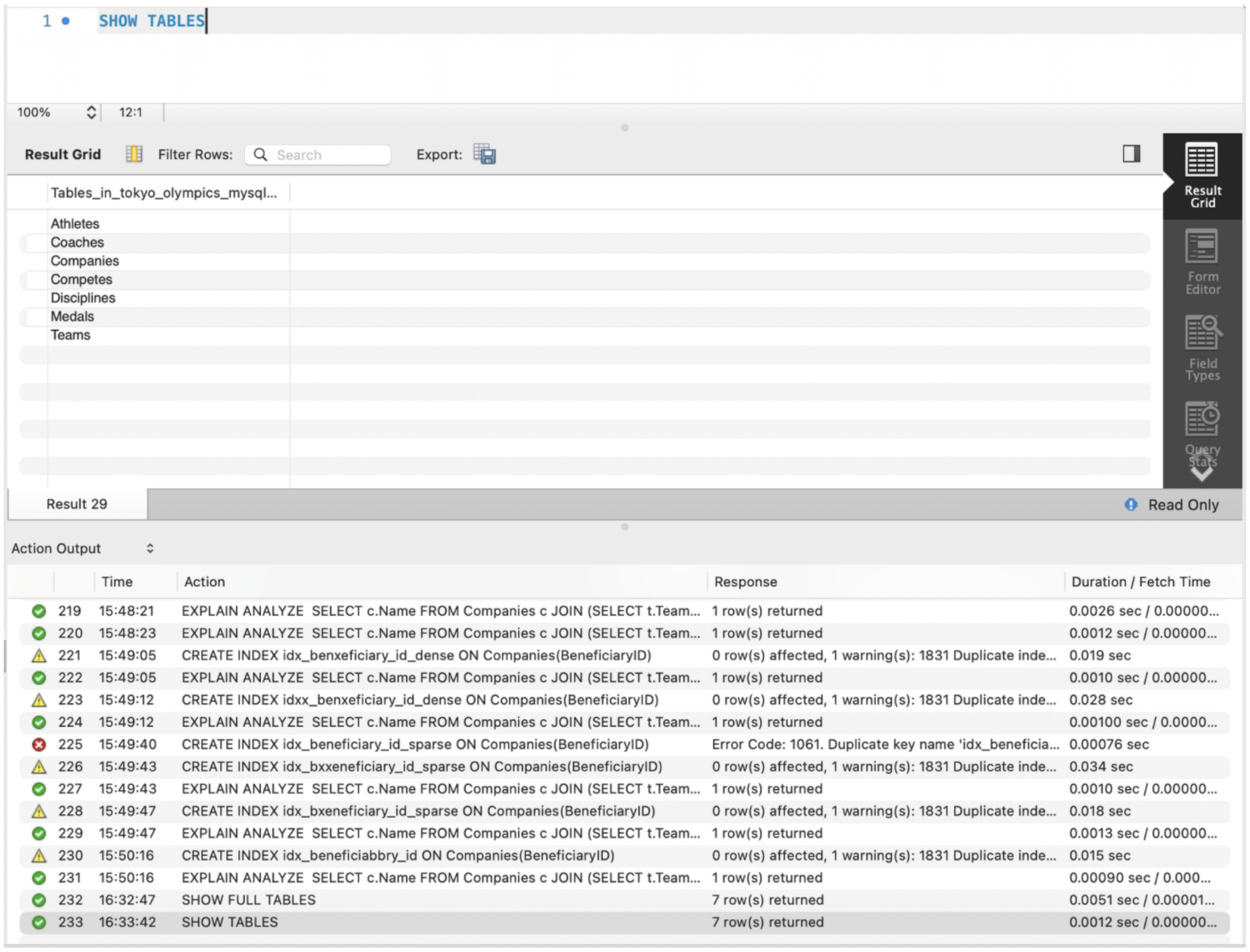
Database Design

Relational Schema DDL

```
CREATE TABLE Teams( TeamID INT PRIMARY KEY, CountryName VARCHAR(255) );

CREATE TABLE Medals( OverallRank INT, TeamID INT PRIMARY KEY, GoldTotal INT,
SilverTotal INT, BronzeTotal INT, Total INT, RankByTotal INT, FOREIGN KEY (teamID)
REFERENCES Teams(teamID), ON DELETE CASCADE );

CREATE TABLE Disciplines( DisciplineID INT PRIMARY KEY, Name VARCHAR(255),
FemaleCount INT, MaleCount INT, TotalCount INT );

CREATE TABLE Companies( CompanyID INT PRIMARY KEY, ForbesRank INT, Name
VARCHAR(255), BeneficiaryID INT, FOREIGN KEY (BeneficiaryID) REFERENCES
Teams(teamID) );

CREATE TABLE Coaches( CoachID INT PRIMARY KEY, Name VARCHAR(255), TeamID INT,
DisciplineID INT, FOREIGN KEY (TeamID) REFERENCES Teams(teamID), FOREIGN KEY
(DisciplineID) REFERENCES Disciplines(DisciplineID) );

CREATE TABLE Athletes( AthleteID INT PRIMARY KEY, Name VARCHAR(255), TeamID INT,
DisciplineID INT, CoachID Int, FOREIGN KEY (TeamID) REFERENCES Teams(teamID),
FOREIGN KEY (DisciplineID) REFERENCES Disciplines(DisciplineID), FOREIGN KEY
(CoachID) REFERENCES Coaches(CoachID) );

CREATE TABLE Competes( TeamID INT, DisciplineID INT, Classification VARCHAR(255),
PRIMARY KEY (TeamID, DisciplineID, Classification), FOREIGN KEY (TeamID) REFERENCES
Teams(teamID), FOREIGN KEY (DisciplineID) REFERENCES Disciplines(DisciplineID) );
```

Connection Using MySQL WorkBench:



```sql
SHOW TABLES
```

Tables_in_tokyo_olympics_mysql...
Athletes
Coaches
Companies
Competes
Disciplines
Medals
Teams

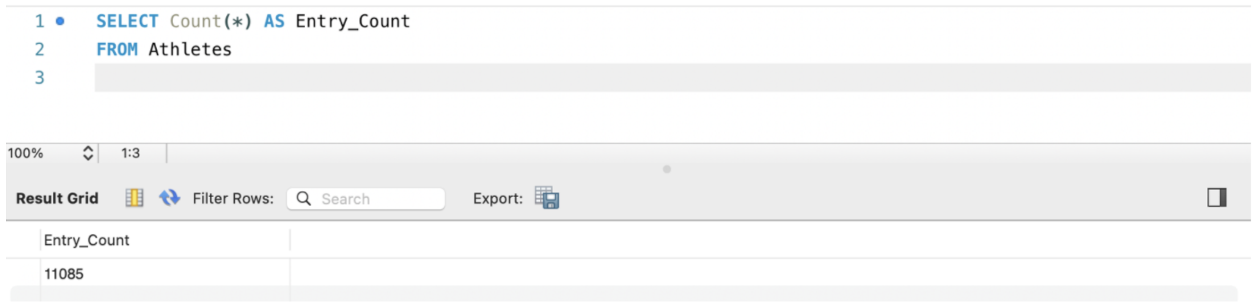
Result 29 Read Only

Action Output

```
     Time      Action                                                                 Response                                                      Duration / Fetch Time
219  15:48:21  EXPLAIN ANALYZE  SELECT c.Name FROM Companies c JOIN (SELECT t.Team...  1 row(s) returned                                             0.0026 sec / 0.00000...
220  15:48:23  EXPLAIN ANALYZE  SELECT c.Name FROM Companies c JOIN (SELECT t.Team...  1 row(s) returned                                             0.0012 sec / 0.00000...
