**UZDEVUMS DATU ARHITEKTA VAKANCEI**

**Scenārijs**

Tu esi datu inženieris/arhitekts nedzīvības apdrošināšanas uzņēmumā. Uzņēmumam nepieciešams izveidot datu plūsmu no izejas failiem līdz analītiskajam modeļa slānim, ko izmantos biznesa analītiķi Power BI vidē.

**Pieejamie dati**

Pieejami dati ir simulēti dati par apdrošināšanas biznesu, dati nereprezentē reālu kompāniju, bet gan ir izmantojami tikai datu arhitekta vakances uzdevumā. Csv failu loģika reprezentē tabulas pirmavota sistēmas loģiku:

Polises dati – policies.csv;

Piedāvājuma dati – offers.csv;

Atlīdzību dati – claims.csv;

Klientu dati – customers.csv;

**Uzdevums**

Repozitorijs

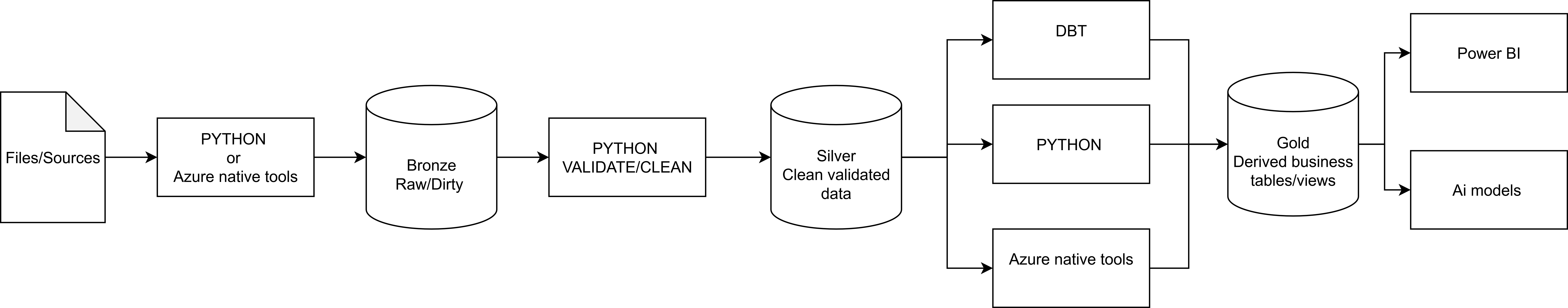
1. Izveidot Git repozitoriju (*Bitbucket* vai *GitHub* vai cits, nav būtiski).
2. Izstrādes kodu organizēt pa mapēm (dbt, pipelines, docs).

GenAI

1. Uzdevumos drīkst izmantot LLM modeļus dokumentējot tā izmantošanu un rezultātus.

Datu ielāde un transformācija (ETL)

