

Final Project

Group I Final Project

Lei Wu & Edward Brown

On Premises

On Premises

Comprehensive Shrinkflation Awareness Platform (CSAP)

Group #1

Lei Wu and Edward Brown

Operational Business Context

The recent studies conducted by Retail Insight highlighted an increase in price awareness among consumers, particularly for grocery items, since the onset of the pandemic (source: [Price-conscious consumers hunger for lower food tab | Supermarket News](#)). In response, numerous companies and stores have used a tactic known as “shrinkflation,” subtly reducing a product size while maintaining price, to address the consumers’ heightened price sensitivity. This approach helps many businesses manage escalating production costs and preserve competitive pricing and profit margins, often without clear disclosure to the consumers. Our initiative is creating a digital platform dedicated to tracking shrinkflation for those consumers who value transparency and informed grocery shopping. We primarily focus on the online shopping platform Amazon.

CSAP Database Structure Overview

Our service requires ongoing collection and analysis of product specification changes. Essential entity types and corresponding attributes in our database will include:

1. **Product:** ASIN (key attribute), Name, Weight, Count, Price, and Timestamp.
2. **Parent Company:** ParentCompanyId (key attribute), Name.
3. **Sector:** SectorID (key attribute), SectorName.
4. **Cohort:** CohortID (key attribute).
5. **Variant:** VariantID (key attribute), ChildASIN.
6. **Shrinkflation Metrics:** ShrinkflationMetricID (key attribute), ChangeWeight, ChangeCount, ChangePrice, %ChangeWeight, %ChangeCount, %ChangePrice.
7. **Trends:** TrendID (key attribute), StartDate, EndDate, ShrinkflationIndex.
8. **ShrinkflationEvent:** EventID (key attribute), DetectedDate.
9. **Notification:** NotificationID (key attribute), MessageContent, MessageTime.

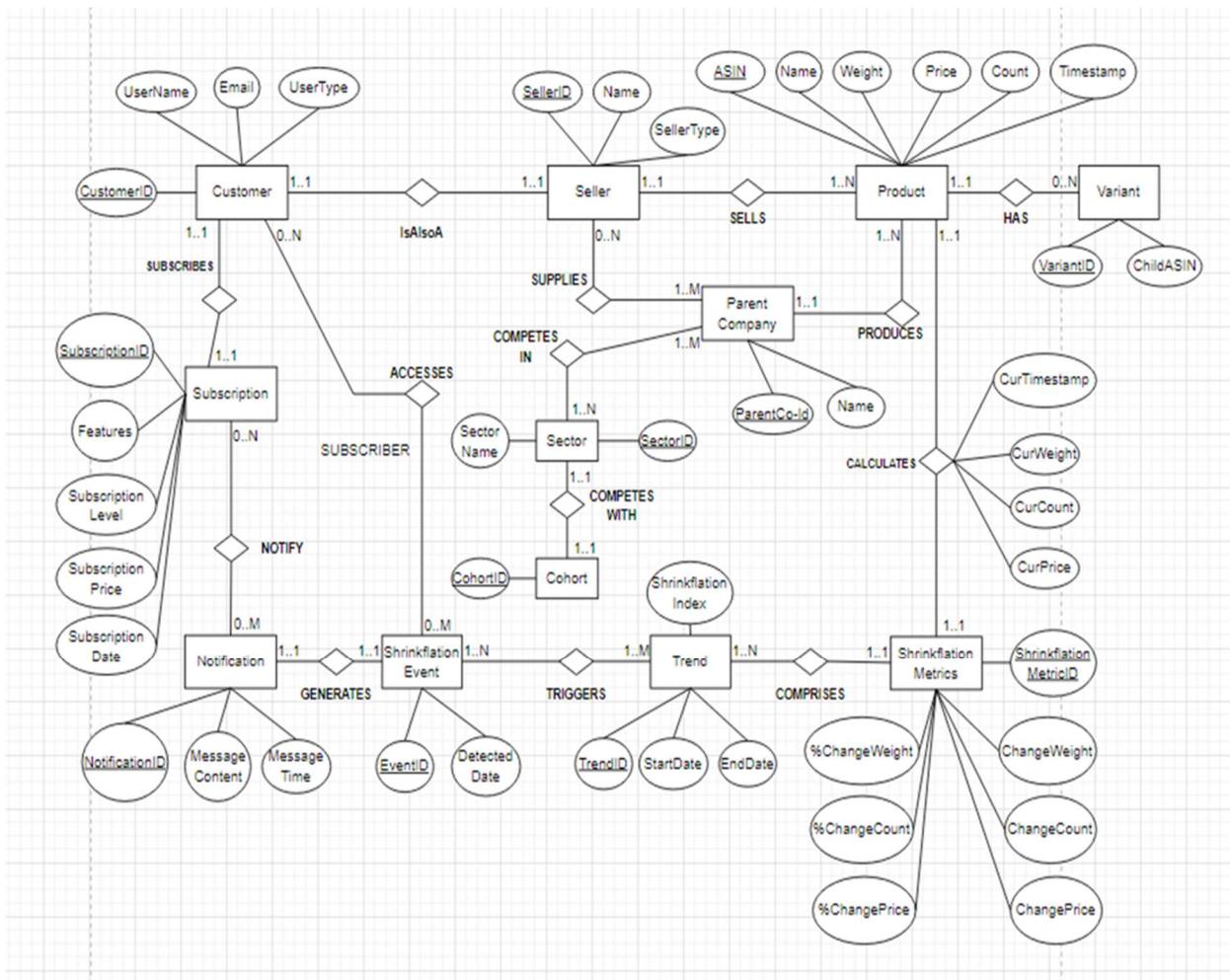
10. **Subscription:** SubscriptionID (key attribute), Features, SubscriptionLevel, SubscriptionPrice.
11. **Seller:** attributes are SellerID (key attribute), Name, SellerType.
12. **Customer:** attributes are CustomerID (key attribute), UserName, Email, UserType.

Data Relationships

1. Seller to Product is one to many relationship, a Seller can sell minimum one and maximum N products.
2. Product to Variant is one to many relationship, each product may not have a variant at all or could have multiple. Conversely each variant has to have a product it is a variant of
3. ParentCompany to Product is one to many relationship, a company produces minimum one to maximum N products.
4. ParentCompany to Seller is many to many relationship, a Seller can be supplied by minimum one and maximum M companies, and a company can supply minimum zero and maximum N sellers.
5. ParentCompany to Sector is many to many relationship, a company can compete in minimum one and maximum N sector, a sector can have minimum one and maximum M companies competing in.
6. Sector to Cohort is one to one relationship, each sector is made up of a single cohort of companies that compete with each other.
7. Product to ShrinkflationMetric is one to one because each product will have one set of shrinkflation metrics and each set of metrics will correspond to one product. The relationship CALCULATES has attributes Current Timestamp, Current Weight, Current Count, and Current Price to calculate corresponding metric.
8. ShrinkflationMetrics to Trends is one to many because each set of metrics could generate multiple trends and any trends will necessarily be tied to a set of metrics.
9. Trends to ShrinkflationEvent is many to many relationships because there can be multiple trends(product count goes down, price goes up, weight goes down all simultaneously).
10. ShrinkflationEvent to Notification is one to one because each shrinkflation event will generate one notification.
11. Customer and Seller might be both, which means a seller can be a customer, and a customer can be a seller, this is a one to one relationship.
12. Customer to Subscription is one to one relationship, a customer can only have one subscription active.

13. Customer to ShrinkflationEvent is many to many relationships as a customer can access zero to M ShrinkflationEvent, and a ShrinkflationEvent can be accessed by zero to N customers.

EER Model Diagram

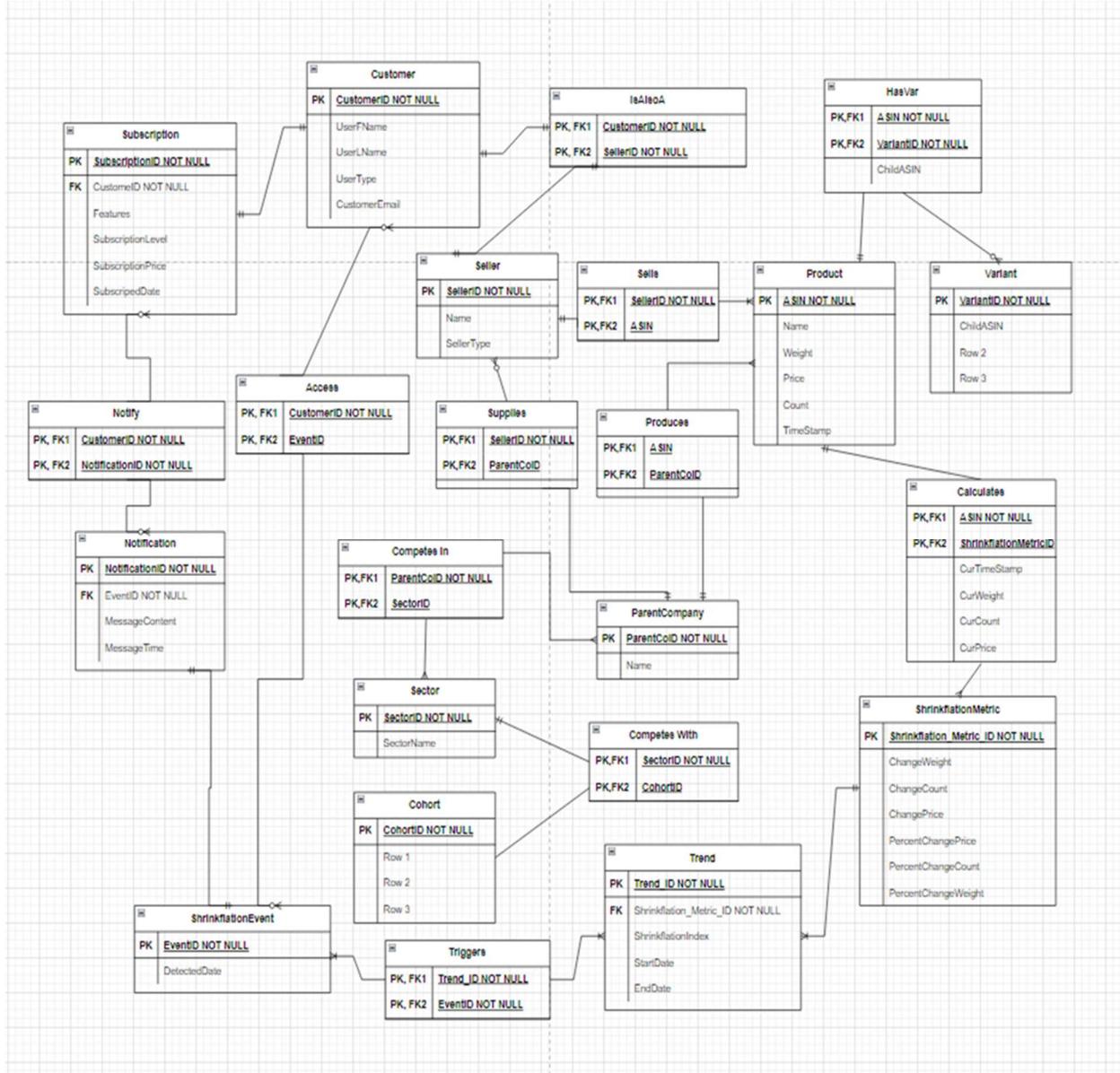


Tables and Attributes of Relational Model

1. Customer: `CustomerID` (PK NOT NULL), `UserName`, `UserType`, `Email`, `Password`, `CreateTime`.
2. IsAlsoA: `CustomerID` (PK, FK NOT NULL), `SellerID` (PK, FK NOT NULL).

3. Seller: SellerID (PK NOT NULL), SellerFName, SellerLName, SellerType.
4. Sells: SellerID (PK, FK NOT NULL), ASIN (PK, FK NOT NULL).
5. Product: ASIN (PK NOT NULL), ProductName, Weight, Price, Count, TimeStamp.
6. HasVar: ASIN (PK, FK NOT NULL), VariantID (PK, FK NOT NULL).
7. Variant: VariantID (PK NOT NULL), ChildASIN.
8. Produces: ASIN (PK, FK NOT NULL), ParentCoID (PK, FK NOT NULL).
9. Supplies: SellerID (PK, FK NOT NULL), ParentCoID (PK, FK NOT NULL).
10. ParentCompany: ParentCoID (PK NOT NULL), CompanyName.
11. CompetesIn: ParentCoID (PK, FK NOT NULL), SectorID (PK, FK NOT NULL).
12. Sector: SectorID (PK NOT NULL).
13. CompetesWith: SectorID (PK, FK NOT NULL), CohortID (PK, FK NOT NULL).
14. Cohort: CohortID (PK NOT NULL).
15. Calculates: ASIN (PK, FK NOT NULL), ShrinkflationMetricID (PK, FK NOT NULL), CurTimeStamp, CurWeight, CurCount, CurPrice.
16. ShrinkflationMetric: ShrinkflationMetricID (PK NOT NULL), ChangeWeight, ChangeCount, ChangePrice, PercentChangePrice, PercentChangeCount, PercentChangeWeight.
17. Trend: TrendID (PK NOT NULL), ShrinkflationMetricID (FK NOT NULL), ShrinkflationIndex, StartDate, EndDate.
18. Triggers: TrendID (PK, FK NOT NULL), EventID (PK, FK NOT NULL).
19. ShrinkflationEvent: EventID (PK NOT NULL), DetectedDate.
20. Notification: NotificationID (PK NOT NULL), EventID (FK NOT NULL), MessageContent, MessageTime.
21. Notify: CustomerID (PK, FK NOT NULL), NotificationID (PK, FK NOT NULL).
22. Subscription: SubscriptionID (PK NOT NULL), CustomerID (FK NOT NULL), Features, SubscriptionLevel, SubscriptionPrice, SubscriptionDate.
23. Access: CustomerID (PK, FK NOT NULL), EventID (PK, FK NOT NULL).

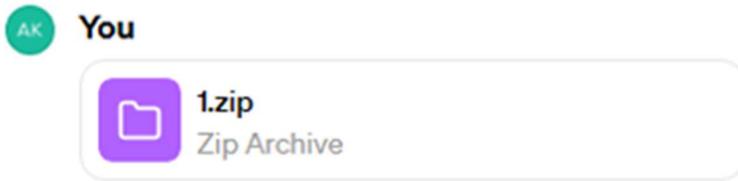
Relational Model Diagram



Data Generation

The data schema contains 23 tables. We will show the details on Milestone 2. All data is synthetic and generated by using ChatGPT 4o through various prompt engineering. Here are the steps:

1. First, we uploaded our schema to ChatGPT for it to understand our data structure and table relations:



2. Then we generate data for the tables without foreign keys:

Parent Company

```
sql Copy code  
  
INSERT INTO parent_company (ParentCoID) VALUES (1);  
INSERT INTO parent_company (ParentCoID) VALUES (2);  
INSERT INTO parent_company (ParentCoID) VALUES (3);  
INSERT INTO parent_company (ParentCoID) VALUES (4);  
INSERT INTO parent_company (ParentCoID) VALUES (5);
```

Sector

```
sql Copy code  
  
INSERT INTO sector (SectorID) VALUES (1);  
INSERT INTO sector (SectorID) VALUES (2);  
INSERT INTO sector (SectorID) VALUES (3);  
INSERT INTO sector (SectorID) VALUES (4);  
INSERT INTO sector (SectorID) VALUES (5);
```

Cohort

```
sql Copy code  
  
INSERT INTO cohort (CohortID) VALUES (1);  
INSERT INTO cohort (CohortID) VALUES (2);  
INSERT INTO cohort (CohortID) VALUES (3);  
INSERT INTO cohort (CohortID) VALUES (4);  
INSERT INTO cohort (CohortID) VALUES (5);
```

3. Copy the SQL code into Query:

```

264 ('033f7f6121501ae9', 'ece5ae58b2d51c16');
265
266
267 -- Inserting data into notification
268 < INSERT INTO notification VALUES
269 ('150f629d91042e19', '280ceb2f47942aeb', 'Notification message 3', '2024-03-24 22:08:35'),
270 ('34c75873ec1bebea', '1e1b3ec9a5e3cffb', 'Notification message 14', '2024-03-24 22:08:35'),
271 ('45f1c30fe48be13b', '938429be5d31dc91', 'Notification message 12', '2024-03-24 22:08:35'),
272 ('4707d8e1511a3d30', '24c1dcde10a01503', 'Notification message 0', '2024-03-24 22:08:35'),
273 ('4bd844aac23efbce', '06ef389db0e8b3e2', 'Notification message 2', '2024-03-24 22:08:35'),
274 ('531c3d49a4b0d25e', '1e1b3ec9a5e3cffb', 'Notification message 6', '2024-03-24 22:08:35'),
275 ('962588e176c0bbdc', '938429be5d31dc91', 'Notification message 1', '2024-03-24 22:08:35'),
276 ('a4abf98f5b376d53', '938429be5d31dc91', 'Notification message 7', '2024-03-24 22:08:35'),
277 ('aa8480503ef4c498', '8efe7273c54bc81d', 'Notification message 4', '2024-03-24 22:08:35'),
278 ('c4aaf5d3704efff9', '6b99e9356b967418', 'Notification message 13', '2024-03-24 22:08:35'),
279 ('c7969086c174e3b7', 'daee4c473df022b3', 'Notification message 9', '2024-03-24 22:08:35'),
280 ('fd9343409404c22a', '8efe7273c54bc81d', 'Notification message 8', '2024-03-24 22:08:35'),
281 ('ef22decdb5ec88fe', '8c47d80f898fa232', 'Notification message 10', '2024-03-24 22:08:35'),
282 ('d4073aa7bbd6f301', 'f4aa19c9a4f9cb11', 'Notification message 11', '2024-03-24 22:08:35'),
283 ('45f1c30fe48be13b', '938429be5d31dc91', 'Notification message 12', '2024-03-24 22:08:35'),
284 ('c4aaf5d3704efff9', '6b99e9356b967418', 'Notification message 13', '2024-03-24 22:08:35'),
285 ('34c75873ec1bebea', '1e1b3ec9a5e3cffb', 'Notification message 14', '2024-03-24 22:08:35')

```

Data Output Messages Notifications

(example)

4. Copy the code from pg4 back to ChatGPT and use it to generate tables with corresponding tables with foreign keys:

```

INSERT INTO notification (NotificationID, EventID, MessageContent) VALUES
('4707d8e1511a3d30', '24c1dcde10a01503', 'Notification message 0'),
('962588e176c0bbdc', '938429be5d31dc91', 'Notification message 1'),
('4bd844aac23efbce', '06ef389db0e8b3e2', 'Notification message 2'),
('150f629d91042e19', '280ceb2f47942aeb', 'Notification message 3'),
('aa8480503ef4c498', '8efe7273c54bc81d', 'Notification message 4'),
('f5083f8acb336ffa', '8efe7273c54bc81d', 'Notification message 5'),
('531c3d49a4b0d25e', '1e1b3ec9a5e3cffb', 'Notification message 6'),
('a4abf98f5b376d53', '938429be5d31dc91', 'Notification message 7'),
('fd9343409404c22a', '8efe7273c54bc81d', 'Notification message 8'),
('c7969086c174e3b7', 'daee4c473df022b3', 'Notification message 9'),
('ef22decdb5ec88fe', '8c47d80f898fa232', 'Notification message 10'),
('d4073aa7bbd6f301', 'f4aa19c9a4f9cb11', 'Notification message 11'),
('45f1c30fe48be13b', '938429be5d31dc91', 'Notification message 12'),
('c4aaf5d3704efff9', '6b99e9356b967418', 'Notification message 13'),
('34c75873ec1bebea', '1e1b3ec9a5e3cffb', 'Notification message 14');

use notification and subscription tables to generate notify table, all 15 rows

```



ChatGPT

Here are the SQL INSERT statements for the `notify` table based on the provided data:

```
sql Copy code  
  
INSERT INTO notify (SubscriptionID, NotificationID) VALUES  
('3204fc25e61b182c', '4707d8e1511a3d30'),  
('53414ff1b618d9d1', '962588e176c0bbdc'),  
('be1f12c2bdc2bc40', '4bd844aac23efbce'),  
('8aa0cae957ef67b6', '150f629d91042e19'),  
('00b088307367c0f9', 'aa8480503ef4c498'),  
('35c39495c2fed54e', 'f5083f8acb336ffa'),  
('dc4f12e7d48e6e3e', '531c3d49a4b0d25e'),  
('e4bea76d5b04bc1a', 'a4abf98f5b376d53'),  
('f99f5f7eff094dd6', 'fd9343409404c22a'),  
('22fb3c6454faac89', 'c7969086c174e3b7'),  
('0cdae352de56aed3', 'ef22decdb5ec88fe'),  
('a63653a8f4585434', 'd4073aa7bbd6f301'),  
('92ae56dfb2659930', '45f1c30fe48be13b'),  
('e08bba5380d9541b', 'c4aaf5d3704efff9'),  
('660812d1c427e21d', '34c75ec1bebea');
```

