

## [ 1-4. 매출 기간인식 코딩 ]

### 1. 데이터베이스에서 데이터 읽어오기

#### 1-1. 프롬프트 작성

mysql DB에 daily\_sales\_raw와 original\_price 테이블이 있어.  
connector 모듈을 이용해서 각각의 테이블의 모든 데이터를 DataFrame 형식으로 가져오는 코드 작성해 줘.

connector 모듈은 아래와 같이 작성해 놓았어.  
# connector.py

```
import mysql.connector

class Connector:
    def __enter__(self):
        self.connection = mysql.connector.connect(
            host="localhost",
            user="root",
            port="3306",
            password="fastcampus1!",
            database="daily_sales"
        )
        self.cursor = self.connection.cursor(dictionary=True)
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.cursor.close()
        self.connection.close()
```

## 1-2. ChatGPT 답변1



아래는 `Connector` 클래스를 사용하여 `daily\_sales\_raw`와 `original\_price` 테이블의 모든 데이터를 `DataFrame` 형식으로 가져오는 Python 코드입니다.

먼저, 데이터를 가져오는 함수 `fetch\_data`를 `Connector` 클래스에 추가합니다.

python

코드 복사

```
# connector.py

import mysql.connector
import pandas as pd

class Connector:
    def __enter__(self):
        self.connection = mysql.connector.connect(
            host="localhost",
            user="root",
            port="3306",
            password="fastcampus1!",
            database="daily_sales"
        )
        self.cursor = self.connection.cursor(dictionary=True)
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.cursor.close()
        self.connection.close()

    def fetch_data(self, query):
        self.cursor.execute(query)
        result = self.cursor.fetchall()
        return pd.DataFrame(result)
```

## ChatGPT 답변 내용을 connector2.py 파일에 반영

```
# connector3.py
import mysql.connector
import pandas as pd

class Connector:
    def __enter__(self):
        self.connection = mysql.connector.connect(
            host="localhost",
            user="root",
            port="3306",
            password="fastcampus1!",
            database="daily_sales"
        )
        self.cursor =
self.connection.cursor(dictionary=True)
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.cursor.close()
        self.connection.close()

    def insert_data(self, query, data):
        self.cursor.executemany(query, data)
        self.connection.commit()

    def fetch_data(self, query):
        self.cursor.execute(query)
        result = self.cursor.fetchall()
        return pd.DataFrame(result)
```

## 1-3. ChatGPT 답변2

다음으로, 이 클래스를 사용하여 데이터를 가져오는 스크립트를 작성합니다.

```
python 📄 코드 복사

from connector import Connector

# daily_sales_raw 테이블의 모든 데이터를 가져오는 쿼리
daily_sales_raw_query = "SELECT * FROM daily_sales_raw"
# original_price 테이블의 모든 데이터를 가져오는 쿼리
original_price_query = "SELECT * FROM original_price"

# 데이터베이스에서 데이터를 DataFrame으로 가져오기
with Connector() as db:
    daily_sales_raw_df = db.fetch_data(daily_sales_raw_query)
    original_price_df = db.fetch_data(original_price_query)

# 가져온 DataFrame 출력 (확인용)
print("daily_sales_raw DataFrame:")
print(daily_sales_raw_df.head())

print("\noriginal_price DataFrame:")
print(original_price_df.head())
```

이 코드는 다음과 같은 기능을 제공합니다:

1. `Connector` 클래스를 사용하여 MySQL 데이터베이스에 연결합니다.
2. `fetch_data` 메서드를 사용하여 쿼리 결과를 `DataFrame` 형식으로 변환하여 반환합니다.
3. 각 테이블의 모든 데이터를 가져오는 쿼리를 실행하고, 결과를 `DataFrame`에 저장합니다.
4. 가져온 데이터를 출력하여 확인합니다.

