פרויקט גמר בארגון נתונים

SQL-ו

Stoxify

תוכן עניינים

תוכן עניינים 2
מבוא 3
מהי בורסה לניירות ערך? 3
סוגי פקודות במערכת..... 3
עקרונות בסיסיים של מסחר 3
תיאור הפרויקט..... 4
מטרת המערכת 4
פונקציות המערכת..... 4
האנשים המשתמשים במסד..... 4
דיאגרמת קשרים..... 5
ישויות..... 6
טבלת חברות 6
טבלת מניות..... 6
טבלת סוחרים..... 6
טבלת הזמנות ((Orders..... 7
טבלת טרנזקציות 8
טבלת היסטוריית אחזקות..... 8
טבלת מטא-דאטה של תמונות מצב ((snapshot_metadata..... 9
תצוגה אחזקות נוכחיות 9
קשרים..... 10
פונקציות ופרוצדורות..... 11
פונקציה - format_big_number..... 11
פונקציה - get_last_price 11
פרוצדורה - TakeHoldingsSnapshot 12
פרוצדורה - IssueInitialShares..... 12
שאילתות 13
שאילתות SELECT..... 13
שאילתות סיכום וסטטיסטיקה..... 13
שאילתות JOINS 14
שאילתות GROUP BY ו- HAVING 16
שאילתות INSERT / UPDATE / DELETE..... 16
שאילתות עם Subqueries..... 17
שאילתות UNION 17
שאילתות מסובכות..... 18
סכמת מסד הנתונים..... 20

מבוא

מהי בורסה לניירות ערך?

בורסה לניירות ערך היא מקום שבו אנשים קונים ומוכרים מניות של חברות. כל מניה מייצגת חלק מהבעלות על חברה, והמחירים משתנים לפי ביקוש והיצע – כלומר, כמה אנשים רוצים לקנות וכמה רוצים למכור.
הבורסה דומה לשוק פתוח – כל אחד יכול להציע מחיר ולקוות שמישהו אחר יסכים לבצע עסקה לפי התנאים האלה.

סוגי פקודות במערכת

במערכת יש שני סוגי פקודות:
פקודת קנייה (Buy) – הסוחר מבקש לקנות מניה מסוימת בכמות מסוימת ובמחיר מקסימלי שהוא מוכן לשלם.
פקודת מכירה (Sell) – הסוחר מציע למכור מניה בכמות מסוימת ובמחיר מינימלי שהוא מוכן לקבל.
אם קיימת התאמה בין קונה למוכר – כלומר, אם המחיר של הקונה מספיק גבוה בשביל אחד המוכרים – מתבצעת עסקה.

עקרונות בסיסיים של מסחר

התאמה בין פקודות – כל פקודת קנייה תתבצע רק אם קיימת פקודת מכירה שתואמת לה במחיר ובכמות.
מחיר קודם, אחר כך זמן – כשיש כמה פקודות מתאימות, העדיפות היא לפי המחיר (למשל, מוכר שדורש פחות מקבל עדיפות), ואם יש כמה באותו מחיר – לפי מי שהקדים.
אין מכירה בלי מניות – סוחר יכול למכור רק מניות שבאמת יש לו.
אין קנייה בלי כסף – סוחר יכול לקנות רק אם יש לו מספיק יתרה לביצוע העסקה.

תיאור הפרויקט

מטרת המערכת

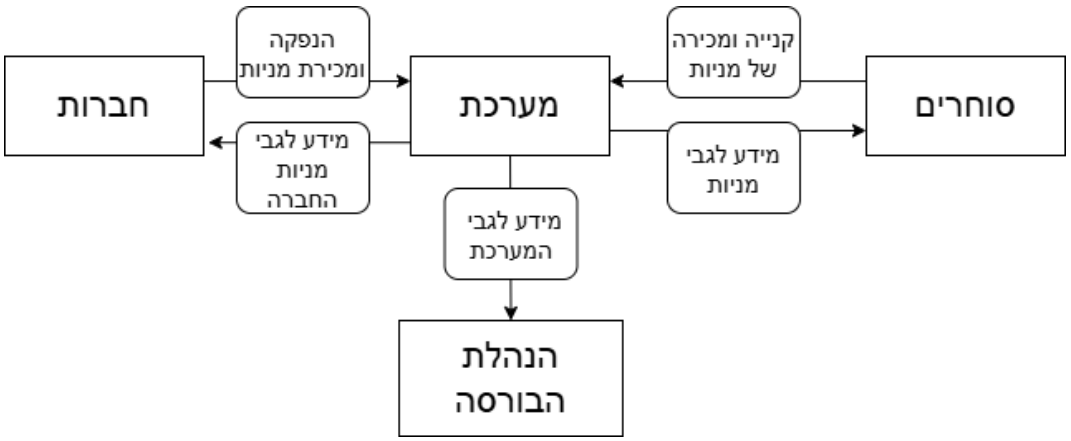
המסד נתונים שלי עוסק במסחר בין סוחרים לבין חברות ציבוריות שמנפיקות מניות.
המערכת נועדה לאפשר לכל סוחר לבצע קנייה או מכירה של מניות, לנהל את האחזקות שלו, ולעקוב אחרי עסקאות שבוצעו. המטרה היא לבנות מערכת שתדמה פעילות בורסאית אמיתית, כולל התנהלות שוק, הנפקות, פקודות, עסקאות וניתוח מידע.

