

PT1 - Stage 3

Team020 IShowCode

Part 1

Database Implementation

We generate 5 tables: Park, User, Trail, Facility, Review and import the real world data for Park, Facility, Trail tables.

```
(pt1s3_env) zjy@zjydeMacBook-Pro-5 PT1S3 % mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 172
Server version: 9.5.0 Homebrew

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
<
mysql> show databases;
+-----+
| Database |
+-----+
| findmypark_nyc |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.001 sec)
```

User Table:

```
DROP TABLE IF EXISTS User;
CREATE TABLE User (
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
);
```

Park table:

```
DROP TABLE IF EXISTS Park;
```

```

CREATE TABLE Park (
    park_id VARCHAR(20) PRIMARY KEY,
    park_name VARCHAR(200) NOT NULL,
    park_size DECIMAL(10,2),

    borough VARCHAR(255),
    zipcode VARCHAR(10),
    park_address VARCHAR(255),
    latitude DECIMAL(10,8) NOT NULL COMMENT 'For map display',
    longitude DECIMAL(11,8) NOT NULL COMMENT 'For map display',

    park_type VARCHAR(50),
    acres DECIMAL(10,2) COMMENT 'Park size for scoring',
    is_waterfront BOOLEAN DEFAULT FALSE,

    avg_rating DECIMAL(3,2),
);

```

Facility Table:

```

DROP TABLE IF EXISTS Facility;
CREATE TABLE Facility (
    facility_id INT AUTO_INCREMENT PRIMARY KEY,
    park_id VARCHAR(20) NOT NULL,
    facility_type VARCHAR(100) NOT NULL,
    dimensions VARCHAR(100),
    surface_type VARCHAR(50) COMMENT 'Natural grass, synthetic, asphalt, etc.',
    is_lighted BOOLEAN DEFAULT FALSE,
    is_accessible BOOLEAN DEFAULT FALSE COMMENT 'ADA accessible',
    field_condition VARCHAR(20) DEFAULT 'Fair' COMMENT 'Good/Fair/Poor',

    avg_facility_rating DECIMAL(3,2) DEFAULT 0.00,
    total_facility_reviews INT DEFAULT 0,

    FOREIGN KEY (park_id) REFERENCES Park(park_id) ON DELETE CASCADE,
);

```

Trail Table:

```

DROP TABLE IF EXISTS Trail;
CREATE TABLE Trail (
    trail_id INT AUTO_INCREMENT PRIMARY KEY,
    park_id VARCHAR(20) NOT NULL,

    trail_name VARCHAR(200),
    width_ft VARCHAR(50),
    surface VARCHAR(50) COMMENT 'Paved, dirt, gravel, etc.',
    difficulty VARCHAR(200),
    length_miles DECIMAL(6,2),

    has_trail_markers BOOLEAN DEFAULT FALSE,

    avg_trail_rating DECIMAL(3,2) DEFAULT 0.00,
    total_trail_reviews INT DEFAULT 0,

```

```
    FOREIGN KEY (park_id) REFERENCES Park(park_id) ON DELETE CASCADE,  
);
```

Review Table:

```
DROP TABLE IF EXISTS Review;  
CREATE TABLE Review (  
    review_id INT PRIMARY KEY AUTO_INCREMENT,  
    user_id INT NOT NULL,  
    park_id VARCHAR(20),  
    facility_id INT,  
    rating DECIMAL(2,1) CHECK (rating >= 0 AND rating <= 5),  
    comment TEXT,  
    create_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    last_update_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
    FOREIGN KEY (user_id) REFERENCES User(user_id) ON DELETE CASCADE,  
    FOREIGN KEY (park_id) REFERENCES Park(park_id) ON DELETE CASCADE,  
    FOREIGN KEY (facility_id) REFERENCES Facility(facility_id) ON DELETE SET NULL,  
);
```

Insert data into these tables

Park table:

Inject from Parks_Properties.csv

[Parks Properties | NYC Open Data](#)

```
mysql> SELECT COUNT(*) FROM Park;  
+-----+  
| COUNT(*) |  
+-----+  
|      2052 |  
+-----+  
1 row in set (0.00 sec)
```

Facility table:

Inject from Athletic_Facilities.csv

[Athletic Facilities | NYC Open Data](#)

```
mysql> SELECT COUNT(*) FROM Facility;
+-----+
| COUNT(*) |
+-----+
|      5510 |
+-----+
1 row in set (0.00 sec)
```

Trail table:

Inject from Parks_Trails.csv
[Parks Trails | NYC Open Data](#)

```
mysql> SELECT COUNT(*) FROM Trail;
+-----+
| COUNT(*) |
+-----+
|      5346 |
+-----+
1 row in set (0.01 sec)
```

4 advanced SQL queries for application

The park, facility number for each borough

```
SELECT
    p.borough,
    COUNT(DISTINCT p.park_id) AS total_parks,
    COUNT(f.facility_id) AS total_facilities,
    ROUND(AVG(p.avg_rating), 2) AS avg_park_rating
FROM Park p
LEFT JOIN Facility f ON p.park_id = f.park_id
WHERE p.borough IS NOT NULL
GROUP BY p.borough
ORDER BY total_parks DESC
LIMIT 15;
```

Output:

Starting Query 1: Borough Statistics

```
=====
borough total_parks      total_facilities      avg_park_rating
Brooklyn        626          2034        0.00
Queens         475          1531        0.00
Bronx          396          716         0.00
Manhattan       394          1012        0.00
Staten Island   161          217         0.00
```

TOP 15 parks with most basketball courts

```
SELECT
    p.park_id,
    p.park_name,
    p.borough,
    COUNT(f.facility_id) AS basketball_court_count
FROM Park p
JOIN Facility f ON p.park_id = f.park_id
WHERE f.facility_type = 'Basketball'
GROUP BY p.park_id, p.park_name, p.borough
HAVING COUNT(f.facility_id) >= (
    SELECT AVG(court_count)
    FROM (
        SELECT COUNT(*) AS court_count
        FROM Facility
        WHERE facility_type = 'Basketball'
        GROUP BY park_id
    ) AS subquery
)
ORDER BY basketball_court_count DESC
LIMIT 15;
```

Output:

Starting Query 2: Parks with Most Basketball Courts

park_id	park_name	borough	basketball_court_count
B057	Marine Park	Brooklyn	19
M071	Riverside Park	Manhattan	11
Q005	Baisley Pond Park	Queens	10
M037	Highbridge Park	Manhattan	9
B052	Leif Ericson Park	Brooklyn	8
B028	Dyker Beach Park	Brooklyn	7
B008	Betsy Head Park	Brooklyn	7
X010	Crotona Park	Bronx	7
Q308	Bowne Playground	Queens	7
M018	Carmansville Playground	Manhattan	7
R071	Great Kills Veterans Playground	Staten Island	7
M200	Booker T. Washington Playground	Manhattan	7
B269	Jesse Owens Playground	Brooklyn	7
X002	Bronx Park	Bronx	6
M010	Central Park	Manhattan	6

Comparison between lighting and unlighting parks

```
SELECT
    'With Lighting' AS facility_lighting,
    COUNT(DISTINCT p.park_id) AS park_count,
    ROUND(AVG(p.avg_rating), 2) AS avg_rating,
    ROUND(AVG(p.acres), 2) AS avg_acres
FROM Park p
JOIN Facility f ON p.park_id = f.park_id
WHERE f.is_lighted = TRUE
GROUP BY f.is_lighted

UNION

SELECT
    'Without Lighting' AS facility_lighting,
    COUNT(DISTINCT p.park_id) AS park_count,
    ROUND(AVG(p.avg_rating), 2) AS avg_rating,
    ROUND(AVG(p.acres), 2) AS avg_acres
FROM Park p
JOIN Facility f ON p.park_id = f.park_id
WHERE f.is_lighted = FALSE
GROUP BY f.is_lighted

LIMIT 15;
```

Output:

Starting Query 3: Lighting Comparison

```
=====
facility_lighting      park_count      avg_rating      avg_acres
With Lighting          84            0.00           74.75
Without Lighting        719           0.00           80.80
```