### 코드 실행하기

```
In [ ]: from connector3 import Connector

# daily_sales_raw 테이블의 모든 데이터를 가져오는 쿼리
daily_sales_raw_query = "SELECT * FROM daily_sales_raw"
# original_price 테이블의 모든 데이터를 가져오는 쿼리
original_price_query = "SELECT * FROM original_price"

# 데이터베이스에서 데이터를 DataFrame으로 가져오기
with Connector() as db:
    daily_sales_raw_df = db.fetch_data(daily_sales_raw_query)
    original_price_df = db.fetch_data(original_price_query)

# 가져온 DataFrame 출력 (확인용)
print("daily_sales_raw DataFrame:")
print(daily_sales_raw_df.head())

print("\noriginal_price DataFrame:")
print(original_price_df.head())
```

```
daily_sales_raw DataFrame:
   id payment_date  stay_days checkin_date checkout_date  original_price
0  1  2024-01-03         1  2024-01-13  2024-01-14         700000
1  2  2024-01-03         2  2024-01-10  2024-01-12        1000000
2  3  2024-01-03         1  2024-01-03  2024-01-04         500000
3  4  2024-01-03         5  2024-01-07  2024-01-12        2700000
4  5  2024-01-03         4  2024-01-09  2024-01-13        2200000
```

```
   payment_amount
0          770000
1          980000
2          540000
3         2280000
4         2210000
```

```
original_price DataFrame:
   date  weekday  original_price
0  2024-01-01         1         500000
1  2024-01-02         2         500000
2  2024-01-03         3         500000
3  2024-01-04         4         500000
4  2024-01-05         5         700000
```

## 2. 매출 기간인식 계산

### 2-1. 주요 라이브러리 Import 및 설정

```
In [ ]: import pandas as pd

# DataFrame 출력 옵션 설정
pd.set_option('display.max_rows', 500)
pd.set_option('display.min_rows', 50)
pd.set_option('display.max_columns', 100)
pd.set_option('display.max_colwidth', 100)
pd.set_option('display.width', 300)
```

## 2-2. 매출인식 코드 작성

### 2-2-1. 프롬프트 작성

위에서 데이터베이스에서 이미 가져온 `daily_sales_raw_df`와 `original_price_df`를 이용해서 아래 작업을 하려고 해.

`daily_sales_raw_df`에 있는 `payment_amount`를 날짜에 맞춰서 기간인식하고 싶어.  
즉, `payment_amount`를 `checkin_date`(포함) 부터 `checkout_date`(미포함) 까지의 날짜에 맞춰서 배분하되 `original_price_df`에 있는 `date`와 `original_price`의 비율에 맞춰서 배분하려고 해.  
최종적으로 만들어 내고 싶은 데이터 테이블은 "date", "payment\_no", "original\_price", "accrued\_revenue" 컬럼으로 구성되어 있고 각 컬럼에 들어갈 값은 아래와 같아.  
"date": 각각의 날짜  
"payment\_no": `daily_sales_raw_df`의 id값  
"original\_price": date에 매칭되는 `original_price` 값  
"accrued\_revenue": `daily_sales_raw_df`의 `payment_amount`을 기간 배분하여 계산한 값.

기간 배분된 `accrued_revenue` 값은 int 값으로 출력하고 싶어.  
그래서 만약 배분되어 계산되는 데이터의 개수가 3개라고 할 경우 최초 2개의 계산결과에는 int를 적용해서 정수로 변경해주고, 마지막 계산결과는 `payment_amount`에서 먼저 계산된 `accrued_revenue`의 합을 빼는 방식으로 계산하고 싶어.

이렇게 최종 데이터 테이블을 DataFrame 형식으로 출력해주는 파이썬 코드 작성해줘.

### 2-2-2. ChatGPT 답변



아래는 `pandas`를 사용하여 `daily\_sales\_raw\_df`와 `original\_price\_df`를 기반으로 설명한 대로 데이터 테이블을 생성하는 Python 코드입니다. 이 코드는 기간에 맞춰 `payment\_amount`를 배분하고, 배분된 금액을 `original\_price`에 맞춰 비율로 분배한 후 최종 결과를 `DataFrame`으로 출력합니다.

