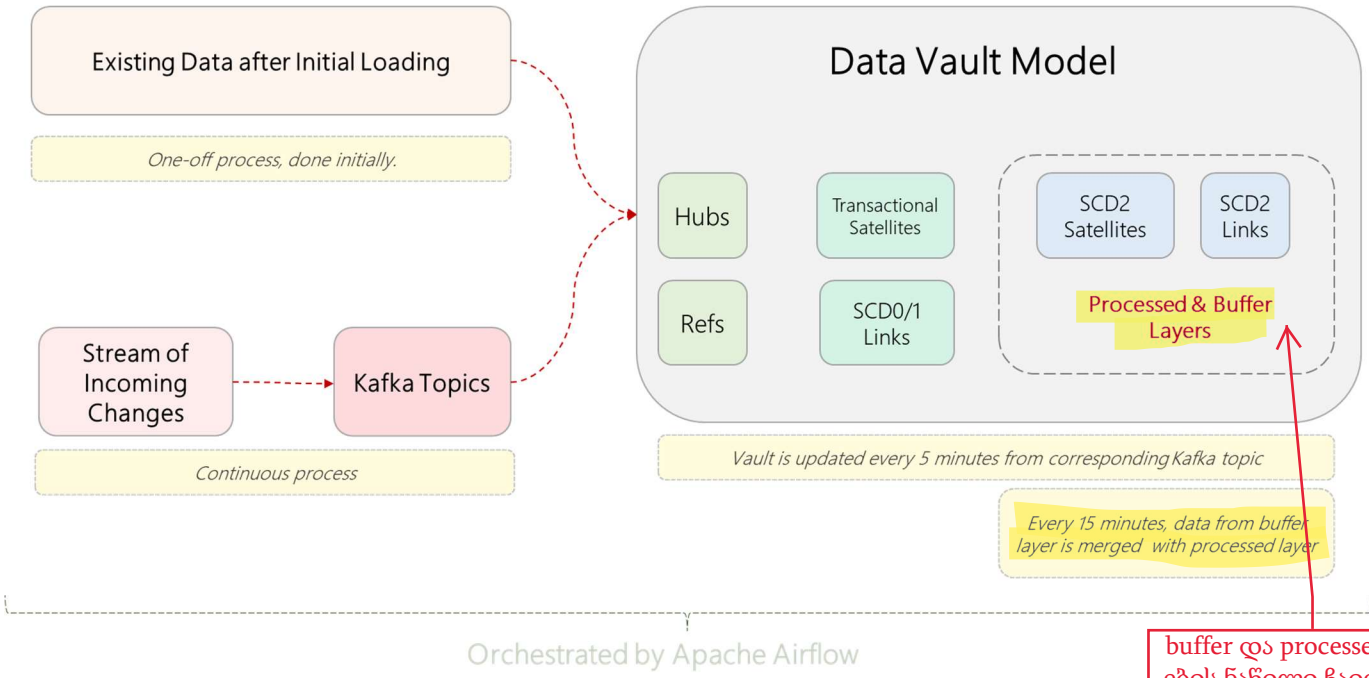
Final Project - Near Real-Time Data Vault 2.0

ფინალური პროექტის სახით, თქვენი მიზანი მცირე ზომის ცენტრალური მონაცემთა საცავისა და შესაბამისი pipeline-ების შექმნაა data vault 2.0 მოდელით. კურსის განმავლობაში განვლილი ტექნოლოგიების მიხედვით, დაგჭირდებათ მოცემული dataset-ების stream-ად ქცევა, near-real time ან micro-batch-ური სახის პროცესინგი, ტრანსფორმაციები და, რა თქმა უნდა, მათი ორკესტრაცია. ქვემოთ დიაგრამაზე მოცემულია ის ზოგადი data flow, რომელიც თქვენ უნდა ააწყოთ და საკმაოდ დეტალურად არის განხილული თითოეული ნაბიჯი. დამატებითი კითხვები შეგიძლიათ ნებისმიერ დროს მოგვწეროთ classroom-ში ან მეილზე. საჭიროების შემთხვევაში, შეგვიძლია მოკლე ონლაინ შეხვედრაც დავნიშნოთ.

