```
postgres=# \dt
           List of relations
 Schema |   Name    | Type  |  Owner
--------+-----------+-------+----------
 public | climate   | table | postgres
 public | country   | table | postgres
 public | economy   | table | postgres
 public | energy    | table | postgres
 public | userinfo  | table | postgres
 public | userinput | table | postgres
(6 rows)

postgres=# SELECT COUNT(*) FROM Climate;
 count
-------
   195
(1 row)

postgres=# SELECT COUNT(*) FROM Country;
 count
-------
   195
(1 row)

postgres=# SELECT COUNT(*) FROM Economy;
 count
-------
   195
(1 row)

postgres=# SELECT COUNT(*) FROM Energy;
 count
-------
  1116
(1 row)

postgres=# SELECT COUNT(*) FROM UserInfo;
 count
-------
  1000
(1 row)

postgres=# SELECT COUNT(*) FROM UserInput;
 count
-------
  1500
(1 row)
```

DDL Commands

```
CREATE TABLE IF NOT EXISTS Country(
   CountryID INT PRIMARY KEY,
   Name VARCHAR(100) UNIQUE,
   Abbreviation VARCHAR(10),
   LandAreaKm2 DECIMAL,
   DensityPerKm2 DECIMAL,
   Population INT,
   CapitalCity VARCHAR(100),
   LargestCity VARCHAR(100),
   OfficialLanguage VARCHAR(100),
   LaborForceParticipationPercent DECIMAL,
   BirthRate DECIMAL,
   FertilityRate DECIMAL,
   InfantMortality DECIMAL,
   LifeExpectancy DECIMAL,
   MaternalMortalityRatio DECIMAL,
   UrbanPopulationPercent DECIMAL,
   PhysicianPerThousand DECIMAL,
   ArmedForcesSize INT,
   Latitude DECIMAL,
   Longitude DECIMAL,
   CallingCode VARCHAR(10)
);
-- @block

CREATE TABLE IF NOT EXISTS Climate(
   CountryID INT PRIMARY KEY,
   AgriculturalLandPercent DECIMAL,
   ForestedAreaPercent DECIMAL,
   CO2Emissions DECIMAL,
   FOREIGN KEY(CountryID) REFERENCES Country(CountryID)
      ON UPDATE CASCADE
      ON DELETE CASCADE
);
-- @block
```

```
CREATE TABLE IF NOT EXISTS Energy(
    CountryID INT,
    EnergyType VARCHAR(50),
    EnergyConsumption DECIMAL,
    EnergyProduction DECIMAL,
    PRIMARY KEY(CountryID, EnergyType),
    FOREIGN KEY(CountryID) REFERENCES Country(CountryID)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);
-- @block

CREATE TABLE IF NOT EXISTS Economy(
    CountryID INT PRIMARY KEY,
    GDP DECIMAL,
    CPI DECIMAL,
    CPIChangePercent DECIMAL,
    CurrencyCode VARCHAR(10),
    MinimumWage DECIMAL,
    UnemploymentRate DECIMAL,
    TaxRevenuePercent DECIMAL,
    TotalTaxRate DECIMAL,
    GasolinePrice DECIMAL,
    OutOfPocketHealthExpenditurePercent DECIMAL,
    GrossPrimaryEducationEnrollmentPercent DECIMAL,
    GrossTertiaryEducationEnrollmentPercent DECIMAL,
    FOREIGN KEY(CountryID) REFERENCES Country(CountryID)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);
-- @block

CREATE TABLE IF NOT EXISTS UserInfo(
    UserID INT PRIMARY KEY,
    Username VARCHAR(50),
    Password VARCHAR(50),
    Email VARCHAR(100),
    PrimaryCitizenshipID INT,
    FOREIGN KEY(PrimaryCitizenshipID) REFERENCES Country(CountryID)
        ON UPDATE CASCADE
```

```
        ON DELETE CASCADE
);
-- @block

CREATE TABLE IF NOT EXISTS UserInput(
    UserInputID INT PRIMARY KEY,
    UserID INT,
    CountryID INT,
    DateVisitedFrom DATE,
    DateVisitedTo DATE,
    FoodRating INT,
    HospitalRating INT,
    ClimateRating INT,
    TourismRating INT,
    SafetyRating INT,
    CostOfLivingRating INT,
    CultureEntertainmentRating INT,
    InfrastructureRating INT,
    HealthcareRating INT,
    Comments TEXT,
    FOREIGN KEY(UserID) REFERENCES UserInfo(UserID)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    FOREIGN KEY(CountryID) REFERENCES Country(CountryID)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);
```

Advanced Queries

Query 1: Country's Average Climate Rating for Users who have Visited that Country in a Specified Time Range

```
-- @block
SELECT Country.Name,
    AVG(ClimateRating) AS AvgClimateRating
FROM UserInput
    JOIN Country ON UserInput.CountryID = Country.CountryID
WHERE DateVisitedFrom > '2014-01-01'
    AND DateVisitedTo < '2018-01-01'
GROUP BY Country.Name
ORDER BY AvgClimateRating DESC
LIMIT 15;
```

Filter...	Filter...
Iran	10.0000000000000...
Jordan	8.00000000000000...
Djibouti	8.00000000000000...
Guyana	8.00000000000000...
Laos	8.00000000000000...
Vietnam	7.0000000000000000
Equatorial Guinea	7.0000000000000000
Guatemala	6.00000000000000...
Portugal	6.00000000000000...
Zimbabwe	6.00000000000000...
Norway	6.00000000000000...
Andorra	6.00000000000000...
Armenia	5.5000000000000000
Malta	5.0000000000000000
Morocco	5.0000000000000000

Query 2: Country's Energy Deficit

```
-- @block
SELECT Country.Name,
    SUM(EnergyConsumption) - SUM(EnergyProduction) AS EnergyDeficit
FROM Energy
    JOIN Country ON Energy.CountryID = Country.CountryID
GROUP BY Country.Name
ORDER BY EnergyDeficit DESC
LIMIT 15;
```

name	energydeficit
Filter...	Filter...
China	56.035052902759927
Japan	32.4264627055787027
India	27.9947349680113731
South Korea	21.5190575848369635
Germany	18.0087553320191273
Italy	10.5599045078900985
France	10.0417616590551523
Turkey	9.0136646048362490
Spain	8.4777375121123703
Singapore	7.1838404190217600
United Kingdom	5.2629888299687232
Thailand	5.2114761164270297
Netherlands	4.8136233206867087
Mexico	4.1532733029542178
Belgium	4.1215478520431245

