjones	Walls-Hayden Resort	Ierápetra	4979
aaronjones	Fowler-Arias Resort	Mirbāţ	4974
aaronjones	Hill Inc Resort	Oer-Erkenschwick	4968
aaronjones	Long LLC Resort	Falköping	4967
aaronjones	Manning, Marshall and Stevens Resort	Hamilton	4954
aaronjones	Duran Inc Resort	Malatya	4949
aaronjones	Gibson, Wilson and Wagner Resort	Argayash	4946
aaronjones	Acosta-Phillips Resort	Mohale's Hoek	4922
aaronjones	Schroeder Inc Resort	Majayjay	4902
aaronjones	Ellis-Valdez Resort	Two Rivers	4900
aaronjones	Turner, Ramirez and Harris Resort	Ban Si Don Chai	4897
aaronjones	Howell Group Resort	Gurh	4890
aaronjones	Anderson Group Resort	Phillipsburg	4889
aaronjones	Garcia, Torres and Rosario Resort	Anan	4882
aaronjones	Mcclain Group Resort	Tangcun	4860

15 rows in set (0.02 sec)

Column Counts

```
mysql> SELECT table_name, table_rows
    -> FROM information_schema.tables
    -> WHERE table_schema = DATABASE();
```

TABLE_NAME	TABLE_ROWS
CostOfLiving	5186
FavoriteSpots	546
Follows	11931
Images	83599
Reviews	1001
Temperatures	9690
UserAccounts	1001
VacationSpotReviews	20
VacationSpots	532
WorldCities	23424

10 rows in set (0.15 sec)

Indexing

-- Query 1

Before Indexing

```
| -> Limit: 15 row(s)  (cost=24.1..24.1 rows=7.97) (actual time=24.9..25 rows=15 loops=1)
    -> Sort: ReviewFeed.CreatedAt DESC, limit input to 15 row(s) per chunk  (cost=24.1..24.1 rows=7.97) (actual time=24.9..25 rows=15 loops=1)
        -> Table scan on ReviewFeed  (cost=18.7..20.9 rows=7.97) (actual time=24.9..24.9 rows=24 loops=1)
            -> Union materialize with deduplication  (cost=18.3..18.3 rows=7.97) (actual time=24.9..24.9 rows=24 loops=1)
                -> Nested loop inner join  (cost=5.07 rows=7.97) (actual time=0.182..0.219 rows=5 loops=1)
                    -> Covering index lookup on f using idx_follows_follower (followerUsername='nancy57')  (cost=1.79 rows=5) (actual time=0.0471..0.0489 rows=5 loops=1)
                    -> Index lookup on r using idx_reviews_username (Username=f.followeeUsername)  (cost=0.53 rows=1.59) (actual time=0.0318..0.0327 rows=1 loops=5)
                -> Nested loop inner join  (cost=12.5 rows=0) (actual time=24.4..24.6 rows=19 loops=1)
                    -> Table scan on TopUsers  (cost=2.5..2.5 rows=0) (actual time=24.4..24.4 rows=20 loops=1)
                        -> Materialize  (cost=0..0 rows=0) (actual time=24.4..24.4 rows=20 loops=1)
                            -> Limit: 20 row(s)  (actual time=24.4..24.4 rows=20 loops=1)
                                -> Sort: `count(distinct f.followerUsername)` DESC, limit input to 20 row(s) per chunk  (actual time=24.4..24.4 rows=20 loops=1)
                                    -> Stream results  (cost=2386 rows=1000) (actual time=0.881..24.2 rows=1000 loops=1)
                                        -> Group aggregate: count(distinct f.followerUsername)  (cost=2386 rows=1000) (actual time=0.878..24 rows=1000 loops=1)
                                            -> Covering index skip scan for deduplication on f using idx_follows_followee  (cost=1193 rows=11932) (actual time=0.0849..16.9 rows=12538 loops=1)
                    -> Index lookup on r using idx_reviews_username (Username=TopUsers.followeeUsername)  (cost=0.506 rows=1.59) (actual time=0.0086..0.00909 rows=0.95 loops=20)
|
```