The park with Facilities and trails

```
SELECT
    p.park_id,
    p.park_name,
    p.borough,
    COUNT(DISTINCT f.facility_type) AS facility_types,
    COUNT(DISTINCT t.trail_id) AS trail_count,
    ROUND(p.acres, 2) AS park_size_acres
FROM Park p
JOIN Facility f ON p.park_id = f.park_id
JOIN Trail t ON p.park_id = t.park_id
WHERE p.acres > (
    SELECT AVG(acres)
    FROM Park
    WHERE acres IS NOT NULL
)
GROUP BY p.park_id, p.park_name, p.borough, p.acres
ORDER BY facility_types DESC, trail_count DESC
LIMIT 15;
```

Output:

Starting Query 4: Parks with Facilities and Trails

```
=====
park_id park_name      borough facility_types   trail_count   park_size_acres
B057   Marine Park    Brooklyn 13             205          800.00
M042   Inwood Hill Park Manhattan 9            252          196.40
Q021   Cunningham Park Queens 9             242          358.00
B125   Calvert Vaux Park Brooklyn 9            11           85.53
B018   Canarsie Park   Brooklyn 8             53           132.20
B166D  McGuire Fields Brooklyn 8             23           77.18
Q015   Forest Park    Queens 7              267          506.86
X002   Bronx Park     Bronx   7              148          718.37
Q300   Kissena Corridor Park Queens 7          68           100.87
M071   Riverside Park  Manhattan 7            34           253.17
M010   Central Park    Manhattan 6            277          840.01
X118   Soundview Park  Bronx   6              85           269.42
M028   Fort Washington Park Manhattan 6          51           184.14
Q001   Alley Pond Park Queens 5              515          635.51
M037   Highbridge Park Manhattan 5            118          130.10
```

Part2

Baseline

Query1

```
EXPLAIN
--> Limit: 15 row(s) (actual time=8.25..8.25 rows=5 loops=1)\n      ->
Sort: total_parks DESC, limit input to 15 row(s) per chunk (actual
time=8.25..8.25 rows=5 loops=1)\n          -> Stream results (cost=5781
rows=45.3) (actual time=1.83..8.25 rows=5 loops=1)\n                  -> Group
aggregate: avg(p.avg_rating), count(distinct p.park_id),
count(f.facility_id) (cost=5781 rows=45.3) (actual time=1.82..8.24
rows=5 loops=1)\n                      -> Nested loop left join (cost=2267
rows=15253) (actual time=0.622..4.18 rows=6835 loops=1)\n
--> Sort: p.borough (cost=205 rows=2052) (actual time=0.612..0.68
rows=2052 loops=1)\n                  -> Filter: (p.borough is not
null) (cost=205 rows=2052) (actual time=0.0208..0.44 rows=2052
loops=1)\n                  -> Table scan on p (cost=205
rows=2052) (actual time=0.0205..0.368 rows=2052 loops=1)\n
--> Covering index lookup on f using park_id (park_id = p.park_id)
(cost=0.291 rows=7.43) (actual time=0.00124..0.00154 rows=2.69
loops=2052)\n
```

Screenshot:

```
Starting Query 1: Borough Statistics
=====
Limit: 15 row(s) (actual time=8.25..8.25 rows=5 loops=1)\n      -> Sort: total_parks DESC, limit input to 15 row(s) per chunk (actual time=8.25..8.25 rows=5 loops=1)\n          -> Stream results (cost=5781 rows=45.3) (actual time=1.83..8.25
5 rows=5 loops=1)\n          -> Group aggregate: avg(p.avg_rating), count(distinct p.park_id), count(f.facility_id) (cost=5781 rows=45.3) (actual time=1.82..8.24 rows=5 loops=1)\n          -> Nested loop left join (cost=2267 rows=15253) (actual time=0.622..4.18 rows=6835 loops=1)\n          -> Sort: p.borough (cost=205 rows=2052) (actual time=0.612..0.68 rows=2052 loops=1)\n          -> Filter: (p.borough is not null) (cost=205 rows=2052) (actual time=0.0205..0.44 rows=2052 loops=1)\n          -> Table scan on p (cost=205 rows=2052) (actual time=0.0205..0.368 rows=2052 loops=1)\n          -> Covering index lookup on f using park_id (park_id = p.park_id) (cost=0.291 rows=7.43) (actual time=0.00124..0.00154 rows=2.69 loops=2052)\n
```

Query2

```
EXPLAIN
--> Limit: 15 row(s) (actual time=7.72..7.72 rows=15 loops=1)\n      ->
Sort: basketball_court_count DESC (actual time=7.72..7.72 rows=15
loops=1)\n          -> Filter: (`count(f.facility_id)` >= (select #2))
(actual time=7.59..7.68 rows=222 loops=1)\n                  -> Table scan on
<temporary> (actual time=3.52..3.58 rows=608 loops=1)\n
--> Aggregate using temporary table (actual time=3.52..3.52 rows=608
loops=1)\n                  -> Nested loop inner join (cost=736
rows=540) (actual time=0.0798..2.59 rows=1484 loops=1)\n
--> Filter: (f.facility_type = 'Basketball') (cost=547 rows=540) (actual
time=0.0753..1.2 rows=1484 loops=1)\n                  ->
Table scan on f (cost=547 rows=5404) (actual time=0.063..0.918
rows=5510 loops=1)\n                  -> Single-row index lookup
on p using PRIMARY (park_id = f.park_id) (cost=0.25 rows=1) (actual
time=824e-6..842e-6 rows=1 loops=1484)\n          -> Select #2
(subquery in condition; run only once)\n                  -> Aggregate:
avg(subquery.court_count) (cost=930..930 rows=1) (actual
time=4.06..4.06 rows=1 loops=1)\n                  -> Table scan on
subquery (cost=796..805 rows=540) (actual time=4.01..4.04 rows=608
loops=1)\n                  -> Materialize (cost=796..796
```

```

rows=540) (actual time=4.01..4.01 rows=608 loops=1)\n
-> Group aggregate: count(0) (cost=672 rows=540) (actual\n
time=0.301..3.97 rows=608 loops=1)\n
Filter: (facility.facility_type = 'Basketball') (cost=547 rows=540)\n
(actual time=0.299..3.84 rows=1484 loops=1)\n
-> Index scan on Facility using park_id (cost=547 rows=5404) (actual\n
time=0.298..3.52 rows=5510 loops=1)\n
Screenshot:

```

```

Starting Query 2: Parks with Most Basketball Courts
=====
EXPLAIN
--> Limit: 15 row(s) (actual time=7.72..7.72 rows=15 loops=1)\n
-> Sort: basketball_court_count DESC (actual time=7.72..7.72 rows=15 loops=1)\n
-> Filter: ('count(f.facility_id) >= (select #2)' (actual time=7.59..7.68 rows=222 l\n
oops=1))\n
-> Table scan on <temporary> (actual time=3.52..3.58 rows=608 loops=1)\n
-> Aggregate using temporary table (actual time=3.52..3.52 rows=608 loops=1)\n
-> Nested loop inner join (cost=7.073..7.073 rows=15 loops=1)\n
-> Filter: (f.facility_type = 'Basketball') (cost=547 rows=540) (actual time=6.873..6.873\n
rows=540) (actual time=6.873..6.873 rows=540 loops=1)\n
-> Single-row index lookup on p using PRIMARY (park_id = f.park_id) (cost=0.25 rows=1)\n
-> Select #2 (subquery in condition; run only once)\n
-> Aggregate: avg(subquery.court_count) (cost=938..938 rows=1) (actual time=4.86..4.86\n
rows=1)\n
-> Filter: Materialized subquery (cost=938..938 rows=1) (actual time=4.86..4.86 rows=1)\n
-> Table scan on subquery (cost=547 rows=540) (actual time=4.86..4.86 rows=1)\n
-> Group aggregate: count(0) (cost=672 rows=540) (actual time=0.299..3.84 rows=1484 loops=1)\n
-> Index scan on Facility using park_id (cost=547 rows=5404) (actual time=0.298..3.52 rows=5510 loops=1)\n