Query 3: Users Favorite Country by Climate Rating

```sql
-- @block
SELECT UserInfo.Username,
    Country.Name
FROM (
        SELECT UserID,
            MAX(ClimateRating) AS maxclimaterating
        FROM UserInput
        GROUP BY UserID
    ) usermaxclimaterating
    JOIN UserInput ON usermaxclimaterating.UserID = UserInput.UserID
    AND usermaxclimaterating.maxclimaterating = UserInput.ClimateRating
    JOIN UserInfo ON UserInput.UserID = UserInfo.UserID
    JOIN Country ON UserInput.CountryID = Country.CountryID
ORDER BY UserInfo.Username
LIMIT 15;
```

username	name
Filter...	Filter...
aandreoneoq	Tajikistan
abiagioni1v	Iceland
abishopkh	Costa Rica
abishopkh	Belize
aboissierra	Belgium
abrachku	Singapore
abridgelandbg	Moldova
abrookton6q	Sierra Leone
acailgn	Kyrgyzstan
acalfgi	Germany
acaplinlq	Andorra
acasari6m	Belarus
aclericoates4c	Vietnam
acoatenee	Somalia
acosgreave6i	Honduras

Query 4: Countries with Greater than Global Average CO2 Emissions and greater than average forested area percentage

```sql
Run on active connection | Select block
-- @block
SELECT c.CountryID,
    c.Name,
    c.Population,
    c.LandAreaKm2,
    c.DensityPerKm2,
    cl.CO2Emissions
FROM Country c
    JOIN Climate cl ON c.CountryID = cl.CountryID
WHERE cl.CO2Emissions > (
        SELECT AVG(CO2Emissions)
        FROM Climate
    )
    AND cl.ForestedAreaPercent > (
        SELECT AVG(ForestedAreaPercent)
        FROM Climate
    )
ORDER BY cl.CO2Emissions DESC
LIMIT 15;
```

countryid	name	population	landareakm2	densityperkm2	co2emissions
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
186	United States	328239523	9833517.0	36.0	5006302.0
143	Russia	144373535	17098240.0	9.0	1732027.0
85	Japan	126226568	377944.0	347.0	1135886.0
64	Germany	83132799	357022.0	240.0	727973.0
162	South Korea	51709098	99720.0	527.0	620302.0
78	Indonesia	270203917	1904569.0	151.0	563325.0
32	Canada	36991981	9984670.0	4.0	544894.0
110	Mexico	126014024	1964375.0	66.0	486406.0
23	Brazil	212559417	8515770.0	25.0	462299.0
173	Thailand	69625582	513120.0	137.0	283763.0
103	Malaysia	32447385	329847.0	99.0	248289.0
164	Spain	47076781	505370.0	94.0	244002.0
191	Vietnam	96462106	331210.0	314.0	192668.0

Indexing Analysis

In the screenshots below, there are two cost values in the format cost1..cost2. Cost1 is the startup cost and cost2 is the total cost. We will be using cost2 for the analysis and ignore cost1.

Query 1 Before Indexing:

```
postgres=# EXPLAIN ANALYZE
SELECT Country.Name,
    AVG(ClimateRating) AS AvgClimateRating
FROM UserInput
    JOIN Country ON UserInput.CountryID = Country.CountryID
WHERE DateVisitedFrom > '2014-01-01'
    AND DateVisitedTo < '2018-01-01'
GROUP BY Country.Name
ORDER BY AvgClimateRating DESC
LIMIT 15;
                                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=54.58..54.62 rows=15 width=41) (actual time=0.624..0.631 rows=15 loops=1)
   ->  Sort  (cost=54.58..54.68 rows=38 width=41) (actual time=0.622..0.626 rows=15 loops=1)
         Sort Key: (avg(userinput.climaterating)) DESC
         Sort Method: top-N heapsort  Memory: 26kB
         ->  HashAggregate  (cost=53.18..53.65 rows=38 width=41) (actual time=0.556..0.585 rows=35 loops=1)
               Group Key: country.name
               Batches: 1  Memory Usage: 24kB
               ->  Hash Join  (cost=9.39..52.99 rows=38 width=13) (actual time=0.201..0.516 rows=36 loops=1)
                     Hash Cond: (userinput.countryid = country.countryid)
                     ->  Seq Scan on userinput  (cost=0.00..43.50 rows=38 width=8) (actual time=0.033..0.327 rows=36 loops=1)
                           Filter: ((datevisitedfrom > '2014-01-01'::date) AND (datevisitedto < '2018-01-01'::date))
                           Rows Removed by Filter: 1464
                     ->  Hash  (cost=6.95..6.95 rows=195 width=13) (actual time=0.157..0.158 rows=195 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 18kB
                           ->  Seq Scan on country  (cost=0.00..6.95 rows=195 width=13) (actual time=0.006..0.061 rows=195 loops=1)
 Planning Time: 0.429 ms
 Execution Time: 0.691 ms
(17 rows)
```