221  15:49:05  CREATE INDEX idx_benxeficiary_id_dense ON Companies(BeneficiaryID)      0 row(s) affected, 1 warning(s): 1831 Duplicate inde...       0.019 sec
222  15:49:05  EXPLAIN ANALYZE  SELECT c.Name FROM Companies c JOIN (SELECT t.Team...  1 row(s) returned                                             0.0010 sec / 0.00000...
223  15:49:12  CREATE INDEX idxx_benxeficiary_id_dense ON Companies(BeneficiaryID)     0 row(s) affected, 1 warning(s): 1831 Duplicate inde...       0.028 sec
224  15:49:12  EXPLAIN ANALYZE  SELECT c.Name FROM Companies c JOIN (SELECT t.Team...  1 row(s) returned                                             0.00100 sec / 0.0000...
225  15:49:40  CREATE INDEX idx_beneficiary_id_sparse ON Companies(BeneficiaryID)      Error Code: 1061. Duplicate key name 'idx_beneficia...       0.00076 sec
226  15:49:43  CREATE INDEX idx_bxxeneficiary_id_sparse ON Companies(BeneficiaryID)    0 row(s) affected, 1 warning(s): 1831 Duplicate inde...       0.034 sec
227  15:49:43  EXPLAIN ANALYZE  SELECT c.Name FROM Companies c JOIN (SELECT t.Team...  1 row(s) returned                                             0.0010 sec / 0.00000...
228  15:49:47  CREATE INDEX idx_bxeneficiary_id_sparse ON Companies(BeneficiaryID)     0 row(s) affected, 1 warning(s): 1831 Duplicate inde...       0.018 sec
229  15:49:47  EXPLAIN ANALYZE  SELECT c.Name FROM Companies c JOIN (SELECT t.Team...  1 row(s) returned                                             0.0013 sec / 0.00000...
230  15:50:16  CREATE INDEX idx_beneficiabbry_id ON Companies(BeneficiaryID)           0 row(s) affected, 1 warning(s): 1831 Duplicate inde...       0.015 sec
231  15:50:16  EXPLAIN ANALYZE  SELECT c.Name FROM Companies c JOIN (SELECT t.Team...  1 row(s) returned                                             0.00090 sec / 0.000...
232  16:32:47  SHOW FULL TABLES                                                        7 row(s) returned                                             0.0051 sec / 0.00001...
233  16:33:42  SHOW TABLES                                                             7 row(s) returned                                             0.0012 sec / 0.00000...
```

