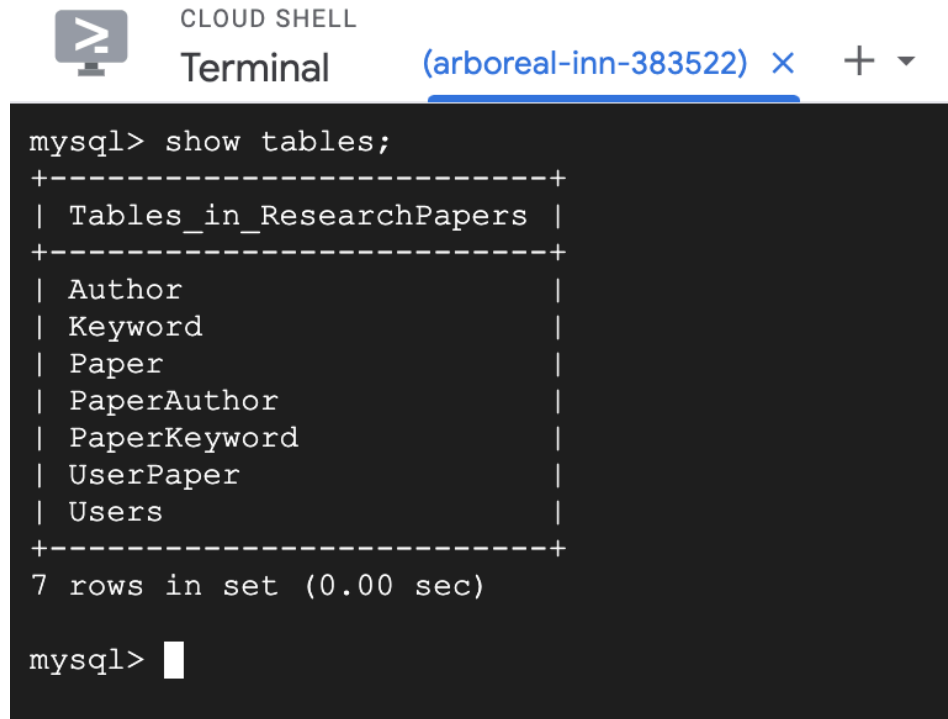


Database Design

Connection

We implement our database tables on GCP. The figure below shows the cloud shell and we are able to establish a connection and import the database:



The screenshot shows a Google Cloud Shell terminal window. At the top, there is a terminal icon, the text 'CLOUD SHELL', and a tab labeled 'Terminal' with the session ID '(arboreal-inn-383522)'. The terminal content shows a MySQL prompt 'mysql>' followed by the command 'show tables;'. The output is a table listing seven tables: 'Tables_in_ResearchPapers', 'Author', 'Keyword', 'Paper', 'PaperAuthor', 'PaperKeyword', 'UserPaper', and 'Users'. Below the table, it says '7 rows in set (0.00 sec)'. The prompt 'mysql>' is shown again with a cursor.

```
mysql> show tables;
+-----+
| Tables_in_ResearchPapers |
+-----+
| Author                    |
| Keyword                   |
| Paper                     |
| PaperAuthor               |
| PaperKeyword              |
| UserPaper                 |
| Users                     |
+-----+
7 rows in set (0.00 sec)

mysql> 
```