Indexing plan 1: index DateVisitedFROM

```
postgres=# EXPLAIN ANALYZE
SELECT Country.Name,
    AVG(ClimateRating) AS AvgClimateRating
FROM UserInput
    JOIN Country ON UserInput.CountryID = Country.CountryID
WHERE DateVisitedFrom > '2014-01-01'
    AND DateVisitedTo < '2018-01-01'
GROUP BY Country.Name
ORDER BY AvgClimateRating DESC
LIMIT 15;
                                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=42.74..42.78 rows=15 width=41) (actual time=0.396..0.401 rows=15 loops=1)
   ->  Sort  (cost=42.74..42.83 rows=38 width=41) (actual time=0.394..0.397 rows=15 loops=1)
         Sort Key: (avg(userinput.climaterating)) DESC
         Sort Method: top-N heapsort  Memory: 26kB
         ->  HashAggregate  (cost=41.33..41.81 rows=38 width=41) (actual time=0.329..0.357 rows=35 loops=1)
               Group Key: country.name
               Batches: 1  Memory Usage: 24kB
               ->  Hash Join  (cost=15.80..41.14 rows=38 width=13) (actual time=0.176..0.296 rows=36 loops=1)
                     Hash Cond: (userinput.countryid = country.countryid)
                     ->  Bitmap Heap Scan on userinput  (cost=6.41..31.65 rows=38 width=8) (actual time=0.048..0.151 rows=36 loops=1)
                           Recheck Cond: (datevisitedfrom > '2014-01-01'::date)
                           Filter: (datevisitedto < '2018-01-01'::date)
                           Rows Removed by Filter: 247
                           Heap Blocks: exact=21
                           ->  Bitmap Index Scan on datevisitedfromindex  (cost=0.00..6.40 rows=283 width=0) (actual time=0.022..0.023 rows=283 loops=1)
                                 Index Cond: (datevisitedfrom > '2014-01-01'::date)
                     ->  Hash  (cost=6.95..6.95 rows=195 width=13) (actual time=0.120..0.120 rows=195 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 18kB
                           ->  Seq Scan on country  (cost=0.00..6.95 rows=195 width=13) (actual time=0.006..0.054 rows=195 loops=1)
 Planning Time: 0.465 ms
 Execution Time: 0.467 ms
(21 rows)
```

Since DataVisitedFrom is being used in the WHERE condition, we created an index using:
CREATE INDEX IF NOT EXISTS DateVisitedFromIndex ON UserInput(DateVisitedFrom);

There is a bitmap index scan on datevisitedindex in the second query plan, and the cost of the UserInput scan reduced from 43.50 to 31.65. The index helped with the range query.

Indexing plan 2: Index DateVisitedTo

```
postgres=# EXPLAIN ANALYZE
SELECT Country.Name,
    AVG(ClimateRating) AS AvgClimateRating
FROM UserInput
    JOIN Country ON UserInput.CountryID = Country.CountryID
WHERE DateVisitedFrom > '2014-01-01'
    AND DateVisitedTo < '2018-01-01'
GROUP BY Country.Name
ORDER BY AvgClimateRating DESC
LIMIT 15;
                                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=40.87..40.91 rows=15 width=41) (actual time=0.387..0.393 rows=15 loops=1)
   ->  Sort  (cost=40.87..40.97 rows=38 width=41) (actual time=0.385..0.389 rows=15 loops=1)
         Sort Key: (avg(userinput.climaterating)) DESC
         Sort Method: top-N heapsort  Memory: 26kB
         ->  HashAggregate  (cost=39.46..39.94 rows=38 width=41) (actual time=0.320..0.348 rows=35 loops=1)
               Group Key: country.name
               Batches: 1  Memory Usage: 24kB
               ->  Hash Join  (cost=15.17..39.27 rows=38 width=13) (actual time=0.179..0.287 rows=36 loops=1)
                     Hash Cond: (userinput.countryid = country.countryid)
                     ->  Bitmap Heap Scan on userinput  (cost=5.79..29.79 rows=38 width=8) (actual time=0.050..0.141 rows=36 loops=1)
                           Recheck Cond: (datevisitedto < '2018-01-01'::date)
                           Filter: (datevisitedfrom > '2014-01-01'::date)
                           Rows Removed by Filter: 164
                           Heap Blocks: exact=20
                           ->  Bitmap Index Scan on datevisitedtoindex  (cost=0.00..5.78 rows=200 width=0) (actual time=0.026..0.027 rows=200 loops=1)
                                 Index Cond: (datevisitedto < '2018-01-01'::date)
                     ->  Hash  (cost=6.95..6.95 rows=195 width=13) (actual time=0.120..0.120 rows=195 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 18kB
                           ->  Seq Scan on country  (cost=0.00..6.95 rows=195 width=13) (actual time=0.006..0.055 rows=195 loops=1)
 Planning Time: 0.525 ms
 Execution Time: 0.483 ms
(21 rows)
```

DateVisitedTo is being used in the WHERE condition, so we can create another index on it. The query plan switches to using the new datevisitedtoindex instead of using datevisitedfromindex and the cost is further lowered from 31.65 to 29.79. It looks like the query optimiser can choose the right index to use based on the query. The optimiser may use datevisitedfromindex for other queries. So it is best to use both indexes and let the optimizer choose the index to use based on individual queries.

Indexing plan 3: Index CountryName

```
postgres=# EXPLAIN ANALYZE
SELECT Country.Name,
    AVG(ClimateRating) AS AvgClimateRating
FROM UserInput
    JOIN Country ON UserInput.CountryID = Country.CountryID
WHERE DateVisitedFrom > '2014-01-01'
    AND DateVisitedTo < '2018-01-01'
GROUP BY Country.Name
ORDER BY AvgClimateRating DESC
LIMIT 15;
                                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=40.87..40.91 rows=15 width=41) (actual time=0.408..0.415 rows=15 loops=1)
   ->  Sort  (cost=40.87..40.97 rows=38 width=41) (actual time=0.406..0.410 rows=15 loops=1)
         Sort Key: (avg(userinput.climaterating)) DESC
         Sort Method: top-N heapsort  Memory: 26kB
         ->  HashAggregate  (cost=39.46..39.94 rows=38 width=41) (actual time=0.336..0.367 rows=35 loops=1)
               Group Key: country.name
               Batches: 1  Memory Usage: 24kB
               ->  Hash Join  (cost=15.17..39.27 rows=38 width=13) (actual time=0.186..0.300 rows=36 loops=1)
                     Hash Cond: (userinput.countryid = country.countryid)
                     ->  Bitmap Heap Scan on userinput  (cost=5.79..29.79 rows=38 width=8) (actual time=0.045..0.139 rows=36 loops=1)
                           Recheck Cond: (datevisitedto < '2018-01-01'::date)
                           Filter: (datevisitedfrom > '2014-01-01'::date)
                           Rows Removed by Filter: 164
                           Heap Blocks: exact=20
                           ->  Bitmap Index Scan on datevisitedtoindex  (cost=0.00..5.78 rows=200 width=0) (actual time=0.019..0.019 rows=200 loops=1)
                                 Index Cond: (datevisitedto < '2018-01-01'::date)
                     ->  Hash  (cost=6.95..6.95 rows=195 width=13) (actual time=0.132..0.133 rows=195 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 18kB
                           ->  Seq Scan on country  (cost=0.00..6.95 rows=195 width=13) (actual time=0.007..0.060 rows=195 loops=1)
 Planning Time: 0.611 ms
 Execution Time: 0.486 ms
(21 rows)
```

Since country name is used in the group by clause, we applied an index on Country(Name). However, the HashAggregate step using the group key country name did not use the index and

the cost remained the same at 39.94 for HashAggregate. The query optimizer is already using hashing to speed up the aggregation, so most likely the index was not needed.