5. Use the same process to generate all the tables across the schema. Example tables:

The screenshot shows a database interface with two tables displayed:

products

	asin [PK] character varying (10)	productname character varying (45)	weight numeric (10,2)	price numeric (10,2)	count integer	timestamp timestamp without time zone
1	B00000000	Product 0	3.58	16.01	92	2020-10-09 00:00:00
2	B00000001	Product 1	1.08	57.29	15	2022-12-08 00:00:00
3	B00000002	Product 2	2.75	90.82	79	2020-11-02 00:00:00
4	B00000003	Product 3	4.75	91.15	38	2021-08-15 00:00:00
5	B00000004	Product 4	0.57	15.28	22	2021-11-26 00:00:00
6	B00000005	Product 5	0.69	60.60	78	2023-01-19 00:00:00
7	B00000006	Product 6	3.96	92.63	96	2022-01-11 00:00:00
8	B00000007	Product 7	3.86	98.54	45	2022-04-08 00:00:00
9	B00000008	Product 8	2.72	75.36	29	2021-12-27 00:00:00
10	B00000009	Product 9	3.99	44.24	30	2020-08-29 00:00:00
11	B00000010	Product 10	3.30	26.22	82	2020-02-22 00:00:00
12	B00000011	Product 11	2.54	16.64	45	2020-03-20 00:00:00
13	B00000012	Product 12	3.75	88.28	1	2023-12-05 00:00:00
14	B00000013	Product 13	2.56	35.00	73	2021-07-07 00:00:00
15	B00000014	Product 14	3.00	60.00	00	2020-09-21 00:00:00

users

	customerid [PK] character varying (16)	username character varying (16)	email character varying (255)	password character varying (32)	usertype character varying (10)	createtime timestamp without time zone
1	0337f6121501ae9	user3	user3@example.com	819b0643d6b89dc9b579fdfc9094f28e	PaidUser	2024-03-24 21:21:16
2	236fd0fa27a9769b	user11	user11@example.com	1e5c2776cf544e213c3d279c40719643	FreeUser	2024-03-24 21:21:16
3	2f72319caec5d639	user9	user9@example.com	5d69dd95ac183c9643780ed7027d12...	FreeUser	2024-03-24 21:21:16
4	3079e3991f94d1b3	user8	user8@example.com	b25ef06be3b6948c0bc431da46c2c738	FreeUser	2024-03-24 21:21:16
5	55feb130be438e68	user4	user4@example.com	34cc93ece0ba9e3f6f235d4af979b16c	FreeUser	2024-03-24 21:21:16
6	5ce4d191fd14ac85	user2	user2@example.com	6cb75f652a9b52798eb6cf2201057c73	PaidUser	2024-03-24 21:21:16
7	5db2d12c8faad9e4	user0	user0@example.com	305e4f55ce0823e111a46a9d5000cb86c	FreeUser	2024-03-24 21:21:16
8	7fde19c6179474de	user14	user14@example.com	8ee736784ce419bd16554ed5677ff35b	PaidUser	2024-03-24 21:21:16
9	81162e1ef3d93f96	user7	user7@example.com	00cdb7bb942cf6b290ceb97d6aca64a3	PaidUser	2024-03-24 21:21:16
10	9e8486cd435beda	user5	user5@example.com	db0edd04aaa450067edab03ac855d56	PaidUser	2024-03-24 21:21:16
11	a18137515c760761	user12	user12@example.com	c24a542f884e144451f9063b79e7994e	FreeUser	2024-03-24 21:21:16

Overall, through this process, we can generate 10 – 15 rows of data for each table. In our schema, static data and transactional data are distinguished based on their nature and usage within the database:

The following tables represent static data:

sector: Defines sectors or categories in which companies operate. It doesn't change frequently.

cohort: Represents defined groups or categories that likely don't change often.

parent_company: Contains information about parent companies, which are generally stable.

variant: Represents variants of products, which are usually set up once and referenced multiple times.

shrinkflation_metric: Holds predefined metrics related to shrinkflation, which are used for calculations and typically don't change frequently.

product: While products can occasionally change, the core details like ASIN and product name are relatively stable, making it closer to static data.

seller: Contains information about sellers, which are relatively stable over time.

the following tables represent transactional data:

access: Logs customer access to events, which reflect user interactions and are frequently updated.

calculates: Holds data about shrinkflation calculations, which are results of transactions or updates.

competes_in: Represents the competition data between companies and sectors, which can change based on market dynamics.

competes_with: Tracks the relationships between sectors and cohorts, reflecting ongoing competitive positioning.

customer: Stores dynamic information about customers, including when they join and their status, which can change frequently.

has_variant: Links products to their variants, reflecting dynamic product data.

is_also_a: Represents relationships between customers and sellers, which can change based on user actions.

notification: Holds notifications sent to users, which are generated frequently based on events.

notify: Links subscriptions to notifications, reflecting ongoing user interactions.

produces: Represents production relationships between products and companies, which can change based on business operations.

sells: Tracks which sellers are selling which products, reflecting marketplace activities.

subscription: Holds information about customer subscriptions, reflecting changes in user engagement and payments.

supplies: Represents supplier relationships, which can change based on agreements and operations.

trend: Stores trend data related to shrinkflation metrics, reflecting ongoing data changes.

triggers: Links trends to events, dynamically reflecting how trends trigger events.

Future Analysis

Our platform will also be capable of providing data to companies and third-party Amazon resellers. The metrics we provide will be available for companies to analyze and utilize for tactical and strategic purposes. Competitors can analyze peer firm's results and inform their decisions. Companies in the same sectors can evaluate their cohort and determine whether their products should be shrinkflated based on their competitor's results. Economists can gather real-world data on consumer tolerance for shrinkflation. Amazon resellers can track which products and sectors have more resilient consumer demand and can withstand shrinkflation. This information could potentially increase profitability for companies and resellers.

Database Implementation

Tables:

Tables (22)

- > access
- > calculates
- > cohort
- > competesin
- > competeswith
- > customer
- > isalsoa
- > notification
- > notify
- > parentcompany
- > produces
- > product
- > sector
- > seller
- > sells
- > shrinkflationevent
- > shrinkflationmetric
- > subscriber
- > subscription
- > trend
- > triggers
- > variant

Access Table Columns and Constraints:

access

Columns (2)

- customerid
- accesstime

Constraints (1)

- access_customerid_fkey

Access Table Data:

SELECT * FROM public.access

Data Output Messages Notifications

	customerid integer	accesstime timestamp without time zone
1	1	2024-01-02 10:00:00
2	2	2024-01-12 11:30:00
3	3	2024-01-20 12:15:00
4	4	2024-02-02 08:45:00
5	5	2024-02-15 09:10:00
6	6	2024-02-28 14:30:00
7	7	2024-03-12 16:45:00
8	8	2024-03-20 13:00:00
9	9	2024-04-01 11:25:00
10	10	2024-04-10 12:50:00
11	11	2024-04-20 15:40:00
12	12	2024-05-01 10:10:00
13	13	2024-05-12 09:45:00
14	14	2024-05-25 16:00:00
15	15	2024-06-02 10:30:00

Total rows: 20 of 20 Query complete 00:00:00.218

Calculates Table Columns and Constraints:

calculates

↳ Columns (6)

- asin
- shrinkflationmetricid
- curtimestamp
- curweight
- curcount
- curprice

↳ Constraints (3)

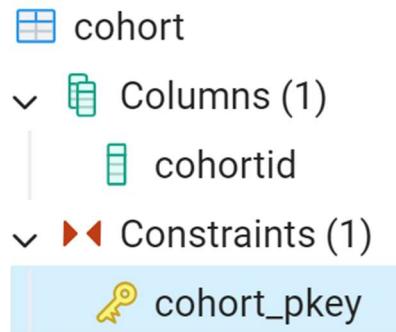
- calculates_asin_fkey
- calculates_pkey
- calculates_shrinkflationmetricid_fkey

Calculates Table Data:

	asin [PK] character varying (20)	shrinkflationmetricid [PK] integer	curtimestamp timestamp without time zone	curweight numeric (10,2)	curcount integer	curprice numeric (10,2)
1	B0012		1 2024-01-15 00:00:00	0.45	190	2.49
2	B0013		2 2024-02-05 00:00:00	0.70	140	5.49
3	B0014		3 2024-02-12 00:00:00	0.72	170	6.59
4	B0015		4 2024-03-01 00:00:00	0.55	230	4.49
5	B0016		5 2024-03-05 00:00:00	0.25	490	1.79
6	B0017		6 2024-04-10 00:00:00	0.35	290	7.49
7	B0018		7 2024-04-15 00:00:00	0.48	110	8.49
8	B0019		8 2024-05-10 00:00:00	0.25	90	9.59
9	B0020		9 2024-05-20 00:00:00	0.15	380	4.49
10	B0021		10 2024-06-05 00:00:00	0.70	170	5.99
11	B0022		11 2024-06-10 00:00:00	0.09	45	48.49
12	B0023		12 2024-06-12 00:00:00	0.13	55	57.49
13	B0024		13 2024-07-01 00:00:00	0.35	200	3.59
14	B0025		14 2024-07-15 00:00:00	0.15	70	9.79
15	B0026		15 2024-08-01 00:00:00	0.04	590	2.19
16	B0027		16 2024-08-05 00:00:00	0.01	690	4.29
17	B0028		17 2024-08-20 00:00:00	0.04	140	5.29
18	B0029		18 2024-09-01 00:00:00	0.25	180	6.19
19	B0030		19 2024-09-15 00:00:00	0.18	280	4.79
20	B0031		20 2024-09-20 00:00:00	0.25	390	2.79

Total rows: 20 of 20 Query complete 00:00:00.138 Ln 1, Col 1

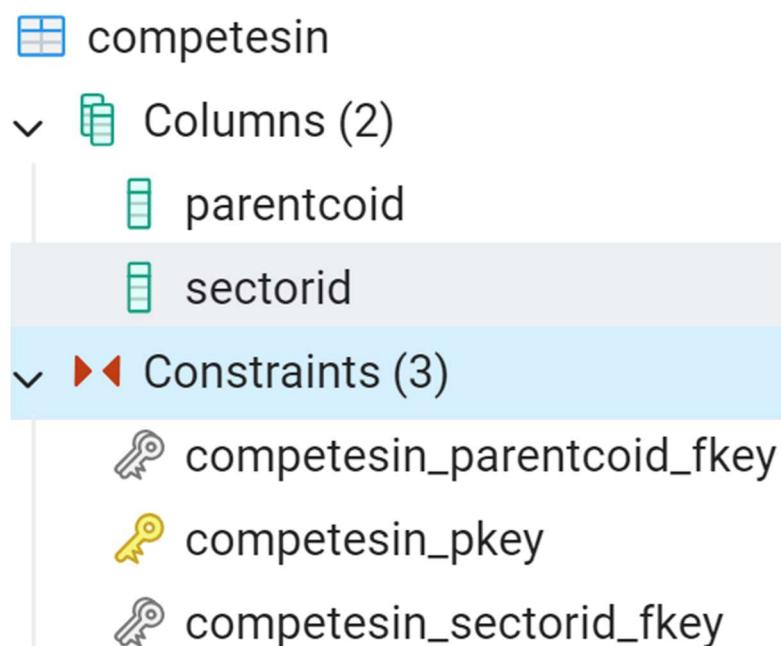
Cohort Table Columns and Constraints:



Cohort Table Data:

	cohortid [PK] integer
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20

Competesin Columns and Constraints:



Competesin Data:

	parentcoid [PK] integer		sectorid [PK] integer	
1		1		1
2		2		1
3		3		1
4		4		1
5		5		5
6		6		4
7		7		2
8		8		2
9		9		2
10		10		7
11		11		3
12		12		3
13		13		2
14		14		5
15		15		2
16		16		5
17		17		5
18		18		5
19		19		5
20		20		5

Competes with Columns and Constraints:

competeswith

⌄ Columns (2)

 sectorid

 cohortid

⌄ Constraints (3)

 competeswith_cohortid_fkey

 competeswith_pkey

 competeswith_sectorid_fkey

Competeswith Data:

	sectorid [PK] integer	cohortid [PK] integer
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
11	11	11
12	12	12
13	13	13
14	14	14
15	15	15
16	16	16
17	17	17
18	18	18
19	19	19
20	20	20

Customer Table Columns and Constraints:

customer

⌄ Columns (5)

-  customerid
-  username
-  userlname
-  usertype
-  customeremail

⌄ Constraints (1)

-  customer_pkey

Customer Table Data:

	customerid [PK] integer	username character varying (100)	username character varying (100)	usertype character varying (50)	customeremail character varying (100)
1	1	John	Doe	Standard	john.doe@example.com
2	2	Jane	Smith	Premium	jane.smith@example.com
3	3	Robert	Brown	Standard	robert.brown@example.com
4	4	Emily	Davis	Premium	emily.davis@example.com
5	5	Michael	Wilson	Standard	michael.wilson@example.com
6	6	Sarah	Johnson	Premium	sarah.johnson@example.com
7	7	William	Miller	Standard	william.miller@example.com
8	8	Olivia	Jones	Premium	olivia.jones@example.com
9	9	James	Garcia	Standard	james.garcia@example.com
10	10	Sophia	Martinez	Premium	sophia.martinez@example.com
11	11	Liam	Rodriguez	Standard	liam.rodriguez@example.com
12	12	Ava	Hernandez	Premium	ava.hernandez@example.com
13	13	Jacob	Lopez	Standard	jacob.lopez@example.com
14	14	Isabella	Gonzalez	Premium	isabella.gonzalez@example.com
15	15	Mason	Perez	Standard	mason.perez@example.com
16	16	Mia	Hall	Premium	mia.hall@example.com
17	17	Benjamin	Lee	Standard	benjamin.lee@example.com
18	18	Amelia	Young	Premium	amelia.young@example.com
19	19	Elijah	Harris	Standard	elijah.harris@example.com
20	20	Charlotte	King	Premium	charlotte.king@example.com

Isalsoa Table Columns and Constraints:

isalsoa

⌄ Columns (2)

-  customerid
-  sellerid

⌄ Constraints (3)

-  isalsoa_customerid_fkey
-  isalsoa_pkey
-  isalsoa_sellerid_fkey

Isalsoa Table Data:

	customerid [PK] integer	sellerid [PK] integer
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
11	11	11
12	12	12
13	13	13
14	14	14
15	15	15
16	16	16
17	17	17
18	18	18
19	19	19
20	20	20

Notification Table Columns and Constraints:

notification

Columns (3)

notificationid

messagecontent

messagetime

Constraints (1)

notification_pkey

Notification Table Data:

	notificationid [PK] integer	messagecontent character varying (255)	messagetime timestamp without time zone
1	1	Price drop on your favorite product!	2024-01-05 10:30:00
2	2	New product added to your subscription!	2024-01-12 11:00:00
3	3	Stock running low on frequently purchased item.	2024-01-25 15:45:00
4	4	Shrinkflation alert on selected products.	2024-02-05 09:20:00
5	5	Price update for a product in your watchlist.	2024-02-18 13:15:00
6	6	Product restocked! Available now.	2024-03-01 16:00:00
7	7	Limited-time offer on premium subscription.	2024-03-10 08:30:00
8	8	Shrinkflation detected in a product.	2024-03-20 14:10:00
9	9	New shrinkflation trend detected.	2024-04-01 12:05:00
10	10	Reminder: Subscription renewal in 3 days.	2024-04-15 11:50:00
11	11	Product on sale: Don't miss out!	2024-05-05 10:00:00
12	12	Limited edition variant now available.	2024-05-18 17:00:00
13	13	New update in shrinkflation index.	2024-06-01 09:15:00
14	14	Shrinkflation alert for popular item.	2024-06-12 11:45:00
15	15	Your subscription is about to expire.	2024-06-25 16:30:00
16	16	Shrinkflation trend analysis available.	2024-07-10 08:45:00
17	17	New notification regarding product metrics.	2024-07-20 14:00:00
18	18	Price comparison now available for your saved item...	2024-08-01 09:00:00
19	19	Reminder: Check the latest shrinkflation trend.	2024-08-10 12:30:00
20	20	Shrinkflation event detected in product range.	2024-08-20 17:30:00

Notify Table Columns and Constraints:

grid notify

columns (2)

customerid

notificationid

►◀ Constraints (3)

key notify_customerid_fkey

key notify_notificationid_fkey

key notify_pkey

Notify Table Data:

	customerid [PK] integer	notificationid [PK] integer
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
11	11	11
12	12	12
13	13	13
14	14	14
15	15	15
16	16	16
17	17	17
18	18	18
19	19	19
20	20	20

Parentcompany Table Columns and Constraints:

parentcompany

Columns (2)

 parentcoid

 companyname

Constraints (1)

 parentcompany_pkey

Parentcompany Data:

	parentcoid [PK] integer	companyname character varying (100)
1	1	Procter & Gamble
2	2	Unilever
3	3	Johnson & Johnson
4	4	Colgate-Palmolive
5	5	Kimberly-Clark
6	6	L'Oréal
7	7	Nestlé
8	8	PepsiCo
9	9	Coca-Cola
10	10	Mars Inc.
11	11	Mondelez International
12	12	Kelloggs
13	13	General Mills
14	14	Reckitt Benckiser
15	15	Danone
16	16	Henkel
17	17	SC Johnson
18	18	Pfizer
19	19	GSK
20	20	Sanofi

Produces Table Columns and Constraints:

produces

⌄ Columns (2)

 asin

 parentcoid

⌄ Constraints (3)

 produces_asin_fkey

 produces_parentcoid_fkey

 produces_pkey

Produces Table Data:

	asin [PK] character varying (20) 	parentcoid [PK] integer 
1	B0012	1
2	B0013	2
3	B0014	3
4	B0015	4
5	B0016	5
6	B0017	6
7	B0018	7
8	B0019	8
9	B0020	9
10	B0021	10
11	B0022	11
12	B0023	12
13	B0024	13
14	B0025	14
15	B0026	15
16	B0027	16
17	B0028	17
18	B0029	18
19	B0030	19
20	B0031	20

Product Table Columns and Constraints:

product

Columns (6)

- asin
- productname
- weight
- price
- count
- timestamp

Constraints (1)

product_pkey

Product Table Data:

	asin [PK] character varying (20)	productname character varying (100)	weight numeric (10,2)	price numeric (10,2)	count integer	timestamp timestamp without time zone
1	B0012	Toothpaste	0.50	2.99	200	2024-01-15 00:00:00
2	B0013	Shampoo	0.75	5.99	150	2024-02-05 00:00:00
3	B0014	Conditioner	0.75	6.99	180	2024-02-12 00:00:00
4	B0015	Body Wash	0.60	4.99	250	2024-03-01 00:00:00
5	B0016	Hand Soap	0.30	1.99	500	2024-03-05 00:00:00
6	B0017	Facial Cleanser	0.40	7.99	300	2024-04-10 00:00:00
7	B0018	Lotion	0.50	8.99	120	2024-04-15 00:00:00
8	B0019	Sunscreen	0.30	9.99	100	2024-05-10 00:00:00
9	B0020	Deodorant	0.20	4.99	400	2024-05-20 00:00:00
10	B0021	Mouthwash	0.75	6.49	180	2024-06-05 00:00:00
11	B0022	Perfume	0.10	49.99	50	2024-06-10 00:00:00
12	B0023	Cologne	0.15	59.99	60	2024-06-12 00:00:00
13	B0024	Shaving Cream	0.40	3.99	220	2024-07-01 00:00:00
14	B0025	Razor Blades	0.20	10.99	80	2024-07-15 00:00:00
15	B0026	Lip Balm	0.05	2.49	600	2024-08-01 00:00:00
16	B0027	Eye Drops	0.01	4.49	700	2024-08-05 00:00:00
17	B0028	Nail Polish	0.05	5.49	150	2024-08-20 00:00:00
18	B0029	Makeup Remover	0.30	6.49	190	2024-09-01 00:00:00
19	B0030	Hair Gel	0.20	4.99	300	2024-09-15 00:00:00
20	B0031	Shaving Foam	0.30	2.99	400	2024-09-20 00:00:00

Sector Columns and Constraints:

sector

⌄ Columns (2)

-  sectorid
-  sectorname

⌄ Constraints (1)

-  sector_pkey

Sector Table Data:

	sectorid [PK] integer	sectorname character varying (100)
1	1	Personal Care
2	2	Food & Beverages
3	3	Healthcare
4	4	Beauty & Cosmetics
5	5	Household Goods
6	6	Pharmaceuticals
7	7	Confectionery
8	8	Beverages
9	9	Snacks
10	10	Frozen Foods
11	11	Dairy Products
12	12	Bakery Products
13	13	Cereals
14	14	Cleaning Products
15	15	Baby Care
16	16	Pet Food
17	17	Oral Care
18	18	Skincare
19	19	Fragrance
20	20	Haircare

Seller Table Columns and Constraints:

seller

⌄ Columns (4)

-  sellerid
-  sellerfname
-  sellerlname
-  sellertype

⌄ Constraints (1)

 seller_pkey

Seller Table Data:

	sellerid [PK] integer	sellerfname character varying (100)	sellerlname character varying (100)	sellertype character varying (50)
1	1	David	Taylor	Wholesale
2	2	Lucas	Anderson	Retail
3	3	Henry	Thomas	Wholesale
4	4	Ella	Moore	Retail
5	5	Alexander	Jackson	Wholesale
6	6	Grace	Martin	Retail
7	7	Ethan	Thompson	Wholesale
8	8	Aiden	White	Retail
9	9	Logan	Harris	Wholesale
10	10	Chloe	Garcia	Retail
11	11	Samuel	Martinez	Wholesale
12	12	Emily	Clark	Retail
13	13	Joseph	Rodriguez	Wholesale
14	14	Sophie	Lewis	Retail
15	15	Daniel	Lee	Wholesale
16	16	Hannah	Walker	Retail
17	17	Matthew	Allen	Wholesale
18	18	Lily	Young	Retail
19	19	Sebastian	King	Wholesale
20	20	Zoe	Wright	Retail

Sells Table Columns and Constraints:

sells

✓ Columns (2)

 sellerid

 asin

✓ Constraints (3)

 sells_asin_fkey

 sells_pkey

 sells_sellerid_fkey

Sells Table Data:

	sellerid [PK] integer	asin [PK] character varying (20)
1	1	B0012
2	2	B0013
3	3	B0014
4	4	B0015
5	5	B0016
6	6	B0017
7	7	B0018
8	8	B0019
9	9	B0020
10	10	B0021
11	11	B0022
12	12	B0023
13	13	B0024
14	14	B0025
15	15	B0026
16	16	B0027
17	17	B0028
18	18	B0029
19	19	B0030
20	20	B0031

