MLDS-413 Introduction to Databases and Information Retrieval

Homework 8: Triggers, Integrity Constraints, Transactions, Views, and Window Functions

Ayush Agarwal

Name 1:

_____

scg1143

NetID 1:

_____

Lowan(Sydney) Li

Name 2:

_____

chr0390

NetID 2:

_____

Instructions

You should submit this homework assignment via Canvas. Acceptable formats are word files, text files, and pdf files. Paper submissions are not allowed and they will receive an automatic zero.

As explained during lecture and in the syllabus, assignments are done in groups. The groups have been created and assigned. Each group needs to submit only one assignment (i.e., there is no need for both partners to submit individually the same homework assignment).

Each group can submit solutions multiple times (for example, you may discover an error in your earlier submission and choose to submit a new solution set). We will grade only the last submission and ignore earlier ones.

Make sure you submit your solutions before the deadline. The policies governing academic integrity, tardiness and penalties are detailed in the syllabus.

SchoolScheduling.sqlite Database (30 points)

1. **(10 points)** Write the SQL statements that perform the following operations.
   a. **(1 point)** First, in preparation for this section of the homework, write and execute a query that inserts a new class status with ID 4 and description "Failed".

```
INSERT
INTO
Student_Class_Status (ClassStatus, ClassStatusDescription) VALUES (4, 'Failed');
```

Output After Select Query

	ClassStatus	ClassStatusDescription
1	4	Failed

   b. **(3 points)** Start a new transaction.

```
BEGIN;
```

   c. **(3 points)** Update the class status in student 1001's schedule for class 4180 to "Completed".

Before Running Update Query

```
SELECT
        ss.StudentID , ss.Grade , ss.ClassStatus , scs.ClassStatusDescription
FROM
        Student_Schedules ss
INNER JOIN
        Student_Class_Status scs
ON ss.ClassStatus = scs.ClassStatus
where
        ss.StudentID = 1001 and ss.ClassID  = 4180
```

	StudentID	Grade	ClassStatus	ClassStatusDescription
1	1,001	0	1	Enrolled

After Running Update Query

UPDATE Query : To Update Table

```
UPDATE Student_Schedules
set ClassStatus = (SELECT ClassStatus from Student_Class_Status where ClassStatusDescription='Completed')
```

```
where
StudentID = 1001 and ClassID = 4180;
```

SELECT Query : To Fetch Results

```
SELECT
        ss.StudentID , ss.Grade , ss.ClassStatus , scs.ClassStatusDescription
FROM
        Student_Schedules ss
INNER JOIN
        Student_Class_Status scs
ON ss.ClassStatus = scs.ClassStatus
where
        ss.StudentID = 1001 and ss.ClassID  = 4180;
```

	StudentID	Grade	ClassStatus	ClassStatusDescription
1	1,001	0	2	Completed

d. **(3 points)** The problem now is that if the administrator forgets to set 1001's grade for class 4180, the student will have a class marked as completed with grade 0. You want to enforce a rule that a class cannot be marked completed unless the student has already received a passing grade, i.e., a grade of at least 60.0. You want to implement your integrity rules first and then do the data updates. Abort the transaction you started in part (b) in order to undo the changes in part (c). Do not undo the changes of part (a), though.

```
ROLLBACK;
```

After Rollback ClassStatusDescription Should be changed to Enrolled

```
SELECT
        ss.StudentID , ss.Grade , ss.ClassStatus , scs.ClassStatusDescription
FROM
        Student_Schedules ss
INNER JOIN
        Student_Class_Status scs
ON ss.ClassStatus = scs.ClassStatus
where
        ss.StudentID = 1001 and ss.ClassID  = 4180;
```