Query 2 before indexing:

```
postgres=# EXPLAIN ANALYZE
SELECT Country.Name,
    SUM(EnergyConsumption) - SUM(EnergyProduction) AS EnergyDeficit
FROM Energy
    JOIN Country ON Energy.CountryID = Country.CountryID
GROUP BY Country.Name
ORDER BY EnergyDeficit DESC
LIMIT 15;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=49.11..49.15 rows=15 width=41) (actual time=2.169..2.176 rows=15 loops=1)
   ->  Sort  (cost=49.11..49.60 rows=195 width=41) (actual time=2.167..2.171 rows=15 loops=1)
         Sort Key: ((sum(energy.energyconsumption) - sum(energy.energyproduction))) DESC
         Sort Method: top-N heapsort  Memory: 26kB
         ->  HashAggregate  (cost=40.91..44.33 rows=195 width=41) (actual time=1.763..1.996 rows=186 loops=1)
               Group Key: country.name
               Batches: 1  Memory Usage: 288kB
               ->  Hash Join  (cost=9.39..32.54 rows=1116 width=25) (actual time=0.137..0.811 rows=1116 loops=1)
                     Hash Cond: (energy.countryid = country.countryid)
                     ->  Seq Scan on energy  (cost=0.00..20.16 rows=1116 width=20) (actual time=0.007..0.160 rows=1116 loops=1)
                     ->  Hash  (cost=6.95..6.95 rows=195 width=13) (actual time=0.119..0.120 rows=195 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 18kB
                           ->  Seq Scan on country  (cost=0.00..6.95 rows=195 width=13) (actual time=0.005..0.054 rows=195 loops=1)
 Planning Time: 0.699 ms
 Execution Time: 2.242 ms
(15 rows)
```

Indexing plan 1: index Country.Name

```
postgres=# EXPLAIN ANALYZE
SELECT Country.Name,
    SUM(EnergyConsumption) - SUM(EnergyProduction) AS EnergyDeficit
FROM Energy
    JOIN Country ON Energy.CountryID = Country.CountryID
GROUP BY Country.Name
ORDER BY EnergyDeficit DESC
LIMIT 15;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=49.11..49.15 rows=15 width=41) (actual time=2.556..2.563 rows=15 loops=1)
   ->  Sort  (cost=49.11..49.60 rows=195 width=41) (actual time=2.554..2.558 rows=15 loops=1)
         Sort Key: ((sum(energy.energyconsumption) - sum(energy.energyproduction))) DESC
         Sort Method: top-N heapsort  Memory: 26kB
         ->  HashAggregate  (cost=40.91..44.33 rows=195 width=41) (actual time=2.125..2.374 rows=186 loops=1)
               Group Key: country.name
               Batches: 1  Memory Usage: 288kB
               ->  Hash Join  (cost=9.39..32.54 rows=1116 width=25) (actual time=0.149..0.954 rows=1116 loops=1)
                     Hash Cond: (energy.countryid = country.countryid)
                     ->  Seq Scan on energy  (cost=0.00..20.16 rows=1116 width=20) (actual time=0.008..0.183 rows=1116 loops=1)
                     ->  Hash  (cost=6.95..6.95 rows=195 width=13) (actual time=0.131..0.132 rows=195 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 18kB
                           ->  Seq Scan on country  (cost=0.00..6.95 rows=195 width=13) (actual time=0.006..0.058 rows=195 loops=1)
 Planning Time: 0.705 ms
 Execution Time: 2.667 ms
(15 rows)
```

Since country name is used in the group by clause, we applied an index on Country(Name). However, the HashAggregate step using the group key country name did not use the index and the cost remained the same at 44.33 for HashAggregate. The query optimizer is already using hashing to speed up the aggregation, so most likely the index was not needed.

Indexing plan 2: index Energy.CountryID

```
postgres=# EXPLAIN ANALYZE
SELECT Country.Name,
    SUM(EnergyConsumption) - SUM(EnergyProduction) AS EnergyDeficit
FROM Energy
    JOIN Country ON Energy.CountryID = Country.CountryID
GROUP BY Country.Name
ORDER BY EnergyDeficit DESC
LIMIT 15;
                                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=49.11..49.15 rows=15 width=41) (actual time=2.224..2.230 rows=15 loops=1)
   ->  Sort  (cost=49.11..49.60 rows=195 width=41) (actual time=2.222..2.226 rows=15 loops=1)
         Sort Key: ((sum(energy.energyconsumption) - sum(energy.energyproduction))) DESC
         Sort Method: top-N heapsort  Memory: 26kB
         ->  HashAggregate  (cost=40.91..44.33 rows=195 width=41) (actual time=1.814..2.052 rows=186 loops=1)
               Group Key: country.name
               Batches: 1  Memory Usage: 288kB
               ->  Hash Join  (cost=9.39..32.54 rows=1116 width=25) (actual time=0.167..0.828 rows=1116 loops=1)
                     Hash Cond: (energy.countryid = country.countryid)
                     ->  Seq Scan on energy  (cost=0.00..20.16 rows=1116 width=20) (actual time=0.007..0.158 rows=1116 loops=1)
                     ->  Hash  (cost=6.95..6.95 rows=195 width=13) (actual time=0.150..0.151 rows=195 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 18kB
                           ->  Seq Scan on country  (cost=0.00..6.95 rows=195 width=13) (actual time=0.006..0.066 rows=195 loops=1)
 Planning Time: 0.720 ms
 Execution Time: 2.289 ms
(15 rows)
```

Currently, the primary key in the energy table is on (CountryID, EnergyType), so we created an individual index on countryID to speed up the join, but the join cost remained at 32.54. Since there is a hash join already being used, the index was not needed.