Shrinkflationevent Table Columns and Constraints:

grid shrinkflationevent

dropdown Columns (2)

eventid

detecteddate

dropdown Constraints (1)

key shrinkflationevent_pkey

Shrinkflationevent Table Data:

	eventid [PK] integer	detecteddate date
1	1	2024-01-01
2	2	2024-02-01
3	3	2024-03-01
4	4	2024-04-01
5	5	2024-05-01
6	6	2024-06-01
7	7	2024-07-01
8	8	2024-08-01
9	9	2024-09-01
10	10	2024-10-01
11	11	2024-11-01
12	12	2024-12-01
13	13	2025-01-01
14	14	2025-02-01
15	15	2025-03-01
16	16	2025-04-01
17	17	2025-05-01
18	18	2025-06-01
19	19	2025-07-01
20	20	2025-08-01

Shrinkflationmetric Table Columns and Constraints:

shrinkflationmetric

Columns (7)

- shrinkflationmetricid
- changeweight
- changecount
- changeprice
- percentchangeweight
- percentchangecount
- percentchangeprice

Constraints (1)

shrinkflationmetric_pkey

Shrinkflationmetric Table Data:

	shrinkflationmetricid [PK] integer	changeweight numeric (10,2)	changecount integer	changeprice numeric (10,2)	percentchangeweight numeric (5,2)	percentchangecount numeric (5,2)	percentchangeprice numeric (5,2)
1	1	0.05	10	0.50	5.00	10.00	5.00
2	2	0.04	8	0.40	4.00	8.00	4.00
3	3	0.03	6	0.30	3.00	6.00	3.00
4	4	0.06	12	0.60	6.00	12.00	6.00
5	5	0.05	10	0.50	5.00	10.00	5.00
6	6	0.07	14	0.70	7.00	14.00	7.00
7	7	0.04	8	0.40	4.00	8.00	4.00
8	8	0.05	10	0.50	5.00	10.00	5.00
9	9	0.03	6	0.30	3.00	6.00	3.00
10	10	0.02	4	0.20	2.00	4.00	2.00
11	11	0.06	12	0.60	6.00	12.00	6.00
12	12	0.05	10	0.50	5.00	10.00	5.00
13	13	0.03	6	0.30	3.00	6.00	3.00
14	14	0.02	4	0.20	2.00	4.00	2.00
15	15	0.07	14	0.70	7.00	14.00	7.00
16	16	0.06	12	0.60	6.00	12.00	6.00
17	17	0.05	10	0.50	5.00	10.00	5.00
18	18	0.04	8	0.40	4.00	8.00	4.00
19	19	0.03	6	0.30	3.00	6.00	3.00
20	20	0.02	4	0.20	2.00	4.00	2.00

Subscriber Table Columns and Constraints:

subscriber

⌄ Columns (2)

 customerid

 subscriptionid

⌄ Constraints (3)

 subscriber_customerid_fkey

 subscriber_pkey

 subscriber_subscriptionid_fkey

Subscriber Table Data:

	customerid [PK] integer	subscriptionid [PK] integer
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
11	11	11
12	12	12
13	13	13
14	14	14
15	15	15
16	16	16
17	17	17
18	18	18
19	19	19
20	20	20

Subscription Table Columns and Constraints:

subscription

Columns (6)

- subscriptionid
- customerid
- features
- subscriptionlevel
- subscriptionprice
- subscriptiondate

Constraints (2)

subscription_customerid_fkey

subscription_pkey

Subscription Table Data:

	subscriptionid [PK] integer	customerid integer	features character varying (255)	subscriptionlevel character varying (50)	subscriptionprice numeric (10,2)	subscriptiondate date
1		1	1 Basic Access	Basic	9.99	2024-01-01
2		2	2 Advanced Features	Premium	19.99	2024-01-10
3		3	3 Basic Access	Basic	9.99	2024-01-20
4		4	4 Premium Features	Premium	29.99	2024-02-01
5		5	5 Basic Access	Basic	9.99	2024-02-15
6		6	6 Advanced Features	Premium	19.99	2024-02-28
7		7	7 Basic Access	Basic	9.99	2024-03-10
8		8	8 Premium Features	Premium	29.99	2024-03-20
9		9	9 Basic Access	Basic	9.99	2024-04-01
10		10	10 Advanced Features	Premium	19.99	2024-04-10
11		11	11 Basic Access	Basic	9.99	2024-04-20
12		12	12 Premium Features	Premium	29.99	2024-05-01
13		13	13 Basic Access	Basic	9.99	2024-05-10
14		14	14 Advanced Features	Premium	19.99	2024-05-20
15		15	15 Basic Access	Basic	9.99	2024-06-01
16		16	16 Premium Features	Premium	29.99	2024-06-10
17		17	17 Basic Access	Basic	9.99	2024-06-20
18		18	18 Advanced Features	Premium	19.99	2024-07-01
19		19	19 Basic Access	Basic	9.99	2024-07-10
20		20	20 Premium Features	Premium	29.99	2024-07-20

Trend Table Columns and Constraints:

grid	trend
▼	Columns (5)
grid	trendid
grid	shrinkflationmetricid
grid	shrinkflationindex
grid	startdate
grid	enddate
▼ ►	Constraints (2)
key	trend_pkey
key	trend_shrinkflationmetricid_fkey

Trend Table Data:

	trendid [PK] integer	shrinkflationmetricid integer	shrinkflationindex numeric (5,2)	startdate date	enddate date
1	1	1	5.00	2022-01-01	2022-06-30
2	2	2	4.50	2022-07-01	2022-12-31
3	3	3	4.00	2023-01-01	2023-06-30
4	4	4	3.50	2023-07-01	2023-12-31
5	5	5	3.00	2024-01-01	2024-06-30
6	6	6	2.50	2024-07-01	2024-12-31
7	7	7	2.00	2025-01-01	2025-06-30
8	8	8	1.50	2025-07-01	2025-12-31
9	9	9	1.00	2026-01-01	2026-06-30
10	10	10	0.50	2026-07-01	2026-12-31
11	11	11	5.50	2027-01-01	2027-06-30
12	12	12	6.00	2027-07-01	2027-12-31
13	13	13	6.50	2028-01-01	2028-06-30
14	14	14	7.00	2028-07-01	2028-12-31
15	15	15	7.50	2029-01-01	2029-06-30
16	16	16	8.00	2029-07-01	2029-12-31
17	17	17	8.50	2030-01-01	2030-06-30
18	18	18	9.00	2030-07-01	2030-12-31
19	19	19	9.50	2031-01-01	2031-06-30
20	20	20	10.00	2031-07-01	2031-12-31

Triggers Table Columns and Constraints:

triggers

⌄ Columns (2)

 trendid

 eventid

⌄ Constraints (3)

 triggers_eventid_fkey

 triggers_pkey

 triggers_trendid_fkey

Triggers Table Data:

	trendid [PK] integer	eventid [PK] integer
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
11	11	11
12	12	12
13	13	13
14	14	14
15	15	15
16	16	16
17	17	17
18	18	18
19	19	19
20	20	20

Variant Table Columns and Constraints:

variant

Columns (3)

-  variantid
-  childasin
-  asin

Constraints (2)

 variant_asin_fkey

 variant_pkey

Variant Table Data:

	variantid [PK] integer	childasin character varying (20)	asin character varying (20)
1	1	V001	B0012
2	2	V002	B0013
3	3	V003	B0014
4	4	V004	B0015
5	5	V005	B0016
6	6	V006	B0017
7	7	V007	B0018
8	8	V008	B0019
9	9	V009	B0020
10	10	V010	B0021
11	11	V011	B0022
12	12	V012	B0023
13	13	V013	B0024
14	14	V014	B0025
15	15	V015	B0026
16	16	V016	B0027
17	17	V017	B0028
18	18	V018	B0029
19	19	V019	B0030
20	20	V020	B0031

Identify Elements for Future Data Warehouse Implementation

The following dimensions, hierarchies and measures are potentially implemented for the Data Warehouse or analytical application:

Dimensions:

Product Dimension

Attributes: ASIN, ProductName, Weight, Price, Count

Hierarchy:

Product → Parent Company → Sector

Customer Dimension

Attributes: CustomerID, Username, Email, UserType, CreateTime

Hierarchy:

Customer → Subscription Level

Shrinkflation Event Dimension

Attributes: EventID, DetectedDate

Hierarchy:

Shrinkflation Event → Shrinkflation Trend

Seller Dimension

Attributes: SellerID, SellerFName, SellerLName, SellerType

Hierarchy:

Seller → Parent Company → Sector

Time Dimension (Derived from various timestamp/date fields)

Attributes: Year, Quarter, Month, Week, Day

Hierarchy:

Year → Quarter → Month → Day

Measures:

Total Product Count: Sum of product counts

Total Product Weight: Sum of product weight

Total Product Sales: Sum of product prices

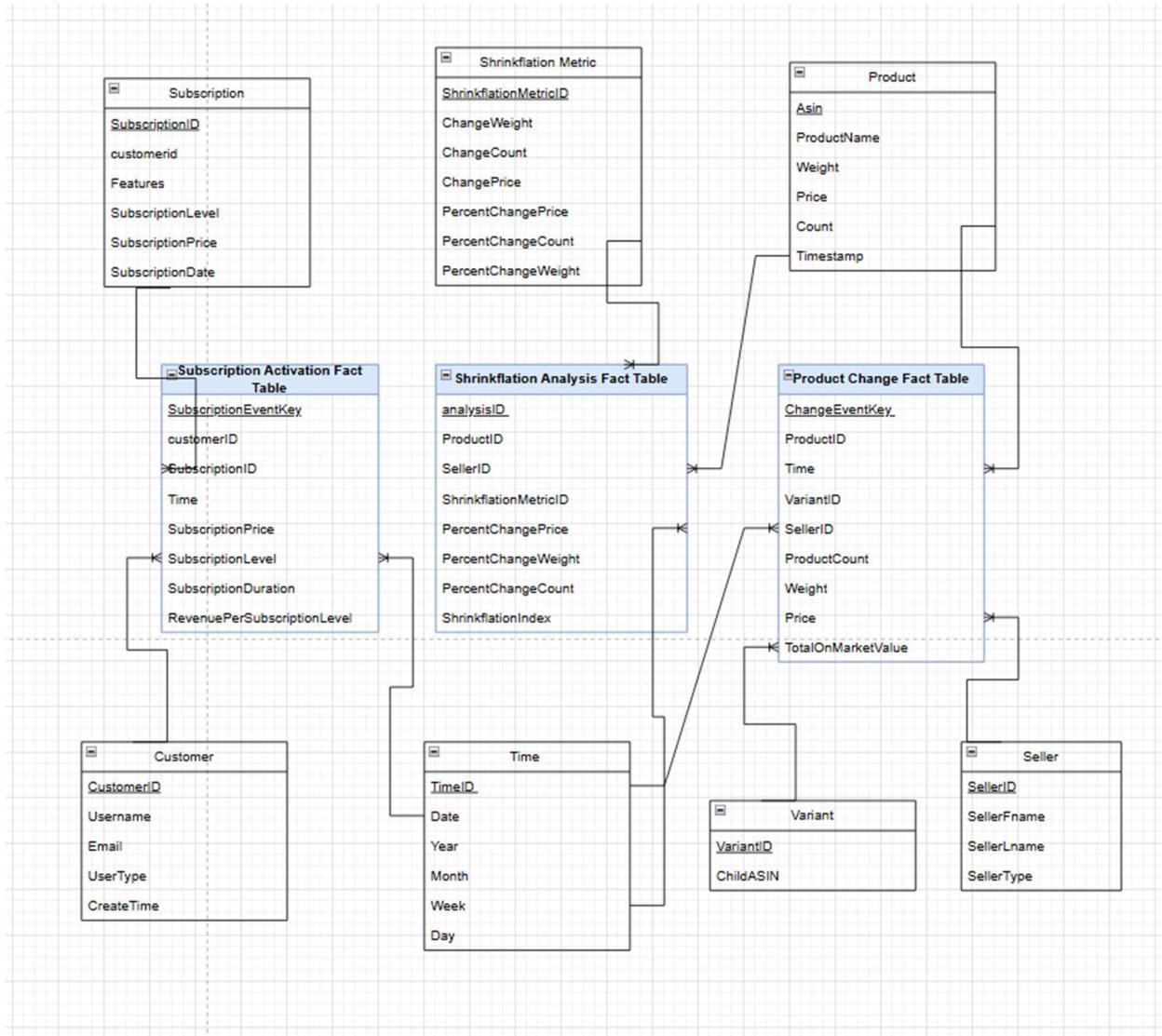
Shrinkflation Percentage Change: Derived from the PercentChangePrice, PercentChangeWeight, and PercentChangeCount in shrinkflation_metric

Number of Shrinkflation Events: Count of shrinkflation events

Total Subscription Revenue: Sum of SubscriptionPrice from subscription

Average Shrinkflation Index: Average of ShrinkflationIndex from trend

Multidim Diagram:



Logical DW Model:

1. Product Change Fact Table

Primary Key: `changeeventid`

Foreign Keys:

`productkey` (FK to `product_dimension`)

`variantkey` (FK to `variant_dimension`)

`sellerkey` (FK to `seller_dimension`)

timekey (FK to time_dimension)

Measures:

productcount

weight

price

totalonmarketvalue

Cardinalities:

Product Dimension: 1

(One product can be associated with multiple product change events)

Variant Dimension: 1

(One variant can be associated with multiple product change events)

Seller Dimension: 1

(One seller can be associated with multiple product change events)

Time Dimension: 1

(One time entry can be associated with multiple product change events)

2. Shrinkflation Analysis Fact Table

Primary Key: analysisid

Foreign Keys:

productkey (FK to product_dimension)

sellerkey (FK to seller_dimension)

shrinkflationmetrickey (FK to shrinkflation_metric_dimension)

Measures:

percentchangeprice

percentchangeweight

percentchangecount

shrinkflationindex

Cardinalities:

Product Dimension: 1

(One product can be associated with multiple shrinkflation analysis events)

Seller Dimension: 1

(One seller can be associated with multiple shrinkflation analysis events)

Shrinkflation Metric Dimension: 1

(One shrinkflation metric can be associated with multiple shrinkflation analysis events)

3. Subscription Activation Fact Table

Primary Key: subscriptioneventkey

Foreign Keys:

customerkey (FK to customer_dimension)

subscriptionkey (FK to subscription_dimension)

timekey (FK to time_dimension)

Measures:

subscriptionprice

subscriptionlevel

subscriptionduration

revenuepersubscriptionlevel

Cardinalities:

Customer Dimension: 1

(One customer can activate multiple subscriptions)

Subscription Dimension: 1

(One subscription can be linked to multiple activation events)

Time Dimension: 1

(One time entry can be linked to multiple subscription activation events)

4. Customer Dimension

Primary Key: customerkey

Attributes:

customerid

username

email

usertype

createtime

Cardinalities:

To Subscription Activation Fact Table: 1

(One customer to many subscription activations)

5. Product Dimension

Primary Key: productkey

Attributes:

asin

productname

weight

price

count

timestamp

Cardinalities:

To Product Change Fact Table: 1

(One product to many product change events)

To Shrinkflation Analysis Fact Table: 1

(One product to many shrinkflation analysis records)

6. Seller Dimension

Primary Key: sellerkey

Attributes:

sellerid

sellerfname

sellernname

sellertype

Cardinalities:

To Product Change Fact Table: 1

(One seller to many product change events)

To Shrinkflation Analysis Fact Table: 1

(One seller to many shrinkflation analysis events)

7. Shrinkflation Metric Dimension

Primary Key: shrinkflationmetrickey

Attributes:

shrinkflationmetricid

changeweight

changeccount

changeprice

percentchangeprice

percentchangecount

percentchangeweight

Cardinalities:

To Shrinkflation Analysis Fact Table: 1

(One shrinkflation metric to many shrinkflation analysis events)

8. Subscription Dimension

Primary Key: subscriptionkey

Attributes:

subscriptionid

customerid

features

subscriptionlevel

subscriptionprice

subscriptiondate

Cardinalities:

To Subscription Activation Fact Table: 1

(One subscription type to many subscription activation events)

9. Time Dimension (flat)

Primary Key: timekey

Attributes:

timeid

day

month

year

Cardinalities:

To Product Change Fact Table: 1

(One time entry to many product change events)

To Subscription Activation Fact Table: 1

(One time entry to many subscription activation events)

10. Variant Dimension

Primary Key: variantkey

Attributes:

variantid

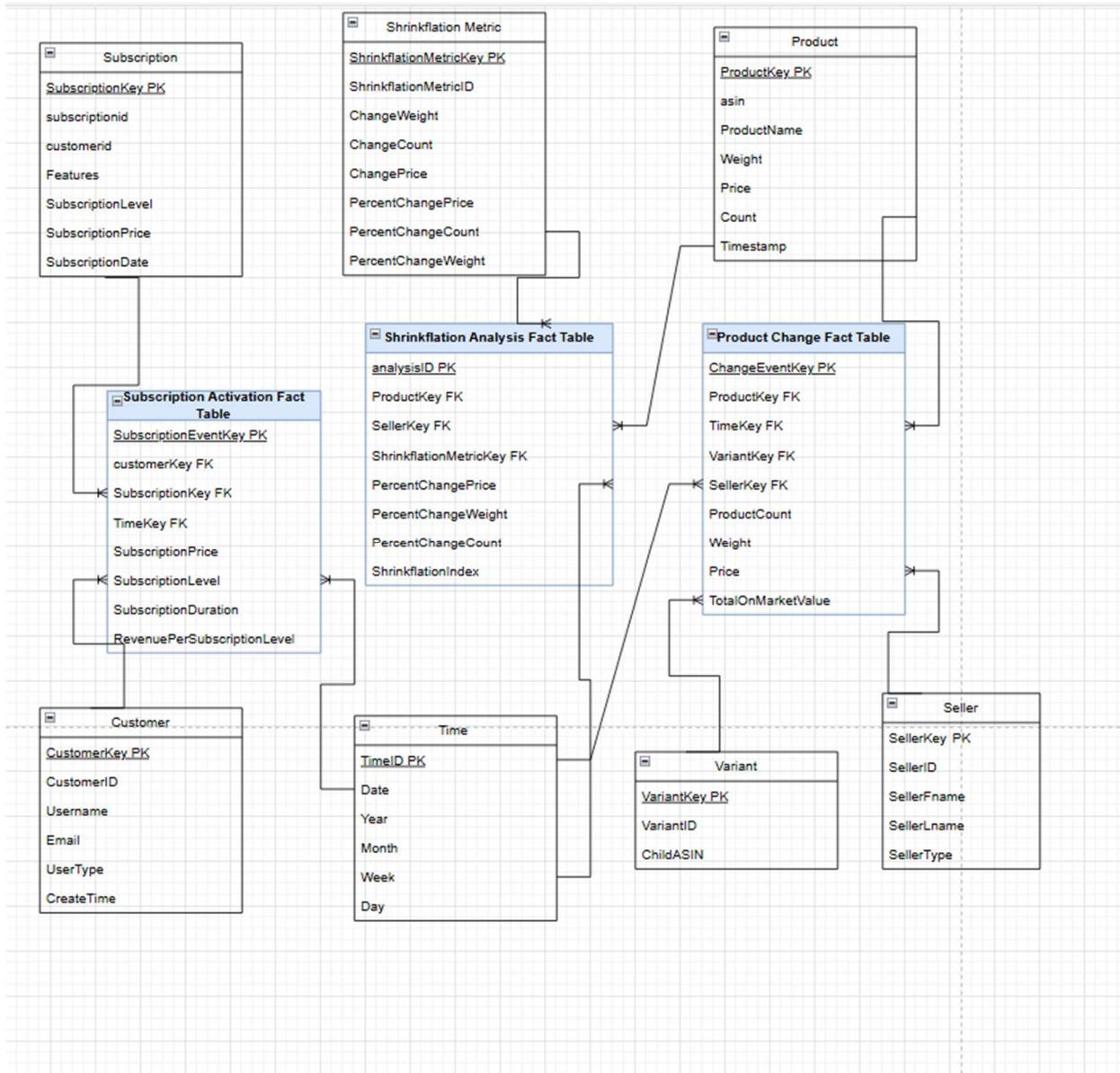
childasin

Cardinalities:

To Product Change Fact Table: 1

(One variant to many product change events)

Logical DW Diagram:



DB Implementation:

The screenshot shows a hierarchical tree structure of database objects. At the top level, there is a node for '1.3 Sequences'. Below it, the 'Tables (10)' node is expanded, revealing ten entries: 'customer_dimension', 'product_change_fact', 'product_dimension', 'seller_dimension', 'shrinkflation_analysis_fact', 'shrinkflation_metric_dimension', 'subscription_activation_fact', 'subscription_dimension', 'time_dimension', and 'variant_dimension'. A separate node for 'Trigger Functions' is also visible at the bottom level.

- > 1.3 Sequences
- ▼ Tables (10)
 - > customer_dimension
 - > product_change_fact
 - > product_dimension
 - > seller_dimension
 - > shrinkflation_analysis_fact
 - > shrinkflation_metric_dimension
 - > subscription_activation_fact
 - > subscription_dimension
 - > time_dimension
 - > variant_dimension
- ▼ Trigger Functions

> sequences
▽ Tables (10)
▽ customer_dimension
▽ Columns (6)
customerkey
customerid
username
email
usertype
createtime
> Constraints
> Indexes
> RLS Policies
> Rules
> Triggers
▽ product_change_fact
▽ Columns (9)
changeeventid
productkey
variantkey
sellerkey
timekey
productcount
weight
price
totalonmarketvalue
> Constraints
> Indexes
> RLS Policies
> Rules
> Triggers
▽ product_dimension
▽ Columns (7)
productkey
asin
productname
weight
price
count
timestamp

The screenshot shows a database schema browser interface with a tree view of table structures. A light gray horizontal bar highlights the 'Rules' section under the 'shrinkflation_analysis_fact' table.

- / triggers
- \ seller_dimension
 - \ Columns (5)
 - sellerkey
 - sellerid
 - sellerfname
 - sellerlname
 - sellertype
 - \ Constraints
 - \ Indexes
 - \ RLS Policies
 - Rules** (highlighted)
 - \ Triggers
- \ shrinkflation_analysis_fact
 - \ Columns (8)
 - analysisid
 - productkey
 - sellerkey
 - shrinkflationmetrickey
 - percentchangeprice
 - percentchangeweight
 - percentchangecount
 - shrinkflationindex
 - \ Constraints
 - \ Indexes
 - \ RLS Policies
 - \ Rules
 - \ Triggers

- | > Triggers
- | ▾ shrinkflation_metric_dimension
 - | > Columns (8)
 - shrinkflationmetrickey
 - shrinkflationmetricid
 - changeweight
 - changecount
 - changeprice
 - percentchangeprice
 - percentchangecount
 - percentchangeweight
 - | > Constraints
 - | > Indexes
 - | > RLS Policies
 - | > Rules
 - | > Triggers
- | ▾ subscription_activation_fact
 - | > Columns (8)
 - subscriptioneventkey
 - customerkey
 - subscriptionkey
 - timekey
 - subscriptionprice
 - subscriptionlevel
 - subscriptionduration
 - revenuepersubscriptionlevel
 - | > Constraints

> Rules
> Triggers
subscription_dimension
Columns (7)
subscriptionkey
subscriptionid
customerid
features
subscriptionlevel
subscriptionprice
subscriptiondate
> Constraints
> Indexes
> RLS Policies
> Rules
> Triggers
time_dimension
Columns (5)
timekey
timeid
day
month
year
> Constraints
> Indexes
> RLS Policies
> Rules
> Triggers
variant_dimension
Columns (3)
variantkey
variantid
childasin
> Constraints
> Indexes
> RLS Policies
> Rules
> Triggers

Primary Events:

Event 1: Shrinkflation Detection Event

This event is triggered when a shrinkflation change is detected in a product's attributes, such as weight, count, or price. This will be recorded in the Shrinkflation Analysis Fact Table.

Key Measures:

PercentChangePrice

PercentChangeWeight

PercentChangeCount

ShrinkflationIndex

Dimensions Involved:

Product Dimension: Identifies the product affected by shrinkflation.

Seller Dimension: Indicates the seller or supplier of the affected product.

Time Dimension: Records the date of the shrinkflation detection.

Shrinkflation Metric Dimension: Captures detailed metrics for weight, count, and price changes.

Shrinkflation Event Dimension: Specifies the type of shrinkflation event detected.

Event 2: Product Change Event

This event is triggered when updates to a product's attributes occur, such as modification of a product variant or discontinuation of a product. This will be recorded in the Product Change Fact Table.

Key Measures:

ProductCount

Revenue

Number of paid users

Number of free users

Dimensions Involved:

Product Dimension: Identifies the specific product involved in the change.

Variant Dimension: Represents different versions or variations of the product.

Seller Dimension: Details the seller associated with the product change.

Time Dimension: Records the time of the product change.

Event 3: Subscription Activation Event

This event will capture instances where a customer activates or upgrades a subscription. This will be recorded in the Subscription Activation Fact Table.

Key Measures:

SubscriptionPrice

SubscriptionLevel

SubscriptionDuration

RevenuePerSubscriptionLevel

Dimensions Involved:

Customer Dimension: Identifies the customer activating or upgrading a subscription.

Subscription Dimension: Provides details about the subscription plan and features.

Time Dimension: Records the activation or upgrade date.

OLAP Queries:

Query 1

Revenue by Year and Month

Roll-Up: Total Subscription Revenue by Year and Month

ROLL-UP(subscription_activation_fact, (Year, Month), SUM(RevenuePerSubscriptionLevel))

Query 2

Number of Products on Price Change per 1%

Drill-Down: Distribution of Products by Percent Change in Price (1% bins)

DRILL-DOWN(shrinkflation_analysis_fact, (PercentChangePrice),
FLOOR(PercentChangePrice))

Query 3

Drill Down: Shrinkflation Events by Product

DRILLDOWN(ShrinkflationEvent, (Product → ProductName), COUNT(EventID))

Query 4

Weight vs. Price

Slice: Filtering Non-Null Weight and Price Change for Correlation Analysis

SLICE(shrinkflation_analysis_fact, (Weight IS NOT NULL AND ChangePrice IS NOT NULL))

Query 5

Seller's Market Share

Dice: Total Market Value by Seller for Active Products

DICE(product_change_fact, (SellerFName, SellerLName), SUM(TotalOnMarketValue))

Query 6

Slice: Average Price Change for a Specific Sector

SLICE(ShrinkflationMetric, SectorName = 'Grocery', AVG(PercentChangePrice))

Query 7

Monthly Notifications Sent to Customers

ROLLUP(Notification, (Time → Month), COUNT(NotificationID))

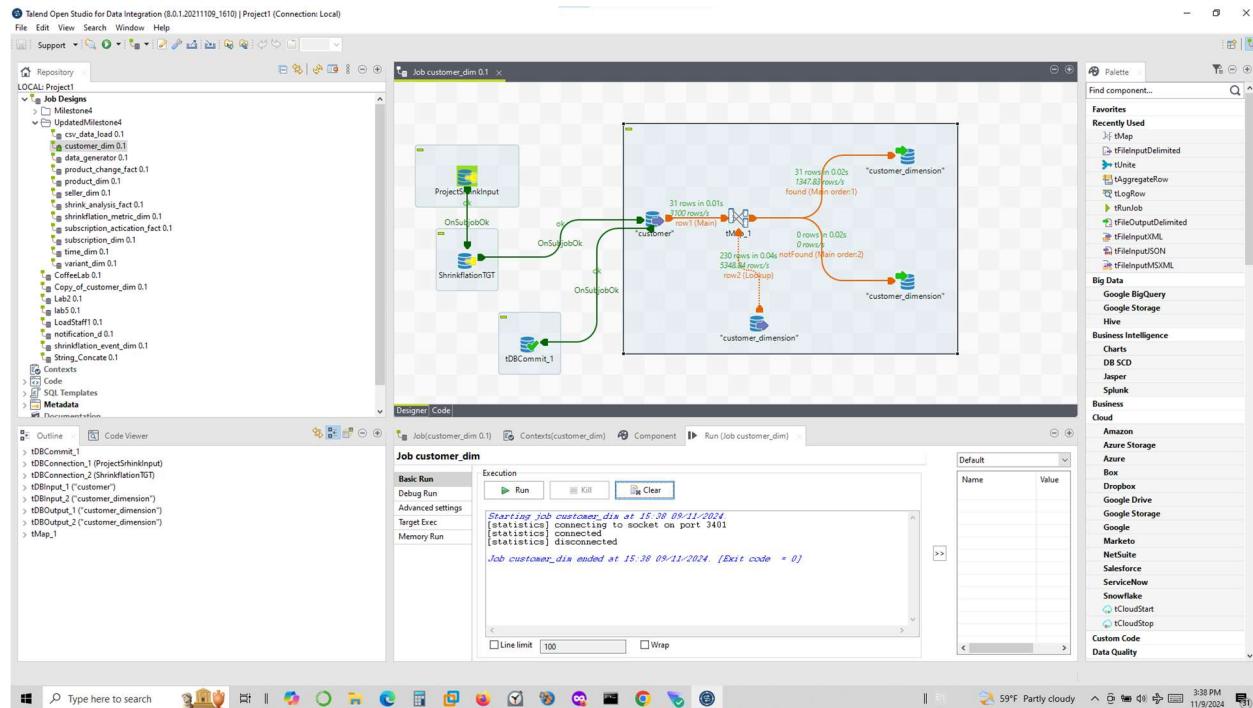
Query 8

Average Price Change Due to Shrinkflation

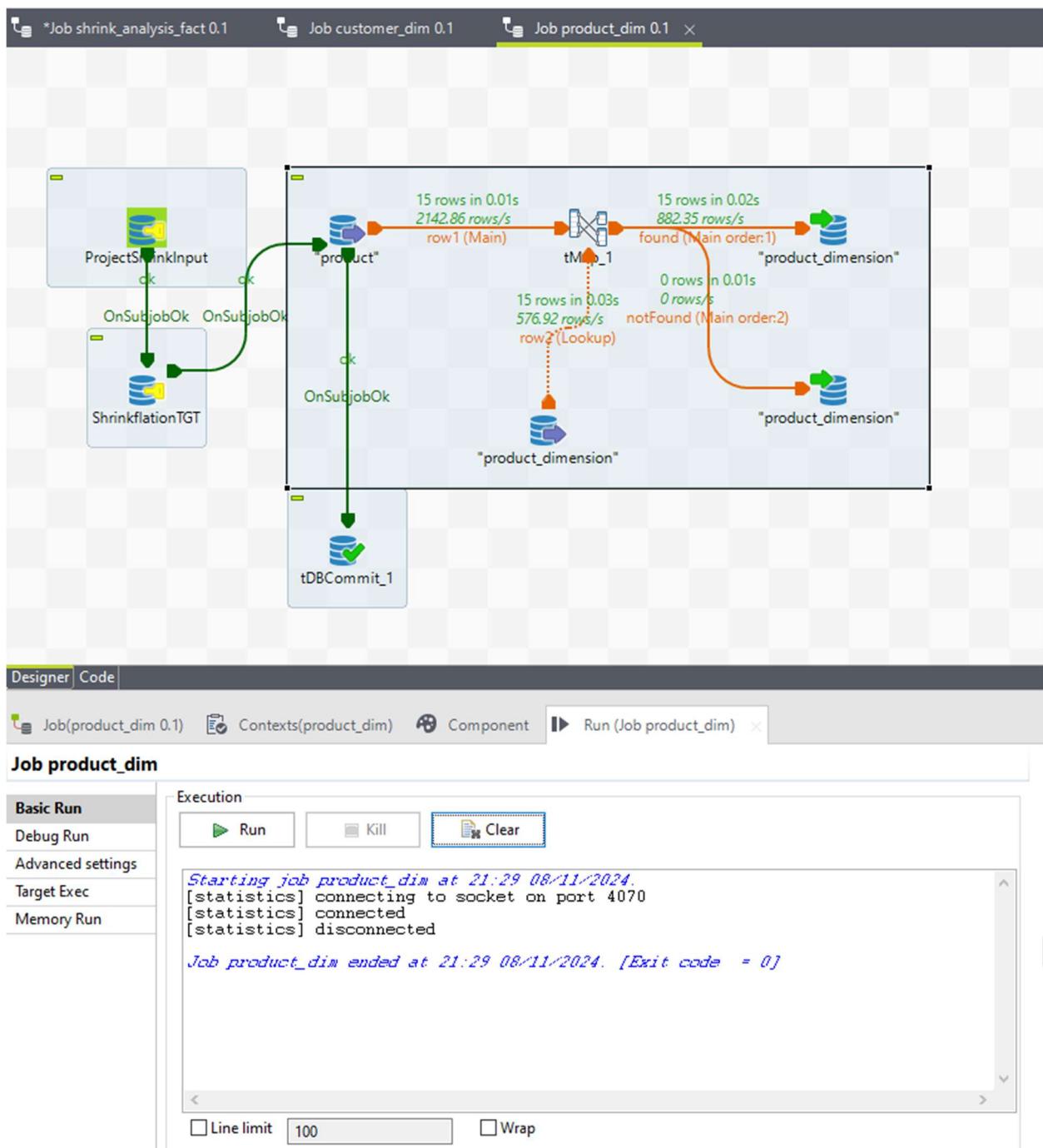
ROLLUP(ShrinkflationMetric, (), AVG(PercentChangePrice))

ETL:

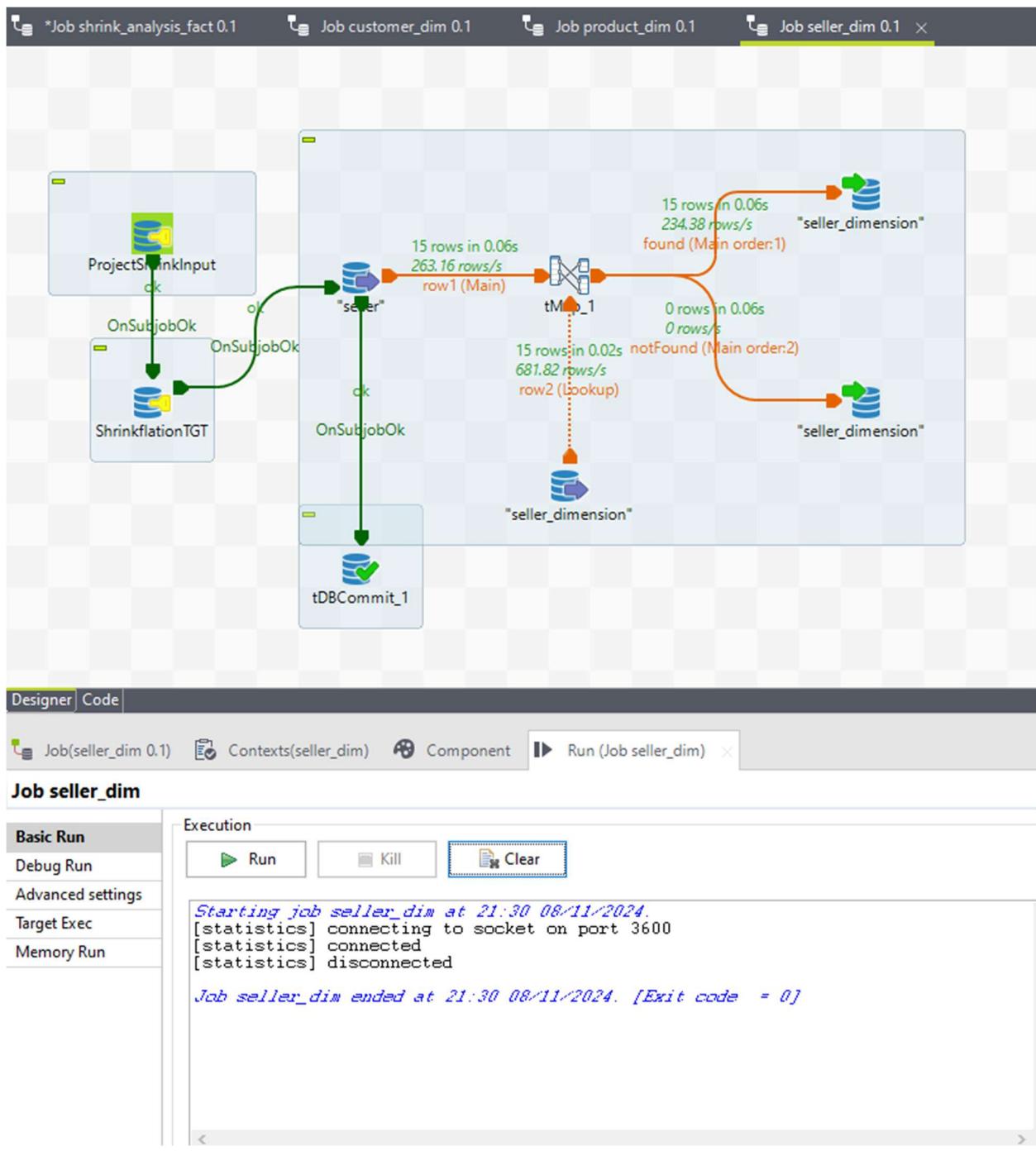
Customer Dimension



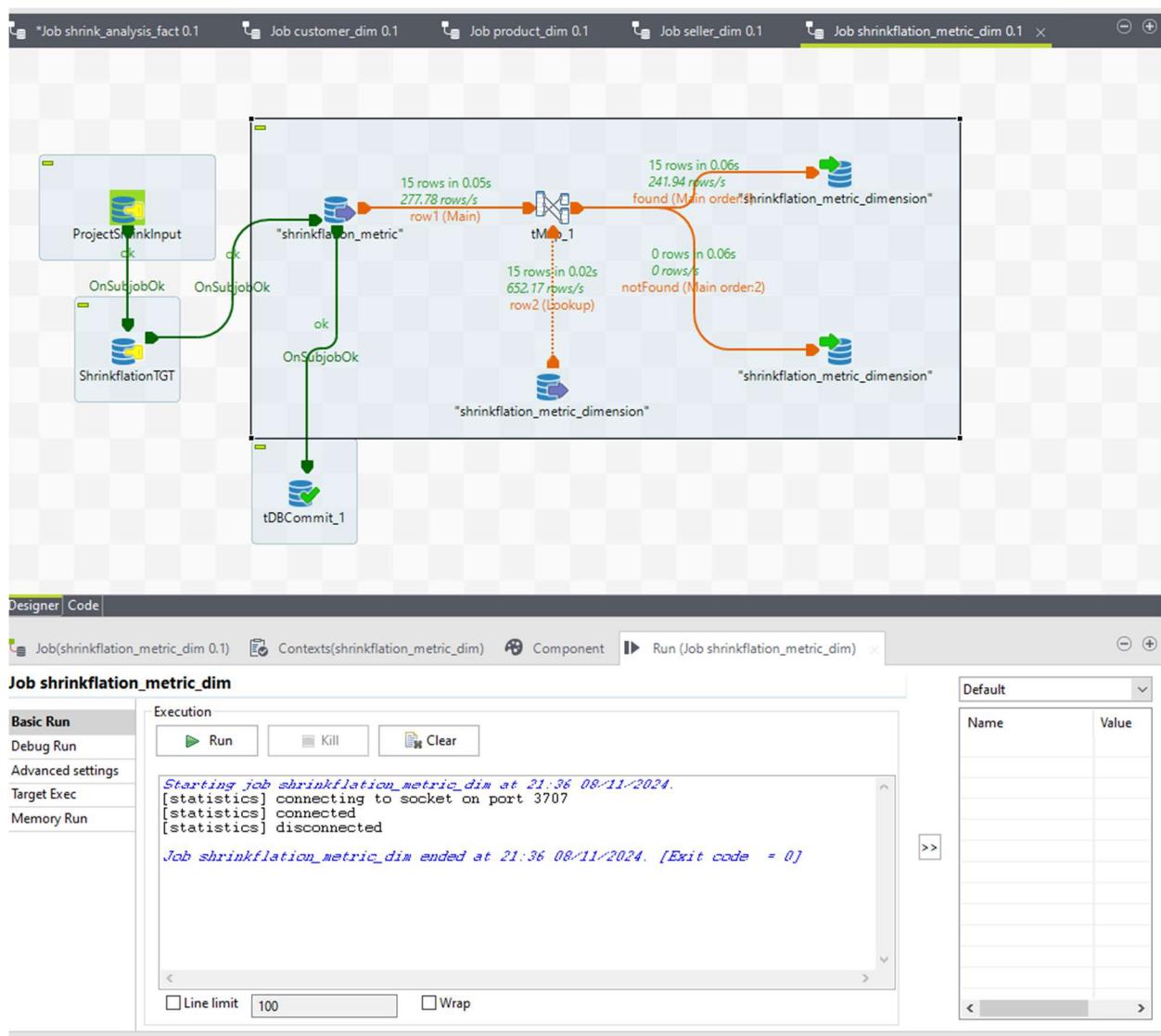
Product Dimension



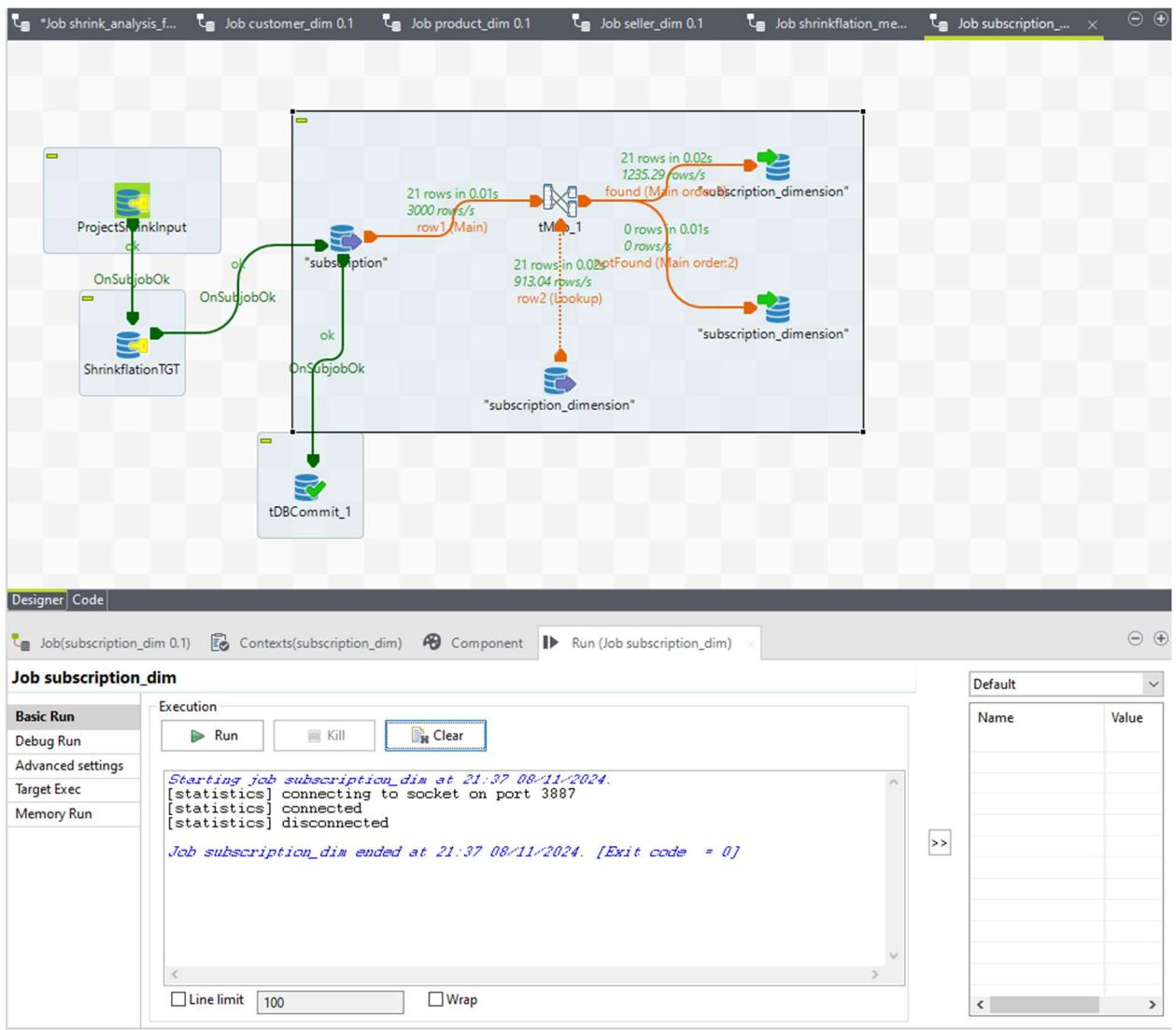
Seller Dimension



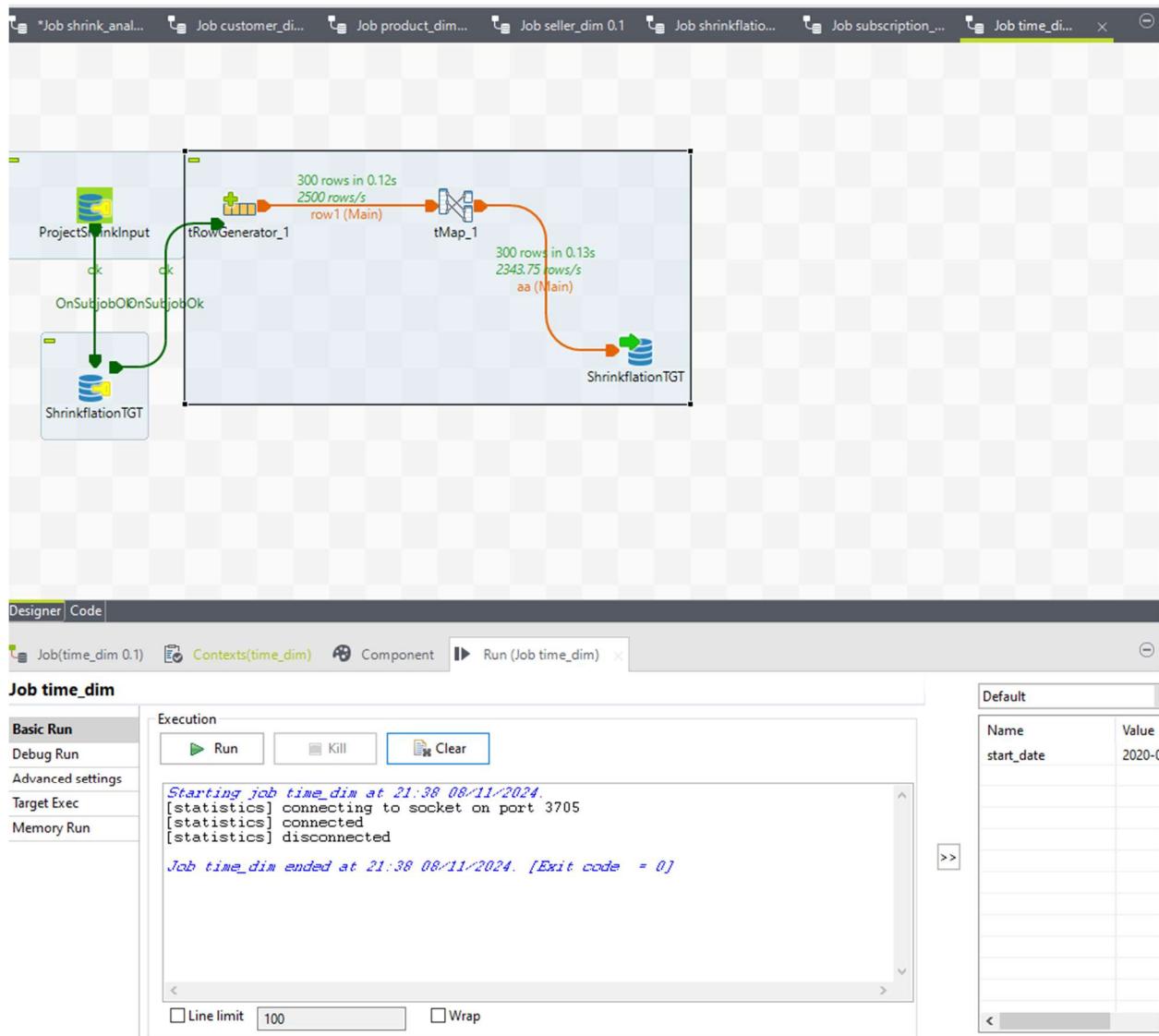
Shrinkflation_Metric Dimension



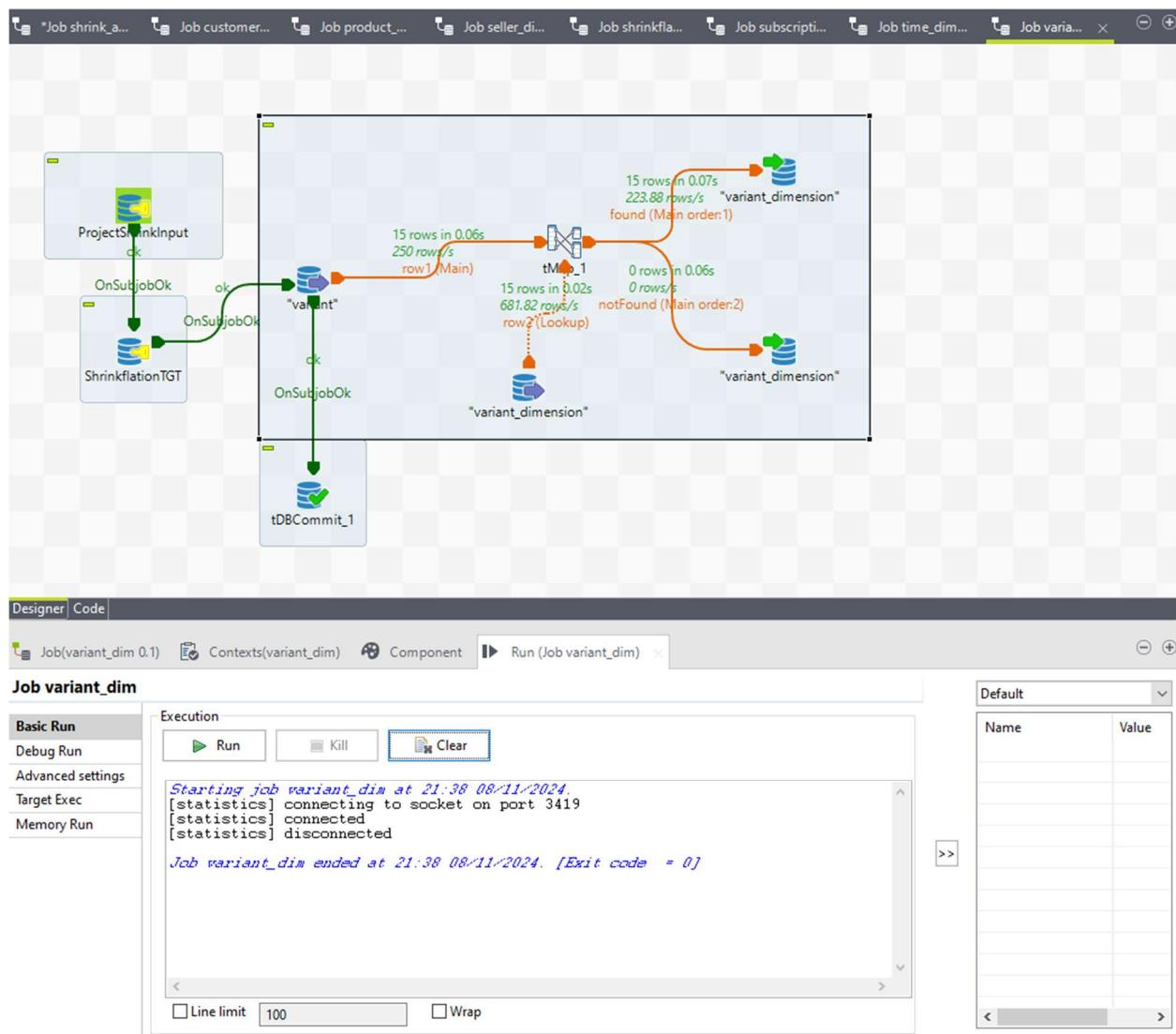
Subscription Dimension



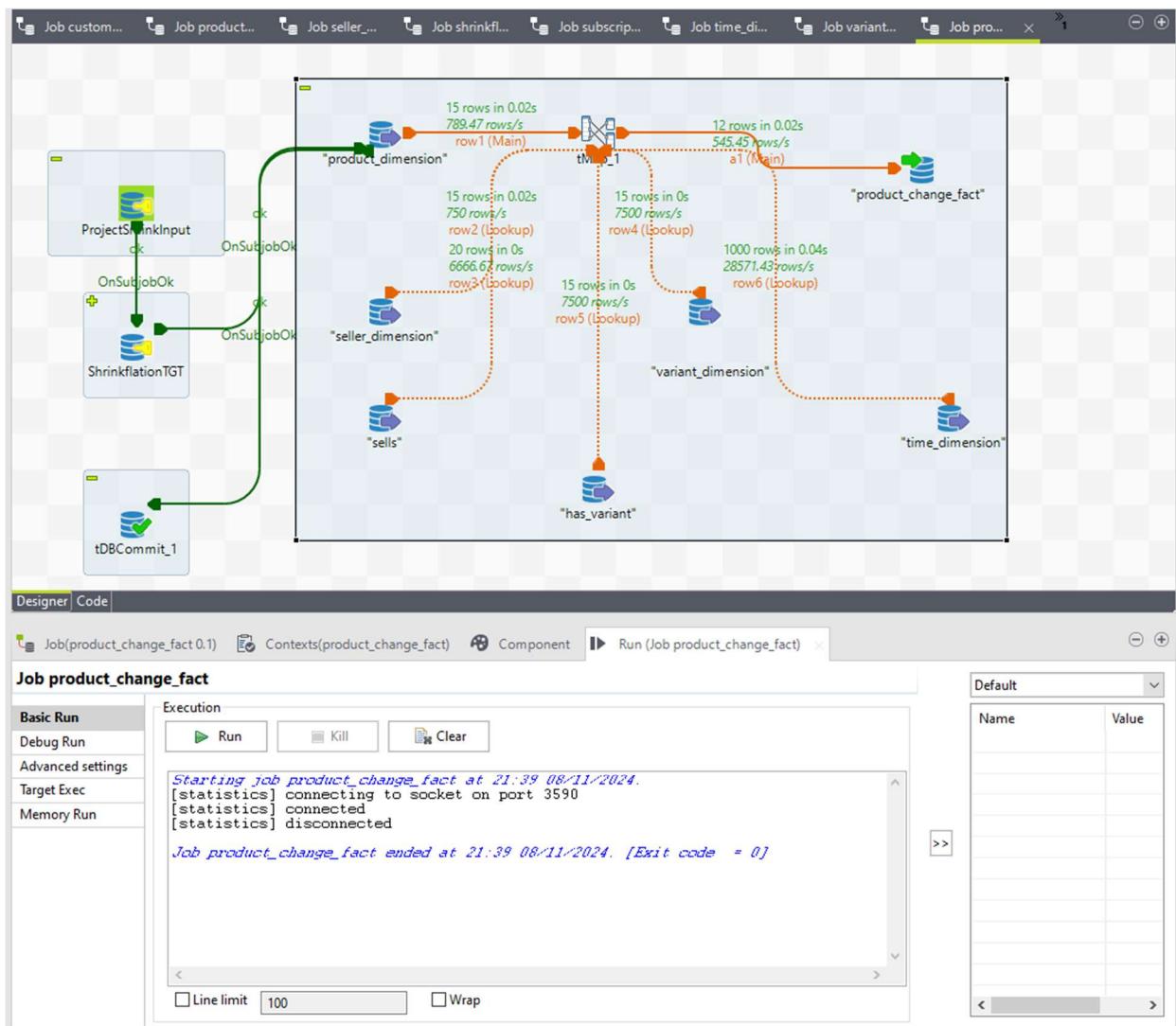
Time Dimension with other data source

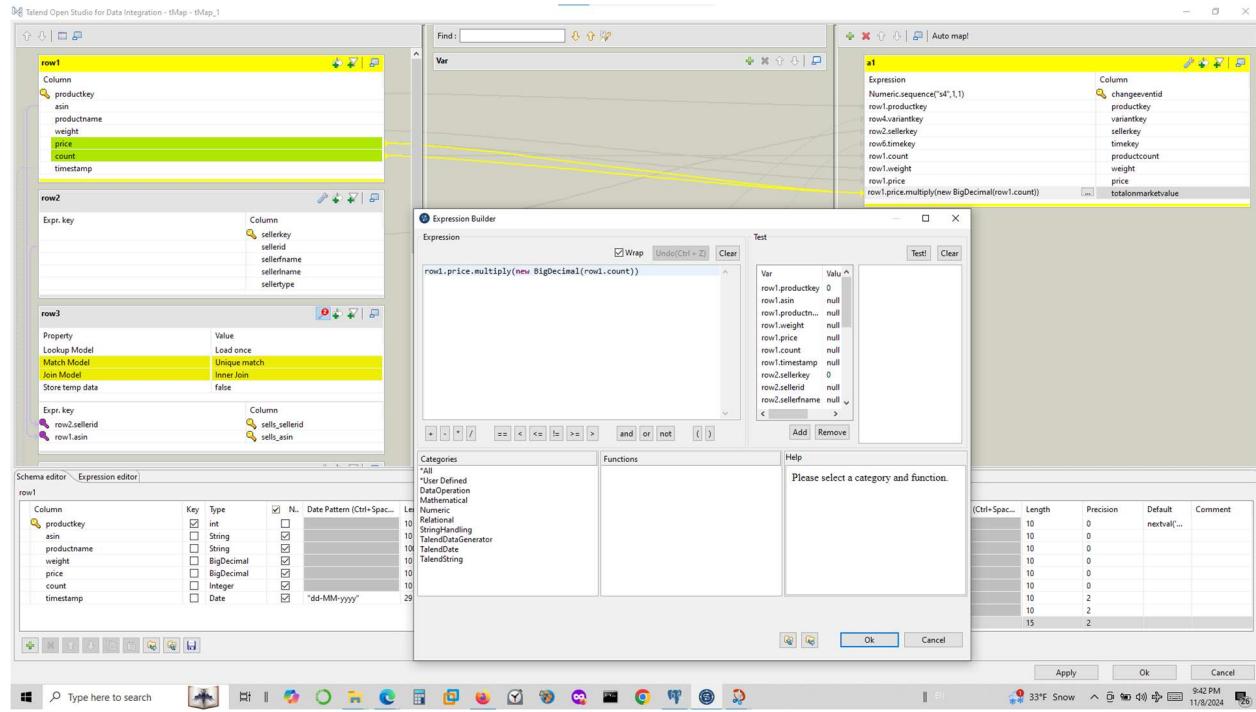


Variant Dimension

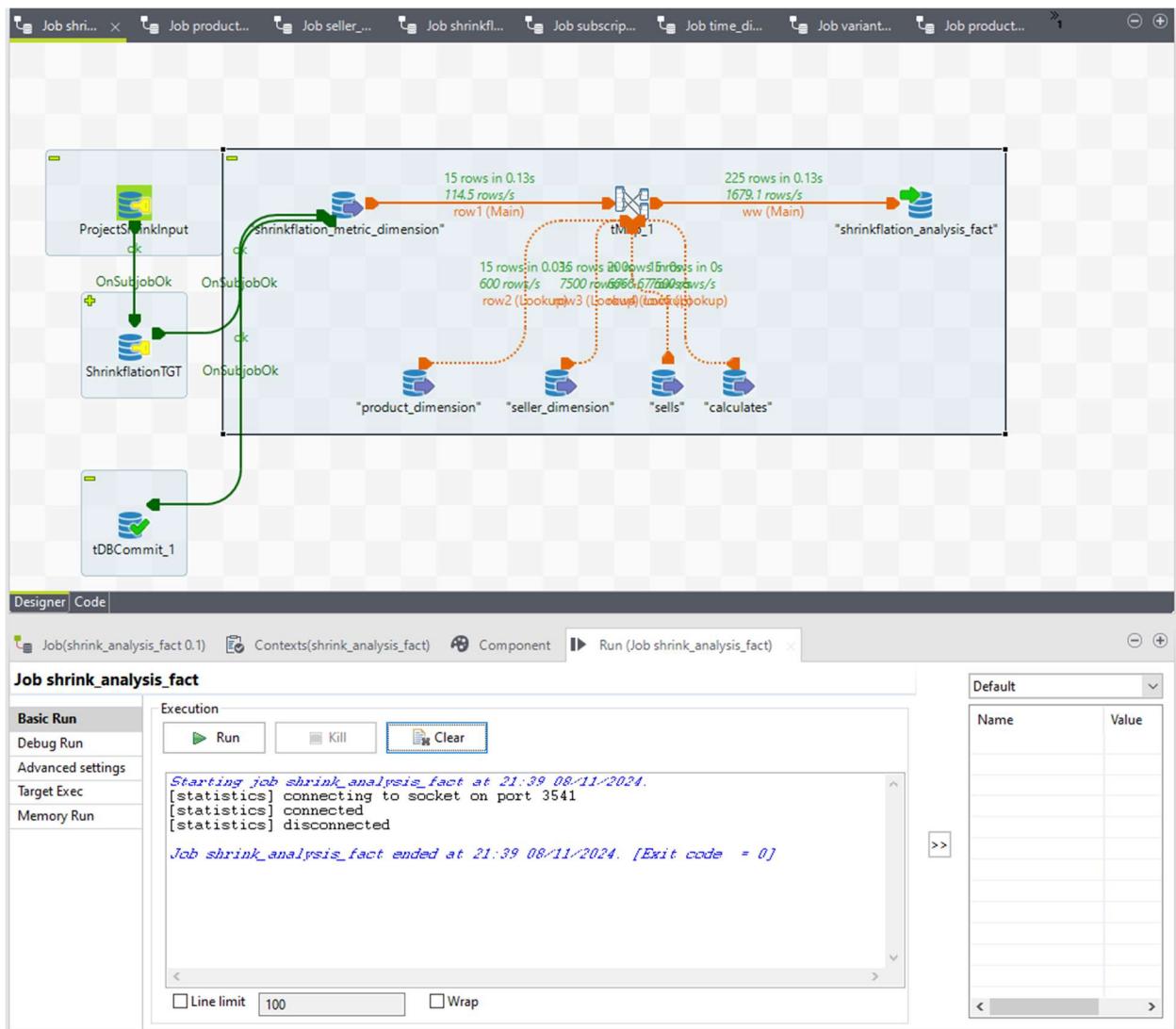


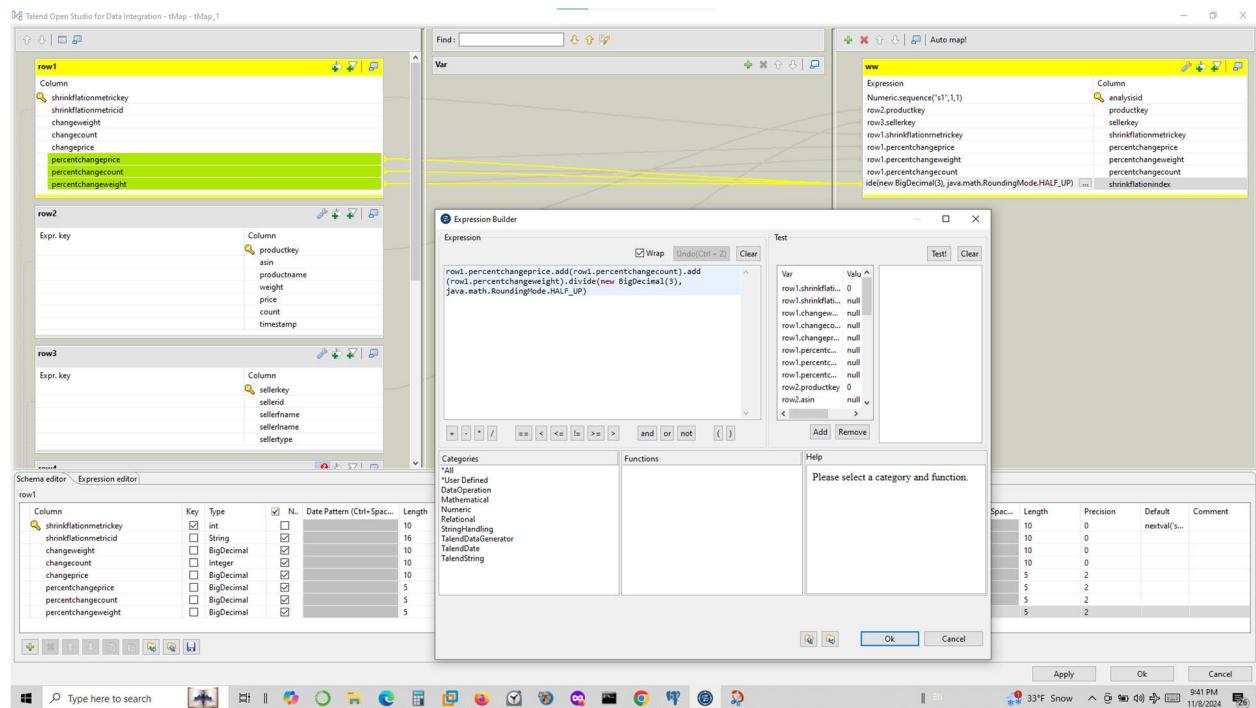
Product Change Fact Table with measures' calculations



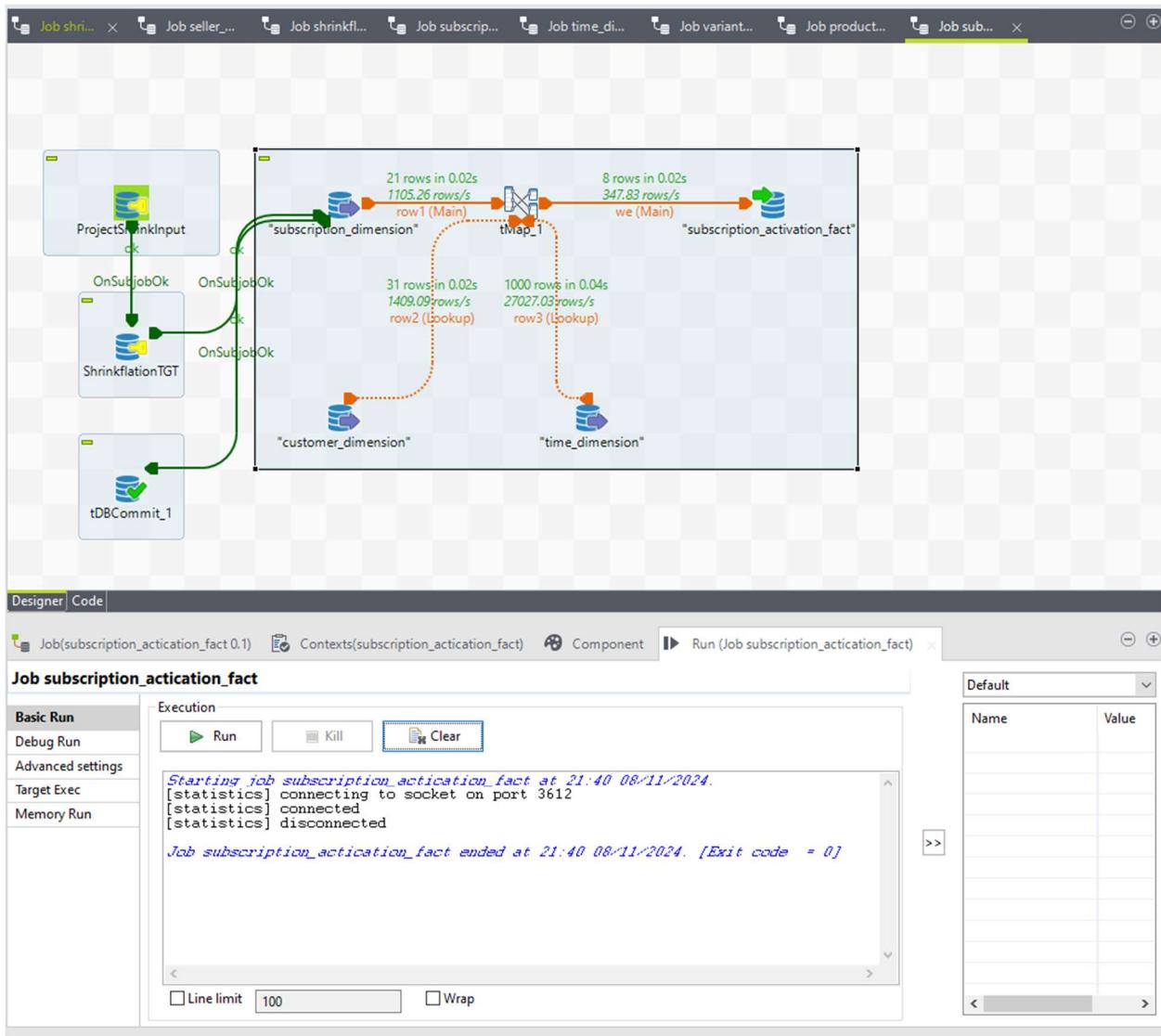


Shrinkflation Analysis Fact with measures' calculations





Subscription Activation Fact with measures' calculations



Talend Open Studio for Data Integration - tMap - tMap_1

Row 1 Schema:

Column	Type	Key	Value
subscriptionkey	String		
subscriptionid	String		
customerid	String		
features	String		
subscriptionlevel	String		
subscriptionprice	BigDecimal		
subscriptiondate	Date		"dd-MM-yyyy"

Row 2 Schema:

Property	Value
Lookup Model	Load once
Match Model	Unique match
Join Model	Inner Join
Store temp data	false

Expr. key:

Column	Value
row1.customerid	customerkey customerid username email userstype createtime

Row 3 Schema:

Property	Value
Lookup Model	Load once
Match Model	Unique match
Join Model	Inner Join

Expression Builder (Top):

```
(int) TalendDate.diffDate(TalendDate.getCurrentDate(),  
row1.subscriptiondate, "dd")
```

Expression Builder (Bottom):

```
row1.subscriptionprice.multiply(  
new BigDecimal((int) TalendDate.diffDate  
(TalendDate.getCurrentDate(), row1.subscriptiondate, "dd")))
```

Variables Table (Top):

Var	Value
row1.subscriptionkey	0
row2.customerkey	null
row1.subscriptionid	null
row1.features	null
row1.subscriptionlevel	null
row1.subscriptionprice	null
row1.subscriptiondate	null
row1.username	null
row2.username	null

Variables Table (Bottom):

Var	Value
row1.subscriptionkey	0
row1.subscriptionid	null
row1.customerid	null
row1.features	null
row1.subscriptionlevel	null
row1.subscriptionprice	null
row1.subscriptiondate	null
row1.username	null
row2.customerid	0
row2.username	null

Schema Editor (Top):

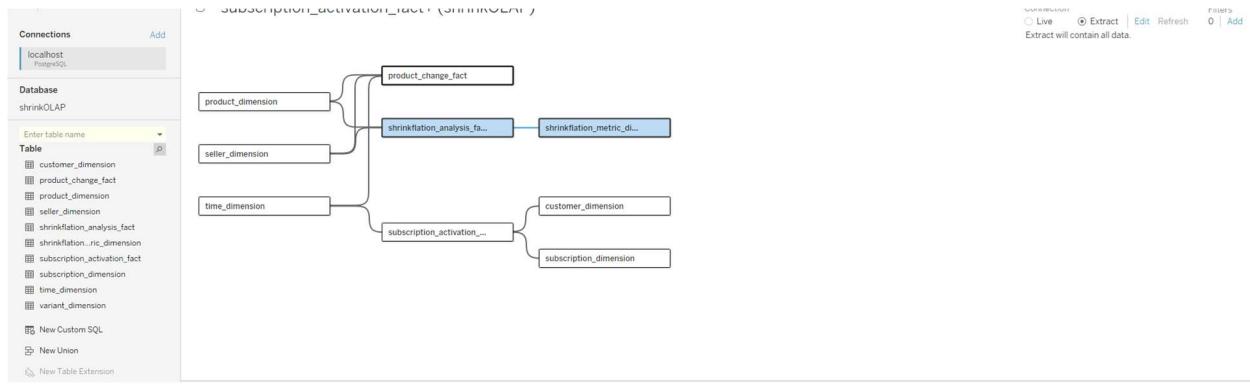
Column	Type	Key	N.	Date Pattern (Ctrl+Space)	Length	Precision	Default	Comment
subscriptionkey	String				10	0	0	
subscriptionid	String				10	0	0	
customerid	String				10	0	0	
features	String				10	0	0	
subscriptionlevel	String				10	2	0	
subscriptionprice	BigDecimal				50	0	0	
subscriptiondate	Date			"dd-MM-yyyy"	10	0	0	

Schema Editor (Bottom):

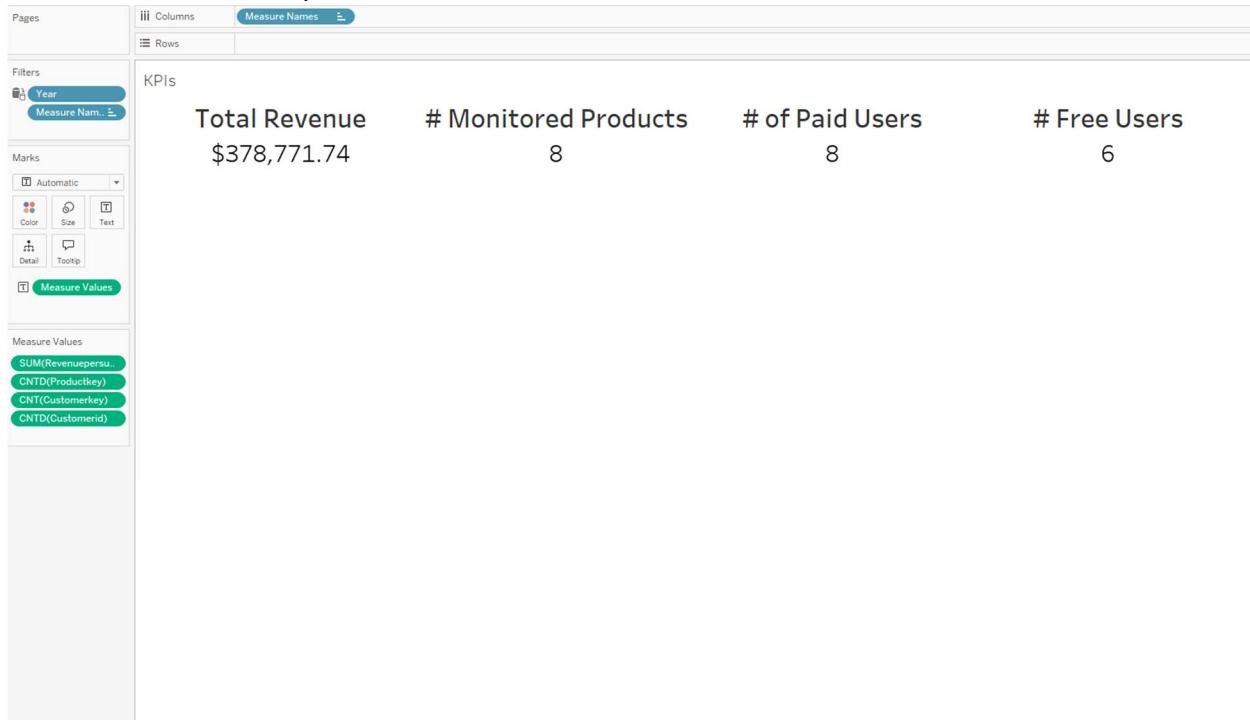
Column	Type	Key	N.	Date Pattern (Ctrl+Space)	Length	Precision	Default	Comment
subscriptionkey	String				10	0	0	
subscriptionid	String				10	0	0	
customerid	String				10	0	0	
features	String				10	0	0	
subscriptionlevel	String				10	2	0	
subscriptionprice	BigDecimal				50	0	0	
subscriptiondate	Date			"dd-MM-yyyy"	10	0	0	

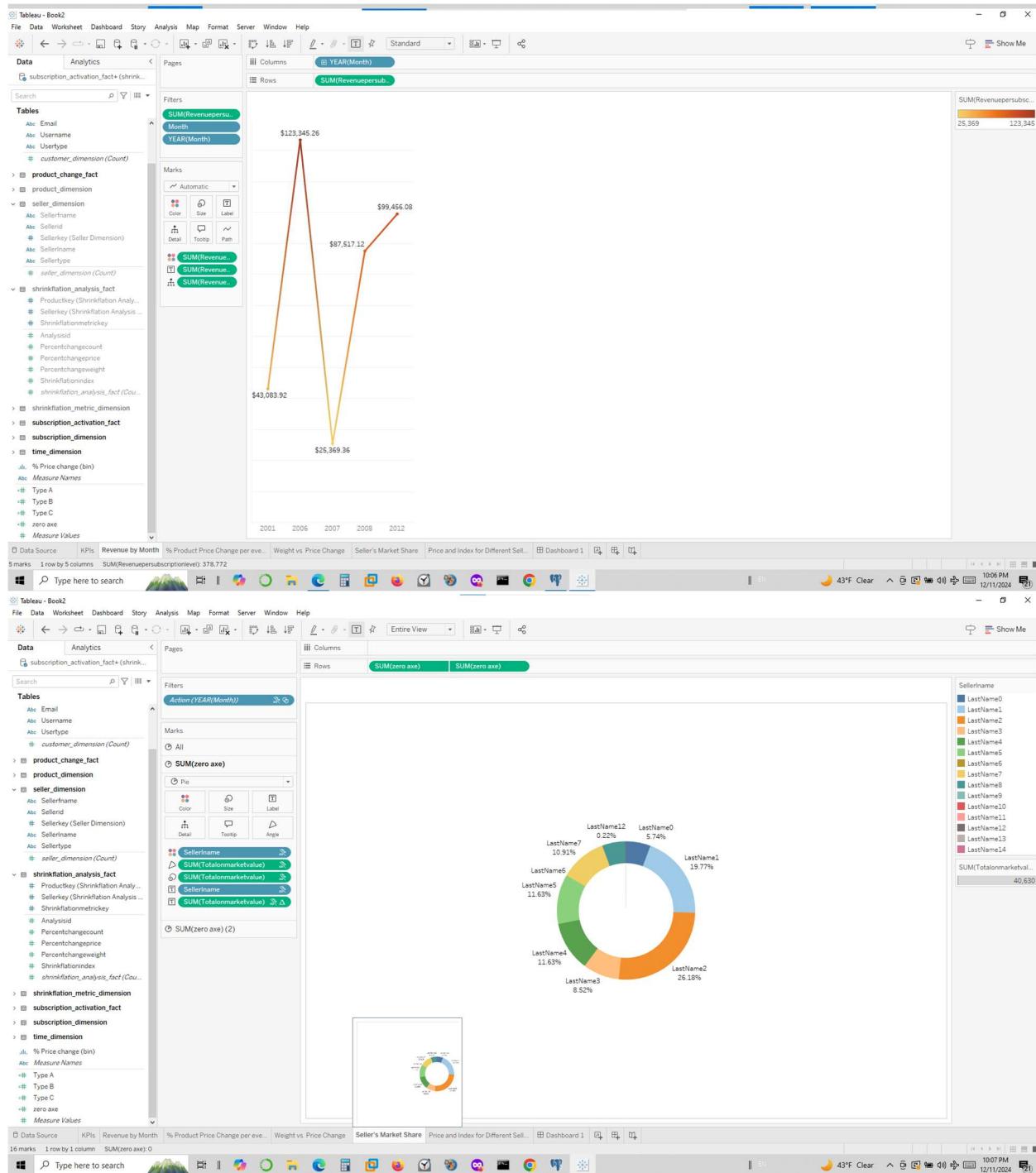
Dashboard on Tableau:

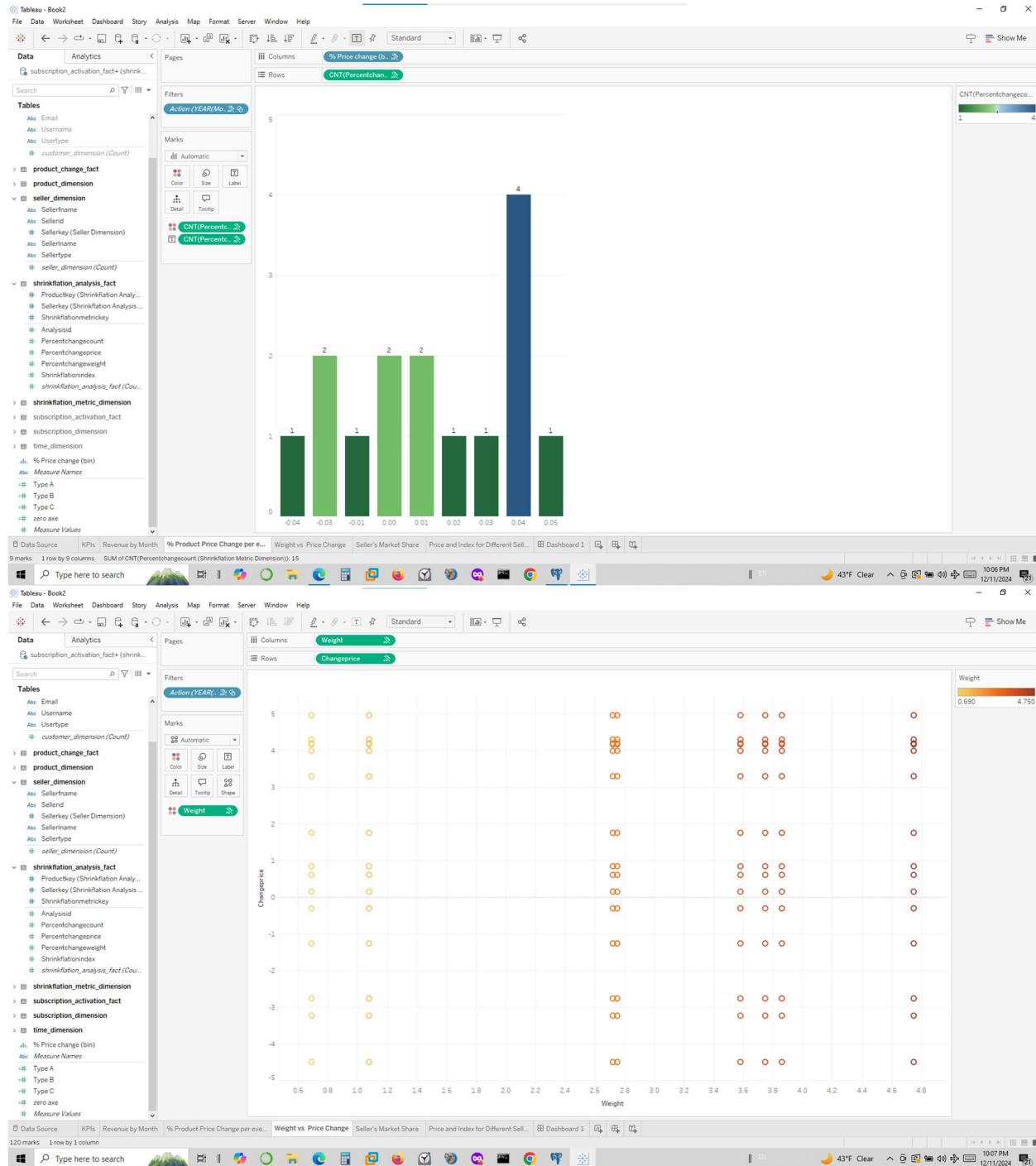
Loading Data from Datawarehouse:



KPIs and Charts Set Up:







The screenshot displays two separate Tableau dashboards side-by-side.

Top Dashboard:

- Sheet 6:** Shows a calculated field editor for "Shrinkflationindex". The formula is: `IF [SellerType] = 'C' THEN [ShrinkflationIndex] END`. A message box indicates "The calculation is valid." with "OK" and "Apply" buttons.
- Marks Card:** Shows three types: Type A (green), Type B (orange), and Type C (yellow).
- Rows:** Shows fields: Percentchangeprice and Shrinkflationindex.
- Columns:** Shows fields: Type A, Type B, and Type C.

Bottom Dashboard:

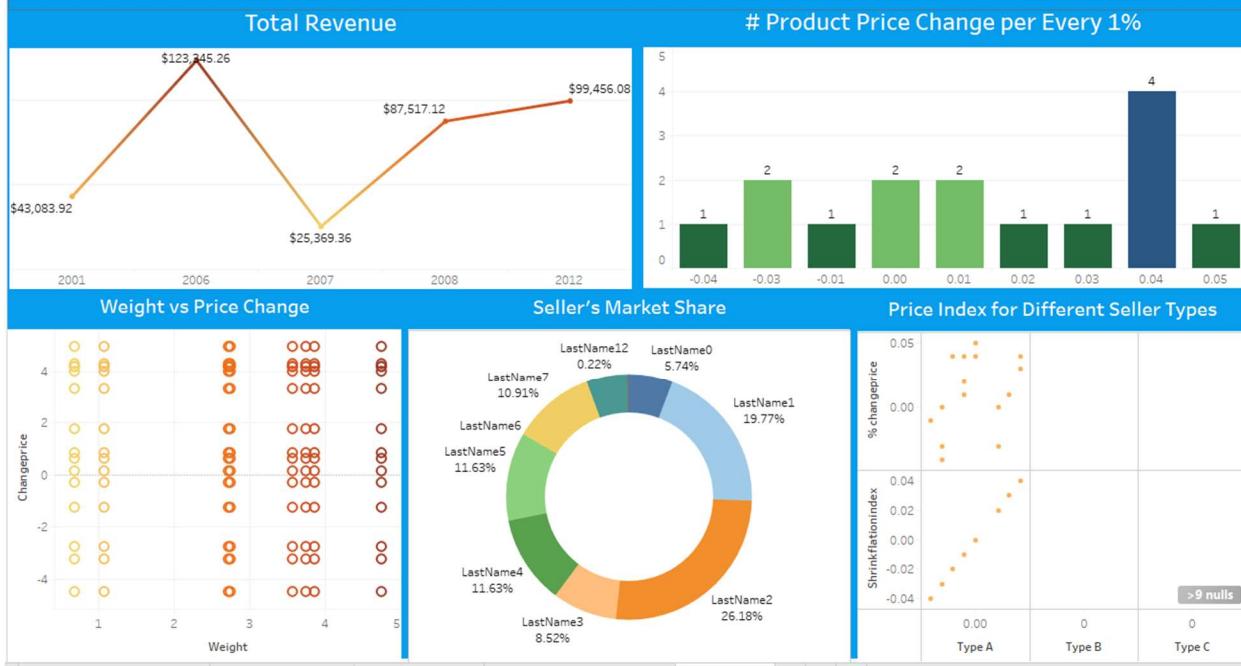
- Sheet 1:** Shows a scatter plot with X-axis "Type A", Y-axis "% changeprice", and a third dimension "Shrinkflationindex".
- Marks Card:** Shows three types: Type A (green), Type B (orange), and Type C (yellow).
- Rows:** Shows fields: Percentchangeprice and Shrinkflationindex.
- Columns:** Shows fields: Type A, Type B, and Type C.

Final Dashboard:

Shrinkflation Analysis

KPIs

Total Revenue	# Monitored Products	# of Paid Users	# Free Users
\$378,771.74	8	9	75



Cloud

Milestone #5

Group I Lei Wu & Edward Brown

Selected Datasets:

<https://www.kaggle.com/datasets/andrewmvd/sp-500-stocks>

The primary datasets we will be utilizing are 3 separate datasets.

sp500_companies.csv : This dataset contains information about the ~500 companies that comprise the S&P 500 index. It includes information such as The exchange the stock trades on, the symbol or ticker the stock trades under, the name of the company, what sector and industry the company operates primarily in, the current price of the stock, market capitalization, EBITDA, and revenue growth.

sp500_index.csv : This dataset contains information about the closing price of the S&P 500 index. This dataset consists of 2 features, a **Date** column, and a corresponding **Closing Price** column that tracks the closing value of the index on a specific date.

sp500_stocks.csv : This dataset contains more detailed information about the individual stocks that comprise the S&P 500 index. The columns it contains are **Date** which indicates what date the data corresponds to, **Symbol** which indicates what particular stock is being referenced, **Adjusted Close** which is similar to the price at market closure, yet also takes into account company actions such as dividends and splits. **Close** which is the price at market close, **High** which is the maximum value of a particular stock on the corresponding day, **Low** is the minimum value of a stock on a day **Open** is the price of a stock at market open, and **Volume** which is the amount of the stock traded on a particular day.

If the need arises we may utilize an ancillary dataset. This would be the **python yfinance** package. Python's yfinance package is based on Yahoo Finances data for the S&P 500. We may need this to reference historical data beyond what our Kaggle datasets support, as yfinance goes back to the origins of the S&P 500 in the early 1920's. This dataset provides additional financial information. This dataset provides real time data for all stocks in the S&P 500 index.

Problem Definition:

Our goal with this project is to create a scalable data pipeline that will ingest and process high volume financial data for the stocks that comprise the S&P 500 stocks. Utilizing our datasets from Kaggle, we will develop a pipeline that can provide real time insights and analytics about the performance of various stocks, and general market behaviors. Our pipeline will continuously ingest data, perform transformations, and store data for ongoing analysis.

Strategic and Operational Insights:

Analyze stock performance by sector to identify trends

Monitor daily changes in stock prices to provide actionable insights to customers

Utilize historical data to develop forecasting models and predictive analytics into macro trends

Analysis Dimensions:

We intend to analyze data across the following dimensions

Time Dimension: daily, weekly, and monthly analysis of stocks

Price Metrics: Evaluate financial metrics such as closing price, opening price, and volume

Market Trends: Identify emerging trends in market performance as well as potential correlations

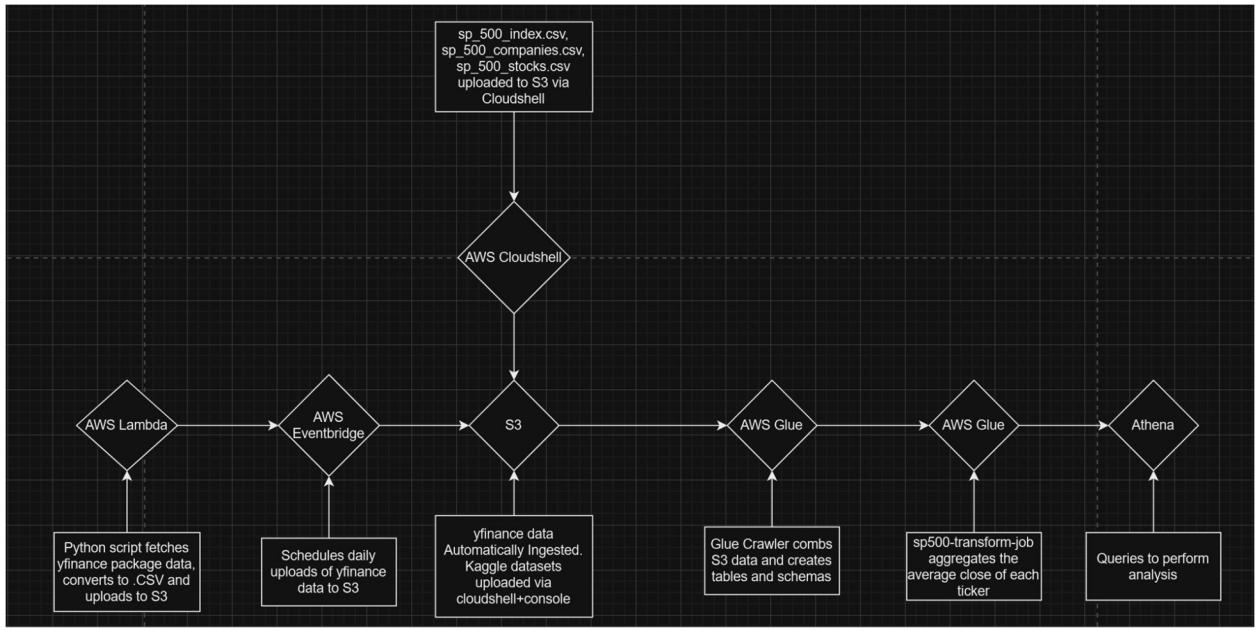
Milestone 6

Milestone #6

Group #1

Lei Wu & Edward Brown

Pipeline Diagram



Tools Used: Amazon S3, Amazon Glue, AWS Lambda, Eventbridge, Cloudshell, PostGRE, Athena

Ingestion Tools: Lambda, Cloudshell

Storage Tool: S3

Transformation Tool: AWS Glue

Analysis Tool: Athena

Programming Language Used: Python, SQL

S3 Ingestion

Uploading files to S3 bucket using cloudshell

```
[CloudShell] us-east-2 +  
[cloudshell-user@ip-10-134-84-174 ~]$ python3 --version  
Python 3.9.16  
[cloudshell-user@ip-10-134-84-174 ~]$ pip3 install boto3 --upgrade  
Defaulting to user installation because normal site-packages is not writeable  
Requirement already satisfied: boto3 in /usr/local/lib/python3.9/site-packages (1.35.20)  
Collecting boto3  
  Downloading boto3-1.35.66-py3-none-any.whl (139 kB)  
    ██████████ | 139 kB 3.7 MB/s  
Requirement already satisfied: s3transfer<0.11.0,>=0.10.0 in /usr/local/lib/python3.9/site-packages (from boto3) (0.10.2)  
Collecting botocore<1.36.0,>=1.35.66  
  Downloading botocore-1.35.66-py3-none-any.whl (12.9 MB)  
    ██████████ | 12.9 MB 6.8 MB/s  
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/local/lib/python3.9/site-packages (from boto3) (1.0.1)  
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/local/lib/python3.9/site-packages (from botocore<1.36.0,>=1.35.66->boto3) (2.9.0.post0)  
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/local/lib/python3.9/site-packages (from botocore<1.36.0,>=1.35.66->boto3) (1.26.20)  
Requirement already satisfied: six<1.5 in /usr/local/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.36.0,>=1.35.66->boto3) (1.16.0)  
Installing collected packages: botocore, boto3  
Successfully installed boto3-1.35.66 botocore-1.35.66  
[cloudshell-user@ip-10-134-84-174 ~]$ nano upload_to_s3.py  
[cloudshell-user@ip-10-134-84-174 ~]$ python3 upload_to_s3.py  
Uploaded sp500_companies.csv to sp500-data-ingestion62715894/raw-data/sp500_companies.csv  
Uploaded sp500_index.csv to sp500-data-ingestion62715894/raw-data/sp500_index.csv  
Uploaded sp500_stocks.csv to sp500-data-ingestion62715894/raw-data/sp500_stocks.csv  
[cloudshell-user@ip-10-134-84-174 ~]$
```

S3 Ingestion Complete (sp500_historical_data.csv was uploaded manual in console)

Amazon S3 > Buckets > sp500-data-ingestion62715894 > raw-data/

raw-data/ Copy S3 URI

Objects Properties

Objects (4) Info Actions Create folder Upload

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	sp500_companies.csv	csv	November 22, 2024, 14:17:27 (UTC-05:00)	786.1 KB	Standard
<input type="checkbox"/>	sp500_historical_data.csv	csv	November 22, 2024, 14:19:50 (UTC-05:00)	478.9 MB	Standard
<input type="checkbox"/>	sp500_index.csv	csv	November 21, 2024, 13:31:47 (UTC-05:00)	46.5 KB	Standard
<input type="checkbox"/>	sp500_stocks.csv	csv	November 21, 2024, 13:31:47 (UTC-05:00)	95.9 MB	Standard

aws | CloudShell | Search [Alt+S] | Ohio | EdwardBrown |

Amazon S3 > Buckets > sp500-data-ingestion62715894 > raw-data/

raw-data/

[Copy S3 URI](#)

[Objects](#) [Properties](#)

Objects (5) info [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#) [Upload](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	sp500_companies.csv	csv	November 22, 2024, 17:39:55 (UTC-05:00)	781.6 KB	Standard
<input type="checkbox"/>	sp500_historical_data.csv	csv	November 22, 2024, 14:19:50 (UTC-05:00)	478.9 MB	Standard
<input type="checkbox"/>	sp500_index.csv	csv	November 22, 2024, 17:46:58 (UTC-05:00)	46.3 KB	Standard
<input type="checkbox"/>	sp500_stocks.csv	csv	November 21, 2024, 13:31:47 (UTC-05:00)	95.9 MB	Standard
<input type="checkbox"/>	sp500-companies-sql	-	November 23, 2024, 16:42:01 (UTC-05:00)	781.5 KB	Standard

[Find objects by prefix](#) | [1](#) | [Last modified](#) | [Size](#) | [Storage class](#)

AWS Marketplace for S3

AWS Glue Crawler

AWS Glue Crawler Settings

Step 2
Choose data sources and classifiers

Step 3
Configure security settings

Step 4
Set output and scheduling

Step 5
Review and create

Step 1: Set crawler properties Edit

Set crawler properties

Name	Description	Tags
sp500-raw-data-crawler	-	-

Step 2: Choose data sources and classifiers Edit

Data sources (1) Info
The list of data sources to be scanned by the crawler.

Type	Data source	Parameters
S3	s3://sp500-data-ingestion62715894/r...	Recrawl all

Step 3: Configure security settings Edit

Configure security settings

IAM role	Security configuration	Lake Formation configuration
AWSGlueServiceRole-6279546	-	-

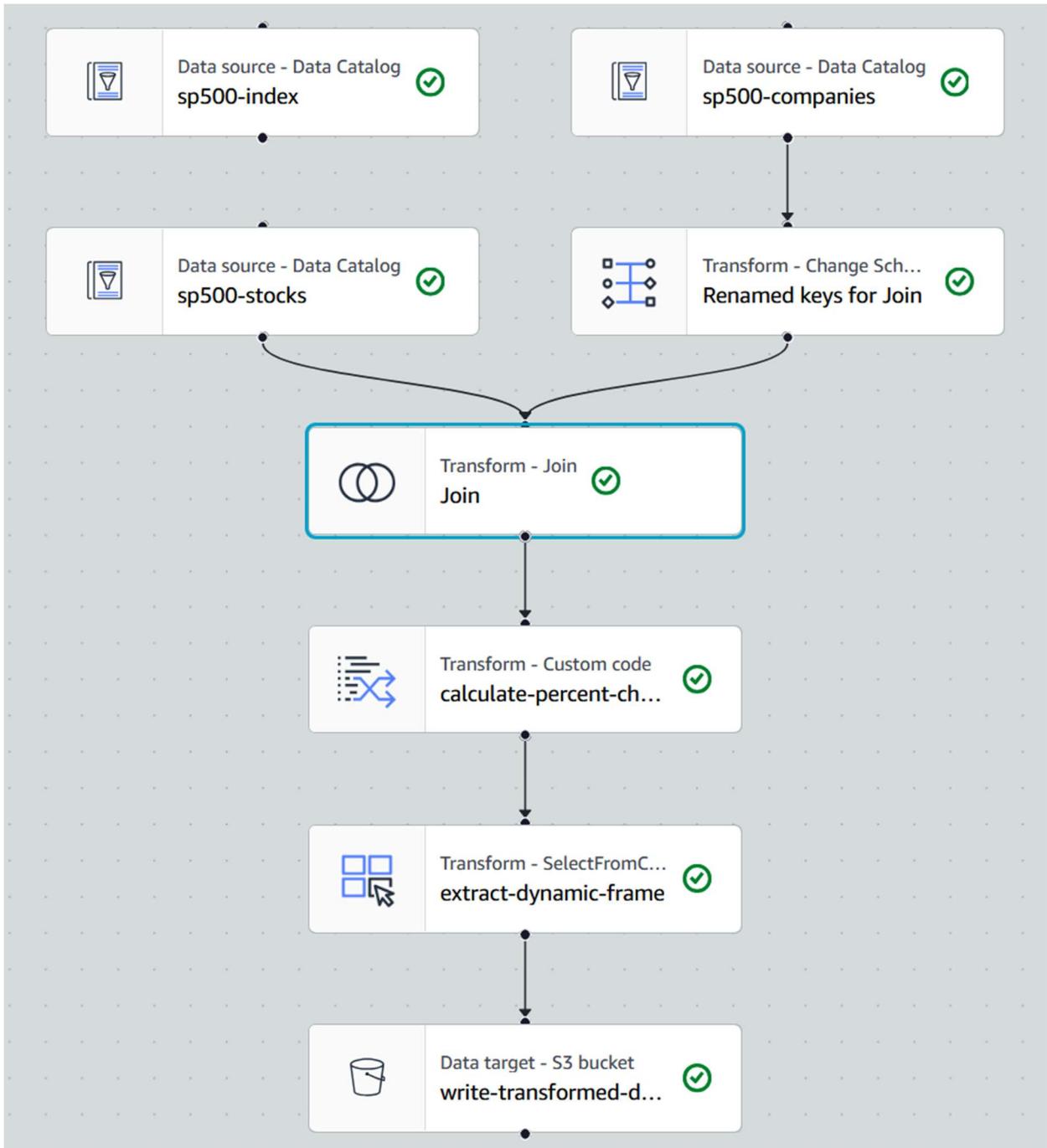
Step 4: Set output and scheduling Edit

Set output and scheduling

Database	Table prefix - <small>optional</small>	Maximum table threshold - <small>optional</small>	Schedule
sp500-db	-	-	On demand

AWS Glue Transformation

AWS GlueStudio transform job setup



AWS GlueStudio Transformation Python Code

```
def MyTransform (glueContext, dfc) -> DynamicFrameCollection:
    from pyspark.sql.functions import col
    from awsglue.dynamicframe import DynamicFrame

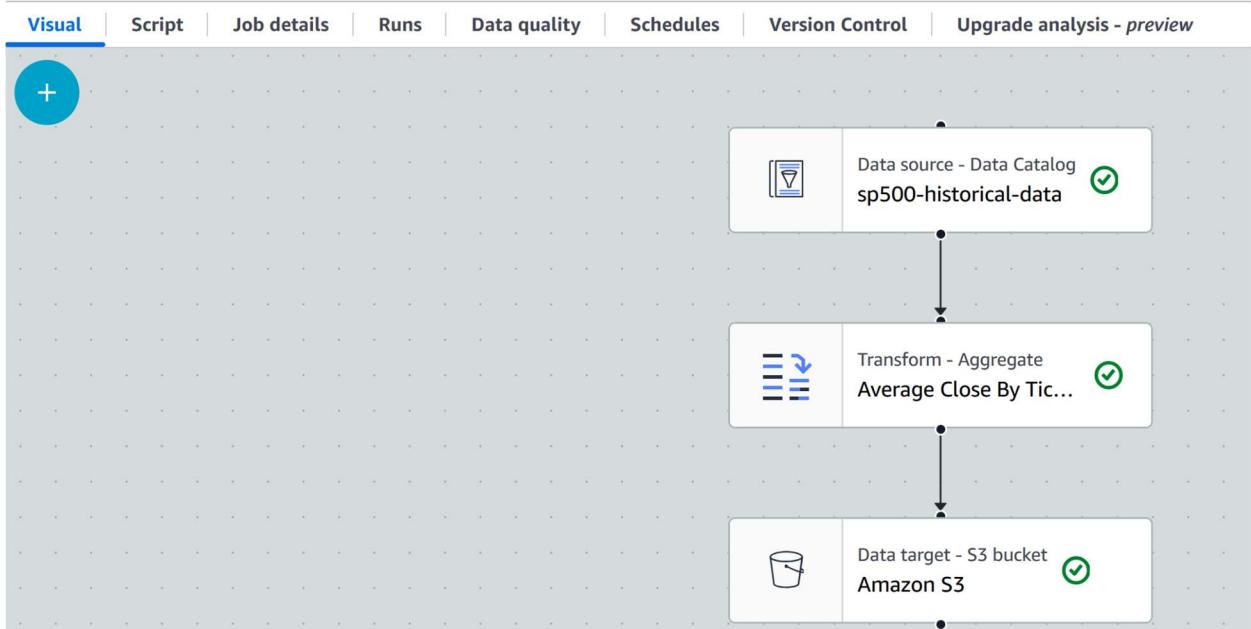
    # Get the first input DynamicFrame from the DynamicFrameCollection
    input_dynamic_frame = dfc.select(list(dfc.keys())[0]).toDF()

    # Perform the transformation
    transformed_df = input_dynamic_frame.withColumn(
        "Percent_Change", (col("Close") - col("Open")) / col("Open") * 100
    )

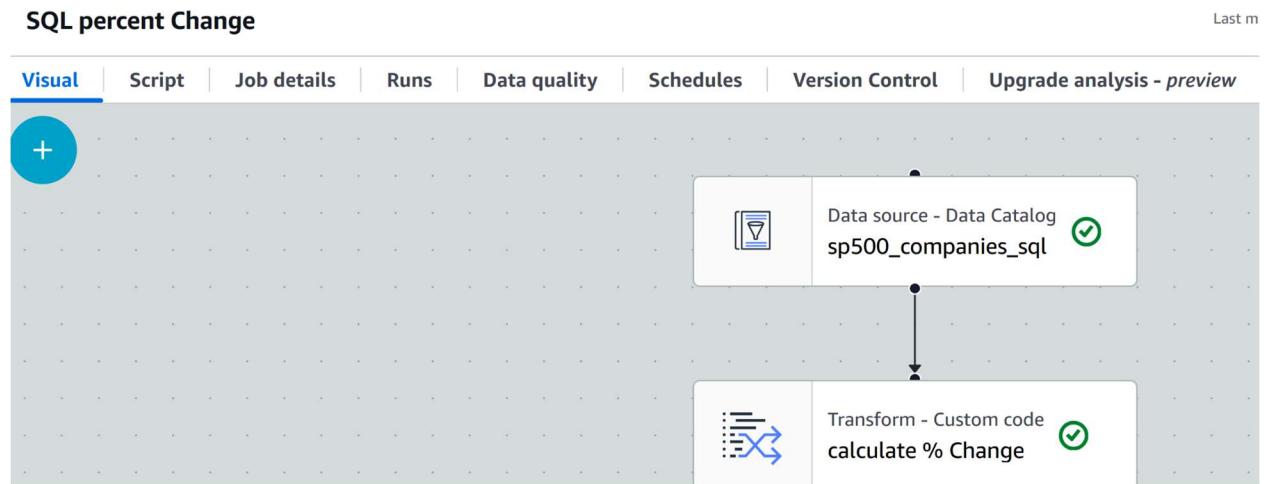
    # Convert the transformed DataFrame back to a DynamicFrame
    transformed_dynamic_frame = DynamicFrame.fromDF(
        transformed_df, glueContext, "transformed_dynamic_frame"
    )

    # Return a collection containing the transformed DynamicFrame
    return DynamicFrameCollection({"Transformed": transformed_dynamic_frame}, glueContext)
```

Average Close by Ticker



S&P 500 companies % change SQL file



Python code for custom transform

```
def MyTransform(glueContext, dfc) -> DynamicFrameCollection:
    from awsglue.dynamicframe import DynamicFrame
    from pyspark.sql.functions import col, lag, when
    from pyspark.sql.window import Window
    # Select the input DynamicFrame and convert it to a Spark DataFrame
    input_dynamic_frame = dfc.select(list(dfc.keys())[0]).toDF()

    # Define a window specification to calculate the previous day's close
    window_spec = Window.partitionBy("Ticker").orderBy("Date")

    # Calculate the percent change
    df = input_dynamic_frame.withColumn("Previous_Close", lag("Close").over(window_spec))
    df = df.withColumn(
        "Percent_Change",
        when(col("Previous_Close").isNotNull(),
            ((col("Close") - col("Previous_Close")) / col("Previous_Close")) * 100)
        .otherwise(None)
    )

    # Convert the resulting DataFrame back into a DynamicFrame
    result_dynamic_frame = DynamicFrame.fromDF(df, glueContext, "result_dynamic_frame")

    # Return the result as a DynamicFrameCollection
    return DynamicFrameCollection({"CustomTransform": result_dynamic_frame}, glueContext)
```

Results from Data Preview

ticker	Previous_Close	Percent_Change
AAPL	0.12203925102949142	1.4082746030774065
AAPL	0.12375789880752563	-2.4303597290003967
AAPL	0.12075013667345047	-2.84679112340992
AAPL	0.11731263250112534	1.098845474894787

Job Success

sp500-transform-job

Last modified on 12/1/2024, 5:19:19 PM [Actions](#) [Save](#) [Run](#)

Visual | Script | Job details | [Runs](#) | Data quality | Schedules | Version Control | Upgrade analysis - preview

Job runs (1/3) [Info](#)