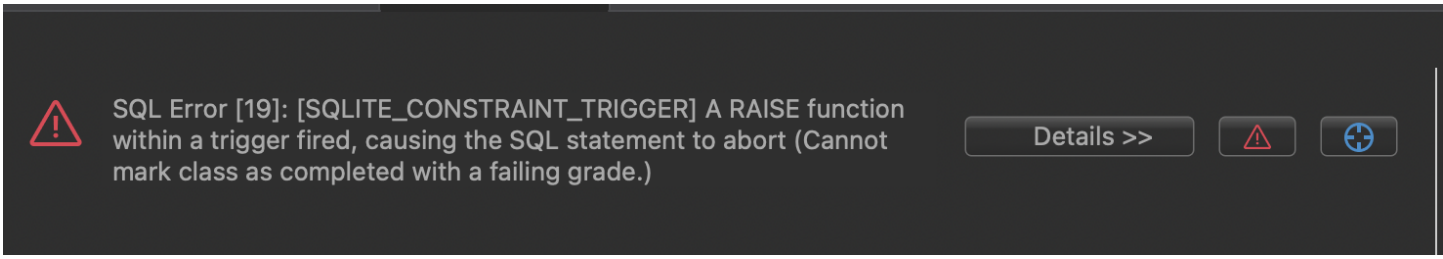
	StudentID	Grade	ClassStatus	ClassStatusDescription
1	1,001	0	1	Enrolled

2. **(10 points)** Write a query that enforces the data integrity constraint described in question Q2.d.

```
create Trigger checkGrade
before UPDATE
on Student_Schedules for each row
WHEN new.ClassStatus = (SELECT ClassStatus from Student_Class_Status where
ClassStatusDescription='Completed')
and new.Grade < 60.0
BEGIN
        SELECT RAISE(ROLLBACK, 'Cannot mark class as completed with a failing grade.');
END;
```

Output after executing Update Statement with trigger enabled

```
UPDATE Student_Schedules
set ClassStatus = (SELECT ClassStatus from Student_Class_Status where ClassStatusDescription='Completed')
where
StudentID = 1001 and ClassID = 4180;
```



3. **(10 points)** Write a query that enforces the following data integrity constraint: when a grade changes from a non-passing grade to a passing grade, automatically set the status of the class to completed. If the grade changes to a non-passing grade, set the class status to failed.

```
create Trigger setGrade
after UPDATE
on Student_Schedules for each row
BEGIN
        UPDATE Student_Schedules
        set ClassStatus = CASE
                WHEN new.Grade >= 60 and old.Grade < 60 then (SELECT ClassStatus from Student_Class_Status
where ClassStatusDescription='Completed')
                WHEN new.Grade < 60 and old.Grade >= 60 then (SELECT ClassStatus from Student_Class_Status
where ClassStatusDescription='Failed')
        END
        WHERE StudentID = NEW.StudentID AND ClassID = NEW.ClassID;
END;
```

Examples:

Case 1: Grade < 60, Status Should be marked as Completed when new grade > 60

- **Before Update Query**

```
SELECT
	ss.StudentID , ss.Grade , ss.ClassStatus , scs.ClassStatusDescription
FROM
	Student_Schedules ss
INNER JOIN
	Student_Class_Status scs
ON ss.ClassStatus = scs.ClassStatus
where
	ss.StudentID = 1001 and ss.ClassID  = 5917;
```

	StudentID	Grade	ClassStatus	ClassStatusDescription
1	1,001	0	1	Enrolled

- **After Update Query**

```
UPDATE Student_Schedules
set Grade = 70
where
StudentID = 1001 and ClassID  = 5917;
```

	StudentID	Grade	ClassStatus	ClassStatusDescription
1	1,001	70	2	Completed

Case 2: Grade > 60 Status should set to failed when new grade is less than 60

- **Before Update Query**

```
SELECT
	ss.StudentID , ss.Grade , ss.ClassStatus , scs.ClassStatusDescription
FROM
	Student_Schedules ss
INNER JOIN
	Student_Class_Status scs
ON ss.ClassStatus = scs.ClassStatus
where
	ss.StudentID = 1003and ss.ClassID  = 2911;
```

	StudentID	Grade	ClassStatus	ClassStatusDescription
1	1,003	85.39	2	Completed

- **After Update Query**

```
UPDATE Student_Schedules
set Grade = 50
where
StudentID = 1003 and ClassID  = 2911;
```

	StudentID	Grade	ClassStatus	ClassStatusDescription
1	1,003	50	4	Failed

Homework 5 Question 6 Solution Database (10 points)

4. **(10 points)** Sometimes you want to only check integrity constraints, not enforce them. One way to do that is to create a view that you examine whenever you want to verify data integrity. One such example are the constraints (e), (g), (i), (k), and (n) in Homework 5 Question 6.