```

Query3

```

EXPLAIN
-> Limit: 15 row(s) (cost=1723..1725 rows=2) (actual time=9.04..9.04\n
rows=2 loops=1)\n
-> Table scan on <union temporary>\n
(cost=1723..1725 rows=2) (actual time=9.04..9.04 rows=2 loops=1)\n
-> Union materialize with deduplication (cost=1722..1722 rows=2)\n
(actual time=9.04..9.04 rows=2 loops=1)\n
-> Limit table\n
size: 15 unique row(s)\n
-> Group aggregate:\n
avg(p.acres), avg(p.avg_rating), count(distinct p.park_id) (cost=861\n
rows=1) (actual time=1.4..1.4 rows=1 loops=1)\n
-> Nested loop inner join (cost=736 rows=540) (actual time=0.0416..1.3\n
rows=338 loops=1)\n
-> Filter: (f.is_lighted = true) (cost=547 rows=540) (actual time=0.0372..1.04\n
rows=338 loops=1)\n
-> Table scan on f (cost=547 rows=5404) (actual time=0.0276..0.9\n
rows=5510 loops=1)\n
-> Single-row index lookup\n
on p using PRIMARY (park_id = f.park_id) (cost=0.25 rows=1) (actual\n
time=634e-6..658e-6 rows=1 loops=338)\n
-> Limit table size:\n
15 unique row(s)\n
-> Group aggregate: avg(p.acres),\n
avg(p.avg_rating), count(distinct p.park_id) (cost=861 rows=1) (actual\n
time=7.64..7.64 rows=1 loops=1)\n
-> Nested loop\n
inner join (cost=736 rows=540) (actual time=0.0311..5.71 rows=5172\n
loops=1)\n
-> Filter: (f.is_lighted = false) (cost=547 rows=540) (actual time=0.0279..1.12\n
rows=5172 loops=1)\n
-> Table scan on f (cost=547 rows=5404) (actual time=0.0278..0.896\n
rows=5510 loops=1)\n
-> Single-row index lookup\n
on p using PRIMARY (park_id = f.park_id) (cost=0.25 rows=1) (actual\n
time=767e-6..787e-6 rows=1 loops=5172)\n
Screenshot:

```

```

Starting Query 3: Lighting Comparison
=====
EXPLAIN
--> Limit: 15 row(s) (cost=1723..1725 rows=2) (actual time=9.04..9.04 rows=2 loops=1)\n
-> Table scan on <union temporary> (cost=1723..1725 rows=2) (actual time=9.04..9.04 rows=2 loops=1)\n
-> Limit table size: 15 unique row(s)\n
-> Group aggregate: avg(p.acres), avg(p.avg_rating), count(distinct p.park_id) (cost=861 rows=1) (actual t\n
ime=1.4..1.4 rows=1 loops=1)\n
-> Nested loop inner join (cost=736 rows=540) (actual time=0.0416..1.3 rows=338 loops=1)\n
-> Table scan on p (cost=547 rows=5404) (actual time=0.0276..0.9 rows=5510 loops=1)\n
-> Filter: (f.is_lighted = true) (cost=547 rows=540) (actual time=0.0372..1.04 rows=338 loops=1)\n
-> Nested loop inner join (cost=736 rows=540) (actual time=0.0311..5.71 rows=5172 loops=1)\n
-> Filter: (f.is_lighted = true) (cost=547 rows=540) (actual time=0.0279..1.12 rows=5172 loops=1)\n
-> Table scan on f (cost=547 rows=5404) (actual time=0.0278..0.896 rows=5510 loops=1)\n
-> Filter: (f.is_lighted = false) (cost=547 rows=540) (actual time=0.0279..1.12 rows=5172 loops=1)\n
-> Table scan on p using PRIMARY (park_id = f.park_id) (cost=0.25 rows=1) (actual time=767e-6..787e-6 rows=1 loops=5172)\n

```

Query4

```
EXPLAIN
--> Limit: 15 row(s)  (actual time=314..314 rows=15 loops=1)\n      ->
Sort: facility_types DESC, trail_count DESC, limit input to 15 row(s)\nper chunk  (actual time=314..314 rows=15 loops=1)\n      -> Stream\nresults  (actual time=204..314 rows=34 loops=1)\n      -> Group\naggregate: count(distinct facility.facility_type), count(distinct\ntrail.trail_id)  (actual time=204..314 rows=34 loops=1)\n-> Sort: p.park_id, p.park_name, p.borough, p.acres  (actual\ntime=202..211 rows=114568 loops=1)\n      -> Stream\nresults  (cost=0.947 rows=0.705)  (actual time=0.0272..148 rows=114568\nloops=1)\n      -> Nested loop inner join  (cost=0.947\nrows=0.705)  (actual time=0.0251..100 rows=114568 loops=1)\n-> Nested loop inner join  (cost=0.7 rows=0.0916)  (actual\ntime=0.0119..3.32 rows=5290 loops=1)\n      ->\nCovering index scan on t using idx_trail_park_id  (cost=0.35 rows=1)\n(actual time=0.00583..0.929 rows=5346 loops=1)\n-> Filter: (p.acres > (select #2))  (cost=0.259 rows=0.0916)  (actual\ntime=288e-6..348e-6 rows=0.99 loops=5346)\n-> Single-row index lookup on p using PRIMARY (park_id = t.park_id)\n(cost=0.259 rows=1)  (actual time=146e-6..166e-6 rows=1 loops=5346)\n-> Select #2 (subquery in condition; run only once)\n-> Aggregate: avg(park.acres)  (cost=678 rows=1)  (actual\ntime=0.355..0.355 rows=1 loops=1)\n-> Filter: (park.acres is not null)  (cost=205 rows=2052)  (actual\ntime=0.0112..0.283 rows=2052 loops=1)\n-> Covering index scan on Park using idx_park_acres  (cost=205\nrows=2052)  (actual time=0.0108..0.208 rows=2052 loops=1)\n-> Index lookup on f using idx_facility_park_id (park_id = t.park_id)\n(cost=10.3 rows=7.69)  (actual time=0.00488..0.0174 rows=21.7\nloops=5290)\n
```

Screenshot:

```
Starting Query 4: Parks with Facilities and Trails
=====
EXPLAIN
--> Limit: 15 row(s)  (actual time=286..286 rows=15 loops=1)\n      -> Sort: facility_types DESC, trail_count DESC, limit input to 35 row(s) per chunk  (actual time=286..286 rows=15 loops=1)\n      -> Stream results  (cost=7842 rows=13245)  (actual time=0.497..129 rows=114568 loops=1)\n      -> Nested loop inner join  (cost=7842 rows=13245)  (actual time=0.495..36.7 row\nsize=114568 loops=1)\n      -> Nested loop inner join  (cost=2406 rows=1782)  (actual time=0.481..3.16 rows=5290 loops=1)\n      -> Covering index scan on t using park_id  (cost=535 rows=5346)  (act\nual time=0.392..0.392 rows=5346 loops=1)\n      -> Filter: (p.acres > (select #2))  (cost=0.25 rows=0.333)  (actual time=317e-6..372e-6 rows=0.99 loops=5346)\n      -> Single-row index\nlookup on p using PRIMARY (park_id = t.park_id)  (cost=0.25 rows=1)  (actual time=317e-6..372e-6 rows=0.99 loops=5346)\n      -> Aggregate: avg(park.acres)  (cost=631 rows=1)  (actual time=0.436..0.436 rows=1 loops=1)\n      -> Filter: (park.acres is not null)  (cost=205 rows=1847)  (actual time=0.0147..0.0357 rows=2052 loops=1)\n      -> Table scan on Park  (cost=205 rows=2052)  (actual time=0.0145..0.284 rows=2052 loops=1)\n      -> Index lookup on f using park_id (park_id = t.park_id)  (cost=1.86 r\nows=7.43)  (actual time=0.00429..0.0149 rows=21.7 loops=5290)\n
```

Index 1

```
CREATE INDEX idx_park_borough ON Park(borough);\nCREATE INDEX idx_park_acres ON Park(acres);\n\nCREATE INDEX idx_facility_park_id ON Facility(park_id);
```

```

CREATE INDEX idx_facility_type ON Facility(facility_type);
CREATE INDEX idx_facility_lighted ON Facility(is_lighted);

CREATE INDEX idx_trail_park_id ON Trail(park_id);

```

Reason:

We added single-column indexes on the most common filter and join attributes. borough and acres help speed up park searches, while park_id, facility_type, and is_lighted make facility lookups faster. Trail(park_id) improves joins between parks and trails. These indexes provide a simple baseline for later performance comparison.

Query1

```

EXPLAIN
--> Limit: 15 row(s) (actual time=10.8..10.8 rows=5 loops=1)
--> Sort: total_parks DESC, limit input to 15 row(s) per chunk (actual time=10.8..10.8 rows=5 loops=1)
--> Stream results (cost=7458 rows=5) (actual time=2.06..10.7 rows=5 loops=1)
--> Group aggregate: avg(p.avg_rating), count(distinct p.park_id),
--> count(f.facility_id) (cost=7458 rows=5) (actual time=2.06..10.7 rows=5 loops=1)
--> Nested loop left join (cost=3820 rows=15787) (actual time=0.714..5.31 rows=6835 loops=1)
--> Sort: p.borough (cost=205 rows=2052) (actual time=0.699..0.784 rows=2052 loops=1)
--> Filter: (p.borough is not null) (cost=205 rows=2052) (actual time=0.0273..0.511 rows=2052 loops=1)
--> Table scan on p (cost=205 rows=2052) (actual time=0.0269..0.427 rows=2052 loops=1)
--> Covering index lookup on f using idx_facility_park_id (park_id = p.park_id) (cost=0.993 rows=7.69) (actual time=0.00156..0.00201 rows=2.69 loops=2052)

```

Screenshot:

```

EXPLAIN
--> Limit: 15 row(s) (actual time=10.8..10.8 rows=5 loops=1)
--> Sort: total_parks DESC, limit input to 15 row(s) per chunk (actual time=10.8..10.8 rows=5 loops=1)
--> Stream results (cost=7458 rows=5) (actual time=2.06..10.7 rows=5 loops=1)
--> Nested loop left join (cost=3820 rows=15787) (actual time=0.714..5.31 rows=6835 loops=1)
--> Sort: p.borough (cost=205 rows=2052) (actual time=0.699..0.784 rows=2052 loops=1)
--> Filter: (p.borough is not null) (cost=205 rows=2052) (actual time=0.0273..0.511 rows=2052 loops=1)
--> Table scan on p (cost=205 rows=2052) (actual time=0.0269..0.427 rows=2052 loops=1)
--> Covering index lookup on f using idx_facility_park_id (park_id = p.park_id) (cost=0.993 rows=7.69) (actual time=0.00156..0.00201 rows=2.69 loops=2052)

```

Query2

```

EXPLAIN
--> Limit: 15 row(s) (actual time=6.53..6.53 rows=15 loops=1)
--> Sort: basketball_court_count DESC (actual time=6.53..6.53 rows=15 loops=1)
--> Filter: (`count(f.facility_id)` >= (select #2)) (actual time=6.4..6.49 rows=222 loops=1)
--> Table scan on <temporary> (actual time=4.79..4.85 rows=608 loops=1)
--> Aggregate using temporary table (actual time=4.79..4.79 rows=608 loops=1)
--> Nested loop inner join (cost=688 rows=1484) (actual time=0.144..3.47 rows=1484 loops=1)
--> Index lookup on f using idx_facility_type (facility_type =

```

```
'Basketball'), with index condition: (f.facility_type = 'Basketball')
(cost=169 rows=1484) (actual time=0.138..1.46 rows=1484 loops=1)\n
-> Single-row index lookup on p using PRIMARY (park_id = f.park_id)
(cost=0.25 rows=1) (actual time=0.00124..0.00126 rows=1 loops=1484)\n
-> Select #2 (subquery in condition; run only once)\n
Aggregate: avg(subquery.court_count) (cost=2.5..2.5 rows=1) (actual
time=1.59..1.59 rows=1 loops=1)\n
-> Table scan on
subquery (cost=2.5..2.5 rows=0) (actual time=1.53..1.56 rows=608
loops=1)\n
-> Materialize (cost=0..0 rows=0)
(actual time=1.53..1.53 rows=608 loops=1)\n
-> Table scan on <temporary> (actual time=1.46..1.5 rows=608 loops=1)\n
-> Aggregate using temporary table (actual time=1.46..1.46 rows=608
loops=1)\n
-> Index lookup on
Facility using idx_facility_type (facility_type = 'Basketball'), with
index condition: (facility.facility_type = 'Basketball') (cost=169
rows=1484) (actual time=0.11..1.11 rows=1484 loops=1)\n
Screenshot:
```

```
EXPLAIN
Limit: 15 row(s) (actual time=6.53..6.53 rows=15 loops=1)\n
-> Sort: basketball_court_count DESC (actual time=6.53..6.53 rows=15 loops=1)\n
-> Filter: ('count(f.facility_id)' >= (select #2)) (actual time=6.4..6.49 rows=222 lo
ops=1)\n
-> Table scan on <temporary> (actual time=4.79..4.79 rows=608 loops=1)\n
-> Aggregate using temporary table (actual time=4.79..4.79 rows=608 loops=1)\n
-> Nested loop inner join (cost=169 rows=1484) (ac
tual time=1.59..1.59 rows=1484 loops=1)\n
-> Single-row index lookup on f using idx_facility_type (facility_type = 'Basketball'), with index condition: (ff.facility_type = 'Basketball') (cost=169 rows=1484) (ac
tual time=1.46..1.46 rows=1484 loops=1)\n
-> Index lookup on p using PRIMARY (park_id = f.park_id) (cost=0.25 rows=1)\n
-> Select #2 (subquery in condition; run only once)\n
Aggregate: avg(subquery.court_count) (cost=2.5..2.5 rows=1) (actual time=1.53..1.53 rows=1)\n
-> Table scan on subquery (cost=2.5..2.5 rows=0) (actual time=1.53..1.53 rows=0)\n
-> Materialize (cost=0..0 rows=0) (actual time=1.53..1.53 rows=608 loops=1)\n
-> Table scan on <temporary> (actual time=1.46..1.5 rows=608 loops=1)\n
-> Index lookup on Facility using idx_facility_type (facility_type = 'Basketball'), with index condition: (fac
ity.facility_type = 'Basketball') (cost=169 rows=1484) (actual time=0.11..1.11 rows=1484 loops=1)\n
```

Query3

```
EXPLAIN
-> Limit: 15 row(s) (cost=3791..3793 rows=2) (actual time=11.2..11.2
rows=2 loops=1)\n
-> Table scan on <union temporary>
(cost=3791..3793 rows=2) (actual time=11.2..11.2 rows=2 loops=1)\n
-> Union materialize with deduplication (cost=3790..3790 rows=2)
(actual time=11.2..11.2 rows=2 loops=1)\n
-> Limit table
size: 15 unique row(s)\n
-> Group aggregate:
avg(p.acres), avg(p.avg_rating), count(distinct p.park_id) (cost=250
rows=1) (actual time=0.582..0.582 rows=1 loops=1)\n
-> Nested loop inner join (cost=172 rows=338) (actual
time=0.0347..0.491 rows=338 loops=1)\n
-> Index
lookup on f using idx_facility_lighted (is_lighted = true) (cost=54.1
rows=338) (actual time=0.0311..0.244 rows=338 loops=1)\n
-> Single-row index lookup on p using PRIMARY (park_id = f.park_id)
(cost=0.25 rows=1) (actual time=616e-6..636e-6 rows=1 loops=338)\n
-> Limit table size: 15 unique row(s)\n
-> Group
aggregate: avg(p.acres), avg(p.avg_rating), count(distinct p.park_id)
(cost=3539 rows=1) (actual time=10.6..10.6 rows=1 loops=1)\n
-> Nested loop inner join (cost=2348 rows=5172) (actual
time=0.103..8.57 rows=5172 loops=1)\n
-> Index
lookup on f using idx_facility_lighted (is_lighted = false) (cost=537
rows=5172) (actual time=0.101..3.67 rows=5172 loops=1)\n
-> Single-row index lookup on p using PRIMARY (park_id = f.park_id)
```