1. Ar dbt (lokāli) izveidot transformācijas pēc “Medallion” arhitektūras principa:
   1. Faktstabula fact\_claims (claims + policies join);
   2. Dimensijas dim\_customers;
   3. Kādas jaunas dimensijas var atvasināt no customers tabulas?;  
      ATBILDE: dim\_location(city/reagion etc.) dim\_segment, dim\_age\_gender(age, gender, birth month/seasson), dim\_creation\_dates( years , months seasons)
   4. Dimensijas dim\_products no policies;
   5. Kādas jaunas dimensijas var atvasināt no policies tabulas?;  
      ATBILDE: dim\_age(policy age),dim\_sales\_chanals, dim\_policy\_statuses
2. Izveidot biznesa loģiku:
   1. Aprēķināt nopelnīto prēmiju earned\_premium katrai polisei  
      Definīcija - Nopelnītā prēmija ir tā parakstītās prēmijas daļa(written\_premium), kas atbilst riskam, kas jau ir “iztērēts” jeb apdrošināšanas seguma periodam, kas līdz attiecīgajam datumam ir beidzies;
   2. Aprēķināt Claim Ratio = Sum(claim\_amount) / Sum(earned\_premium) pēc produktiem;
   3. Uzrakstīt optimizētu SQL, kas aprēķina vidējo prēmiju un vidējo atlīdzību pa klientu vecuma grupām;
   4. Apdrošināšanā ir svarīgi uzskaitīt objektu/polišu/klientu skaitus. Kāds ir veiksmīgākais risinājums fakta izveidē, kur saskaitīšanas loģika var atšķirties pa produktiem? Demonstrē piedāvāto risinājumu vizuālizācijas vai teksta veidā, neveidojot skriptu.  
      ATBILDE: Ideālā gadījumā dažāda tipa biznesa objekti tiek nodalīti tabulās, kas nesaturētu dublikātus, pielīdzinot to skaitu rindu skaitam tabulās. Situācijās kur saskaitīšanas loģika var atšķirties pa produktiem ieteicams loģikas rezultātu saglabāt klasifikatoru vērtībās, kuras skaitīt būs daudz ātrāk un kuru unikālais skaits būs fiksēts to klasifikatoru vērtību tabulā. Un protams kur tas nav iespējams vai nav praktiski, nāksies pielietot DISTINCT CASE IIF un COUNT skatos/materializētās tabulās atkarībā no datu pielietojuma.
3. Python integrācija:
   1. Python skripts, kas validē datus loģiskai biznesa būtībai (piemēram, vai nav dublētu policy\_id, vai claim\_date < end\_date);
   2. Python skripts kurš veic unikalitātes identifikāciju polišu piedāvājumu datos (offers). Izveidot šādas dimensijas, pēc rezultāta ieguves izanalizēt vai izveidotais risinājums ir efektīvs:
      1. UniqeDay – Pēdējais unikālais piedāvājums dienas ietvaros (24h)
      2. UniqueWeek - Pēdējais unikālais piedāvājums nedēļas ietvaros (7dienas)
      3. UniqueMonth- Pēdējais unikālais piedāvājums mēneša ietvaros(30dienas);
   3. Vai Python izveidotais algoritms unikalitātes aprēķiniem ir efektīvākais veids kā izveidot šos parametrus? Vai ir kāda efektīva alternatīva šo parametru veidošanai?  
      ATBILDE: Atbilde ir atkarīga no datu apjoma un izmantotā datu apstrādes ietvara. Lielākajā daļā gadījumu šāda parametra veidošana būtu ātrāka datubāzes līmenī izmantojot row\_number un group by. Pie ļoti lieliem datu apjomiem, datu apstrādes ietvari kā Spark var apsteigt datu apstrādi datubāzē, tikai tapēc ka Spark ietvaram var būt priekšrocība horizontālā mērogošanā un spējā kārtošanas operācijas izpildīt uz vairāk serveriem paralēli.
4. Ar brīvas izvēles rīkiem/metodēm validēt *customer* tabulas biznesa loģiku pret polises un atlīdzību tabulām. Kādas datu kvalitātes vājības ir novērojamas *customer* tabulā;
5. CI/CD (*Azure pipelines*)
   1. Izveidot vizuālā formā datu plūsmas “pipeline” shēmu (nav jāizpilda DWH vidē);



1. Power BI Semantic Model
   1. Izveidot vienkāršu Tabular Editor TOM modeli, kas:
      1. sasaista *claims* ar polišu dimensijām;
      2. izveido faktu mērījumus *Claim Ratio, Total Premiums, Total Claims, Total Earned Premium;*
   2. nodemonstrēt modeli ar mock-up Power BI;

**Veidojot uzdevumu jāņem vērā standarts (skatīt nākamajā lapā):**

**Column naming:**

|  |  |  |
| --- | --- | --- |
| **Object** | **Format** | **Example** |
| Primary key ID (in dimensions) | id<table\_name> | idd\_pol\_policy |
| Foreign key ID | id<foreign\_table\_name> | idd\_wh\_date |
| Several foreign key IDs to the same dimension and where several choices and reference is needed for context | id<foreign\_table\_name>\_\_<context> | idd\_wh\_date\_\_policy\_created  idd\_wh\_date\_\_policy\_cov\_start  idd\_wh\_date\_\_accounting |
| Dates | d\_<name> | d\_created |
| Datetime | dt\_<name> | dt\_updated |
| Time | t\_<name> | t\_issue |
| Amounts | amt\_<name>\_<additional context>\_<[group]> | amt\_invoice  amt\_invoice\_group |
| Quantities | qty\_<name>\_<additional context> | qty\_invoice  qty\_invoice\_ytd |
| Counts | cnt\_<name>\_<additional context> | cnt\_invoices  cnt\_days\_approved\_to\_receive |
| Rates | exchange\_rate | exchange\_rate |
| Attributes (Codes) | <attribute name>\_code | item\_code |
| Attributes (Names) | <attribute name>\_name | item\_name |
| Combined attributes | <attribute name>\_and\_<attribute\_name> | item\_code\_and\_name |
| Attributes (Descriptions) | <attribute\_name>\_descr | item\_descr |
| Attributes (Flags) | is\_<attribute> | is\_delivered |
| Attributes (Descriptive information, labels) | <attribute\_descriptive\_information> | contact\_person\_phone\_name |

**Schemas naming:**

|  |  |  |
| --- | --- | --- |
| **Object** | **Example** | **Description** |
| dwh |  | main BI database schema. Holds all dimensions and facts used for analysis |
| dwh\_conf |  | schema that holds configuration (K) tables (parameters, mappings, etc.) and manually entered info that is not otherwise recoverable |
| dwh\_hist |  | observed to store historic ids from source system to ids generated in DWH, $H table name? |
| staging\_% | staging\_system | schema that holds stage data from % system |
| staging\_%\_\_hist | staging\_system\_\_hist | schema that holds historic stage data from % system. Table names J$? |
| key\_% |  | KEY tables that store key mapping for table IDD mapping |