פונקציות המערכת

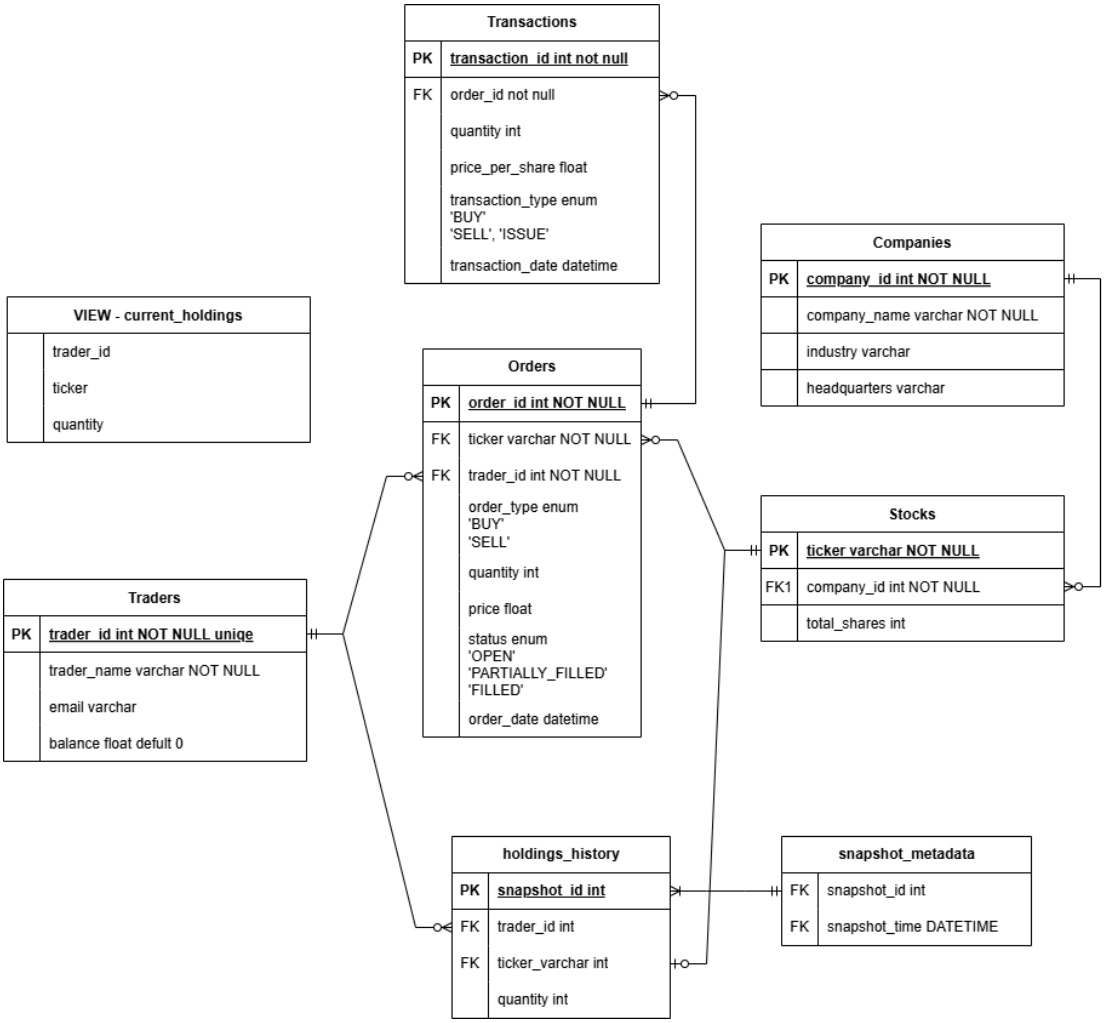
- **הוספת חברות ומניות**
- **הוספת סוחרים (Traders)**
- **פקודות קנייה ומכירה**
- **מנגנון התאמת פקודות קנייה ומכירה**
- **תיעוד עסקאות (Transactions)**
- **שאילתות להצגת סטטיסטיקות**

האנשים המשתמשים במסד

במערכת יש שלושה סוגים של משתמשים:
סוחרים (Traders) – משתמשים פרטיים שמבצעים פקודות קנייה ומכירה, משתדלים להרוויח ממסחר במניות.
חברות (Companies) – החברות הציבוריות שהנפיקו מניות. בתחילת הדרך כל המניות שלהן מוחזקות על ידי סוחר מיוחד שנקרא `SystemIssuer`, ומכאן הן עוברות לסוחרים אמיתיים דרך פקודות מכירה.
הנהלת הבורסה – גוף מנהלי שאינו מבצע פקודות בעצמו, אלא אחראי לניטור פעילות המסחר והפקת תובנות וסטטיסטיקות.



דיאגרמת קשרים



ישויות

טבלת חברות

טבלה זו מכילה מידע בסיסי על חברות שמניותיהן נסחרות במערכת.

שם השדה	סוג הנתון	מפתח ראשי	מפתח זר	not null	הסבר
מזהה חברה	מספר	✅		✅	מספר ייחודי שמייצג כל חברה
שם חברה	טקסט			✅	שם החברה
תעשייה	טקסט				באיזה תחום החברה עוסקת
מטה ראשי	טקסט				מיקום המשרדים הראשיים של החברה

טבלת מניות

טבלה זו מרכזת מידע על ניירות ערך (המניות) הספציפיים שניתן לסחור בהם

שם השדה	סוג הנתון	מפתח ראשי	מפתח זר	not null	הסבר
טיקר	טקסט	✅		✅	הסימול הייחודי של המניה בבורסה
מזהה חברה	מספר		✅	✅	מצביע על החברה שהנפיקה את המניה
סך כל המניות	מספר			✅	מספר המניות הכולל שהונפקו וקיימות בשוק עבור סימול זה.

טבלת סוחרים

טבלה זו מאחסנת מידע על הסוחרים הרשומים במערכת המסחר. כל רשומה מייצגת סוחר בודד

שם השדה	סוג הנתון	מפתח ראשי	מפתח זר	not null	הסבר
מזהה סוחר	מספר	✅		✅	מספר ייחודי שמייצג כל סוחר
שם סוחר	טקסט			✅	שם הסוחר
אימייל	טקסט				אימייל של הסוחר
יתרה	מספר עשרוני				יתרת חשבון הסוחר (ברירת מחדל 0)

טבלת הזמנות (Orders)

טבלה זו מתעדת את הוראות הקנייה או המכירה שסוחרים הזינו למערכת. כל רשומה מייצגת הזמנה אחת, שעשויה להיות במצב פתוח, מבוצע חלקית או מבוצע במלואו.