A screenshot of the connection

DDL commands

The DDL commands for all tables are as follows:

```
CREATE TABLE Paper (
  id VARCHAR(64) PRIMARY KEY,
  title VARCHAR(4096),
  abstract VARCHAR(8000),
  published_date VARCHAR(10)
);
```

```
CREATE TABLE Keyword (
  id INTEGER PRIMARY KEY,
  keyword VARCHAR(255) UNIQUE
);
```

```
CREATE TABLE PaperKeyword (  
    paperId VARCHAR(64),  
    keywordId INTEGER,  
    score Decimal,  
    FOREIGN KEY (paperId) REFERENCES Paper(id) ON DELETE CASCADE,  
    FOREIGN KEY (keywordId) REFERENCES Keyword(id) ON DELETE CASCADE,  
    PRIMARY KEY (paperId, keywordId)  
);
```

```
CREATE TABLE Users (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    username VARCHAR(255),  
    email VARCHAR(255),  
    password VARCHAR(255)  
);
```

```
CREATE TABLE UserPaper (  
    userId INTEGER,  
    paperId VARCHAR(64),  
    notes VARCHAR(8000),  
    rating INTEGER,  
    comment VARCHAR(8000),  
    FOREIGN KEY (userId) REFERENCES Users(id) ON DELETE CASCADE,  
    FOREIGN KEY (paperId) REFERENCES Paper(id) ON DELETE CASCADE,  
    PRIMARY KEY (userId, paperId)  
);
```

```
CREATE TABLE Author (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    name VARCHAR(255) UNIQUE  
);
```

```
CREATE TABLE PaperAuthor (  
    paperId VARCHAR(64),  
    authorId INTEGER,  
    FOREIGN KEY (paperId) REFERENCES Paper(id) ON DELETE CASCADE,  
    FOREIGN KEY (authorId) REFERENCES Author(id) ON DELETE CASCADE,  
    PRIMARY KEY (paperId, authorId)  
);
```

Count query

The count query for all tables are as follows:

```
mysql> SELECT COUNT(*) FROM Author;
+-----+
| COUNT(*) |
+-----+
|      1143 |
+-----+
1 row in set (0.01 sec)

mysql> █
```

```
mysql> SELECT COUNT(*) FROM Keyword;
+-----+
| COUNT(*) |
+-----+
|     11384 |
+-----+
1 row in set (0.01 sec)

mysql> █
```

```
mysql> SELECT COUNT(*) FROM Paper;
+-----+
| COUNT(*) |
+-----+
|      1500 |
+-----+
1 row in set (0.01 sec)

mysql> █
```

```
mysql> SELECT COUNT(*) FROM PaperKeyword;
+-----+
| COUNT(*) |
+-----+
|     15000 |
+-----+
1 row in set (0.01 sec)

mysql> █
```

```
mysql> SELECT COUNT(*) FROM PaperAuthor;
+-----+
| COUNT(*) |
+-----+
|      1497 |
+-----+
1 row in set (0.05 sec)

mysql> █
```

```
mysql> SELECT COUNT(*) FROM UserPaper;
+-----+
| COUNT(*) |
+-----+
|      1999 |
+-----+
1 row in set (0.01 sec)

mysql> █
```

```
mysql> SELECT COUNT(*) FROM Users;
+-----+
| COUNT(*) |
+-----+
|      1000 |
+-----+
1 row in set (0.00 sec)

mysql> █
```

SQL queries:

1. Get all papers with their authors and keywords:

```
SELECT p.title, GROUP_CONCAT(a.name), GROUP_CONCAT(k.keyword)
FROM Paper p
JOIN PaperAuthor pa ON p.id = pa.paperId
JOIN Author a ON pa.authorId = a.id
JOIN PaperKeyword pk ON p.id = pk.paperId
JOIN Keyword k ON pk.keywordId = k.id
GROUP BY p.id
LIMIT 15;
```

[illegible]

2. Find papers with the highest average rating:

```
SELECT p.title, AVG(up.rating)
FROM Paper p
JOIN UserPaper up ON p.id = up.paperId
GROUP BY p.id
ORDER BY AVG(up.rating) DESC
LIMIT 15;
```

```
mysql> SELECT p.title, AVG(up.rating)
-> FROM Paper p
-> JOIN UserPaper up ON p.id = up.paperId
-> GROUP BY p.id
-> ORDER BY AVG(up.rating) DESC
-> LIMIT 15;
```