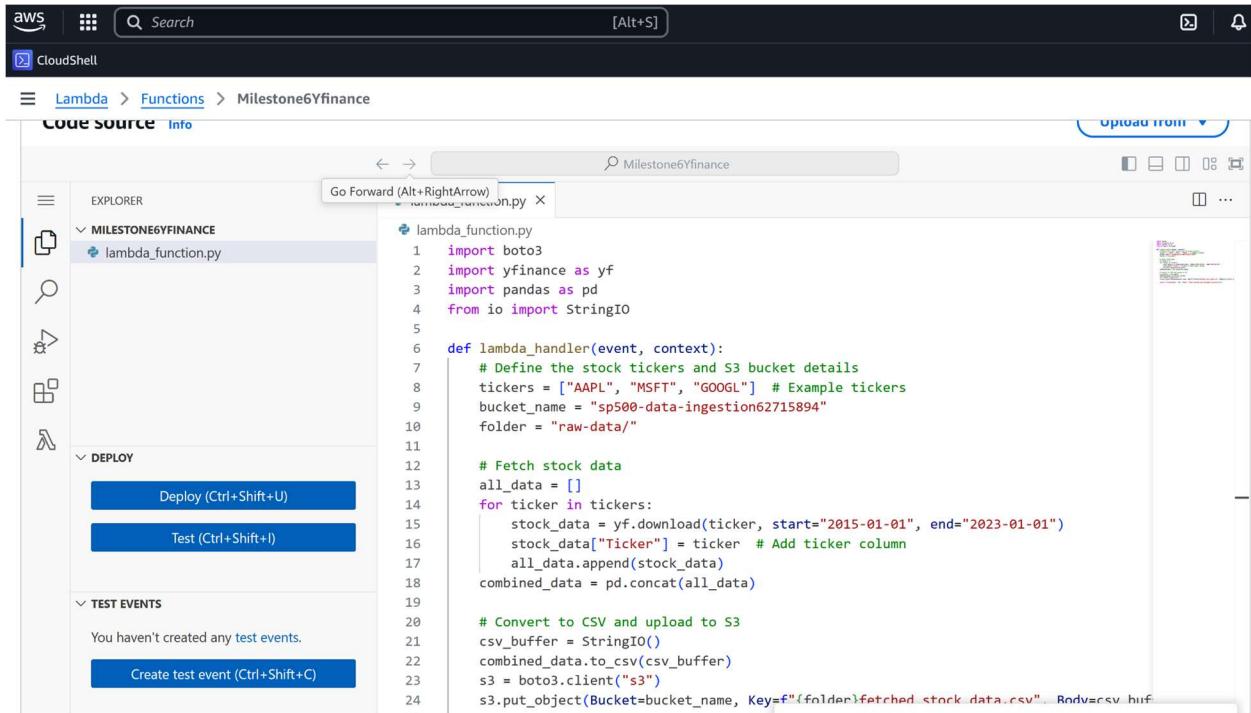
Last updated (UTC) December 1, 2024 at 22:23:55 [View details](#) [Stop job run](#) [Troubleshoot with AI](#)

[Table View](#) [Card View](#)

Run status	Retries	Start time (Local)	End time (Local)	Duration	Capacity (DPUs)	Worker type	Glue version
Success Succeeded	0	12/01/2024 17:19:22	12/01/2024 17:20:58	1 m 29 s	10 DPUs	G.1X	4.0
Success Succeeded	0	11/21/2024 15:51:35	11/21/2024 15:53:26	1 m 40 s	10 DPUs	G.1X	4.0
Stop Stopped	0	11/21/2024 15:21:56	11/21/2024 15:23:16	1 m 10 s	10 DPUs	G.1X	4.0

Lambda Yfinance

Lambda Python code to automate Yfinance data fetch



The screenshot shows the AWS Lambda Function Editor interface. The top navigation bar includes the AWS logo, CloudShell, and a search bar. Below the navigation is a breadcrumb trail: Lambda > Functions > Milestone6Yfinance. The main area is titled "Code Source" with tabs for "Code Source" and "Info". A search bar at the top of the code editor is set to "Milestone6Yfinance". The code editor displays the file "lambda_function.py" with the following content:

```
lambda_function.py
1 import boto3
2 import yfinance as yf
3 import pandas as pd
4 from io import StringIO
5
6 def lambda_handler(event, context):
7     # Define the stock tickers and S3 bucket details
8     tickers = ["AAPL", "MSFT", "GOOGL"] # Example tickers
9     bucket_name = "sp500-data-ingestion62715894"
10    folder = "raw-data/"
11
12    # Fetch stock data
13    all_data = []
14    for ticker in tickers:
15        stock_data = yf.download(ticker, start="2015-01-01", end="2023-01-01")
16        stock_data["Ticker"] = ticker # Add ticker column
17        all_data.append(stock_data)
18    combined_data = pd.concat(all_data)
19
20    # Convert to CSV and upload to S3
21    csv_buffer = StringIO()
22    combined_data.to_csv(csv_buffer)
23    s3 = boto3.client("s3")
24    s3.put_object(Bucket=bucket_name, Key=f"{folder}fetched_stock_data.csv", Body=csv_buffer.getvalue())
25
```

The left sidebar features an "EXPLORER" panel with a "MILESTONE6YFINANCE" folder containing "lambda_function.py". It also includes sections for "DEPLOY" (with "Deploy" and "Test" buttons) and "TEST EVENTS" (with a message: "You haven't created any test events." and a "Create test event" button).

Eventbridge Scheduling

Eventbridge Lambda function scheduling

daily-lambda-trigger

[Disable](#) [Edit](#) [Delete](#)

Schedule detail

Schedule name daily-lambda-trigger	Status <input checked="" type="checkbox"/> Enabled	Schedule start time -	Flexible time window 5 minutes
Description -	Schedule ARN <input type="text"/> arn:aws:scheduler:us-east-2:314146318449:schedule/default/daily-lambda-trigger	Schedule end time -	Created date Nov 21, 2024, 18:12:26 (UTC-05:00)
Schedule group name default	Action after completion NONE	Execution time zone America/New_York	Last modified date Nov 21, 2024, 18:12:26 (UTC-05:00)
Schedule Target Retry policy Dead-letter queue Encryption			

Schedule

Cron expression [Info](#)

0 0 * * ? *

Minutes Hours Day of month Month Day of week Year

Python Data Fetching

Python code to fetch historical data from Yfinance for all companies in sp500_companies

```
✓ 4m ⏴ import yfinance as yf
    import pandas as pd

    # Load the sp500_companies.csv file
    sp500_companies_file = "sp500_companies.csv"
    df_companies = pd.read_csv(sp500_companies_file)

    # Extract tickers from the "Symbol" column
    tickers = df_companies['Symbol'].unique()
    print(f"Found {len(tickers)} tickers.")

    # Initialize an empty DataFrame to hold all stock data
    all_stock_data = pd.DataFrame()

    # Fetch historical data for each ticker and append to the DataFrame
    for ticker in tickers:
        print(f"Fetching data for: {ticker}")
        try:
            # Fetch maximum available historical data
            stock_data = yf.Ticker(ticker).history(period="max")
            if not stock_data.empty:
                # Add a column for the ticker symbol
                stock_data['Ticker'] = ticker
                # Reset index to turn the Date into a column
                stock_data.reset_index(inplace=True)
                # Append to the collective DataFrame
                all_stock_data = pd.concat([all_stock_data, stock_data], ignore_index=True)
            else:
                print(f"No data found for {ticker}.")
        except Exception as e:
            print(f"Failed to fetch data for {ticker}: {e}")

    # Check if data was collected
    if not all_stock_data.empty:
        # Save the combined data to a local CSV file
        csv_filename = "sp500_historical_data.csv"
        all_stock_data.to_csv(csv_filename, index=False)
        print(f"Saved combined data to {csv_filename}.")
    else:
        print("No stock data was collected.")

    print("Data fetching completed.")
```

Fetching data for: UHS
Fetching data for: CAG

IAM Roles

AWSGlueRole

IAM > Roles > AWSGlueServiceRole-6279546

Entity and Access management (IAM)

Search IAM

Permissions policies (0) [Info](#)

You can attach up to 10 managed policies.

Filter by Type

Policy name	Type	Attached ent
AmazonAthenaFullAccess	AWS managed	2
AmazonS3FullAccess	AWS managed	4
AmazonS3ReadOnlyAccess	AWS managed	1
AWSGlueConsoleFullAccess	AWS managed	1
AWSGlueServiceRole	AWS managed	1
AWSGlueServiceRole-6279546-EZCRC-s3Pol...	Customer managed	1

Simulate

SQL Data

Due to an issue with my dataset s&p500_companies.csv I took the advice of the TA and loaded my file into Postgre in order to convert the .csv into a SQL query.

Creating Table in PGadmin

11/23/2024 11:41:45 AM	63 msec
Date	Rows affected
Duration	

CREATE TABLE sp500_companies (Exchange VARCHAR(255) , Symbol VARCHAR(255) **PRIMARY KEY** , Shortname VARCHAR(255) , Longname VARCHAR(255) , Sector VARCHAR(255) , Industry VARCHAR(255) , Currentprice NUMERIC(10, 2) , Marketcap BIGINT , Ebitda NUMERIC(20, 2) , Revenuegrowth NUMERIC(10, 3) , City VARCHAR(255) , State VARCHAR(255) , Country VARCHAR(255) , Fulltimeemployees INTEGER , Longbusinesssummary TEXT , Weight NUMERIC(10, 6));

Messages

Query returned successfully in 63 msec.

Postgre Schema

▼	Tables (1)
▼	sp500_companies
▼	Columns (16)
	exchange
	symbol
	shortname
	longname
	sector
	industry
	currentprice
	marketcap
	ebitda
	revenuegrowth
	city
	state
	country
	fulltimeemployees
	longbusinesssummary
	weight

After Creating the table in Postgre, I uploaded my s&p500_companies.csv into this table. Then I executed a backup of this table to obtain the SQL command needed to create this table manually.

Storage of Transformed Data

Storage of Transformed Data

transformed-data/

[Copy S3 URI](#)

Objects Properties

Objects (36) [Info](#)



[Copy S3 URI](#)

[Copy URL](#)

[Download](#)

[Open](#)

[Delete](#)

[Actions](#) ▾

[Create folder](#)

[Upload](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

[Find objects by prefix](#)

< 1 > |

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	run-1733091630125-part-block-0-r-00000-snappy.parquet	parquet	December 1, 2024, 17:20:44 (UTC-05:00)	536.0 B	Standard
<input type="checkbox"/>	run-1733091630125-part-block-0-r-00001-snappy.parquet	parquet	December 1, 2024, 17:20:44 (UTC-05:00)	558.0 B	Standard
<input type="checkbox"/>	run-1733091630125-part-block-0-r-00002-snappy.parquet	parquet	December 1, 2024, 17:20:44 (UTC-05:00)	530.0 B	Standard
<input type="checkbox"/>	run-1733091630125-part-block-0-r-00003-snappy.parquet	parquet	December 1, 2024, 17:20:44 (UTC-05:00)	668.0 B	Standard
<input type="checkbox"/>	run-1733091630125-part-block-0-r-00004-snappy.parquet	parquet	December 1, 2024, 17:20:48 (UTC-05:00)	594.0 B	Standard
<input type="checkbox"/>	run-1733091630125-part-block-0-r-00005-snappy.parquet	parquet	December 1, 2024, 17:20:44 (UTC-05:00)	653.0 B	Standard

Athena Queries

Average Close functionality

Query 38 : X | Query 41 : X | Query 42 : X | Query 43 : X | Query 44 : X | [Query 45](#) : X

1 | SELECT * FROM "AwsDataCatalog"."sp500-db"."ticker-average-close" limit 10;

SQL Ln 1, Col 1

[Run again](#) [Explain](#) [Cancel](#) [Clear](#) [Create](#)

[Query results](#) [Query stats](#)

[Completed](#)

Results (10)

Search rows

#	▼	ticker	▼	avg_close_#0
1		CL		21.633993552367947
2		AME		30.66288012718029
3		WBA		18.62018082718893
4		GIS		19.309879500399067
5		LYV		38.84524038912693
6		MCK		110.45776860810436
7		GWW		104.39367125847812
8		MTD		377.68426641492283
9		ULTA		199.46531114179035
10		CHTR		285.18271331878714

Top 10 Highest Average Closes

Query 38 : X | Query 41 : X | Query 42 : X | Query 43 : X | Query 44 : X | Query 45 : X | Query 46 : X

```
1 SELECT *
2 FROM "AwsDataCatalog"."sp500-db"."ticker-average-close"
3 ORDER BY "avg_close_#0" DESC
4 LIMIT 10;
5
```

SQL Ln 5, Col 1

Run again Explain ↗ Cancel Clear Create ▾

Query results Query stats

Completed

Results (10)

Search rows

#	▼	ticker	▼	avg_close_#0
1		NVR		1195.7925884321892
2		BKNG		971.3967128053326
3		AZO		499.14869953959675
4		MTD		377.68426641492283
5		CHTR		285.18271331878714
6		NOW		283.1572557765297
7		TDG		262.17180780597573
8		BLK		259.94236241399716
9		EQIX		253.378540464587
10		AIG		213.8161279907072

Top 10 Lowest Average Closes

Query 38 : X | Query 41 : X | Query 42 : X | Query 43 : X | Query 44 : X | Query 45 : X | Query 46 : X

```
1 SELECT *
2 FROM "AwsDataCatalog"."sp500-db"."ticker-average-close"
3 ORDER BY "avg_close_#0" ASC
4 LIMIT 10;
5
```

SQL Ln 5, Col 1

Run again Explain ↗ Cancel Clear Create ▾

Query results Query stats

Completed

Results (10)

Search rows

#	▼	ticker	▼	avg_close_#0
1		F		4.8224108805369035
2		NI		6.060729594983678
3		HBAN		6.217043957678634
4		HPQ		6.2895089332527245
5		SMCI		6.481849456042574
6		CNP		6.525560220223145
7		CSX		6.6011419727643
8		ROL		7.088841940996719
9		HST		7.124862734917359
10		WRB		7.349869291113331

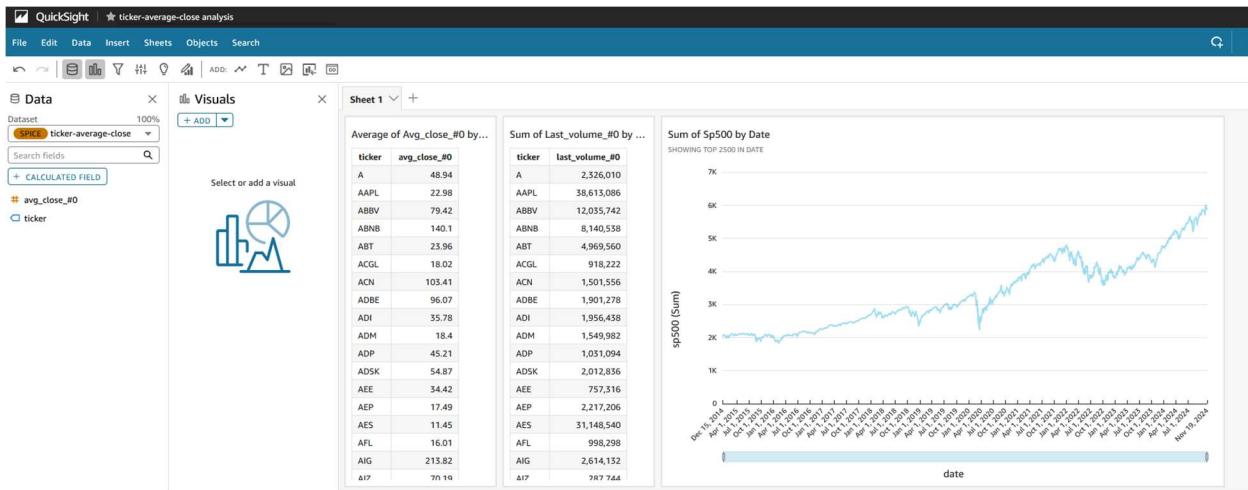
Milestone 7

Milestone #7

Group I Lei Wu & Edward Brown

Dashboard

Dashboard

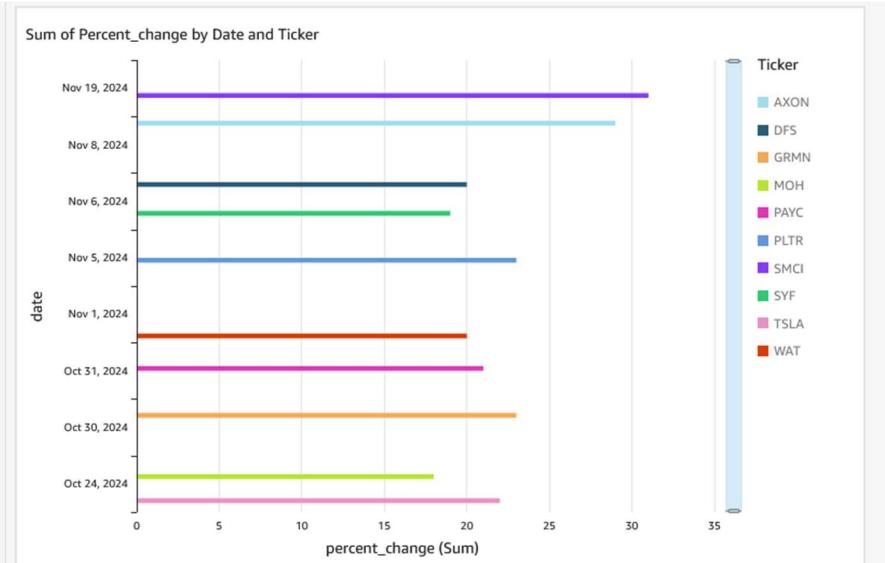


This portion of the dashboard shows us actionable insights.

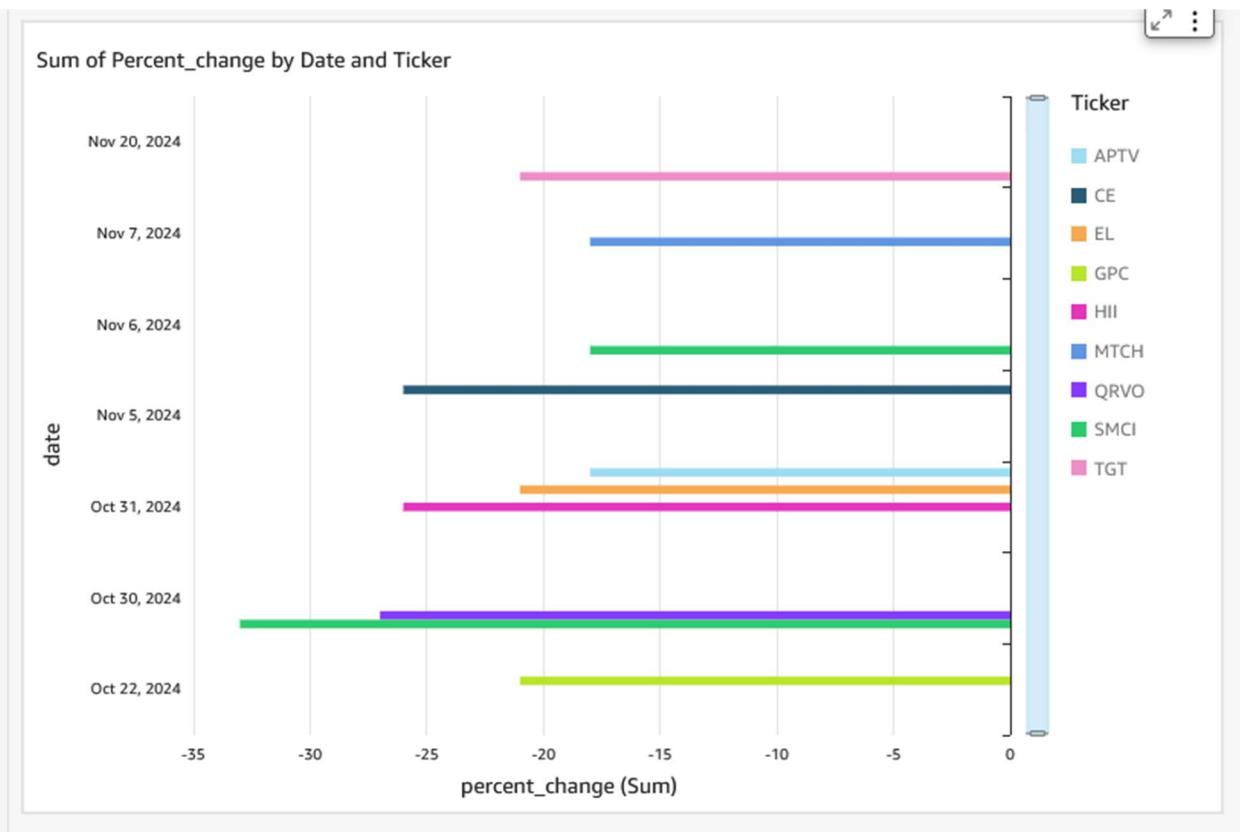
S&P 500 Chart: shows us macro trends about the index and stock market over time. We can identify the directionality of the market. We can drill down to particular hours, or drill up to specific weeks to analyze market performance during specific times.

Sum of Last Volume: Shows us the volume traded of a particular ticker during the previous days trading session. We can sort by Ascending to see the stocks that are the least traded, and we can sort by Descending to see the most traded stocks in the index.

Average Close: Shows us the average close of a particular stock over time. This allows the user to compare current closing price with average closing price. Possibly indicating a buy signal.



PercentChange Descending: This shows us the biggest gainers over the past month. This metric could indicate what holdings could be sellable due to being “overbought”. This KPI also indicates what date the percent change peaked.



PercentChange Ascending: This KPI shows us what stocks were the top 10 “biggest losers” of the past month. This could be used to identify which stocks may be oversold, and now present an attractive buying opportunity.

Conclusion

Conclusion:

Being able to interact with such a large dataset in a detailed way is incredibly powerful. Being able to see the volume of trading of a particular stock really put it into perspective for me how much of a juggernaut Nvidia continues to be. Trading nearly 100 million more shares than the next most traded stock in the index.

I also think it was really useful to see the top gainers and losers. Seeing the top gainers like Palantir and Tesla gaining such substantial amounts was eye opening to their volatility. What was even more surprising was seeing that Nvidia wasn't a top gainer, meaning its relatively stable at its price.

Overall, financial markets are irrational and as a result, the benefits of single stock investing is outweighed by the risks. However, being able to break down the most important financial index, and analyze its components, allows for a greater deal of insight into the components and how they fit together.