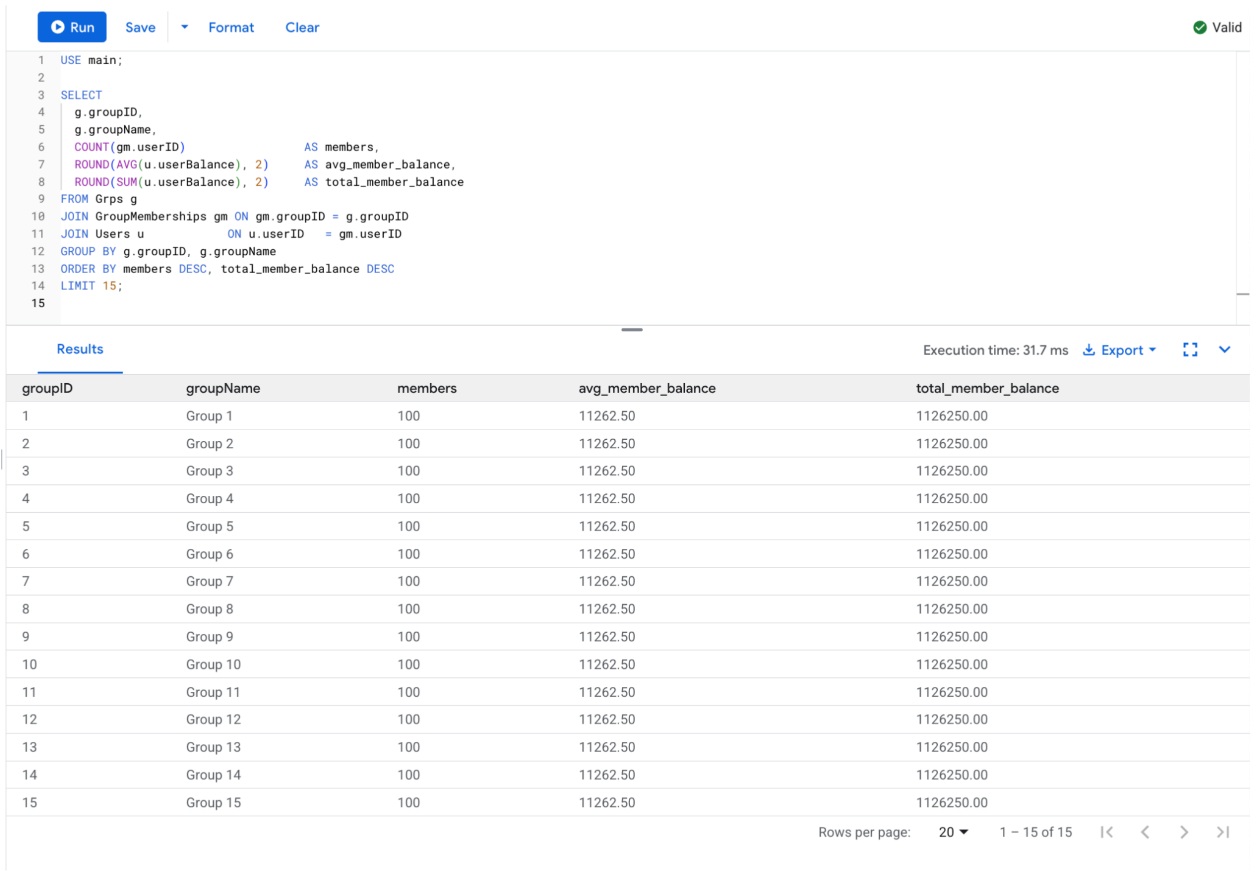
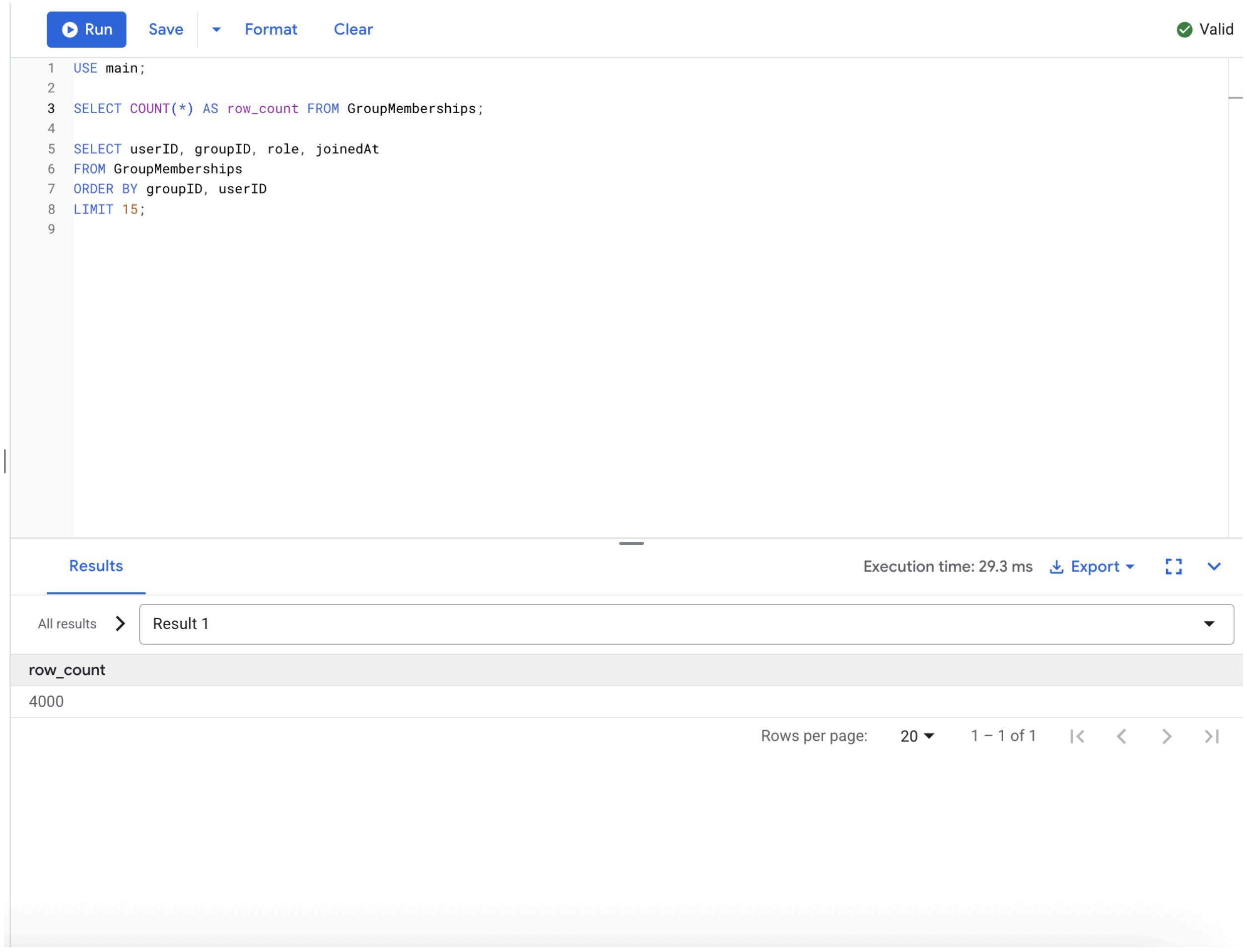
Part 1:



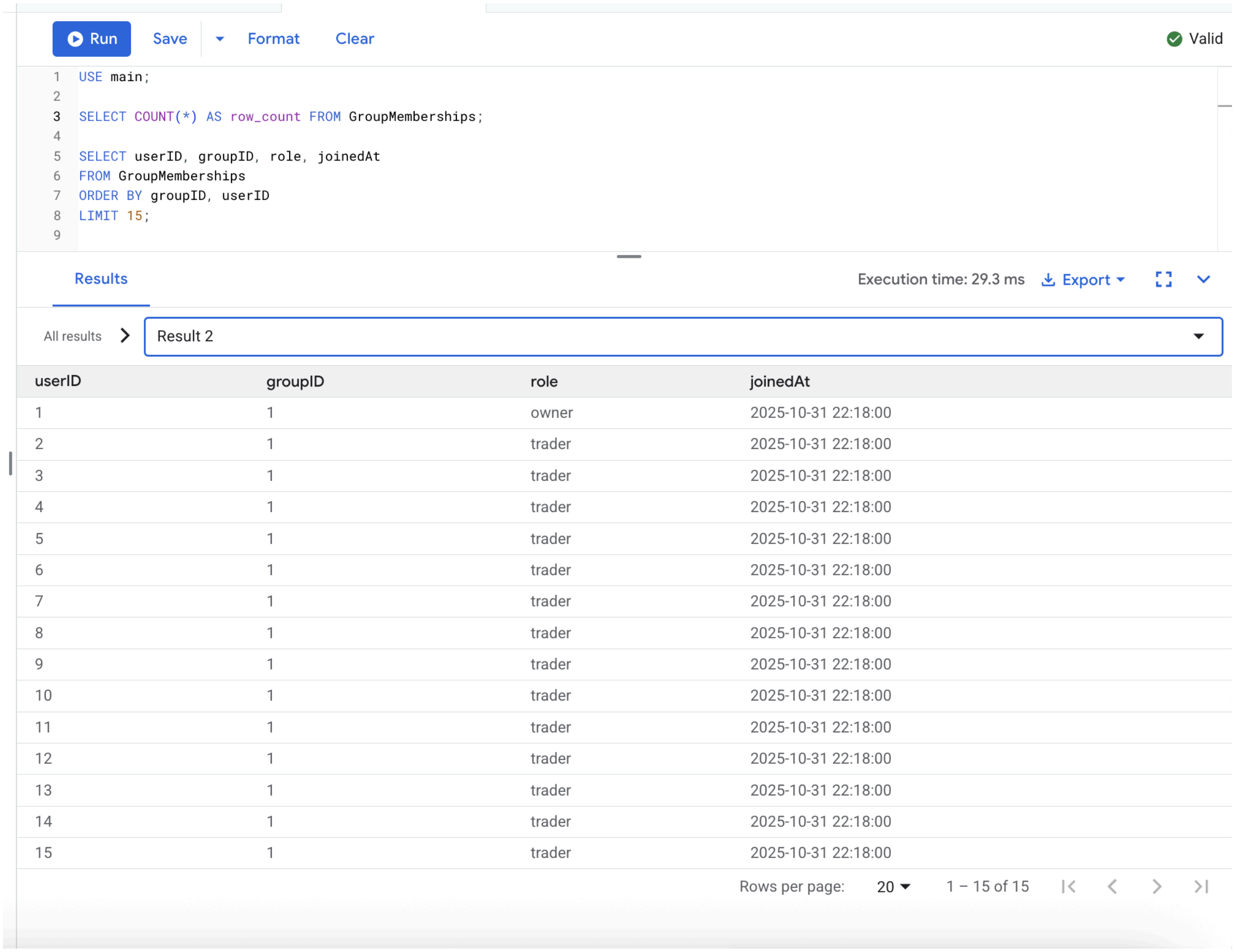
```sql
USE main;

SELECT
  g.groupID,
  g.groupName,
  COUNT(gm.userID)                  AS members,
  ROUND(AVG(u.userBalance), 2)      AS avg_member_balance,
  ROUND(SUM(u.userBalance), 2)      AS total_member_balance
FROM Grps g
JOIN GroupMemberships gm ON gm.groupID = g.groupID
JOIN Users u            ON u.userID    = gm.userID
GROUP BY g.groupID, g.groupName
ORDER BY members DESC, total_member_balance DESC
LIMIT 15;
```

groupID	groupName	members	avg_member_balance	total_member_balance
1	Group 1	100	11262.50	1126250.00
2	Group 2	100	11262.50	1126250.00
3	Group 3	100	11262.50	1126250.00
4	Group 4	100	11262.50	1126250.00
5	Group 5	100	11262.50	1126250.00
6	Group 6	100	11262.50	1126250.00
7	Group 7	100	11262.50	1126250.00
8	Group 8	100	11262.50	1126250.00
9	Group 9	100	11262.50	1126250.00
10	Group 10	100	11262.50	1126250.00
11	Group 11	100	11262.50	1126250.00
12	Group 12	100	11262.50	1126250.00
13	Group 13	100	11262.50	1126250.00
14	Group 14	100	11262.50	1126250.00
15	Group 15	100	11262.50	1126250.00



```sql
USE main;

SELECT COUNT(*) AS row_count FROM GroupMemberships;

SELECT userID, groupID, role, joinedAt
FROM GroupMemberships
ORDER BY groupID, userID
LIMIT 15;
```