python

코드 복사

```
import pandas as pd
import numpy as np

# 데이터프레임을 가져오는 코드 (가정)
# from connector import Connector

# # 데이터베이스에서 데이터를 DataFrame으로 가져오기
# with Connector() as db:
#     daily_sales_raw_df = db.fetch_data("SELECT * FROM daily_sales_raw")
#     original_price_df = db.fetch_data("SELECT * FROM original_price")

# 테스트 데이터 생성 (실제 데이터베이스에서 가져온 데이터 사용 시 주석 처리)
# 데이터 형식: 'id', 'payment_date', 'stay_days', 'checkin_date', 'checkout_date', 'original_price', 'payment_amount'
daily_sales_raw_df = pd.DataFrame({
    'id': [1, 2],
    'payment_date': ['2024-01-01', '2024-01-03'],
    'stay_days': [2, 3],
    'checkin_date': ['2024-01-01', '2024-01-03'],
    'checkout_date': ['2024-01-03', '2024-01-06'],
    'original_price': [200000, 300000],
    'payment_amount': [150000, 250000]
})
```

```
# 데이터 형식: 'date', 'weekday', 'original_price'
original_price_df = pd.DataFrame({
    'date': ['2024-01-01', '2024-01-02', '2024-01-03', '2024-01-04', '2024-01-05'],
    'weekday': [1, 2, 3, 4, 5],
    'original_price': [100000, 100000, 100000, 100000, 100000]
})

# 날짜 컬럼을 datetime 형식으로 변환
daily_sales_raw_df['checkin_date'] = pd.to_datetime(daily_sales_raw_df['checkin_date'])
daily_sales_raw_df['checkout_date'] = pd.to_datetime(daily_sales_raw_df['checkout_date'])
original_price_df['date'] = pd.to_datetime(original_price_df['date'])

# 최종 결과를 담은 DataFrame 생성
result_df = pd.DataFrame(columns=['date', 'payment_no', 'original_price', 'accrued_revenue'])

# 날짜에 맞춰서 데이터 분배 및 매칭
for _, row in daily_sales_raw_df.iterrows():
    checkin_date = row['checkin_date']
    checkout_date = row['checkout_date']
    payment_amount = row['payment_amount']
    stay_days = (checkout_date - checkin_date).days
    payment_no = row['id']

    # 해당 기간의 original_price 가져오기
    period_prices = original_price_df[(original_price_df['date'] >= checkin_date) &
                                       (original_price_df['date'] < checkout_date)]
    period_prices = period_prices.copy()
    period_prices['ratio'] = period_prices['original_price'] / period_prices['original_price']
```

```
# payment_amount를 날짜별로 배분
accrued_revenue = []
for i, (date, ratio) in enumerate(zip(period_prices['date'], period_prices['ratio'])):
    if i < len(period_prices) - 1:
        accrued_revenue.append(int(payment_amount * ratio))
    else:
        accrued_revenue.append(payment_amount - sum(accrued_revenue))

# 결과 DataFrame에 추가
temp_df = pd.DataFrame({
    'date': period_prices['date'],
    'payment_no': payment_no,
    'original_price': period_prices['original_price'],
    'accrued_revenue': accrued_revenue
})

result_df = pd.concat([result_df, temp_df])

# 인덱스 재설정
result_df.reset_index(drop=True, inplace=True)

# 결과 출력
print(result_df)
```



이 코드는 다음과 같은 작업을 수행합니다:

1. ``daily_sales_raw_df``와 ``original_price_df`` 데이터를 가져옵니다.
2. 각 ``payment_amount``를 체크인 날짜부터 체크아웃 날짜까지의 기간에 맞춰 분배합니다.
3. ``original_price`` 비율에 맞춰 ``payment_amount``를 날짜별로 나누어 ``accrued_revenue``를 계산합니다.
4. 최종 결과를 ``DataFrame``으로 생성하고 출력합니다.

이 코드에서는 ``period_prices``에서 ``original_price``의 합계에 대한 비율을 계산하고, 그 비율에 맞춰 ``payment_amount``를 분배합니다. 마지막 날짜의 ``accrued_revenue``는 ``payment_amount``에서 앞서 계산된 ``accrued_revenue``의 합을 뺀 값으로 설정하여 정확한 합계를 맞춥니다.