Cost = 24.1

Design 1

CREATE INDEX rating ON Reviews(ReviewRating);

```
| -> Limit: 15 row(s)  (cost=21.6..21.6 rows=7.97) (actual time=22.5..22.5 rows=15 loops=1)
    -> Sort: ReviewFeed.CreatedAt DESC, limit input to 15 row(s) per chunk  (cost=21.6..21.6 rows=7.97) (actual time=22.5..22.5 rows=15 loops=1)
        -> Table scan on ReviewFeed  (cost=16.2..18.4 rows=7.97) (actual time=22.5..22.5 rows=24 loops=1)
            -> Union materialize with deduplication  (cost=15.8..15.8 rows=7.97) (actual time=22.5..22.5 rows=24 loops=1)
                -> Nested loop inner join  (cost=4.58 rows=7.97) (actual time=0.0445..0.093 rows=5 loops=1)
                    -> Covering index lookup on f using idx_follows_follower (followerUsername='nancy57')  (cost=1.79 rows=5) (actual time=0.0206..0.0236 rows=5 loops=1)
                    -> Index lookup on r using idx_reviews_username (Username=f.followeeUsername)  (cost=0.43 rows=1.59) (actual time=0.0125..0.0134 rows=1 loops=5)
                -> Nested loop inner join  (cost=10.5 rows=0) (actual time=22.2..22.3 rows=19 loops=1)
                    -> Table scan on TopUsers  (cost=2.5..2.5 rows=0) (actual time=22.1..22.1 rows=20 loops=1)
                        -> Materialize  (cost=0..0 rows=0) (actual time=22.1..22.1 rows=20 loops=1)
                            -> Limit: 20 row(s)  (actual time=22.1..22.1 rows=20 loops=1)
                                -> Sort: `count(distinct f.followerUsername)` DESC, limit input to 20 row(s) per chunk  (actual time=22.1..22.1 rows=20 loops=1)
                                    -> Stream results  (cost=2386 rows=1000) (actual time=0.0955..21.9 rows=1000 loops=1)
                                        -> Group aggregate: count(distinct f.followerUsername)  (cost=2386 rows=1000) (actual time=0.0936..21.7 rows=1000 loops=1)
                                            -> Covering index skip scan for deduplication on f using idx_follows_followee  (cost=1193 rows=11932) (actual time=0.0172..15.3 rows=12538 loops=1)
                    -> Index lookup on r using idx_reviews_username (Username=TopUsers.followeeUsername)  (cost=0.406 rows=1.59) (actual time=0.00865..0.00943 rows=0.95 loops=20)
|
```

Cost = 21.6

Design 2

CREATE INDEX rating ON Reviews(ReviewRating);
CREATE INDEX followerUsername ON Follower(FollowerUsername);

```
| -> Limit: 15 row(s)  (cost=20.7..20.7 rows=7.97) (actual time=21.6..21.6 rows=15 loops=1)
    -> Sort: ReviewFeed.CreatedAt DESC, limit input to 15 row(s) per chunk  (cost=20.7..20.7 rows=7.97) (actual time=21.6..21.6 rows=15 loops=1)
        -> Table scan on ReviewFeed  (cost=15.2..17.5 rows=7.97) (actual time=21.6..21.6 rows=24 loops=1)
            -> Union materialize with deduplication  (cost=14.9..14.9 rows=7.97) (actual time=21.6..21.6 rows=24 loops=1)
                -> Nested loop inner join  (cost=3.63 rows=7.97) (actual time=0.0517..0.0795 rows=5 loops=1)
                    -> Covering index lookup on f using PRIMARY (followerUsername='nancy57')  (cost=0.841 rows=5) (actual time=0.0222..0.0239 rows=5 loops=1)
                    -> Index lookup on r using idx_reviews_username (Username=f.followeeUsername)  (cost=0.43 rows=1.59) (actual time=0.00975..0.0105 rows=1 loops=5)
                -> Nested loop inner join  (cost=10.5 rows=0) (actual time=21.3..21.4 rows=19 loops=1)
                    -> Table scan on TopUsers  (cost=2.5..2.5 rows=0) (actual time=21.3..21.3 rows=20 loops=1)
                        -> Materialize  (cost=0..0 rows=0) (actual time=21.3..21.3 rows=20 loops=1)
                            -> Limit: 20 row(s)  (actual time=21.2..21.2 rows=20 loops=1)
                                -> Sort: `count(distinct f.followerUsername)` DESC, limit input to 20 row(s) per chunk  (actual time=21.2..21.2 rows=20 loops=1)
                                    -> Stream results  (cost=2386 rows=1000) (actual time=0.0561..21.1 rows=1000 loops=1)
                                        -> Group aggregate: count(distinct f.followerUsername)  (cost=2386 rows=1000) (actual time=0.0548..20.9 rows=1000 loops=1)
                                            -> Covering index skip scan for deduplication on f using idx_follows_followee  (cost=1193 rows=11932) (actual time=0.0119..14.7 rows=12538 loops=1)
                    -> Index lookup on r using idx_reviews_username (Username=TopUsers.followeeUsername)  (cost=0.406 rows=1.59) (actual time=0.00694..0.00741 rows=0.95 loops=20)
|
```