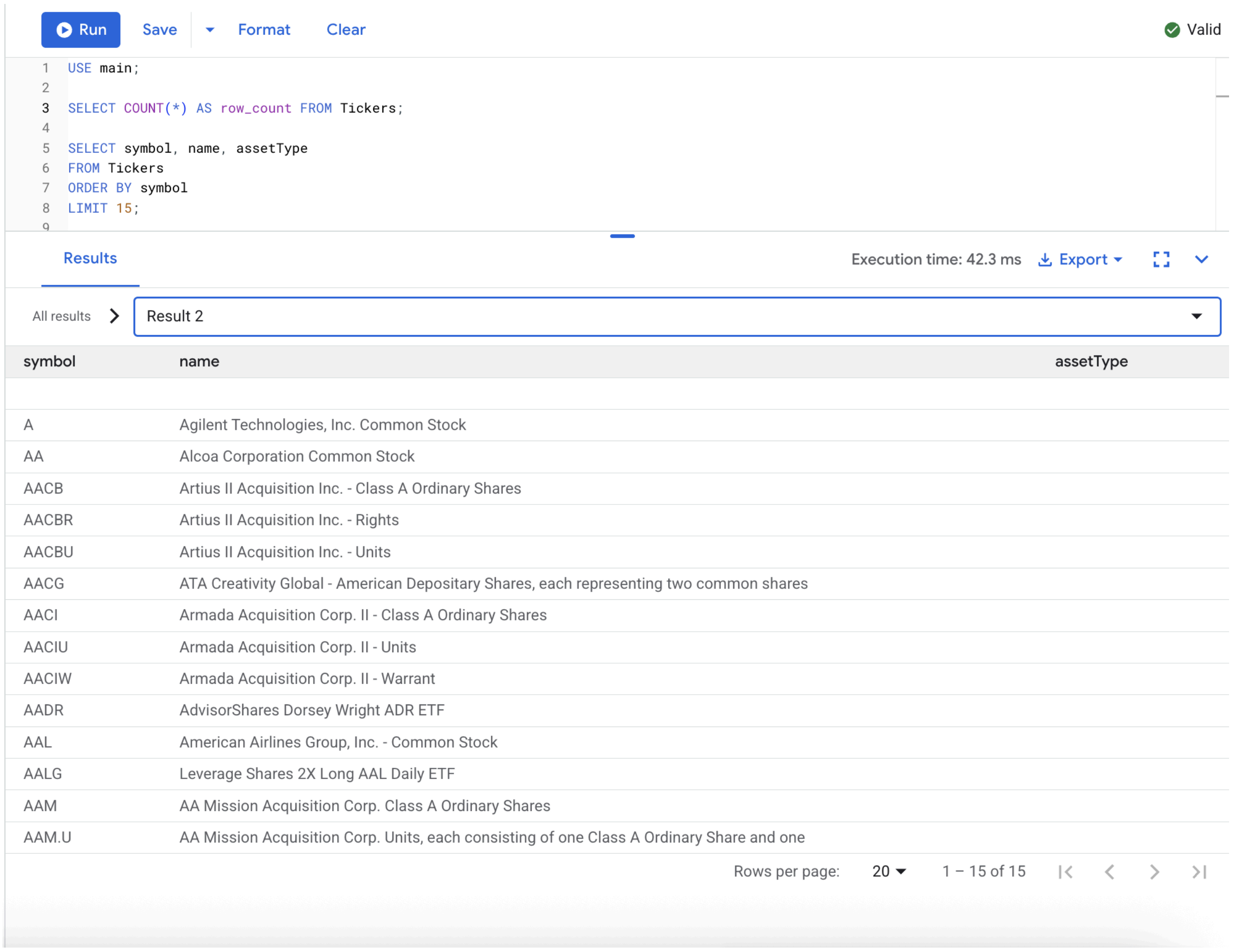
```
row_count
4000
```



```
USE main;

SELECT COUNT(*) AS row_count FROM GroupMemberships;

SELECT userID, groupID, role, joinedAt
FROM GroupMemberships
ORDER BY groupID, userID
LIMIT 15;
```

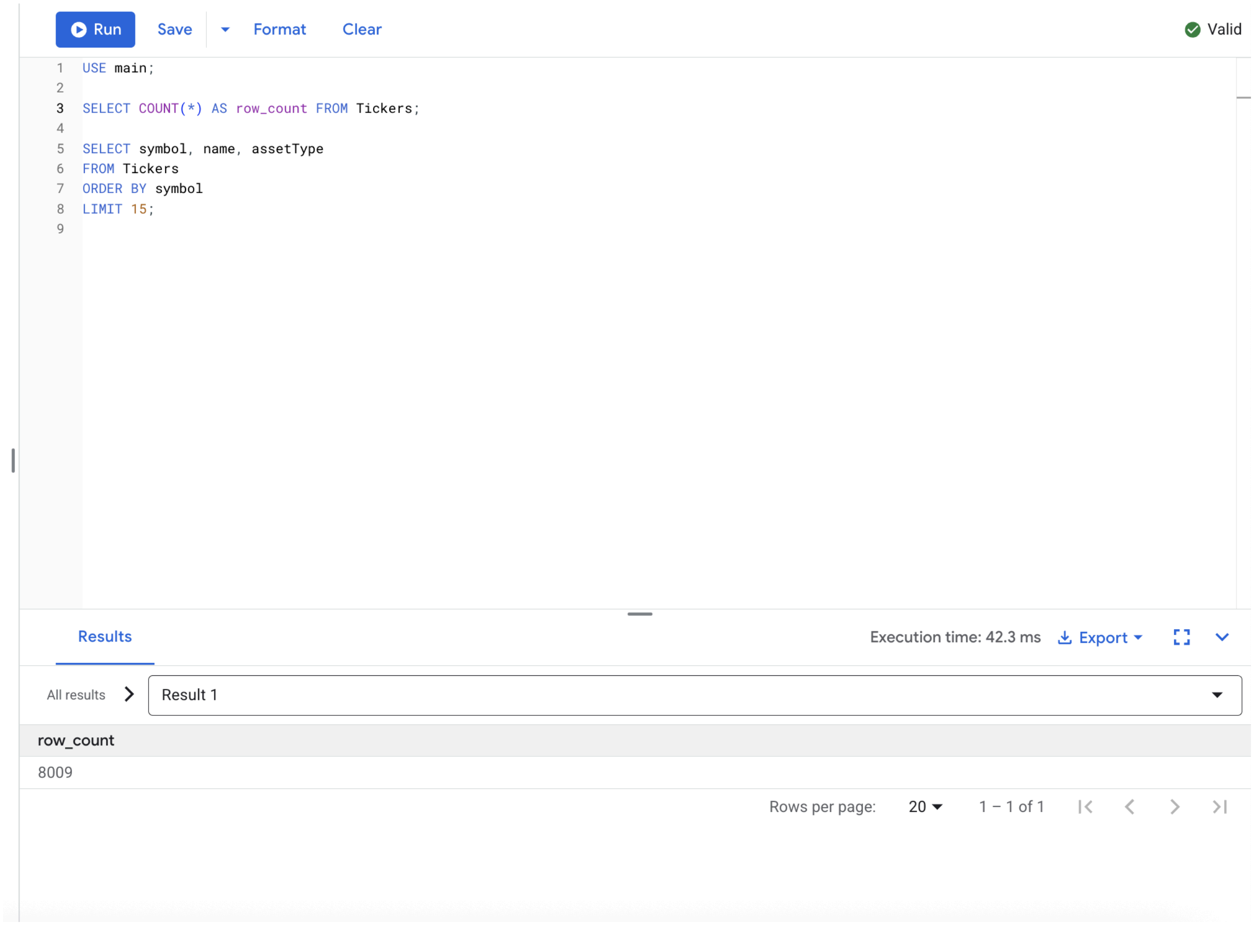
userID	groupID	role	joinedAt
1	1	owner	2025-10-31 22:18:00
2	1	trader	2025-10-31 22:18:00
3	1	trader	2025-10-31 22:18:00
4	1	trader	2025-10-31 22:18:00
5	1	trader	2025-10-31 22:18:00
6	1	trader	2025-10-31 22:18:00
7	1	trader	2025-10-31 22:18:00
8	1	trader	2025-10-31 22:18:00
9	1	trader	2025-10-31 22:18:00
10	1	trader	2025-10-31 22:18:00
11	1	trader	2025-10-31 22:18:00
12	1	trader	2025-10-31 22:18:00
13	1	trader	2025-10-31 22:18:00
14	1	trader	2025-10-31 22:18:00
15	1	trader	2025-10-31 22:18:00

Rows per page: 20 1 – 15 of 15



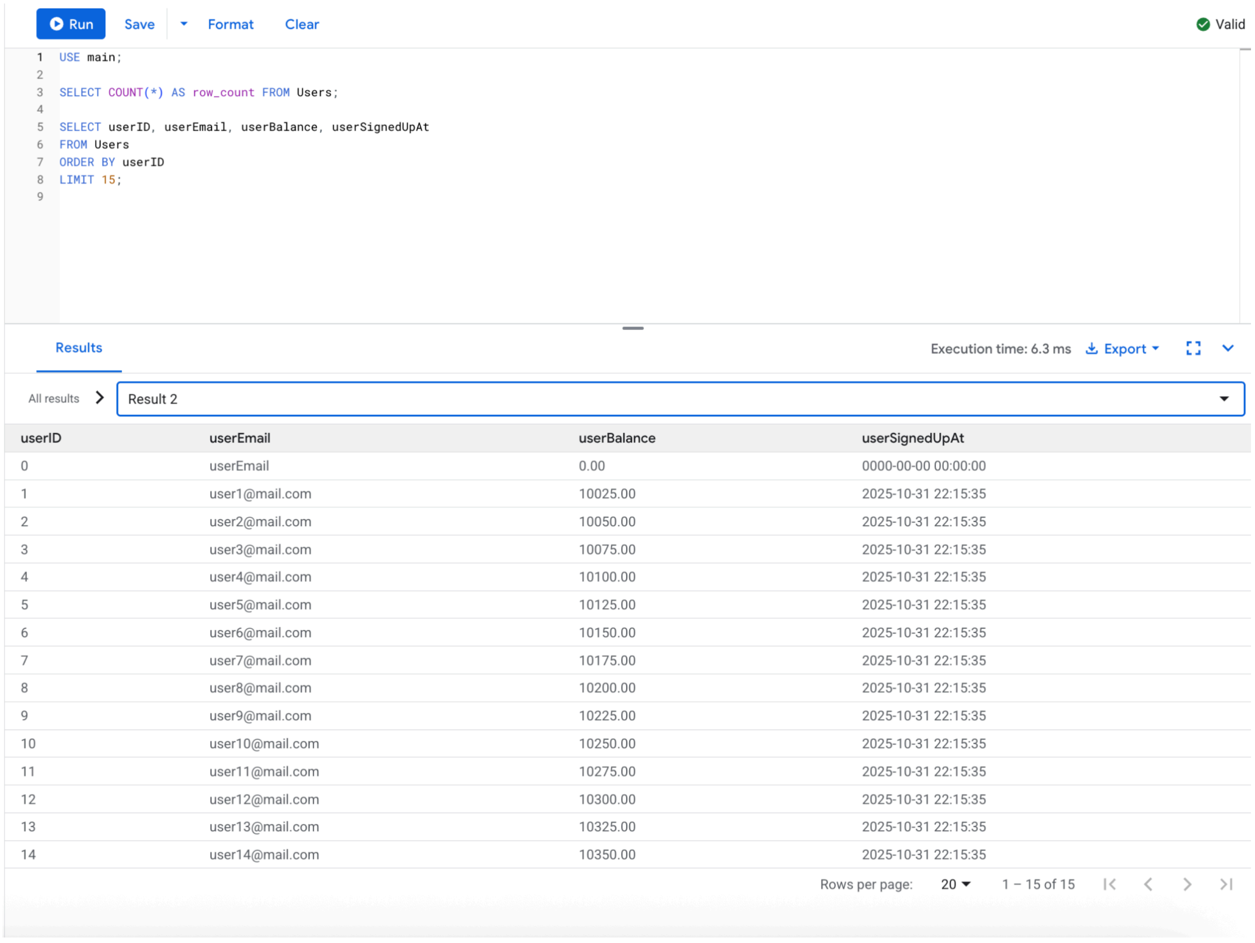
```
USE main;

SELECT COUNT(*) AS row_count FROM Tickers;

SELECT symbol, name, assetType
FROM Tickers
ORDER BY symbol
LIMIT 15;
```

symbol	name	assetType
A	Agilent Technologies, Inc. Common Stock	
AA	Alcoa Corporation Common Stock	
AACB	Artius II Acquisition Inc. - Class A Ordinary Shares	
AACBR	Artius II Acquisition Inc. - Rights	
AACBU	Artius II Acquisition Inc. - Units	
AACG	ATA Creativity Global - American Depositary Shares, each representing two common shares	
AACI	Armada Acquisition Corp. II - Class A Ordinary Shares	
AACIU	Armada Acquisition Corp. II - Units	
AACIW	Armada Acquisition Corp. II - Warrant	
AADR	AdvisorShares Dorsey Wright ADR ETF	
AAL	American Airlines Group, Inc. - Common Stock	
AALG	Leverage Shares 2X Long AAL Daily ETF	
AAM	AA Mission Acquisition Corp. Class A Ordinary Shares	
AAM.U	AA Mission Acquisition Corp. Units, each consisting of one Class A Ordinary Share and one	