### 2-2-3. ChatGPT 답변 코드

```
import pandas as pd
import numpy as np

# 데이터프레임을 가져오는 코드 (가정)
# from connector import Connector

# # 데이터베이스에서 데이터를 DataFrame으로 가져오기
# with Connector() as db:
#     daily_sales_raw_df = db.fetch_data("SELECT * FROM
#     daily_sales_raw")
#     original_price_df = db.fetch_data("SELECT * FROM
#     original_price")

# 테스트 데이터 생성 (실제 데이터베이스에서 가져온 데이터 사용 시 주석 처리)
# 데이터 형식: 'id', 'payment_date', 'stay_days',
# 'checkin_date', 'checkout_date', 'original_price',
# 'payment_amount'
daily_sales_raw_df = pd.DataFrame({
    'id': [1, 2],
    'payment_date': ['2024-01-01', '2024-01-03'],
    'stay_days': [2, 3],
    'checkin_date': ['2024-01-01', '2024-01-03'],
    'checkout_date': ['2024-01-03', '2024-01-06'],
    'original_price': [200000, 300000],
    'payment_amount': [150000, 250000]
})

# 데이터 형식: 'date', 'weekday', 'original_price'
original_price_df = pd.DataFrame({
    'date': ['2024-01-01', '2024-01-02', '2024-01-03',
    '2024-01-04', '2024-01-05'],
    'weekday': [1, 2, 3, 4, 5],
    'original_price': [100000, 100000, 100000, 100000,
    100000]
```

```

})

# 날짜 컬럼을 datetime 형식으로 변환
daily_sales_raw_df['checkin_date'] =
pd.to_datetime(daily_sales_raw_df['checkin_date'])
daily_sales_raw_df['checkout_date'] =
pd.to_datetime(daily_sales_raw_df['checkout_date'])
original_price_df['date'] =
pd.to_datetime(original_price_df['date'])

# 최종 결과를 담은 DataFrame 생성
result_df = pd.DataFrame(columns=['date', 'payment_no',
'original_price', 'accrued_revenue'])

# 날짜에 맞춰서 데이터 분배 및 매칭
for _, row in daily_sales_raw_df.iterrows():
    checkin_date = row['checkin_date']
    checkout_date = row['checkout_date']
    payment_amount = row['payment_amount']
    stay_days = (checkout_date - checkin_date).days
    payment_no = row['id']

    # 해당 기간의 original_price 가져오기
    period_prices =
original_price_df[(original_price_df['date'] >=
checkin_date) & (original_price_df['date'] <
checkout_date)]
    period_prices = period_prices.copy()
    period_prices['ratio'] =
period_prices['original_price'] /
period_prices['original_price'].sum()

    # payment_amount를 날짜별로 배분
    accrued_revenue = []
    for i, (date, ratio) in
enumerate(zip(period_prices['date'],
period_prices['ratio'])):
        if i < len(period_prices) - 1:
            accrued_revenue.append(int(payment_amount *
ratio))
        else:
            accrued_revenue.append(payment_amount -
sum(accrued_revenue))

    # 결과 DataFrame에 추가
    temp_df = pd.DataFrame({
        'date': period_prices['date'],
        'payment_no': payment_no,
        'original_price': period_prices['original_price'],
        'accrued_revenue': accrued_revenue
    })

```

```

        result_df = pd.concat([result_df, temp_df])

# 인덱스 재설정
result_df.reset_index(drop=True, inplace=True)

# 결과 출력
print(result_df)

```

## 2-2-4. 작성된 코드 적용

```

In [ ]: import pandas as pd
import numpy as np

# 날짜 컬럼을 datetime 형식으로 변환
daily_sales_raw_df['checkin_date'] = pd.to_datetime(
    daily_sales_raw_df['checkin_date']
)
daily_sales_raw_df['checkout_date'] = pd.to_datetime(
    daily_sales_raw_df['checkout_date']
)
original_price_df['date'] = pd.to_datetime(
    original_price_df['date']
)

# 최종 결과를 담은 DataFrame 생성
result_df = pd.DataFrame(
    columns=['date', 'payment_no', 'original_price', 'accrued_revenue']
)

# 날짜에 맞춰서 데이터 분배 및 매칭
for _, row in daily_sales_raw_df.iterrows():
    checkin_date = row['checkin_date']
    checkout_date = row['checkout_date']
    payment_amount = row['payment_amount']
    stay_days = (checkout_date - checkin_date).days
    payment_no = row['id']

    # 해당 기간의 original_price 가져오기
    period_prices = original_price_df[
        (original_price_df['date'] >= checkin_date)
        & (original_price_df['date'] < checkout_date)
    ]
    period_prices = period_prices.copy()
    period_prices['ratio'] = (
        period_prices['original_price']
        / period_prices['original_price'].sum()
    )

    # payment_amount를 날짜별로 배분
    accrued_revenue = []
    for i, (date, ratio) in enumerate(

```

```

        zip(period_prices['date'], period_prices['ratio'])
    ):
    if i < len(period_prices) - 1:
        accrued_revenue.append(int(payment_amount * ratio))
    else:
        accrued_revenue.append(payment_amount - sum(accrued_revenue))

# 결과 DataFrame에 추가
temp_df = pd.DataFrame({
    'date': period_prices['date'],
    'payment_no': payment_no,
    'original_price': period_prices['original_price'],
    'accrued_revenue': accrued_revenue
})

result_df = pd.concat([result_df, temp_df])

# 인덱스 재설정
result_df.reset_index(drop=True, inplace=True)

# 결과 출력
print(result_df)