שם השדה	סוג הנתון	מפתח ראשי	מפתח זר	not null	הסבר
מזהה הזמנה	מספר	✅		✅	מספר ייחודי שמייצג כל הזמנה
טיקר	טקסט		✅	✅	מצביע על מניה עבורה ניתנה ההזמנה
מזהה סוחר	מספר		✅	✅	מצביע על הסוחר אשר ביצע את ההזמנה
סוג הזמנה	ENUM ‘buy’,’sell’			✅	מציין אם זו הזמנת קנייה או מכירה
כמות	מספר			✅	מספר המניות המבוקש בהזמנה
מחיר	מספר עשרוני			✅	המחיר המבוקש קניית LIMIT - במקרה של מכירה מייצג את המחיר המינימלי, במקרה של קנייה מייצג את המחיר המקסימלי
סטטוס	ENUM ‘OPEN’, ‘PARTIALLY_FILLED’, ‘FILLED’, 'ISSUED’			✅	מצב ההזמנה הנוכחי
תאריך הזמנה	תאריך			✅	התאריך והשעה בהם נוצרה ההזמנה

טבלת טרנזקציות

טבלה זו מתעדת את הביצוע בפועל של עסקאות קנייה ומכירה. כל רשומה מייצגת עסקה שהתרחשה כתוצאה מהתאמה בין הזמנות קנייה ומכירה (או הנפקה).

שם השדה	סוג הנתון	מפתח ראשי	מפתח זר	not null	הסבר
מזהה טרנזקציה	מספר	✅		✅	מספר ייחודי שמייצג כל טרנזקציה
מזהה הזמנה	מספר		✅	✅	מצביע על ההזמנה שעבורה בוצעה הטרנזקציה
כמות	מספר			✅	מספר המניות שהוחלפו בטרנזקציה הספציפית הזו.
מחיר למניה	מספר עשרוני			✅	המחיר בו בוצעה העסקה עבור כל מניה.
סוג טרנזקציה	ENUM ‘Buy’ ‘SELL’ ‘ISSUE’			✅	מציין את אופי הטרנזקציה (קניה/מכירה בין סוחרים, או הנפקה ראשונית/נוספת)
תאריך טרנזקציה	תאריך			✅	התאריך והשעה המדויקים בהם בוצעה הטרנזקציה.

טבלת היסטוריית אחזקות

טבלה זו מתעדת את כמות המניות מכל סוג שהוחזקה על ידי כל סוחר בנקודות זמן ספציפיות

שם השדה	סוג הנתון	מפתח ראשי	מפתח זר	not null	הסבר
מזהה תמונת מצב	מספר		✅	✅	מצביע על תמונת המצב הרלוונטית בטבלת snapshot_metadata
מזהה סוחר	מספר		✅	✅	מצביע על הסוחר שהחזיק במניות
טיקר	טקסט		✅	✅	מצביע על המניה שהוחזקה
כמות	מספר			✅	מספר המניות שהוחזקו על ידי הסוחר הנתון בזמן תמונת המצב הנתונה

טבלת מטא-דאטה של תמונות מצב (snapshot_metadata)

טבלה זו מכילה מידע על נקודות הזמן שבהן נלקחו תמונות מצב של אחזקות המניות של כל הסוחרים לצורך שימור היסטוריה של נקודה מסוימת.

שם השדה	סוג הנתון	מפתח ראשי	מפתח זר	not null	הסבר
מזהה תמונת מצב	מספר	✅		✅	מספר ייחודי שמייצג כל תמונת מצב
תאריך תמונת מצב	תאריך			✅	תאריך שבו נלקחה השמירה

תצוגה אחזקות נוכחיות

זו אינה טבלה פיזית אלא תצוגה וירטואלית. היא מציגה סיכום של כמות המניות הנוכחית שכל סוחר מחזיק בכל מניה. נתונים אלו מחושבים בזמן אמת מהטבלאות האחרות (כמו טרנזקציות או היסטוריית אחזקות)

שם השדה	סוג הנתון	מפתח ראשי	מפתח זר	not null	הסבר
מזהה סוחר	מספר		✅	✅	מצביע על הסוחר שמחזיק במניות
טיקר	טקסט		✅	✅	מצביע על המניה שמוחזקת
כמות	מספר			✅	הכמות הנוכחית המוחזקת על ידי הסוחר במניה זו

קשרים

סוחרים ↔ הזמנות

- כל סוחר יכול לבצע מספר הזמנות (n:1).
- כל הזמנה מבוצעת על ידי סוחר אחד בלבד.

סוחרים ↔ היסטוריית אחזקות

- כל סוחר יכול להופיע במספר רשומות בהיסטוריית האחזקות (בזמנים שונים או עבור מניות שונות) (n:1).
- כל רשומת היסטוריית אחזקות שייכת לסוחר אחד בלבד (עבור תמונת מצב ומניה ספציפיות).

חברות ↔ מניות

- כל חברה יכולה להנפיק מניה אחת או יותר (n:1).
- כל מניה שייכת לחברה אחת בלבד.

מניות ↔ הזמנות

- כל מניה יכולה להופיע במספר הזמנות (n:1).
- כל הזמנה מתייחסת למניה אחת בלבד.

מניות ↔ היסטוריית אחזקות

- כל מניה יכולה להופיע במספר רשומות בהיסטוריית האחזקות (עבור סוחרים שונים או תמונות מצב שונות) (n:1).
- כל רשומת היסטוריית אחזקות מתייחסת למניה אחת בלבד (עבור סוחר ותמונת מצב ספציפיים).

הזמנות ↔ טרנזקציות

- כל הזמנה יכולה להתבצע באמצעות טרנזקציה אחת או יותר (למשל, מילוי חלקי) (n:1).
- כל טרנזקציה שייכת להזמנה אחת בלבד.

מטא-דאטה של תמונות מצב ↔ היסטוריית אחזקות

- כל תמונת מצב (רשומת מטא-דאטה) יכולה להכיל רשומות אחזקה רבות (אחת לכל שילוב סוחר-מניה באותו זמן) (n:1).
- כל רשומת היסטוריית אחזקות שייכת לתמונת מצב אחת בלבד.