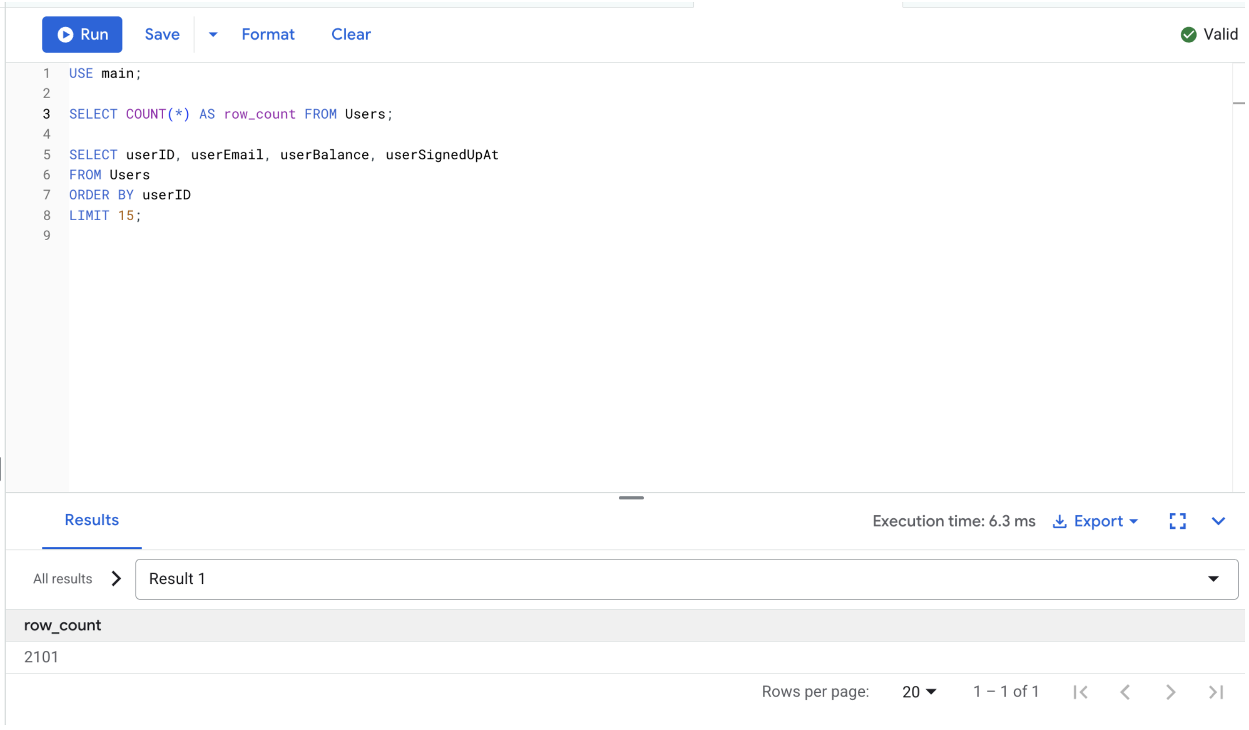
```
USE main;

SELECT COUNT(*) AS row_count FROM Tickers;

SELECT symbol, name, assetType
FROM Tickers
ORDER BY symbol
LIMIT 15;
```

row_count
8009



```sql
USE main;

SELECT COUNT(*) AS row_count FROM Users;

SELECT userID, userEmail, userBalance, userSignedUpAt
FROM Users
ORDER BY userID
LIMIT 15;
```

userID	userEmail	userBalance	userSignedUpAt
0	userEmail	0.00	0000-00-00 00:00:00
1	user1@mail.com	10025.00	2025-10-31 22:15:35
2	user2@mail.com	10050.00	2025-10-31 22:15:35
3	user3@mail.com	10075.00	2025-10-31 22:15:35
4	user4@mail.com	10100.00	2025-10-31 22:15:35
5	user5@mail.com	10125.00	2025-10-31 22:15:35
6	user6@mail.com	10150.00	2025-10-31 22:15:35
7	user7@mail.com	10175.00	2025-10-31 22:15:35
8	user8@mail.com	10200.00	2025-10-31 22:15:35
9	user9@mail.com	10225.00	2025-10-31 22:15:35
10	user10@mail.com	10250.00	2025-10-31 22:15:35
11	user11@mail.com	10275.00	2025-10-31 22:15:35
12	user12@mail.com	10300.00	2025-10-31 22:15:35
13	user13@mail.com	10325.00	2025-10-31 22:15:35
14	user14@mail.com	10350.00	2025-10-31 22:15:35



```sql
USE main;

SELECT COUNT(*) AS row_count FROM Users;

SELECT userID, userEmail, userBalance, userSignedUpAt
FROM Users
ORDER BY userID
LIMIT 15;
```

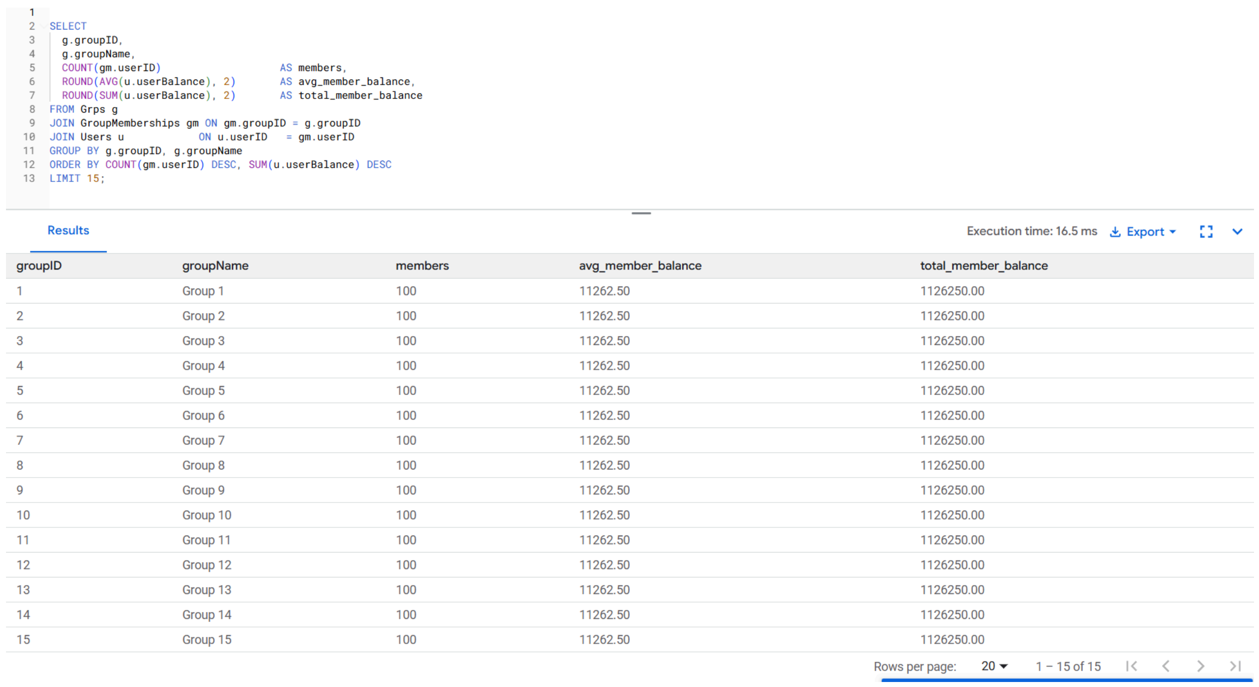
row_count
2101

Part 2:

Query 1:

Variant A reduced plan cost from 4216 to 2216 by aligning the join and GROUP BY with groupID, enabling range scans over GroupMemberships. Variant B did not help because the leading column (userID) did not match the group-driven access pattern; the optimizer continued to choose the groupID path. Adding covering indexes on Users and Grps (Variant C) did not further reduce cost on our dataset. Users is small and remain cached, so index-only reads brought negligible savings. We select Variant A as the final design. Trade-off: slightly higher write cost and index storage, acceptable for our read-heavy analytics.

Baseline Results: Cost = 4216



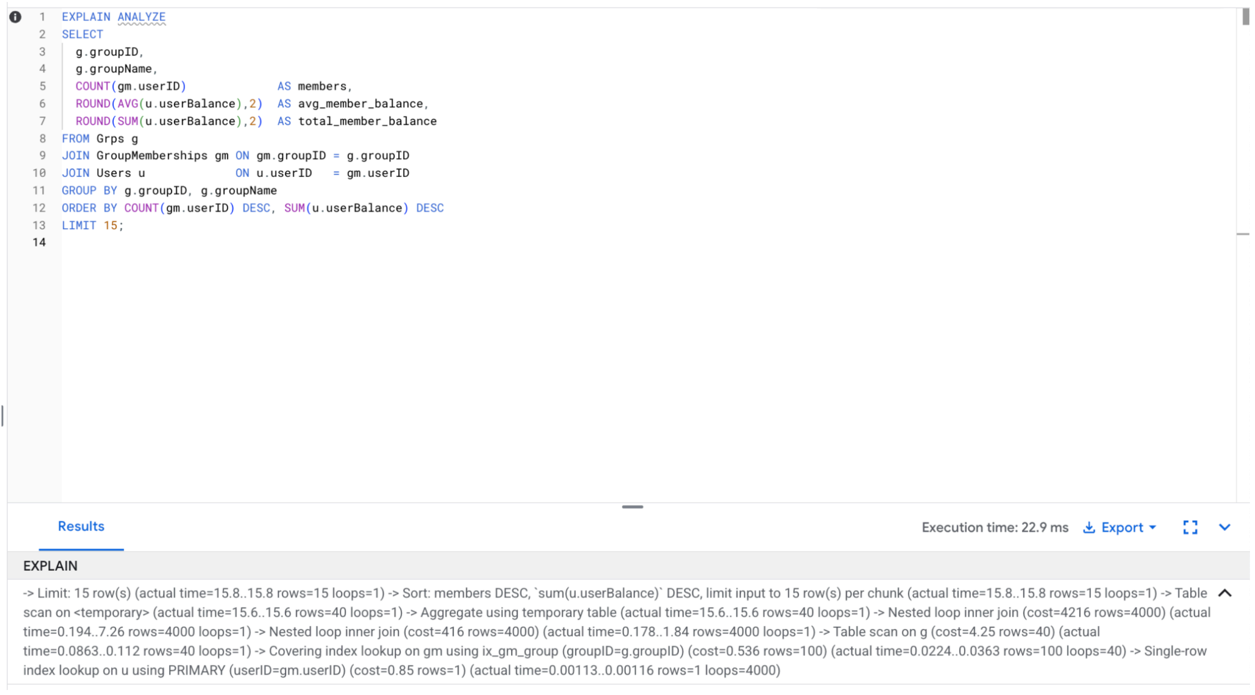
```sql

SELECT
  g.groupID,
  g.groupName,
  COUNT(gm.userID)                    AS members,
  ROUND(AVG(u.userBalance), 2)        AS avg_member_balance,
  ROUND(SUM(u.userBalance), 2)        AS total_member_balance
FROM Grps g
JOIN GroupMemberships gm ON gm.groupID = g.groupID
JOIN Users u             ON u.userID   = gm.userID
GROUP BY g.groupID, g.groupName
ORDER BY COUNT(gm.userID) DESC, SUM(u.userBalance) DESC
LIMIT 15;
```