title	AVG(up.rating)
Mechanisms for Automated Negotiation in State Oriented Domains	5.0000
Cost Efficient Design of Reversible Adder Circuits for Low Power, Applications	5.0000
HAPPY: Hybrid Address-based Page Policy in DRAMs	5.0000
Recognizing When Heuristics Can Approximate Minimum Vertex Covers Is, Complete for Parallel Access to NP	5.0000
Multi Core SSL/TLS Security Processor Architecture Prototype Design with, automated Preferential Algorithm in FPGA	5.0000
Building and Refining Abstract Planning Cases by Change of, Representation Language	5.0000
An Extension to DNA Based Fredkin Gate Circuits: Design of Reversible, Sequential Circuits using Fredkin Gates	5.0000
System approach to synthesis, modeling and control of complex dynamical, systems	5.0000
Basic properties for sand automata	5.0000
Difficulties in the Implementation of Quantum Computers	5.0000
Unambiguous Computation: Boolean Hierarchies and Sparse Turing-Complete, Sets	5.0000
Is Neural Network a Reliable Forecaster on Earth? A MARS Query!	5.0000
Polynomial Threshold Functions: Structure, Approximation and, Pseudorandomness	5.0000
Reversible Programmable Logic Array (RPLA) using Fredkin & Feynman Gates, for Industrial Electronics and Applications	5.0000
Interpolation in Valliant's theory	5.0000

```
15 rows in set (0.01 sec)

mysql>
```

3. Find keywords with the highest total score across all papers:

```
SELECT k.keyword, SUM(pk.score)
FROM Keyword k
JOIN PaperKeyword pk ON k.id = pk.keywordId
GROUP BY k.id
ORDER BY SUM(pk.score) DESC
LIMIT 15;
```

```
mysql> SELECT k.keyword, SUM(pk.score)
-> FROM Keyword k
-> JOIN PaperKeyword pk ON k.id = pk.keywordId
-> GROUP BY k.id
-> ORDER BY SUM(pk.score) DESC
-> LIMIT 15;
```

keyword	SUM(pk.score)
equally	511
competence	510
natural	420
compromising	419
simulationtime	394
sized	368
interfacial	364
monotone	341
dvt	335
launch	334
relativizations	334
criticism	333
undertook	324
bichromatic	324
escaped	307

15 rows in set (0.04 sec)

4. Get papers rated at least 4 by users and papers with a total score over 100 on keywords:

```
SELECT p.title, up.rating
FROM Paper p
JOIN UserPaper up ON p.id = up.paperId
WHERE up.rating >= 4
UNION
SELECT p.title, SUM(pk.score)
FROM Paper p
JOIN PaperKeyword pk ON p.id = pk.paperId
GROUP BY p.id
HAVING SUM(pk.score) > 100
LIMIT 15;
```

```
mysql> SELECT p.title, up.rating
-> FROM Paper p
-> JOIN UserPaper up ON p.id = up.paperId
-> WHERE up.rating >= 4
->
-> UNION
->
-> SELECT p.title, SUM(pk.score)
-> FROM Paper p
-> JOIN PaperKeyword pk ON p.id = pk.paperId
-> GROUP BY p.id
-> HAVING SUM(pk.score) > 100
-> LIMIT 15;
+-----+-----+
| title | rating |
+-----+-----+
| A Novel Methodology for Thermal Analysis & 3-Dimensional Memory, | 4 |
| Integration | 4 |
| HAPPY: Hybrid Address-based Page Policy in DRAMs | 5 |
| Building and Refining Abstract Planning Cases by Change of, | 5 |
| Representation Language | 5 |
| Modeling structural change in spatial system dynamics: A Daisyworld, | 4 |
| example | 4 |
| Hygro-thermo-mechanical analysis of spalling in concrete walls at high, | 4 |
| temperatures as a moving boundary problem | 4 |
| Effective Strong Dimension, Algorithmic Information, and Computational, | 4 |
| Complexity | 4 |
| Polynomial Threshold Functions: Structure, Approximation and, | 5 |
| Pseudorandomness | 5 |
| On Some Recent Insights in Integral Biomathics | 4 |
| DATALOG with constraints - an answer-set programming system | 5 |
| The evolution of the parametric models of drawings (modules) in the, | 4 |
| enterprises reconstruction CAD system | 4 |
| Computational Power of P Systems with Small Size Insertion and Deletion, | 4 |
| Rules | 4 |
| Analysis of approximate nearest neighbor searching with clustered point, | 5 |
| sets | 5 |
| Meeting the Embedded Design Needs of Automotive Applications | 4 |
| Languages recognized by nondeterministic quantum finite automata | 4 |
| From Statistical Knowledge Bases to Degrees of Belief | 4 |
+-----+-----+
15 rows in set (0.00 sec)
```