   For this assignment, you will use the Homework 5 Question 6 solution database. Your goal is to create a view named `check_db` that checks if the database violates any of the constraints (e), (g), (i), (k), and (n). The view should return a table that lists all the violated constraints, or an empty row if there are no violations. For example, if constraints e, g, i, k, and n are all violated, the view will be the table below (do not worry about the row order):

	ERRORS_FOUND
1	e
2	g
3	i
4	k
5	n

   Note that it is OK if some of the rows of your result are empty rows. Similarly, if no constraints are violated, the view could simply return a table with an empty row:

	ERRORS_FOUND
1	

   To check that your view works properly, you can execute the following deletions on the Homework 5 Question 6 solution database and check the output of your query after each deletion set, as the comments and the SQL queries below show. The queries should be executed in the exact order below to achieve each of the stated results.

   Suggestion: use transactions when you are experimenting in this question. This way, when things don't work, you can simply rollback the changes. Note that if you rollback once you will need to start a new transaction again to be able to rollback your changes a second time (`ROLLBACK` will undo your changes AND terminate the transaction). So, it is better to use `SAVEPOINT X` and `ROLLBACK TO X`. This way you can issue `ROLLBACK TO X` as many times as you want without taking a new savepoint (`ROLLBACK TO X` will undo the changes but the

transaction remains active, so there is no need to remember to start a new one each time).

```
-- check that your view works by examining it on the HW5 Q6 solution database before any deletions
-- your view should be just an empty row (i.e., there are no violations)
SELECT * FROM check_db;
```

	ErrorsFound
1	NULL

```
-- performing the following deletions would violate the following constraint
-- k. Each invoice has at least one invoice item
-- your view should contain a row for k
DELETE FROM invoice_items WHERE invoiceId IN (2, 3);
SELECT * FROM check_db;
```

	ErrorsFound
1	k

```
-- performing the following additional deletions would violate the following additional constraint
-- e. Each album has at least one track
-- your view should contain rows for e, k
DELETE FROM tracks WHERE albumId=2;
SELECT * FROM check_db;
```

	ErrorsFound
1	e
2	k

```
-- performing the following additional deletions would violate the following additional constraint
-- g. Each genre is represented by at least one track
-- your view should contain rows e, g, k
DELETE FROM tracks WHERE trackId=3451;
SELECT * FROM check_db;
```

	ErrorsFound
1	e
2	g
3	k

```
-- performing the following additional deletions would violate the following additional constraint
-- i. Each media type is used by at least one track
-- your view should contain rows for e, g, i, k
DELETE FROM invoice_items WHERE trackId IN (SELECT trackId FROM tracks WHERE mediaTypeId=4);
DELETE FROM tracks WHERE mediaTypeId=4;
SELECT * FROM check_db;
```

	ErrorsFound
1	e
2	g
3	i
4	k

```
-- performing the following additional deletions would violate the following additional constraint
-- n. Each customer has been issued at least one invoice
-- your view should contain rows for e, g, i, k, n
DELETE FROM invoice_items WHERE invoiceId IN (SELECT invoiceId FROM invoices WHERE customerId=20);
```

```
DELETE FROM invoices WHERE customerId=20;
SELECT * FROM check_db;
```