Results

Execution time: 16.5 ms Export

groupID	groupName	members	avg_member_balance	total_member_balance
1	Group 1	100	11262.50	1126250.00
2	Group 2	100	11262.50	1126250.00
3	Group 3	100	11262.50	1126250.00
4	Group 4	100	11262.50	1126250.00
5	Group 5	100	11262.50	1126250.00
6	Group 6	100	11262.50	1126250.00
7	Group 7	100	11262.50	1126250.00
8	Group 8	100	11262.50	1126250.00
9	Group 9	100	11262.50	1126250.00
10	Group 10	100	11262.50	1126250.00
11	Group 11	100	11262.50	1126250.00
12	Group 12	100	11262.50	1126250.00
13	Group 13	100	11262.50	1126250.00
14	Group 14	100	11262.50	1126250.00
15	Group 15	100	11262.50	1126250.00

Rows per page: 20 1 – 15 of 15

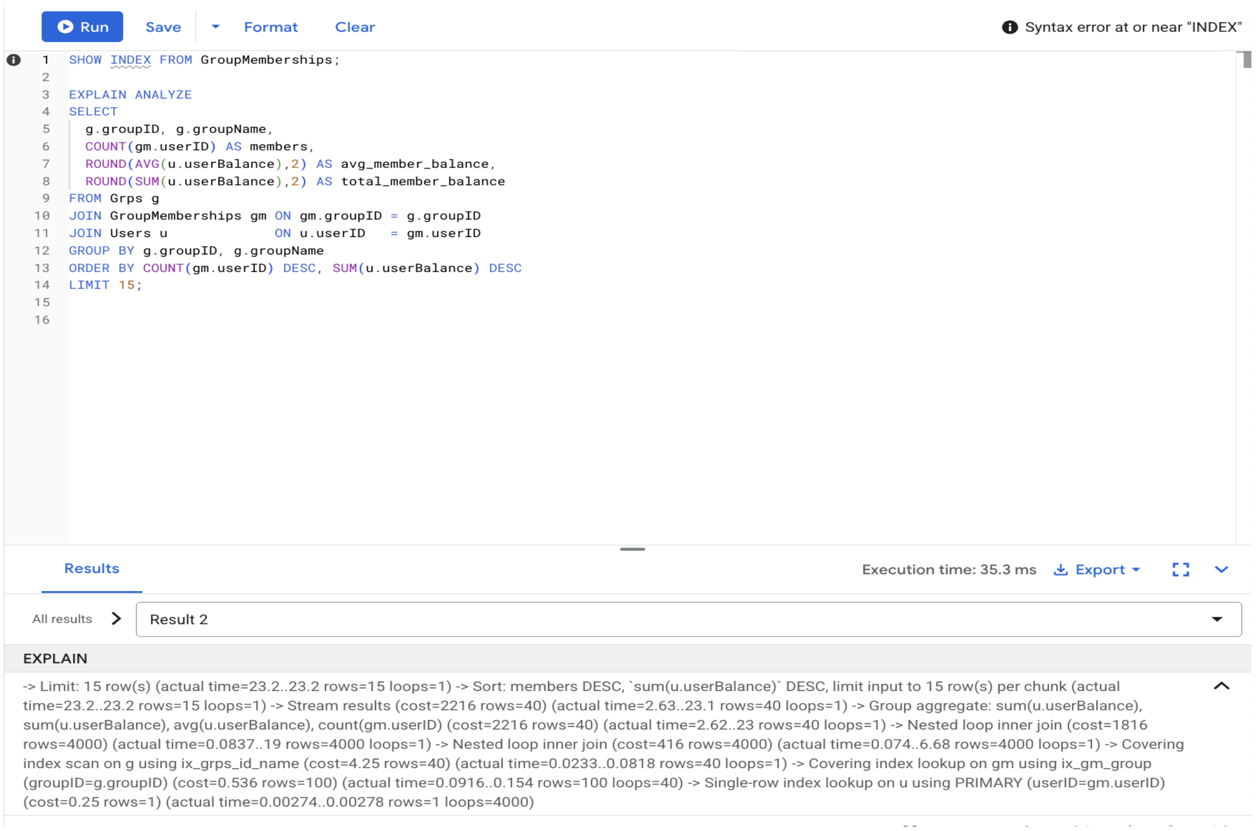


```
EXPLAIN ANALYZE
SELECT
  g.groupID,
  g.groupName,
  COUNT(gm.userID)             AS members,
  ROUND(AVG(u.userBalance),2)  AS avg_member_balance,
  ROUND(SUM(u.userBalance),2)  AS total_member_balance
FROM Grps g
JOIN GroupMemberships gm ON gm.groupID = g.groupID
JOIN Users u             ON u.userID   = gm.userID
GROUP BY g.groupID, g.groupName
ORDER BY COUNT(gm.userID) DESC, SUM(u.userBalance) DESC
LIMIT 15;
```

EXPLAIN

```
-> Limit: 15 row(s) (actual time=15.8..15.8 rows=15 loops=1) -> Sort: members DESC, `sum(u.userBalance)` DESC, limit input to 15 row(s) per chunk (actual time=15.8..15.8 rows=15 loops=1) -> Table scan on <temporary> (actual time=15.6..15.6 rows=40 loops=1) -> Aggregate using temporary table (actual time=15.6..15.6 rows=40 loops=1) -> Nested loop inner join (cost=4216 rows=4000) (actual time=0.194..7.26 rows=4000 loops=1) -> Nested loop inner join (cost=416 rows=4000) (actual time=0.178..1.84 rows=4000 loops=1) -> Table scan on g (cost=4.25 rows=40) (actual time=0.0863..0.112 rows=40 loops=1) -> Covering index lookup on gm using ix_gm_group (groupID=g.groupID) (cost=0.536 rows=100) (actual time=0.0224..0.0363 rows=100 loops=40) -> Single-row index lookup on u using PRIMARY (userID=gm.userID) (cost=0.85 rows=1) (actual time=0.00113..0.00116 rows=1 loops=4000)
```

Variant A Results: Stream results (cost=2216 rows=40)



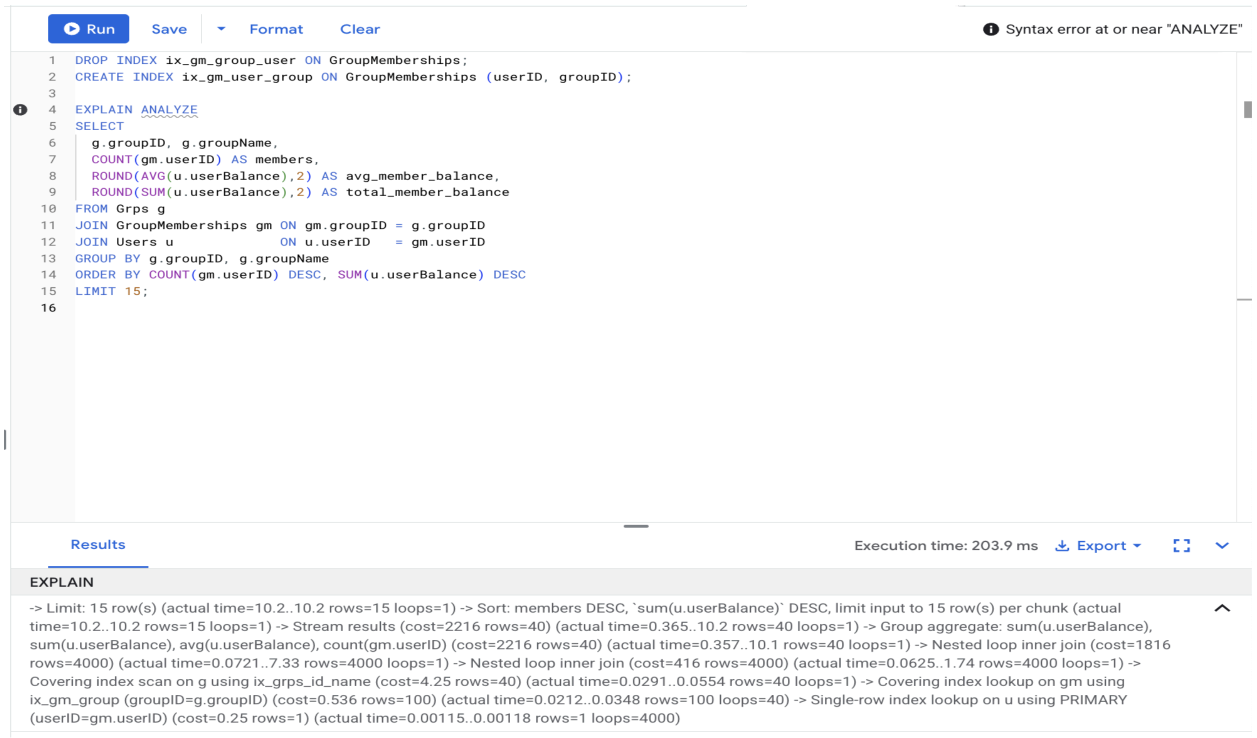
Syntax error at or near "INDEX"

```sql
SHOW INDEX FROM GroupMemberships;

EXPLAIN ANALYZE
SELECT
  g.groupID, g.groupName,
  COUNT(gm.userID) AS members,
  ROUND(AVG(u.userBalance),2) AS avg_member_balance,
  ROUND(SUM(u.userBalance),2) AS total_member_balance
FROM Grps g
JOIN GroupMemberships gm ON gm.groupID = g.groupID
JOIN Users u             ON u.userID   = gm.userID
GROUP BY g.groupID, g.groupName
ORDER BY COUNT(gm.userID) DESC, SUM(u.userBalance) DESC
LIMIT 15;
```

EXPLAIN

```
-> Limit: 15 row(s) (actual time=23.2..23.2 rows=15 loops=1) -> Sort: members DESC, `sum(u.userBalance)` DESC, limit input to 15 row(s) per chunk (actual time=23.2..23.2 rows=15 loops=1) -> Stream results (cost=2216 rows=40) (actual time=2.63..23.1 rows=40 loops=1) -> Group aggregate: sum(u.userBalance), sum(u.userBalance), avg(u.userBalance), count(gm.userID) (cost=2216 rows=40) (actual time=2.62..23 rows=40 loops=1) -> Nested loop inner join (cost=1816 rows=4000) (actual time=0.0837..19 rows=4000 loops=1) -> Nested loop inner join (cost=416 rows=4000) (actual time=0.074..6.68 rows=4000 loops=1) -> Covering index scan on g using ix_grps_id_name (cost=4.25 rows=40) (actual time=0.0233..0.0818 rows=40 loops=1) -> Covering index lookup on gm using ix_gm_group (groupID=g.groupID) (cost=0.536 rows=100) (actual time=0.0916..0.154 rows=100 loops=40) -> Single-row index lookup on u using PRIMARY (userID=gm.userID) (cost=0.25 rows=1) (actual time=0.00274..0.00278 rows=1 loops=4000)
```

Variant B Results: Stream results (cost=2216 rows=40)