Indexing plan 3: Index EnergyConsumption and EnergyProduction

```
postgres=# EXPLAIN ANALYZE
SELECT Country.Name,
    SUM(EnergyConsumption) - SUM(EnergyProduction) AS EnergyDeficit
FROM Energy
    JOIN Country ON Energy.CountryID = Country.CountryID
GROUP BY Country.Name
ORDER BY EnergyDeficit DESC
LIMIT 15;
                                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=49.11..49.15 rows=15 width=41) (actual time=2.448..2.454 rows=15 loops=1)
   ->  Sort  (cost=49.11..49.60 rows=195 width=41) (actual time=2.446..2.450 rows=15 loops=1)
         Sort Key: ((sum(energy.energyconsumption) - sum(energy.energyproduction))) DESC
         Sort Method: top-N heapsort  Memory: 26kB
         ->  HashAggregate  (cost=40.91..44.33 rows=195 width=41) (actual time=2.022..2.271 rows=186 loops=1)
               Group Key: country.name
               Batches: 1  Memory Usage: 288kB
               ->  Hash Join  (cost=9.39..32.54 rows=1116 width=25) (actual time=0.164..0.961 rows=1116 loops=1)
                     Hash Cond: (energy.countryid = country.countryid)
                     ->  Seq Scan on energy  (cost=0.00..20.16 rows=1116 width=20) (actual time=0.007..0.175 rows=1116 loops=1)
                     ->  Hash  (cost=6.95..6.95 rows=195 width=13) (actual time=0.147..0.147 rows=195 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 18kB
                           ->  Seq Scan on country  (cost=0.00..6.95 rows=195 width=13) (actual time=0.006..0.061 rows=195 loops=1)
 Planning Time: 0.910 ms
 Execution Time: 2.511 ms
(15 rows)
```

There were no other attributes in the group and where clauses to apply an index, so we tried to see if the index on EnergyConsumption and EnergyProduction would speed it up, but as expected, the total cost remained the same at 49.15.

Query 3 before indexing:

```
postgres=# EXPLAIN ANALYZE SELECT UserInfo.Username,
    Country.Name
FROM (
        SELECT UserID,
            MAX(ClimateRating) AS maxclimaterating
        FROM UserInput
        GROUP BY UserID
    ) usermaxclimaterating
    JOIN UserInput ON usermaxclimaterating.UserID = UserInput.UserID
    AND usermaxclimaterating.maxclimaterating = UserInput.ClimateRating
    JOIN UserInfo ON UserInput.UserID = UserInfo.UserID
    JOIN Country ON UserInput.CountryID = Country.CountryID
ORDER BY UserInfo.Username
LIMIT 15;
                                                                          QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=119.16..119.18 rows=8 width=19) (actual time=7.411..7.420 rows=15 loops=1)
   ->  Sort  (cost=119.16..119.18 rows=8 width=19) (actual time=7.409..7.415 rows=15 loops=1)
         Sort Key: userinfo.username
         Sort Method: top-N heapsort  Memory: 27kB
         ->  Nested Loop  (cost=71.12..119.04 rows=8 width=19) (actual time=1.449..6.815 rows=823 loops=1)
               ->  Nested Loop  (cost=70.97..117.59 rows=8 width=14) (actual time=1.443..4.923 rows=823 loops=1)
                     Join Filter: (userinput_1.userid = userinfo.userid)
                     ->  Hash Join  (cost=70.70..114.57 rows=8 width=12) (actual time=1.428..2.442 rows=823 loops=1)
                           Hash Cond: ((userinput.userid = userinput_1.userid) AND (userinput.climaterating = (max(userinput_1.climaterating))))
                           ->  Seq Scan on userinput  (cost=0.00..36.00 rows=1500 width=12) (actual time=0.009..0.248 rows=1500 loops=1)
                           ->  Hash  (cost=59.04..59.04 rows=777 width=8) (actual time=1.408..1.409 rows=777 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 39kB
                                 ->  HashAggregate  (cost=43.50..51.27 rows=777 width=8) (actual time=1.027..1.198 rows=777 loops=1)
                                       Group Key: userinput_1.userid
                                       Batches: 1  Memory Usage: 105kB
                                       ->  Seq Scan on userinput userinput_1  (cost=0.00..36.00 rows=1500 width=8) (actual time=0.003..0.222 rows=1500 loops=1)
                     ->  Index Scan using userinfo_pkey on userinfo  (cost=0.28..0.36 rows=1 width=14) (actual time=0.002..0.002 rows=1 loops=823)
                           Index Cond: (userid = userinput.userid)
               ->  Index Scan using country_pkey on country  (cost=0.14..0.18 rows=1 width=13) (actual time=0.002..0.002 rows=1 loops=823)
                     Index Cond: (countryid = userinput.countryid)
 Planning Time: 0.952 ms
 Execution Time: 7.507 ms
(22 rows)
```