	ErrorsFound
1	e
2	g
3	i
4	k
5	n

```
DROP VIEW IF EXISTS check_db;

CREATE VIEW check_db AS
SELECT ErrorsFound
FROM (
   SELECT 'e' AS ErrorsFound
   FROM albums
   WHERE albumId NOT IN (SELECT albumId FROM tracks)
   UNION
   SELECT 'g' AS ErrorsFound
   FROM genres
   WHERE genreId NOT IN (SELECT genreId FROM tracks)
   UNION
   SELECT 'i' AS ErrorsFound
   FROM media_types
   WHERE mediaTypeId NOT IN (SELECT mediaTypeId FROM tracks)
   UNION
   SELECT 'k' AS ErrorsFound
   FROM invoices
   WHERE invoiceId NOT IN (SELECT invoiceId FROM invoice_items)
   UNION
   SELECT 'n' AS ErrorsFound
   FROM customers
   WHERE customerId NOT IN (SELECT DISTINCT customerId FROM invoices)

   UNION ALL

   SELECT NULL AS ErrorsFound
   WHERE NOT EXISTS (
      SELECT 1
      FROM albums
      WHERE albumId NOT IN (SELECT albumId FROM tracks)
      UNION
      SELECT 1
      FROM genres
      WHERE genreId NOT IN (SELECT genreId FROM tracks)
      UNION
      SELECT 1
      FROM media_types
      WHERE mediaTypeId NOT IN (SELECT mediaTypeId FROM tracks)
      UNION
      SELECT 1
      FROM invoices
      WHERE invoiceId NOT IN (SELECT invoiceId FROM invoice_items)
      UNION
      SELECT 1
      FROM customers
      WHERE customerId NOT IN (SELECT DISTINCT customerId FROM invoices)
   )
) AS Result;
```

SalesOrders.sqlite Database (10 points)