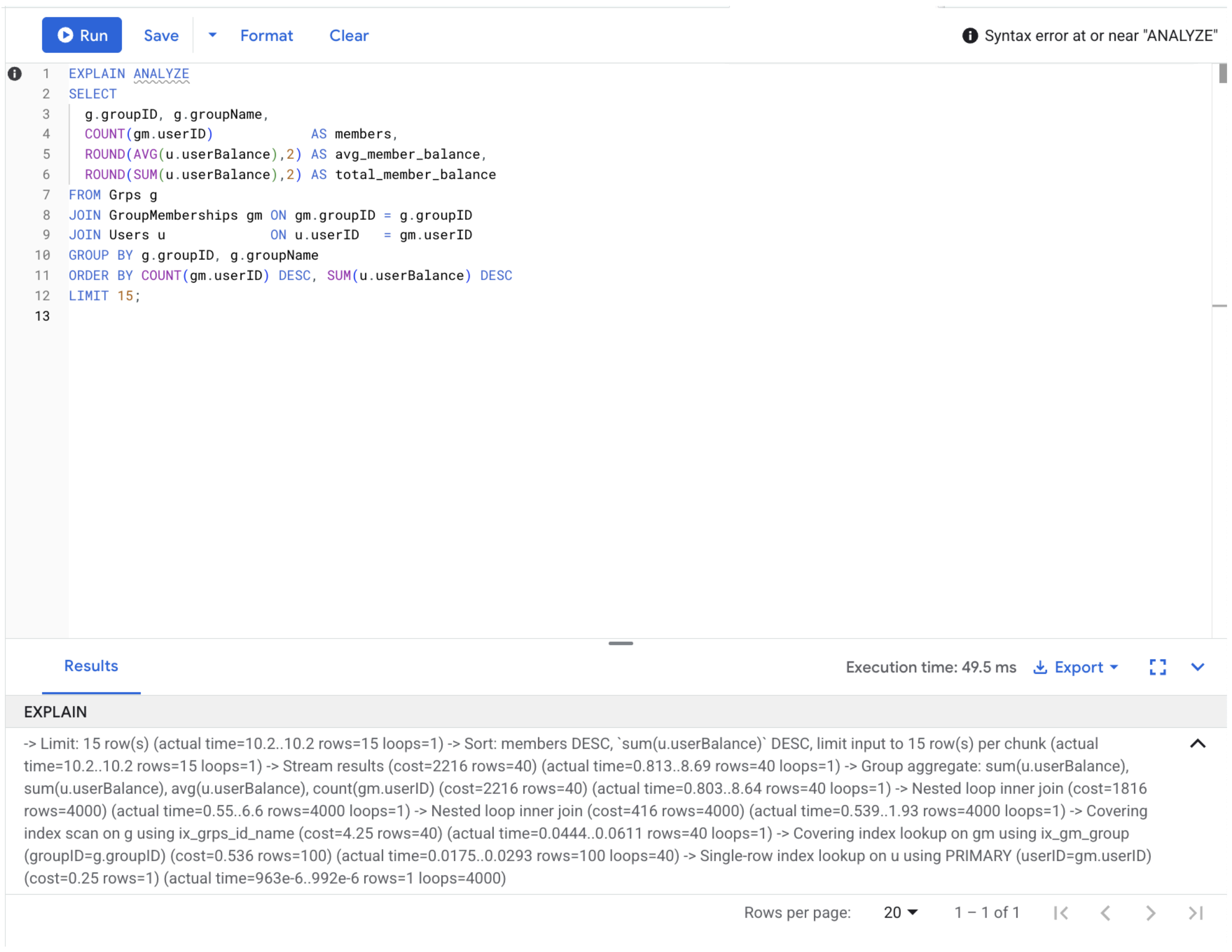
Syntax error at or near "ANALYZE"

```sql
DROP INDEX ix_gm_group_user ON GroupMemberships;
CREATE INDEX ix_gm_user_group ON GroupMemberships (userID, groupID);

EXPLAIN ANALYZE
SELECT
  g.groupID, g.groupName,
  COUNT(gm.userID) AS members,
  ROUND(AVG(u.userBalance),2) AS avg_member_balance,
  ROUND(SUM(u.userBalance),2) AS total_member_balance
FROM Grps g
JOIN GroupMemberships gm ON gm.groupID = g.groupID
JOIN Users u             ON u.userID   = gm.userID
GROUP BY g.groupID, g.groupName
ORDER BY COUNT(gm.userID) DESC, SUM(u.userBalance) DESC
LIMIT 15;
```

EXPLAIN

```
-> Limit: 15 row(s) (actual time=10.2..10.2 rows=15 loops=1) -> Sort: members DESC, `sum(u.userBalance)` DESC, limit input to 15 row(s) per chunk (actual time=10.2..10.2 rows=15 loops=1) -> Stream results (cost=2216 rows=40) (actual time=0.365..10.2 rows=40 loops=1) -> Group aggregate: sum(u.userBalance), sum(u.userBalance), avg(u.userBalance), count(gm.userID) (cost=2216 rows=40) (actual time=0.357..10.1 rows=40 loops=1) -> Nested loop inner join (cost=1816 rows=4000) (actual time=0.0721..7.33 rows=4000 loops=1) -> Nested loop inner join (cost=416 rows=4000) (actual time=0.0625..1.74 rows=4000 loops=1) -> Covering index scan on g using ix_grps_id_name (cost=4.25 rows=40) (actual time=0.0291..0.0554 rows=40 loops=1) -> Covering index lookup on gm using ix_gm_group (groupID=g.groupID) (cost=0.536 rows=100) (actual time=0.0212..0.0348 rows=100 loops=40) -> Single-row index lookup on u using PRIMARY (userID=gm.userID) (cost=0.25 rows=1) (actual time=0.00115..0.00118 rows=1 loops=4000)
```

Variant C Results: Stream results (cost=2216 rows=40)



Syntax error at or near "ANALYZE"

```
EXPLAIN ANALYZE
SELECT
  g.groupID, g.groupName,
  COUNT(gm.userID)            AS members,
  ROUND(AVG(u.userBalance),2) AS avg_member_balance,
  ROUND(SUM(u.userBalance),2) AS total_member_balance
FROM Grps g
JOIN GroupMemberships gm ON gm.groupID = g.groupID
JOIN Users u             ON u.userID   = gm.userID
GROUP BY g.groupID, g.groupName
ORDER BY COUNT(gm.userID) DESC, SUM(u.userBalance) DESC
LIMIT 15;
```

Execution time: 49.5 ms

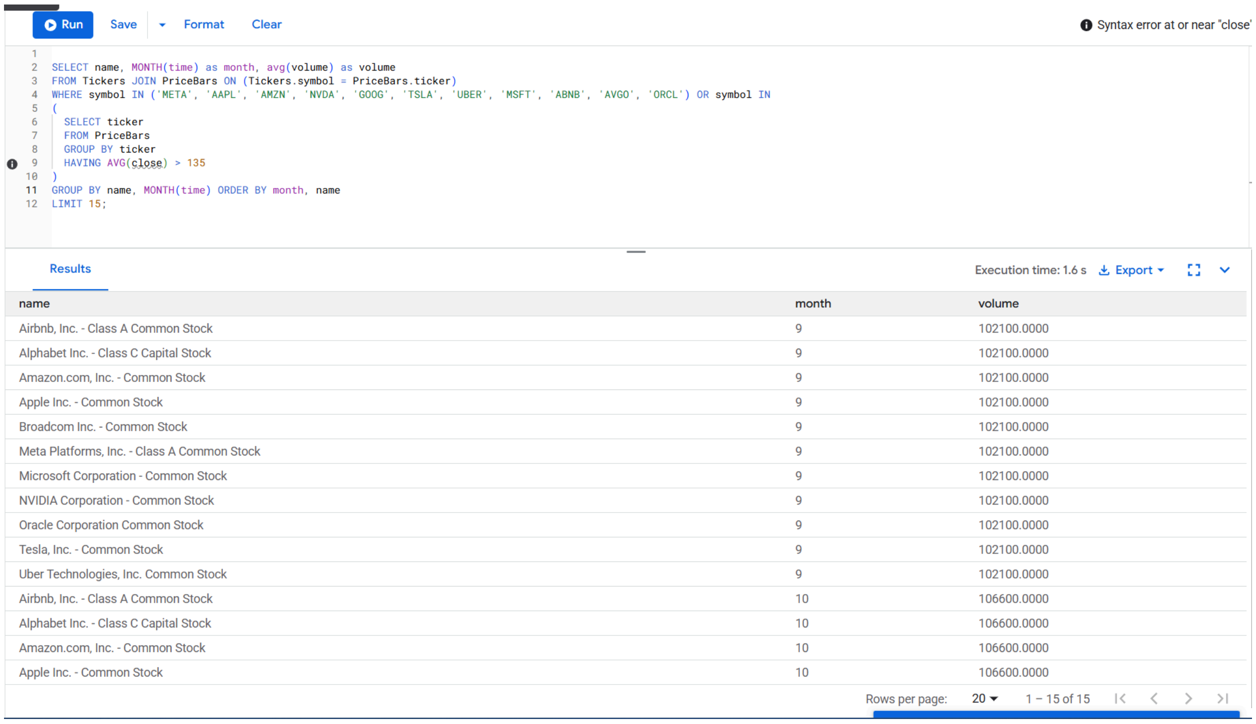
EXPLAIN

```
-> Limit: 15 row(s) (actual time=10.2..10.2 rows=15 loops=1) -> Sort: members DESC, `sum(u.userBalance)` DESC, limit input to 15 row(s) per chunk (actual time=10.2..10.2 rows=15 loops=1) -> Stream results (cost=2216 rows=40) (actual time=0.813..8.69 rows=40 loops=1) -> Group aggregate: sum(u.userBalance), sum(u.userBalance), avg(u.userBalance), count(gm.userID) (cost=2216 rows=40) (actual time=0.803..8.64 rows=40 loops=1) -> Nested loop inner join (cost=1816 rows=4000) (actual time=0.55..6.6 rows=4000 loops=1) -> Nested loop inner join (cost=416 rows=4000) (actual time=0.539..1.93 rows=4000 loops=1) -> Covering index scan on g using ix_grps_id_name (cost=4.25 rows=40) (actual time=0.0444..0.0611 rows=40 loops=1) -> Covering index lookup on gm using ix_gm_group (groupID=g.groupID) (cost=0.536 rows=100) (actual time=0.0175..0.0293 rows=100 loops=40) -> Single-row index lookup on u using PRIMARY (userID=gm.userID) (cost=0.25 rows=1) (actual time=963e-6..992e-6 rows=1 loops=4000)
```

Query 2:

Based on the three variants, all decreased the total cost, although to varying degrees. When we did index on close and volume (A and B), both had a similar decrease to the cost, which makes sense as both are used in aggregation (close being in the having clause in subquery, and volume being returned in the outer query). Variant C had the largest impact, which is reasonable since it combined volume (which was related to the aggregation) and time which was related to grouping.

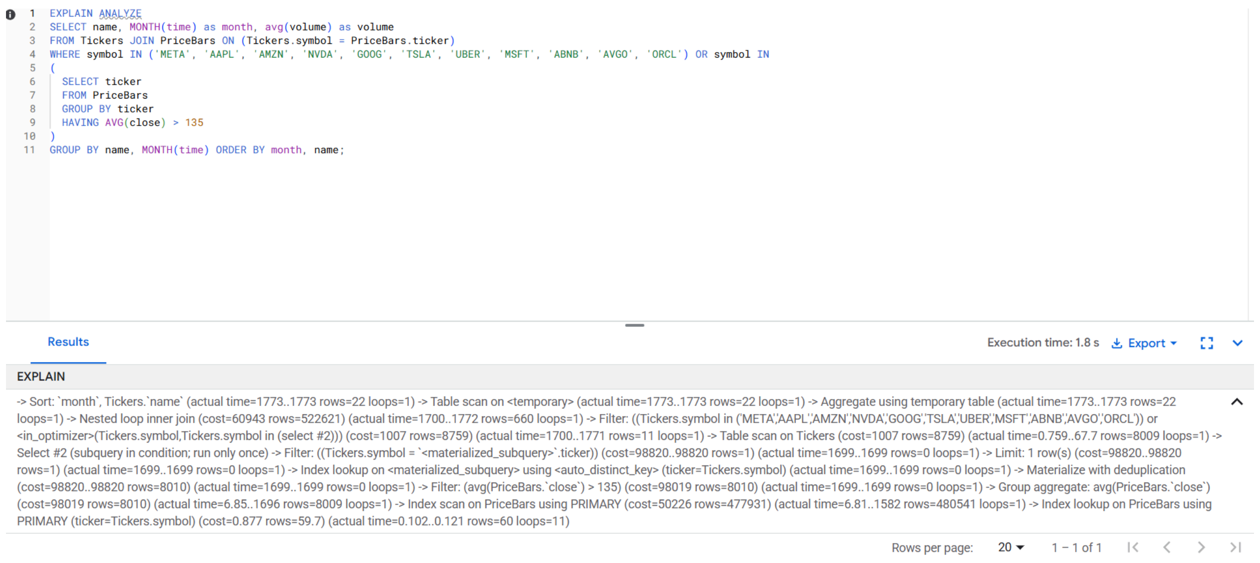
Baseline cost: 61204



```sql

SELECT name, MONTH(time) as month, avg(volume) as volume
FROM Tickers JOIN PriceBars ON (Tickers.symbol = PriceBars.ticker)
WHERE symbol IN ('META', 'AAPL', 'AMZN', 'NVDA', 'GOOG', 'TSLA', 'UBER', 'MSFT', 'ABNB', 'AVGO', 'ORCL') OR symbol IN
(
  SELECT ticker
  FROM PriceBars
  GROUP BY ticker
  HAVING AVG(close) > 135
)
GROUP BY name, MONTH(time) ORDER BY month, name
LIMIT 15;
```

name	month	volume
Airbnb, Inc. - Class A Common Stock	9	102100.0000
Alphabet Inc. - Class C Capital Stock	9	102100.0000
Amazon.com, Inc. - Common Stock	9	102100.0000
Apple Inc. - Common Stock	9	102100.0000
Broadcom Inc. - Common Stock	9	102100.0000
Meta Platforms, Inc. - Class A Common Stock	9	102100.0000
Microsoft Corporation - Common Stock	9	102100.0000
NVIDIA Corporation - Common Stock	9	102100.0000
Oracle Corporation Common Stock	9	102100.0000
Tesla, Inc. - Common Stock	9	102100.0000
Uber Technologies, Inc. Common Stock	9	102100.0000
Airbnb, Inc. - Class A Common Stock	10	106600.0000
Alphabet Inc. - Class C Capital Stock	10	106600.0000
Amazon.com, Inc. - Common Stock	10	106600.0000
Apple Inc. - Common Stock	10	106600.0000