Tables with 1000+ Entries

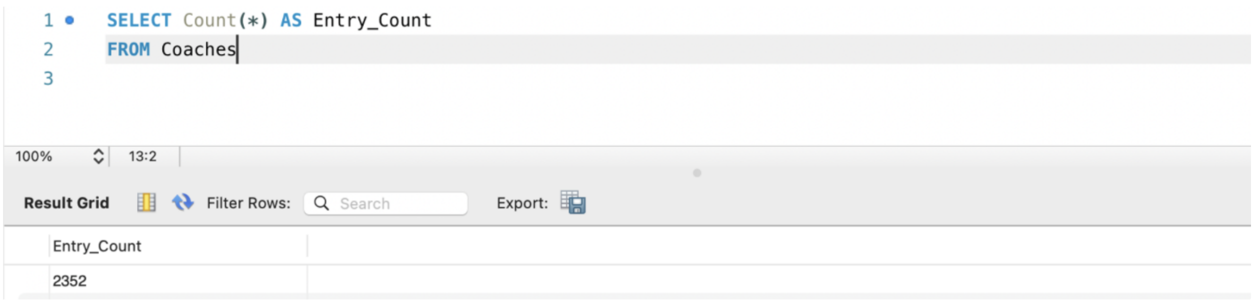
Athletes



```sql
SELECT Count(*) AS Entry_Count
FROM Athletes
```

Entry_Count
11085

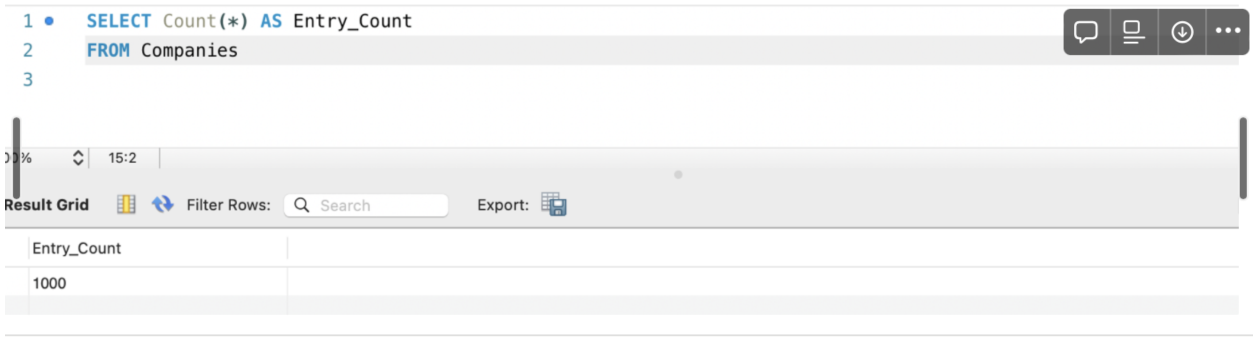
Coaches



```
SELECT Count(*) AS Entry_Count
FROM Coaches
```

```
Entry_Count
2352
```