Cost = 15.2

Design 3
CREATE INDEX rating ON Reviews(ReviewRating);
CREATE INDEX followerUsername ON Follower(FollowerUsername);
CREATE INDEX followeeUsername ON FavoriteSpots(Username);

```
| -> Limit: 15 row(s)  (cost=20.7..20.7 rows=7.97) (actual time=21.9..21.9 rows=15 loops=1)
    -> Sort: ReviewFeed.CreatedAt DESC, limit input to 15 row(s) per chunk  (cost=20.7..20.7 rows=7.97) (actual time=21.9..21.9 rows=15 loops=1)
        -> Table scan on ReviewFeed  (cost=15.2..17.5 rows=7.97) (actual time=21.9..21.9 rows=24 loops=1)
            -> Union materialize with deduplication  (cost=14.9..14.9 rows=7.97) (actual time=21.9..21.9 rows=24 loops=1)
                -> Nested loop inner join  (cost=3.63 rows=7.97) (actual time=0.0463..0.0732 rows=5 loops=1)
                    -> Covering index lookup on f using PRIMARY (followerUsername='nancy57')  (cost=0.841 rows=5) (actual time=0.0198..0.0214 rows=5 loops=1)
                    -> Index lookup on r using idx_reviews_username (Username=f.followeeUsername)  (cost=0.43 rows=1.59) (actual time=0.00901..0.00979 rows=1 loops=5)
                -> Nested loop inner join  (cost=10.5 rows=0) (actual time=21.6..21.7 rows=19 loops=1)
                    -> Table scan on TopUsers  (cost=2.5..2.5 rows=0) (actual time=21.6..21.6 rows=20 loops=1)
                        -> Materialize  (cost=0..0 rows=0) (actual time=21.6..21.6 rows=20 loops=1)
                            -> Limit: 20 row(s)  (actual time=21.6..21.6 rows=20 loops=1)
                                -> Sort: `count(distinct f.followerUsername)` DESC, limit input to 20 row(s) per chunk  (actual time=21.6..21.6 rows=20 loops=1)
                                    -> Stream results  (cost=2386 rows=1000) (actual time=0.0524..21.4 rows=1000 loops=1)
                                        -> Group aggregate: count(distinct f.followerUsername)  (cost=2386 rows=1000) (actual time=0.0511..21.2 rows=1000 loops=1)
                                            -> Covering index skip scan for deduplication on f using idx_follows_followee  (cost=1193 rows=11932) (actual time=0.0113..14.9 rows=12538 loops=1)
                    -> Index lookup on r using idx_reviews_username (Username=TopUsers.followeeUsername)  (cost=0.406 rows=1.59) (actual time=0.00608..0.00654 rows=0.95 loops=20)
|
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Cost = 15.2

Design 3 was the best because it optimizes the two most important aspects of the query: filtering by followerUsername in Follows and joining Reviews by Username. The indexes reduce the number of rows scanned, significantly improving the efficiency of both the JOIN operations and the filtering process, leading to the lowest query cost.

-- Query 2
Without indexing:

```
-> Limit: 3 row(s)  (actual time=0.394..0.395 rows=3 loops=1)
    -> Sort: r.LikeCount DESC, r.CreatedAt DESC, limit input to 3 row(s) per chunk  (actual time=0.393..0.394 rows=3 loops=1)
        -> Stream results  (cost=230 rows=1407) (actual time=0.353..0.373 rows=3 loops=1)
            -> Nested loop left join  (cost=230 rows=1407) (actual time=0.345..0.361 rows=3 loops=1)
                -> Nested loop inner join  (cost=2.55 rows=1.67) (actual time=0.324..0.335 rows=3 loops=1)
                    -> Covering index lookup on vsr using VacationSpotName (VacationSpotName='Santana-Flowers Resort')  (cost=0.798 rows=5) (actual time=0.117..0.12 rows=5 loops=1)
                    -> Filter: (r.LikeCount >= (select #2))  (cost=0.257 rows=0.333) (actual time=0.0415..0.0415 rows=0.6 loops=5)
                        -> Single-row index lookup on r using PRIMARY (ReviewID=vsr.ReviewId)  (cost=0.257 rows=1) (actual time=0.0164..0.0164 rows=1 loops=5)
                        -> Select #2 (subquery in condition; run only once)
                            -> Aggregate: avg(Reviews.LikeCount)  (cost=3.05 rows=1) (actual time=0.0878..0.0879 rows=1 loops=1)
                                -> Nested loop inner join  (cost=2.55 rows=5) (actual time=0.065..0.0814 rows=5 loops=1)
                                    -> Covering index lookup on VacationSpotReviews using VacationSpotName (VacationSpotName='Santana-Flowers Resort')  (cost=0.798 rows=5) (actual time=0.0384..0.0424 rows=5 loops=1)
                                    -> Single-row index lookup on Reviews using PRIMARY (ReviewID=VacationSpotReviews.ReviewId)  (cost=0.27 rows=1) (actual time=0.00742..0.00747 rows=1 loops=5)
                -> Covering index lookup on i using ReviewID (ReviewID=vsr.ReviewId)  (cost=102 rows=844) (actual time=0.00804..0.00804 rows=0 loops=3)
|
```

Cost is 230

Design 1:
CREATE INDEX idx_vsr_spotname_reviewid ON
VacationSpotReviews(VacationSpotName, ReviewID);