Rows per page: 20 1 – 15 of 15



```sql
EXPLAIN ANALYZE
SELECT name, MONTH(time) as month, avg(volume) as volume
FROM Tickers JOIN PriceBars ON (Tickers.symbol = PriceBars.ticker)
WHERE symbol IN ('META', 'AAPL', 'AMZN', 'NVDA', 'GOOG', 'TSLA', 'UBER', 'MSFT', 'ABNB', 'AVGO', 'ORCL') OR symbol IN
(
  SELECT ticker
  FROM PriceBars
  GROUP BY ticker
  HAVING AVG(close) > 135
)
GROUP BY name, MONTH(time) ORDER BY month, name;
```

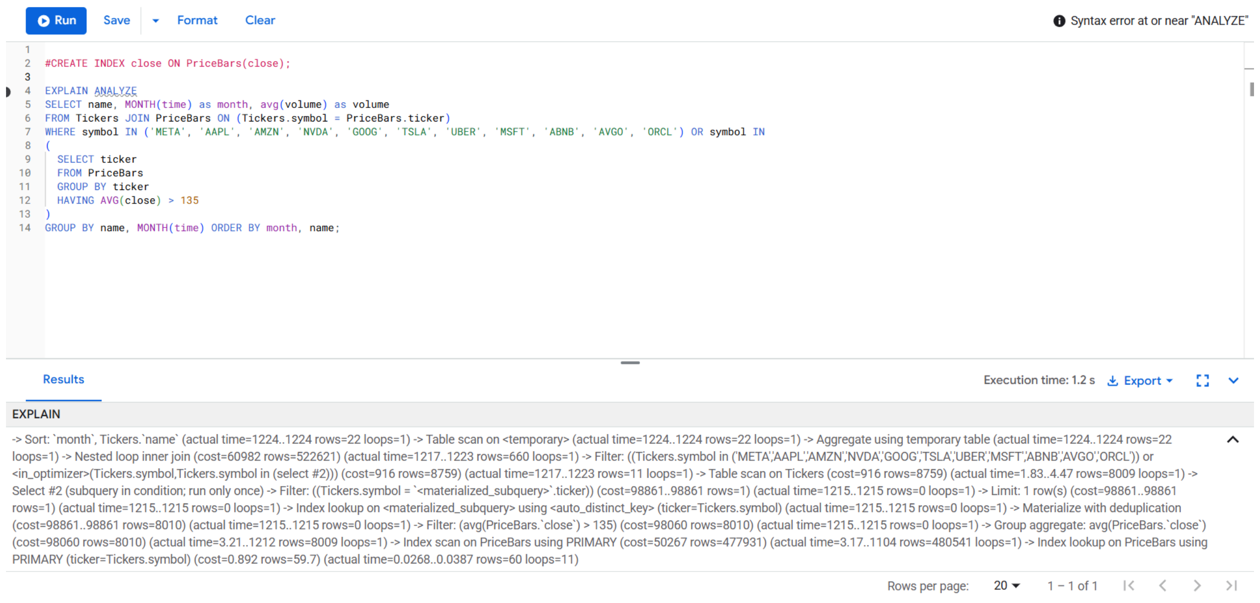
Execution time: 1.8 s

EXPLAIN

```
-> Sort: `month`, Tickers.`name` (actual time=1773..1773 rows=22 loops=1) -> Table scan on <temporary> (actual time=1773..1773 rows=22 loops=1) -> Aggregate using temporary table (actual time=1773..1773 rows=22 loops=1) -> Nested loop inner join (cost=60943 rows=522621) (actual time=1700..1772 rows=660 loops=1) -> Filter: ((Tickers.symbol in ('META','AAPL','AMZN','NVDA','GOOG','TSLA','UBER','MSFT','ABNB','AVGO','ORCL')) or <in_optimizer>(Tickers.symbol,Tickers.symbol in (select #2))) (cost=1007 rows=8759) (actual time=1700..1771 rows=11 loops=1) -> Table scan on Tickers (cost=1007 rows=8759) (actual time=0.759..67.7 rows=8009 loops=1) -> Select #2 (subquery in condition; run only once) -> Filter: ((Tickers.symbol = `<materialized_subquery>`.ticker)) (cost=98820..98820 rows=1) (actual time=1699..1699 rows=0 loops=1) -> Limit: 1 row(s) (cost=98820..98820 rows=1) (actual time=1699..1699 rows=0 loops=1) -> Index lookup on <materialized_subquery> using <auto_distinct_key> (ticker=Tickers.symbol) (actual time=1699..1699 rows=0 loops=1) -> Materialize with deduplication (cost=98820..98820 rows=8010) (actual time=1699..1699 rows=0 loops=1) -> Filter: (avg(PriceBars.`close`) > 135) (cost=98019 rows=8010) (actual time=1699..1699 rows=0 loops=1) -> Group aggregate: avg(PriceBars.`close`) (cost=98019 rows=8010) (actual time=6.85..1696 rows=8009 loops=1) -> Index scan on PriceBars using PRIMARY (cost=50226 rows=477931) (actual time=6.81..1582 rows=480541 loops=1) -> Index lookup on PriceBars using PRIMARY (ticker=Tickers.symbol) (cost=0.877 rows=59.7) (actual time=0.102..0.121 rows=60 loops=11)
```

Rows per page: 20 1 – 1 of 1

Variant A:
Cost: 60982



Run Save Format Clear

Syntax error at or near "ANALYZE"

```sql

#CREATE INDEX close ON PriceBars(close);

EXPLAIN ANALYZE
SELECT name, MONTH(time) as month, avg(volume) as volume
FROM Tickers JOIN PriceBars ON (Tickers.symbol = PriceBars.ticker)
WHERE symbol IN ('META', 'AAPL', 'AMZN', 'NVDA', 'GOOG', 'TSLA', 'UBER', 'MSFT', 'ABNB', 'AVGO', 'ORCL') OR symbol IN
(
  SELECT ticker
  FROM PriceBars
  GROUP BY ticker
  HAVING AVG(close) > 135
)
GROUP BY name, MONTH(time) ORDER BY month, name;
```

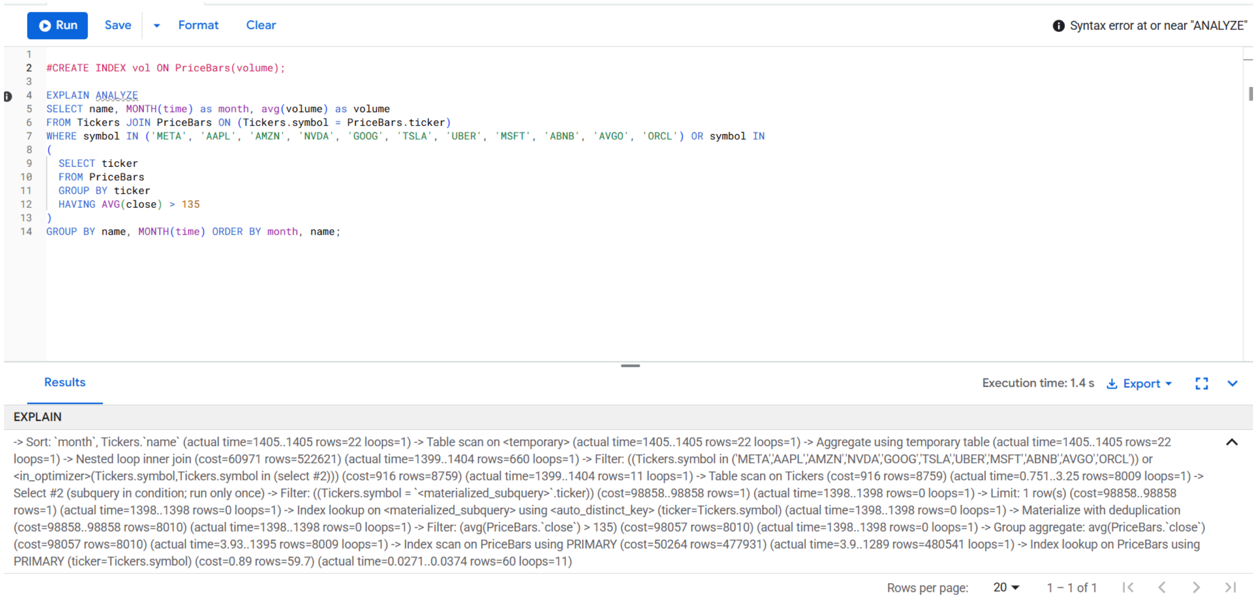
Execution time: 1.2 s

EXPLAIN

```
-> Sort: `month`, Tickers.`name` (actual time=1224..1224 rows=22 loops=1) -> Table scan on <temporary> (actual time=1224..1224 rows=22 loops=1) -> Aggregate using temporary table (actual time=1224..1224 rows=22 loops=1) -> Nested loop inner join (cost=60982 rows=522621) (actual time=1217..1223 rows=660 loops=1) -> Filter: ((Tickers.symbol in ('META','AAPL','AMZN','NVDA','GOOG','TSLA','UBER','MSFT','ABNB','AVGO','ORCL')) or <in_optimizer>(Tickers.symbol,Tickers.symbol in (select #2))) (cost=916 rows=8759) (actual time=1217..1223 rows=11 loops=1) -> Table scan on Tickers (cost=916 rows=8759) (actual time=1.83..4.47 rows=8009 loops=1) -> Select #2 (subquery in condition; run only once) -> Filter: ((Tickers.symbol = `<materialized_subquery>`.ticker)) (cost=98861..98861 rows=1) (actual time=1215..1215 rows=0 loops=1) -> Limit: 1 row(s) (cost=98861..98861 rows=1) (actual time=1215..1215 rows=0 loops=1) -> Index lookup on <materialized_subquery> using <auto_distinct_key> (ticker=Tickers.symbol) (actual time=1215..1215 rows=0 loops=1) -> Materialize with deduplication (cost=98861..98861 rows=8010) (actual time=1215..1215 rows=0 loops=1) -> Filter: (avg(PriceBars.`close`) > 135) (cost=98060 rows=8010) (actual time=1215..1215 rows=0 loops=1) -> Group aggregate: avg(PriceBars.`close`) (cost=98060 rows=8010) (actual time=3.21..1212 rows=8009 loops=1) -> Index scan on PriceBars using PRIMARY (cost=50267 rows=477931) (actual time=3.17..1104 rows=480541 loops=1) -> Index lookup on PriceBars using PRIMARY (ticker=Tickers.symbol) (cost=0.892 rows=59.7) (actual time=0.0268..0.0387 rows=60 loops=11)
```

