Untitled Notebook

Anonymous

2025 年 6 月 19 日

终极约束设计

my_stock(stock_id, volume, avg_price, profit)：表示所持有的股票编号、数量、持仓平均价格、利润

trans(trans_id,stock_id, date, price, amount, sell_or_buy)：表示一次交易的编号、股票编号、交易日期、成交价格、成交数量、买入还是卖出

使用触发器完成下面的工作：

1. 往 trans 里面插入一条记录时，根据其是买入还是卖出，调整 my_stock 中的 volume 以及 avg_price。如果是初次插入的股票交易，就在 my_stock 中为该股票新建一条记录，profit 置为 0。注意，如果一笔卖出交易的 amount 大于 my_stock 中该股票的 volume，说明是无效的下单交易，应该加以拒绝，直接抛弃。 平均价格的计算：

$$\text{avg\_ price} = \frac{\text{volume} \times \text{avg\_ price} + \text{price} \times \text{amount}}{\text{volume} + \text{amount}}$$

1. profit 的计算方式如下：每当有卖出交易发生时，将其与尽可能远的买入交易进行匹配，比如如果 trans 中现有的记录为{(t01,s01,d01,10,1000,buy), (t02,s01,d02,12,500,buy)},如果现在插入{(t03,s01,d03,11,700,sold)}，本次交易产生的 profit=(11-10)*700,如果再插入{(t04,s01,d04,9,700,sold)}，本次交易产生的 profit=(9-10)*300 + (9-12)*400 = -1500.将每次卖出交易的 profit 都累加到 my_stock 的 profit 上。

```
# 建表
import sqlite3

sqlite3.enable_callback_tracebacks(True)

conn = sqlite3.connect("stocks")
cursor = conn.cursor()

cursor.execute('''
CREATE TABLE IF NOT EXISTS my_stock (
  stock_id  TEXT    PRIMARY KEY,
  volume    INTEGER NOT NULL,
  avg_price REAL    NOT NULL,
  profit    REAL    NOT NULL DEFAULT 0
);
''')

cursor.execute('''
CREATE TABLE IF NOT EXISTS trans (
  trans_id    TEXT    PRIMARY KEY,
  stock_id    TEXT      NOT NULL REFERENCES my_stock(stock_id),
  date        DATE      NOT NULL,
  price       REAL      NOT NULL,
  amount      INTEGER   NOT NULL,
  sell_or_buy TEXT      NOT NULL CHECK(sell_or_buy IN ('buy','sold'))
);
''')

conn.commit()
print("建表完成")
```
[1]:

```
建表完成
```

```
# 打印表结构
def print_table_schema(table_name, db_path="stocks"):
    """
    打印指定 SQLite 数据库中某个表的结构信息。

    参数:
        table_name (str): 要查看结构的表名。
    """
    try:
        # 连接到 SQLite 数据库
        conn = sqlite3.connect(db_path)
        cursor = conn.cursor()

        # 执行 PRAGMA table_info 命令获取表的结构信息
        cursor.execute(f"PRAGMA table_info('{table_name}')")
        columns = cursor.fetchall()

        if not columns:
            print(f"表 '{table_name}' 不存在或没有列信息。")
            return

        # 打印表结构信息
        print(f"表 '{table_name}' 的结构信息:")
        print("{:<5} {:<20} {:<15} {:<10} {:<15} {:<5}".format(
            "cid", "name", "type", "notnull", "dflt_value", "pk"
        ))
        print("-" * 80)
        for col in columns:
            cid, name, col_type, notnull, dflt_value, pk = col
            print("{:<5} {:<20} {:<15} {:<10} {:<15} {:<5}".format(
                cid, name, col_type, notnull, str(dflt_value), pk
            ))
    except sqlite3.Error as e:
        print(f"发生错误: {e}")
    finally:
        # 关闭数据库连接
        if conn:
            conn.close()
```
[2]:

```
def print_table_data(table_name, db_path="stocks"):
    """
    打印指定 SQLite 数据库中某个表的所有数据，并对齐列。

    参数:
        table_name (str): 要查看数据的表名。
    """
    try:
        # 连接到 SQLite 数据库
        conn = sqlite3.connect(db_path)
        cursor = conn.cursor()
        # 构建 SQL 查询语句
        query = f"SELECT * FROM {table_name}"
        # 执行查询
        cursor.execute(query)
        rows = cursor.fetchall()
        # 获取列名
        column_names = [description[0] for description in cursor.description]
        # 获取列类型信息
        cursor.execute(f"PRAGMA table_info('{table_name}')")
        table_info = cursor.fetchall()
        column_types = [info[2].upper() for info in table_info]
        # 计算每列的最大宽度
        col_widths = [len(col) for col in column_names]
        for row in rows:
            for idx, item in enumerate(row):
                if isinstance(item, float):
                    item_str = f"{item:.2f}"
                else:
                    item_str = str(item)
                col_widths[idx] = max(col_widths[idx], len(item_str))

        # 构建格式化字符串
        format_str = " | ".join(f"{{:<{width}}}" for width in col_widths)
        # 打印列名
        print(f"表 '{table_name}' 的数据：")
        print(format_str.format(*column_names))
        print("-" * (sum(col_widths) + 3 * (len(col_widths) - 1)))
        # 打印每一行数据
        for row in rows:
            formatted_row = []
            for idx, item in enumerate(row):
             if column_types[idx] in ('REAL', 'FLOAT', 'DOUBLE') and isinstance(item,
float):
                    formatted_item = f"{item:.2f}"
                else:
                    formatted_item = str(item)
                formatted_row.append(formatted_item)
            print(format_str.format(*formatted_row))

    except sqlite3.Error as e:
        print(f"发生错误: {e}")
    finally:
        # 关闭数据库连接
        if conn:
            conn.close()
```
[3]:

[4]:
```
print_table_schema("my_stock")
```

```
表 'my_stock' 的结构信息:cid   name             type         notnull    dflt_value     pk
--------------------------------------------------------------------------------0
stock_id              TEXT            0           None           1     1       volume
INTEGER            1             None            0     2      avg_price             REAL
1            None             0     3      profit               REAL               1
0             0
```

[5]:
```
print_table_schema("trans")
```

```
表 'trans' 的结构信息:cid   name             type         notnull    dflt_value     pk
--------------------------------------------------------------------------------0
trans_id              TEXT            0           None           1     1      stock_id
TEXT               1             None            0     2      date                 DATE
1            None             0     3      price                REAL               1
None              0     4      amount             INTEGER          1             None
0     5      sell_or_buy          TEXT            1           None            0
```

根据要求，我们需要完成 3 个触发器：

1. 拒绝无效卖出交易
2. 买入后更新持仓数量与加权平均价
3. 卖出后更新持仓数量与加权平均价

[6]:
```
# 拒绝无效卖出交易
cursor.execute('''
DROP TRIGGER IF EXISTS before_insert_trans
''')

cursor.execute('''
CREATE TRIGGER before_insert_trans
BEFORE INSERT ON trans
WHEN NEW.sell_or_buy = 'sold'
BEGIN
  -- 如果卖出数量超过当前持仓，Abort 并抛错
  -- 当前甚至没有持仓的话，数量为0
  SELECT RAISE(ABORT, '卖出数量超过持仓，交易被拒绝')
  WHERE NEW.amount > COALESCE(
    (SELECT volume FROM my_stock WHERE stock_id = NEW.stock_id), 0
  );
END;
''')
conn.commit()
```

```
# 买入后更新持仓数量与加权平均价
cursor.execute('''
DROP TRIGGER IF EXISTS after_insert_buy
''')

cursor.execute('''
CREATE TRIGGER after_insert_buy
AFTER INSERT ON trans
WHEN NEW.sell_or_buy = 'buy'
BEGIN
```

```
  -- 如果已有持仓，更新 volume 和 avg_price；否则插入新记录
  INSERT INTO my_stock(stock_id, volume, avg_price, profit)
  VALUES (
    NEW.stock_id,
    NEW.amount,
    NEW.price,
    0
  )
  ON CONFLICT(stock_id) DO UPDATE SET
    volume    = volume + NEW.amount,
    avg_price = (volume * avg_price + NEW.price * NEW.amount)
                / (volume + NEW.amount);
END;
''')
```

[7]:
```
conn.commit()
```

```
# 卖出后更新持仓数量与加权平均价
# 注意：检查是否存在这只股票的触发器已经在前面定义了
cursor.execute('''
DROP TRIGGER IF EXISTS after_insert_sell
''')

cursor.execute('''
CREATE TRIGGER after_insert_sell
AFTER INSERT ON trans
WHEN NEW.sell_or_buy = 'sold'
BEGIN
  -- 扣减持仓
  UPDATE my_stock
    SET volume = volume - NEW.amount
    WHERE stock_id = NEW.stock_id;
END;
''')
```

[8]:
```
conn.commit()
```

由于 SQLite3 不支持循环等复杂操作，我们在 Python 中实现先入先出（FIFO）式的匹配股票操作，并且使用 Sqlite3 的 `create_function` 将 Python 函数暴露给 SQL 引擎。同时，我们创建一个新的表 `sale_buy_alloc` 来进行配对记录。

考虑到在读数据库的时候数据库被上锁的问题，我们新建一个数据库来保存分配信息。

但是问题在于 SQLite 不支持在触发器中使用 UDF。不过，Gunter Hick 给出了一个解决方案："The trigger program is compiled when the `CREATE TRIGGER` statement is executed (during initial execution or when the schema is loaded from the file). Any functions referenced in triggers need to be defined at that point in time. Your schema will fail to load unless the functions referenced in trigger programs are defined.