Companies



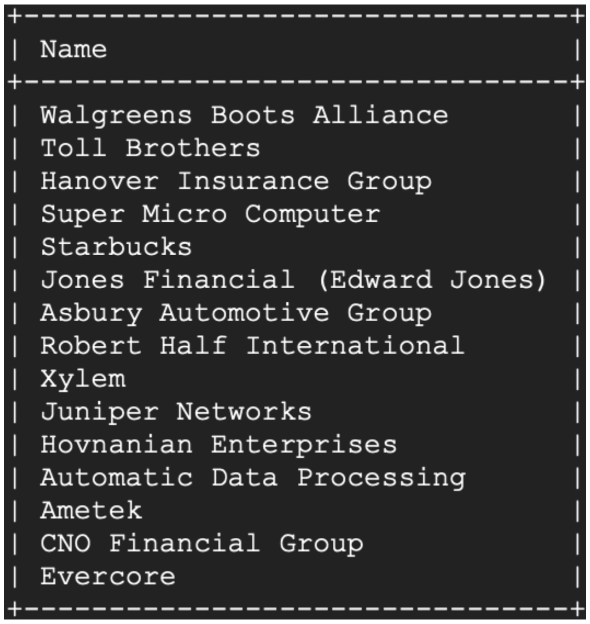
```
SELECT Count(*) AS Entry_Count
FROM Companies
```

```
Entry_Count
1000
```

Advanced Query #1

Query to list all companies that sponsor the best teams (top ten)

```
SELECT c.Name
FROM Companies c
JOIN (
   SELECT t.TeamID
   FROM Teams t
   JOIN Medals m ON t.TeamID = m.TeamID
   ORDER BY m.Total DESC
   LIMIT 10
) TopTen
ON c.BeneficiaryID = TopTen.TeamID
LIMIT 15;
```



```
+-------------------------------+
| Name                          |
+-------------------------------+
| Walgreens Boots Alliance      |
| Toll Brothers                 |
| Hanover Insurance Group       |
| Super Micro Computer          |
| Starbucks                     |
| Jones Financial (Edward Jones) |
| Asbury Automotive Group       |
| Robert Half International     |
| Xylem                         |
| Juniper Networks              |
| Hovnanian Enterprises         |
| Automatic Data Processing     |
| Ametek                        |
| CNO Financial Group           |
| Evercore                      |
+-------------------------------+
```

Index Constructions (Query 1)

Primary Index

Raw Output:

```
-> Limit: 15 row(s)  (cost=62.80 rows=15) (actual time=0.154..0.197 rows=15 loops=1)

    -> Nested loop inner join  (cost=62.80 rows=49) (actual time=0.154..0.195 rows=15 loops=1)

        -> Table scan on TopTen  (cost=43.36..45.72 rows=10) (actual time=0.119..0.120 rows=4
loops=1)

            -> Materialize  (cost=43.10..43.10 rows=10) (actual time=0.119..0.119 rows=10 loops=1)

                -> Limit: 10 row(s)  (cost=42.10 rows=10) (actual time=0.092..0.105 rows=10 loops=1)

                    -> Nested loop inner join  (cost=42.10 rows=93) (actual time=0.091..0.104 rows=10
loops=1)

                        -> Sort: m.Total DESC  (cost=9.55 rows=93) (actual time=0.079..0.079 rows=10
loops=1)

                            -> Table scan on m  (cost=9.55 rows=93) (actual time=0.039..0.049 rows=93
loops=1)
```

```
                        -> Single-row covering index lookup on t using PRIMARY (TeamID=m.TeamID) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=10)
        -> Index lookup on c using idx_beneficiary_id_dense (BeneficiaryID=TopTen.TeamID) (cost=1.27 rows=5) (actual time=0.017..0.018 rows=4 loops=4)
```

Design 1:

```
CREATE INDEX idx_companies_benefid ON Companies(BeneficiaryID);
CREATE INDEX idx_medals_total ON Medals(Total);
```

```
-> Limit: 15 row(s)  (cost=45.10 rows=15) (actual time=0.128..0.175 rows=15 loops=1)
    -> Nested loop inner join  (cost=45.10 rows=49) (actual time=0.127..0.173 rows=15 loops=1)
        -> Table scan on TopTen  (cost=25.65..28.01 rows=10) (actual time=0.097..0.097 rows=4 loops=1)
            -> Materialize  (cost=25.38..25.38 rows=10) (actual time=0.096..0.096 rows=10 loops=1)
                -> Limit: 10 row(s)  (cost=24.38 rows=10) (actual time=0.067..0.084 rows=10 loops=1)
                    -> Nested loop inner join  (cost=24.38 rows=10) (actual time=0.067..0.083 rows=10 loops=1)
                        -> Covering index scan on m using idx_medals_total (reverse)  (cost=0.13 rows=10) (actual time=0.026..0.027 rows=10 loops=1)
                        -> Single-row covering index lookup on t using PRIMARY (TeamID=m.TeamID) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=10)
        -> Index lookup on c using idx_beneficiary_id_dense (BeneficiaryID=TopTen.TeamID) (cost=1.27 rows=5) (actual time=0.017..0.018 rows=4 loops=4)
```