Rows per page: 20 1 – 1 of 1

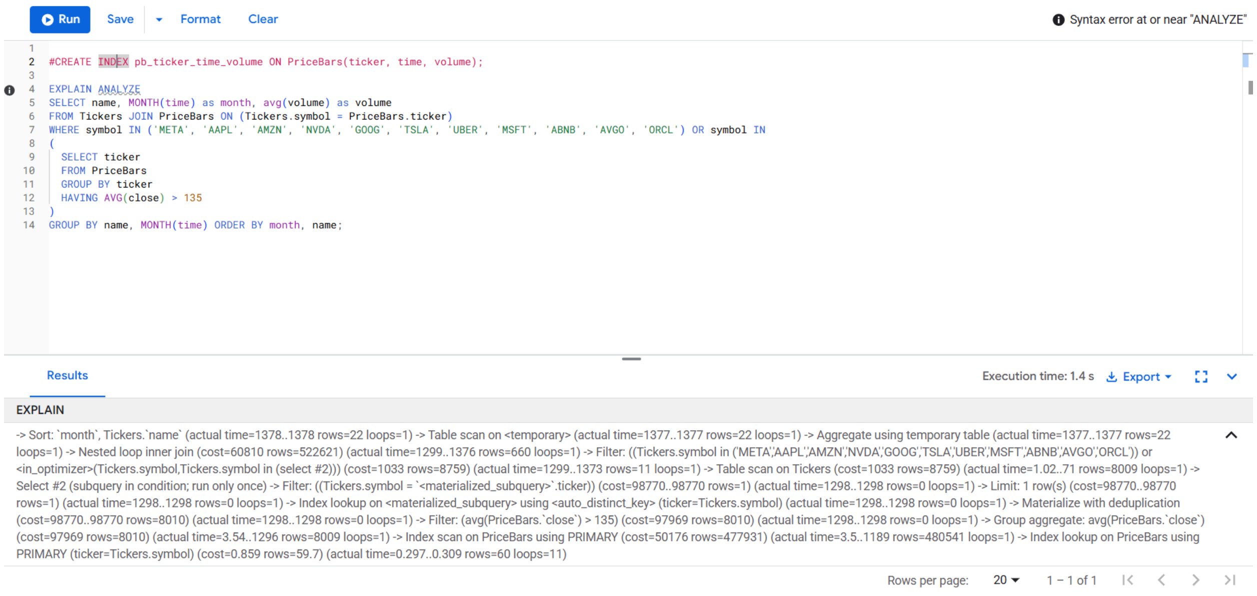
Variant B:
Cost: 60971



```sql

#CREATE INDEX vol ON PriceBars(volume);

EXPLAIN ANALYZE
SELECT name, MONTH(time) as month, avg(volume) as volume
FROM Tickers JOIN PriceBars ON (Tickers.symbol = PriceBars.ticker)
WHERE symbol IN ('META', 'AAPL', 'AMZN', 'NVDA', 'GOOG', 'TSLA', 'UBER', 'MSFT', 'ABNB', 'AVGO', 'ORCL') OR symbol IN
(
  SELECT ticker
  FROM PriceBars
  GROUP BY ticker
  HAVING AVG(close) > 135
)
GROUP BY name, MONTH(time) ORDER BY month, name;
```

```
-> Sort: `month`, Tickers.`name` (actual time=1405..1405 rows=22 loops=1) -> Table scan on <temporary> (actual time=1405..1405 rows=22 loops=1) -> Aggregate using temporary table (actual time=1405..1405 rows=22 loops=1) -> Nested loop inner join (cost=60971 rows=522621) (actual time=1399..1404 rows=660 loops=1) -> Filter: ((Tickers.symbol in ('META','AAPL','AMZN','NVDA','GOOG','TSLA','UBER','MSFT','ABNB','AVGO','ORCL')) or <in_optimizer>(Tickers.symbol,Tickers.symbol in (select #2))) (cost=916 rows=8759) (actual time=1399..1404 rows=11 loops=1) -> Table scan on Tickers (cost=916 rows=8759) (actual time=0.751..3.25 rows=8009 loops=1) -> Select #2 (subquery in condition; run only once) -> Filter: ((Tickers.symbol = `<materialized_subquery>`.ticker)) (cost=98858..98858 rows=1) (actual time=1398..1398 rows=0 loops=1) -> Limit: 1 row(s) (cost=98858..98858 rows=1) (actual time=1398..1398 rows=0 loops=1) -> Index lookup on <materialized_subquery> using <auto_distinct_key> (ticker=Tickers.symbol) (actual time=1398..1398 rows=0 loops=1) -> Materialize with deduplication (cost=98858..98858 rows=8010) (actual time=1398..1398 rows=0 loops=1) -> Filter: (avg(PriceBars.`close`) > 135) (cost=98057 rows=8010) (actual time=1398..1398 rows=0 loops=1) -> Group aggregate: avg(PriceBars.`close`) (cost=98057 rows=8010) (actual time=3.93..1395 rows=8009 loops=1) -> Index scan on PriceBars using PRIMARY (cost=50264 rows=477931) (actual time=3.9..1289 rows=480541 loops=1) -> Index lookup on PriceBars using PRIMARY (ticker=Tickers.symbol) (cost=0.89 rows=59.7) (actual time=0.0271..0.0374 rows=60 loops=11)
```

Variant C:
Cost: 60810



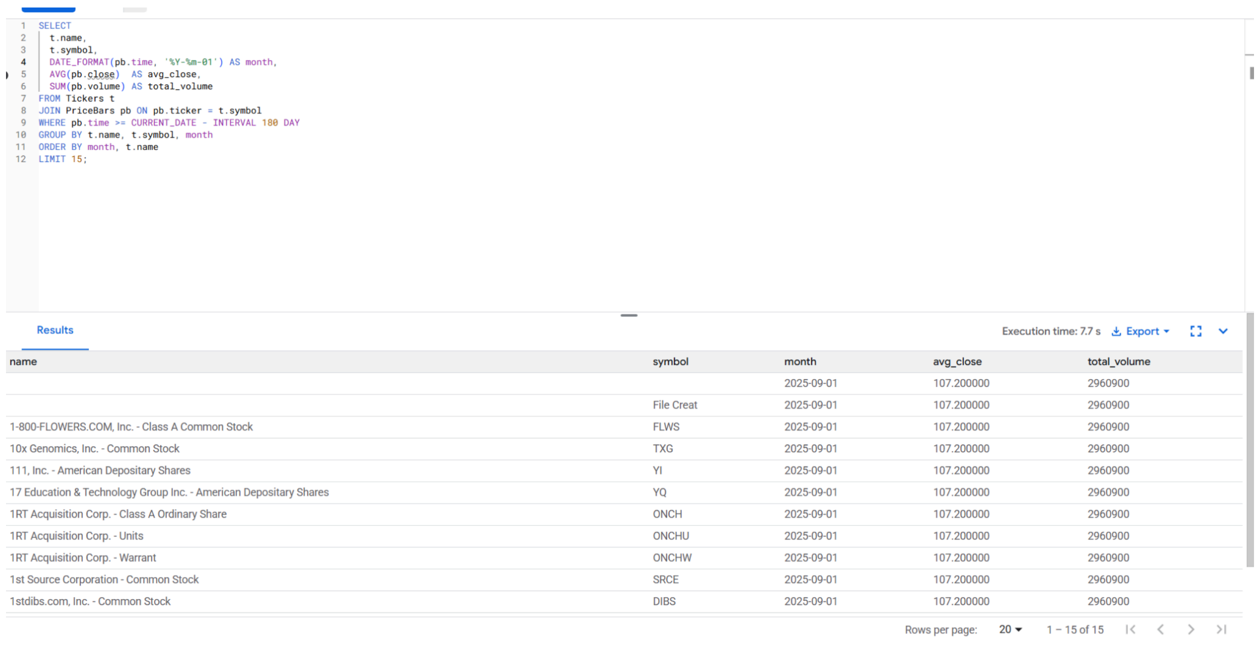
```sql

#CREATE INDEX pb_ticker_time_volume ON PriceBars(ticker, time, volume);

EXPLAIN ANALYZE
SELECT name, MONTH(time) as month, avg(volume) as volume
FROM Tickers JOIN PriceBars ON (Tickers.symbol = PriceBars.ticker)
WHERE symbol IN ('META', 'AAPL', 'AMZN', 'NVDA', 'GOOG', 'TSLA', 'UBER', 'MSFT', 'ABNB', 'AVGO', 'ORCL') OR symbol IN
(
  SELECT ticker
  FROM PriceBars
  GROUP BY ticker
  HAVING AVG(close) > 135
)
GROUP BY name, MONTH(time) ORDER BY month, name;
```

```
-> Sort: `month`, Tickers.`name` (actual time=1378..1378 rows=22 loops=1) -> Table scan on <temporary> (actual time=1377..1377 rows=22 loops=1) -> Aggregate using temporary table (actual time=1377..1377 rows=22 loops=1) -> Nested loop inner join (cost=60810 rows=522621) (actual time=1299..1376 rows=660 loops=1) -> Filter: ((Tickers.symbol in ('META','AAPL','AMZN','NVDA','GOOG','TSLA','UBER','MSFT','ABNB','AVGO','ORCL')) or <in_optimizer>(Tickers.symbol,Tickers.symbol in (select #2))) (cost=1033 rows=8759) (actual time=1299..1373 rows=11 loops=1) -> Table scan on Tickers (cost=1033 rows=8759) (actual time=1.02..71 rows=8009 loops=1) -> Select #2 (subquery in condition; run only once) -> Filter: ((Tickers.symbol = `<materialized_subquery>`.ticker)) (cost=98770..98770 rows=1) (actual time=1298..1298 rows=0 loops=1) -> Limit: 1 row(s) (cost=98770..98770 rows=1) (actual time=1298..1298 rows=0 loops=1) -> Index lookup on <materialized_subquery> using <auto_distinct_key> (ticker=Tickers.symbol) (actual time=1298..1298 rows=0 loops=1) -> Materialize with deduplication (cost=98770..98770 rows=8010) (actual time=1298..1298 rows=0 loops=1) -> Filter: (avg(PriceBars.`close`) > 135) (cost=97969 rows=8010) (actual time=1298..1298 rows=0 loops=1) -> Group aggregate: avg(PriceBars.`close`) (cost=97969 rows=8010) (actual time=3.54..1296 rows=8009 loops=1) -> Index scan on PriceBars using PRIMARY (cost=50176 rows=477931) (actual time=3.5..1189 rows=480541 loops=1) -> Index lookup on PriceBars using PRIMARY (ticker=Tickers.symbol) (cost=0.859 rows=59.7) (actual time=0.297..0.309 rows=60 loops=11)
```

Query 3:

Results: The Baseline plan cost 63,100.87. Variant A (ticker,time) decreased cost to 62,301.61 by aligning with the join on ticker and the time filter, allowing range scans over the 180-day window before grouping. Variant B (time) only dropped to 62,965.90. This was optimized over time, but extra lookups by ticker limited gains. Variant C (covering: ticker,time,close,volume) matched baseline (63,100.87), meaning that index-only reads didn't trigger or didn't reduce work on this dataset. Choice: We select Variant A—it provides the best cost reduction with modest storage/write overhead.

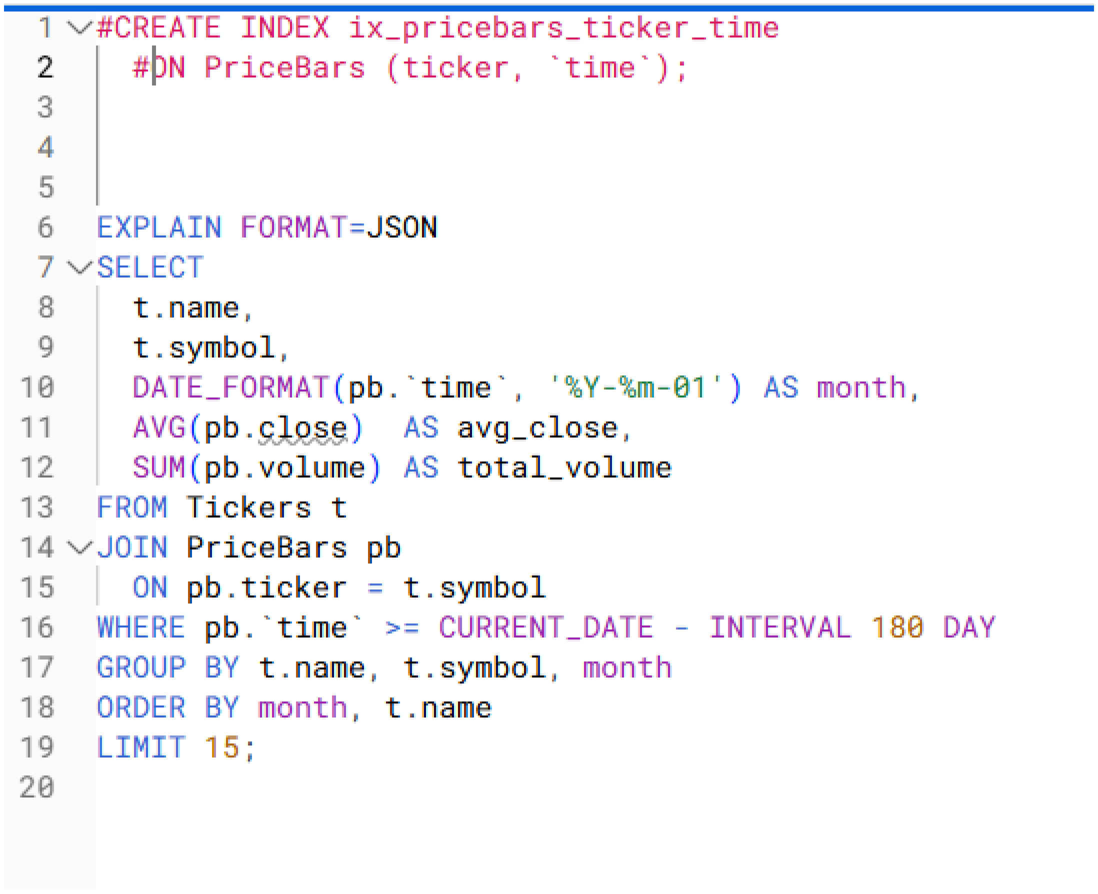
Baseline cost: 63100.87



```sql
SELECT
  t.name,
  t.symbol,
  DATE_FORMAT(pb.time, '%Y-%m-01') AS month,
  AVG(pb.close)  AS avg_close,
  SUM(pb.volume) AS total_volume
FROM Tickers t
JOIN PriceBars pb ON pb.ticker = t.symbol
WHERE pb.time >= CURRENT_DATE - INTERVAL 180 DAY
GROUP BY t.name, t.symbol, month
ORDER BY month, t.name
LIMIT 15;
```