Indexing plan 1: indexing UserInput(ClimateRating)

```
postgres=# EXPLAIN ANALYZE SELECT UserInfo.Username,
    Country.Name
FROM (
        SELECT UserID,
            MAX(ClimateRating) AS maxclimaterating
        FROM UserInput
        GROUP BY UserID
    ) usermaxclimaterating
    JOIN UserInput ON usermaxclimaterating.UserID = UserInput.UserID
    AND usermaxclimaterating.maxclimaterating = UserInput.ClimateRating
    JOIN UserInfo ON UserInput.UserID = UserInfo.UserID
    JOIN Country ON UserInput.CountryID = Country.CountryID
ORDER BY UserInfo.Username
LIMIT 15;
                                                                          QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=119.16..119.18 rows=8 width=19) (actual time=8.154..8.164 rows=15 loops=1)
   ->  Sort  (cost=119.16..119.18 rows=8 width=19) (actual time=8.152..8.159 rows=15 loops=1)
         Sort Key: userinfo.username
         Sort Method: top-N heapsort  Memory: 27kB
         ->  Nested Loop  (cost=71.12..119.04 rows=8 width=19) (actual time=1.640..7.483 rows=823 loops=1)
               ->  Nested Loop  (cost=70.97..117.59 rows=8 width=14) (actual time=1.632..5.444 rows=823 loops=1)
                     Join Filter: (userinput_1.userid = userinfo.userid)
                     ->  Hash Join  (cost=70.70..114.57 rows=8 width=12) (actual time=1.611..2.761 rows=823 loops=1)
                           Hash Cond: ((userinput.userid = userinput_1.userid) AND (userinput.climaterating = (max(userinput_1.climaterating))))
                           ->  Seq Scan on userinput  (cost=0.00..36.00 rows=1500 width=12) (actual time=0.010..0.262 rows=1500 loops=1)
                           ->  Hash  (cost=59.04..59.04 rows=777 width=8) (actual time=1.590..1.593 rows=777 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 39kB
                                 ->  HashAggregate  (cost=43.50..51.27 rows=777 width=8) (actual time=1.120..1.318 rows=777 loops=1)
                                       Group Key: userinput_1.userid
                                       Batches: 1  Memory Usage: 105kB
                                       ->  Seq Scan on userinput userinput_1  (cost=0.00..36.00 rows=1500 width=8) (actual time=0.002..0.236 rows=1500 loops=1)
                     ->  Index Scan using userinfo_pkey on userinfo  (cost=0.28..0.36 rows=1 width=14) (actual time=0.002..0.002 rows=1 loops=823)
                           Index Cond: (userid = userinput.userid)
               ->  Index Scan using country_pkey on country  (cost=0.14..0.18 rows=1 width=13) (actual time=0.002..0.002 rows=1 loops=823)
                     Index Cond: (countryid = userinput.countryid)
 Planning Time: 1.208 ms
 Execution Time: 8.253 ms
(22 rows)
```

Since climaterating is being used in a join, we applied an index, but the total cost remains at 119.18. It looks like sequential scan needs to be done to scan through the unindexed userId. As the sequential scan is being used, the index is not used.

Indexing plan 2: indexing UserInput(UserID)

```
postgres=# EXPLAIN ANALYZE SELECT UserInfo.Username,
    Country.Name
FROM (
        SELECT UserID,
            MAX(ClimateRating) AS maxclimaterating
        FROM UserInput
        GROUP BY UserID
    ) usermaxclimaterating
    JOIN UserInput ON usermaxclimaterating.UserID = UserInput.UserID
    AND usermaxclimaterating.maxclimaterating = UserInput.ClimateRating
    JOIN UserInfo ON UserInput.UserID = UserInfo.UserID
    JOIN Country ON UserInput.CountryID = Country.CountryID
ORDER BY UserInfo.Username
LIMIT 15;
                                                                            QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=119.16..119.18 rows=8 width=19) (actual time=7.148..7.157 rows=15 loops=1)
   ->  Sort  (cost=119.16..119.18 rows=8 width=19) (actual time=7.146..7.152 rows=15 loops=1)
         Sort Key: userinfo.username
         Sort Method: top-N heapsort  Memory: 27kB
         ->  Nested Loop  (cost=71.12..119.04 rows=8 width=19) (actual time=1.539..6.585 rows=823 loops=1)
               ->  Nested Loop  (cost=70.97..117.59 rows=8 width=14) (actual time=1.532..4.796 rows=823 loops=1)
                     Join Filter: (userinput_1.userid = userinfo.userid)
                     ->  Hash Join  (cost=70.70..114.57 rows=8 width=12) (actual time=1.513..2.440 rows=823 loops=1)
                           Hash Cond: ((userinput.userid = userinput_1.userid) AND (userinput.climaterating = (max(userinput_1.climaterating))))
                           ->  Seq Scan on userinput  (cost=0.00..36.00 rows=1500 width=12) (actual time=0.012..0.227 rows=1500 loops=1)
                           ->  Hash  (cost=59.04..59.04 rows=777 width=8) (actual time=1.490..1.491 rows=777 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 39kB
                                 ->  HashAggregate  (cost=43.50..51.27 rows=777 width=8) (actual time=1.065..1.251 rows=777 loops=1)
                                       Group Key: userinput_1.userid
                                       Batches: 1  Memory Usage: 105kB
                                       ->  Seq Scan on userinput userinput_1  (cost=0.00..36.00 rows=1500 width=8) (actual time=0.003..0.226 rows=1500 loops=1)
                     ->  Index Scan using userinfo_pkey on userinfo  (cost=0.28..0.36 rows=1 width=14) (actual time=0.002..0.002 rows=1 loops=823)
                           Index Cond: (userid = userinput.userid)
               ->  Index Scan using country_pkey on country  (cost=0.14..0.18 rows=1 width=13) (actual time=0.002..0.002 rows=1 loops=823)
                     Index Cond: (countryid = userinput.countryid)
 Planning Time: 1.242 ms
 Execution Time: 7.241 ms
(22 rows)
```

Since userid is being used in a join, we applied an index, but the total cost remains at 119.18. It looks like sequential scan needs to be done to scan through the unindexed climate rating. As the sequential scan is being used, the index is not used.

Indexing plan 3: indexing UserInput(UserID) and UserInput(ClimateRating)

```
postgres=# EXPLAIN ANALYZE SELECT UserInfo.Username,
    Country.Name
FROM (
        SELECT UserID,
            MAX(ClimateRating) AS maxclimaterating
        FROM UserInput
        GROUP BY UserID
    ) usermaxclimaterating
    JOIN UserInput ON usermaxclimaterating.UserID = UserInput.UserID
    AND usermaxclimaterating.maxclimaterating = UserInput.ClimateRating
    JOIN UserInfo ON UserInput.UserID = UserInfo.UserID
    JOIN Country ON UserInput.CountryID = Country.CountryID
ORDER BY UserInfo.Username
LIMIT 15;
                                                                            QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=119.16..119.18 rows=8 width=19) (actual time=9.814..9.824 rows=15 loops=1)
   ->  Sort  (cost=119.16..119.18 rows=8 width=19) (actual time=9.811..9.819 rows=15 loops=1)
         Sort Key: userinfo.username
         Sort Method: top-N heapsort  Memory: 27kB
         ->  Nested Loop  (cost=71.12..119.04 rows=8 width=19) (actual time=1.491..9.026 rows=823 loops=1)
               ->  Nested Loop  (cost=70.97..117.59 rows=8 width=14) (actual time=1.484..6.332 rows=823 loops=1)
                     Join Filter: (userinput_1.userid = userinfo.userid)
                     ->  Hash Join  (cost=70.70..114.57 rows=8 width=12) (actual time=1.469..2.910 rows=823 loops=1)
                           Hash Cond: ((userinput.userid = userinput_1.userid) AND (userinput.climaterating = (max(userinput_1.climaterating))))
                           ->  Seq Scan on userinput  (cost=0.00..36.00 rows=1500 width=12) (actual time=0.007..0.330 rows=1500 loops=1)
                           ->  Hash  (cost=59.04..59.04 rows=777 width=8) (actual time=1.451..1.453 rows=777 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 39kB
                                 ->  HashAggregate  (cost=43.50..51.27 rows=777 width=8) (actual time=1.037..1.224 rows=777 loops=1)
                                       Group Key: userinput_1.userid
                                       Batches: 1  Memory Usage: 105kB
                                       ->  Seq Scan on userinput userinput_1  (cost=0.00..36.00 rows=1500 width=8) (actual time=0.002..0.219 rows=1500 loops=1)
                     ->  Index Scan using userinfo_pkey on userinfo  (cost=0.28..0.36 rows=1 width=14) (actual time=0.003..0.003 rows=1 loops=823)
                           Index Cond: (userid = userinput.userid)
               ->  Index Scan using country_pkey on country  (cost=0.14..0.18 rows=1 width=13) (actual time=0.002..0.002 rows=1 loops=823)
                     Index Cond: (countryid = userinput.countryid)
 Planning Time: 1.118 ms
 Execution Time: 9.921 ms
(22 rows)
```

Indexing both userid and climaterating should have prevented the sequential scan, but the cost remained at 119.18. The query optimizer might have chosen to do a sequential scan as it is already doing one in user_input_1, so it might be reused in the where clause as well.

Query 4 before indexing:

```
postgres=# EXPLAIN ANALYZE SELECT c.CountryID,
    c.Name,
    c.Population,
    c.LandAreaKm2,
    c.DensityPerKm2,
    cl.CO2Emissions
FROM Country c
    JOIN Climate cl ON c.CountryID = cl.CountryID
WHERE cl.CO2Emissions > (
        SELECT AVG(CO2Emissions)
        FROM Climate
    )
    AND cl.ForestedAreaPercent > (
        SELECT AVG(ForestedAreaPercent)
        FROM Climate
    )
ORDER BY cl.CO2Emissions DESC
LIMIT 15;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=22.06..22.10 rows=15 width=34) (actual time=0.457..0.465 rows=13 loops=1)
   InitPlan 1 (returns $0)
     ->  Aggregate  (cost=4.44..4.45 rows=1 width=32) (actual time=0.137..0.138 rows=1 loops=1)
           ->  Seq Scan on climate  (cost=0.00..3.95 rows=195 width=6) (actual time=0.006..0.044 rows=195 loops=1)
   InitPlan 2 (returns $1)
     ->  Aggregate  (cost=4.44..4.45 rows=1 width=32) (actual time=0.087..0.087 rows=1 loops=1)
           ->  Seq Scan on climate climate_1  (cost=0.00..3.95 rows=195 width=6) (actual time=0.005..0.030 rows=195 loops=1)
   ->  Sort  (cost=13.16..13.22 rows=22 width=34) (actual time=0.455..0.459 rows=13 loops=1)
         Sort Key: cl.co2emissions DESC
         Sort Method: quicksort  Memory: 26kB
         ->  Hash Join  (cost=5.20..12.67 rows=22 width=34) (actual time=0.366..0.434 rows=13 loops=1)
               Hash Cond: (c.countryid = cl.countryid)
               ->  Seq Scan on country c  (cost=0.00..6.95 rows=195 width=28) (actual time=0.011..0.045 rows=195 loops=1)
               ->  Hash  (cost=4.92..4.92 rows=22 width=10) (actual time=0.335..0.335 rows=13 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on climate cl  (cost=0.00..4.92 rows=22 width=10) (actual time=0.252..0.327 rows=13 loops=1)
                           Filter: ((co2emissions > $0) AND (forestedareapercent > $1))
                           Rows Removed by Filter: 182
 Planning Time: 0.497 ms
 Execution Time: 0.529 ms
(20 rows)
```

Indexing plan 1: indexing co2emissions

```
postgres=# EXPLAIN ANALYZE SELECT c.CountryID,
    c.Name,
    c.Population,
    c.LandAreaKm2,
    c.DensityPerKm2,
    cl.CO2Emissions
FROM Country c
    JOIN Climate cl ON c.CountryID = cl.CountryID
WHERE cl.CO2Emissions > (
        SELECT AVG(CO2Emissions)
        FROM Climate
    )
    AND cl.ForestedAreaPercent > (
        SELECT AVG(ForestedAreaPercent)
        FROM Climate
    )
ORDER BY cl.CO2Emissions DESC
LIMIT 15;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=22.06..22.10 rows=15 width=34) (actual time=0.402..0.408 rows=13 loops=1)
   InitPlan 1 (returns $0)
     ->  Aggregate  (cost=4.44..4.45 rows=1 width=32) (actual time=0.092..0.093 rows=1 loops=1)
           ->  Seq Scan on climate  (cost=0.00..3.95 rows=195 width=6) (actual time=0.004..0.029 rows=195 loops=1)
   InitPlan 2 (returns $1)
     ->  Aggregate  (cost=4.44..4.45 rows=1 width=32) (actual time=0.079..0.079 rows=1 loops=1)
           ->  Seq Scan on climate climate_1  (cost=0.00..3.95 rows=195 width=6) (actual time=0.004..0.028 rows=195 loops=1)
   ->  Sort  (cost=13.16..13.22 rows=22 width=34) (actual time=0.400..0.403 rows=13 loops=1)
         Sort Key: cl.co2emissions DESC
         Sort Method: quicksort  Memory: 26kB
         ->  Hash Join  (cost=5.20..12.67 rows=22 width=34) (actual time=0.292..0.353 rows=13 loops=1)
               Hash Cond: (c.countryid = cl.countryid)
               ->  Seq Scan on country c  (cost=0.00..6.95 rows=195 width=28) (actual time=0.007..0.038 rows=195 loops=1)
               ->  Hash  (cost=4.92..4.92 rows=22 width=10) (actual time=0.269..0.269 rows=13 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on climate cl  (cost=0.00..4.92 rows=22 width=10) (actual time=0.193..0.262 rows=13 loops=1)
                           Filter: ((co2emissions > $0) AND (forestedareapercent > $1))
                           Rows Removed by Filter: 182
 Planning Time: 0.673 ms
 Execution Time: 0.489 ms
(20 rows)
```

Since co2emissions is being used in the where and order by clause, we created an index on it, however, the total cost remains at 22.10. It looks like the query optimizer found a sequential scan

more performant. The table is less than 200 rows, so the overhead of using the index might have made it slower than just a sequential scan.

Indexing plan 2: indexing forested area percent

```
postgres=# EXPLAIN ANALYZE SELECT c.CountryID,
    c.Name,
    c.Population,
    c.LandAreaKm2,
    c.DensityPerKm2,
    cl.CO2Emissions
FROM Country c
    JOIN Climate cl ON c.CountryID = cl.CountryID
WHERE cl.CO2Emissions > (
        SELECT AVG(CO2Emissions)
        FROM Climate
    )
    AND cl.ForestedAreaPercent > (
        SELECT AVG(ForestedAreaPercent)
        FROM Climate
    )
ORDER BY cl.CO2Emissions DESC
LIMIT 15;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=22.06..22.10 rows=15 width=34) (actual time=0.447..0.454 rows=13 loops=1)
   InitPlan 1 (returns $0)
     ->  Aggregate  (cost=4.44..4.45 rows=1 width=32) (actual time=0.101..0.102 rows=1 loops=1)
           ->  Seq Scan on climate  (cost=0.00..3.95 rows=195 width=6) (actual time=0.004..0.032 rows=195 loops=1)
   InitPlan 2 (returns $1)
     ->  Aggregate  (cost=4.44..4.45 rows=1 width=32) (actual time=0.103..0.103 rows=1 loops=1)
           ->  Seq Scan on climate climate_1  (cost=0.00..3.95 rows=195 width=6) (actual time=0.004..0.030 rows=195 loops=1)
   ->  Sort  (cost=13.16..13.22 rows=22 width=34) (actual time=0.445..0.448 rows=13 loops=1)
         Sort Key: cl.co2emissions DESC
         Sort Method: quicksort  Memory: 26kB
         ->  Hash Join  (cost=5.20..12.67 rows=22 width=34) (actual time=0.337..0.404 rows=13 loops=1)
               Hash Cond: (c.countryid = cl.countryid)
               ->  Seq Scan on country c  (cost=0.00..6.95 rows=195 width=28) (actual time=0.008..0.041 rows=195 loops=1)
               ->  Hash  (cost=4.92..4.92 rows=22 width=10) (actual time=0.312..0.313 rows=13 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on climate cl  (cost=0.00..4.92 rows=22 width=10) (actual time=0.230..0.305 rows=13 loops=1)
                           Filter: ((co2emissions > $0) AND (forestedareapercent > $1))
                           Rows Removed by Filter: 182
 Planning Time: 0.722 ms
 Execution Time: 0.514 ms
(20 rows)
```

Since forested area percent is used in the where condition, an index might have sped the query up, but the cost remains at 22.10. Again, this is most likely that the table is less than 200 rows and a sequential scan was fast enough.

Indexing plan 3: indexing both forested area percent and co2 emissions

```
postgres=# EXPLAIN ANALYZE SELECT c.CountryID,
    c.Name,
    c.Population,
    c.LandAreaKm2,
    c.DensityPerKm2,
    cl.CO2Emissions
FROM Country c
    JOIN Climate cl ON c.CountryID = cl.CountryID
WHERE cl.CO2Emissions > (
        SELECT AVG(CO2Emissions)
        FROM Climate
    )
    AND cl.ForestedAreaPercent > (
        SELECT AVG(ForestedAreaPercent)
        FROM Climate
    )
ORDER BY cl.CO2Emissions DESC
LIMIT 15;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=22.06..22.10 rows=15 width=34) (actual time=0.411..0.418 rows=13 loops=1)
   InitPlan 1 (returns $0)
     ->  Aggregate  (cost=4.44..4.45 rows=1 width=32) (actual time=0.101..0.102 rows=1 loops=1)
           ->  Seq Scan on climate  (cost=0.00..3.95 rows=195 width=6) (actual time=0.004..0.032 rows=195 loops=1)
   InitPlan 2 (returns $1)
     ->  Aggregate  (cost=4.44..4.45 rows=1 width=32) (actual time=0.086..0.086 rows=1 loops=1)
           ->  Seq Scan on climate climate_1  (cost=0.00..3.95 rows=195 width=6) (actual time=0.004..0.030 rows=195 loops=1)
   ->  Sort  (cost=13.16..13.22 rows=22 width=34) (actual time=0.409..0.413 rows=13 loops=1)
         Sort Key: cl.co2emissions DESC
         Sort Method: quicksort  Memory: 26kB
         ->  Hash Join  (cost=5.20..12.67 rows=22 width=34) (actual time=0.322..0.389 rows=13 loops=1)
               Hash Cond: (c.countryid = cl.countryid)
               ->  Seq Scan on country c  (cost=0.00..6.95 rows=195 width=28) (actual time=0.008..0.042 rows=195 loops=1)
               ->  Hash  (cost=4.92..4.92 rows=22 width=10) (actual time=0.295..0.296 rows=13 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on climate cl  (cost=0.00..4.92 rows=22 width=10) (actual time=0.212..0.288 rows=13 loops=1)
                           Filter: ((co2emissions > $0) AND (forestedareapercent > $1))
                           Rows Removed by Filter: 182
 Planning Time: 0.840 ms
 Execution Time: 0.480 ms
(20 rows)
```

Perhaps indexing both columns at the same time would have sped it up, but the cost remains at 22.10. Again, this is most likely that the table is less than 200 rows and a sequential scan was fast enough.