Data Flow



ამ დოკუმენტში დეტალურადაა აღწერილი თითოეული საფეხური.

Dataset

დავიწყოთ dataset-ის მიმოხილვით. მოწოდებული dataset რამდენიმე მცირე ან საშუალო ზომის csv ფაილისაგან შედგება, რომელიც airbnb-ის საჯარო მონაცემებს ეყრდნობა: შინაარსობრივად, მოცემული გაქვთ მონაცემები airbnb-ის ნიდერნალდელი host-ების, მათი property-ების (listing-ების) და listing-ების შეფასებები (reviews). მონაცემები მეტწილად რეალურია, თუმცა ტრანზაქციებისა და change-ების ნაწილი ჩვენ მიერ არის გენერირებული/მოდიფიცირებული.

მონაცემები ორ ტიპად შეგვიძლია დავყოთ: არსებული მონაცემები 7 სექტემბრის მდგომარეობით (რასაც existing data-ს ვეძახით) და ტრანზაქციები 8-10 სექტემბრის განმავლობაში. წესით ტრანზაქციებში მხოლოდ ცვილებები უნდა იყოს, მაგრამ შეიძლება ეს პირობა აუცილებლად არ სრულდებოდეს და ტრანზაქცია existing-ის იდენტური იყოს. ქვემოთ უფრო კონკრეტულადაა ამის შესახებ საუბარი.

თითოეულ csv-ის ფაილის დასახელებიდან შეგიძლიათ გაარჩიოთ, რომელი ტიპის კონტენტია შიგნით. უფრო კონკრეტულად, ზემოთჩამოთვლილი ტიპის მონაცემები მოცემული გაქვთ შემდეგ ფაილებად:

1. **Hosts:**

   hosts_existing_7_sep_2022.csv ფაილში მოცემულია host-ების შესახებ existing მონაცემები 7 სექტემბრის მდგომარეობით, ხოლო *hosts_transactions_8_10_sep_2022.csv* ფაილში მოცემულია ტრანზაქციები ამ host-ების შესახებ. transaction_datetime ველში ჩაწერილია ის თარიღი და დრო, როდესაც ეს ტრანზაქცია მოხდა. გაითვალისწინეთ, რომ ერთი დღის განმავლობაში host-ზე რამდენიმე ტრანზაქცია შეიძლება მოვიდეს განსხვავებული transaction_datetime-ით. ტრანზაქციებში ყველა host-ზე არაა ინფორმაცია. ჩათვალეთ, რომ თუკი host-ზე ტრანზაქცია არ მომხდარა, ეს ნიშნავს, რომ host-ზე ცვლილება არ დაფიქსირებულა. თუმცა, თუკი host-ზე ტრანზაქციის ჩანაწერი გვაქვს, ეს არ ნიშნავს, რომ აუცილებლად ცვლილებაა. ცვლილებაა არის თუ არა - ეს თქვენ უნდა გაარკვიოთ change data capture-ის პროცესში.

2. **Listings:**

   listings_existing_7_sep_2022.csv ფაილში მოცემულია იმ უძრავი ქონებების მონაცემები 7 სექტემბრის მდგომარეობით, რომლებიც ზემოთაღნიშნულ host-ებს ეკუთვნით. *listings_transactions_8_10_sep_2022.csv* ფაილში მოცემულია ცვლილებების ტრანზაქციები ამ listing-ების შესახებ. აქაც, ტრანზაქციები

ზუსტად იგივენაირი პრინციპით გენერირდება, როგორც ეს host-ების შემთხვევაში ხდება.

3. **Listing to Host Mapping:**

listing_to_host ფაილებში მოცემულია კავშირები host-სა და უძრავ ქონებას შორის: ანუ რომელი უძრავი ქონება რომელ პიროვნებას ეკუთვნის. შეიძლება, რომ ერთ host-ს რამდენიმე listing ჰქონდეს. ზემოთ აღწერილის მსგავსად, *listing_to_host_mapping_existing_7_sep_2022.csv* ფაილში existing data არის მოცემული, ხოლო *listing_to_host_mapping_transactions _8_10_sep_2022.csv* ფაილში - ცვლილებები. ცვლილებებში იგულისხმება ის, რომ შეიძლება ახალი host დარეგისტრირდეს საიტზე და ახალი listing-ები დაამატოს, ან არსებულმა host-მა რომელიმე listing წაშალოს. listing-ს რომ host შეეცვალოს, ასეთი ცვლილება არ შეგხვდებათ. ასევე, ამ შემთხვევაში ჩავთვალოთ, რომ გარანტირებულია, რომ ამ ტრანზაქციებში მხოლოდ და მხოლოდ change-ებია.

4. **Reviews:**

reviews_8_10_sep_2022.csv მოცემული გაქვთ ასევე user-ების (სტუმრების, და არა host-ების, ცხადია) შეფასებები იმ listing-ებზე, რომლებსაც ისინი სტუმრობდნენ. გაითვალისწინეთ, რომ აქ existing მონაცემები არ გაქვთ და აღნიშნულ ფაილში მხოლოდ ის შეფასებებია, რომლებიც 8-10 სექტემბრის განმავლობაში გაკეთდა.

მოცემული dataset-ის შინაარსობრივი მიმოხილვა სულ ეს არის. რაც შეეხება იმას, თუ კონკრეტულად რა ინფორმაცია/ველები გაქვთ მოწოდებული თითოეულ ზემოთაღწერილ ჯგუფზე, გირჩევთ, რომ გადახედოთ csv ფაილებს: ველებს საკმაოდ ლოგიკური სახელები ჰქვიათ და მონაცემებიდანაც მიხვდებით, რას წარმოადგენს თითოეული სვეტი. თუმცა, ტრადიციულად, ბუნდოვანი ველები შეგვიძლია classroom-ზე ან მეილზეც გავარჩიოთ.

გაითვალისწინეთ, რომ existing და transactions ფაილებში ერთი და იგივე ველები გაქვთ, გარდა შემდეგი ტექნიკური ველებისა: მხოლოდ existing-ში შეგხვდებათ ველი valid_as_of, რომელიც აღნიშნავს, რა თარიღისა და დროისათვის არის ეს ჩანაწერი მიმდინარე. მხოლოდ transactions ფაილში შეგხვდებათ ველი transaction_datetime, რომლის მნიშვნელობაც ზევით აღვწერეთ.

Step #1 - Create Data Vault Model

როგორც წესი, პირველი, რასაც მონაცემთა საცავის შექმნის პროექტის განმავლობაში ვაკეთებთ, არის საცავის არქიტექტურისა და მოდელის მოფიქრება. ამ ამოცანის ფარგლებში, აუცილებელი მოთხოვნაა, რომ თქვენი საცავის არქიტექტურა data vault-ის მოდელის მიხედვით იყოს აგებული. ამისათვის, გირჩევთ, რომ თვალი გადაავლოთ მეორე კვირის სლაიდებს Data Warehousing-ის შესახებ: Data Warehousing Crash Course

რეალურად, არ არსებობს მაინცდამაინც ერთადერთი სწორი მოდელი. ამ პროექტის განმავლობაში არ ვითხოვთ, რომ შექმნათ იდეალური data vault. მთავარია, უხეში შეცდომები არ იყოს დაშვებული და დაცული იყოს შემდეგი მინიმალური მოთხოვნები:

- უნდა შექმნათ ცნობარი თქვენი მონაცემთა საცავის წყარო სისტემების შესახებ. ამ ცნობარს დავარქვათ სახელად ref_source_systems, სადაც იქნება სულ ორი ველი - id და source_system_name. ამ პროექტში, მართალია, ერთადერთი source გვაქვს (დავარქვათ airbnb netherlands), თუმცა პოტენციურად, ბევრი სორსი შეიძლება არსებობდეს და დავიჭიროთ თადარიგი. შეგიძლიათ, ამ ცხრილის ან ველების დასახელებები სურვილისამებრ შეცვალოთ, თუმცა მსგავსი დანიშნულების ცხრილი აუცილებლად უნდა გქონდეთ.

- უნდა გქონდეთ მინიმუმ ორი hub. შეგახსენებთ, რომ hub-ებში უნდა დააგენერიროთ ალტერნატიული გასაღებები, რომელსაც hash key-ებს ვეძახით ხოლმე. hash key-ის გენერაციის ფორმულა შემდეგია: უნდა აიღოთ ველები source key (ანუ ის იდენტიფიკატორი, რაც წყარო სისტემიდან მოყვება. მაგალითად, ჰოსტების შემთხვევაში host_id არის ეს ველი) და source_system_id (რომელსაც წინა პუნქტში შექმნილი ref_source_systems ცხრილიდან აიღებთ), უნდა გააკეთოთ მათი კონკატენაცია აუცილებლად რომელიმე სეპარატორით და დაჰეშოთ სასურველი ჰეშირების ალგორითმით. გირჩევდით md5-ის ან sha256-ის გამოყენებას. hash key-ები იყოს string ან binary ტიპის. გაითვალისწინეთ, რომ თქვენი მონაცემთა საცავის ყველა სხვა ცხრილში source key-ების მაგივრად თქვენ მიერ დაგენერირებული hash key უნდა გამოიყენოთ. მაგალითად, hub-ის გარდა ყველა სხვა ცხრილში, სადაც host_id გხვდებოდათ, თქვენ უნდა გამოიყენოთ host_hash_key.

- უნდა გქონდეთ 3 მაინც reference ტიპის ცხრილი. შეგახსენებთ, რომ ესენი პატარა, არაისტორიული, ცნობარის დატვირთვის მქონე ცხრილებია, სადაც ძირითადად ორი ველია - უნიკალური იდენტიფიკატორი და შესაბამისი მნიშვნელობა. მაგალითები:

ref_country

id	country_name_geo	country_iso_code	country_name_eng
1	საქართველო	GEO	Georgia
2	ამერიკის შეერთებული შტატები	USA	United States of America
3	გერმანია	DEU	Germany

ref_property_type

id	property_type
1	Private room
2	Entire loft

- უნდა გქონდეთ 4 მაინც satellite. სატელიტებიც შეიძლება დროში სხვადასხვანაირად იმართებოდეს. კერძოდ:
    - ზოგ სატელიტში შეიძლება update არასდროს ხდებოდეს და საერთოდ არ იმართებოდეს ისტორიულად (SCD Type 0-ის მსგავსად. ეს ტიპები და მაგალითები შეგიძლიათ იხილოთ ზემოთ დალინკულ სლაიდებში ან გუგლში 😊);
    - ზოგი სატელიტი შეიძლება იყოს ტრანზაქციული სახის: მხოლოდ append-ით ემატებოდეს event-ები, რომლებიც არასდროს დააფდეითდება. ისტორიულობის სამართავად გვაქვს ერთი თარიღობრივი ველი - message/event/transaction_timestamp, სადაც უბრალოდ event-ის მოხდენის დრო წერია. აუცილებელი მოთხოვნაა, რომ მინიმუმ ერთი ასეთი მაინც სატელიტი გქონდეთ. ცხადია, სწორად უნდა დააიდენტიფიციროთ, მოცემული დატასეტიდან რომელი მონაცემებისთვის იქნებოდა ლოგიკური ამ მოდელის გამოყენება;
    - ზოგი სატელიტი შეიძლება იყოს SCD Type 2-ის ტიპის და მონაცემებს კუმშავდეს valid_from/valid_to თარიღებს შორის (იგივე effective_start_date/effective_end_date). ამ შემთხვევაში გჭირდებათ

change detection (change data capture) პროცესის აწყობა. ასეთი ტიპის სატელიტები უნდა გქონდეთ მინიმუმ 2 ცალი.

აქამდე მოცემული დავალებებიდან გამომდინარე, change data capture პროცესი თქვენთვის მეტნაკლებად ნაცნობი უნდა იყოს. პრინციპი აქაც იგივეა, თუმცა სხვაობა არის შემდეგში:

- აუცილებელი არაა, რამდენიმე დღის change detection ერთდროულად გააკეთოთ;

- აქ არ იქნება ე. წ. multi-row change detection და ერთი entity-სათვის (მაგალითად, host-ისათვის) ყოველთვის ერთი ერთი row უნდა შევადაროთ ერთ row-ს;

- არ გვჭირდება is_first და is_last ველები;

- და მთავარი: effective start/end აქამდე შეგხვედრიათ 1 დღის სიზუსტით. ამ ამოცანაში კი, SCD Type 2-ის შემთხვევაში, ჩანაწერის ვალიდურობის (ეფექტურობის) პერიოდი უნდა იყოს მილიწამამდე სიზუსტით, ანუ timestamp ფორმატში. უკეთ გასაგებად განვიხილოთ შემდეგი მაგალითი, სადაც ცვლილება მოხდა:

Existing Data:

id	content	valid_from	valid_to	comment
1	A	7-Sep-2022 10:15:30.400	20-Sep-2022 22:40:13.420	აღარაა ვალიდური
1	B	20-Sep-2022 22:40:13.420	null	მიმდინარე
2	T	10-Jan-2022 14:24:43.256	null	მიმდინარე

Transaction Data:

id	content	transaction_datetime	comment
1	C	25-Sep-2022 13:09:55.514	ცვლილება
2	T	25-Sep-2022 21:23:34.000	უცვლელი

Existing Data After Change Detection:

id	content	valid_from	valid_to	comment
1	A	7-Sep-2022 10:15:30.400	20-Sep-2022 22:40:13.420	აღარაა ვალიდური
1	B	20-Sep-2022 22:40:13.420	25-Sep-2022 13:09:55.514	აღარაა ვალიდური
1	C	25-Sep-2022 13:09:55.514	null	მიმდინარე
2	T	10-Jan-2022 14:24:43.256	null	მიმდინარე

ყურადღება მიაქციეთ, რომ:

1. **მიმდინარე ჩანაწერებს valid_to-ში აუცილებლად null უწერიათ;**
2. **მიმდინარე ჩანაწერის valid_from და მისი უშუალო წინამორბედის valid_to იდენტურია.**

- და ბოლოს, ასევე უნდა გქონდეთ ერთი მაინც link ცხრილი - ორ entity-ს შორის კავშირების ცხრილი (მაგალითად, listing to host კავშირის აღმწერი. ლინკები შეიძლება იყოს საერთოდ ისტორიული, ან შეიძლება SCD Type 2-ის მსგავსად, valid_from/valid_to თარიღების ჭრილში იმართებოდეს.

გაითვალისწინეთ, რომ მოწოდებული დატასეტიდან ყველა ველი საჭიროა და უნდა გამოიყენოთ მოდელში. ტექნიკური ველები შეგიძლიათ დაამატოთ ისე, როგორც საჭიროდ ჩათვლით.

მოდელის დიაგრამის დახატვა საჭირო არ არის.

Step #2 - Initial Loading

ლოგიკური მოდელის შექმნის შემდეგ, ფიზიკურად უნდა შექმნათ ეს ცხრილები და საწყისი მონაცემებით შეავსოთ. ამ პროცესს initial loading-ს ვუწოდებთ: უნდა გამოიყენოთ დატასეტის ყველა existing მონაცემების ფაილი და თქვენ მიერ მოდელირებულ სტრუქტურებში ერთჯერადად ჩაწეროთ. ამ დროს, SCD2 ტიპის ცხრილების valid_from/valid_to თარიღებში ორივეგან ჩაწერეთ null. initial loading-ი სულ ესაა. ეს პროცესი ერთჯერადად ხელით გასაშვებია. შეგიძლიათ, გამოიყენოთ როგორც სპარკი, ასევე პანდასი. არჩევანი თქვენზეა. გამოიყენეთ ასევე Airflow: ააწყვეთ initial loading-ის ერთი DAG, სადაც, თქვენი მოდელიდან გამომდინარე, მაქსიმალურად დააპარელებთ ამ ცხრილების შევსებას.

Steps #3 and #4 - Streaming and Incremental Loading

initial-ის შემდეგი ნაბიჯია incremental loading-ის აწყობა. ამ პროექტის ფარგლებში incremental პროცესში იგულისხმება, რომ მონაცემები stream-ის სახით უნდა წაიკითხოთ, საბოლოო მიზანი კი ამ მონაცემების micro batch-ებად თქვენს data vault-ში ჩაწერაა.

დავიწყოთ იმით, რომ stream-ებიც თქვენ უნდა ააწყოთ Kafka-ს გამოყენებით შემდეგნაირად:

- dataset-ის ყველა transactions ფაილისათვის უნდა შექმნათ შესაბამისი topic-ები Kafka-ში. ერთი ფაილი აუცილებლად ერთ ტოპიკში უნდა იწერებოდეს და პირიქით, ერთ ტოპიკში - ერთი ფაილი.
- თითოეული ფაილიდან პარალელურად, მაგრამ ფაილის დონეზე ქრონოლოგიურად (transactional date-ის მიხედვით) კითხულობთ სათითაო row-ს და მიახლოებით 20-40 წამის შუალედებით აგდებთ შესაბამის ტოპიკში. ეს დრო, სასურველია, კონფიგურებადი იყოს. მესიჯს ზუსტად იგივე ველები უნდა ჰქონდეს, როგორც ეს csv ფაილშია. ეს არ ნიშნავს იმას, რომ data vault-შიც ეს მესიჯი იდენტური სტრუქტურის ცხრილში უნდა ჩაწეროთ. პირიქით, უფრო მეტად მოსალოდნელია, რომ ამ მესიჯით ერთზე მეტი ცხრილი ივსებოდეს.

stream-ის გენერირების ნაწილი სულ ეს არის. ამ ეტაპისთვისაც შექმენით ცალკე DAG Airflow-ში.

სთრიმებთან ერთად, ასევე პარალელურად უნდა მუშაობდეს პროცესი kafta_to_vault, რომელიც 5 წუთში ერთხელ გაეშვება, Kafka-ს ყველა ტოპიკიდან ერთდროულად მოაგროვებს ამ ტოპიკიდან უკანასკნელი წაკითხვის შემდეგ ჩავარდნილ მესიჯებს და data vault-ში შესაბამის ცხრილებში გადაიტანს. ეს ინტერვალიც კონფიგურირებადი უნდა იყოს. აუცილებელი არაა, მაგრამ ამ პროცესში შეგიძლიათ შუალედური ცხრილები გამოიყენოთ.

დასასრულს, მონაცემების Kafka-დან SCD Type 2 ცხრილებში გადატანის შემთხვევაში, უნდა დაიცვათ შემდეგი არქიტექტურა:

- data vault-ის SCD Type 2 ცხრილი უნდა შედგებოდეს ორი layer-ისაგან: buffer & processed. ცხრილშივე უნდა გქონდეთ ტექნიკური ველი, რომელშიც ეწერება, რომელი layer-ის მონაცემია;

buffer და processed layer-ების ნაწილი ჩაითვლება extension-ად, აღარაა სავალდებულო

Kafka-დან წაკითხული მონაცემები პირველ რიგში buffer layer-ზე უნდა ხვდებოდნენ ყველანაირი change detection-ის გარეშე, valid_from-ში უწერთ transaction_datetime-ის მნიშვნელობას, valid_to-ში - null-ს. მაგალითად:

Message in Kafka:

```
{
        "hero_id": 5595,
        "hero_name": "Hiro Nakamura",
        "country": "Japan",
        "superpowers": null,
        “transaction_datetime”: “08-Nov-2022 20:42:24.444”
}
```

Sat_Hero table in Data Vault:

hero_hash_key	name	country	superpower	valid_from	valid_to	layer
5ced8dba	Dr. Robert Bruce Banner	USA	Superhuman strength, stamina	13-Jun-2008 12:00:00.000	null	processed
ff9c4a60ge	Hiro Nakamura	Japan	None	25-Sep-2006 22:00:00.000	null	processed
ff9c4a60ge	Hiro Nakamura	Japan	Space-time manipulation	08-Nov-2022 20:42:24.444	null	buffer

- და ბოლოს, გარკვეული კონფიგურირებადი შუალედით (დავუშვათ, ნაგულისხმევად 15 წუთში ერთხელ), buffer და processed layer-ებს შორის უნდა ხდებოდეს change detection პროცესი და ეს layer-ების საჭიროებისამებრ ი-merge-ებოდნენ. ზემოთ აღწერილი ცხრილის merger-ის შემდეგ უნდა მივიღოთ შემდეგი სიტუაცია:

Merged Sat_Hero table in Data Vault:

hero_hash_key	name	country	superpower	valid_from	valid_to	layer
5ced8dba	Dr. Robert Bruce Banner	USA	Superhuman strength, stamina	13-Jun-2008 12:00:00.000	null	processed
ff9c4a60ge	Hiro Nakamura	Japan	None	25-Sep-2006 22:00:00.000	08-Nov-2022 20:42:24.444	processed
ff9c4a60ge	Hiro Nakamura	Japan	Space-time manipulation	08-Nov-2022 20:42:24.444	null	processed

ამ შემთხვევაში, თუ Hiro Nakamura-ზე არანაირი ცვლილება არ დაფიქსირდებოდა, მაშინ აღნიშნულ ცხრილში გვექნებოდა ერთადერთი ჩანაწერი:

ff9c4a60ge	Hiro Nakamura	Japan	None	25-Sep-2006 22:00:00.000	null	processed

აღნიშნული merger-ის შემდეგ, იმ ჩანაწერებისათვის, რომლებიც merger-ში მონაწილებას იღებდნენ, buffer უნდა ,,გაუსუფთავდეს" და ყველა ჩანაწერი processed სტატუსში უნდა გადავიდეს. თუკი ამ merger-ის დროს hive-ის ცხრილის მცირეხნიანი downtime გექნებათ (არ იქნება ცხრილი ხელმისაწვდომი user-ისათვის), ეს დასაშვებია.

ეს პროცესიც უნდა დააორკესტიროთ Airflow-ის გამოყენებით. ამ ნაწილში უნდა გამოიყენოთ PySpark.

Extensions:

პროექტს შეგიძლიათ დაუწეროთ extension-ები საკუთარი შეხედულებისამებრ, ოღონდ ეს იდეები წინასწარ უნდა შეგვითანხმოთ.

ზოგადად, extension-ები შეიძლება, რომ:

- აგვარდებდეს data quality management-ის საკითხებს. მაგალითად, შეგიძლიათ, დაწეროთ job-ები, რომლებიც შეამოწმებს, თქვენი data vault-ი რამდენად სწორად არის შევსებული და data loss ხომ არ გაქვთ სადმე;
- იყოს იმპლემენტირებული ალტერნატიული tool-ებით. მაგალითად, შეგიძლიათ data vault-ის შევსების ნაწილში გამოიყენოთ delta lake;
- მოიცავდეს ai-ის ნაწილსაც. მაგალითად, შეგიძლიათ ჩაატაროთ streaming review-ების სენტიმენტ ანალიზი რეალურ ან თითქმის რეალურ დროში. სენტიმენტ ანალიზი გულისხმობს მომხმარებლის კომენტარის განწყობის დადგენას - პოზიტიური შეფასებაა ეს, ნეიტრალური თუ საერთოდაც უარყოფითი. ამ ნაწილში დაგჭირდებათ nlp - natural language processing ბიბლიოთეკები. ასეთი ბიბლიოთეკებს პითონზე იოლად ნახავთ და შეგიძლიათ, სასურველი მათგანი გამოიყენოთ.

extension-ების დაწერის შემთხვევაში, საბოლოოდ შეიძლება დაიმსახუროთ პროექტის მაქსიმალური ქულის 110%-120%.

გირჩევთ, დროულად დაიწყოთ პროექტზე მუშაობა. შეკითხვები ნებისმიერ დროს გავიაროთ classroom-ზე ან mail-ით. გისურვებთ წარმატებებს! 😊