פונקציות ופרוצדורות

פונקציה - format_big_number

הפונקציה מקבלת מספר עשרוני ומחזירה מחרוזת עם סיומת המתאימה לגודלו (K לאלפים, M למיליונים, B למיליארדים).

דוגמא - הפונקציה מקבלת 1,900,000 ותחזיר 1.9M

```
DELIMITER //
CREATE FUNCTION format_big_number(val DOUBLE) RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
  RETURN CASE
    WHEN val >= 1000000000 THEN CONCAT(ROUND(val / 1000000000, 2), 'B')
    WHEN val >= 1000000 THEN CONCAT(ROUND(val / 1000000, 2), 'M')
    WHEN val >= 1000 THEN CONCAT(ROUND(val / 1000, 2), 'K')
    ELSE ROUND(val, 2)
  END;
END;
//
DELIMITER ;
```

פונקציה - get_last_price

הפונקציה מקבלת טיקר של מניה ומחזירה את מחיר העסקה האחרון שבוצע על אותה מניה. פונקציה זו מאפשרת שליפה מהירה של מחיר השוק העדכני ביותר עבור מניות ספציפיות, ומשמשת לצורכי ניתוחים פיננסיים, הצגת נתונים עדכניים למשתמשים או לצורך חישובים נוספים במערכת.

```
DELIMITER //

CREATE FUNCTION get_last_price(p_ticker VARCHAR(10)) RETURNS FLOAT
DETERMINISTIC
BEGIN
  DECLARE v_price FLOAT;

  SELECT t.price_per_share
  INTO v_price
  FROM Transactions t
  JOIN Orders o ON t.order_id = o.order_id
  WHERE o.ticker = p_ticker
  ORDER BY t.transaction_date DESC, t.transaction_id DESC
  LIMIT 1;

  RETURN v_price;  -- If no result, v_price remains NULL
END;
//
DELIMITER ;
```

פרוצדורה - TakeHoldingsSnapshot

תיאור:
פרוצדורה שיוצרת צילום מצב (snapshot) של כל האחזקות של המשתמשים במערכת.
מה היא עושה בפועל:
1. יוצרת רשומת מטא-דאטה חדשה בטבלת snapshot_metadata (עם מזהה ייחודי לצילום המצב).
2. מעתיקה את כל האחזקות הנוכחיות (מ-current_holdings) לטבלת ההיסטוריה holdings_history תוך קישור למזהה צילום המצב.

מטרה:
לשמור גרסאות היסטוריות של האחזקות בכל רגע נתון – לצורכי ניתוח, השוואה בין תאריכים, ובדיקות רטרוספקטיביות.

```
DELIMITER //
CREATE PROCEDURE TakeHoldingsSnapshot()
BEGIN
  DECLARE v_snapshot_id INT;

  -- Step 1: Insert new snapshot metadata
  INSERT INTO snapshot_metadata () VALUES ();
  SET v_snapshot_id = LAST_INSERT_ID();

  -- Step 2: Insert current holdings into history
  INSERT INTO holdings_history (snapshot_id, trader_id, ticker, quantity)
  SELECT v_snapshot_id, trader_id, ticker, quantity
  FROM current_holdings;
END;
//
DELIMITER ;
```

פרוצדורה - IssueInitialShares

הפרוצדורה משמשת להנפקת מניות ראשונית עבור מניה חדשה. היא יוצרת פקודת קנייה מלאכותית שמיוחסת ל-SystemIssuer (סוחר פנימי של המערכת) ומייצרת עסקה פיקטיבית, כדי לדמות שהמניות הונפקו ונמצאות בבעלות ראשונית לפני מכירה לציבור.

```
DELIMITER //
CREATE PROCEDURE IssueInitialShares (
  IN p_ticker VARCHAR(10),
  IN p_quantity INT )
BEGIN
  DECLARE v_order_id INT;
  -- Insert a fake "BUY" order to grant SystemIssuer ownership
  INSERT INTO orders (
    ticker, trader_id, order_type, quantity, price, status, order_date
  ) VALUES (
    p_ticker, 1, 'BUY', p_quantity, 0.00, 'ISSUED', CURRENT_TIMESTAMP
  );
  SET v_order_id = LAST_INSERT_ID();
  -- Insert a fake transaction to reflect the issued shares
  INSERT INTO transactions (
    order_id, quantity, price_per_share, transaction_type, transaction_date
  ) VALUES (
    v_order_id, p_quantity, 0.00, 'BUY', CURRENT_TIMESTAMP
  );
END;
//
DELIMITER ;
```

שאילתות

שאילתות SELECT

1.מציג את כל החברות בתחום הביוטכנולוגיה.

```
SELECT * FROM companies WHERE industry = 'Biotech';
```

2.מציג את כל המניות שהונפקו בהיקף של מעל 900,000 מניות.

```
SELECT * FROM stocks WHERE total_shares > 900000;
```

3.מציג את כל הסוחרים עם יתרה של מעל 100,000.

```
SELECT * FROM traders WHERE balance > 100000;
```

4.מציג את כל זמני צילומי המצב לפי סדר יורד.

```
SELECT snapshot_id, snapshot_time
FROM snapshot_metadata
ORDER BY snapshot_time DESC;
```

שאילתות סיכום וסטטיסטיקה

1.סופר כמה חברות יש בתחומים AI או סייבר.

```
SELECT COUNT(*) AS num_companies
FROM companies
WHERE industry IN ('AI', 'Cybersecurity');
```

2.מחשב את ממוצע המניות לחברות ביוטכנולוגיה.

```
SELECT AVG(s.total_shares) AS avg_biotech_shares
FROM stocks s
JOIN companies c ON s.company_id = c.company_id
WHERE c.industry = 'Biotech';
```

3.מציג את היתרה הגבוהה ביותר מבין הסוחרים.

```
SELECT MAX(balance) AS richest_balance FROM traders;
```

4.מציג את סך כל המניות של חברת SolarNova Energy.

```
SELECT SUM(s.total_shares) AS total_sne_shares
FROM stocks s
JOIN companies c ON s.company_id = c.company_id
WHERE c.company_name = 'SolarNova Energy';
```

שאילתות JOINS

1.מציג כל מניה יחד עם שם החברה שהיא שייכת אליה.

```
SELECT s.ticker, c.company_name
FROM stocks s
JOIN companies c ON s.company_id = c.company_id;
```

2.מציג כל סוחר יחד עם הפקודות שהכניס (אם יש).

```
SELECT t.trader_name, o.order_id, o.ticker
FROM traders t
LEFT JOIN orders o ON t.trader_id = o.trader_id;
```

3.מציג מניות עם יותר מ־800,000 מניות ושם החברה שלהן.

```
SELECT s.ticker, c.company_name,
format_big_number(s.total_shares) AS total_shares
FROM stocks s
JOIN companies c ON s.company_id = c.company_id
WHERE s.total_shares > 800000;
```

4.מציג את אחזקות הסוחר בכל מניה בצילום מצב מסוים.

```
SELECT t.trader_name, h.ticker, h.quantity
FROM holdings_history h
JOIN traders t ON h.trader_id = t.trader_id
WHERE snapshot_id = 1 AND t.trader_name = 'Alice';  -- replace with
desired par
```

5.מציג אילו מניות היו הכי מוחזקות בצילום מצב נתון.

```
SELECT h.ticker, format_big_number(SUM(h.quantity)) AS
total_held
FROM holdings_history h
WHERE h.snapshot_id = 1  -- replace with desired snapshot_id
GROUP BY h.ticker
ORDER BY total_held asc;
```

6.מאתר סוחרים שלא שינו את האחזקות שלהם בין שני צילומי מצב.

```
SELECT DISTINCT t.trader_name
FROM holdings_history h1
JOIN holdings_history h2
  ON h1.trader_id = h2.trader_id AND h1.ticker = h2.ticker
JOIN traders t ON t.trader_id = h1.trader_id
WHERE h1.snapshot_id = 1 AND h2.snapshot_id = 2
  AND h1.quantity = h2.quantity;
```

שאילתות GROUP BY ו- HAVING

1.מחלק את הסוחרים לפי רמות יתרה (גבוהה, בינונית, נמוכה) וסופר כמה יש בכל רמה.

```
SELECT
  CASE
    WHEN balance >= 100000 THEN 'High'
    WHEN balance >= 50000 THEN 'Medium'
    ELSE 'Low'
  END AS balance_level,
  COUNT(*) AS count
FROM traders
GROUP BY balance_level;
```

2.סופר את מספר החברות בכל תעשייה, רק אם יש יותר מחברה אחת.

```
SELECT industry, COUNT(*) AS company_count
FROM companies
GROUP BY industry
HAVING COUNT(*) > 1;
```

3.מחשב ממוצע מניות לכל תחום תעשייה.

```
SELECT c.industry, format_big_number(AVG(s.total_shares)) AS
avg_shares
FROM companies c
JOIN stocks s ON c.company_id = s.company_id
GROUP BY c.industry;
```

שאילתות INSERT / UPDATE / DELETE

1.מוסיף סוחר חדש בשם Karen עם כתובת אימייל ויתרה.

```
INSERT INTO traders (trader_name, email, balance)
VALUES ('Karen', 'karen@example.com', 90000);
```

2.מעדכן את היתרה של הסוחר Eli ל־75,000.

```
UPDATE traders
SET balance = 75000
WHERE trader_name = 'Eli';
```

3.מוחק את הסוחר Ian ממסד הנתונים.

```
DELETE FROM traders
WHERE trader_name = 'Ian';
```

שאילתות עם Subqueries

1.מציג חברות שהנפיקו יותר מניות מהממוצע הכללי.

```
SELECT c.company_name
FROM companies c
JOIN stocks s ON c.company_id = s.company_id
WHERE s.total_shares > (
  SELECT AVG(total_shares) FROM stocks
);
```

2.מציג סוחרים שיש להם את אותה יתרה כמו Judy.

```
SELECT trader_name
FROM traders
WHERE balance = (
  SELECT balance FROM traders WHERE trader_name = 'Judy'
);
```

שאילתות UNION

מאחדת את כל פקודות הקנייה והמכירה הפעילות לרשימה אחת, חוץ מ־SystemIssuer.

```
SELECT trader_id, ticker, 'BUY' AS type, quantity, price
FROM orders
WHERE order_type = 'BUY' AND status IN ('OPEN',
'PARTIALLY_FILLED') AND trader_id != 1
UNION
SELECT trader_id, ticker, 'SELL' AS type, quantity, price
FROM orders
WHERE order_type = 'SELL' AND status IN ('OPEN',
'PARTIALLY_FILLED') AND trader_id != 1
ORDER BY trader_id;
```

שאילתות מסובכות

1.מציג את המניה עם סך כמות המסחר הגבוהה ביותר.

```
SELECT
    o.ticker,
    format_big_number(SUM(t.quantity)) AS total_volume
FROM transactions t
JOIN orders o ON t.order_id = o.order_id
WHERE o.trader_id != 1
GROUP BY o.ticker
ORDER BY SUM(t.quantity) DESC
LIMIT 1;
```

2.מחשב מי הסוחר הרווחי ביותר על פי סכום מכירות פחות סכום קניות.

```
SELECT
    o.trader_id,
    t1.trader_name,
    format_big_number(SUM(CASE WHEN t.transaction_type =
'SELL' THEN t.price_per_share * t.quantity ELSE 0 END) -
                        SUM(CASE WHEN t.transaction_type =
'BUY' THEN t.price_per_share * t.quantity ELSE 0 END)) AS
net_profit
FROM transactions t
JOIN orders o ON t.order_id = o.order_id
JOIN traders t1 ON o.trader_id = t1.trader_id
WHERE o.trader_id != 1
GROUP BY o.trader_id
ORDER BY net_profit DESC
LIMIT 1;
```

3.מציג מניות שמעולם לא בוצעה עליהן עסקה.

```
SELECT s.ticker, c.company_name
FROM stocks s
LEFT JOIN (
    SELECT DISTINCT ticker FROM orders WHERE trader_id != 1
) o ON s.ticker = o.ticker
LEFT JOIN companies c ON s.company_id = c.company_id
WHERE o.ticker IS NULL;
```