```

/var/folders/01/c04y5bhn61l8kzg4jnpj\_8dw0000gp/T/ipykernel\_11914/4292018161.py:57: FutureWarning: The behavior of DataFrame concatenation with empty or all-NA entries is deprecated. In a future version, this will no longer exclude empty or all-NA columns when determining the result dtypes. To retain the old behavior, exclude the relevant entries before the concat operation.

```

result_df = pd.concat([result_df, temp_df])

```

	date	payment_no	original_price	accrued_revenue
0	2024-01-13	1	700000	770000
1	2024-01-10	2	500000	490000
2	2024-01-11	2	500000	490000
3	2024-01-03	3	500000	540000
4	2024-01-07	4	700000	591111
5	2024-01-08	4	500000	422222
6	2024-01-09	4	500000	422222
7	2024-01-10	4	500000	422222
8	2024-01-11	4	500000	422223
9	2024-01-09	5	500000	502272
10	2024-01-10	5	500000	502272
11	2024-01-11	5	500000	502272
12	2024-01-12	5	700000	703184
13	2024-01-04	6	500000	441666
14	2024-01-05	6	700000	618333
15	2024-01-06	6	700000	618333
16	2024-01-07	6	700000	618333
17	2024-01-08	6	500000	441666
18	2024-01-09	6	500000	441669
19	2024-01-06	7	700000	575365
20	2024-01-07	7	700000	575365
21	2024-01-08	7	500000	410975
22	2024-01-09	7	500000	410975

23	2024-01-10	7	500000	410975
24	2024-01-11	7	500000	410975
...	...	...	...	...
8359	2024-05-31	2224	700000	647500
8360	2024-06-03	2225	500000	409259
8361	2024-06-04	2225	500000	409259
8362	2024-06-05	2225	500000	409259
8363	2024-06-06	2225	500000	409259
8364	2024-06-07	2225	700000	572964
8365	2024-06-07	2226	700000	720000
8366	2024-05-30	2227	500000	454166
8367	2024-05-31	2227	700000	635834
8368	2024-05-31	2228	700000	730000
8369	2024-06-01	2229	700000	619705
8370	2024-06-02	2229	700000	619705
8371	2024-06-03	2229	500000	442647
8372	2024-06-04	2229	500000	442647
8373	2024-06-05	2229	500000	442647
8374	2024-06-06	2229	500000	442649
8375	2024-06-01	2230	700000	745000
8376	2024-06-02	2230	700000	745000
8377	2024-06-03	2231	500000	457352
8378	2024-06-04	2231	500000	457352
8379	2024-06-05	2231	500000	457352
8380	2024-06-06	2231	500000	457352
8381	2024-06-07	2231	700000	640294
8382	2024-06-08	2231	700000	640298
8383	2024-06-08	2232	700000	750000

[8384 rows x 4 columns]

## 3. 계산결과 검토

### 3-1. 프롬프트 작성

위에서 계산된 result\_df를 payment\_no를 기준으로 accrued\_revenue 합계를 계산한 후 daily\_sales\_raw의 payment\_amount와 비교하여 계산이 올바르게 되었는지 확인할 수 있는 코드 작성 해줘.

### 3-2. ChatGPT 답변



아래는 `result\_df`에서 `payment\_no`를 기준으로 `accrued\_revenue`의 합계를 계산하고, 이를 `daily\_sales\_raw\_df`의 `payment\_amount`와 비교하여 계산이 올바르게 되었는지 확인하는 코드입니다.

python

코드 복사

```
# accrued_revenue를 payment_no 기준으로 합계 계산
accrued_revenue_sum = result_df.groupby('payment_no')['accrued_revenue'].sum().reset_index()

# daily_sales_raw_df에서 id와 payment_amount 선택
payment_amount_df = daily_sales_raw_df[['id', 'payment_amount']]

# payment_no와 payment_amount를 합치기 위해 컬럼 이름 변경
accrued_revenue_sum.rename(columns={'payment_no': 'id'}, inplace=True)

# payment_amount_df와 accrued_revenue_sum을 병합
comparison_df = pd.merge(payment_amount_df, accrued_revenue_sum, on='id', how='left')

# payment_amount와 accrued_revenue 합계를 비교
comparison_df['is_correct'] = comparison_df['payment_amount'] == comparison_df['accrued_revenue']

# 결과 출력
print(comparison_df)

# 최종 확인
if comparison_df['is_correct'].all():
    print("All accrued_revenue calculations are correct.")
else:
    print("Some accrued_revenue calculations are incorrect.")
```