Indexing analysis

Query 1

No index (Cost=7754.90)

```
| -> Group aggregate: group_concat(Author.'name' separator ','), group_concat(Keyword.keyword separator ',') (actual time=64.821..73.867 rows=1497 loops=1)
-> Sort: p.id (actual time=64.791..66.728 rows=14970 loops=1)
-> Stream results (cost=7754.90 rows=13685) (actual time=0.154..49.852 rows=14970 loops=1)
-> Nested loop inner join (cost=7754.90 rows=13685) (actual time=0.151..40.087 rows=14970 loops=1)
-> Nested loop inner join (cost=2965.00 rows=13685) (actual time=0.142..20.631 rows=14970 loops=1)
-> Nested loop inner join (cost=1077.11 rows=1497) (actual time=0.107..8.017 rows=1497 loops=1)
-> Nested loop inner join (cost=553.16 rows=1497) (actual time=0.088..3.607 rows=1497 loops=1)
-> Covering index scan on a using name (cost=115.55 rows=1143) (actual time=0.068..0.537 rows=1143 loops=1)
-> Covering index lookup on pa using authorId (authorId=a.id) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1143)
-> Single-row index lookup on p using PRIMARY (id=pa.paperId) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1497)
-> Covering index lookup on pk using PRIMARY (paperId=pa.paperId) (cost=0.35 rows=9) (actual time=0.006..0.008 rows=10 loops=1497)
-> Single-row index lookup on k using PRIMARY (id=pk.keywordId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=14970)
```

CREATE INDEX author_name_idx ON Author(name); (Cost=7754.90, no change)

```
| -> Group aggregate: group_concat(Author.'name' separator ','), group_concat(Keyword.keyword separator ',') (actual time=60.494..69.522 rows=1497 loops=1)
-> Sort: p.id (actual time=60.469..62.406 rows=14970 loops=1)
-> Stream results (cost=7754.90 rows=13685) (actual time=0.094..46.035 rows=14970 loops=1)
-> Nested loop inner join (cost=7754.90 rows=13685) (actual time=0.092..36.965 rows=14970 loops=1)
-> Nested loop inner join (cost=2965.00 rows=13685) (actual time=0.086..18.598 rows=14970 loops=1)
-> Nested loop inner join (cost=1077.11 rows=1497) (actual time=0.064..7.423 rows=1497 loops=1)
-> Nested loop inner join (cost=553.16 rows=1497) (actual time=0.051..3.345 rows=1497 loops=1)
-> Covering index scan on a using name (cost=115.55 rows=1143) (actual time=0.038..0.453 rows=1143 loops=1)
-> Covering index lookup on pa using authorId (authorId=a.id) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1143)
-> Single-row index lookup on p using PRIMARY (id=pa.paperId) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1497)
-> Covering index lookup on pk using PRIMARY (paperId=pa.paperId) (cost=0.35 rows=9) (actual time=0.005..0.007 rows=10 loops=1497)
-> Single-row index lookup on k using PRIMARY (id=pk.keywordId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=14970)
```