You should be able to build an extension that creates your user defined functions and then executes one or more `CREATE TEMP TRIGGER` statements that use the now-defined functions. See https://sqlite.org/loadext.html"。

我们在这里使用这种方案解决 SQLite 的这个问题。

```
conn_alloc = sqlite3.connect("alloc")
cursor_alloc = conn_alloc.cursor()

cursor_alloc.execute('''
CREATE TABLE IF NOT EXISTS sale_buy_alloc (
  sale_trans_id TEXT NOT NULL,
  buy_trans_id  TEXT NOT NULL,
  alloc_amt     INTEGER NOT NULL,
  PRIMARY KEY (sale_trans_id, buy_trans_id)
);
''')

cursor_alloc.execute('''
DELETE FROM sale_buy_alloc
''')
conn_alloc.commit()
```

[9]:
```
print_table_schema("sale_buy_alloc", db_path="alloc")
```

```
表        'sale_buy_alloc'     的    结    构    信    息    :     cid                name
type                                          notnull            dflt_value                    pk
-----------------------------------------------------------------------------------------------0
sale_trans_id        TEXT           1         None           1    1      buy_trans_id
TEXT              1           None             2    2      alloc_amt               INTEGER
1          None              0
```

```
def calculate_profit(sale_trans_id, stock_id, sell_price, sell_amt):
    conn = sqlite3.connect('stocks')
    conn.row_factory = sqlite3.Row  # 让 fetch 回来的是 dict-like 行
    cursor = conn.cursor()

    profit = 0.0

    # 把另一个数据库里的东西挂载进来
    cursor.execute("ATTACH DATABASE ? AS allocdb", ('alloc',))

    # 查询所有未完全分配的买入记录
    cursor.execute("""
        SELECT
            t.trans_id,
            t.price AS buy_price,
            t.amount - COALESCE(SUM(a.alloc_amt), 0) AS remain
        FROM trans t
        LEFT JOIN allocdb.sale_buy_alloc a
            ON t.trans_id = a.buy_trans_id
        WHERE t.stock_id = ?
          AND t.sell_or_buy = 'buy'
        GROUP BY t.trans_id, t.price, t.amount
        HAVING remain > 0
        ORDER BY t.date ASC
    """, (stock_id,))

    conn_alloc = sqlite3.connect('alloc')
    cursor_alloc = conn_alloc.cursor()

    for buy in cursor:
        if sell_amt <= 0:
            break
        buy_trans_id, buy_price, remain = buy[0], buy[1], buy[2]
        match_amt = min(remain, sell_amt)
        profit += (sell_price - buy_price) * match_amt
        # 写入分配关系
        cursor_alloc.execute("INSERT INTO sale_buy_alloc "
                             "(sale_trans_id, buy_trans_id, alloc_amt) "
                             "VALUES (?, ?, ?)",
                             (sale_trans_id, buy_trans_id, match_amt))
        sell_amt -= match_amt

    conn_alloc.commit()
    conn.commit()

    conn_alloc.close()
    conn.close()
[ ]:    return profit
```

```
# 卖出后计算利润
cursor.execute('''
DROP TRIGGER IF EXISTS profit_after_buy
''')

# 注册 UDF，-1表示可以有不限制的参数个数
conn.create_function("calculate_profit", -1, calculate_profit)

cursor.execute('''
CREATE TEMP TRIGGER profit_after_buy
AFTER INSERT ON trans
WHEN NEW.sell_or_buy = 'sold'
BEGIN
  -- 调用 Python 注册的 UDF，执行 FIFO 配对 & 更新 profit 并返回本次 profit
  UPDATE my_stock
     SET profit = profit + calculate_profit(NEW.trans_id, NEW.stock_id, NEW.price,
NEW.amount)
    WHERE stock_id = NEW.stock_id;
END;
''')
[11]: conn.commit()
```

现在我们使用提供的触发器的测试数据进行测试。

```
# 清空表
cursor.execute("DELETE FROM my_stock")
cursor.execute("DELETE FROM trans")
[12]: conn.commit()
```

```
def test(trans_id, stock_id, date, price, amount, sell_or_buy):
    try:
        cursor.execute("INSERT INTO trans"
                       "(trans_id, stock_id, date, price, amount, sell_or_buy)"
                       "VALUES (?,?,?,?,?,?)",
                       (trans_id, stock_id, date, price, amount, sell_or_buy))
        conn.commit()
        print_table_data("my_stock")
        print_table_data("trans")
    except Exception as e:
        print("触发器拦截非法操作:", e)
[13]:   conn.rollback()
```

```
[14]: test('1', '1', '2025-01-01', 10, 1000, 'buy')
```

```
表    'my_stock'   的  数  据  :  stock_id   |   volume   |   avg_price   |
profit---------------------------------------1             | 1000     | 10.00       |
0.00   表 'trans' 的数据: trans_id | stock_id | date           | price | amount |
sell_or_buy--------------------------------------------------------------------1        |
1         | 2025-01-01 | 10.00 | 1000    | buy
```

```
[15]: test('2', '1', '2025-01-02', 11, 500, 'buy')
```

```
表    'my_stock'   的  数  据  :  stock_id   |   volume   |   avg_price   |
profit---------------------------------------1             | 1500     | 10.33       |
0.00   表 'trans' 的数据: trans_id | stock_id | date           | price | amount |
sell_or_buy--------------------------------------------------------------------1        |
1         | 2025-01-01 | 10.00 | 1000    | buy         2         | 1          | 2025-01-02
| 11.00 | 500     | buy
```

```
[16]: test('3', '1', '2025-01-03', 12, 800, 'sold')
```

```
10.0表   'my_stock'   的 数 据 :  stock_id   |   volume   |  avg_price   |   profit
---------------------------------------1              | 700        | 10.33        |
1600.00表 'trans' 的数据: trans_id | stock_id | date           | price | amount |
sell_or_buy--------------------------------------------------------------------1        |
1         | 2025-01-01 | 10.00 | 1000    | buy         2         | 1          | 2025-01-02
| 11.00 | 500      | buy          3          | 1           | 2025-01-03 | 12.00 | 800
| sold
```

```
[17]: test('4', '1', '2025-01-04', 12.0, 1000, 'sold')
```

```
触发器拦截非法操作: 卖出数量超过持仓, 交易被拒绝
```

```
[18]: test('5', '1', '2025-01-05', 9.0, 1000, 'buy')
```

```
表    'my_stock'   的  数  据  :  stock_id   |   volume   |  avg_price   |   profit
---------------------------------------1              | 1700      | 9.55         |
1600.00表 'trans' 的数据: trans_id | stock_id | date           | price | amount |
sell_or_buy--------------------------------------------------------------------1        |
1         | 2025-01-01 | 10.00 | 1000    | buy         2         | 1          | 2025-01-02
| 11.00 | 500     | buy         3          | 1          | 2025-01-03 | 12.00 | 800     |
sold       5          | 1          | 2025-01-05 | 9.00  | 1000    | buy
```

```
[19]: test('6', '1', '2025-01-06', 12.0, 800, 'sold')
```

```
10.011.09.0表   'my_stock'  的 数 据 : stock_id  |  volume  |  avg_price  |  profit
---------------------------------------1              |  900       |  9.55          |
2800.00表 'trans' 的数据: trans_id | stock_id | date         | price | amount |
sell_or_buy----------------------------------------------------------------1       |
1        | 2025-01-01 | 10.00 | 1000    | buy        2         | 1         | 2025-01-02
| 11.00 | 500     | buy        3         | 1          | 2025-01-03 | 12.00 | 800    |
sold       5         | 1         | 2025-01-05 | 9.00  | 1000    | buy        6         |
1         | 2025-01-06 | 12.00 | 800     | sold
```

```
[20]: test('7', '1', '2025-01-07', 7.0, 800, 'sold')
```

```
9.0表    'my_stock'   的  数  据  :  stock_id   |   volume   |   avg_price   |   profit
---------------------------------------1              |  100       |  9.55          |
1200.00表 'trans' 的数据: trans_id | stock_id | date         | price | amount |
sell_or_buy----------------------------------------------------------------1       |
1        | 2025-01-01 | 10.00 | 1000    | buy        2         | 1         | 2025-01-02
| 11.00 | 500     | buy        3         | 1          | 2025-01-03 | 12.00 | 800    |
sold       5         | 1         | 2025-01-05 | 9.00  | 1000    | buy        6         |
1        | 2025-01-06 | 12.00 | 800     | sold       7         | 1         | 2025-01-07
| 7.00  | 800     | sold
```

```
[21]: print_table_data("sale_buy_alloc", db_path="alloc")
```

```
表      'sale_buy_alloc'    的   数   据   :   sale_trans_id    |    buy_trans_id    |
alloc_amt----------------------------------------3              | 1            | 800
6             | 1             | 200      6             | 2             | 500      6
| 5             | 100      7             | 5             | 800
```