(cost=0.25 rows=1) (actual time=832e-6..852e-6 rows=1 loops=5172)\nScreenshot:

```

EXPLAIN
-- Limit: 15 rows\(\) (cost=3791..3793 rows=2) [actual time=11.2..11.2 rows=2 loops=1]\n    -> Table scan on union temporary` (\cost=3791..3793 rows=2) [actual time=11.2..11.2 rows=2 loops=1]\n        -> Union materialize with deduplication\n        (cost=3790..3798 rows=2) [actual time=11.2..11.2 rows=2 loops=1]\n            -> Limit table size: 15 unique rows\(\)\n            -> Group aggregate: avg(p.acres), avg(p.avg_rating), count(distinct p.id) (\cost=250 rows=1) [actual time=10.5..10.5 rows=1]\n                -> Nested loop inner join (cost=172 rows=338)\n                    -> Index lookup on f using idx_facility_lighted (is_lighted = false)\n                    -> Index lookup on p using PRIMARY [park_id = f.park_id] (\cost=8..16 rows=1 loops=1)\n                    -> L\n            cost=54.1 rows=338) [actual time=0..0.311..0.244 rows=338 loops=1]\n                -> Single-row index lookup on p using PRIMARY [park_id = f.park_id] (\cost=8..16 rows=1 loops=1)\n                -> Limit table size: 15 unique rows\(\)\n                -> Group aggregate: avg(p.acres), avg(p.avg_rating), count(distinct p.id) (\cost=250 rows=1) [actual time=10.6..10.6 rows=1 loops=1]\n                    -> Nested loop inner join (cost=172 rows=338)\n                        -> Single-row index lookup on f using idx_facility_lighted (is_lighted = false) (\cost=537 rows=1) [actual time=0..101..1.367 rows=5172 loops=1]\n                            -> Index lookup on f using idx_facility_lighted (is_lighted = false) (\cost=0.25 rows=1) [actual time=832e-6..8.572e-6 rows=1 loops=5172]\n
```

Query4

```
EXPLAIN
-> Limit: 15 row(s) (actual time=314..314 rows=15 loops=1)\n      ->
Sort: facility_types DESC, trail_count DESC, limit input to 15 row(s)
per chunk (actual time=314..314 rows=15 loops=1)\n          -> Stream
results (actual time=204..314 rows=34 loops=1)\n          -> Group
aggregate: count(distinct facility.facility_type), count(distinct
trail.trail_id) (actual time=204..314 rows=34 loops=1)\n
-> Sort: p.park_id, p.park_name, p.borough, p.acres (actual
time=202..211 rows=114568 loops=1)\n          -> Stream
results (cost=0.947 rows=0.705) (actual time=0.0272..148 rows=114568
loops=1)\n          -> Nested loop inner join (cost=0.947
rows=0.705) (actual time=0.0251..100 rows=114568 loops=1)\n
-> Nested loop inner join (cost=0.7 rows=0.0916) (actual
time=0.0119..3.32 rows=5290 loops=1)\n          ->
Covering index scan on t using idx_trail_park_id (cost=0.35 rows=1)
(actual time=0.00583..0.929 rows=5346 loops=1)\n
-> Filter: (p.acres > (select #2)) (cost=0.259 rows=0.0916) (actual
time=288e-6..348e-6 rows=0.99 loops=5346)\n
-> Single-row index lookup on p using PRIMARY (park_id = t.park_id)
(cost=0.259 rows=1) (actual time=146e-6..166e-6 rows=1 loops=5346)\n
-> Select #2 (subquery in condition; run only once)\n
-> Aggregate: avg(park.acres) (cost=678 rows=1) (actual
time=0.355..0.355 rows=1 loops=1)\n
-> Filter: (park.acres is not null) (cost=205 rows=2052) (actual
time=0.0112..0.283 rows=2052 loops=1)\n
-> Covering index scan on Park using idx_park_acres (cost=205
rows=2052) (actual time=0.0108..0.208 rows=2052 loops=1)\n
-> Index lookup on f using idx_facility_park_id (park_id = t.park_id)
(cost=10.3 rows=7.69) (actual time=0.00488..0.0174 rows=21.7
loops=5290)\n
Screenshot:
```

Index 2

```
CREATE INDEX idx_park_borough_rating ON Park(borough, avg_rating);

CREATE INDEX idx_park_acres_id ON Park(acres, park_id);

CREATE INDEX idx_facility_type_park ON Facility(facility_type, park_id);

CREATE INDEX idx_facility_lighted_park ON Facility(is_lighted, park_id);

CREATE INDEX idx_trail_park_id ON Trail(park_id);
```

Reason:

Park(borough, avg_rating) and Park(acres, park_id) speed up grouping by borough and filtering by park size.

Facility(facility_type, park_id) and Facility(is_lighted, park_id) optimize queries involving facility type or lighting conditions.

Trail(park_id) improves join performance between parks and trails.

These composite indexes provide better coverage and significantly reduce full table scans, improving overall query efficiency.

Query1

```
EXPLAIN
-> Limit: 15 row(s) (actual time=10.4..10.4 rows=5 loops=1)\n      ->
Sort: total_parks DESC, limit input to 15 row(s) per chunk (actual
time=10.4..10.4 rows=5 loops=1)\n          -> Stream results (cost=5740
rows=5) (actual time=1.71..10.4 rows=5 loops=1)\n                  -> Group
aggregate: avg(p.avg_rating), count(distinct p.park_id),
count(f.facility_id) (cost=5740 rows=5) (actual time=1.7..10.4 rows=5
loops=1)\n                      -> Nested loop left join (cost=2294
rows=14954) (actual time=0.0355..5.28 rows=6835 loops=1)\n
-> Filter: (p.borough is not null) (cost=205 rows=2052) (actual
time=0.0257..0.545 rows=2052 loops=1)\n                  ->
Covering index scan on p using idx_park_borough_rating (cost=205
rows=2052) (actual time=0.0253..0.448 rows=2052 loops=1)\n
-> Covering index lookup on f using park_id (park_id = p.park_id)
(cost=0.29 rows=7.29) (actual time=0.00167..0.00207 rows=2.69
loops=2052)\n
```

Screenshot:

```
Starting Query 1: Borough Statistics
=====
EXPLAIN
-> Limit: 15 row(s) (actual time=10.4..10.4 rows=5 loops=1)\n      -> Sort: total_parks DESC, limit input to 15 row(s) per chunk (actual time=10.4..10.4 rows=5 loops=1)\n          -> Stream results (cost=5740 rows=5) (actual time=1.71..10.4 r
ows=5 loops=1)\n          aggregate: avg(p.avg_rating), count(distinct p.park_id), count(f.facility_id) (cost=5740 rows=5 loops=1)\n          -> Nested loop left join (cost=2294 rows=14954)\n          -> Covering index scan on p using idx_park_borough_rating (cost=205 rows=2052) (actual time=0.0355..5.28 rows=6835 loops=1)\n
-> Filter: (p.borough is not null) (cost=205 rows=2052) (actual time=0.0257..0.545 rows=2052 loops=1)\n          -> Covering index lookup on f using park_id (park_id = p.park_id) (cost=0.29 rows=7.29) (actual time=0.00167..0.00207 rows=2.69 loops=2052)\n
```

Query2

```
EXPLAIN
-> Limit: 15 row(s) (actual time=3.39..3.39 rows=15 loops=1)\n      ->
```

```

Sort: basketball_court_count DESC  (actual time=3.39..3.39 rows=15
loops=1)\n      -> Filter: (`count(f.facility_id)` >= (select #2))
(actual time=3.23..3.32 rows=222 loops=1)\n                  -> Table scan on
<temporary>  (actual time=2.61..2.67 rows=608 loops=1)\n
-> Aggregate using temporary table  (actual time=2.61..2.61 rows=608
loops=1)\n                  -> Nested loop inner join  (cost=701
rows=1484)  (actual time=0.0249..1.6 rows=1484 loops=1)\n
-> Filter: (f.facility_type = 'Basketball')  (cost=182 rows=1484)
(actual time=0.0195..0.424 rows=1484 loops=1)\n
-> Covering index lookup on f using idx_facility_type_park
(facility_type = 'Basketball')  (cost=182 rows=1484) (actual
time=0.0191..0.256 rows=1484 loops=1)\n
Single-row index lookup on p using PRIMARY (park_id = f.park_id)
(cost=0.25 rows=1) (actual time=664e-6..685e-6 rows=1 loops=1484)\n
-> Select #2 (subquery in condition; run only once)\n
Aggregate: avg(subquery.court_count)  (cost=870..870 rows=1) (actual
time=0.614..0.614 rows=1 loops=1)\n
-> Table scan on
subquery  (cost=691..703 rows=727) (actual time=0.556..0.594 rows=608
loops=1)\n
-> Materialize  (cost=691..691
rows=727) (actual time=0.556..0.556 rows=608 loops=1)\n
-> Group aggregate: count(0)  (cost=524 rows=727) (actual
time=0.018..0.521 rows=608 loops=1)\n
Filter: (facility.facility_type = 'Basketball')  (cost=182 rows=1484)
(actual time=0.0165..0.384 rows=1484 loops=1)\n
-> Covering index lookup on Facility using idx_facility_type_park
(facility_type = 'Basketball')  (cost=182 rows=1484) (actual
time=0.0162..0.219 rows=1484 loops=1)\n
Screenshot:

```

```

Starting Query 2: Parks with Most Basketball Courts
EXPLAIN
--> Limit: 15 row(s)  (cost=3873..3874 rows=2) (actual time=5.34..5.34
rows=2 loops=1)\n      -> Table scan on <union temporary>
(cost=3873..3874 rows=2) (actual time=5.34..5.34 rows=2 loops=1)\n
-> Union materialize with deduplication  (cost=3872..3872 rows=2)
(actual time=5.34..5.34 rows=2 loops=1)\n
size: 15 unique row(s)\n      -> Group aggregate:
avg(p.acres), avg(p.avg_rating), count(distinct p.park_id)  (cost=238
rows=1) (actual time=0.345..0.345 rows=1 loops=1)\n
-> Nested loop inner join  (cost=160 rows=338) (actual
time=0.0107..0.242 rows=338 loops=1)\n

```

Query3

```

EXPLAIN
--> Limit: 15 row(s)  (cost=3873..3874 rows=2) (actual time=5.34..5.34
rows=2 loops=1)\n      -> Table scan on <union temporary>
(cost=3873..3874 rows=2) (actual time=5.34..5.34 rows=2 loops=1)\n
-> Union materialize with deduplication  (cost=3872..3872 rows=2)
(actual time=5.34..5.34 rows=2 loops=1)\n
size: 15 unique row(s)\n      -> Group aggregate:
avg(p.acres), avg(p.avg_rating), count(distinct p.park_id)  (cost=238
rows=1) (actual time=0.345..0.345 rows=1 loops=1)\n
-> Nested loop inner join  (cost=160 rows=338) (actual
time=0.0107..0.242 rows=338 loops=1)\n

```

```

Covering index lookup on f using idx_facility_lighted_park (is_lighted =
true) (cost=42 rows=338) (actual time=0.00688..0.0552 rows=338
loops=1)\n                                -> Single-row index lookup on p using
PRIMARY (park_id = f.park_id) (cost=0.25 rows=1) (actual
time=430e-6..452e-6 rows=1 loops=338)\n                                -> Limit table size:
15 unique row(s)\n                                -> Group aggregate: avg(p.acres),
avg(p.avg_rating), count(distinct p.park_id) (cost=3633 rows=1) (actual
time=4.99..4.99 rows=1 loops=1)\n                                -> Nested loop
inner join (cost=2441 rows=5172) (actual time=0.0175..2.75 rows=5172
loops=1)\n                                -> Covering index lookup on f using
idx_facility_lighted_park (is_lighted = false) (cost=631 rows=5172)
(actual time=0.0152..0.724 rows=5172 loops=1)\n-> Single-row index lookup on p using PRIMARY (park_id = f.park_id)
(cost=0.25 rows=1) (actual time=267e-6..288e-6 rows=1 loops=5172)\n

Screenshot:

```

```

Starting Query 3: Lighting Comparison
=====
EXPLAIN
--> Limit: 15 row(s) (cost=3873..3874 rows=2 loops=1)\n      -> Table scan on cunis temporary < (cost=3873..3874 rows=2) (actual time=5.34..5.34 rows=2 loops=1)\n      -> Limit table size: 15 unique row(s)\n      -> Group aggregate: avg(p.acres), avg(p.avg_rating), count(distinct p.park_id) (cost=3633 rows=1)\n      -> Nested loop inner join (cost=168 rows=338) (actual time=0.0187..0.242 rows=338 loops=1)\n          -> Covering index lookup on f using idx_facility_lighted_park (is_lighted = true) (cost=42 rows=338) (actual time=0.00688..0.0552 rows=338 loops=1)\n          -> Single-row index lookup on p using PRIMARY (park_id = f.park_id) (cost=0.25 rows=1) (actual time=430e-6..452e-6 rows=1 loops=338)\n          -> Limit table size: 15 unique row(s)\n          -> Group aggregate: avg(p.acres), avg(p.avg_rating), count(distinct p.park_id) (cost=3633 rows=1)\n          -> Nested loop inner join (cost=151 rows=5172) (actual time=0.0175..2.75 rows=5172 loops=1)\n              -> Covering index lookup on t using idx_trail_park_id (cost=0.35 rows=1)\n              -> Filter: (p.acres > (select #2)) (cost=0.259 rows=0.99 loops=5346)\n                  -> Single-row index lookup on p using PRIMARY (park_id = t.park_id) (cost=0.259 rows=1) (actual time=114e-6..134e-6 rows=1 loops=5346)\n                  -> Select #2 (subquery in condition; run only once)\n                  -> Aggregate: avg(park.acres) (cost=678 rows=1) (actual time=0.357..0.357 rows=1 loops=1)\n                  -> Filter: (park.acres is not null) (cost=205 rows=2052) (actual time=0.0085..0.284 rows=2052 loops=1)\n
```

Query4

```

EXPLAIN
--> Limit: 15 row(s) (actual time=284..284 rows=15 loops=1)\n      ->
Sort: facility_types DESC, trail_count DESC, limit input to 15 row(s)
per chunk (actual time=284..284 rows=15 loops=1)\n      -> Stream
results (actual time=184..284 rows=34 loops=1)\n      -> Group
aggregate: count(distinct facility.facility_type), count(distinct
trail.trail_id) (actual time=184..284 rows=34 loops=1)\n
--> Sort: p.park_id, p.park_name, p.borough, p.acres (actual
time=182..191 rows=114568 loops=1)\n      -> Stream
results (cost=0.934 rows=0.668) (actual time=0.022..129 rows=114568
loops=1)\n      -> Nested loop inner join (cost=0.934
rows=0.668) (actual time=0.0198..86.2 rows=114568 loops=1)\n
--> Nested loop inner join (cost=0.7 rows=0.0916) (actual
time=0.00771..2.75 rows=5290 loops=1)\n
--> Covering index scan on t using idx_trail_park_id (cost=0.35 rows=1)
(actual time=0.00313..0.726 rows=5346 loops=1)\n
--> Filter: (p.acres > (select #2)) (cost=0.259 rows=0.99 loops=5346)\n
--> Single-row index lookup on p using PRIMARY (park_id = t.park_id)
(cost=0.259 rows=1) (actual time=114e-6..134e-6 rows=1 loops=5346)\n
--> Select #2 (subquery in condition; run only once)\n
--> Aggregate: avg(park.acres) (cost=678 rows=1) (actual
time=0.357..0.357 rows=1 loops=1)\n
--> Filter: (park.acres is not null) (cost=205 rows=2052) (actual
time=0.0085..0.284 rows=2052 loops=1)
```

```

-> Covering index scan on Park using idx_park_acres_id (cost=205
rows=2052) (actual time=0.00821..0.204 rows=2052 loops=1)\n
-> Index lookup on f using park_id (park_id = t.park_id) (cost=9.78
rows=7.29) (actual time=0.00425..0.0149 rows=21.7 loops=5290)\n
Screenshot:

```

```

Starting Query 4: Parks with Facilities and Trails
=====
EXPLAIN
-> Limit: 15 row(s) (actual time=284..284 rows=15 loops=1)\n    -> Sort: facility_types DESC, trail_count DESC, limit input to 15 row(s) per chunk (actual time=284..284 rows=15 loops=1)\n        -> Stream results (actual time=184..284 rows=15 loops=1)\n        -> Group aggregate: count(facility_type), count(trail_id), count(park_id), p.park_id, f.facility_id, p.borough\n        -> Nested loop left join (cost=182..191 rows=11456 loops=1)\n            -> Stream results (cost=8..934 rows=8,668) (actual time=8..022..129 rows=114568 loops=1)\n            -> Nested loop inner join (cost=8..934 rows=8,668) (actual time=8..019..086.2)\n                -> Covering index scan on t using idx_trail_park_id (cost=8..3\n                -> Single-row index lookup on p using PRIMARY (park_id = t.park_id) (cost=8..259 rows=1) (actual time=8..077..2.75 rows=5298 loops=1)\n                    -> Filter: (p.borough is not null) (cost=8..259 rows=8,936) (actual time=232..6..288..6)\n                        -> Covering index scan on t using idx_trail_park_id (cost=8..3\n                            -> Select #2 (subquery in condition; run only once)\n                            -> Single-row index lookup on p using PRIMARY (park_id = t.park_id) (cost=8..259 rows=1) (actual time=8..357..0.357 rows=1 loops=346)\n                                -> Aggregate: avg(park.acres) (cost=8..78 rows=1) (actual time=8..357..0.357 rows=1 loops=346)\n                                    -> Filter: (park.acres is not null) (cost=205 rows=2052) (actual time=0.00821..0.284 rows=2052 loops=1)\n                                        -> Index lookup on p using idx_park_acres_id (cost=205 rows=2052) (actual time=0.00821..0.284 rows=2052 loops=1)\n                                            -> Index lookup on p using park_id (park_id = t.park_id) (cost=9..78 rows=7..29) (actual time=0.00821..0.0149 rows=21..7 loops=5290)\n
```

Index 3

```

CREATE INDEX idx_park_complete ON Park(borough, avg_rating, acres);

CREATE INDEX idx_facility_complete ON Facility(park_id, facility_type,
is_lighted);

CREATE INDEX idx_trail_park ON Trail(park_id, trail_id);

```

Reason:

idx_park_complete helps with grouping and aggregation on borough, rating, and acres. idx_facility_complete and idx_trail_park improve joins and eliminate full table scans by providing full coverage for the most frequently accessed attributes.

Query1

```

EXPLAIN
-> Limit: 15 row(s) (actual time=9.74..9.74 rows=5 loops=1)\n    -> Sort: total_parks\nDESC, limit input to 15 row(s) per chunk (actual time=9.74..9.74 rows=5 loops=1)\n-> Stream results (cost=7322 rows=5) (actual time=1.55..9.73 rows=5 loops=1)\n-> Group aggregate: avg(p.avg_rating), count(distinct p.park_id), count(f.facility_id)\n(cost=7322 rows=5) (actual time=1.55..9.73 rows=5 loops=1)\n                -> Nested loop\nleft join (cost=3812 rows=15231) (actual time=0.06..5.09 rows=6835 loops=1)\n-> Filter: (p.borough is not null) (cost=205 rows=2052) (actual time=0.0481..0.512\nrows=2052 loops=1)\n                -> Covering index scan on p using\nidx_park_complete (cost=205 rows=2052) (actual time=0.0473..0.419 rows=2052\nloops=1)\n                -> Covering index lookup on f using idx_facility_complete (park_id\n= p.park_id) (cost=1.02 rows=7.42) (actual time=0.00163..0.00202 rows=2.69\nloops=2052)\nScreenshot:

```

```

Starting Query 1: Borough Statistics
=====
EXPLAIN
-> Limit: 15 row(s) (actual time=9.74..9.74 rows=5 loops=1)\n    -> Sort: total_parks DESC, limit input to 15 row(s) per chunk (actual time=9.74..9.74 rows=5 loops=1)\n        -> Stream results (cost=7322 rows=5) (actual time=1.55..9.73 rows=5 loops=1)\n        -> Group aggregate: avg(p.avg_rating), count(distinct p.park_id), count(f.facility_id)\n        (actual time=9.66..9.73 rows=6835 loops=1)\n            -> Filter: (p.borough is not null) (cost=205 rows=2052) (actual time=9.0473..9.419 rows=2052 loops=1)\n                -> Covering index scan on p using idx_park_complete (park_id = p.park_id) (cost=1.02 rows=7.42) (actual time=0.00163..0.00202 rows=2.69 loops=2052)\n
```

Query2

```
EXPLAIN
--> Limit: 15 row(s) (actual time=5.62..5.63 rows=15 loops=1)\n      ->
Sort: basketball_court_count DESC (actual time=5.62..5.62 rows=15\nloops=1)\n      -> Filter: (`count(f.facility_id)` >= (select #2))\n(actual time=5.49..5.58 rows=222 loops=1)\n      -> Table scan on\n<temporary> (actual time=4.03..4.08 rows=608 loops=1)\n-> Aggregate using temporary table (actual time=4.02..4.02 rows=608\nloops=1)\n      -> Nested loop inner join (cost=735\nrows=540) (actual time=0.0838..2.73 rows=1484 loops=1)\n-> Filter: (f.facility_type = 'Basketball') (cost=546 rows=540) (actual\ntime=0.0715..1.39 rows=1484 loops=1)\n      ->\nCovering index scan on f using idx_facility_complete (cost=546\nrows=5396) (actual time=0.0701..0.996 rows=5510 loops=1)\n-> Single-row index lookup on p using PRIMARY (park_id = f.park_id)\n(cost=0.25 rows=1) (actual time=769e-6..793e-6 rows=1 loops=1484)\n-> Select #2 (subquery in condition; run only once)\n      ->\nAggregate: avg(subquery.court_count) (cost=929..929 rows=1) (actual\ntime=1.44..1.44 rows=1 loops=1)\n      -> Table scan on\nsubquery (cost=795..804 rows=540) (actual time=1.39..1.42 rows=608\nloops=1)\n      -> Materialize (cost=795..795\nrows=540) (actual time=1.39..1.39 rows=608 loops=1)\n-> Group aggregate: count(0) (cost=671 rows=540) (actual\ntime=0.0571..1.33 rows=608 loops=1)\n      ->\nFilter: (facility.facility_type = 'Basketball') (cost=546 rows=540)\n(actual time=0.0543..1.19 rows=1484 loops=1)\n-> Covering index scan on Facility using idx_facility_complete\n(cost=546 rows=5396) (actual time=0.054..0.828 rows=5510 loops=1)\nScreenshot:
```

```
Starting Query 2: Parks With Most Basketball Courts\n\nEXPLAIN\n--> Limit: 15 row(s) (actual time=5.62..5.63 rows=15 loops=1)\n      -> Sort: basketball_court_count DESC (actual time=5.62..5.62 rows=15 loops=1)\n      -> Filter: ('count(f.facility_id)' >= (select #2)) (actual time=5.49..5.58 rows=222 loops=1)\n      -> Table scan on <temporary> (actual time=4.03..4.08 rows=608 loops=1)\n      -> Aggregate using temporary table (actual time=4.02..4.02 rows=608 loops=1)\n      -> Nested loop inner join (cost=735 rows=540) (actual time=0.0838..2.73 rows=1484 loops=1)\n      -> Covering index scan on f using idx_facility_complete (cost=546 rows=5396) (actual time=0.0701..0.996 rows=5510 loops=1)\n      -> Single-row index lookup on p using PRIMARY (park_id = f.park_id) (cost=0.25 rows=1) (actual time=769e-6..793e-6 rows=1 loops=1484)\n      -> Select #2 (subquery in condition; run only once)\n      -> Aggregate: avg(subquery.court_count) (cost=929..929 rows=1) (actual time=1.44..1.44 rows=1 loops=1)\n      -> Table scan on subquery (cost=795..804 rows=540) (actual time=1.39..1.42 rows=608 loops=1)\n      -> Materialize (cost=795..795 rows=540) (actual time=1.39..1.39 rows=608 loops=1)\n      -> Group aggregate: count(0) (cost=671 rows=540) (actual time=0.0571..1.33 rows=608 loops=1)\n      -> Covering index scan on Facility using idx_facility_complete (cost=546 rows=5396) (actual time=0.054..0.828 rows=5510 loops=1)\n
```

Query3

```
EXPLAIN
--> Limit: 15 row(s) (cost=1721..1722 rows=2) (actual time=6.28..6.28\nrows=2 loops=1)\n      -> Table scan on <union temporary>\n(cost=1721..1722 rows=2) (actual time=6.28..6.28 rows=2 loops=1)\n-> Union materialize with deduplication (cost=1720..1720 rows=2)\n(actual time=6.28..6.28 rows=2 loops=1)\n      -> Limit table size: 15 unique row(s)\n      -> Group aggregate:\n        avg(p.acres), avg(p.avg_rating), count(distinct p.park_id) (cost=860\nrows=1) (actual time=1.21..1.21 rows=1 loops=1)\n      ->
```

```

Nested loop inner join  (cost=735 rows=540) (actual time=0.0329..1.1
rows=338 loops=1)\n                                -> Filter: (f.is_lighted =
true)  (cost=546 rows=540) (actual time=0.0253..0.891 rows=338
loops=1)\n                                -> Covering index scan on f using
idx_facility_complete  (cost=546 rows=5396) (actual time=0.0217..0.748
rows=5510 loops=1)\n                                -> Single-row index lookup
on p using PRIMARY (park_id = f.park_id)  (cost=0.25 rows=1) (actual
time=510e-6..532e-6 rows=1 loops=338)\n                                -> Limit table size:
15 unique row(s)\n                                -> Group aggregate: avg(p.acres),
avg(p.avg_rating), count(distinct p.park_id)  (cost=860 rows=1) (actual
time=5.06..5.06 rows=1 loops=1)\n                                -> Nested loop
inner join  (cost=735 rows=540) (actual time=0.0167..2.9 rows=5172
loops=1)\n                                -> Filter: (f.is_lighted = false)
(cost=546 rows=540) (actual time=0.0133..0.931 rows=5172 loops=1)\n-> Covering index scan on f using idx_facility_complete  (cost=546
rows=5396) (actual time=0.0132..0.703 rows=5510 loops=1)\n-> Single-row index lookup on p using PRIMARY (park_id = f.park_id)
(cost=0.25 rows=1) (actual time=257e-6..277e-6 rows=1 loops=5172)\n
Screenshot:

```

```

Starting Query 3: Lighting Comparison
=====
EXPLAIN
Limit: 15 row(s)  (cost=1721..1722 rows=2) (actual time=6.28..6.28 rows=2 loops=1)\n      -> Table scan on union temporary  (cost=1721..1722 rows=2) (actual time=6.28..6.28 rows=2 loops=1)\n      -> Union materialize with deduplication
(cost=1720..1720 rows=2) (actual time=6.28..6.28 rows=2 loops=1)\n      -> Limit table size: 15 unique row(s)\n      -> Group aggregate: avg(p.acres), avg(p.avg_rating), count(distinct p.park_id)  (cost=860 rows=1) (actual time=0.0253..0.0253
rows=1 loops=1)\n      -> Nested loop inner join  (cost=735 rows=540) (actual time=0.0253..0.1 rows=338 loops=1)\n      -> Covering index scan on f using idx_facility_complete  (cost=546 rows=5396) (actual time=0.0217..0.748 rows=5510 loops=1)\n      -> Single-row index lookup on p using PRIMARY (park_id = f.park_id)  (cost=0.25 rows=1) (actual time=510e-6..532e-6 rows=1 loops=338)\n      -> Limit table size: 15 unique row(s)\n      -> Group aggregate: avg(p.acres), avg(p.avg_rating), count(distinct p.park_id)  (cost=860 rows=1) (actual time=6.06..6.06 rows=1 loops=1)\n      -> Nested loop inner join  (cost=735 rows=540) (actual time=6.06..6.06 rows=1 loops=1)\n      -> Covering index scan on f using idx_facility_complete  (cost=546 rows=5396) (actual time=6.06..6.06 rows=1 loops=1)\n      -> Single-row index lookup on p using PRIMARY (park_id = f.park_id)  (cost=0.25 rows=1) (actual time=257e-6..277e-6 rows=1 loops=5172)\n
```

Query4

```

EXPLAIN
-> Limit: 15 row(s)  (actual time=230..230 rows=15 loops=1)\n      ->
Sort: facility_types DESC, trail_count DESC, limit input to 15 row(s)
per chunk  (actual time=230..230 rows=15 loops=1)\n      -> Stream
results  (actual time=130..230 rows=34 loops=1)\n      -> Group
aggregate: count(distinct facility.facility_type), count(distinct
trail.trail_id)  (actual time=130..230 rows=34 loops=1)\n-> Sort: p.park_id, p.park_name, p.borough, p.acres  (actual
time=129..137 rows=114568 loops=1)\n      -> Stream
results  (cost=1.29 rows=2.47) (actual time=0.421..78.9 rows=114568
loops=1)\n      -> Nested loop inner join  (cost=1.29
rows=2.47) (actual time=0.417..35.3 rows=114568 loops=1)\n-> Nested loop inner join  (cost=0.7 rows=0.333) (actual time=0.41..3.4
rows=5290 loops=1)\n      -> Covering index
scan on t using idx_trail_park  (cost=0.35 rows=1) (actual
time=0.0114..0.804 rows=5346 loops=1)\n-> Filter: (p.acres > (select #2))  (cost=0.283 rows=0.333) (actual
time=335e-6..390e-6 rows=0.99 loops=5346)\n-> Single-row index lookup on p using PRIMARY (park_id = t.park_id)

```

```
(cost=0.283 rows=1) (actual time=126e-6..146e-6 rows=1 loops=5346)\n
-> Select #2 (subquery in condition; run only once)\n
--> Aggregate: avg(park.acres) (cost=631 rows=1) (actual\n
time=0.386..0.386 rows=1 loops=1)\n
-> Filter: (park.acres is not null) (cost=205 rows=1847) (actual\n
time=0.00792..0.309 rows=2052 loops=1)\n
-> Covering index scan on Park using idx_park_complete (cost=205\n
rows=2052) (actual time=0.00783..0.231 rows=2052 loops=1)\n
-> Covering index lookup on f using idx_facility_complete (park_id =\n
t.park_id) (cost=3.24 rows=7.42) (actual time=0.00172..0.00514\n
rows=21.7 loops=5290)\n
```

Screenshot:

```
Starting Query 4: Parks with Facilities and Trails
=====
EXPLAIN
+- Limit: 15 row(s) (actual time=238..238 rows=15 loops=1)      -> Sort: facility_types DESC, trail_count DESC, limit input to 15 row(s) per chunk (actual time=230..230 rows=15 loops=1)
s=34 (loops=1)\n      -> Group Aggregate: count(distinct facility.facility_type), count(distinct trail.trail_id) (actual time=138..238 rows=34 loops=1)\n      -> Stream results (cost=1..29 rows=1..29 loops=1)      -> Sort: p.park_name, p.borough, p.acres (actual time=129..137 rows=14568 loops=1)\n      -> Stream results (cost=1..29 rows=2..47) (actual time=0..421..78.9 rows=14568 loops=1)\n      -> Nested loop inner join (cost=1..29 rows=1..35.3 loops=1)      -> Nested loop inner join (cost=1..29 rows=2..47) (actual time=0..417..35.3 rows=2..47 loops=1)\n      -> Covering index scan on t using idx_park (cost=0..39 rows=1) (actual time=0..114..0.884 rows=5346 loops=1)\n      -> Filter: (p.acres > (select #2)) (cost=0..283 rows=0..333) (actual time=335e-6..398e-6 rows=0..99 loops=5346)\n      -> Covering index scan on p using PRIMARY (park_id = t.park_id) (cost=0..283 rows=1) (actual time=126e-6..146e-6 rows=1 loops=5346)\n      -> Filter: (p.acres is not null) (cost=205 rows=1847) (actual time=0.00792..0.309 rows=2052 loops=1)\n      -> Covering index scan on Park using idx_park_complete (cost=205 rows=2052) (actual time=0.00783..0.231 rows=2052 loops=1)\n      -> Covering index lookup on f using idx_facility_complete (park_id = t.park_id) (cost=3.24 rows=7.42) (actual time=0..00172..0.00514 rows=21.7 loops=5290)\n
```

Conclusion

Final Design

We mainly use the composite indexes (Index-2) for Q1, Q2, and Q4 since they give the best balance between speed and simplicity.

These indexes help SQL quickly find rows by combining the columns used in filters and joins, so queries don't have to scan the whole table.

For Q3, we go with the full covering design (Index-3) because it gives the lowest cost.

It includes all the columns the query needs, so the database can answer directly from the index without going back to the main table.

Overall, Index-2 works best for most queries, but Index-3 is the right choice for Q3 since it minimizes the optimizer's estimated cost.

```
CREATE INDEX idx_park_borough_rating ON Park(borough, avg_rating);

CREATE INDEX idx_park_acres_id ON Park(acres, park_id);

CREATE INDEX idx_facility_type_park ON Facility(facility_type, park_id);

CREATE INDEX idx_facility_lighted_park ON Facility(is_lighted, park_id);

CREATE INDEX idx_trail_park_id ON Trail(park_id);
```