CREATE INDEX keyword_idx ON Keyword(keyword); (Cost=7754.90, no change)

```
| -> Group aggregate: group_concat(Author.'name' separator ','), group_concat(Keyword.keyword separator ',') (actual time=68.927..78.562 rows=1497 loops=1)
-> Sort: p.id (actual time=68.897..70.967 rows=14970 loops=1)
-> Stream results (cost=7754.90 rows=13685) (actual time=0.102..50.767 rows=14970 loops=1)
-> Nested loop inner join (cost=7754.90 rows=13685) (actual time=0.100..41.596 rows=14970 loops=1)
-> Nested loop inner join (cost=2965.00 rows=13685) (actual time=0.093..21.143 rows=14970 loops=1)
-> Nested loop inner join (cost=1077.11 rows=1497) (actual time=0.070..8.818 rows=1497 loops=1)
-> Nested loop inner join (cost=553.16 rows=1497) (actual time=0.056..4.026 rows=1497 loops=1)
-> Covering index scan on a using name (cost=115.55 rows=1143) (actual time=0.043..0.481 rows=1143 loops=1)
-> Covering index lookup on pa using authorId (authorId=a.id) (cost=0.25 rows=1) (actual time=0.002..0.003 rows=1 loops=1143)
-> Single-row index lookup on p using PRIMARY (id=pa.paperId) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1497)
-> Covering index lookup on pk using PRIMARY (paperId=pa.paperId) (cost=0.35 rows=9) (actual time=0.006..0.007 rows=10 loops=1497)
-> Single-row index lookup on k using PRIMARY (id=pk.keywordId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=14970)
```

The two indices we created did not improve the cost of the query. One possible reason for this is that the tables involved in the query are not large enough to result in an obvious change. Another possible reason for this is because this query is joining on primary keys, so the additional indices may not result in further optimization.

Query 2

No index (Cost=901.80)

```
| -> Sort: AVG(up.rating) DESC (actual time=14.140..14.303 rows=1092 loops=1)
-> Table scan on <temporary> (actual time=12.837..13.160 rows=1092 loops=1)
-> Aggregate using temporary table (actual time=12.833..12.833 rows=1092 loops=1)
-> Nested loop inner join (cost=901.80 rows=1999) (actual time=0.047..8.316 rows=1999 loops=1)
-> Table scan on up (cost=202.15 rows=1999) (actual time=0.031..1.544 rows=1999 loops=1)
-> Single-row index lookup on p using PRIMARY (id=up.paperId) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1999)
```

CREATE INDEX paper_rating_idx ON UserPaper(rating); (Cost=901.80, no change)

```
| -> Sort: AVG(up.rating) DESC (actual time=9.159..9.256 rows=1092 loops=1)
-> Table scan on <temporary> (actual time=8.413..8.648 rows=1092 loops=1)
-> Aggregate using temporary table (actual time=8.408..8.408 rows=1092 loops=1)
-> Nested loop inner join (cost=901.80 rows=1999) (actual time=0.109..5.386 rows=1999 loops=1)
-> Covering index scan on up using paper_rating_idx (cost=202.15 rows=1999) (actual time=0.043..0.816 rows=1999 loops=1)
-> Single-row index lookup on p using PRIMARY (id=up.paperId) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1999)
```

CREATE INDEX paper_title_idx ON Paper(title(255)); (Cost=901.80, no change)

```
| -> Sort: AVG(up.rating) DESC (actual time=9.026..9.114 rows=1092 loops=1)
-> Table scan on <temporary> (actual time=8.111..8.435 rows=1092 loops=1)
-> Aggregate using temporary table (actual time=8.106..8.106 rows=1092 loops=1)
-> Nested loop inner join (cost=901.80 rows=1999) (actual time=0.040..5.193 rows=1999 loops=1)
-> Table scan on up (cost=202.15 rows=1999) (actual time=0.025..0.838 rows=1999 loops=1)
-> Single-row index lookup on p using PRIMARY (id=up.paperId) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1999)
```


The two indices we created did not improve the cost of the query. One possible reason for this is that the tables involved in the query are not large enough to result in an obvious change. Another possible reason for this is because this query is joining on primary keys, so the additional indices may not result in further optimization.