Design 2:

```
CREATE INDEX idx_medals_teamid_total ON Medals(teamID, Total)
CREATE INDEX idx_companies_benefid ON Companies(beneficiaryID);
```

```
-> Limit: 15 row(s)  (cost=45.10 rows=15) (actual time=0.109..0.152 rows=15 loops=1)
    -> Nested loop inner join  (cost=45.10 rows=49) (actual time=0.108..0.150 rows=15 loops=1)
        -> Table scan on TopTen  (cost=25.65..28.01 rows=10) (actual time=0.072..0.072 rows=4 loops=1)
            -> Materialize  (cost=25.38..25.38 rows=10) (actual time=0.070..0.070 rows=10 loops=1)
                -> Limit: 10 row(s)  (cost=24.38 rows=10) (actual time=0.042..0.058 rows=10 loops=1)
                    -> Nested loop inner join  (cost=24.38 rows=10) (actual time=0.042..0.057 rows=10 loops=1)
                        -> Covering index scan on m using idx_medals_total (reverse)  (cost=0.13 rows=10) (actual time=0.028..0.029 rows=10 loops=1)
                        -> Single-row covering index lookup on t using PRIMARY (TeamID=m.TeamID) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=10)
        -> Index lookup on c using idx_beneficiary_id_dense (BeneficiaryID=TopTen.TeamID) (cost=1.27 rows=5) (actual time=0.018..0.019 rows=4 loops=4)
```

Design 3:

```
-- Create indexes

CREATE INDEX idx_companies_name ON Companies(Name);

CREATE INDEX idx_medals_total ON Medals(Total);

CREATE INDEX idx_companies_benefid ON Companies(BeneficiaryID);

-> Limit: 15 row(s)  (cost=45.10 rows=15) (actual time=0.139..0.187 rows=15 loops=1)
    -> Nested loop inner join  (cost=45.10 rows=49) (actual time=0.138..0.185 rows=15 loops=1)
        -> Table scan on TopTen  (cost=25.65..28.01 rows=10) (actual time=0.113..0.113 rows=4 loops=1)
            -> Materialize  (cost=25.38..25.38 rows=10) (actual time=0.112..0.112 rows=10 loops=1)
            -> Limit: 10 row(s)  (cost=24.38 rows=10) (actual time=0.086..0.101 rows=10 loops=1)
                -> Nested loop inner join  (cost=24.38 rows=10) (actual time=0.086..0.100 rows=10 loops=1)
                    -> Covering index scan on m using idx_medals_total (reverse)  (cost=0.13 rows=10) (actual time=0.072..0.073 rows=10 loops=1)
                    -> Single-row covering index lookup on t using PRIMARY (TeamID=m.TeamID) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=10)
        -> Index lookup on c using idx_beneficiary_id_dense (BeneficiaryID=TopTen.TeamID) (cost=1.27 rows=5) (actual time=0.016..0.017 rows=4 loops=4)
```

Conclusion:

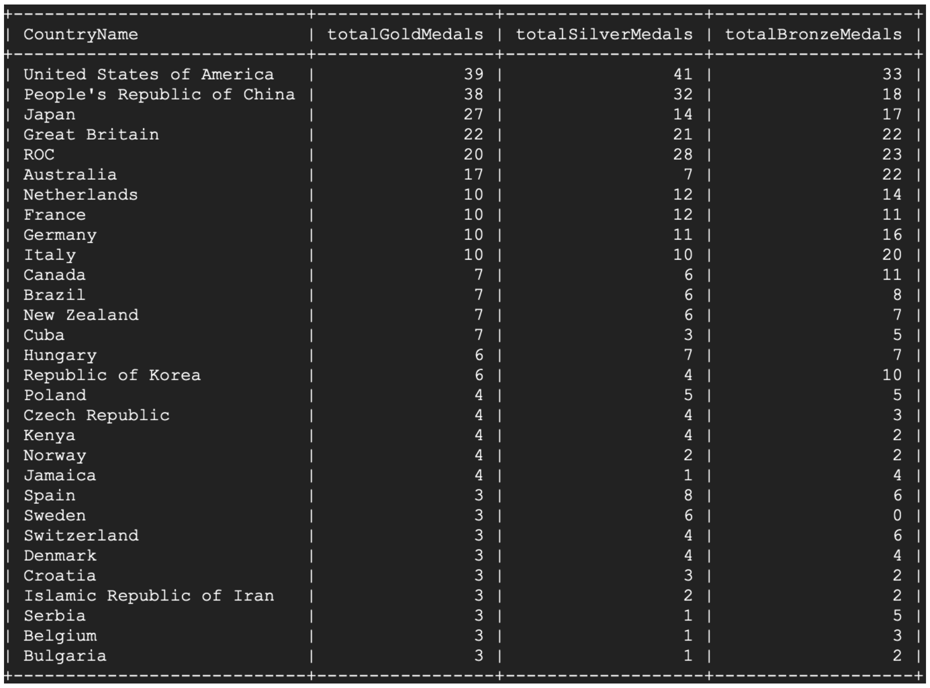
Looking at the execution times of the different indexing designs, we can see there are small differences in the execution times. From this and some of the other provided results in the EXPLAIN ANALYZE output, we can conclude that design 2 is the best because design 1 has slightly longer times in part of the subquery, as well as design 3 which also has slightly longer times in the subquery, but there is not much difference in cost.

Advanced Query #2

Query to output countries with the highest number of medals in descending order, minimum of 3 medals

```
SELECT DISTINCT Teams.CountryName,
   Medals.GoldTotal AS totalGoldMedals,
   Medals.SilverTotal AS totalSilverMedals,
   Medals.BronzeTotal AS totalBronzeMedals
FROM Teams
```

```
JOIN Athletes ON Athletes.teamID = Teams.teamID

JOIN Medals ON Athletes.teamID = Medals.teamID

JOIN Companies ON Companies.BeneficiaryID = Athletes.teamID

WHERE Medals.GoldTotal + Medals.SilverTotal + Medals.BronzeTotal >= 3

ORDER BY totalGoldMedals DESC, totalSilverMedals DESC, totalBronzeMedals DESC

LIMIT 30;
```



```
+----------------------------+-----------------+-------------------+-------------------+
| CountryName                | totalGoldMedals | totalSilverMedals | totalBronzeMedals |
+----------------------------+-----------------+-------------------+-------------------+
| United States of America   |              39 |                41 |                33 |
| People's Republic of China |              38 |                32 |                18 |
| Japan                      |              27 |                14 |                17 |
| Great Britain              |              22 |                21 |                22 |
| ROC                        |              20 |                28 |                23 |
| Australia                  |              17 |                 7 |                22 |
| Netherlands                |              10 |                12 |                14 |
| France                     |              10 |                12 |                11 |
| Germany                    |              10 |                11 |                16 |
| Italy                      |              10 |                10 |                20 |
| Canada                     |               7 |                 6 |                11 |
| Brazil                     |               7 |                 6 |                 8 |
| New Zealand                |               7 |                 6 |                 7 |
| Cuba                       |               7 |                 3 |                 5 |
| Hungary                    |               6 |                 7 |                 7 |
| Republic of Korea          |               6 |                 4 |                10 |
| Poland                     |               4 |                 5 |                 5 |
| Czech Republic             |               4 |                 4 |                 3 |
| Kenya                      |               4 |                 4 |                 2 |
| Norway                     |               4 |                 2 |                 2 |
| Jamaica                    |               4 |                 1 |                 4 |
| Spain                      |               3 |                 8 |                 6 |
| Sweden                     |               3 |                 6 |                 0 |
| Switzerland                |               3 |                 4 |                 6 |
| Denmark                    |               3 |                 4 |                 4 |
| Croatia                    |               3 |                 3 |                 2 |
| Islamic Republic of Iran   |               3 |                 2 |                 2 |
| Serbia                     |               3 |                 1 |                 5 |
| Belgium                    |               3 |                 1 |                 3 |
| Bulgaria                   |               3 |                 1 |                 2 |
+----------------------------+-----------------+-------------------+-------------------+
```

Index Constructions (Query 1)

Primary Index

Raw Output:

```
-> Limit: 30 row(s)  (actual time=3.891..3.894 rows=30 loops=1)

    -> Sort: Medals.GoldTotal DESC, Medals.SilverTotal DESC, Medals.BronzeTotal DESC, limit input to 30 row(s) per chunk  (actual time=3.890..3.892 rows=30 loops=1)

        -> Table scan on <temporary>  (cost=321.68..329.84 rows=454) (actual time=3.833..3.846 rows=65 loops=1)
```

```
            -> Temporary table with deduplication  (cost=321.66..321.66 rows=454) (actual
time=3.830..3.830 rows=65 loops=1)

                -> Nested loop inner join  (cost=276.25 rows=454) (actual time=0.127..3.616
rows=299 loops=1)

                    -> Nested loop inner join  (cost=110.86 rows=454) (actual time=0.088..0.477
rows=299 loops=1)

                        -> Nested loop inner join  (cost=42.10 rows=93) (actual time=0.073..0.208
rows=65 loops=1)

                            -> Filter: (((Medals.GoldTotal + Medals.SilverTotal) + Medals.BronzeTotal) >=
3)  (cost=9.55 rows=93) (actual time=0.057..0.090 rows=65 loops=1)

                                -> Table scan on Medals  (cost=9.55 rows=93) (actual time=0.053..0.070
rows=93 loops=1)

                            -> Single-row index lookup on Teams using PRIMARY
(TeamID=Medals.TeamID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=65)

                        -> Covering index lookup on Companies using idx_beneficiary_id_dense
(BeneficiaryID=Medals.TeamID)  (cost=0.26 rows=5) (actual time=0.003..0.004 rows=5
loops=65)

                    -> Limit: 1 row(s)  (cost=0.28 rows=1) (actual time=0.010..0.010 rows=1 loops=299)

                        -> Covering index lookup on Athletes using TeamID (TeamID=Medals.TeamID)
(cost=0.28 rows=53) (actual time=0.010..0.010 rows=1 loops=299)
```

Design 1:

```
CREATE INDEX idx_teams_countryname ON Teams(CountryName);

CREATE INDEX idx_athletes_teamid ON Athletes(teamID);

CREATE INDEX idx_medals_teamid_totals ON Medals(teamID, GoldTotal, SilverTotal,
BronzeTotal);

CREATE INDEX idx_companies_beneficiaryid ON Companies(BeneficiaryID);
```

```
-> Limit: 30 row(s)  (actual time=3.614..3.617 rows=30 loops=1)

    -> Sort: Medals.GoldTotal DESC, Medals.SilverTotal DESC, Medals.BronzeTotal DESC, limit
input to 30 row(s) per chunk  (actual time=3.613..3.615 rows=30 loops=1)
```

```
    -> Table scan on <temporary>  (cost=321.68..329.84 rows=454) (actual time=3.564..3.576
rows=65 loops=1)

        -> Temporary table with deduplication  (cost=321.66..321.66 rows=454) (actual
time=3.562..3.562 rows=65 loops=1)

            -> Nested loop inner join  (cost=276.25 rows=454) (actual time=0.139..3.348
rows=299 loops=1)

                -> Nested loop inner join  (cost=110.86 rows=454) (actual time=0.103..0.463
rows=299 loops=1)

                    -> Nested loop inner join  (cost=42.10 rows=93) (actual time=0.058..0.185
rows=65 loops=1)

                        -> Filter: (((Medals.GoldTotal + Medals.SilverTotal) + Medals.BronzeTotal) >=
3)  (cost=9.55 rows=93) (actual time=0.044..0.077 rows=65 loops=1)

                            -> Covering index scan on Medals using idx_medals_teamid_totals
(cost=9.55 rows=93) (actual time=0.041..0.056 rows=93 loops=1)

                        -> Single-row index lookup on Teams using PRIMARY
(TeamID=Medals.TeamID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=65)

                    -> Covering index lookup on Companies using idx_beneficiary_id_dense
(BeneficiaryID=Medals.TeamID)  (cost=0.26 rows=5) (actual time=0.003..0.004 rows=5
loops=65)

                -> Limit: 1 row(s)  (cost=0.28 rows=1) (actual time=0.009..0.009 rows=1 loops=299)

                    -> Covering index lookup on Athletes using TeamID (TeamID=Medals.TeamID)
(cost=0.28 rows=53) (actual time=0.009..0.009 rows=1 loops=299)
```

Design 2:

```
CREATE INDEX idx_teams_teamid ON Teams(teamID); - UNUSED PRIMARY KEY

CREATE INDEX idx_athletes_country_teamid ON Athletes(Country, teamID);

CREATE INDEX idx_medals_teamid_totals ON Medals(teamID, GoldTotal, SilverTotal,
BronzeTotal);

CREATE INDEX idx_comp2anies_beneficiaryid ON Companies(BeneficiaryID);
```

```
-> Limit: 30 row(s)  (actual time=3.292..3.297 rows=30 loops=1)
```

```
   -> Sort: Medals.GoldTotal DESC, Medals.SilverTotal DESC, Medals.BronzeTotal DESC, limit input to 30 row(s) per chunk  (actual time=3.292..3.295 rows=30 loops=1)

       -> Table scan on <temporary>  (cost=321.68..329.84 rows=454) (actual time=3.247..3.257 rows=65 loops=1)

           -> Temporary table with deduplication  (cost=321.66..321.66 rows=454) (actual time=3.244..3.244 rows=65 loops=1)

               -> Nested loop inner join  (cost=276.25 rows=454) (actual time=0.090..3.060 rows=299 loops=1)

                   -> Nested loop inner join  (cost=110.86 rows=454) (actual time=0.065..0.422 rows=299 loops=1)

                       -> Nested loop inner join  (cost=42.10 rows=93) (actual time=0.055..0.168 rows=65 loops=1)

                           -> Filter: (((Medals.GoldTotal + Medals.SilverTotal) + Medals.BronzeTotal) >= 3)  (cost=9.55 rows=93) (actual time=0.043..0.071 rows=65 loops=1)

                               -> Covering index scan on Medals using idx_medals_teamid_totals (cost=9.55 rows=93) (actual time=0.040..0.053 rows=93 loops=1)

                           -> Single-row index lookup on Teams using PRIMARY (TeamID=Medals.TeamID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=65)

                       -> Covering index lookup on Companies using idx_beneficiary_id_dense (BeneficiaryID=Medals.TeamID)  (cost=0.26 rows=5) (actual time=0.002..0.004 rows=5 loops=65)

                   -> Limit: 1 row(s)  (cost=0.28 rows=1) (actual time=0.009..0.009 rows=1 loops=299)

                       -> Covering index lookup on Athletes using TeamID (TeamID=Medals.TeamID) (cost=0.28 rows=53) (actual time=0.008..0.008 rows=1 loops=299)
```

Design 3:

```
CREATE INDEX idx_tea4233ms_countryname ON Teams(CountryName);

CREATE INDEX idx_athl43423etes_country ON Athletes(Country);

CREATE INDEX idx_meda324ls_teamid_totals ON Medals(teamID, GoldTotal, SilverTotal, BronzeTotal);
CREATE INDEX idx_compa2342nies_beneficiaryid ON Companies(BeneficiaryID);

-> Limit: 30 row(s)  (actual time=3.236..3.240 rows=30 loops=1)

    -> Sort: Medals.GoldTotal DESC, Medals.SilverTotal DESC, Medals.BronzeTotal DESC, limit input to 30 row(s) per chunk  (actual time=3.236..3.237 rows=30 loops=1)

        -> Table scan on <temporary>  (cost=321.68..329.84 rows=454) (actual time=3.194..3.204 rows=65 loops=1)

            -> Temporary table with deduplication  (cost=321.66..321.66 rows=454) (actual time=3.192..3.192 rows=65 loops=1)

                -> Nested loop inner join  (cost=276.25 rows=454) (actual time=0.084..3.010 rows=299 loops=1)

                    -> Nested loop inner join  (cost=110.86 rows=454) (actual time=0.062..0.413 rows=299 loops=1)

                        -> Nested loop inner join  (cost=42.10 rows=93) (actual time=0.052..0.170 rows=65 loops=1)

                            -> Filter: (((Medals.GoldTotal + Medals.SilverTotal) + Medals.BronzeTotal) >= 3)  (cost=9.55 rows=93) (actual time=0.040..0.067 rows=65 loops=1)

                                -> Covering index scan on Medals using idx_medals_teamid_totals (cost=9.55 rows=93) (actual time=0.038..0.050 rows=93 loops=1)

                            -> Single-row index lookup on Teams using PRIMARY (TeamID=Medals.TeamID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=65)

                        -> Covering index lookup on Companies using idx_beneficiary_id_dense (BeneficiaryID=Medals.TeamID)  (cost=0.26 rows=5) (actual time=0.002..0.003 rows=5 loops=65)

                    -> Limit: 1 row(s)  (cost=0.28 rows=1) (actual time=0.008..0.009 rows=1 loops=299)

                        -> Covering index lookup on Athletes using TeamID (TeamID=Medals.TeamID) (cost=0.28 rows=53) (actual time=0.008..0.008 rows=1 loops=299)
```

Conclusion:

After looking through the different execution times based on the different indexing designs, we can see that there is only a small difference in the execution times. Based on this, we can see that design 3 has the lowest execution time so it makes it the most optimal design for this query. There is not much difference in the cost.