4.מציג סוחרים שמעולם לא ביצעו פקודת קנייה או מכירה.

```
SELECT t.trader_id, t.trader_name
FROM traders t
LEFT JOIN orders o ON t.trader_id = o.trader_id AND o.trader_id !=
1
WHERE o.order_id IS NULL AND t.trader_id != 1;
```

5.מציג סוחרים שביצעו שלוש פקודות ומעלה.

```
SELECT t.trader_id, t.trader_name, COUNT(o.order_id) AS num_orders
FROM traders t
JOIN orders o ON t.trader_id = o.trader_id
WHERE t.trader_id != 1
GROUP BY t.trader_id
HAVING COUNT(o.order_id) >= 3;
```

6.מדמה הפקדת כסף לחשבון של סוחר (לדוגמה: אליס).

```
UPDATE traders
SET balance = balance + 5000
WHERE trader_name = 'Alice';
```

7.מדמה משיכת כסף מחשבון של סוחר (לדוגמה: בוב).

```
UPDATE traders
SET balance = balance - 3000
WHERE trader_name = 'Bob';
```

8.איחוד של כל פקודות הקנייה והמכירה הפעילות

```
SELECT order_id, trader_id, ticker, 'BUY' AS order_type, quantity,
price, status
FROM orders
WHERE status IN ('OPEN', 'PARTIALLY_FILLED') AND order_type =
'BUY' AND trader_id != 1
UNION
SELECT order_id, trader_id, ticker, 'SELL' AS order_type,
quantity, price, status
FROM orders
WHERE status IN ('OPEN', 'PARTIALLY_FILLED') AND order_type =
'SELL' AND trader_id != 1
ORDER BY ticker;
```

סכמת מסד הנתונים

```
MySQL Script generated by MySQL Workbench
-- Fri Apr 25 13:06:21 2025
-- Model: New Model    Version: 1.0
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERR
OR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8mb3 ;
USE `mydb` ;

-- -----------------------------------------------------
-- Table `mydb`.`companies`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`companies` (
  `company_id` INT NOT NULL AUTO_INCREMENT,
  `company_name` VARCHAR(45) NOT NULL,
  `industry` VARCHAR(45) NULL DEFAULT NULL,
  `headquarters` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`company_id`),
  UNIQUE INDEX `company_name_UNIQUE` (`company_name` ASC) VISIBLE)
ENGINE = InnoDB
AUTO_INCREMENT = 16
DEFAULT CHARACTER SET = utf8mb3;


-- -----------------------------------------------------
-- Table `mydb`.`snapshot_metadata`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`snapshot_metadata` (
  `snapshot_id` INT NOT NULL AUTO_INCREMENT,
  `snapshot_time` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`snapshot_id`))
ENGINE = InnoDB
AUTO_INCREMENT = 2
DEFAULT CHARACTER SET = utf8mb3;


-- -----------------------------------------------------
-- Table `mydb`.`traders`
-- -----------------------------------------------------
```

```sql
CREATE TABLE IF NOT EXISTS `mydb`.`traders` (
  `trader_id` INT NOT NULL AUTO_INCREMENT,
  `trader_name` VARCHAR(45) NOT NULL,
  `email` VARCHAR(45) NULL DEFAULT NULL,
  `balance` FLOAT NOT NULL DEFAULT '0',
  PRIMARY KEY (`trader_id`),
  UNIQUE INDEX `trader_name_UNIQUE` (`trader_name` ASC) VISIBLE)
ENGINE = InnoDB
AUTO_INCREMENT = 12
DEFAULT CHARACTER SET = utf8mb3;


-- -----------------------------------------------------
-- Table `mydb`.`stocks`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`stocks` (
  `ticker` VARCHAR(10) NOT NULL,
  `company_id` INT NOT NULL,
  `total_shares` INT NOT NULL,
  PRIMARY KEY (`ticker`),
  UNIQUE INDEX `ticker_UNIQUE` (`ticker` ASC) VISIBLE,
  INDEX `company_id_idx` (`company_id` ASC) VISIBLE,
  CONSTRAINT `company_id`
    FOREIGN KEY (`company_id`)
    REFERENCES `mydb`.`companies` (`company_id`)
    ON DELETE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb3;


-- -----------------------------------------------------
-- Table `mydb`.`holdings_history`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`holdings_history` (
  `snapshot_id` INT NOT NULL,
  `trader_id` INT NOT NULL,
  `ticker` VARCHAR(10) NOT NULL,
  `quantity` INT NOT NULL,
  INDEX `snapshot_id` (`snapshot_id` ASC) VISIBLE,
  INDEX `trader_id` (`trader_id` ASC) VISIBLE,
  INDEX `ticker` (`ticker` ASC) VISIBLE,
  CONSTRAINT `holdings_history_ibfk_1`
    FOREIGN KEY (`snapshot_id`)
    REFERENCES `mydb`.`snapshot_metadata` (`snapshot_id`)
    ON DELETE RESTRICT,
  CONSTRAINT `holdings_history_ibfk_2`
    FOREIGN KEY (`trader_id`)
    REFERENCES `mydb`.`traders` (`trader_id`)
    ON DELETE RESTRICT,
  CONSTRAINT `holdings_history_ibfk_3`
    FOREIGN KEY (`ticker`)
    REFERENCES `mydb`.`stocks` (`ticker`)
    ON DELETE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb3;
```

```sql
-- -----------------------------------------------------
-- Table `mydb`.`orders`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`orders` (
  `order_id` INT NOT NULL AUTO_INCREMENT,
  `ticker` VARCHAR(10) NOT NULL,
  `trader_id` INT NOT NULL,
  `order_type` ENUM('BUY', 'SELL') NOT NULL,
  `quantity` INT NOT NULL,
  `price` FLOAT NOT NULL,
  `status` ENUM('OPEN', 'PARTIALLY_FILLED', 'FILLED', 'ISSUED') NOT NULL,
  `order_date` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`order_id`),
  INDEX `trader_id_idx` (`trader_id` ASC) VISIBLE,
  INDEX `fk_orders_ticker` (`ticker` ASC) VISIBLE,
  CONSTRAINT `fk_orders_ticker`
    FOREIGN KEY (`ticker`)
    REFERENCES `mydb`.`stocks` (`ticker`),
  CONSTRAINT `trader_id`
    FOREIGN KEY (`trader_id`)
    REFERENCES `mydb`.`traders` (`trader_id`)
    ON DELETE RESTRICT)
ENGINE = InnoDB
AUTO_INCREMENT = 301
DEFAULT CHARACTER SET = utf8mb3;


-- -----------------------------------------------------
-- Table `mydb`.`transactions`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`transactions` (
  `transaction_id` INT NOT NULL AUTO_INCREMENT,
  `order_id` INT NOT NULL,
  `quantity` INT NOT NULL,
  `price_per_share` FLOAT NOT NULL,
  `transaction_type` ENUM('BUY', 'SELL', 'ISSUE') NOT NULL,
  `transaction_date` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`transaction_id`),
  INDEX `order_id_idx` (`order_id` ASC) VISIBLE,
  CONSTRAINT `order_id`
    FOREIGN KEY (`order_id`)
    REFERENCES `mydb`.`orders` (`order_id`)
    ON DELETE RESTRICT)
ENGINE = InnoDB
AUTO_INCREMENT = 376
DEFAULT CHARACTER SET = utf8mb3;

USE `mydb` ;

-- -----------------------------------------------------
-- Placeholder table for view `mydb`.`current_holdings`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`current_holdings` (`trader_id` INT, `ticker`
INT, `quantity` INT);
```

```
-- -------------------------------------------------------
-- procedure BuyStock
-- -------------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `BuyStock`(
  IN p_trader_id INT,
  IN p_ticker VARCHAR(10),
  IN p_quantity INT,
  IN p_max_price FLOAT
)
BEGIN
  DECLARE v_buy_order_id INT;
  DECLARE v_remaining_to_buy INT;
  DECLARE v_seller_order_id INT;
  DECLARE v_seller_id INT;
  DECLARE v_sell_price FLOAT;
  DECLARE v_seller_filled INT;
  DECLARE v_seller_order_quantity INT;
  DECLARE v_to_transfer INT;

  START TRANSACTION;

  -- Insert the BUY order
  INSERT INTO Orders (
    ticker, trader_id, order_type, quantity, price, status, order_date
  ) VALUES (
    p_ticker, p_trader_id, 'BUY', p_quantity, p_max_price, 'OPEN',
CURRENT_TIMESTAMP
  );

  SET v_buy_order_id = LAST_INSERT_ID();
  SET v_remaining_to_buy = p_quantity;

  -- Try to match against open SELL orders
  WHILE_LOOP: WHILE v_remaining_to_buy > 0 DO
    SELECT o.order_id, o.trader_id, o.price, o.quantity,
           IFNULL(SUM(t.quantity), 0) AS filled_quantity
    INTO v_seller_order_id, v_seller_id, v_sell_price, v_seller_order_quantity,
v_seller_filled
    FROM Orders o
    LEFT JOIN Transactions t ON o.order_id = t.order_id
    WHERE o.ticker = p_ticker
      AND o.order_type = 'SELL'
      AND o.status IN ('OPEN', 'PARTIALLY_FILLED')
      AND o.price <= p_max_price
    GROUP BY o.order_id
    ORDER BY o.price ASC, o.order_date ASC
    LIMIT 1;

    IF v_seller_order_id IS NULL THEN
      LEAVE WHILE_LOOP;
    END IF;

    SET v_to_transfer = LEAST(v_remaining_to_buy, v_seller_order_quantity -
```

```
v_seller_filled);

    -- Record transactions for both buyer and seller
    INSERT INTO Transactions (order_id, quantity, price_per_share,
transaction_type, transaction_date)
    VALUES