Query 3

No index (Cost=6824.00)

```
| -> Sort: SUM(pk.score) DESC (actual time=40.756..41.275 rows=5660 loops=1)
    -> Table scan on <temporary> (actual time=35.902..37.192 rows=5660 loops=1)
        -> Aggregate using temporary table (actual time=35.900..35.900 rows=5660 loops=1)
            -> Nested loop inner join (cost=6824.00 rows=15075) (actual time=0.049..22.727 rows=15000 loops=1)
                -> Table scan on pk (cost=1547.75 rows=15075) (actual time=0.038..4.829 rows=15000 loops=1)
                -> Single-row index lookup on k using PRIMARY (id=pk.keywordId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=15000)
```

CREATE INDEX pk_score_idx ON PaperKeyword(score); (Cost=6824.00, no change)

```
| -> Sort: SUM(pk.score) DESC (actual time=37.424..37.910 rows=5660 loops=1)
    -> Table scan on <temporary> (actual time=33.278..34.381 rows=5660 loops=1)
        -> Aggregate using temporary table (actual time=33.275..33.275 rows=5660 loops=1)
            -> Nested loop inner join (cost=6824.00 rows=15075) (actual time=0.079..21.083 rows=15000 loops=1)
                -> Covering index scan on pk using pk_score_idx (cost=1547.75 rows=15075) (actual time=0.063..4.111 rows=15000 loops=1)
                -> Single-row index lookup on k using PRIMARY (id=pk.keywordId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=15000)
```

CREATE INDEX keyword_idx ON Keyword(keyword); (Cost=6824.00, no change)

```
| -> Sort: SUM(pk.score) DESC (actual time=42.165..42.677 rows=5660 loops=1)
    -> Table scan on <temporary> (actual time=37.563..38.852 rows=5660 loops=1)
        -> Aggregate using temporary table (actual time=37.560..37.560 rows=5660 loops=1)
            -> Nested loop inner join (cost=6824.00 rows=15075) (actual time=0.074..23.804 rows=15000 loops=1)
                -> Table scan on pk (cost=1547.75 rows=15075) (actual time=0.061..5.214 rows=15000 loops=1)
                -> Single-row index lookup on k using PRIMARY (id=pk.keywordId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=15000)
```

The two indices we created did not improve the cost of the query. One possible reason for this is that the tables involved in the query are not large enough to result in an obvious change. Another possible reason for this is because this query is joining on primary keys, so the additional indices may not result in further optimization.

Query 4

No index (Cost=5239.21)

```
| -> Table scan on <union temporary> (cost=5239.21..5422.80 rows=14489) (actual time=24.925..25.152 rows=1660 loops=1)
    -> Union materialize with deduplication (cost=5239.20..5239.20 rows=14489) (actual time=24.921..24.921 rows=1660 loops=1)
        -> Nested loop inner join (cost=435.34 rows=666) (actual time=0.092..3.650 rows=1041 loops=1)
            -> Filter: (up.rating >= 4) (cost=202.15 rows=666) (actual time=0.065..0.984 rows=1041 loops=1)
                -> Table scan on up (cost=202.15 rows=1999) (actual time=0.061..0.787 rows=1999 loops=1)
                -> Single-row index lookup on p using PRIMARY (id=up.paperId) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1041)
            -> Filter: (sum(pk.score) > 100) (cost=3354.98 rows=13823) (actual time=0.112..18.386 rows=812 loops=1)
                -> Group aggregate: sum(pk.score), sum(pk.score) (cost=3354.98 rows=13823) (actual time=0.083..18.002 rows=1500 loops=1)
                    -> Nested loop inner join (cost=1972.72 rows=13823) (actual time=0.055..13.251 rows=15000 loops=1)
                        -> Index scan on p using PRIMARY (cost=207.45 rows=1512) (actual time=0.030..1.126 rows=1500 loops=1)
                        -> Index lookup on pk using PRIMARY (paperId=p.id) (cost=0.25 rows=9) (actual time=0.005..0.007 rows=10 loops=1500)
```