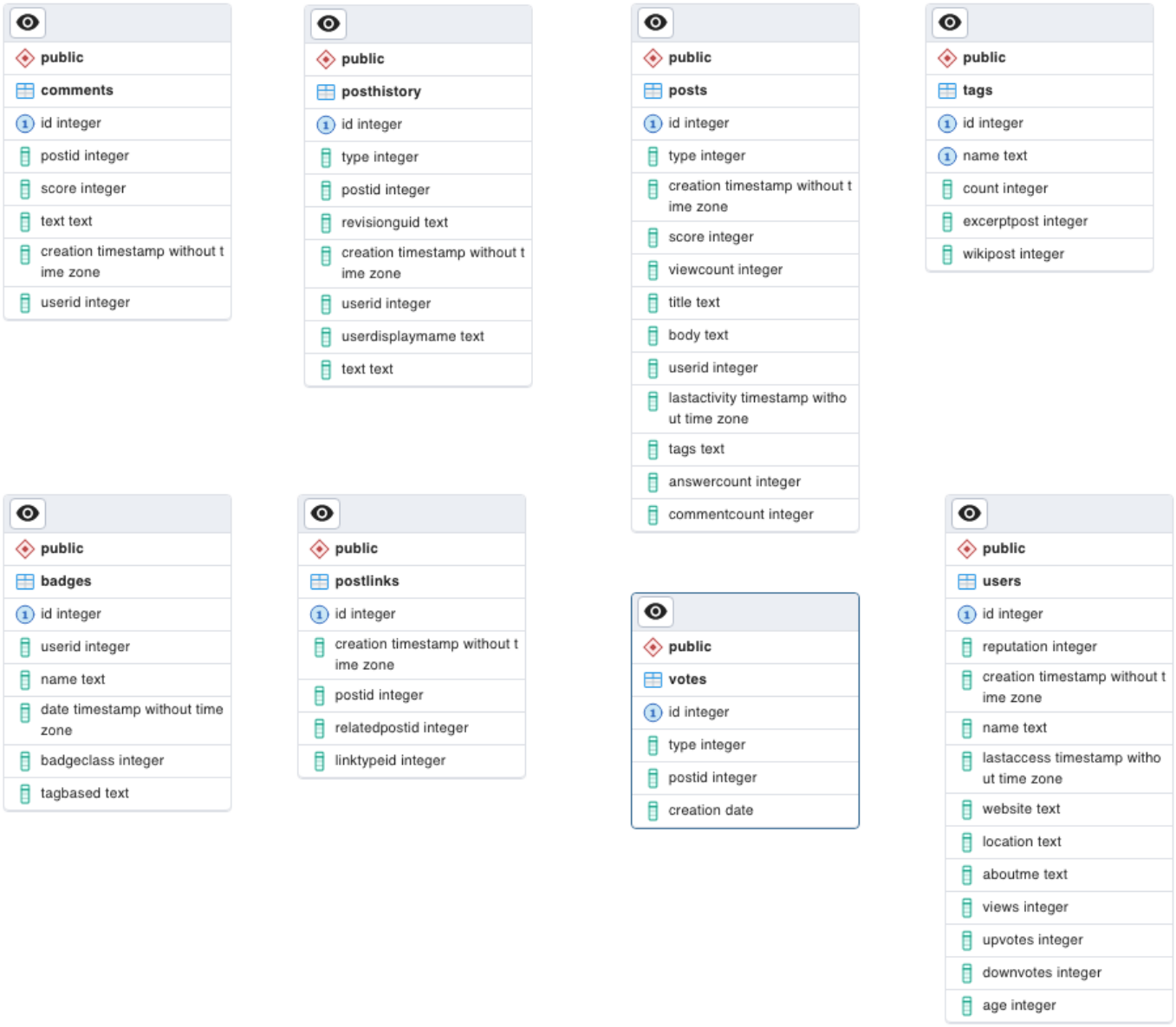
5. **(10 points)** Monthly revenue growth is defined as the percent of revenue change of a month relative to the previous month, i.e., $(M_i - M_{i-1}) / M_{i-1}$. Write a query that will return the revenue growth of the sales in the SalesOrders database and provide your query's output. This should be a single query (CTE, windowing allowed).

```
WITH MonthlyRevenue as (
        SELECT
                substr(o.OrderDate, 1, 7) as YearMonth ,
                SUM(od.QuotedPrice * od.QuantityOrdered) as MonthlyRevenue
        FROM
                Orders o
        INNER JOIN
                Order_Details od
        on o.OrderNumber = od.OrderNumber
        group by
                YearMonth
        order by
                YearMonth
),
PreviousMonthRevenue as (
        SELECT
                mr.YearMonth,
                mr.MonthlyRevenue,
                LAG(mr.MonthlyRevenue, 1, 0) OVER (order by mr.YearMonth) as PrevMonthRevenue
        FROM
                MonthlyRevenue as mr
)
SELECT
        YearMonth,
        MonthlyRevenue,
        PrevMonthRevenue,
        CASE
                WHEN PrevMonthRevenue = 0 THEN NULL
                ELSE ROUND((MonthlyRevenue - PrevMonthRevenue) / PrevMonthRevenue * 100.0,2)
        END as RevenueGrowthPerc
FROM
PreviousMonthRevenue
```