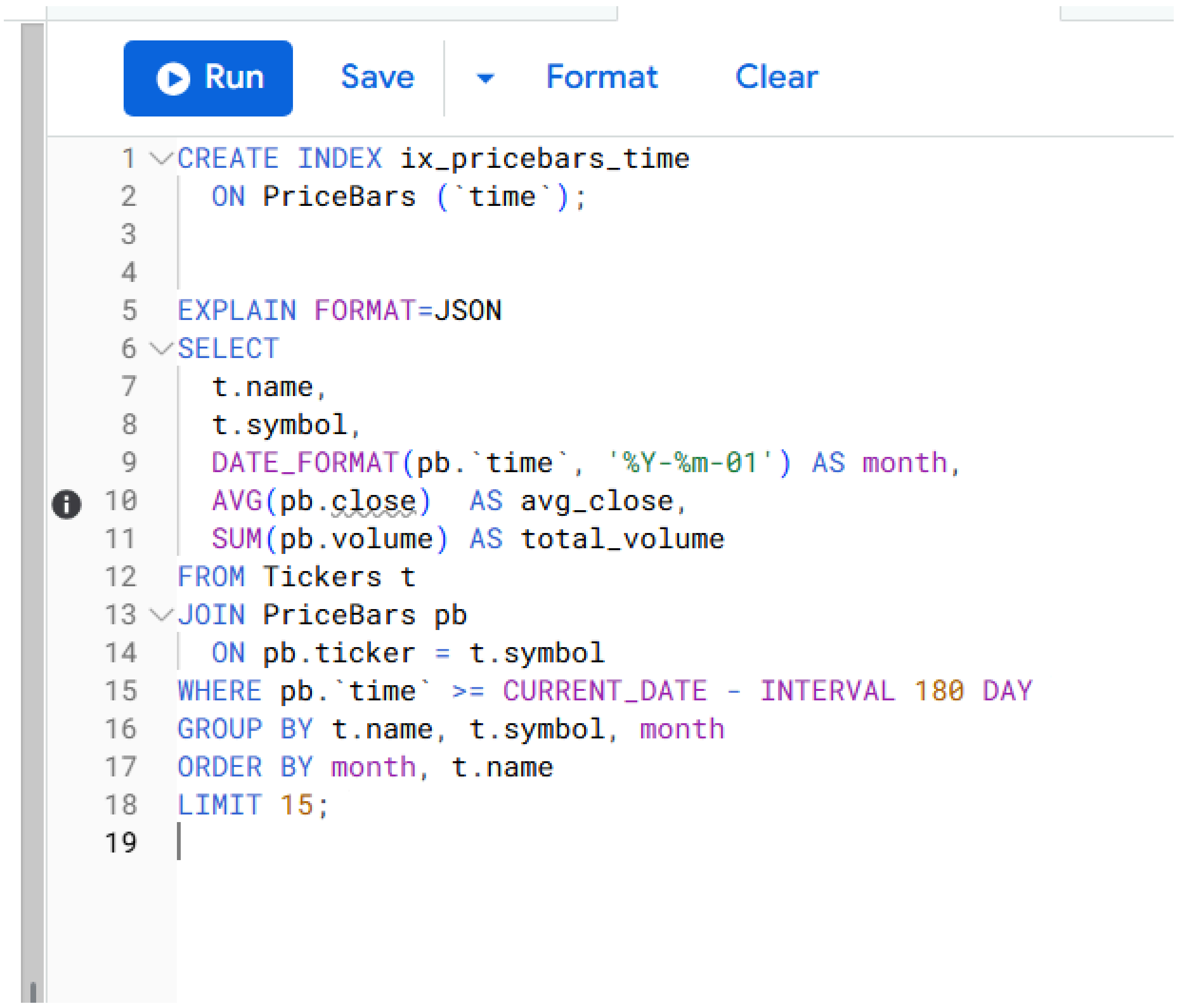
name	symbol	month	avg_close	total_volume
		2025-09-01	107.200000	2960900
	File Creat	2025-09-01	107.200000	2960900
1-800-FLOWERS.COM, Inc. - Class A Common Stock	FLWS	2025-09-01	107.200000	2960900
10x Genomics, Inc. - Common Stock	TXG	2025-09-01	107.200000	2960900
111, Inc. - American Depositary Shares	YI	2025-09-01	107.200000	2960900
17 Education & Technology Group Inc. - American Depositary Shares	YQ	2025-09-01	107.200000	2960900
1RT Acquisition Corp. - Class A Ordinary Share	ONCH	2025-09-01	107.200000	2960900
1RT Acquisition Corp. - Units	ONCHU	2025-09-01	107.200000	2960900
1RT Acquisition Corp. - Warrant	ONCHW	2025-09-01	107.200000	2960900
1st Source Corporation - Common Stock	SRCE	2025-09-01	107.200000	2960900
1stdibs.com, Inc. - Common Stock	DIBS	2025-09-01	107.200000	2960900

Variant A: cost 62301.61



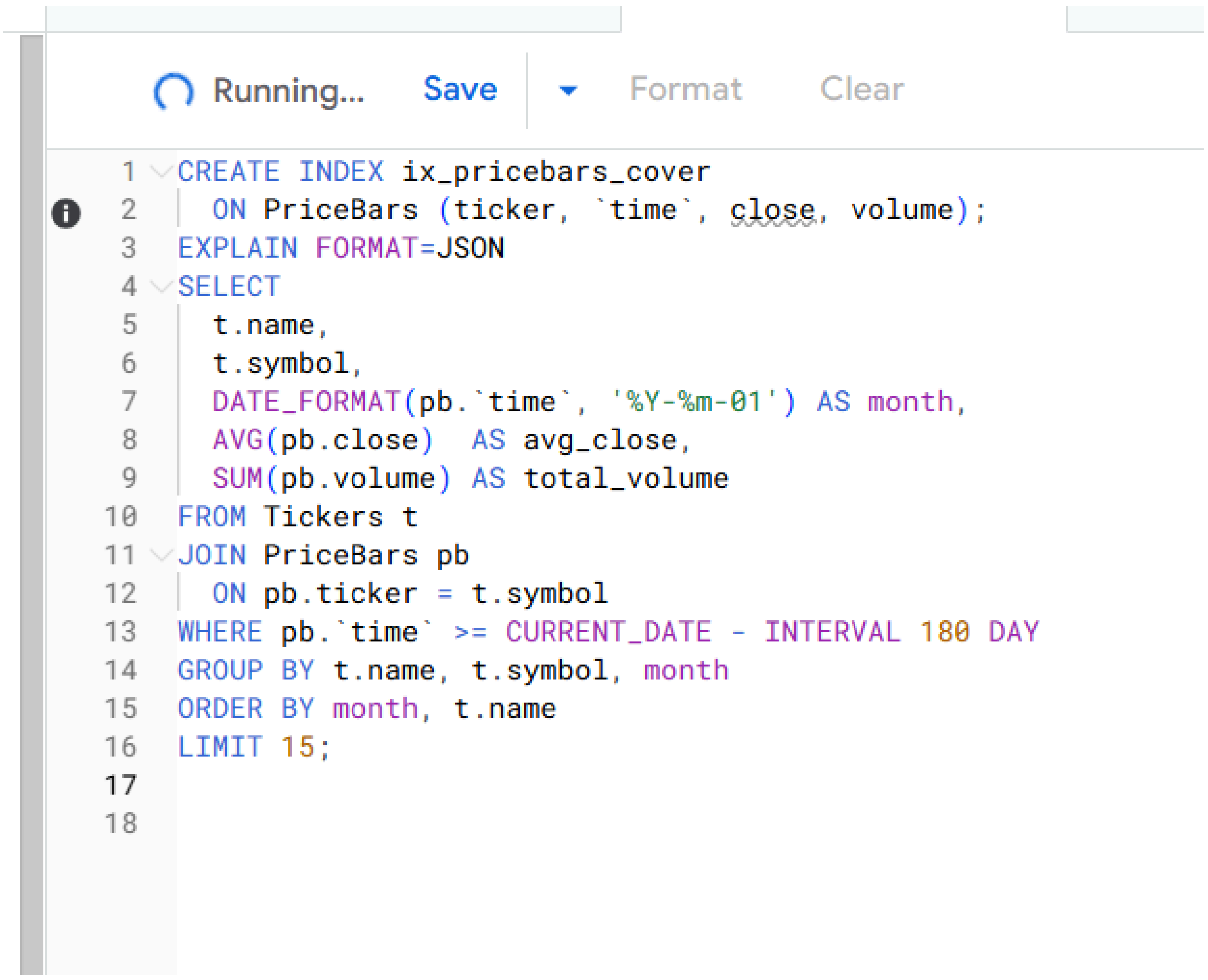
```
#CREATE INDEX ix_pricebars_ticker_time
  #ON PriceBars (ticker, `time`);



EXPLAIN FORMAT=JSON
SELECT
  t.name,
  t.symbol,
  DATE_FORMAT(pb.`time`, '%Y-%m-01') AS month,
  AVG(pb.close)  AS avg_close,
  SUM(pb.volume) AS total_volume
FROM Tickers t
JOIN PriceBars pb
  ON pb.ticker = t.symbol
WHERE pb.`time` >= CURRENT_DATE - INTERVAL 180 DAY
GROUP BY t.name, t.symbol, month
ORDER BY month, t.name
LIMIT 15;
```

Variant B: 62965.90



```sql
CREATE INDEX ix_pricebars_time
  ON PriceBars (`time`);


EXPLAIN FORMAT=JSON
SELECT
  t.name,
  t.symbol,
  DATE_FORMAT(pb.`time`, '%Y-%m-01') AS month,
  AVG(pb.close)  AS avg_close,
  SUM(pb.volume) AS total_volume
FROM Tickers t
JOIN PriceBars pb
  ON pb.ticker = t.symbol
WHERE pb.`time` >= CURRENT_DATE - INTERVAL 180 DAY
GROUP BY t.name, t.symbol, month
ORDER BY month, t.name
LIMIT 15;
```

Variant C: 63100.87



```sql
CREATE INDEX ix_pricebars_cover
  ON PriceBars (ticker, `time`, close, volume);
EXPLAIN FORMAT=JSON
SELECT
  t.name,
  t.symbol,
  DATE_FORMAT(pb.`time`, '%Y-%m-01') AS month,
  AVG(pb.close)  AS avg_close,
  SUM(pb.volume) AS total_volume
FROM Tickers t
JOIN PriceBars pb
  ON pb.ticker = t.symbol
WHERE pb.`time` >= CURRENT_DATE - INTERVAL 180 DAY
GROUP BY t.name, t.symbol, month
ORDER BY month, t.name
LIMIT 15;
```