```
-> Limit: 3 row(s)  (actual time=0.211..0.212 rows=3 loops=1)
    -> Sort: r.LikeCount DESC, r.CreatedAt DESC, limit input to 3 row(s) per chunk  (actual time=0.21..0.211 rows=3 loops=1)
        -> Stream results  (cost=78.6 rows=337) (actual time=0.148..0.191 rows=3 loops=1)
            -> Nested loop left join  (cost=78.6 rows=337) (actual time=0.138..0.173 rows=3 loops=1)
                -> Nested loop inner join  (cost=2.55 rows=0.25) (actual time=0.107..0.136 rows=3 loops=1)
                    -> Covering index lookup on vsr using idx_vsr_spotname_reviewid (VacationSpotName='Santana-Flowers Resort')  (cost=0.8 rows=5) (actual time=0.0847..0.0893 rows=5 loops=1)
                    -> Filter: (r.LikeCount >= (select #2))  (cost=0.251 rows=0.05) (actual time=0.00809..0.00817 rows=0.6 loops=5)
                        -> Single-row index lookup on r using PRIMARY (ReviewID=vsr.ReviewId)  (cost=0.251 rows=1) (actual time=0.00557..0.00562 rows=1 loops=5)
                        -> Select #2 (subquery in condition; run only once)
                            -> Aggregate: avg(Reviews.LikeCount)  (cost=3.05 rows=1) (actual time=0.0753..0.0755 rows=1 loops=1)
                                -> Nested loop inner join  (cost=2.55 rows=5) (actual time=0.0504..0.0662 rows=5 loops=1)
                                    -> Covering index lookup on VacationSpotReviews using idx_vsr_spotname_reviewid (VacationSpotName='Santana-Flowers Resort')  (cost=0.8 rows=5) (actual time=0.0331..0.0375 rows=5 loops=1)
                                    -> Single-row index lookup on Reviews using PRIMARY (ReviewID=VacationSpotReviews.ReviewId)  (cost=0.27 rows=1) (actual time=0.00519..0.00523 rows=1 loops=5)
                -> Covering index lookup on i using idx_images_reviewid (ReviewID=vsr.ReviewId)  (cost=709 rows=1348) (actual time=0.0113..0.0113 rows=0 loops=3)
|
```

Cost is 78.6

Design 2:
CREATE INDEX idx_reviews_likecount_createdat ON Reviews (LikeCount, CreatedAt);

```
-> Limit: 3 row(s)  (actual time=0.128..0.129 rows=3 loops=1)
    -> Sort: r.LikeCount DESC, r.CreatedAt DESC, limit input to 3 row(s) per chunk  (actual time=0.128..0.128 rows=3 loops=1)
        -> Stream results  (cost=510 rows=2247)  (actual time=0.0868..0.104 rows=3 loops=1)
            -> Nested loop left join  (cost=510 rows=2247)  (actual time=0.0808..0.0957 rows=3 loops=1)
                -> Nested loop inner join  (cost=2.86 rows=1.67)  (actual time=0.0683..0.0774 rows=3 loops=1)
                    -> Covering index lookup on vsr using idx_vsr_spotname_reviewid (VacationSpotName='Santana-Flowers Resort')  (cost=0.8 rows=5)  (actual time=0.013..0.0144 rows=5 loops=1)
                    -> Filter: (r.LikeCount >= (select #2))  (cost=0.319 rows=0.333)  (actual time=0.0119..0.012 rows=0.6 loops=5)
                        -> Single-row index lookup on r using PRIMARY (ReviewID=vsr.ReviewId)  (cost=0.00295..0.00298 rows=1 loops=5)
                        -> Select #2 (subquery in condition; run only once)
                            -> Aggregate: avg(Reviews.LikeCount)  (cost=3.36 rows=1)  (actual time=0.0281..0.0282 rows=1 loops=1)
                                -> Nested loop inner join  (cost=2.86 rows=5)  (actual time=0.00958..0.0234 rows=5 loops=1)
                                    -> Covering index lookup on VacationSpotReviews using idx_vsr_spotname_reviewid (VacationSpotName='Santana-Flowers Resort')  (cost=0.8 rows=5)  (actual time=0.00665..0.00924 rows=5 loops=1)
                                    -> Single-row index lookup on Reviews using PRIMARY (ReviewID=VacationSpotReviews.ReviewId)  (cost=0.333 rows=1)  (actual time=0.00252..0.00256 rows=1 loops=5)
                -> Covering index lookup on i using idx_images_reviewid (ReviewID=vsr.ReviewId)  (cost=250 rows=1348)  (actual time=0.00548..0.00548 rows=0 loops=3)
```

Cost is 510

Design 3:
CREATE INDEX idx_vsr_spotname_reviewid ON VacationSpotReviews(VacationSpotName, ReviewID);
CREATE INDEX idx_images_reviewid ON Images (ReviewID);

```
-> Limit: 3 row(s)  (actual time=0.0799..0.0805 rows=3 loops=1)
    -> Sort: r.LikeCount DESC, r.CreatedAt DESC, limit input to 3 row(s) per chunk  (actual time=0.0784..0.0789 rows=3 loops=1)
        -> Stream results  (cost=40.3 rows=211)  (actual time=0.0422..0.0624 rows=3 loops=1)
            -> Nested loop left join  (cost=40.3 rows=211)  (actual time=0.0366..0.0542 rows=3 loops=1)
                -> Nested loop inner join  (cost=2.55 rows=0.25)  (actual time=0.0248..0.037 rows=3 loops=1)
                    -> Covering index lookup on vsr using idx_vsr_spotname_reviewid (VacationSpotName='Santana-Flowers Resort')  (cost=0.8 rows=5)  (actual time=0.0144..0.0167 rows=5 loops=1)
                    -> Filter: (r.LikeCount >= (select #2))  (cost=0.251 rows=0.05)  (actual time=0.00362..0.00369 rows=0.6 loops=5)
                        -> Single-row index lookup on r using PRIMARY (ReviewID=vsr.ReviewId)  (cost=0.251 rows=1)  (actual time=0.00294..0.00298 rows=1 loops=5)
                        -> Select #2 (subquery in condition; run only once)
                            -> Aggregate: avg(Reviews.LikeCount)  (cost=3.05 rows=1)  (actual time=0.0563..0.0564 rows=1 loops=1)
                                -> Nested loop inner join  (cost=2.55 rows=5)  (actual time=0.0361..0.0508 rows=5 loops=1)
                                    -> Covering index lookup on VacationSpotReviews using idx_vsr_spotname_reviewid (VacationSpotName='Santana-Flowers Resort')  (cost=0.8 rows=5)  (actual time=0.0225..0.026 rows=5 loops=1)
                                    -> Single-row index lookup on Reviews using PRIMARY (ReviewID=VacationSpotReviews.ReviewId)  (cost=0.27 rows=1)  (actual time=0.00444..0.00448 rows=1 loops=5)
                -> Covering index lookup on i using ReviewID (ReviewID=vsr.ReviewId)  (cost=404 rows=844)  (actual time=0.0053..0.0053 rows=0 loops=3)
```

Cost is 40.3

Design 3 was the best because it optimizes the two most important parts of the query: filtering by VacationSpotName in VacationSpotReviews and joining Images by ReviewID. The indexes reduce the number of rows the database has to scan, improving the performance of both the join operations and the filtering process, which results in the lowest cost.

-- Query 3
Without indexing

```
-----------------------------+
| -> Sort: TotalReviews DESC, AverageRating DESC  (actual time=5.11..5.11 rows=6 loops=1)
    -> Table scan on <temporary>  (actual time=5.07..5.08 rows=6 loops=1)
        -> Aggregate using temporary table  (actual time=5.07..5.07 rows=6 loops=1)
            -> Nested loop left join  (cost=325 rows=1689)  (actual time=0.154..1.6 rows=3624 loops=1)
                -> Nested loop inner join  (cost=23.2 rows=2)  (actual time=0.0947..0.207 rows=20 loops=1)
                    -> Nested loop inner join  (cost=16.2 rows=20)  (actual time=0.0812..0.173 rows=20 loops=1)
                        -> Nested loop inner join  (cost=9.25 rows=20)  (actual time=0.0694..0.113 rows=20 loops=1)
                            -> Covering index scan on vr using PRIMARY  (cost=2.25 rows=20)  (actual time=0.0431..0.0519 rows=20 loops=1)
                            -> Single-row index lookup on vs using PRIMARY (VacationSpotName=vr.VacationSpotName)  (cost=0.255 rows=1)  (actual time=0.00266..0.0027 rows=1 loops=20)
                        -> Single-row index lookup on r using PRIMARY (ReviewID=vr.ReviewId)  (cost=0.255 rows=1)  (actual time=0.00275..0.00279 rows=1 loops=20)
                    -> Filter: (c.city_ascii = 'Jalalabad')  (cost=0.251 rows=0.1)  (actual time=0.00132..0.00144 rows=1 loops=20)
                        -> Single-row index lookup on c using PRIMARY (id=vs.CityId)  (cost=0.251 rows=1)  (actual time=681e-6..716e-6 rows=1 loops=20)
                -> Covering index lookup on i using ReviewID (ReviewID=vr.ReviewId)  (cost=109 rows=844)  (actual time=0.0165..0.0574 rows=181 loops=20)
|
```

Cost is 325

Design 1
CREATE INDEX CityId ON VacationSpots(CityId);

```
| -> Sort: TotalReviews DESC, AverageRating DESC  (actual time=105..105 rows=6 loops=1)
    -> Table scan on <temporary>  (actual time=104..104 rows=6 loops=1)
        -> Aggregate using temporary table  (actual time=104..104 rows=6 loops=1)
            -> Nested loop left join  (cost=636 rows=2697)  (actual time=34.6..99.9 rows=3624 loops=1)
                -> Nested loop inner join  (cost=27.9 rows=2)  (actual time=0.0626..1.67 rows=20 loops=1)
                    -> Nested loop inner join  (cost=16.2 rows=20)  (actual time=0.0552..1.58 rows=20 loops=1)
                        -> Nested loop inner join  (cost=9.25 rows=20)  (actual time=0.0466..1.43 rows=20 loops=1)
                            -> Covering index scan on vr using PRIMARY  (cost=2.25 rows=20)  (actual time=0.0301..0.0521 rows=20 loops=1)
                            -> Single-row index lookup on vs using PRIMARY (VacationSpotName=vr.VacationSpotName)  (cost=0.255 rows=1)  (actual time=0.0682..0.0683 rows=1 loops=20)
                        -> Single-row index lookup on r using PRIMARY (ReviewID=vr.ReviewId)  (cost=0.255 rows=1)  (actual time=0.00729..0.00734 rows=1 loops=20)
                    -> Filter: (c.city_ascii = 'Jalalabad')  (cost=0.481 rows=0.1)  (actual time=0.00241..0.00368 rows=1 loops=20)
                        -> Single-row index lookup on c using PRIMARY (id=vs.CityId)  (cost=0.481 rows=1)  (actual time=825e-6..904e-6 rows=1 loops=20)
                -> Covering index lookup on i using idx_images_reviewid (ReviewID=vr.ReviewId)  (cost=237 rows=1348)  (actual time=1.75..4.9 rows=181 loops=20)
|
```

Cost is 636

Design 2

CREATE INDEX idx_vsr_reviewid_q3 ON VacationSpotReviews(ReviewID);

```
| -> Sort: TotalReviews DESC, AverageRating DESC  (actual time=4.38..4.38 rows=6 loops=1)
   -> Table scan on <temporary>  (actual time=4.34..4.34 rows=6 loops=1)
      -> Aggregate using temporary table  (actual time=4.34..4.34 rows=6 loops=1)
         -> Nested loop left join  (cost=636 rows=2697) (actual time=0.128..1.46 rows=3624 loops=1)
            -> Nested loop inner join  (cost=27.9 rows=2) (actual time=0.0703..0.162 rows=20 loops=1)
               -> Nested loop inner join  (cost=16.2 rows=20) (actual time=0.062..0.137 rows=20 loops=1)
                  -> Nested loop inner join  (cost=9.25 rows=20) (actual time=0.0528..0.0857 rows=20 loops=1)
                     -> Covering index scan on vr using idx_vsr_reviewid_q3  (cost=2.25 rows=20) (actual time=0.0308..0.0381 rows=20 loops=1)
                     -> Single-row index lookup on vs using PRIMARY (VacationSpotName=vr.VacationSpotName)  (cost=0.255 rows=1) (actual time=0.00206..0.00209 rows=1 lo
ops=20)
                  -> Single-row index lookup on r using PRIMARY (ReviewID=vr.ReviewId)  (cost=0.255 rows=1) (actual time=0.00232..0.00235 rows=1 loops=20)
               -> Filter: (c.city_ascii = 'Jalalabad')  (cost=0.481 rows=0.1) (actual time=940e-6..0.00104 rows=1 loops=20)
                  -> Single-row index lookup on c using PRIMARY (id=vs.CityId)  (cost=0.481 rows=1) (actual time=468e-6..501e-6 rows=1 loops=20)
            -> Covering index lookup on i using idx_images_reviewid (ReviewID=vr.ReviewId)  (cost=237 rows=1348) (actual time=0.0155..0.0533 rows=181 loops=20)
```

Cost is 636

Design 3
CREATE INDEX asciiindex ON WorldCities(city_ascii)

```
| -> Sort: TotalReviews DESC, AverageRating DESC  (actual time=196..196 rows=6 loops=1)
   -> Table scan on <temporary>  (actual time=195..195 rows=6 loops=1)
      -> Aggregate using temporary table  (actual time=195..195 rows=6 loops=1)
         -> Nested loop left join  (cost=323 rows=1348) (actual time=107..190 rows=3624 loops=1)
            -> Nested loop inner join  (cost=18.7 rows=1) (actual time=73.1..74.7 rows=20 loops=1)
               -> Nested loop inner join  (cost=18.3 rows=1) (actual time=48.6..50 rows=20 loops=1)
                  -> Nested loop inner join  (cost=10 rows=20) (actual time=48.6..49.9 rows=20 loops=1)
                     -> Covering index scan on vr using idx_vsr_reviewid_q3  (cost=3 rows=20) (actual time=47.6..47.6 rows=20 loops=1)
                     -> Single-row index lookup on vs using PRIMARY (VacationSpotName=vr.VacationSpotName)  (cost=0.255 rows=1) (actual time=0.0655..0.0655 row
s=1 loops=20)
                  -> Filter: (c.city_ascii = 'Jalalabad')  (cost=0.313 rows=0.05) (actual time=0.00366..0.00392 rows=1 loops=20)
                     -> Single-row index lookup on c using PRIMARY (id=vs.CityId)  (cost=0.313 rows=1) (actual time=0.00153..0.0016 rows=1 loops=20)
               -> Single-row index lookup on r using PRIMARY (ReviewID=vr.ReviewId)  (cost=0.412 rows=1) (actual time=1.23..1.23 rows=1 loops=20)
            -> Covering index lookup on i using idx_images_reviewid (ReviewID=vr.ReviewId)  (cost=304 rows=1348) (actual time=2.87..5.74 rows=181 loops=20)
```

Cost is 323

The design for this advanced query is design 3 since it decreases the cost by 2.It optimized the query so that when we were searching through worldcities it found faster searches.

-- Query 4
Before Indexing

```
| -> Limit: 15 row(s)  (cost=681..681 rows=15) (actual time=7.29..7.29 rows=15 loops=1)
   -> Sort: LikeCount DESC, limit input to 15 row(s) per chunk  (cost=681..681 rows=15) (actual time=7.29..7.29 rows=15 loops=1)
      -> Table scan on <intersect temporary>  (cost=652.657 rows=182) (actual time=7.16..7.22 rows=262 loops=1)
         -> Intersect materialize with deduplication  (cost=652..652 rows=182) (actual time=7.15..7.15 rows=532 loops=1)
            -> Nested loop inner join  (cost=475 rows=532) (actual time=0.0673..3.63 rows=532 loops=1)
               -> Nested loop inner join  (cost=240 rows=532) (actual time=0.0577..1.5 rows=532 loops=1)
                  -> Table scan on v  (cost=54 rows=532) (actual time=0.046..0.232 rows=532 loops=1)
                  -> Single-row covering index lookup on fs using PRIMARY (Username='aaronjones', VacationSpotName=v.VacationSpotName)  (cost=0.25 rows=1) (actual time=0.00215..0.00218 rows=1 loops=532)
               -> Single-row index lookup on c using PRIMARY (id=v.CityId)  (cost=0.341 rows=1) (actual time=0.00379..0.00382 rows=1 loops=532)
            -> Nested loop inner join  (cost=159 rows=182) (actual time=0.3..2.13 rows=268 loops=1)
               -> Nested loop inner join  (cost=96.6 rows=177) (actual time=0.283..1.14 rows=262 loops=1)
                  -> Filter: (v.LikeCount >= (select #3))  (cost=18.5 rows=177) (actual time=0.272..0.563 rows=262 loops=1)
                     -> Table scan on v  (cost=18.5 rows=532) (actual time=0.044..0.193 rows=532 loops=1)
                     -> Select #3 (subquery in condition; run only once)
                        -> Aggregate: avg(VacationSpots.LikeCount)  (cost=107 rows=1) (actual time=0.202..0.202 rows=1 loops=1)
                           -> Table scan on VacationSpots  (cost=54 rows=532) (actual time=0.0273..0.151 rows=532 loops=1)
                  -> Single-row index lookup on c using PRIMARY (id=v.CityId)  (cost=0.341 rows=1) (actual time=0.00198..0.002 rows=1 loops=262)
               -> Covering index lookup on fs using VacationSpotName (VacationSpotName=v.VacationSpotName)  (cost=0.251 rows=1.03) (actual time=0.00279..0.00357 rows=1.02 loops=262)
```

Cost = 681

Design 1
CREATE INDEX idx_fs_username ON FavoriteSpots(Username);

```
| -> Limit: 15 row(s)  (cost=780..780 rows=15) (actual time=314..314 rows=15 loops=1)
   -> Sort: LikeCount DESC, limit input to 15 row(s) per chunk  (cost=780..780 rows=15) (actual time=314..314 rows=15 loops=1)
      -> Table scan on <intersect temporary>  (cost=751..756 rows=182) (actual time=313..313 rows=262 loops=1)
         -> Intersect materialize with deduplication  (cost=751..751 rows=182) (actual time=313..313 rows=532 loops=1)
            -> Nested loop inner join  (cost=549 rows=532) (actual time=0.342..305 rows=532 loops=1)
               -> Nested loop inner join  (cost=240 rows=532) (actual time=0.153..2.41 rows=532 loops=1)
                  -> Table scan on v  (cost=54 rows=532) (actual time=0.108..0.481 rows=532 loops=1)
                  -> Single-row covering index lookup on fs using PRIMARY (Username='aaronjones', VacationSpotName=v.VacationSpotName)  (cost=0.25 rows=1) (actual time=0.00328..0.00338 rows=1 loops=532)
               -> Single-row index lookup on c using PRIMARY (id=v.CityId)  (cost=0.48 rows=1) (actual time=0.569..0.569 rows=1 loops=532)
            -> Nested loop inner join  (cost=184 rows=182) (actual time=0.259..2.67 rows=268 loops=1)
               -> Nested loop inner join  (cost=121 rows=177) (actual time=0.244..1.51 rows=262 loops=1)
                  -> Filter: (v.LikeCount >= (select #3))  (cost=18.5 rows=177) (actual time=0.236..0.513 rows=262 loops=1)
                     -> Table scan on v  (cost=18.5 rows=532) (actual time=0.0381..0.219 rows=532 loops=1)
                     -> Select #3 (subquery in condition; run only once)
                        -> Aggregate: avg(VacationSpots.LikeCount)  (cost=107 rows=1) (actual time=0.174..0.174 rows=1 loops=1)
                           -> Table scan on VacationSpots  (cost=54 rows=532) (actual time=0.0258..0.125 rows=532 loops=1)
                  -> Single-row index lookup on c using PRIMARY (id=v.CityId)  (cost=0.481 rows=1) (actual time=0.0035..0.00353 rows=1 loops=262)
               -> Covering index lookup on fs using VacationSpotName (VacationSpotName=v.VacationSpotName)  (cost=0.251 rows=1.03) (actual time=0.00352..0.00422 rows=1.02 loops=262)
```

Cost = 780

Design 2

CREATE INDEX idx_vs_likecount ON VacationSpots(LikeCount);

```
---------------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=802..802 rows=15) (actual time=8.03..8.04 rows=15 loops=1)
    -> Sort: LikeCount DESC, limit input to 15 row(s) per chunk  (cost=802..802 rows=15) (actual time=8.03..8.03 rows=15 loops=1)
        -> Table scan on <intersect temporary>  (cost=763..769 rows=269) (actual time=7.9..7.97 rows=262 loops=1)
            -> Intersect materialize with deduplication  (cost=763..763 rows=269) (actual time=7.9..7.9 rows=532 loops=1)
                -> Nested loop inner join  (cost=475 rows=532) (actual time=0.0723..4.34 rows=532 loops=1)
                    -> Nested loop inner join  (cost=240 rows=532) (actual time=0.0484..2.03 rows=532 loops=1)
                        -> Table scan on v  (cost=54 rows=532) (actual time=0.0352..0.308 rows=532 loops=1)
                        -> Single-row covering index lookup on fs using PRIMARY (Username='aaronjones', VacationSpotName=v.VacationSpotName)  (cost=0.25 rows=1) (actual time=0.00296..0.003 rows=1 loops=532)
                    -> Single-row index lookup on c using PRIMARY (id=v.CityId)  (cost=0.341 rows=1) (actual time=0.00408..0.00412 rows=1 loops=532)
                -> Nested loop inner join  (cost=262 rows=269) (actual time=0.0985..1.86 rows=268 loops=1)
                    -> Nested loop inner join  (cost=169 rows=262) (actual time=0.079..0.833 rows=262 loops=1)
                        -> Filter: (v.LikeCount >= (select #3))  (cost=54 rows=262) (actual time=0.0673..0.337 rows=262 loops=1)
                            -> Table scan on v  (cost=54 rows=532) (actual time=0.0583..0.209 rows=532 loops=1)
                            -> Select #3 (subquery in condition; run only once)
                                -> Aggregate: avg(VacationSpots.LikeCount)  (cost=107 rows=1) (actual time=0.241..0.241 rows=1 loops=1)
                                    -> Covering index scan on VacationSpots using likes  (cost=54 rows=532) (actual time=0.0326..0.18 rows=532 loops=1)
                        -> Single-row index lookup on c using PRIMARY (id=v.CityId)  (cost=0.341 rows=1) (actual time=0.00167..0.0017 rows=1 loops=262)
                    -> Covering index lookup on fs using VacationSpotName (VacationSpotName=v.VacationSpotName)  (cost=0.251 rows=1.03) (actual time=0.0029..0.0037 rows=1.02 loops=262)
|
+---------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

Cost = 802

Design 3
CREATE INDEX idx_vs_likecount ON VacationSpots(LikeCount);
CREATE INDEX idx_cities_cityid ON WorldCities(city_ascii);

```
---------------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=794..794 rows=15) (actual time=148..148 rows=15 loops=1)
    -> Sort: LikeCount DESC, limit input to 15 row(s) per chunk  (cost=794..794 rows=15) (actual time=148..148 rows=15 loops=1)
        -> Table scan on <intersect temporary>  (cost=755..761 rows=269) (actual time=148..148 rows=262 loops=1)
            -> Intersect materialize with deduplication  (cost=755..755 rows=269) (actual time=148..148 rows=532 loops=1)
                -> Nested loop inner join  (cost=469 rows=532) (actual time=0.0655..144 rows=532 loops=1)
                    -> Nested loop inner join  (cost=240 rows=532) (actual time=0.0575..1.8 rows=532 loops=1)
                        -> Table scan on v  (cost=54 rows=532) (actual time=0.0447..0.332 rows=532 loops=1)
                        -> Single-row covering index lookup on fs using PRIMARY (Username='aaronjones', VacationSpotName=v.VacationSpotName)  (cost=0.25 rows=1) (actual time=0.00251..0.00254 rows=1 loops=532)
                    -> Single-row index lookup on c using PRIMARY (id=v.CityId)  (cost=0.33 rows=1) (actual time=0.268..0.268 rows=1 loops=532)
                -> Nested loop inner join  (cost=259 rows=269) (actual time=0.0796..1.9 rows=268 loops=1)
                    -> Nested loop inner join  (cost=167 rows=262) (actual time=0.0644..0.945 rows=262 loops=1)
                        -> Filter: (v.LikeCount >= (select #3))  (cost=54 rows=262) (actual time=0.0561..0.301 rows=262 loops=1)
                            -> Table scan on v  (cost=54 rows=532) (actual time=0.048..0.199 rows=532 loops=1)
                            -> Select #3 (subquery in condition; run only once)
                                -> Aggregate: avg(VacationSpots.LikeCount)  (cost=107 rows=1) (actual time=0.245..0.245 rows=1 loops=1)
                                    -> Covering index scan on VacationSpots using likes  (cost=54 rows=532) (actual time=0.0779..0.178 rows=532 loops=1)
                        -> Single-row index lookup on c using PRIMARY (id=v.CityId)  (cost=0.331 rows=1) (actual time=0.00218..0.00221 rows=1 loops=262)
                    -> Covering index lookup on fs using VacationSpotName (VacationSpotName=v.VacationSpotName)  (cost=0.251 rows=1.03) (actual time=0.00278..0.00341 rows=1.02 loops=262)
|
+---------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

Cost = 794

The design for this advanced query is the default indexing, without adding create index. When we added indexes, it actually increased the cost so we feel the best design in this case would just be going with the default indexing. Likely since the default indexing was picked as the create index, we actually created duplicate indexes here which took up extra memory/writing space rather than helping.