	YearMonth	MonthlyRevenue	PrevMonthRevenue	RevenueGrowthPerc
1	2012-09	816,015.8099999996	0	[NULL]
2	2012-10	714,377.6000000006	816,015.8099999996	-12.46
3	2012-11	749,934.0700000002	714,377.6000000006	4.98
4	2012-12	612,794.2400000005	749,934.0700000002	-18.29
5	2013-01	927,569.6699999999	612,794.2400000005	51.37
6	2013-02	764,849.8400000003	927,569.6699999999	-17.54

Stackoverflow Database (50 points)

Please follow the instructions from Homework 6 to connect to the Stackoverflow (so) database on MLDS's Postgres server. The database schema is provided below:



Please note that the Stackoverflow database does not give any information about the relationship between different entities. You need to analyze each table and sample some data to **infer yourself** the relationships between tables. Unfortunately, the real world is often messy.

You will use this database to answer the following questions. Please make sure that your queries in this homework are read-only.

Unless otherwise noted, for each question please provide:

- The query you constructed
- The output of that query
- Any other information requested by the question

6. **(10 points)** How many posts are there that have never been edited after creation. Please provide **two different solutions** for this question. Hint: You can use many operations such as LEFT JOIN, EXCEPT, and EXIST.

1st Approach

```
select COUNT(*) as never_edited_count
from
posts p
left join
posthistory p2
on p.id = p2.postid
where p2.postid is null;
```

2nd Approach

```
select COUNT(*) as never_edited_count
from (
        select id
        from posts p
        except
        select postid
        from posthistory p2
) as q;
```

7. **(10 points)** Write a SQL query to count the number of posts that were created on Christmas Day (December 25th) for each year. Present the results in ascending years.

```
select
        EXTRACT(year from p.creation) as year,
        COUNT(*) as posts_count
from
        posts p
where
        EXTRACT(month from p.creation) = 12 and EXTRACT(day from p.creation) = 25
group by
        year
order by
        year asc;
```

	123 year	123 posts_count
1	2,008	545
2	2,009	1,670
3	2,010	2,861
4	2,011	3,848
5	2,012	6,590
6	2,013	7,625
7	2,014	6,358
8	2,015	6,554
9	2,016	5,786
10	2,017	5,841
11	2,018	5,696
12	2,019	6,215
13	2,020	5,918
14	2,021	4,332
15	2,022	3,945

8. **(10 points)** Rank users by their reputation and assign a percentile rank. Print the id, name, reputation, and the percentile rank of user 19787814 (user id).

```
with RankedUsers as (
        select
                id,
                reputation ,
                name,
                percent_rank()  over (order by reputation DESC) as percentile_rank
        from
                users
)
select
        id,
        reputation ,
        name,
        percentile_rank
from
        RankedUsers
where
        id = 19787814;
```

	123 id	123 reputation	ABC name	123 percentile_rank
1	19,787,814	406	Nova	0.0253775167

9. **(10 points)** For the post with ID 7518463, find the related post (directly or indirectly) with the highest number of answers. In this question, you should only consider the postlink with linktypeid = 1. Hint: You need to use recursive query in this question. The directly related posts can be found in the table postlinks.

```
with recursive relatedPosts as (
    SELECT p.postid as postId, p.relatedpostid as relatedPostId, p2.answercount as answerCount
    FROM postlinks p
    INNER JOIN posts p2 ON p.relatedpostid = p2.id
    WHERE p.postid = 7518463 AND p.linktypeid = 1
    union ALL
    select p3.postid as postId, p3.relatedpostid as relatedPostId, p4.answercount  as answerCount
    from
    postlinks p3
    inner join
    relatedPosts as rp on p3.postid = rp.relatedPostId
    inner join
    posts p4 on p3.relatedpostid = p4.id
    where p3.linktypeid = 1
)
select relatedPostId, max(answerCount) as MaxAnswerCount
from
relatedPosts
where answerCount > 0
group by
    relatedPostId
order by
    MaxAnswerCount
desc
limit 1;
```

Ans:

relatedPostId = 406760

maxAnswerCount = 407

10. **(10 points)** Find the month-over-month percentage growth in new posts in the year of 2022. Hint1: You may need to create some CTEs first. Hint2: You need to protect against the "divide by zero" error; divide only when it is safe to do so, otherwise set the corresponding percentage growth to NULL.

```
with MonthlyPosts as (
        select
```

```
        EXTRACT(year from p.creation) as year,
        EXTRACT(month from p.creation) as month,
        COUNT(*) as post_count
    from
        posts p
    where
        EXTRACT(year from p.creation) = 2022
    group by
        year, month
),
MonthlyGrowth as (
    select
        year,
        month,
        post_count,
        coalesce(LAG(post_count) over (order by year, month),0) as prev_month_post_count
    from
        MonthlyPosts
)
select
    year,
    month,
    post_count,
    prev_month_post_count,
    case
        when prev_month_post_count > 0 then ROUND(100.0 * (post_count-
prev_month_post_count)/prev_month_post_count,2)
        else NULL
    end as percent_growth_in_posts
from
    MonthlyGrowth;
```

	year	month	post_count	prev_month_post_count	percent_growth_in_posts
1	2,022	1	276,181	0	[NULL]
2	2,022	2	261,219	276,181	-5.42
3	2,022	3	279,929	261,219	7.16
4	2,022	4	261,687	279,929	-6.52
5	2,022	5	266,492	261,687	1.84
6	2,022	6	257,706	266,492	-3.3
7	2,022	7	257,563	257,706	-0.06
8	2,022	8	264,549	257,563	2.71
9	2,022	9	266,918	264,549	0.9
10	2,022	10	271,929	266,918	1.88
11	2,022	11	280,300	271,929	3.08
12	2,022	12	247,007	280,300	-11.88