CREATE INDEX pk_score_idx ON PaperKeyword(score); (Cost=5239.21, no change)

```
| -> Table scan on <union temporary> (cost=5239.21..5422.80 rows=14489) (actual time=22.315..22.525 rows=1660 loops=1)
    -> Union materialize with deduplication (cost=5239.20..5239.20 rows=14489) (actual time=22.313..22.313 rows=1660 loops=1)
        -> Nested loop inner join (cost=435.34 rows=666) (actual time=0.103..3.603 rows=1041 loops=1)
            -> Filter: (up.rating >= 4) (cost=202.15 rows=666) (actual time=0.081..1.008 rows=1041 loops=1)
                -> Table scan on up (cost=202.15 rows=1999) (actual time=0.078..0.834 rows=1999 loops=1)
                -> Single-row index lookup on p using PRIMARY (id=up.paperId) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1041)
            -> Filter: (sum(pk.score) > 100) (cost=3354.98 rows=13823) (actual time=0.101..16.220 rows=812 loops=1)
                -> Group aggregate: sum(pk.score), sum(pk.score) (cost=3354.98 rows=13823) (actual time=0.073..15.970 rows=1500 loops=1)
                    -> Nested loop inner join (cost=1972.72 rows=13823) (actual time=0.037..11.408 rows=15000 loops=1)
                        -> Index scan on p using PRIMARY (cost=207.45 rows=1512) (actual time=0.017..0.853 rows=1500 loops=1)
                        -> Index lookup on pk using PRIMARY (paperId=p.id) (cost=0.25 rows=9) (actual time=0.005..0.006 rows=10 loops=1500)
```

CREATE INDEX paper_rating_idx ON UserPaper(rating); (Cost=5435.33)

```

| -> Table scan on <union temporary> (cost=5435.33..5623.60 rows=14864) (actual time=22.742..22.973 rows=1660 loops=1)
|   -> Union materialize with deduplication (cost=5435.31..5435.31 rows=14864) (actual time=22.738..22.738 rows=1660 loops=1)
|     -> Nested loop inner join (cost=593.98 rows=1041) (actual time=0.060..3.208 rows=1041 loops=1)
|       -> Filter: (up.rating >= 4) (cost=229.63 rows=1041) (actual time=0.039..0.524 rows=1041 loops=1)
|         -> Covering index range scan on up using paper_rating_idx over (4 <= rating) (cost=229.63 rows=1041) (actual time=0.037..0.412 rows=1041 loops=1)
|           -> Single-row index lookup on p using PRIMARY (id=up.paperId) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1041)
|         -> Filter: (sum(pk.score) > 100) (cost=3354.98 rows=13823) (actual time=0.098..16.894 rows=812 loops=1)
|       -> Group aggregate: sum(pk.score), sum(pk.score) (cost=3354.98 rows=13823) (actual time=0.071..16.617 rows=1500 loops=1)
|         -> Nested loop inner join (cost=1972.72 rows=13823) (actual time=0.045..11.978 rows=15000 loops=1)
|           -> Index scan on p using PRIMARY (cost=207.45 rows=1512) (actual time=0.021..0.944 rows=1500 loops=1)
|           -> Index lookup on pk using PRIMARY (paperId=p.id) (cost=0.25 rows=9) (actual time=0.005..0.007 rows=10 loops=1500)

```

The first index we created did not improve the cost of the query. One possible reason for this is that the tables involved in the query are not large enough to result in an obvious change. The second index results in a higher cost, which may be due to the low cardinality of the “rating” attribute.

Final Index Design

Since these index designs did not improve the query performance, we will not use any additional indices.

PT1 Stage 0 Resubmission: We fixed the issue of not being able to access the repository.

References:

[1] https://www.w3resource.com/mysql/aggregate-functions-and-grouping/aggregate-functions-and-grouping-group_concat.php