      (v_buy_order_id, v_to_transfer, v_sell_price, 'BUY', CURRENT_TIMESTAMP),
      (v_seller_order_id, v_to_transfer, v_sell_price, 'SELL',
CURRENT_TIMESTAMP);

    -- Update SELL order status
    UPDATE Orders
    SET status = CASE
      WHEN (v_seller_filled + v_to_transfer) = v_seller_order_quantity THEN
'FILLED'
      ELSE 'PARTIALLY_FILLED'
    END
    WHERE order_id = v_seller_order_id;

    SET v_remaining_to_buy = v_remaining_to_buy - v_to_transfer;
  END WHILE;

  -- Update BUY order status
  UPDATE Orders
  SET status = CASE
    WHEN v_remaining_to_buy = p_quantity THEN 'OPEN'
    WHEN v_remaining_to_buy = 0 THEN 'FILLED'
    ELSE 'PARTIALLY_FILLED'
  END


  WHERE order_id = v_buy_order_id;

  COMMIT;
END$$

DELIMITER ;

-- ----------------------------------------------------
-- procedure IssueInitialShares
-- ----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `IssueInitialShares`(
  IN p_ticker VARCHAR(10),
  IN p_quantity INT
)
BEGIN
  DECLARE v_order_id INT;

  -- Insert a fake "BUY" order to grant SystemIssuer ownership
  INSERT INTO orders (
    ticker, trader_id, order_type, quantity, price, status, order_date
  ) VALUES (
    p_ticker, 1, 'BUY', p_quantity, 0.00, 'ISSUED', CURRENT_TIMESTAMP
```

```
  );

  SET v_order_id = LAST_INSERT_ID();

  -- Insert a fake transaction to reflect the issued shares
  INSERT INTO transactions (
    order_id, quantity, price_per_share, transaction_type, transaction_date
  ) VALUES (
    v_order_id, p_quantity, 0.00, 'BUY', CURRENT_TIMESTAMP
  );
END$$

DELIMITER ;

-- ----------------------------------------------------
-- procedure SellStock
-- ----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `SellStock`(
  IN p_trader_id INT,
  IN p_ticker VARCHAR(10),
  IN p_quantity INT,
  IN p_min_price FLOAT
)
BEGIN
  DECLARE v_sell_order_id INT;
  DECLARE v_remaining_to_sell INT;
  DECLARE v_buyer_order_id INT;
  DECLARE v_buyer_id INT;
  DECLARE v_buyer_price FLOAT;
  DECLARE v_buyer_filled INT;
  DECLARE v_buyer_order_quantity INT;
  DECLARE v_to_transfer INT;
  DECLARE v_current_quantity INT;

  START TRANSACTION;

  -- Check if the trader has enough holdings using the view
  SELECT quantity INTO v_current_quantity
  FROM current_holdings
  WHERE trader_id = p_trader_id AND ticker = p_ticker;

  IF v_current_quantity >= p_quantity THEN
    -- Insert the SELL order
    INSERT INTO Orders (
      ticker, trader_id, order_type, quantity, price, status, order_date
    ) VALUES (
      p_ticker, p_trader_id, 'SELL', p_quantity, p_min_price, 'OPEN',
CURRENT_TIMESTAMP
    );

    SET v_sell_order_id = LAST_INSERT_ID();
    SET v_remaining_to_sell = p_quantity;
```

```
    -- Try to match against open BUY orders
    WHILE_LOOP: WHILE v_remaining_to_sell > 0 DO
      SELECT o.order_id, o.trader_id, o.price, o.quantity,
             IFNULL(SUM(t.quantity), 0) AS filled_quantity
      INTO v_buyer_order_id, v_buyer_id, v_buyer_price, v_buyer_order_quantity,
v_buyer_filled
      FROM Orders o
      LEFT JOIN Transactions t ON o.order_id = t.order_id
      WHERE o.ticker = p_ticker
        AND o.order_type = 'BUY'
        AND o.status IN ('OPEN', 'PARTIALLY_FILLED')
        AND o.price >= p_min_price
      GROUP BY o.order_id
      ORDER BY o.price DESC, o.order_date ASC
      LIMIT 1;

      IF v_buyer_order_id IS NULL THEN
        LEAVE WHILE_LOOP;
      END IF;

      SET v_to_transfer = LEAST(v_remaining_to_sell, v_buyer_order_quantity -
v_buyer_filled);

      -- Record transactions for both buyer and seller
      INSERT INTO Transactions (order_id, quantity, price_per_share,
transaction_type, transaction_date)
      VALUES
        (v_buyer_order_id, v_to_transfer, v_buyer_price, 'BUY',
CURRENT_TIMESTAMP),
        (v_sell_order_id, v_to_transfer, v_buyer_price, 'SELL',
CURRENT_TIMESTAMP);

      -- Update BUY order status
      UPDATE Orders
      SET status = CASE
        WHEN (v_buyer_filled + v_to_transfer) = v_buyer_order_quantity THEN
'FILLED'
        ELSE 'PARTIALLY_FILLED'
      END
      WHERE order_id = v_buyer_order_id;

      SET v_remaining_to_sell = v_remaining_to_sell - v_to_transfer;
    END WHILE;

    -- Update SELL order status
    UPDATE Orders
    SET status = CASE
         WHEN v_remaining_to_sell = p_quantity THEN 'OPEN'
      WHEN v_remaining_to_sell = 0 THEN 'FILLED'
      ELSE 'PARTIALLY_FILLED'
    END
    WHERE order_id = v_sell_order_id;

    COMMIT;
  ELSE
    ROLLBACK;
```

```
  END IF;
END$$

DELIMITER ;

-- ----------------------------------------------------
-- procedure TakeHoldingsSnapshot
-- ----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `TakeHoldingsSnapshot`()
BEGIN
  DECLARE v_snapshot_id INT;

  -- Step 1: Insert new snapshot metadata
  INSERT INTO snapshot_metadata () VALUES ();
  SET v_snapshot_id = LAST_INSERT_ID();

  -- Step 2: Insert current holdings into history
  INSERT INTO holdings_history (snapshot_id, trader_id, ticker, quantity)
  SELECT v_snapshot_id, trader_id, ticker, quantity
  FROM current_holdings;

END$$

DELIMITER ;

-- ----------------------------------------------------
-- function format_big_number
-- ----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE DEFINER=`root`@`localhost` FUNCTION `format_big_number`(val DOUBLE)
RETURNS varchar(20) CHARSET utf8mb3
    DETERMINISTIC
BEGIN
  RETURN CASE
    WHEN val >= 1000000000 THEN CONCAT(ROUND(val / 1000000000, 2), 'B')
    WHEN val >= 1000000 THEN CONCAT(ROUND(val / 1000000, 2), 'M')
    WHEN val >= 1000 THEN CONCAT(ROUND(val / 1000, 2), 'K')
    ELSE ROUND(val, 2)
  END;
END$$

DELIMITER ;

-- ----------------------------------------------------
-- function get_last_price
-- ----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE DEFINER=`root`@`localhost` FUNCTION `get_last_price`(p_ticker VARCHAR(10))
RETURNS float
```

```
    DETERMINISTIC
BEGIN
  DECLARE v_price FLOAT;

  SELECT t.price_per_share
  INTO v_price
  FROM Transactions t
  JOIN Orders o ON t.order_id = o.order_id
  WHERE o.ticker = p_ticker
  ORDER BY t.transaction_date DESC, t.transaction_id DESC
  LIMIT 1;

  RETURN v_price;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- View `mydb`.`current_holdings`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`current_holdings`;
USE `mydb`;
CREATE  OR REPLACE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY
DEFINER VIEW `mydb`.`current_holdings` AS select `o`.`trader_id` AS
`trader_id`,`o`.`ticker` AS `ticker`,sum((case `t`.`transaction_type` when 'BUY'
then `t`.`quantity` when 'SELL' then -(`t`.`quantity`) end)) AS `quantity` from
(`mydb`.`transactions` `t` join `mydb`.`orders` `o` on((`t`.`order_id` =
`o`.`order_id`))) group by `o`.`trader_id`,`o`.`ticker` having (`quantity` > 0);

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```