이 코드는 다음과 같은 작업을 수행합니다:

1. `result\_df`에서 `payment\_no`를 기준으로 `accrued\_revenue`의 합계를 계산하여 `accrued\_revenue\_sum` DataFrame을 생성합니다.
2. `daily\_sales\_raw\_df`에서 `id`와 `payment\_amount` 컬럼을 선택하여 `payment\_amount\_df` DataFrame을 생성합니다.
3. `accrued\_revenue\_sum`의 컬럼 이름을 `id`로 변경하여 `payment\_amount\_df`와 병합할 수 있도록 준비합니다.
4. `payment\_amount\_df`와 `accrued\_revenue\_sum`을 병합하여 `comparison\_df` DataFrame을 생성합니다.
5. `payment\_amount`와 `accrued\_revenue` 합계를 비교하여 결과를 `is\_correct` 컬럼에 저장합니다.
6. 결과를 출력하고, 모든 계산이 올바르게 되었는지 최종 확인합니다.

이 코드를 실행하면 각 `payment\_no`에 대한 `accrued\_revenue`의 합계가 `payment\_amount`와 일치하는지 확인할 수 있습니다. 모든 계산이 올바르게 되었는지 여부를 출력해줍니다.

```
In [ ]: # accrued_revenue를 payment_no 기준으로 합계 계산
accrued_revenue_sum = result_df.groupby('payment_no')[
    'accrued_revenue'
].sum().reset_index()

# daily_sales_raw_df에서 id와 payment_amount 선택
payment_amount_df = daily_sales_raw_df[['id', 'payment_amount']]

# payment_no와 payment_amount를 합치기 위해 컬럼 이름 변경
accrued_revenue_sum.rename(columns={'payment_no': 'id'}, inplace=True)

# payment_amount_df와 accrued_revenue_sum을 병합
comparison_df = pd.merge(
    payment_amount_df, accrued_revenue_sum,
    on='id', how='left'
)

# payment_amount와 accrued_revenue 합계를 비교
comparison_df['is_correct'] = (
    comparison_df['payment_amount'] == comparison_df['accrued_revenue']
)

# 결과 출력
print(comparison_df)

# 최종 확인
if comparison_df['is_correct'].all():
    print("All accrued_revenue calculations are correct.")
else:
    print("Some accrued_revenue calculations are incorrect.")
```

	id	payment_amount	accrued_revenue	is_correct
0	1	770000	770000	True
1	2	980000	980000	True
2	3	540000	540000	True
3	4	2280000	2280000	True
4	5	2210000	2210000	True
5	6	3180000	3180000	True
6	7	3370000	3370000	True
7	8	930000	930000	True
8	9	1550000	1550000	True
9	10	2220000	2220000	True
10	11	2510000	2510000	True
11	12	3430000	3430000	True
12	13	2190000	2190000	True
13	14	780000	780000	True
14	15	3860000	3860000	True
15	16	1460000	1460000	True
16	17	740000	740000	True
17	18	3490000	3490000	True
18	19	1850000	1850000	True
19	20	3110000	3110000	True
20	21	1680000	1680000	True

21	22	3580000	3580000	True
22	23	2990000	2990000	True
23	24	1100000	1100000	True
24	25	4060000	4060000	True
...	...	...	...	...
2207	2208	2550000	2550000	True
2208	2209	1870000	1870000	True
2209	2210	3140000	3140000	True
2210	2211	3660000	3660000	True
2211	2212	1200000	1200000	True
2212	2213	1050000	1050000	True
2213	2214	1960000	1960000	True
2214	2215	1630000	1630000	True
2215	2216	2800000	2800000	True
2216	2217	1000000	1000000	True
2217	2218	2490000	2490000	True
2218	2219	2680000	2680000	True
2219	2220	1350000	1350000	True
2220	2221	3090000	3090000	True
2221	2222	1210000	1210000	True
2222	2223	3960000	3960000	True
2223	2224	1110000	1110000	True
2224	2225	2210000	2210000	True
2225	2226	720000	720000	True
2226	2227	1090000	1090000	True
2227	2228	730000	730000	True
2228	2229	3010000	3010000	True
2229	2230	1490000	1490000	True
2230	2231	3110000	3110000	True
2231	2232	750000	750000	True

[2232 rows x 4 columns]  
All accrued\_revenue calculations are correct.

In [ ]: