УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Сара Живковић 1071/2023

ДИЗАЈН И ИМПЛЕМЕНТАЦИЈА АНАЛИТИЧКЕ ПЛАТФОРМЕ ЗАСНОВАНЕ НА АРХИТЕКТУРИ У ОБЛАКУ НАД ПОДАЦИМА СИСТЕМА ЗА УЧЕЊЕ НА ДАЉИНУ

мастер рад

Београд, 2026.

Ментор:

проф. др Александар Картељ

Чланови комисије:

проф. др Саша Малков

доц. др Мирко Спасић

Датум одбране:

Наслов мастер рада: Дизајн и имплементација аналитичке платформе засноване на архитектури у облаку над подацима система за учење на даљину

Резиме: У раду је пројектована и имплементирана аналитичка платформа у облаку која обухвата комплетан ток обраде података из домена учења на даљину. Платформа обрађује податке о студентима, инструкторима, курсевима које студенти могу уписати и похађати и њиховом садржају. Због недоступности продукционог извора формиран је контролисан синтетички скуп података који генерише вишегодишњу историју, почев од 2024. године, дневно пристизање нових догађаја и промене података кроз време.

Решење је засновано на `ELT` приступу. За пружаоца услуга у облаку изабран је `Amazon Web Services` (`AWS`). Изворне датотеке чувају се у `Amazon S3` објектном складишту и у дневним партицијама учитавају у `ClickHouse Cloud` базу, где се кроз више слојева припремају за анализу. Систем `Dagster` управља редоследом и праћењем обраде, док се алат `dbt` користи за трансформације, аналитичко моделовање и тестирање података. Дневно генерисање података реализовано је помоћу сервиса `AWS Lambda`, а компоненте оркестрације покренуте су у `Docker` контејнерима на инстанци `Amazon EC2`.

Формирани аналитички модел омогућава анализу ангажованости и напретка студената кроз курсеве, резултате квизова, рецензије курсева, прихода и осталих метрика. Посебна пажња посвећена је очувању историјског контекста, дневној и историјској обради, поновном покретању након грешке, спречавању дупликата и проверама квалитета у више фаза. Евалуација решења обухвата исправност и поузданост обраде, време извршавања, понашање аналитичких упита при већем оптерећењу, трошкове и захтеве одржавања.

Кључне речи: инжењерство података, рачунарство у облаку, `ELT`, димензионо моделовање, оркестрација података, колумнарно складиштење, `ClickHouse`, `dbt`, `Dagster`, `Amazon Web Services` (`AWS`), `Amazon S3`, `Amazon EC2`, `Docker`.

Садржај

1 Увод **1**

1.1 Мотивација 1

1.2 Циљ рада 1

1.3 Обим и границе рада 2

1.4 Основни појмови 3

1.5 Структура рада 15

2 Преглед коришћених алата **16**

2.1 Улоге алата у току обраде података 16

2.2 `Dagster` као алат за оркестрацију 16

2.3 `dbt` као алат за трансформацију и моделовање података 18

2.4 Окружење у облаку 19

2.5 Складиштење података: `ClickHouse Cloud` и `Amazon S3` 20

2.6 Извршно окружење: контејнери и функције без сервера 21

2.7 Помоћни алати и конфигурација 21

3 Пројектовање платформе **22**

3.1 Домен и захтеви система 22

3.2 Архитектура платформе: ток података и инфраструктура у облаку 26

3.3 Пројектовање аналитичког модела 29

3.4 Пројектовање тока обраде података 34

3.5 Квалитет, поузданост и надгледање података 37

3.6 Ограничења система 39

4 Имплементација **41**
4.1 Извршно окружење 42
4.2 Формирање и складиштење изворних података 43
4.3 Учитавање у аналитичку базу 49
4.4 Трансформација података кроз `dbt` слојеве 53
4.5 Повезивање у аутоматизовани ток 60
4.6 Валидација и надгледање 64

5 Евалуација **66**
5.1 Функционална исправност и поузданост 66
5.2 Перформансе 70
5.3 Трошкови решења 78
5.4 Ограничења 80

6 Закључак **82**

Литература **84**

Додатак: Понављање експеримента **86**
A.1 Предуслови 86
A.2 Конфигурација 86
A.3 Провера успешности 89

Глава 1

Увод

1.1 Мотивација

На познатим системима за учење на даљину сваки упис на курс, преглед садржаја, покушај решавања квиза и остављена рецензија показују како студенти уче и напредују. Посматрани заједно, ти подаци могу указати на то где студенти одустају, који курсеви дају добре резултате и како су ангажовање студената, оцене и приходи од курсева међусобно повезани. Ове информације су важне за оцену рада система и планирање будућих побољшања, али до њих се не долази само прикупљањем појединачних догађаја.

Подаци у оперативним базама организовани су пре свега за свакодневни рад система, а не за анализу већег броја повезаних података и догађаја кроз време. Зато их је потребно објединити, уједначити и међусобно повезати ради лакше анализе.

Посебан изазов представљају промене које се дешавају током времена: студент може променити претплату, цена и статус курса се такође могу изменити, а студент кроз курс временом напредује до потенцијалног завршетка. Ако се те промене не обраде пажљиво, ранији догађаји биће тумачени на основу вредности које у тренутку њиховог настанка нису важиле.

1.2 Циљ рада

Циљ рада је пројектовање и имплементација аналитичке платформе која омогућава анализу ангажованости и напретка студената, успешности курсева, резултата квизова и рецензија, као и претплатничких модела и прихода, а све на основу података са система за учење на даљину.

Пошто поузданост оваквих анализа зависи од начина на који је сама платформа изграђена, посебна пажња посвећена је одлукама које обезбеђују поузданост целог тока података, а не само повезивању изабраних технологија: очувању историјског контекста, дневној и историјској обради, поновном покретању након грешке, спречавању дупликата и провери квалитета у више фаза.

1.3 Обим и границе рада

Пошто продукциони изворни систем није био доступан, формиран је контролисан скуп података који представља рад система за учење на даљину, укључујући почетну историју, свакодневно пристизање нових догађаја и промене података кроз време. То омогућава контролу над обимом података, временским распоном података и уносом историјских промена. Контрола над скупом не значи да се подаци генеришу насумично, без икаквог правила: подаци о курсевима потичу из стварног, јавно доступног `Udemy` скупа података (наслови, категорије, цене и редослед објављивања), а понашање студената и промене претплата и цена прате правила која опонашају реалне обрасце (нпр. број регистрација временом расте, а вероватноћа да студент промени претплату расте што дуже користи платформу), уместо да буду потпуно случајни. Ова правила одређена су независно од упита и модела који се касније анализирају у Поглављу 5, тако да подаци нису прилагођени да потврде очекиване резултате. Начин генерисања детаљније је описан у Одељку 4.2.

Резултати добијени над овим синтетичким скупом зато не доказују понашање платформе у стварном продукционом окружењу, већ показују да ли предложено решење ради исправно и поуздано над контролисаним, репрезентативним скупом података.

Обим рада обухвата целокупан ток података, од изворних података до коначног слоја над којим се могу градити аналитички упити. Циљ рада није изградња комплетног продукционог система за учење на даљину, нити развој посебних апликација за пословну интелигенцију (енгл. *Business Intelligence*, `BI`).

1.4 Основни појмови

Основни принципи инжењерства података

Инжењерство података (енгл. *data engineering*) обухвата пројектовање, изградњу и одржавање система за прикупљање, складиштење, обраду и припрему података за аналитичку употребу. Његов основни циљ није сама анализа података, већ изградња поуздане инфраструктуре и процеса који обезбеђују да подаци буду доступни, тачни, правовремени и у облику погодном за даљу обраду. Као резултат таквих процеса настају скупови података које користе аналитичари, научници за податке (енгл. *data scientists*), системи за пословно извештавање и друге аналитичке апликације [1].

Област рада инжењера података може се описати кроз животни циклус података (енгл. *data lifecycle*), односно кроз низ корака које подаци пролазе од тренутка настанка до њихове аналитичке употребе. Иако се конкретна имплементација разликује у зависности од архитектуре система и пословних захтева, основне фазе тог процеса углавном су исте.

1. **Настанак и прикупљање података:** Подаци настају у различитим изворним системима, попут трансакционих база података које бележе пословне операције или веб и мобилне апликације које бележе податке о својим корисницима. Унос тих података (енгл. *data ingestion*) је први корак у циклусу.

2. **Складиштење података:** Након преузимања, подаци се смештају у складиште које омогућава њихово трајно чување и даљу обраду. Савремене платформе често користе више типова складишта, при чему избор зависи од карактеристика података и начина њихове употребе.

3. **Трансформација и моделовање података:** Подаци преузети из изворних система ретко су одмах спремни за аналитичку употребу. Често садрже дупликате, недостајуће вредности, различите формате записа или структуру прилагођену оперативним процесима. Током трансформације, такви недостаци се отклањају, подаци се повезују са другим изворима и организују према аналитичким потребама. У овој фази имплементира се и највећи део пословне логике.

4. **Испоручивање података:** Последња фаза животног циклуса односи се на омогућавање припремљених података доступним крајњим корисницима. Резултати обраде могу се користити за извештавање, истраживачке анализе, развој модела машинског учења или као улаз за друге системе.

   Поред наведених фаза, кроз читав животни циклус присутне су и активности које нису везане за једну конкретну фазу: контрола квалитета, оркестрација, управљање приступом, праћење извршавања и порекла података (енгл. *data lineage*). Циљ ових активности јесте обезбеђивање поузданог и поновљивог процеса обраде, уз могућност правовременог уочавања и отклањања грешака.

Токови обраде података

Активности животног циклуса података у пракси се повезују кроз аутоматизоване токове обраде (енгл. *data pipelines*). Ток дефинише начин преузимања и складиштења података, трансформације које се над њима извршавају, као и начин на који резултат постаје доступан за аналитичку употребу [1]. Стандард у индустрији је аутоматизација ових процеса чиме се смањује могућност људске грешке и омогућава обрада великих количина података.

Редослед извршавања појединих корака зависи од архитектуре тока обраде. Традиционални приступ заснива се на `ETL` моделу (енгл. *Extract, Transform, Load*) код кога се подаци најпре преузимају из изворних система, затим трансформишу, а тек потом учитавају у аналитичко складиште. Овај модел био је доминантан у периоду када су ресурси за складиштење и обраду били ограничени, па је циљ био да се у складиште уносе искључиво већ припремљени подаци. Развојем инфраструктуре у облаку и аналитичких база високих перформанси све чешће се примењује `ELT` модел (енгл. *Extract, Load, Transform*). Код овог приступа подаци се одмах након преузимања учитавају у складиште и тамо се и извршавају трансформације. У савременим аналитичким архитектурама често се задржава и слој сирових или минимално обрађених података. Тиме сирови подаци остају доступни у облику блиском изворном, што омогућава поновно извршавање трансформација или њихову измену без новог преузимања података из извора [1].

Токови обраде разликују се и према учесталости обраде података. **Пакетна обрада** (енгл. *batch processing*) подразумева обраду ограниченог скупа података који је прикупљен током одређеног периода. Таква обрада обично се покреће у унапред дефинисаним интервалима или када се прикупи одговарајућа количина података. Погодна је за аналитичке системе код којих подаци не морају бити доступни непосредно након настанка. За разлику од ње **обрада у реалном времену** (енгл. *stream processing*) подразумева континуирану обраду података током њиховог пристизања. Она је погодна за системе који захтевају тренутну реакцију, на пример за детекцију превара, надзор индустријских процеса или персонализацију садржаја [2].

Ток обраде најчешће се састоји из више међусобно зависних корака које је потребно извршити одговарајућим редоследом. Управљање њиховим покретањем, зависностима и извршавањем назива се **оркестрација токова обраде**. Зависности између корака обично се представљају усмереним ацикличним графом (енгл. *directed acyclic graph,* `DAG`), у коме чворови представљају појединачне задатке или податке, док гране описују зависности између њих. Оркестратор на основу графа одређује које кораке може покренути и којим редоследом. Такође, прати њихов статус, омогућава њихово поновно извршавање и олакшава опоравак након грешке [1].

Поред редоследа корака, потребно је дефинисати и када и под којим условима ће се ток покренути. Покретање може бити временски засновано, када се обрада извршава према унапред одређеном распореду, или засновано на догађају, на пример након пристизања нове датотеке, поруке или другог сигнала. Ова два начина могу се комбиновати.

Најчешће није потребно обрађивати целокупну историју података при сваком извршавању. Скуп података се често дели на партиције (енгл. *partitions*) (према датуму, другом временском интервалу или другом својству). Појединачна извршавања тада се могу ограничити на једну или више партиција. На овај начин смањује се обим појединачног извршавања и поједностављује се проналазак грешака. Оваква подела омогућава и накнадну обраду историјских или пропуштених периода (енгл. *backfill*). Он се користи приликом почетног учитавања историјских података, учитавања након измене трансформационе логике или када је потребно обрадити партиције које раније нису успешно обрађене. Накнадна обрада обично обухвата намерно покретање над изабраним скупом историјских партиција.

Инкрементална обрада (енгл. *incremental processing*). Нови записи могу бити додати постојећој табели, док измењени записи могу бити ажурирани или замењени. У неким случајевима промена изворних података или трансформационе логике утиче на већ израчунате резултате. У тим ситуацијама, постојећи резултат треба потпуно одбацити и поново изградити модел над целокупним релевантним улазним скупом података. Овакав приступ назива се **потпуна обрада** (енгл. *full refresh*). Таква обрада је једноставнија са становишта исправности података, али је временски и ресурсно захтевнија од инкременталне [3].

Складиштење и организација података

Избор система за складиштење зависи од начина на који се подаци користе. Трансакциона обрада (енгл. *Online Transaction Processing*, `OLTP`), карактеристична за оперативне базе, обухвата велики број кратких операција над појединачним записима (на пример унос, измену или читање мањег броја редова). Аналитички системи имају другачију врсту оптерећења. Њихови упити обухватају велики број редова, повезују податке из више табела и израчунавају агрегације над изабраним колонама. Таква обрада означава се као `OLAP` (енгл. *Online Analytical Processing*).

Ова разлика утиче и на физички запис података. Редно оријентисане базе (енгл. *row-oriented*) чувају вредности које припадају једном реду заједно, због чега ефикасно подржавају читање, додавање и измену појединачних редова, што одговара потребама `OLTP` система. За аналитичке упите овакав распоред је неповољан, јер упит који агрегира две или три колоне мора прочитати и све остале. Колонски оријентисане базе (енгл. *column-oriented*) вредности исте колоне чувају заједно, па систем може прочитати само оне колоне које упит користи. Овакав распоред је добар за аналитичке упите и омогућава ефикаснију компресију, јер су вредности унутар колоне истог типа и често имају сличне или поновљене вредности [6].

Редна и колонска организација описују начин на који систем физички чува и приступа подацима. Аналитичке платформе за имплементацију различитих корака у току података користе различите типове складишта. Савремене платформе често комбинују аналитичко и објектно складиште.

Аналитичко складиште

Складиште података (енгл. *data warehouse*) обједињује податке из различитих извора на једном месту и организује их према структури погодној за анализу. Над њим се извршавају аналитички упити, по чему се разликује од оперативних база. Пошто његов профил употребе одговара `OLAP`–у, савремена аналитичка складишта често користе колонску организацију података.

Објектно складиште

Пре складиштења података у организованом облику, потребно је сачувати скупове података у њиховом изворном формату. Једно од уобичајених решења за ово складиште је **објектно складиште**. Објектно складиште представља модел у коме се подаци чувају као независни објекти, заједно са идентификатором и припадајућим метаподацима. За разлику од класичних база података, оно није намењено извршавању сложених упита над структурираним подацима, већ поузданом и скалабилном чувању великих количина података у различитим форматима, као што су `CSV`, `JSON`, `Parquet` и слично. Објектно складиште допуњује аналитичко складиште, али га не замењује [1, 8].

Слојевита архитектура обраде података

Директно претварање сирових података у финалне аналитичке табеле може довести до трансформација које је тешко разумети, тестирати и мењати. Појединачни сложени упит у том случају истовремено врши чишћење, повезивање извора и примену пословних правила. *Слојевита архитектура* раздваја ове одговорности и организује обраду података у неколико јасно дефинисаних целина.

Први ниво обично је **сирови слој** (енгл. *raw, src layer*). Он садржи податке у облику у којем су преузети из извора, без значајних измена. Сирови подаци могу се чувати у објектном складишту, посебној зони аналитичког складишта или у оба система. Његова улога је да сачува веродостојну копију изворних података и да буде полазна тачка за све даље трансформације. Ако се у каснијим слојевима открије грешка или промени трансформациона логика, обрада се може поновити без новог преузимања података из изворног система.

Након сировог слоја често следи **припремни слој** (енгл. *staging, stg layer*). У њему се подаци доводе у облик погодан за даљу обраду. Додељују им се одговарајући типови, стандардизују се називи и формати, обрађују недостајуће вредности и по потреби уклањају дупликати. Пословну логику у овом слоју треба свести на минимум, како би модели остали блиски изворним. Због тога

овај слој обично приказује само тренутно стање сваког податка без дупликата, док се његове историјске верзије, ако су потребне за касније моделовање чувају у ранијем, сировом слоју.

Између припремног и финалног аналитичког слоја може се увести **међуслој** (енгл. *intermediate, int layer*). У њему се обично налазе трансформације које повезују више извора или примењују пословна правила која се користе у више финалних табела. Увођење овог слоја није обавезно, али је корисно када спречава понављање исте логике и поједностављује финалне моделе.

Финални (енгл. *marts*) слој садржи табеле прилагођене аналитичким потребама. У њему се подаци организују према потребама крајњих корисника, и ради формирања аналитичких упита и извештаја, најчешће кроз табеле чињеница, димензионе табеле и изведене агрегатне табеле. Табеле у овом слоју представљају пословно смислен поглед на податке.

Слична подела позната је као **медаљон архитектура** (енгл. *medallion architecture*), у којој се користе називи *бронзани*, *сребрни* и *златни* слој. Називи могу да се разликују, али је основни принцип исти: свака наредна фаза повећава квалитет, структуру и пословну вредност података [4].

Слика 1.1: Слојеви обраде података и називи у медаљон архитектури

Димензионо моделовање

Модели оперативних база првенствено су прилагођени уносу и измени појединачних записа, због чега често садрже велики број нормализованих табела и међусобних веза. Таква структура није увек погодна за аналитичке упите којима су потребни подаци организовани око пословних појмова, за једноставно филтрирање и груписање.

Стандардан одговор на овај захтев је **димензионо моделовање** [5], чија је основна идеја подела података на чињенице и димензије. Овим моделовањем најчешће се обликује структура финалног слоја, јер се управо над њим аналитички упити и извршавају.

Табела чињеница (енгл. *fact table*) бележи догађаје или стање одређеног пословног процеса. Пре изградње табеле чињеница потребно је одредити њену **грануларност**, односно ниво детаља на ком ће подаци бити забележени. Грануларност прецизно дефинише шта представља један ред табеле чињеница. На пример, један ред може представљати појединачни покушај решавања квиза, дневну активност једног студента на курсу или укупан број активности на курсу током једног месеца. Избор грануларности одређује степен детаљности података у табели, као и врсту анализе коју је могуће спровести.

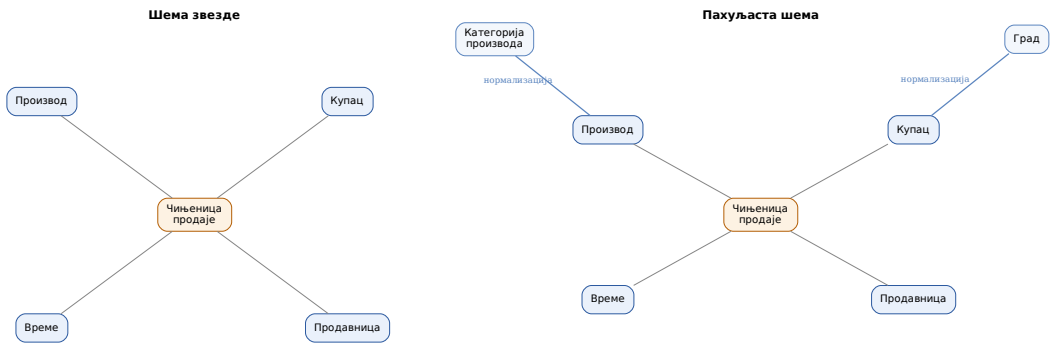
Мере (енгл. *measures*) су квантитативне вредности које табела чињеница бележи за сваки ред, на пример: износ, трајање или неки резултат. Мере и везе са димензијама унутар једне табеле чињеница морају бити усклађене са њеном грануларношћу. На пример, уколико један ред представља појединачни покушај решавања квиза, све вредности у том реду морају се односити искључиво на тај конкретан покушај. Ако се у једној табели помешају подаци различите грануларности, повезивање и агрегација могу произвести дупликате или нетачне резултате.

Табеле чињеница могу се разликовати према начину на који представљају догађаје и стања која се анализирају [5].

- **Трансакциона табела чињеница** (енгл. *transaction fact table*) – садржи један ред за сваки појединачни догађај, нови догађаји додају нове редове, док се постојећи ред мења само када се исправља раније забележени догађај.
- **Периодични пресек** (енгл. *periodic snapshot*) – бележи стање процеса у правилним временским интервалима, тако да свака нова временска тачка производи нов ред. Један ред притом не мора да представља стање тачно једног ентитета: може представљати стање једног ентитета у том тренутку, па за исти ентитет кроз време постоји више редова, или може обједињавати (агрегирати) стање више ентитета у један заједнички ред за посматрани период.
- **Акумулирајући пресек** (енгл. *accumulating snapshot*) – један ред представља целокупан ток једног процеса и ажурира се како процес напредује, нови корак процеса не мора произвести нови ред, већ може променити стање постојећег.

Димензионе табеле (енгл. *dimension tables*) обезбеђују контекст у коме се догађаји посматрају и описују пословне ентитете. Њихови атрибути омогућавају филтрирање, груписање и поређење резултата из различитих перспектива. Посебну врсту димензије представља временска димензија. Она за сваки датум садржи календарске атрибуте, попут године, квартала, месеца, дана у недељи и ознаке викенда.

Најчешћа организација димензионог модела је **шема звезде** (енгл. *star schema*), у којој је табела чињеница у центру, а димензије су повезане директно са њом. Структура је једноставна за разумевање и погодна за упите. Димензије се могу и додатно нормализовати, чиме настаје **пахуљаста шема** (енгл. *snowflake schema*). Тиме се смањује дуплирање података, али модел постаје сложенији за употребу, па се у аналитичким системима предност најчешће даје шеми звезде.



Слика 1.2: Шема звезде и пахуљаста шема

Аналитичка платформа често садржи више табела чињеница које деле исте димензије. Такве димензије називају се заједничким или усклађеним димензијама (енгл. *conformed dimensions*). Када више табела чињеница дели димензије, таква организација димензионог модела може се посматрати као **констелација чињеница** (енгл. *fact constellation*), при чему свака појединачна табела чињеница са својим димензијама и даље формира одговарајућу звездасту структуру.

Поред табела чињеница, могу се формирати и **агрегатне табеле**. Њихова сврха је да поједноставе често коришћење анализе и смање количину података коју је потребно обрадити приликом извршавања аналитичких упита. Агрегатне структуре не замењују детаљне табеле чињеница, већ представљају њихове изведене аналитичке приказе.

Споро променљиве димензије

Атрибути димензија могу се мењати током времена. Када се у димензији чува само последња вредност, историја се губи и прошли догађаји се погрешно тумаче кроз тренутно стање.

Овај проблем решава **концепт споро променљивих димензија** (енгл. *slowly changing dimensions*, `SCD`), који описује начине евидентирања промена у димензионим табелама.

Постоје два основна типа евидентирања промена:

- `SCD` типа 1 где нова вредност преписује постојећу. Приступ је погодан када претходно стање нема аналитичку вредност или када се исправља грешка у податку. Предност је једноставност, али трајно губи претходно стање.
- `SCD` типа 2 где се историја промена чува додавањем новог реда за сваку значајну измену. Претходна верзија остаје сачувана и означава се као неактуелна, док нови ред представља стање које важи од тренутне промене. Димензија зато садржи период важења, индикатор актуелне верзије и сурогат кључ који разликује историјске верзије истог пословног ентитета. Природни кључ остаје непромењен јер идентификује посматрани пословни ентитет, док сурогат кључ раздваја његове историјске верзије.

Табела чињеница повезује се са верзијом димензије која је била важећа у тренутку настанка догађаја. Овакво повезивање назива се темпорално повезивање (енгл. *temporal join*). На овај начин историјски извештаји се не мењају накнадно услед каснијих промена у димензији. Тип 2 се не примењује на све атрибуте, већ само на оне чије је праћење историје од аналитичког значаја [5].

Квалитет и поузданост обраде

Успешно извршен ток података не гарантује да су његови резултати исправни. Подаци могу бити учитани без техничке грешке, а да ипак садрже недостајуће вредности, дупликате, нарушене везе или нелогичне резултате те је обезбеђивање квалитета података кључни сегмент пројектовања и имплементације поуздане аналитичке платформе. Квалитет података најчешће се посматра кроз неколико својстава [1]:

- Тачност: вредности одговарају поузданом извору или стварном стању;
- Потпуност: присуство свих очекиваних података;
- Конзистентност: исти податак нема различита значења у различитим деловима система;
- Јединственост: одсуство дупликата;
- Валидност: подаци одговарају дефинисаним типовима, форматима, дозвољеним вредностима и опсезима;
- Ажурност: брзина којом подаци постају доступни за анализу након настанка.

Део ових захтева зависи и од односа према изворном систему, не само од саме обраде. Трансформациона логика се у великој мери ослања на претпоставке о извору, које колоне постоје, ког су типа и која правила важе. Тај скуп претпоставки назива се уговор о подацима (енгл. *data contract*). За разлику од валидности, која се тиче појединачне вредности, кршење уговора значи промену саме шеме или правила по коме је ток обраде пројектован. У таквим околностима ток обраде треба обуставити све док се узрок промене не утврди и трансформациона логика одговарајуће не прилагоди.

Провере квалитета уводе се у различитим фазама обраде. У ранијим слојевима испитују се типови података, формати, обавезна поља и дупликати. При прављењу каснијих модела захтевају се провере пословних правила, прихватљивих опсега вредности и референцијалних односа између табела. Њихова улога је да проверавају стање скупа који се стално мења, па тестови морају чинити интегрални део редовног процеса обраде, а не представљати једнократну активност. У зависности од значаја правила, неуспешна провера може само евидентирати упозорење или зауставити објављивање непоузданих података.

Поузданост резултата повезана је и са идемпотентношћу обраде. То значи да вишеструко извршавање истог корака над истим улазним подацима не сме произвести дупликате нити нарушити конзистентност резултата, без обзира на то да ли се покретање понавља услед грешке или поновне изградње модела [2].

Са квалитетом је повезан и концепт **порекла података** (енгл. *data lineage*). Он омогућава праћење пута од извора, преко трансформација, до финалних табела. Када се у аналитичком резултату појави неочекивана вредност, информације о пореклу олакшавају проналажење трансформације или улазног скупа у коме је проблем настао.

Провере квалитета одговарају на питање да ли су подаци исправни, али не и на питање да ли је обрада уопште могућа. Трансформације могу бити логички тачне, а да се не изврше зато што неки од сервиса није доступан или зато што заказано покретање није извршено. Поред праћења исправности података, неопходно је имплементирати и механизме за правовремену идентификацију инфраструктурних проблема.

Рачунарство у облаку

Како савремене аналитичке платформе често захтевају складиштење великих количина података, изградња и одржавање такве инфраструктуре у сопственом окружењу може бити сложена и скупа, нарочито када се оптерећење мења током времена.

Рачунарство у облаку (енгл. *cloud computing*) омогућава коришћење рачунарских, мрежних и складишних ресурса без непосредног управљања физичком инфраструктуром. Ресурси се обезбеђују на захтев и могу се прилагођавати тренутном оптерећењу (скалирати). То је посебно важно код система за обраду података, јер се потреба за ресурсима често мења. У једном тренутку систем може само чувати податке, док у другом извршава захтевне трансформације или аналитичке упите [9].

Важна карактеристика облака је и доступност управљаних услуга (енгл. *managed services*). Уместо самосталног инсталирања, подешавања и одржавања сваке компоненте система, део тих одговорности може преузети пружалац услуга у облаку и тиме смањити време потребно за одржавање основне инфраструктуре. Корисник и даље остаје одговоран за начин коришћења услуге, конфигурацију приступа и заштиту својих података.

Посебан облик управљаних услуга представља безсерверно рачунарство (енгл. *serverless computing*): пружалац услуге у потпуности преузима управљање инфраструктуром и њено скалирање, укључујући и то да ли је и када сервер уопште активан. Најпознатији облик овог приступа је извршавање апликационе логике као краткотрајне функције (енгл. *function-as-a-service*), која се покреће одређеним догађајем или распоредом, без сталног сервера који чека на захтев. Пружалац услуге додељује ресурсе само за време извршавања функције, а наплата прати стварно потрошено време рада, а не резервисан капацитет [8]. Безсерверни модел обухвата и друге врсте услуга, попут управљаних база или складишта података, код којих се ресурси такође додељују и наплаћују према стварној употреби.

Поред техничких предности, рачунарство у облаку мења и начин планирања трошкова инфраструктуре. Уместо набавке инфраструктуре за максимално очекивано оптерећење, ресурси се најчешће плаћају према употреби. Трошкови се тада посматрају као оперативни трошкови (енгл. *Operational Expenditure*, `OPEX`). То не значи да су трошкови аутоматски нижи. Код платформи за обраду података потребно је пратити потрошњу складишта, процесорско време и пренос података, јер свака од тих категорија може постати значајан део укупне цене система.

Контејнеризација

Апликације за свој рад зависе од одређених верзија програмског језика, библиотека и системских пакета. Уколико се ове зависности разликују између развојног и продукционог окружења, иста апликација може радити исправно на једној машини, док на другој може доћи до неочекиваних грешака.

Контејнеризација (енгл. *containerization*) омогућава паковање апликације, њених зависности и пратеће конфигурације у јединствену целину. На тај начин апликација се може покретати у уједначеном окружењу на различитим машинама. Основна јединица контејнеризације је слика (енгл. *image*), која садржи апликациони код и све што је потребно за његово извршавање. Покретањем слике настаје контејнер, односно активна инстанца апликације. За разлику од виртуелних машина, контејнери не садрже засебан оперативни систем, већ користе језгро оперативног система на ком се покрећу. Када се систем састоји од више контејнеризованих сервиса, потребно је управљати њиховим покретањем и међусобним повезивањем [7]. Управљање овим задацима назива се оркестрација контејнера.

1.5 Структура рада

Друго поглавље приказује улоге алата у току обраде података и оправдава изабране технологије. Треће поглавље приказује пројекат платформе и образлаже главне одлуке у имплементацији. Четврто поглавље прати њихову имплементацију, од формирања извора до изградње аналитичких модела и имплементације провера квалитета. Пето поглавље садржи евалуацију решења: оцењују се исправност, поузданост, брзина извршавања упита и понашање система под повећаним оптерећењем, уз анализу трошкова одржавања и уочених ограничења.

Глава 2

Преглед коришћених алата

2.1 Улоге алата у току обраде података

Платформа развијена у овом раду прати општи ток обраде података описан у Одељку 1.4.

Генерисање изворних података је такође део имплементације и за њихово складиштење изабрано је објектно складиште у облаку, оркестрацију преузима `Dagster`, док су трансформације и провере квалитета података реализоване кроз `dbt` моделе, а припремљени подаци чувају се у аналитичком складишту. Компоненте се извршавају у контејнерима на виртуелној машини у облаку.

2.2 `Dagster` као алат за оркестрацију

`Dagster` има улогу централног оркестратора токова обраде података. Он одређује редослед извршавања корака обраде, прати њихов статус и повезује учитавање и трансформацију података у јединствен процес. Такође може пратити и доступност спољних сервиса од којих обрада података зависи. Основни концепт у `Dagster`–у назива се поседован податак (енгл. *software-defined asset*, `asset`). Он представља податак или скуп података који настаје извршавањем одређене логике, на пример, табелу учитану из изворног система или аналитички модел добијен трансформацијом других табела. Ако један такав податак зависи од другог, та веза се експлицитно дефинише у коду. На основу тих зависности `Dagster` формира граф који приказује како подаци настају и којим редоследом поједини кораци треба да се изврше [10].

Овакав приступ природно се повезује са dbt–ом. Сваки dbt модел представља табелу или поглед који настаје из других модела или изворних табела, док су зависности између њих већ дефинисане у самом dbt пројекту. Dagster може преузети те информације и сваки dbt модел приказати као засебан asset. Због тога није потребно ручно дефинисати посебан задатак за извршавање сваког модела.

У делу платформе која се бави уносом изворних података, asset-и одговарају табелама које се пуне сировим подацима из објектног складишта. На њих се затим надовезују dbt модели који такође представљају Dagster asset-е.

Важан део Dagster–а је праћење извршавања кроз кориснички интерфејс. Кориснички интерфејс омогућава увид у статус покренутих процеса, трајање појединачних корака, евентуалне грешке и међусобне зависности asset-а, што значајно олакшава развој и тестирање платформе.

Dagster за покретање и организацију обраде користи три концепта [10]:

- распореди (енгл. *schedules*) покрећу токове обраде у унапред дефинисаним интервалима;
- сензори (енгл. *sensors*) покрећу обраду на основу догађаја, на пример доласка нових података или испуњења одређеног услова;
- партиције омогућавају да се исти процес извршава над одређеним временским опсегом, на пример за један дан, што олакшава поновну обраду само дела података уколико је то потребно.

Као алтернатива разматран је Apache Airflow, у коме се ток рада описује као DAG састављен од задатака и њихових зависности [11]. Чворови таквог графа представљају кораке извршавања: систем зна да је одређени задатак успешно завршен, али сам по себи не чува податак о томе који конкретан скуп података тај задатак производи, нити прати његово стање – да ли уопште постоји, за коју партицију и када је последњи пут ажуриран. Зависности између задатака при том се дефинишу ручно, независно од тога да ли ти задаци заиста деле исте податке.

За ову платформу изабран је Dagster, код кога чворове графа не представљају кораци извршавања, већ сами подаци које ток производи – asset-и, сваки са сопственом историјом материјализације. Ова разлика посебно долази до изражаја у комбинацији са dbt–ом, чији модели већ по себи представљају конкретне податке, а не кораке обраде: Dagster зависности између њих може

директно преузети из `dbt` пројекта, уместо да их поново дефинише на нивоу оркестратора (детаљи ове интеграције дати су у Одељку 4.5). Иста логика примењена је и на унос изворних података, тако да читав ток, од учитавања сирових података до финалних модела, чини један непрекинут граф зависности, а не два одвојена система задатака које би тек требало ручно повезати.

2.3 `dbt` као алат за трансформацију и моделовање података

`dbt` је алат намењен трансформацији и моделовању података у аналитичком складишту. У `ELT` приступу (описаном у Одељку 1.4), `dbt` покрива фазу трансформације. Подаци се најпре учитавају у изворне табеле у `ClickHouse`–у, а затим се над њима извршавају `SQL` упити који их чисте, повезују и организују за аналитичку употребу.

Основни елемент рада у `dbt`–у је **модел**. Модел најчешће представља `SQL` упит који дефинише како се једна табела или поглед формира на основу постојећих података [3].

Сваки модел има дефинисану **материјализацију**, која одређује како се његов резултат чува у бази. Три најчешће коришћена типа материјализација су:

- поглед (енгл. *view*) не заузима додатни простор, а резултат се изнова израчунава при сваком упиту;
- табела (енгл. *table*) чува резултат физички и при сваком покретању се гради изнова;
- инкрементална материјализација (енгл. *incremental*) обрађује само нове или измењене податке, уместо поновне изградње читавог скупа.

Избор материјализације зависи од величине података, учесталости измена и тога да ли се модел користи у контекстима који захтевају поновно израчунавање при сваком упиту [3].

Зависности између модела не дефинишу се ручно. Користе се функције `ref()` и `source()` у зависности од порекла података над којима се упити модела извршавају. Прва упућује на неки `dbt` модел, а друга на изворну табелу. На основу тих позива `dbt` сам гради усмерен ациклични граф зависности (видети Одељак 1.4) и из њега одређује редослед извршавања [3]. `dbt` такође може генерисати документацију пројекта, укључујући и визуелни приказ графа зависности између модела.

Правила која се понављају у више модела не морају се дуплирати. `dbt` обрађује `SQL` упите кроз систем шаблона, што омогућава издвајање заједничких израза у макрое и њихово поновно коришћење у различитим моделима. Статички референтни подаци, који не долазе из изворног система, уносе се као *seed* датотеке – `CSV` датотеке из репозиторијума које `dbt` учитава као обичне табеле.

Поред трансформација, `dbt` подржава и **тестове** над подацима. Уобичајене провере, као на пример присуство и јединственост вредности или дозвољени скуп вредности задају се декларативно, у конфигурационој датотеци уз модел. Сложенија правила, која обухватају више колона или табела, пишу се као засебни `SQL` упити који не смеју вратити ниједан ред да би тест био успешан. Сваком тесту додељује се ниво озбиљности, чиме се разликује одступање које треба само евидентирати од грешке која зауставља обраду [3].

Разматране су и алтернативе, попут писања `SQL` скрипти које оркестратор директно покреће. Такав приступ не нуди уграђено тестирање, креирање документације нити аутоматско извођење зависности између модела, па би те механизме требало посебно правити. `dbt` је изабран јер је намењен управо трансформацији података унутар постојећег складишта, где данас нема правог конкурента за ову врсту трансформације у `SQL` окружењу. Такође поседује и готов адаптер за рад са `ClickHouse` базом (`dbt–clickhouse`) [12].

2.4 Окружење у облаку

Компоненте платформе немају исти образац рада. Оркестрација и трансформације захтевају стално активно окружење, док се генерисање изворних података покреће једном дневно и траје кратко. Истовремено, количина сачуваних података непрекидно расте. У окружењу у облаку ови захтеви могу се раздвојити, чиме се за сваки од њих примењује одговарајућа услуга. На тај начин рачунарски ресурси не морају бити стално резервисани када нису у употреби, док капацитет складишта може да расте паралелно са количином података.

Коришћене су управљане услуге, код којих пружалац преузима део послова одржавања, обезбеђивања доступности и надоградње инфраструктуре. При избору пружаоца разматрани су `Amazon Web Services`, `Google Cloud Platform` и `Microsoft Azure`. Сва три нуде услуге потребне за реализацију платформе, као што су објектно складиштење, виртуелне инстанце, покретање функција

према распореду и управљање приступним овлашћењима. Изабран је AWS због претходног искуства аутора рада стеченог у пословном окружењу. Познавање његових услуга и IAM (*Identity and Access Management*) система смањило је ризик од грешака при подешавању инфраструктуре и права приступа [8].

У наставку су описане услуге употребљене у платформи: Amazon S3 за објектно складиштење, Amazon EC2 за покретање стално активних компоненти и AWS Lambda, заједно са EventBridge Scheduler-ом, за потребе дневног генерисања изворних података.

2.5 Складиштење података: ClickHouse Cloud и Amazon S3

Платформа користи два система за складиштење података, у складу са поделом описаном у Одељку 1.4.

ClickHouse као аналитичко складиште. ClickHouse је колонски оријентисана база података намењена аналитичкој обради великих количина података [13]. У њој се чувају табеле са изворним подацима, као и припремни, међуслојни и финални модели које dbt формира. У раду је коришћен ClickHouse Cloud, управљана варијанта овог система. Њеним избором послови као што су одржавање инфраструктуре, обезбеђивање доступности, креирање система за надгледање и скалирање базе препуштени су пружаоцу услуге.

Као друга могућност разматран је Snowflake, такође управљано аналитичко складиште погодно за исту врсту оптерећења. ClickHouse је изабран јер је његова основна верзија отвореног кода и може се покренути и на сопственој инфраструктури [13]. То оставља могућност за касније премештање базе из управљаног окружења, без измене самог система за складиштење. Оба решења имају одговарајућу подршку у dbt-у и могу се уклопити у исти приступ трансформацији података.

Amazon S3 као објектно складиште. Amazon S3 је услуга за објектно складиштење података у облаку [8]. За разлику од ClickHouse-а, не користи се за извршавање аналитичких упита. У њему се улазни скупови чувају пре учитавања у аналитичку базу, у облику блиском њиховом изворном формату. S3 је изабран зато што не захтева одржавање засебног сервера или диска, а расположиви капацитет се прилагођава количини сачуваних података.

2.6 Извршно окружење: контејнери и функције без сервера

Стално активне компоненте платформе покрећу се у `Docker` контејнерима. Слика контејнера садржи апликациони код и тачно одређене верзије потребних алата и зависности, чиме се обезбеђује исто окружење током локалног развоја и извршавања у облаку. Сервиси платформе описани су у једној `Docker Compose` датотеци. Она садржи њихове променљиве окружења, дељене дискове и мрежна подешавања, па се сви сервиси могу покренути и зауставити као целина [7].

Контејнери се извршавају на `Amazon EC2` виртуелној инстанци. Ова услуга омогућава избор инстанци са различитим процесорским, меморијским и мрежним капацитетима [8]. Разматране су и управљане услуге које самостално распоређују контејнере на доступне рачунарске ресурсе. Њихове предности долазе до изражаја када број покренутих примерака треба мењати у складу са оптерећењем. У овој платформи покреће се мали и унапред познат скуп сталних процеса, па једна виртуелна инстанца испуњава ове захтеве.

Генерисање симулираних изворних података обавља функција покренута помоћу услуге `AWS Lambda`, која омогућава краткотрајно извршавање кода без потребе за одржавањем засебног сервера. Функцију једном дневно позива `EventBridge Scheduler`. У овој услузи дефинишу се време извршавања и сервис који треба позвати, уз могућност избора временске зоне и подешавања поновног покушаја у случају неуспелог позива [8].

Генератор се могао покретати и на постојећој `EC2` инстанци. Међутим, он у овом раду представља замену за продукциони изворни систем, па је издвојен из окружења у којем се извршава остатак платформе.

2.7 Помоћни алати и конфигурација

`Python` је програмски језик који се широко користи у инжењерству података и аутоматизацији. Поседује велики број библиотека за рад са базама података, датотекама, мрежним сервисима и *cloud* ресурсима, што га чини погодним за развој различитих делова аналитичке платформе.

Током развоја често се мењају апликациони код, `SQL` модели и конфигурационе датотеке. `Git` служи за праћење тих измена и чување историје развоја. Он омогућава поређење различитих верзија, паралелан рад у одвојеним гранама као и враћање на раније верзије кода.

Глава 3

Пројектовање платформе

3.1 Домен и захтеви система

Платформа развијена у раду моделује податке система за учење на даљину чија је конфигурација инспирисана неким јавно доступним популарним платформама (нпр. `Udemy` или `Coursera`).

Изворни подаци и њихове везе

Основу домена чине две групе података (на Слици 3.1 налази се детаљнији приказ њихових главних атрибута и међусобних веза):

- пословни ентитети: студенти, курсеви, инструктори и садржаји курсева, тј. појединачне наставне јединице (квиз, видео, задатак);
- догађаји који настају њиховом интеракцијом: уписи на курсеве, приступање садржају, покушаји решавања квизова и остављање рецензија.

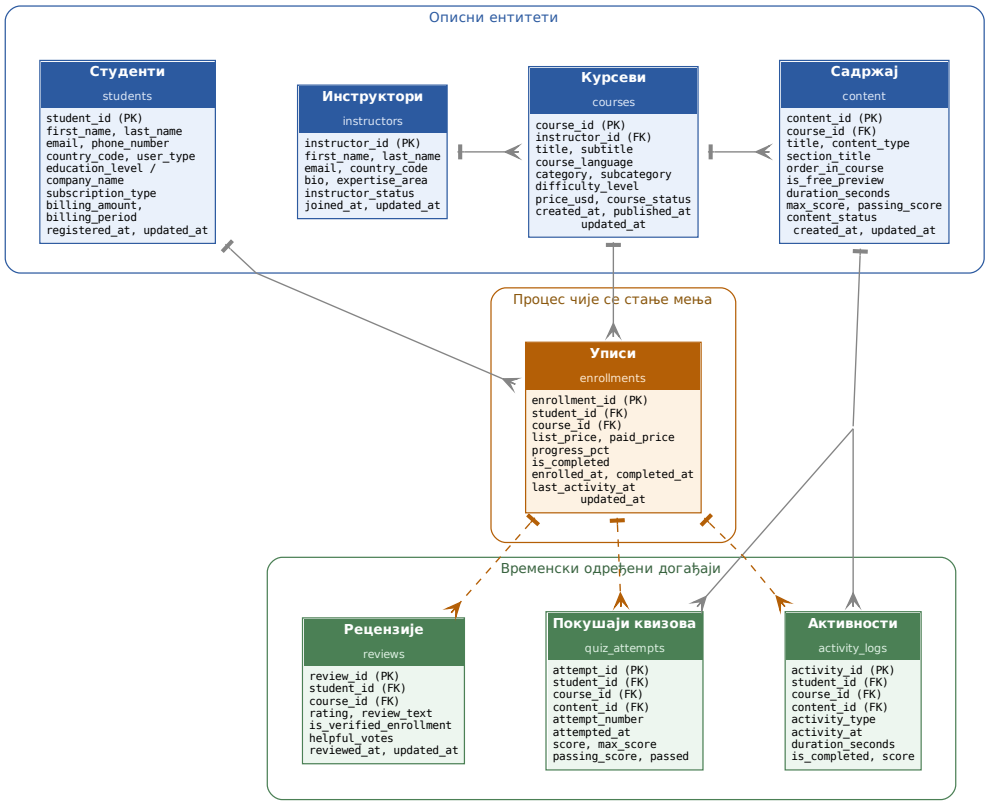
Подаци имају три различите природе. Студенти, инструктори, курсеви и садржај описују ентитете; упис је процес чије се стање мења; активности, покушаји и рецензије представљају временски одређене догађаје.

Студент припада једној од две категорије корисника: самосталном полазнику (енгл. *student*) или запосленом кориснику који похађа курсеве преко одређене компаније (енгл. *employee*). Ове категорије се међусобно искључују, па профил студента, поред основних података, садржи тачно једно од два додатна поља: ниво образовања за самосталне полазнике, односно назив компаније за запослене кориснике. Сваки студент може изабрати тип претплате приликом регистрације. Она може бити бесплатна (енгл. *free*), месечна (енгл. *monthly*), годишња (енгл. *annual*) или корпоративна (енгл. *enterprise*).

Бесплатна претплата не доноси попуст, док остале доносе попуст од 20%, 30% и 40% при упису. Тип корисника и претплата могу се мењати током коришћења платформе.

Један инструктор може бити аутор више курсева, док сваки курс припада једном инструктору. Сваки курс припада одређеној области и има додељен ниво тежине, намењен почетницима, средње напредним или напредним полазницима. Курс се састоји од уређеног скупа садржаја три типа: видео-лекција, квиз и задатак. За видео-лекције значајно је трајање, док квиз и задатак имају максималан број поена и праг пролазности потребан за пролаз. Цена и статус курса такође се могу променити, па тренутно стање курса не мора одговарати условима под којима су настали ранији уписи на тај курс.

Везе на Слици 3.1 приказане су ER нотацијом облика *crow's foot*: попречна црта уз ентитет означава страну „један", а рашчлањени крак уз ентитет страну „много" у датој вези.



Слика 3.1: Изворни скупови података, њихови главни атрибути и везе

Упис повезује једног студента и један курс и представља **централни процес домена**. У тренутку уписа бележе се редовна цена курса и износ стварно плаћен након попуста ако је студент на неком плану претплате. Током похађања исти упис добија нови проценат напретка, време последње активности и, ако је курс завршен, датум завршетка. Студент у посматраном систему може само једном уписати исти курс, па се сав његов напредак наставља у оквиру постојећег уписа. Ово ограничење одговара понашању стварних платформи попут `Udemy`-ја, где се курс који се већ поседује не може поново купити. Тип претплате само одређује попуст при том једном упису, а не даје приступ који временом истиче и мора да се обнавља, па нема ситуације у којој би исти студент морао поново да упише исти курс. Након уписа настају активности, покушаји решавања квизова и остављање рецензије. Активност и покушај односе се на конкретан садржај, док се рецензија односи на курс у целини и садржи оцену и коментар студента.

Слика 3.1 показује и везе између група података. Пуна линија значи да је веза директно записана у подацима, на пример, курс у себи носи идентификатор инструктора коме припада, а упис идентификаторе студента и курса. Испрекидана линија значи да веза са уписом није директно записана, већ се до ње долази комбиновањем истог студента и истог курса. То важи и за активност, покушај решавања квиза и рецензију: ниједна од те три табеле не садржи идентификатор уписа, већ се одговарајући упис одређује проналажењем уписа истог студента на исти курс на који се дати ред односи. Та веза је једнозначна због наметнутог правила да поновни упис није дозвољен. Ова претпоставка касније утиче и на пројектовање аналитичког модела.

Ради прегледности дијаграма, директне везе табела `activity_logs`, `quiz_attempts` и `reviews` ка студенту и курсу нису исцртане, иако те табеле заиста садрже колоне `student_id` и `course_id` (видети њихове атрибуте на самој слици) – исти контекст студента и курса до тих табела стиже и посредно, преко садржаја или преко изведене везе са уписом.

Технички захтеви

Описани подаци настају у изворном, оперативном систему и организовани су према његовим потребама. Циљ платформе је да ове податке повеже и припреми за анализу.

Конкретно, платформа треба да омогући:

- праћење ангажованости и напретка студената;
- анализу завршавања курсева и резултата квизова;
- поређење успешности курсева на основу рецензија;
- анализу прихода и коришћења претплатничких модела;
- збирне показатеље по студенту и курсу, укључујући преглед више покушаја истог квиза;
- праћење различитих верзија података кроз време.

Резултати не морају бити доступни одмах по настанку сваког догађаја, па је уместо континуиране (енгл. *streaming*) обраде изабрана дневна пакетна (енгл. *batch*) обрада (видети Одељак 1.4). Континуирана обрада би додатно усложнила извор, инфраструктуру и управљање стањем, без значајне користи за анализе које су предмет рада.

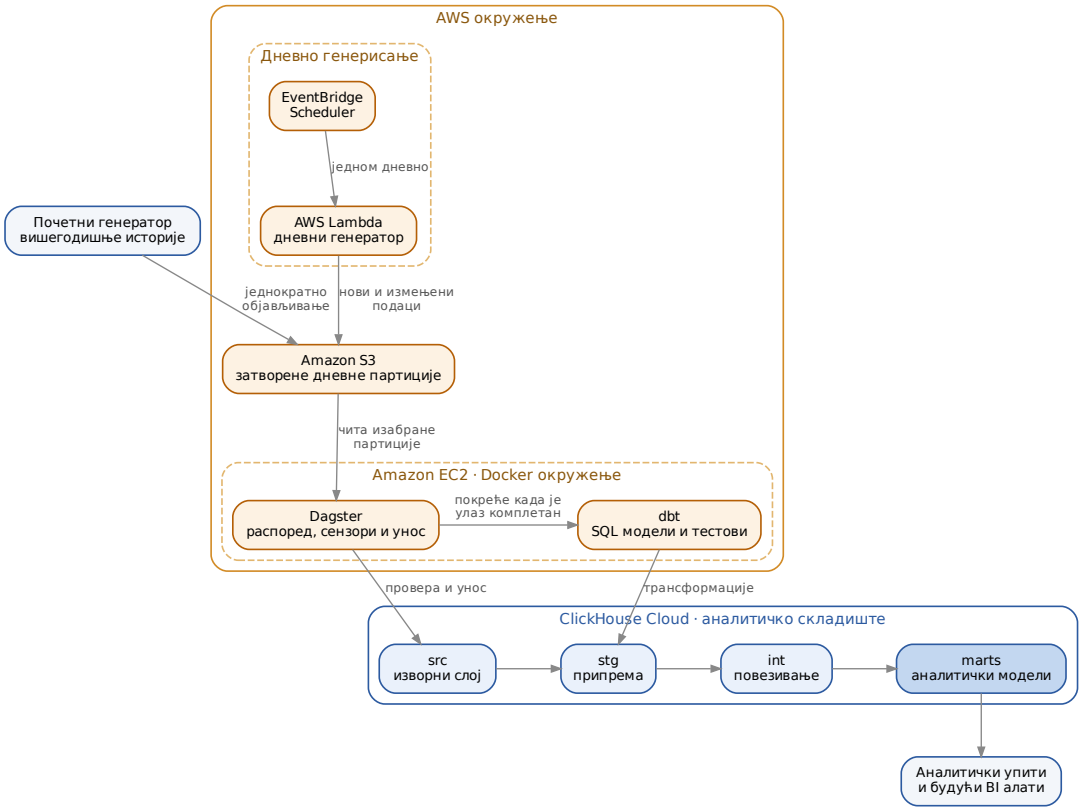
Обрада треба да подржи четири ситуације:

- почетно учитавање вишегодишње историје;
- редовну дневну обраду нових података;
- поновно покретање обраде за дан у којем је настала грешка;
- поновну изградњу модела када се промене правила трансформације или начин чувања историје.

У свакој од њих систем мора да буде идемпотентан, односно поновно извршавање над истим улазним подацима не сме стварати дупликате нити нарушавати постојеће резултате (видети Одељак 1.4). Такође, у сваком тренутку тока обраде мора бити могуће утврђивање тачног извора сваког податка, док се исход извршавања бележи ради накнадне провере (видети Одељак 3.5). Улазни подаци могу садржати недостајуће вредности, дупликате, неважеће идентификаторе или догађаје који нису у складу са повезаним ентитетима, па платформа треба да омогући и њихову проверу.

3.2 Архитектура платформе: ток података и инфраструктура у облаку

Архитектура платформе може се посматрати из два повезана угла: логички поглед прати податке од њиховог настанка до аналитичких модела, а инфраструктурни показује где се компоненте извршавају и за шта су одговорне. Оба погледа обједињена су на Слици 3.2.



Слика 3.2: Ток података и архитектура аналитичке платформе

Ток података од извора до аналитичког слоја

Генератор одговоран за креирање синтетичког скупа података има два режима рада: почетни, који једном формира вишегодишњу историју (до датума покретања генератора), и дневни, који наставља да производи нове, измењене податке за сваки завршен дан. Оба режима објављују исти скуп од осам изворних група података у `Amazon S3`-у, у виду датотека организованих по

датуму. За сваку групу постоји фолдер за сваки календарски дан, а сав садржај тог фолдера чини дневну партицију групе. Ова организација по данима директно прати одлуку о дневној пакетној обради јер сваки `Dagster` *run* касније чита тачно једну овакву партицију. Пошто историјско и дневно учитавање производе податке истог облика и складиште их на исто место, за остатак платформе не постоје два различита извора. Разлика је само у обиму: почетно учитавање обухвата низ покретања преко свих партиција од почетка периода (Одељак 3.4), а дневно само једно покретање, за последњу партицију.

Дневни генератор, за разлику од почетног, не полази сваки пут из празног стања. Како би се нове активности повезале са већ познатим студентима, курсевима и садржајем, и напредак започетих уписа наставио, генератор мора знати шта је раније произведено. Зато се поред дневних партиција у `S3`-у чува и посебан приказ (енгл. *snapshot*) тј. листа постојећих ентитета и отворених уписа која се ажурира при сваком дневном покретању. Овај приказ није део аналитичке платформе већ служи искључиво за генерисање података.

`Dagster` преузима податке из `S3`-а, проверава улаз и учитава га у слој `src` `ClickHouse`-а. Тек када су скупови за посматрани дан комплетни, покрећу се `dbt` трансформације. Подаци се унутар `ClickHouse`-а припремају, међусобно повезују и обликују у финалне моделе. Тиме се завршава пут од изворних датотека до структура над којима се могу извршавати аналитички упити.

Оваква организација прати `ELT` приступ описан у Одељку 1.4: `Dagster` управља редоследом и исходом извршавања, `dbt` дефинише трансформације и њихове зависности, а `ClickHouse Cloud` чува податке и извршава `SQL` упите. Захваљујући подели улога, промена правила моделовања не захтева измену начина на који се датотеке преузимају из `S3`-а.

Организација података у `ClickHouse`-у

Унутар `ClickHouse`-а подаци пролазе кроз слојеве `src`, `stg`, `int` и `marts`. Сваки од њих има различиту улогу, у складу са слојевитом организацијом аналитичких складишта (види Одељак 1.4)

Слој `src` садржи податке у облику блиском изворним датотекама. Иако се исти садржај већ налази у `S3`-у, улоге се разликују. `S3` чува оригиналне датотеке и њихову организацију по партицијама, а `src` табеле омогућавају да подаци буду доступни за `SQL` трансформације. У њему се не примењује аналитичка логика, што омогућава поновну изградњу виших модела из исте полазне тачке.

У слоју `stg` се обављају чишћење и стандардизација: вредности се доводе у јединствен облик, додељују им се одговарајући типови, обрађују празна поља, уклањају поновљени записи и, по потреби, маскирају осетљиви подаци. У овом слоју различити изворни скупови се још не повезују, тај процес се започиње у слоју `int`. У њему се издваја заједничка логика која захтева спајање или агрегацију више скупова, као што је формирање приказа напретка студента на курсу.

Слојеви се разликују и по начину материјализације. `stg` модели су погледи: свака вредност у њима изводи се из појединачног реда слоја `src`, без спајања и агрегација, па би чување у посебним табелама дуплирало податке, а не би донело рачунску уштеду. Поглед уз то гарантује да очишћени приказ увек одговара тренутном садржају слоја `src`. `int` модели су, насупрот томе, физичке табеле јер њихови резултати настају спајањем више скупова, па се израчунавају једном и затим користе у више финалних модела.

Слој `marts` представља финални аналитички слој и границу према корисницима података. У њему су процеси из домена повезани са описним контекстом студената, курсева, садржаја и времена, а ту се налазе и сажети показатељи на нивоу студента, курса или квиза. Сви модели овог слоја материјализовани су као физичке табеле, јер аналитички корисници и `BI` алати приступају искључиво њему, па перформансе и стабилност упита не смеју да зависе од сложености трансформација у нижим слојевима.

Финални модели могли су се формирати директно над `src` табелама, али би се тада иста правила чишћења и припреме података понављала на више места. То би отежало одржавање и могло да доведе до различитог тумачења истих вредности.

Конфигурација окружења у облаку

Компоненте платформе (генератор података, оркестрација и аналитичка база) нису смештене на једном серверу, већ су распоређене на посебне сервисе, у складу са природом посла који обавља. У `AWS` окружењу `EventBridge Scheduler` покреће `Lambda` функцију ради генерисања података за претходни дан и смешта их у `S3` (видети Одељак 2.5).

`Dagster` захтева непрекидно активан процес јер покреће дневни унос података по унапред одређеном распореду, преко сензора проверава да ли су сви подаци за тај дан стигли и тек тада покреће трансформације, чува историју

претходних извршавања и опслужује кориснички интерфејс. Разлог избора EC2 за овакав рад описан је у Одељку 2.6. Компоненте се покрећу у Docker контејнерима, чиме су верзије зависности и начин покретања одвојени од саме виртуелне машине.

Приступ AWS ресурсима заснован је на принципу најмањих привилегија: и Lambda функција и EC2 инстанца добијају само она овлашћења која су им заиста неопходна, додељена преко IAM улога, а не преко трајних приступних кључева сачуваних на локалним машинама.

Аналитичка база издвојена је у ClickHouse Cloud, уместо да дели исту EC2 инстанцу са оркестрацијом. Тако се ресурси за складиштење и извршавање упита могу мењати независно од ресурса које користи Dagster. Управљани начин рада уводи и другачију динамику доступности: ради уштеде, ClickHouse Cloud инстанца се аутоматски успављује после периода неактивности од 15 минута, а буђење траје знатно дуже од стандардног успостављања конекције. Због тога провера доступности овог сервиса захтева прилагођен приступ, објашњен у Одељку 3.5.

3.3 Пројектовање аналитичког модела

Раније представљене особине података одређују структуру финалног marts слоја. Упис, активност, покушај решавања квиза и рецензија издвојени су као четири процеса; сваки добија сопствену fact табелу која бележи шта се догодило и шта се у оквиру тог процеса мери, док јој димензије додају контекст студента, курса, инструктора, садржаја и времена у ком се догађај десио.

Моделовање пословних процеса

Аналитички циљ	Пословни процес	Одговарајући модел
праћење напретка, завршавања и прихода	упис студента на курс	fact_enrollments
анализа ангажованости и коришћења садржаја	активност студента	fact_activity_logs
анализа резултата и поновљених покушаја	покушај квиза или задатка	fact_quiz_attempts
анализа оцена и задовољства курсом	остављање рецензије	fact_reviews

Табела 3.1: Модели према пословним циљевима

Процеси описани у Табели 3.1 нису обједињена у једну табелу чињеница, већ деле заједничке димензије и на тај начин формирају констелацију чињеница. Разлог одвајања је различит број извршавања сваког процеса у односу на упис: један упис може имати велики број активности и више покушаја квизова. Када би се сви подаци спојили у једну широку табелу, информације о упису понављале би се за сваки придружени догађај. Одвојене табеле чињеница задржавају грануларност сваког процеса и омогућавају исправно агрегирање њихових мера.

Грануларност табеле чињеница одређује шта представља један њен ред, а у платформи постоје два начина на која се пословни процеси појављују у подацима, што директно одређује врсту табеле чињеница коју сваки од њих добија (види Одељак 1.4). Активности над садржајем, покушаји квизова и рецензије имају тачно одређено време настанка и накнадно се не мењају. То су појединачни догађаји, па им одговара трансакциона табела чињеница, у којој један ред представља један догађај. Упис студента на курс, с друге стране, представља процес чије се стање мења од пријаве до евентуалног завршетка, па га боље описује акумулирајући пресек, у ком један ред представља једну инстанцу процеса која се ажурира како процес напредује. Табела 3.2 приказује ову поделу за сва четири модела.

Модел	**Значење једног реда**	**Врста `fact` табеле**
`fact_enrollments`	један упис студента на курс	акумулирајући пресек
`fact_activity_logs`	једна активност студента над садржајем	трансакциона `fact` табела
`fact_quiz_attempts`	један покушај решавања квиза или задатка	трансакциона `fact` табела
`fact_reviews`	једна рецензија курса	трансакциона `fact` табела

Табела 3.2: Грануларност табела чињеница

У тренутку уписа на курс студент може имати нулти напредак, без започетог курса. Касније се уз исти упис појављују активности, мења се проценат напретка и, у случају завршетка курса, постаје познат и датум завршетка. Последње познато стање остаје записано у `fact_enrollments`. Пошто се цена, попуст и плаћени износ сусрећу управо у тренутку уписа, приход се рачуна и чува уз сам упис, у истој табели. Проценти попуста при томе чувају се у засебном референтном скупу, уместо уграђивања директно у `SQL` логику.

Код преостале три, трансакционе табеле чињеница, нови покушај истог квиза не мења претходни, већ формира нови ред; исто важи и за нову активност над садржајем. Ако извор накнадно пошаље исправку већ постојећег догађаја, она треба да замени претходно забележену верзију са истим идентификатором, а не да буде тумачена као нови пословни догађај.

Поред табела чињеница, у финалном слоју постоје и агрегатни модели који одговарају на честа питања на нивоу студента, курса или квиза. Један ред агрегатног модела представља тренутно збирно стање једног ентитета. На пример, укупан број уписа и просечан напредак једног студента израчунат над свим повезаним редовима чињеница, за разлику од табела чињеница где један ред представља појединачан пословни догађај. Цео агрегатни модел се при сваком извршавању у потпуности поново израчунава, па увек одражава тренутно стање, без чувања претходних верзија. Табела 3.3 приказује значење једног реда за сва три агрегатна модела.

Модел	**Значење једног реда**
`agg_student_stats`	резултати једног студента кроз све његове уписе
`agg_course_stats`	резултати једног курса кроз уписе и рецензије
`agg_quiz_attempt_stats`	резултати једног студента кроз све покушаје истог квиза или задатка

Табела 3.3: Грануларност агрегатних модела

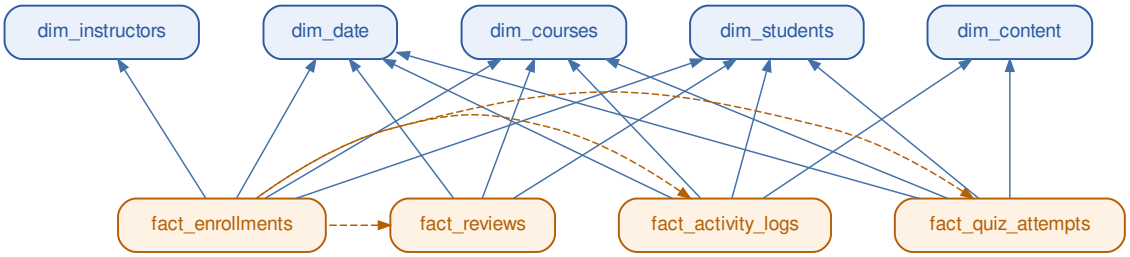
Димензије и везе у моделу

Контекст пословним процесима дају димензије (Табела 3.4).

Димензија	Аналитичка улога	SCD тип
dim_students	опис студента, типа корисника и претплате	SCD2 – тип претплате, тип корисника, припадност компанији
dim_courses	опис курса, категорије, нивоа тежине, цене и статуса	SCD2 – цена, статус
dim_instructors	опис инструктора и области његове стручности	SCD1 – тренутно стање
dim_content	опис видео-лекције, квиза или задатка унутар курса	SCD1 – тренутно стање
dim_date	заједнички календарски контекст догађаја	унапред припремљена

Табела 3.4: Улога и SCD тип димензија у аналитичком моделу

Више табела чињеница дели исте димензије, па се исти студент или курс тумаче на идентичан начин кроз упис, активност, квиз и рецензију. Поједностављене везе између димензија и табела чињеница приказане су на Слици 3.3.



Слика 3.3: Поједностављени дијаграм димензионог модела

dim_students и dim_courses користе све табеле чињеница. dim_content је повезана са активностима и покушајима квизова, јер ти догађаји настају над одређеним делом курса, док се рецензија односи на курс у целини. Инструктор је у контексту уписа заступљен преко уписаног курса, па само fact_enrollments има везу са dim_instructors.

`fact_reviews`, `fact_activity_logs` и `fact_quiz_attempts` повезани су са одговарајућим уписом у `fact_enrollments` преко студента и курса, испрекидане линије на слици.

Димензије се са табелама чињеница повезују сурогат кључевима, а не природним. Разлог су историјске (`SCD2`) димензије: `dim_students` и `dim_courses`, приказане у Табели 3.4. Код њих се, за разлику од осталих, чува више верзија истог ентитета кроз време. На пример, ако студент промени тип претплате, `dim_students` ће садржати два реда са истим `student_id`: један за период пре промене, други за период после. Природни кључ тада више не одређује једнозначно на који од та два реда се мисли, јер га оба деле. Зато се сваком реду додељује сурогат кључ, изведен детерминистичком функцијом из природног кључа и времена важења те верзије. Тако сваки ред добија јединствен кључ чак и када дели природни кључ са другом верзијом истог ентитета, а исти улаз увек даје исти резултат, па поновна изградња модела не мења постојеће везе. Ради доследности, исти образац повезивања користи се и код осталих димензија, иако код њих овај проблем не постоји.

`dim_date` се, за разлику од осталих димензија, не формира из изворних података. Она је унапред припремљена за одређени временски опсег како би све `fact` табеле користиле исте календарске атрибуте, као што су месец, квартал, дан у недељи и ознака викенда. Ако се опсег података прошири, потребно је проширити и ову димензију.

Формирање историјских верзија. У Табели 3.4 приказано је које димензије чувају историју промена. Историја се чува само за промене које имају аналитички значај (механизам познат као `SCD` тип 2, види Одељак 1.4).

Верзије студената и курсева ређају се према пословном времену, односно тренутку када се промена догодила у изворном систему. Време учитавања није погодно као основна временска оса, јер се током `backfill`-а промене настале у различитим периодима могу учитати у кратком року. Оно служи као резервна вредност када пословно време није доступно и у ситуацијама у којима треба изабрати последњу пристиглу исправку исте пословне верзије.

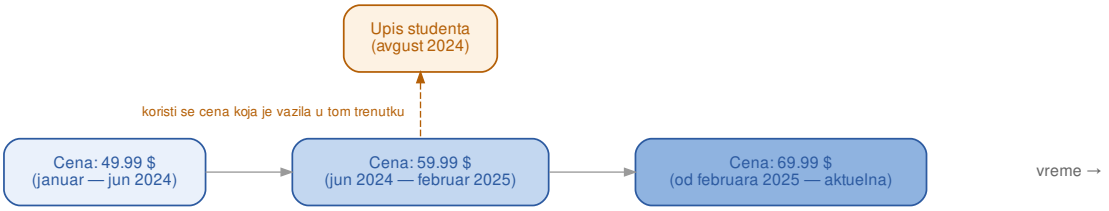
Историјске верзије нису формиране уграђеним `dbt` механизмом за праћење промена (енгл. *snapshot*), него су изведене из свих верзија сачуваних у сировом слоју. Разлог је временска оса: тај механизам верзију бележи према тренутку у коме је обрада покренута, а не према тренутку у коме се промена догодила у изворном систему. При почетном учитавању вишегодишње историје све верзије би тако добиле готово исто време, границе важења пале

би у један дан, а сваки историјски догађај повезао би се са првом верзијом.

Друга последица тиче се поновљивости. Такав механизам гради историју постепено, додајући верзије при сваком покретању, па се она не може реконструисати без непрекинутог низа претходних извршавања. Пошто сирови слој чува све пристигле верзије (видети Одељак 1.4), димензије се уместо тога у целини израчунавају при сваком извршавању. Исти улаз тако увек даје исти резултат, у складу са захтевом за идемпотентношћу (видети Одељак 1.4), а верзија која стигне са закашњењем уклапа се на своје место у историји, уместо да се забележи као најновија.

Свака верзија има сопствени период важења. Приликом формирања `fact` табеле бира се верзија која је била активна у тренутку настанка догађаја и та вредност остаје записана у табели. Слика 3.4 приказује овај принцип на примеру курса. За упис се користи време уписа, за активност време њеног извршавања, за покушај квиза време покушаја, а за рецензију време њеног остављања. На тај начин сваки догађај задржава контекст који је важио када је настао, уместо повезивања са тренутним стањем студента или курса.

Изузетак је приход у `fact_enrollments`. Историјска верзија курса и претплате студента, добијена оваквим повезивањем, ту не служи као извор прихода, већ само као провера: ауторитативна вредност је износ који је изворни систем забележио као стварно плаћен у тренутку уписа, док цена и попуст реконструисани из историјских димензија само потврђују подударност са условима који су тада важили.



Слика 3.4: Повезивање догађаја са историјском верзијом курса

3.4 Пројектовање тока обраде података

Одељак 3.1 навео је четири потребе које ток обраде треба да подржи. Полазна тачка обраде података је дневна партиција у `S3`-у: генератор, описан у Одељку 3.2, сваког дана у њу уписује податке за претходни, завршени дан.

Дневни ток података

Редован циклус почиње пошто генератор заврши упис партиције за претходни дан, чиме се она затвара. `Dagster` потом, по сопственом, унапред одређеном распореду, учитава ту партицију у слој `src`.

Унос и `dbt` трансформације раздвојени су у два посла. Трансформације не покреће засебан распоред, већ сензор који проверава да ли су сви очекивани скупови учитани у табеле слоја `src` и тек тада покреће `dbt`. Тако ново аналитичко стање увек настаје над комплетним улазом за тај дан.

Кад сензор покрене `dbt`, не обрађују се сви модели на исти начин. Начин обраде зависи искључиво од тога шта један ред тог модела представља, односно грануларности. Модели чији један ред представља појединачан догађај обрађују се **инкрементално**: читају само нове или исправљене редове пристигле након претходног извршавања. Модели чији резултат зависи од целокупног стања пролазе кроз **потпуну обраду** (Одељак 1.4): поново се формирају над свим релевантним подацима.

Група модела	**Начин обраде**	**Шта потпуна обрада поново гради**
трансакционе `fact` табеле	инкрементална обрада	—
`int` табеле	потпуна обрада	заједничке међурезултате
агрегатни модели	потпуна обрада	показатеље по ентитету
димензије	потпуна обрада	сва учитана стања и верзије
`fact_enrollments`	потпуна обрада	све уписе

Табела 3.5: Начин обраде различитих група модела

`fact_activity_logs`, `fact_quiz_attempts` и `fact_reviews` садрже појединачне догађаје, па нема потребе за поновним читањем њихове целе историје при сваком покретању. Код `fact_enrollments` тај приступ не би био довољан. Нове активности мењају напредак и статус већ постојећег уписа, због чега се ова табела поново израчунава над целокупним скупом уписа.

Агрегатни модели и историјске димензије такође пролазе кроз потпуну изградњу при сваком извршавању, али из два различита разлога. Агрегатни модел сажима велики број `fact` редова у један ред по ентитету. Нов упис, покушај или рецензија може променити тај збир, па се цео модел рачуна

изнова. Историјске димензије, са друге стране, уопште не користе филтер за нове редове; при сваком извршавању оне поново реконструишу све верзије и периоде важења из целокупног садржаја изворних података о студентима и курсевима. Разлог је могућност закаснеле верзије: она може да помери већ израчунате границе важења, па би без потпуне реконструкције остала погрешно уклопљена у историју.

Ово поновно формирање димензија не мења редове који су у трансакционе `fact` табеле већ уписани јер они задржавају верзију димензије која је важила у тренутку њихове обраде. Промена већ формираних периода важења зато захтева поновну изградњу тих трансакционих `fact` табела, што је одвојено од редовне обраде.

Из описа начина обраде закључује се да дневно `dbt` извршавање није у целини ограничено на један дан и да модели који се поново изграђују временом обрађују све већу историју.

Опоравак након неуспешне обраде

Раздвајање фаза одређује и како се систем опоравља од грешке. Ако унос успе, а `dbt` трансформације не успеју, довољно је поново покренути `dbt` посао над већ учитаним `src` подацима. Неуспех током самог уноса је сложенији случај, јер је `ClickHouse` могао да прихвати само део партиције пре него што је извршавање прекинуто. У том случају понавља се исти унос, за исти дан. Да би систем знао да ли је претходни покушај оставио потпун или делимичан резултат, пореди се број редова у слоју `src` са бројем редова у одговарајућој `S3` датотеци за тај дан: ако се бројеви поклапају, скуп је већ потпуно учитан и прескаче се, а ако не, постојећи редови тог скупа се бришу и цела партиција се учитава испочетка. Оваквим механизмом учитавања поновљено извршавање не ствара дупликате и увек даје исто коначно стање, тј. систем је идемпотентан. (Одељак 1.4)

Обрада историје и потпуна изградња

Поступак учитавања података који одговарају једној партицији проширује се и на историјски период. Почетно или накнадно учитавање историје у слој `src` изводи се као `backfill`: исти унос се покреће редом над низом дневних партиција, дан по дан. Трансформације се при томе не покрећу после уноса сваке партиције јер би `dbt` тада изнова обрађивао све већи део истих података,

а међурезултати би остали неискоришћени. Уместо тога, након учитавања свих изабраних партиција у слој `src`, модели се само једном потпуно изграђују, над целокупном учитаном историјом.

Ако се промени начин на који `dbt` обрађује податке, подаци не морају поново да се учитавају из `S3`-а јер су већ у слоју `src`. Довољно је поново покренути `dbt`, овог пута над свим подацима, не само новим, тако да нова логика обухвати и већ учитану историју. Као и `backfill`, и ово се покреће ручно, одвојено од редовног дневног циклуса.

3.5 Квалитет, поузданост и надгледање података

Провере квалитета распоређене су дуж целог тока података, кроз сваки слој посебно, а не само над коначним резултатом.

Граница `S3`–`src`. Сваки улазни скуп, који одговара једној табели у слоју, има очекиване колоне, па се његова структура проверава пре уписа у `ClickHouse`. Недостајућа или неочекивана колона прекида унос те партиције јер таква датотека не одговара очекиваном облику табеле. Сврха слоја `src` је да сачува тачно оно што је стигло из извора, чак и кад појединачна вредност није исправна. Неуспела конверзија тако постаје видљива као недостајућа вредност, уместо да заустави цео ток без трага о проблематичном реду.

Слој `stg`. Проверавају се обавезни и јединствени кључеви, везе ка повезаним ентитетима, дозвољене категорије и очекивани опсези вредности. Исправност сваке колоне појединачно није довољна јер на пример, покушај квиза може имати валидног студента, курс и садржај, али да сам садржај припада другом курсу, због чега су неопходне и провере међусобних односа између више атрибута.

Слој `int`. Користе се исте технике као у `stg`-у (кључеви, опсези, категорије, везе), сада над спојеним и агрегираним резултатима. Издваја се провера временске логике: датум уписа не сме бити касније од тренутка обраде.

Слој `marts`. Проверавају се правила која настају тек као последица моделовања: за `SCD2` димензије, постојање тачно једне тренутно важеће верзије; за табеле чињеница, јединственост грануларности, везе са димензијама и логичка сагласност изведених мера. На пример поређење цене и попуста реконструисаних из историјских димензија са вредностима забележеним уз упис.

Провере су подељене на критичне и упозоравајуће, у зависности од последица које неправилност може проузроковати. Критичне грешке су на пример: нарушена структура улазне датотеке, недостајући или поновљени кључеви, нарушене везе, преклапање `SCD2` периода, противречне мере. Оне заустављају одговарајућу фазу и спречавају да се њен резултат прихвати као поуздан. Упозорења су на пример: вредност ван очекиваног опсега, непозната категорија, временски однос који захтева проверу. Оне само бележе проблем, али ред се учитава у табелу и остаје доступан за анализу.

Историја извршавања показује не само да ли је посао успео, већ и шта је стварно обрађено. За свако извршавање уноса података бележе се обрађена партиција и датотеке, број учитаних редова, трајање и примењени поступак. Овим, ако провере открију проблем могуће је извршити накнадну истрагу изворних података.

Оперативно надгледање

Исправност података треба разликовати од доступности система. Трансформације могу бити логички исправне, а да обрада ипак није могућа зато што `S3` или `ClickHouse` нису доступни. Због тога се повезивање са ова два сервиса проверава независно од дневног тока обраде. Оваква провера омогућава разликовање инфраструктурног проблема од грешке у одређеној партицији или моделу.

Због аутоматског успављивања `ClickHouse Cloud`-а описаног у Одељку 3.2, провера његове доступности овде добија и додатну улогу: не покреће се произвољно често, него непосредно пре почетка дневног циклуса уноса података, чиме се истовремено потврђује доступност и унапред доводи сервис у активно стање пре него што крену унос и трансформације.

3.6 Ограничења система

Поред граница рада описаних у Одељку 1.3 постоје и услови под којима модел и ток обраде задржавају описано понашање.

Ограничења изворног скупа података

Подаци над којима платформа ради потичу од генератора, па се конфигурација модела ослања на неколико претпоставки о томе како се извор понаша. Прва је правило да студент може само једном да упише исти курс. Захваљујући њему, активности, покушаји квизова и рецензије могу се повезати са уписом на основу студента и курса, без посебног идентификатора уписа у самим догађајима. Ако би поновни упис био дозвољен, овај пар више не би једнозначно одређивао упис.

Друга претпоставка тиче се редоследа у ком верзије студената и курсева стижу. Платформа је пројектована да исправно обради и закаснелу верзију тј. ситуацију у којој накнадно стигла исправка помера период важења неке раније учитане историјске верзије (Одељак 3.4). Затворене дневне партиције значе да се такав случај не решава сам од себе: постојећи редови трансакционих табела чињеница не повезују се аутоматски са новом верзијом, па исправка историјског контекста остаје одвојен, ручни корак. Реч је о хипотетичком случају у ком би исправка из извора имала пословно време (`_effective_at`) раније од верзије која је већ учитана и већ искоришћена у табелама чињеница – то генератор никад није произвео, па није обухваћено тестом у поглављу 5.

Треће ограничење тиче се самог генератора, тачније поступка генерисања: `snapshot` стања који чита и ажурира при сваком дневном покретању (Одељак 3.2) расте са бројем ентитета, па са растом броја студената, курсева и отворених уписа расту и потрошња меморије и време потребно да се стање прочита и поново упише.

Четврто ограничење тиче се провере комплетности партиције при поновном учитавању: она пореди само број редова у `ClickHouse`-у са бројем редова у `S3` датотекама. Ако би извор ретроактивно исправио вредност у већ учитаном реду (нпр. погрешно унет податак исправљен недељу дана касније), а број редова остане непромењен, платформа то не би препознала јер би партиција и даље изгледала комплетна и упис би се прескочио. Решење би подразумевало поређење и садржајног отиска партиције, на пример `checksum`-а изведеног из `S3 ETag` вредности сваке датотеке доступних без додатног читања садржаја.

Овај случај генератор коришћен у раду не производи, па ова провера није део реализоване платформе нити је тестирана.

Капацитет обраде

Као што је описано, редовна `dbt` обрада није у целини инкрементална. Модел `fact_enrollments`, димензије, међумодели и агрегатни модели поново се формирају над свим релевантним подацима. Зато ће са растом историје расти и количина података коју ови модели обрађују током дневног извршавања.

Оркестрација, унос података и `dbt` клијент покренути су на једној `EC2` инстанци фиксног капацитета. Реализовано решење нема хоризонтално скалирање ни високу доступност. Због тога капацитет овог дела платформе зависи од ресурса једне инстанце, а њен отказ би прекинуо обраду до поновног успостављања окружења. Због истог ограничења, `backfill` над већим бројем партиција може да покрене онолико паралелних извршавања колико је партиција затражено, па при већим опсезима њихов број треба ограничити.

Глава 4

Имплементација

Имплементација је подељена у неколико целина које одговарају функционалностима и одговорностима представљеним у Поглављу 3. Сав код платформе налази се у једном `GitHub` репозиторијуму, организованом по одговорности сваког дела (Табела 4.1).

Линк ка `GitHub` репозиторијуму: cloud-data-platform

Део пројекта	**Улога у имплементацији**
`infra`	`Docker` слике и зависности извршног окружења (Одељак 4.1)
`data` фолдер	`Python` скрипте за почетно и дневно генерисање података (Одељак 4.2)
`dwh` фолдер	`SQL` миграције за иницијализацију база, `src` табела и табеле `dim_date`
`dagster_project` фолдер	Ресурси, `asset`-и, послови, распореди и сензор (Одељак 4.3)
`dbt_project` фолдер	Модели слојева `stg`, `int` и `marts`, тестови и seed подаци (Одељак 4.4)

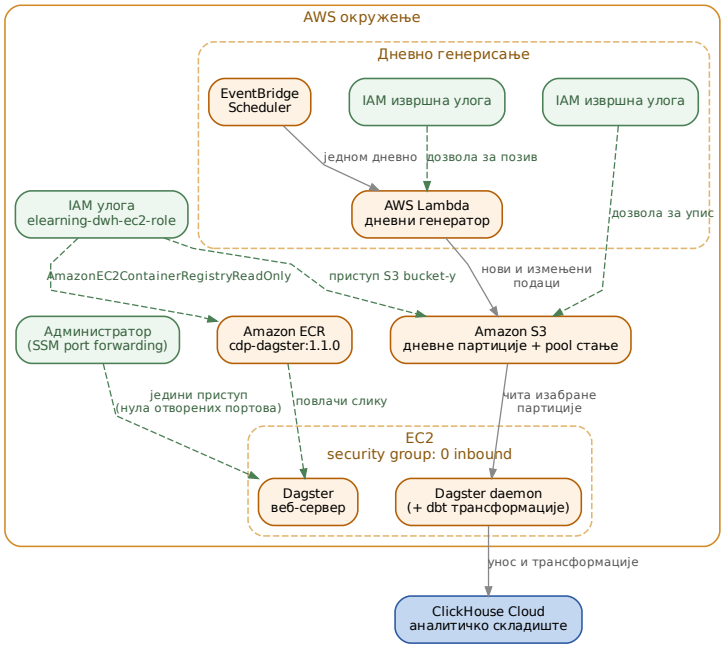
Табела 4.1: Главне целине репозиторијума

Пре првог покретања тока извршавају се `SQL` миграције из `dwh` фолдера, којима се креирају базе `src`, `stg`, `int` и `marts`, осам улазних табела, као и статичка димензија датума.

4.1 Извршно окружење

За извршно окружење изабрана је инфраструктура у облаку: оркестрација и унос података покрећу се на `Amazon Web Services` (`AWS`) инфраструктури, а аналитичка база у управљаном сервису `ClickHouse Cloud`. Распоред компоненти и начин њиховог повезивања приказани су на Слици 4.1.

За `EC2` инстанцу изабран је тип `m7i-flex.large` (2 `vCPU`, 8 `GB RAM`, `Ubuntu x86_64`), мања инстанца се показала меморијски неодговарајућом за истовремено извршавање `Dagster` веб-сервера и `daemon`-а, укључујући `dbt` трансформације које `daemon` покреће. Варијанта `flex` намењена је променљивом оптерећењу и повољнија је од стандардне, што одговара платформи која интензивније ради само током дневне обраде. Са инстанцом је повезан `EBS` диск капацитета 30 `GiB` типа `gp3`, који се користи за оперативни систем, контејнере и трајне податке сервиса. Аналитичка база покренута је као `ClickHouse Cloud` сервис са једном репликом од 12 `GiB` меморије и 3 `vCPU`.



Слика 4.1: Инфраструктура и `IAM` улоге у `AWS` окружењу

`Dagster` компоненте покренуте су у два `Docker` контејнера. Веб-сервер обезбеђује кориснички интерфејс, док `daemon` покреће распореде, сензоре и `dbt` команде. Оба контејнера користе исте директоријуме пројекта и исто складиште

Dagster метаподатака. Пошто оба контејнера читају и пишу у исто складиште, веб-сервер у свом интерфејсу приказује и извршавања која је покренуо daemon, иако сам веб-сервер није директно учествовао у њиховом покретању. Dagster Docker слика већ садржи dbt и потребне зависности, па на EC2-у није потребан посебан dbt контејнер. docker-compose.yml преузима слику из приватног ECR репозиторијума. Слика се гради локално и убацује на ECR.

Локално и AWS окружење користе исти код и исту Compose конфигурацију. Разликују се искључиво по датотеци са променљивама окружења која се прослеђује при покретању. У њој се задају назив S3, AWS регион, адреса ClickHouse сервиса и подаци за повезивање.

Приступ AWS сервисима решен је IAM улогама. EventBridge има дозволу за позивање Lambda функције, која приступа одговарајућој S3 локацији. Са друге стране, EC2 улога омогућава приступ S3-у и преузимање Docker слике из ECR-а. На EC2 инстанци и у Lambda функцији не чувају се трајни AWS кључеви. Потребне дозволе добијају преко додељених улога. Дозволе над S3 ресурсима ограничене су искључиво на операције које обрада користи: излиставање садржаја, као и читање, упис и брисање појединачних објеката.

На EC2 инстанци нису отворени долазни портови. Администратор приступа Dagster интерфејсу преко SSM port forwarding-а. Интерфејс тако није јавно доступан, али се и даље може користити за праћење и покретање обрада.

Исти принцип најмањих привилегија примењен је и у самом ClickHouse-у: подразумевани кориснички налог ограничен је на локални приступ и само читање, док сва обрада иде преко посебног налога.

4.2 Формирање и складиштење изворних података

Изворне скупове података и њихове међусобне везе производе две Python скрипте:

- generate_csv.py формира почетну вишегодишњу историју
- generate_daily_increment.py наставља свакодневно генерисање

Оба генератора производе датотеке са истим колонама за сваки извор, па унос касније не мора да разликује који их је генератор произвео.

Датотеке се објављују на исту путању у S3-у:

```
raw/<skup>/year=YYYY/month=MM/day=DD/<naziv_datoteke>.csv
```

где је `<skup>` један од осам изворних скупова (нпр. `activity_logs`), а `<naziv_datoteke>` име тог скупа са временском ознаком настанка датотеке (нпр. `activity_logs_20260315_000000.csv`).

Ако за неки од осам скупова тог дана нема нових редова (нпр. ниједан нови инструктор тог дана није регистрован) генератор за њега уопште не прави датотеку.

Идентификатори ентитета и догађаја, као и вредности атрибута и број догађаја по ентитету, формирају се коришћењем фиксно дефинисане псеудослучајне вредности (*seed* вредност 42 за `Python` генератор и библиотеку `Faker`). Исто покретање генератора зато увек производи исти скуп података. Оба генератора читају и исти референтни скуп попуста `subscription_discounts.csv`. Почетни генератор га преузима из dbt пројекта, а копија исте датотеке укључена је у `Lambda` пакет дневног генератора.

Правила генерисања вредности

Да би подаци деловали реалистичније, велики део вредности не бира се равномерно случајно, већ на начин који опонаша стварне обрасце понашања на сличним платформама.

Датум регистрације студента бира се тако да их временом стиже све више: користи се троугаона расподела чији је врх на 75% посматраног периода, па број регистрација постепено расте ка крају, слично стварном расту корисничке базе платформе. Тип претплате додељује се у следећим размерама: 40% бесплатна, 35% месечна, 15% годишња и 10% корпоративна, уместо да свака од четири опције буде подједнако вероватна. На сличан начин, ни инструктори се не придружују платформи равномерно кроз читав период: 70% њих придружује се у прве две године опсега, а остатак постепено током преосталог времена.

Што је студент дуже присутан на платформи, то је већа шанса да ће променити претплату, а исто важи и за цену курса – у просеку то током целог периода доживи око петине свих ентитета, уместо да сви имају исту, фиксну шансу без обзира на то колико дуго постоје. Промене при том иду само у

једном смеру: претплата се само надограђује (бесплатна → месечна → годишња → корпоративна), а цена курса само расте (за 10, 15 или 20% при свакој промени) – никад обрнуто.

Број активности и покушаја квизова по упису не бира се потпуно случајно, већ прати забележени напредак тог уписа: упис са 60% напретка генерише активности над приближно 60% садржаја курса. Тако упис, активности и покушаји увек причају усклађену причу о истом студенту, уместо да буду у међусобној противречности. Сваки следећи покушај истог квиза има већи очекивани резултат од претходног, као да студент вежбом постаје бољи, а активност запослених корисника чешће пада у вечерње сате и викенде него током радног дана.

Рецензије се генеришу само за завршене уписе, и то за 60% њих, а не за сваки завршен упис аутоматски. Оцене намерно чешће бивају високе него ниске: за оцене од 1 до 5 вероватноће су редом 2%, 5%, 15%, 40% и 38%, баш као и на стварним платформама за учење. Мали, намерно убачен део рецензија (око 8%) припада студентима који уопште нису уписали тај курс – ови редови постоје да би тест који проверава да ли је рецензент заиста уписан на курс имао стваран случај да открије, а не да увек аутоматски прође.

Табела 4.2 приказује по један илустративан пример генерисаног реда за сваки од осам изворних скупова (приказана су само нека од поља, ради прегледности).

Извор	**Пример генерисаног реда (изабрана поља)**
`instructors`	Danielle Johnson, земља `de`, област `web_development`, статус `active`, придружена `2024-01-05`
`courses`	„Code a personal splash page in 1 hour”, `web_development`, ниво `Beginner`, цена $20.00, статус `published`
`content`	„Lesson 1: Matrix Revolutionary Schemas”, тип `video`, редослед 1, трајање 600 s
`students`	Gary Hall, тип `employee`, претплата `monthly`, износ 19.99
`enrollments`	редовна цена 195.0, плаћена цена 195.0, напредак 63.57%, недовршен
`activity_logs`	тип `video_watch`, трајање 1140 s, завршено
`quiz_attempts`	покушај бр. 1, резултат 21.42/50 (праг 35), није положио
`reviews`	оцена 5/5, верификована, 6 корисних гласова

Табела 4.2: Пример по једног генерисаног реда за сваки изворни скуп

Стварни подаци о курсевима

Полазни подаци о курсевима преузети су из датотеке `udemy_courses.csv`[14]. Верзија датотеке која се користи у раду садржи 3.678 редова и 12 колона са подацима о курсевима објављеним између 2011. и 2017. године. Курсеви припадају областима `Business Finance`, `Graphic Design`, `Musical Instruments` и `Web Development`. Приликом учитавања, на основу изворног `course_id` идентификатора уклања се шест дуплираних редова, након чега генератор формира 3.672 јединствена курса.

Генератор не пресликава целу датотеку у изворни модел, већ користи само поља која су потребна за формирање реалистичне основе курсева. Начин њихове употребе приказан је у Табели 4.3.

Поља у `udemy_courses.csv`	**Употреба у генератору**
`course_title`	преузима се без измене као назив курса
`subject`	пресликава се у категорију курса интерне шеме (нпр. `Business Finance` → `business_finance`)
`level`	пресликава се у ниво тежине интерне шеме (нпр. `All Levels` → `all_levels`)
`price`	формира почетну цену курса; мањи део вредности намерно остаје без знака валуте, ради каснијих провера чишћења
`published_timestamp`	релативни положај у изворном опсегу (2011--2017) пресликава се у распон `DATA_START_DATE`–`DATA_END_DATE`, тако да се временски распоред курсева чува али усклађује са периодом платформе
`num_lectures`	одређује број генерисаних садржајних елемената курса (највише 30)
`content_duration`	користи се за израчунавање трајања видео-лекција у колони `duration_seconds`
`url`, `is_paid`, `num_subscribers`, `num_reviews`	не користе се при формирању генерисаног извора

Табела 4.3: Употреба поља јавног скупа података

Подаци о броју претплатника и рецензија намерно се не користе, јер се уписи, активности и рецензије формирају као засебни синтетички догађаји и морају остати међусобно усклађени. Изворни идентификатор курса такође се не преноси у платформу; након дедупликације сваком курсу додељује се нови `UUID`. Преостали подаци формирају се помоћу библиотеке `Faker`.

Почетни историјски скуп

Временски опсег историјског скупа контролише се преко две променљиве окружења, `DATA_START_DATE` и `DATA_END_DATE`. Прва одређује од ког датума почиње генерисана историја, а друга до ког датума сеже, закључно са њим. У пракси, `DATA_END_DATE` представља датум покретања скрипте, чиме се обезбеђује да историјски скуп обухвати податке до текућег дана. У овом раду `DATA_START_DATE` има вредност `2024-01-01`, а `DATA_END_DATE` `2026-07-23`.

Редослед генерисања прати зависности домена. Најпре настају инструктори, курсеви, њихов садржај и студенти. Пре формирања уписа генеришу се и историјске промене студената и курсева. Промена не преписује почетни запис, већ се представља новим редом са истим природним кључем и новим `updated_at` временом.

Из ових редова граде се временски уређене листе промена по студенту и курсу. Приликом формирања сваког уписа генератор у њима проналази претплату и цену које су важиле у `enrolled_at` тренутку. Важећа цена се уписује у `list_price`, а након примене попуста у `paid_price`. Тако изворни запис садржи износ који је студент стварно платио, док се цена и попуст касније могу независно реконструисати у аналитичком моделу.

Након уписа генеришу се активности и покушаји квизова, и то само над садржајем курса који је студент уписао. Када су сви догађаји познати, функција `finalize_enrollments` на основу њих ажурира време последње активности, напредак и статус завршетка одговарајућег уписа. Рецензије се формирају након овог корака, јер верификована рецензија зависи од постојећег уписа и његовог завршетка. Додаје се и мањи број неверификованих рецензија, које представљају оцене корисника без одговарајућег уписа.

Дневни наставак генерисања

Дневни генератор не почиње од празног стања, јер је за нове догађаје неопходно коришћење постојећих ентитета и наставак раније започетих уписа.

Прелазак са историјског на дневно генерисање се зато одвија у четири корака:

- почетни генератор формира вишегодишњу историју и објављује је у S3;
- те партиције се учитавају у src табеле;
- скрипта build_pool_snapshot.py на основу src табела формира почетно стање генератора;
- дневни генератор се укључује и одатле наставља.

Тренутно стање скупа података чува се у S3 објекту _state/pool.json. Он садржи познате инструкторе, курсеве, садржаје и студенте, као и ограничен скуп отворених уписа који се могу наставити наредних дана. Ова датотека служи само генератору и не улази у аналитички ток. Скрипта за сваког студента и курс бира верзију са најновијим updated_at временом и издваја скорије незавршене уписе. Историјски генератор производи податке закључно са DATA_END_DATE, а дневни генератор при сваком покретању генерише податке за дан пре стварног тренутка извршавања. Уколико се распоред не активира дан након историјског учитавања, дани између остају непокривени, па се генерисање за њих покреће ручно, пре укључивања распореда.

При сваком покретању програм учитава pool, након чега може додати нове ентитете, промену претплате или цене, нове уписе и догађаје над постојећим уписима. Када нова активност промени стање уписа, у дневну CSV датотеку додаје се корективни ред са истим enrollment_id и новим напретком, временом последње активности и евентуалним датумом завршетка. Претходни ред се не мења у S3-у нити у слоју src; staging модел ће касније изабрати последњу верзију. На крају успешног генерисања ажурирано стање уписује се назад у pool.

За извршавање у AWS Lambda окружењу дефинисана је улазна функција lambda_handler. EventBridge Scheduler позива је после поноћи, а она као циљни датум увек поставља претходни завршени дан. Временска зона Lambda функције подешена је на Europe/Belgrade, исто као и Dagster партиције.

Праћени пословни пример

Ради јаснијег приказа извршавања dbt трансформација, у наставку је кроз цео ток праћен упис студенткиње Amy Goodwin. Пример је изабран јер обухвата промену цене курса и претплате студенткиње, више активности и покушаја,

завршетак курса и рецензију. Користи се у одељцима о SCD2 димензијама, табелама чињеница и агрегатним моделима.

Елемент	Податак из примера	Шта се њиме показује
студенткиња	Amy Goodwin; годишња, а касније enterprise претплата	избор историјске верзије студенткиње
курс	Surpassing Your Kickstarter Goals; цена мењана 30 → 34,49 → 37,93 USD	формирање историјских верзија курса
упис	4. јануар 2025; важећа цена 34,49 USD, попуст 30%, плаћено 24,14 USD	временско повезивање и провера прихода
активности	12 активности, прва 11. јануара	израчунавање напретка студента на курсу
оцењивани садржаји	9 покушаја над 5 квизова или задатака	повезивање догађаја са садржајем и укупан број покушаја
изабрани квиз	3 покушаја решавања квиза Quiz_9: _Generate_Mission-Critical_ Interfaces, 24-25. фебруара	разлика између детаљне и агрегатне табеле покушаја
завршетак и рецензија	курс завршен 12. марта; рецензија остављена 14. марта, оцена 3,5 и верификован упис	редослед догађаја и повезивање рецензије са уписом

Табела 4.4: Елементи праћеног примера

Пример је једнозначно одређен вредношћу enrollment_id = 4be1b219-61b1-49eb-b526-6fd688cd316b, која се поново користи при провери резултата у поглављу о евалуацији(Глава 5).

4.3 Учитавање у аналитичку базу

Унос је реализован кроз осам Dagster asset-а са префиксом raw_, подељених у две групе: entities за описне ентитете и events за догађаје. Сваки представља један изворни скуп и његову одговарајућу src табелу. Иако asset-и имају посебна имена и зависности, сам поступак читања и уписа налази се у заједничкој функцији _ingest_daily_partition. Тако су правила за одређивање S3 путање, проверу структуре, поновно покретање и бележење метаподатака примењена на исти начин над свим скуповима.

Заједничка функција за унос не приступа директно ни S3-у ни ClickHouse-у. Комуникација са та два спољна система издвојена је у Dagster ресурсе: S3Resource иницијализује S3 клијента, лоцира објекте и преузима CSV садржај, док ClickHouseResource успоставља везу са базом и извршава упите уписа.

За сваки скуп и датум заједничка функција формира S3 префикс (нпр. raw/students/year=2025/month=01/day=04/), а S3Resource проналази све CSV датотеке испод њега. Свака датотека се привремено учитава у меморију Dagster процеса као pandas DataFrame, при чему се све изворне вредности читају као текст. Празне вредности постају празан текст, а сваком реду додаје се _source_file са пуним кључем S3 објекта из ког потиче. Ако партиција садржи више датотека, њихови DataFrame-ови се након провере колона спајају у један.

Очекиване колоне сваког скупа дефинисане су у ingestion_schema.py. У тој датотеци је за сваки од осам скупова унапред записано који називи колона морају да постоје. Одмах по читању сваке датотеке, њене колоне се упоређују са том листом; тек ако све датотеке прођу проверу, њихови редови се спајају и могу бити уписани. Проверава се само да ли су одговарајуће колоне присутне, а не и да ли је свака појединачна вредност у њима исправна.

Ако структура прође проверу и партиција није раније комплетно учитана, ClickHouseResource редове обједињеног DataFrame-а трајно уписује у одговарајућу src табелу. Све изворне колоне, укључујући и _source_file, у слоју src се чувају као String. Свака src табела има и колону _loaded_at која се израчунава у тренутку уписа. Преко ње сваки ред добија време свог доласка у базу. Преко префикса у _source_file проверава се да ли су редови одређене партиције већ присутни у src табели, док _loaded_at служи за праћење времена њиховог стварног учитавања.

Обрада једне дневне партиције

Подаци се уносе партиција по партицију, а не одједном за целу историју, будући да сваки Dagster run обрађује тачно један изабрани датум. Партиција има облик YYYY-MM-DD и може да садржи више датотека истог скупа, на пример основне редове и засебну датотеку са историјским променама. Захваљујући оваквој подели, редовно извршавање, опоравак од грешака и ретроактивно попуњавање података (енгл. *backfill*) ослањају се на идентичан механизам обраде.

За изабрани скуп и датум најпре се формира одговарајући `S3` префикс. Ако испод њега нема ниједне `CSV` датотеке, обрада се завршава материјализацијом са нула редова. У супротном се датотеке појединачно читају и проверавају, а њихови редови спајају у један `DataFrame`.

Након тога се број прочитаних редова пореди са бројем редова који у циљној `src` табели већ имају исти префикс у колони `_source_file`. Целокупан поступак може се представити следећим псеудокодом:

```
префикс = путања(скуп, датум_партиције)
датотеке = пронађи_csv_датотеке(S3, префикс)
ако нема датотека:
    забележи материјализацију са нула редова
    заврши обраду
улазни_редови = прочитај_и_провери(датотеке)
постојећи_број = преброј(src_табела, префикс)
ако је постојећи_број једнак броју улазних редова:
    прескочи упис
иначе:
    ако постоје редови за префикс:
        обриши их
    упиши целу партицију
    провери број сачуваних редова
```

Псеудокод обухвата сва три исхода уноса; провера сачуваних редова на крају је последња линија одбране, неслагање прекида извршавање. Поступак се ослања на претпоставку да се затворена дневна партиција у систему `S3` накнадно не мења. Под тим условом, поређење броја редова довољно је да се разликују комплетно и непотпуно учитана партиција.

Колона `_loaded_at` служи да се препозна шта је ново стигло од последњег покретања и да се, ако накнадно стигне исправка, узме последња верзија реда.

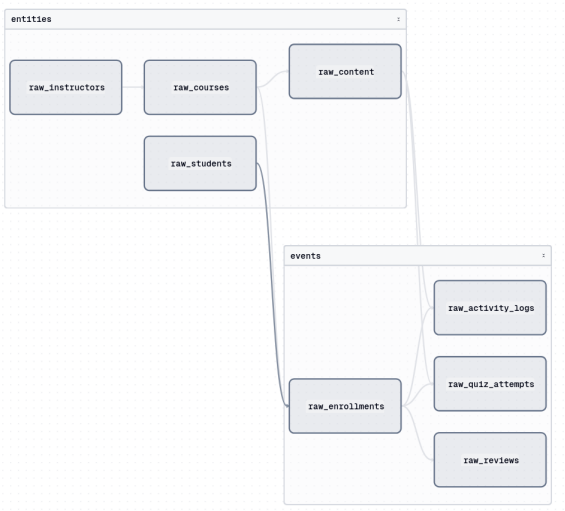
Приликом сваке материјализације *asset*-а, `Dagster` ускладиштеним ресурсима придружује следеће метаподатке:

```
материјализација = {
    датум_партиције: ...,
    S3_префикс: ...,
    број_датотека: ...,
    редова_пре: ...,
    редова_после: ...,
    трајање: ...,
    load_action: ''нов упис'' | ''прескакање'' | ''замена''
}
```

`load_action` бележи који је од три исхода описана псеудокодом примењен: „нов упис”, „прескакање” или „замена”. Ако за неки скуп тог дана нема ниједне датотеке, запис се ипак бележи, само са нула редова.

Зависности између ingestion asset-а

Зависности између изворних скупова изражене су кроз `Dagster asset` граф. `raw_courses` директно зависи од инструктора, `raw_content` од курсева, а `raw_enrollments` од студената и курсева. Активности и покушаји квизова зависе од уписа и садржаја, док рецензије зависе од уписа. `Dagster` на основу тих веза одређује редослед покретања корака за исту дневну партицију. Уколико унос курсева не успе, `Dagster` ће аутоматски обуставити извршавање уноса садржаја за тај дан, спречавајући сензор да некомплетно учитану партицију означи као спремну. Слика 4.2 приказује овако одређен граф зависности у `Dagster` корисничком интерфејсу.

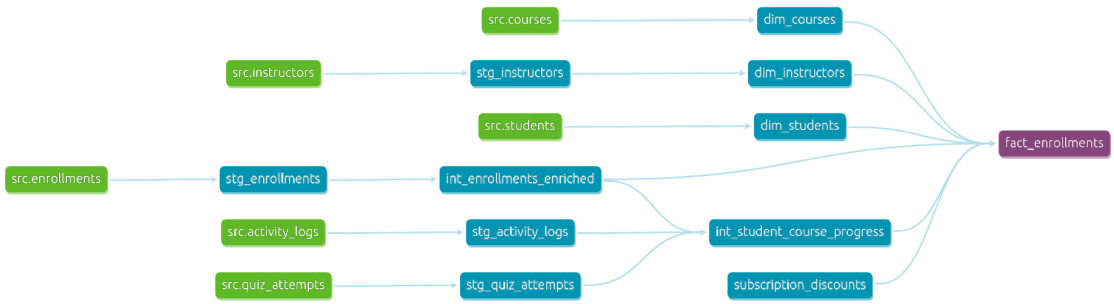


Слика 4.2: Граф зависности

4.4 Трансформација података кроз dbt слојеве

Након учитавања података у `src` табеле, даља обрада се одвија унутар `ClickHouse` система коришћењем `dbt` модела. Пројекат је подељен на три слоја модела: `staging`, `intermediate` и `marts`. Начин редовне обраде сваке групе модела приказан је у Табели 3.5. У имплементацији је ова подела подешена у датотеци `dbt_project.yml`.

Сваки модел путем `source` и `ref` функција дефинише зависности над изворним подацима и другим моделима, а `dbt` на основу тих веза одређује редослед извршавања. Као пример зависности унутар `dbt` пројекта, на Слици 4.3 приказан је граф зависности модела `fact_enrollments`, преузет из документације коју `dbt` генерише.



Слика 4.3: Зависности `dbt` модела при формирању `fact_enrollments`

Staging модели

За сваку `src` табелу дефинисан је одговарајући `stg_*` модел. Ови модели су материјализовани у форми погледа (*views*).

Обављају четири основна корака:

- дедупликацију;
- нормализацију текстуалних вредности;
- конверзију типова;
- формирање једноставних изведених колона.

Неважеће вредности се приликом конверзије не преводе у подразумеване вредности, већ у NULL, чиме се олакшава њихова детекција кроз системске тестове. Ово понашање постигуно је коришћењем ClickHouse функције са наставком OrNull. За правила која се понављају на више места користе се макрои: to_bool претвара познате текстуалне ознаке (нпр. 'yes', 'no') у логичке вредности, а parse_price уклања знак валуте и цену претвара у број.

Staging модели израчунавају и једноставне вредности које зависе само од једног реда. На пример, stg_quiz_attempts рачуна проценат резултата и означава поновљени покушај, док stg_students маскира број телефона.

На примеру модела stg_students може се видети начин уклањања дупликата. У src.students може постојати више верзија истог студента, односно више редова са истим student_id. Функција ROW_NUMBER те редове групише према идентификатору студента и додељује им редне бројеве на основу времена измене:

```
ROW_NUMBER() OVER (
    PARTITION BY student_id
    ORDER BY
        parseDateTimeBestEffortOrNull(
            nullIf(trim(updated_at), '')
        ) DESC NULLS LAST,
        _loaded_at DESC,
        _source_file DESC
) AS _row_num
```

Дедупликација се затим спроводи филтером WHERE_row_num=1. На овај начин се за сваки природни кључ задржава искључиво један запис — онај са најновијим временским отиском измене. Ако више записа има исто updated_at време или ова вредност недостаје, редослед се додатно одређује према времену учитавања _loaded_at и називу изворне датотеке _source_file.

Исти образац користи се и код осталих ентитета, уписа и рецензија, сваки са својим природним кључем јер сви они имају updated_at атрибут. Активности и покушаји квиза немају updated_at јер су то су догађаји, не ентитети који се мењају. Зато се код њих редослед одређује по времену учитавања _loaded_at: задржава се последње учитани запис. Тако staging модел враћа тачно један ред за сваки природни кључ, изабран по унапред дефинисаном редоследу, па поновно извршавање над истим улазом увек бира исти запис.

Intermediate модели

`Intermediate` слој садржи два модела:

- `int_enrollments_enriched` – Преузима последње познато стање сваког уписа из `stg_enrollments` модела и издваја колоне које користи више наредних модела: идентификаторе студента, курса и уписа, цене, напредак, статус и времена активности. Не додају се тренутни подаци о студенту и курсу будући да они не морају одговарати стању у моменту настанка уписа. На пример, у праћеном примеру курс је после уписа поскупео, а студент прешао на другу претплату.
- `int_student_course_progress` – Повезује упис са активностима и покушајима квизова. За сваки упис израчунава број активности и покушаја решавања квизова, време гледања, време прве активности, као и просечан и најбољи резултат на квизу. Лево спајање табела задржава и уписе који још немају активност.

Пошто се резултат касније користи у `fact_enrollments`, ови модели су материјализовани као табеле и рачунају само једном током `dbt` извршавања.

Димензије и `SCD2` верзије

`dim_date` се, за разлику од осталих marts модела, не гради из изворних података: SQL миграција је једнократно попуњава датумима од 2020. до краја 2027. године и изводи календарске атрибуте као што су месец, квартал, дан у недељи и ознака викенда. Пошто њен садржај не зависи од извора, ова табела није део свакодневног `dbt` извршавања.

`dim_instructors` и `dim_content` настају из одговарајућих `staging` модела и садрже тренутно стање инструктора и садржаја. Њихови кључеви добијају се функцијом `sipHash64` над природним кључем, па исти улаз увек даје исти резултат.

`dim_students` и `dim_courses` морају да сачувају историју промена. Зато читају све верзије непосредно из `src` табела, а не `staging` моделе у којима је већ изабрано само тренутно стање. Због тога се основно чишћење вредности понавља и у овим моделима. Издвајање у заједнички међумодел било би изводљиво, али би такав модел морао да чува све историјске верзије сваког ентитета, што би непотребно удвостручило обим података.

Код студента нову верзију покреће промена претплате, типа корисника или компаније. Код курса прате се цена и статус.

Курс у бази носи три различита времена. `created_at` је фиксан тренутак настанка курса и не мења се кроз верзије. `updated_at` бележи тренутак сваке појединачне промене (цене или статуса) — извор га евидентира кад год до промене дође, али може и недостајати. `_loaded_at` нема везе са самим курсом; то је тренутак када је тај ред стигао у базу, чисто техничка евиденција уноса.

Верзије се формирају према следећим правилима:

- **Пословно време верзије** – тренутак стварног настанка промене у изворном систему (`updated_at`), а не тренутак приспећа тог реда у базу. Ако `updated_at` недостаје, користи се `_loaded_at` као резерва. Ово време одређује редослед и границе верзија, не редослед учитавања.
- **Један ред по пословном тренутку** – извор понекад пошаље исправку за исти тренутак промене (нпр. накнадно откривена грешка у унетим подацима). Такви редови не представљају две различите верзије, већ исту верзију у две варијанте; Задржава се она варијанта која је последња учитана у систем (дефинисана највећом вредношћу атрибута `_loaded_at`).
- **Праћени атрибути и поређење са претходном вредношћу** – праћени атрибути су колоне чија промена дефинише нову верзију (за курс: цена и статус). Сваки ред се пореди са претходним редом истог ентитета по вредностима тих колона. Проблем настаје када је вредност непозната (`NULL`): поређење тада не препознаје промену, чак ни ако се цена стварно променила ка непознатој вредности или из ње. Зато се непозната вредност, пре поређења, замењује вредношћу која се у стварним подацима не може појавити – за цену курса то је `-1` (цена је увек ненегативна или непозната), а за атрибуте попут статуса курса или назива компаније празан низ карактера – чиме промена постаје видљива и када једна од страна поређења има непознату вредност.
- **Издвајање стварних промена** – задржава се прва верзија ентитета и сваки ред у коме се, по претходном поређењу, стварно променио бар један праћени атрибут. Редови без промене се одбацују, јер описују исту верзију као претходни ред.
- **Границе важења** – `valid_from` означава почетак периода у коме је верзија важила, а `valid_to` његов крај. Прва верзија по правилу почиње

датумом настанка ентитета (креирање курса, регистрација студента), а не сопственим пословним временом – чиме се период важења ослобађа од `_loaded_at` ако прва промена нема забележено `updated_at`. Свака следећа верзија почиње сопственим пословним временом. `valid_to` верзије једнак је пословном времену наредне верзије; последња верзија нема наредну, па добија границу у далекој будућности, чиме је и означена као тренутно важећа (*is_ current*).

– **Кључ верзије** – сурогат кључ који јединствено идентификује тачно ову верзију ентитета, не ентитет уопште. Добија се функцијом `sipHash64` над природним кључем (нпр. `course_id`) и пословним временом верзије. Пошто је функција детерминистичка, исти улаз увек даје исти кључ, па поновна изградња димензије не мења већ постојеће везе у табелама чињеница.

Наредна табела показује овај поступак примењен на курс из праћеног примера.

updated_at	price_usd	нова верзија	valid_from	valid_to
2024-10-08	30,00	да, прва верзија	2024-10-08	2025-01-02
2025-01-02	34,49	да, цена промењена	2025-01-02	2025-11-09
2025-11-09	37,93	да, цена промењена	2025-11-09	2099-12-31

Табела 4.5: Формирање верзија курса из праћеног примера (`dim_courses`)

У примеру се датум креирања курса поклапа са временом прве забележене измене јер генератор синтетичких података намерно поставља `updated_at` прве верзије на исто време као `created_at`. У општем случају то не мора важити, зато прва верзија користи `created_at` директно уместо сопственог пословног времена. Курс из примера такође не мења статус, тако да табела показује само цену као окидач промене; измена статуса би на идентичан начин иницирала нову верзију, независно од цене.

Последња верзија је тренутно важећа (енгл. *is_ current*), препознаје се по граничној вредности `2099-12-31` у `valid_to`.

Исти поступак примењен је и на студенткињу из праћеног примера. Њена годишња претплата важила је од 01.12.2024. до 29.06.2025. године, када је почела нова верзија са `enterprise` претплатом. Пошто је упис настао

04.01.2025. године, модел га повезује са годишњом претплатом. При формирању `fact_enrollments` време уписа се користи за избор тадашње верзије и студенткиње и курса: за студенткињу се бира годишња претплата, а за курс цена од 34,49 долара.

Упит који имплементира ове кораке налази се у `dim_courses.sql` у пројекту; исти образац користи и `dim_students.sql`.

Табеле чињеница и временско повезивање

Табела `fact_enrollments` регенерише се прилииком сваког `dbt` извршавања, јер се напредак и статус постојећег уписа мењају. Модел повезује последње познато стање уписа са показатељима напретка из слоја `int` (број активности, покушаја и резултати квизова). На основу времена `enrolled_at` бира верзије студента и курса које су тада важиле.

```
JOIN {{ ref('dim_courses') }} c
  ON e.course_id     = c.course_id
 AND e.enrolled_at  >= c.valid_from
 AND e.enrolled_at   < c.valid_to
```

Вредност `valid_from` припада периоду, док `valid_to` не припада. Због тога се упис настао тачно у тренутку почетка нове верзије везује искључиво за нову верзију. Исти поступак примењује се и при избору историјске верзије студента.

У праћеном примеру упис је настао док је важила верзија са ценом од 34,49 долара, па се повезује са том вредношћу, а не са каснијом ценом од 37,93 долара. На исти начин бира се годишња претплата коју је студенткиња тада имала, уместо касније `enterprise` претплате.

Тако модел добија тадашњу претплату, попуст из референтне `seed` табеле `subscription_discounts`, цену курса и инструктора. Изворна колона `paid_price` чува се као `enrollment_revenue` и представља стварни приход. Реконструисана цена и попуст остају посебне колоне за анализу и проверу. Модел садржи и број активности и покушаја, време до прве активности, резултате квизова и ознаку неактивног уписа. За мере које зависе од текућег датума користи се макро `as_of_date`, којем се по потреби може проследити фиксна вредност ради добијања идентичног резултата при поновној изградњи, уместо зависности од дана извршавања.

Табеле `fact_activity_logs`, `fact_quiz_attempts` и `fact_reviews` обрађују се инкрементално. Поступак је исти у сва три модела:

```
нови_редови =
    редови припремног слоја за које важи
        _loaded_at >= највећи _loaded_at у циљној табели

обриши постојеће редове са истим природним кључем
упиши нове_редове
```

Услов користи оператор `>=` јер више редова може имати исто време учитавања. Користи се стратегија `delete+insert`, једна од инкрементaлних стратегија које `dbt-clickhouse` подржава. Она је изабрана уместо простог додавања (*append*), будући да `ClickHouse` не подржава модификацију постојећих записа на начин на који то реализују трансакционе базе. Без ње би исправка истог догађаја произвела дупликат уместо да замени стари ред. Природни кључеви су `activity_id`, `attempt_id` и `review_id`.

Сваки догађај повезује се са верзијом студента и курса која је важила у време његовог настанка. Са уписом се повезује преко пара студент–курс, у складу са правилом да исти студент не може више пута уписати исти курс. Активности и покушаји добијају и кључ садржаја, док се датумски кључеви формирају у облику `YYYYMMDD`, што је и приматни кључ табеле `dim_date`.

Накнадна промена `SCD2` историје не мења аутоматски раније редове инкрементaлних табела чињеница. Уколико такве промене утичу на историјске догађаје, неопходно је извршити комплетно регенерисање модела применом опције `full-refresh`.

Агрегатни модели

Платформа садржи три модела са сажетим показатељима. Они не замењују `fact` табеле, већ омогућавају брже извршавање најчешћих аналитичких упита. `agg_course_stats` даје број уписа, стопу завршетка, приход и оцене по курсу. `agg_student_stats` сумира уписе и потрошњу по студенту. `agg_quiz_attempt_stats` приказује исход свих покушаја једног студента на истом квизу.

Ови модели се поново формирају при сваком покретању `dbt` јер нови упис, рецензија или покушај квиза може променити постојећи резултат.

Ток праћеног примера кроз моделе

Како се описани кораци надовезују један на други приказано је у Табели 4.6, на основу улазних података из Табеле 4.4. Овде се илуструје улога сваког модела у обради примера, док се стварни резултат упита над изграђеним слојем `marts` детаљније анализира у Поглављу 5.

Модел	Улога модела у обради примера
`stg_courses`	Цена $34,49 претворена је у бројчану вредност 34,49
`dim_courses`	Формиране су верзије са ценама 30, 34,49 и 37,93
`dim_students`	Сачуване су годишња и каснија `enterprise` претплата
`int_student_course_progress`	12 активности и укупно 9 покушаја над оцењиваним садржајима сведено је на један ред напретка
`fact_enrollments`	Изабрани су годишња претплата, попуст 0,30 и цена 34,49; изворно плаћени износ од 24,14 долара сачуван је као приход
`fact_quiz_attempts`	Три покушаја над изабраним садржајем `Quiz 9: Generate Mission-Critical Interfaces` сачувана су као три засебна реда
`agg_quiz_attempt_stats`	Иста три покушаја сведена су на један ред: пролаз из другог покушаја, најбољи резултат 84,92% и просек 71,41%
`fact_reviews`	Рецензија је повезана са уписом, док су оцена 3,5 и изворна ознака верификованог уписа сачуване у моделу

Табела 4.6: Праћени пример кроз `dbt` моделе

4.5 Повезивање у аутоматизовани ток

Унос података и њихове даље трансформације у јединствени ток повезује `Dagster`. Његова конфигурација обухвата осам `asset`-а за унос изворних скупова података, `asset`-е за изградњу `dbt` модела, два `asset`-а за проверу доступности сервиса, ресурсе за приступ `S3`-у, `ClickHouse`-у и `dbt`-у.

На основу ових компоненти дефинисана су четири радна задатка: дневни унос података, редовне dbt трансформације, потпуна `dbt` изградња и надгледање доступности ресурса. Дневни распоред покреће унос, посебан распоред покреће проверу доступности непосредно пре њега (образложено у

одељку 3.5), а сензор повезује завршен унос са dbt трансформацијама. Тиме Dagster централизовано дефинише редослед, услове и логику извршавања свих процеса.

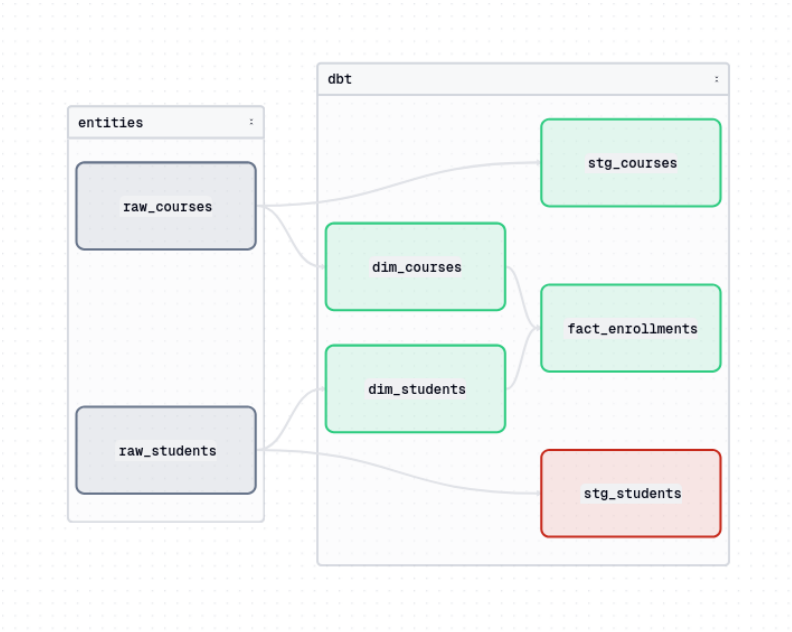
Интеграција dbt модела

Унутар Dagster окружења, сваки dbt модел се третира као појединачни asset, што омогућава преглед зависности између модела у интерфејсу, редослед њиховог извршавања и исход сваке појединачне трансформације. Сви трансформациони модели покрећу се кроз једну заједничку функцију, приказану у наставку.

```
@dbt_assets(manifest=dbt_project.manifest_path)
def elearning_dbt_assets(context, dbt, config):
    args = (
        ['build', '--full-refresh']
        if config.full_refresh
        else ['build']
    )
    yield from dbt.cli(
        args,
        context=context
    ).stream()
```

При редовном извршавању параметар full_refresh има вредност False, па се позива команда dbt build. Она формира моделе и покреће њихове тестове у редоследу зависности. Метода stream прослеђује ток извршавања у Dagster, тако да се за сваки модел може видети да ли је успешно формиран или је обрада прекинута грешком. Редовни посао daily_dbt_job користи ову конфигурацију без додатних параметара. За потпуну изградњу dbt_full_refresh_job поставља full_refresh на True, па се извршава команда dbt build-full-refresh. Тада се и инкрементални модели поново формирају над целокупном историјом.

Сваки Dagster asset има свој кључ (AssetKey) који представља јединствен назив по коме се идентификује и повезује са другима у графу зависности. dbt извори (source() позиви ка src табелама) обично добијају свој, посебан кључ, који не одговара ниједном постојећем asset-у.



Слика 4.4: Повезивање уноса и трансформација у оркестрационом графу

Тиме би на графу настао изолован чвор, уместо да се повеже са `raw_*` *asset*-ом кроз који су подаци иницијално учитани. Посебно правило доделе кључева то решава: сваки `dbt` извор се повезује баш са тим `raw_*` `asset`-ом. Тако граф приказује један непрекинут ток, од уноса у `src` табеле до финалних модела.

Дневни унос података

Дневни унос и пратеће `dbt` трансформације координисани су кроз два одвојена радна задатка:

- `daily_ingestion_job` ког покреће распоред `daily_ingestion_schedule`: активира се сваког дана у 6 часова по `Europe/Belgrade` временској зони и бира партицију за претходни завршени дан. Једно извршавање обрађује тачно једну партицију, чиме одређује који датум `S3` података се учитава.
- `daily_dbt_job` који нема распоред већ га покреће сензор `dbt_trigger_sensor`, који прати статус најновијих материјализација за свих осам улазних `asset`-а. Његова логика објашњена је следећим псеудокодом:

```
прочитај последњу материјализацију сваког улазног скупа
ако нека материјализација недостаје:
    сачекај
ако материјализације не припадају истом датуму:
    сачекај
у супротном:
    покрени редовни посао трансформације
```

Сензор `dbt_trigger_sensor` периодично поново проверава услове за покретање, па би, без додатне заштите, сваки пут кад су услови испуњени могао поново да затражи покретање `daily_dbt_job`-а за исти дан. Ова функција се реализује путем механизма `run_key`, где захтев за покретање добија јединствени идентификатор састављен од префикса `dbt_` и датума партиције (нпр. `dbt_2026-03-15`). Пошто је тај кључ исти при свакој наредној провери за исти дан, `Dagster` препознаје да је захтев са тим кључем већ поднет и игнорише понављање.

Датум партиције користи се само за одабир `run_key`-а и не прослеђује се `dbt` моделима као ограничење – `dbt build` увек чита цео садржај слоја `src`, не само један дан. Зато инкрементални модели сами проналазе шта је ново: пореде колону `_loaded_at` са временом свог претходног извршавања, независно од тога која је `Dagster` партиција управо покренула трансформацију. Модели који зависе од целокупног стања немају ту разлику бећ се они при сваком извршавању у потпуности поново формирају, без обзира на партицију.

Историјски унос података

Почетна историја учитава се поступком `backfill`, коришћењем истог посла `daily_ingestion_job` као у редовном раду. Опсег одговара периоду генерисања историјског скупа (Одељак 4.2). У систему `Dagster` ручно се бирају дневне партиције из тог периода, а за сваки датум покреће се засебно извршавање које чита одговарајућу путању у систему `S3`.

Током овог поступка сензор `dbt_trigger_sensor` се привремено деактивира како би се спречило прерано иницирање трансформација након сваке појединачно учитане партиције. Тек када се све изабране партиције учитају у слој `src`, једном се ручно покреће посао `dbt_full_refresh_job`. Потпуна изградња формира аналитичке моделе над целокупним садржајем слоја `src`. Након њеног успешног завршетка сензор се поново укључује и платформа наставља редован дневни рад.

Исти посао користи се и када промена SCD2 или инкрементaлне логике захтева поновну обраду старих догађаја. Он нема распоред и покреће се ручно, јер замењује постојеће резултате инкрементaлних модела.

Backfill над већим бројем партиција покреће онолико паралелних извршавања колико је партиција затражено, па се при већим опсезима њихов број може ограничити компонентом QueuedRunCoordinator у датотеци dagster.yaml.

4.6 Валидација и надгледање

Провере су распоређене дуж тока обраде, тако да се проблем уочи што је могуће раније. Током уноса контролишу се структура и садржај дневних датотека, док се након трансформације проверавају кључеви, везе и правила аналитичких модела. Доступност система S3 и ClickHouse прати се одвојено, јер успешна веза са сервисом не гарантује исправност самих података.

Провере током уноса и трансформација

Пре уписа у ClickHouse, функција validate_csv_schema пореди колоне прочитане CSV датотеке са очекиваном шемом. Ако колона недостаје или се појави неочекивана колона, извршавање се прекида и подаци се не уписују.

После провере структуре, исти asset ради и проверу садржаја. Ове контроле извршавају се над учитаним подацима, па не прекидају унос. Код уписа, активности и покушаја квизова траже се поновљени природни кључеви унутар дневног скупа. Код скупова који садрже везе ка другим ентитетима броје се редови за које студент, курс или садржај не постоје у одговарајућој табели слоја src. За курсеве се додатно проверава усклађеност формата изворно записане цене са очекиваним шаблоном.

Покретање наредбе dbt build, описано у претходном одељку, поред изградње модела извршава и тестове дефинисане у пројекту. У датотекама YAML наведене су провере обавезности и јединствености кључева, дозвољених вредности, бројчаних опсега и веза између модела. Правила која зависе од више колона или табела издвојена су у засебне SQL тестове. Њима се, на пример, проверава да активност и покушај квиза не претходе упису, припадност садржаја догађаја наведеном курсу, као и подударност ознаке пролаза са резултатом и прагом квиза. За димензионе табеле SCD2 типа додати су специјализовани тестови no_overlapping_ranges и exactly_one_current_record. Први потврђује одсуство преклапања периода важења верзија истог ентитета, док други

гарантује да студент или курс поседују тачно једну тренутно важећу верзију. Посебно се проверава да ли нова верзија курса доноси стварну промену цене или статуса. У табелама чињеница контролишу се њихова грануларност, везе са димензијама и сагласност изведених показатеља. Код уписа то обухвата и поређење изворно плаћеног износа са реконструисаном ценом и попустом.

Критични тестови носе ниво ознаке `error` и њихово неиспуњавање прекида процес `dbt build`, па аналитички модели не могу бити прихваћени као успешно изграђени. Одступања која треба испитати, али не чине модел структурно неупотребљивим, имају ниво `warn`. Пројекат је конфигуриран тако да се редови који нису прошли тест чувају у посебним табелама ради накнадне анализе.

Оперативно надгледање

Оперативно надгледање реализовано је помоћу два непартиционисана `Dagster asset`-а, груписана у посао `monitoring_job`. Први извршава операцију `head_bucket` над изабраним `S3 bucket`-ом ради провере доступности сервиса и валидације приступних права у оквиру конфигурације. Други над системом `ClickHouse` извршава упит `SELECT version()` и бележи верзију доступног сервиса, као и трајање и стварни број покушаја конектовања. Веза са базом се успоставља тек при овом упиту, не унапред; ако тада не успе, покушај се понавља, до одређеног максималног броја покушаја. Тако провера сачека буђење успаваног сервиса (Одељак 3.2), уместо тренутног пријављивања грешке.

Систем `Dagster` поред тога чува статус и дневник сваког извршавања, као и метаподатке о броју прочитаних, уписаних, прескочених или замењених редова.

raw_activity_logs

STEP_OUTPUT

Yielded output "result" of type "Any". (Type check passed).

row_count	252
partition_row_count	252
previous_row_count	0
files_processed	1
partition_date	2026-08-09
source_prefix	raw/activity_logs/year=2026/month=08/day=09/
load_action	inserted
duration_seconds	0.38

Слика 4.5: Метаподаци материјализације `raw_activity_logs` asset-а у `Dagster` интерфејсу

Глава 5

Евалуација

Евалуацијом решења проверава се да ли ток података даје очекиване резултате и да ли систем поседује способност опоравка након евентуалних грешака. Затим се анализирају време обраде, употреба ресурса под већим оптерећењем, трошкови и ограничења реализованог решења.

Подаци о трајању послова преузети су из базе извршавања система `Dagster`, резултат изградње модела из датотеке `run_results.json` [3], док су трајање и потрошња ресурса при упитима очитани из системске табеле `system.query_log` [13]. Параметри оптерећења `EC2` инстанце из сервиса `Amazon CloudWatch` [8]. Унутар директоријума `docs/report_scripts` и `docs/evaluation` сачувани су добијени резултати и коришћене скрипте.

5.1 Функционална исправност и поузданост

Пример аутоматизованог тока

За проверу је изабран дневни циклус од 10.08.2026. године.

Посао `daily_ingestion_job` обрадио је партицију за 09.08.2026. и успешно материјализовао свих осам `raw_* asset`-а. Након тога је сензор активирао `daily_dbt_job`, који над свим моделима извршава команду `dbt build`. Резултат тог извршавања приказан је у Табели 5.1.

Део извршавања	Број	Резултат
модели материјализовани као погледи	8	успешно
модели материјализовани као табеле, пуна изградња	10	успешно
инкрементални модели	3	успешно
помоћни скуп података `seed`	1	успешно
тестови квалитета података	489	480 прошло, 9 упозорења

Табела 5.1: Резултат извршавања команде `dbt build`, 10.08.2026.

Девет упозорења показује да су тестови пронашли одступања, приказана у Табели 5.2.

Врста одступања	Број тестова	Пронађени редови
догађај је забележен пре уписа на курс	2	431 активност и 99 покушаја решавања оцењиваног садржаја
догађај није повезан са важећом верзијом података о студенту	2	108 активности и 26 покушаја решавања оцењиваног садржаја
упис није повезан са важећом верзијом података о студенту	5	исти упис у свих пет провера: недостају `student_key`, тип претплате, попуст, ознака плаћања и старост налога

Табела 5.2: Упозорења тестова квалитета података, 10.08.2026.

Свих девет упозорења узроковано је истом појавом, накнадно идентификованом у самом генератору података. Дневни генератор смешта упис и припадајуће догађаје у исти дан, али им не уређује редослед по сату, услед чега се мањи број догађаја евидентира неколико сати пре самог уписа. Исти записи јављају се и у другој групи провера: догађај забележен пре уписа налази се и пре прве верзије података о студенту. Временско повезивање зато не проналази важећу верзију, па `student_key` остаје празан. На овај начин је обухваћено мање од 0,1% свих активности и покушаја решавања оцењиваног садржаја. Одступање не нарушава квалитет података о студенту, курсу нити дану догађаја, због чега је оцењено као упозорење, а не као критична грешка.

За разлику од упозорења, неуспех критичног теста прекида процес извршавања команде `dbt build`. То је потврђено у јулу 2026. године, када је тест `unique_stg_students_email` открио колизију. Две адресе електронске поште из изворних података разликовале су се искључиво по величини слова, али пошто `stg` слој `email` нормализује на мала слова, обе су после тога постале идентичне и нарушиле јединственост. Узрок је отклоњен директно у генератору података уместо ублажавањем критеријума теста, будући да оваква одступања морају онемогућити даљу изградњу модела.

Провера резултата на пословном примеру

Упит над изграђеним слојем `marts` проверава да ли платформа стварно производи резултат описан у Одељку 4, за упис студенткиње `Amy Goodwin` (идентификатор `4be1b219-61b1-49eb-b526-6fd688cd316b`, упит доступан у `docs/report_scripts/business_example.sql`). Упит повезује моделе чињеница `fact_enrollments`, `fact_quiz_attempts` и `fact_reviews` са димензионим табелама `dim_students`, `dim_courses` и `dim_content`.

Упит проверава и три могуће аномалије: покушаје решавања пре уписа, покушаје везане за садржај другог курса и рецензије остављене пре завршетка курса. Ниједан од ова три случаја није детектован.

Остале добијене вредности у потпуности одговарају Табели 4.4: цена и тип претплате потичу од верзија курса и студенткиње важећих у тренутку уписа, а не тренутног стања, а примењени попуст даје тачно плаћени износ забележен уз упис.

Опоравак и поновљена обрада партиција

Способност платформе да настави рад после прекида, без губитка или дуплирања података, ослања се на механизам провере броја редова описан у Одељку 3.4.2: упис се прескаче ако се број редова у `ClickHouse`-у поклапа са бројем редова у `S3` датотекама, а понавља ако не.

Идемпотентност. Поновним покретањем већ учитане партиције (05.01.2024) овај механизам је нумерички потврђен. За седам од осам `raw_*` `asset`-а тог дана није постојала ниједна датотека у `S3`-у, па су материјализовани са нула редова, у складу са поступком описаним у одељку 4.3.1. Једини `asset` за који је постојала датотека, `raw_instructors`, вратио је метаподатке приказане у Табели 5.3.

Поље	Вредност
`row_count`	0
`partition_row_count`	1
`previous_row_count`	1
`load_action`	`skipped_already_loaded`
`duration_seconds`	0,17

Табела 5.3: Метаподаци поновљене материјализације асета `raw_instructors` за већ учитану партицију

Број редова у `src` слоју пре и после овог извршавања остао је идентичан – један ред за ту партицију, 25.110 укупно у бази – чиме је, независно од Dagster метаподатака, нумерички потврђено да поновљено извршавање не мења постојеће стање.

Опоравак од стварног прекида. Значај овог механизма показује инцидент из августа 2026. године, када је процес дневног уноса прекинут. Извршавања за партиције 01.08. и 03.08. нису успешно окончана услед преласка сервиса `ClickHouse Cloud` у стање мировања, при чему је конфигурисано време за успостављање везе било недовољно. До грешке је долазило пре почетка учитавања, па подаци из тих партиција делимично нису уписани.

Следећа успешно учитана партиција указала је на одређена ограничења сензора. Наиме, сензор је проверавао да ли последње материјализације свих осам `raw_* asset`-а припадају истој партицији, али није проверавао да ли између партиција постоји пропуштен дан. Зато је покренуо `daily_dbt_job`, иако претходна партиција није била учитана. Тестови су затим пријавили недостајуће везе између података.

Након идентификованог инцидента, повећан је временски интервал за успостављање везе и додати су поновни покушаји повезивања. За опоравак самих података није вршена ручна модификација садржаја табела путем упита. Уместо тога, 05.08.2026. године иницран је `backfill` захтев за укупно пет партиција (у периоду од 31.07. до 04.08), чиме су обухваћене и пропуштене и суседне партиције. Изворне датотеке су остале сачуване унутар сервиса `S3`, па њихово поновно генерисање није било потребно. Табела 5.4 приказује трајање појединачних извршавања и број обрађених редова, представљен као сума редова из осам изворних скупова за посматрану партицију.

Партиција	Број редова	Трајање извршавања (s)
31.07.2026.	557	32,3
01.08.2026.	588	28,2
02.08.2026.	492	25,5
03.08.2026.	467	24,1
04.08.2026.	477	23,5
Укупно	**2.581**	–

Табела 5.4: Трајање извршавања партиција у оквиру `backfill` захтева

Од формирања захтева до завршетка последње партиције протекло је 45,1 секунду, док је од почетка првог до завршетка последњег извршавања прошло 33,9 секунди. Свих пет процеса извршено је паралелно и сви су успешно завршени.

Након учитавања, ручно је иницран редован `daily_dbt_job`, без примене опције `-full-refresh`. Извршавање је трајало 68,4 секунде и успешно је завршено. Потпуна реизградња свих модела није била неопходна јер су накнадно учитани редови добили нову вредност `_loaded_at`, што је омогућило инкрементaлним моделима да их препознају као нове податке. По поновном укључивању, сензор је за последњу материјализовану партицију аутоматски иницирао још једно `dbt` извршавање, које је такође успешно завршено.

Захваљујући овом механизму, `backfill` процес је безбедно обухватио и суседне партиције које су раније већ биле учитане.

5.2 Перформансе

Перформансе платформе посматране су кроз пет узастопних дневних циклуса, од 06.08. до 10.08.2026. године. Детаљна мерења dbt модела, аналитичких упита и оптерећења EC2 инстанце изведена су над последњим од њих, чије је трајање готово једнако медијани узорка, па тај циклус није изузетак у односу на остале. Тај циклус обрадио је партицију 09.08.2026, па посматрани скуп обухвата податке закључно са тим даном.

Трајање дневног циклуса и свежина података

Два броја описују брзину дневног циклуса. **Активно трајање** представља време од почетка уноса до завршетка `dbt` посла, односно трајање саме обраде.

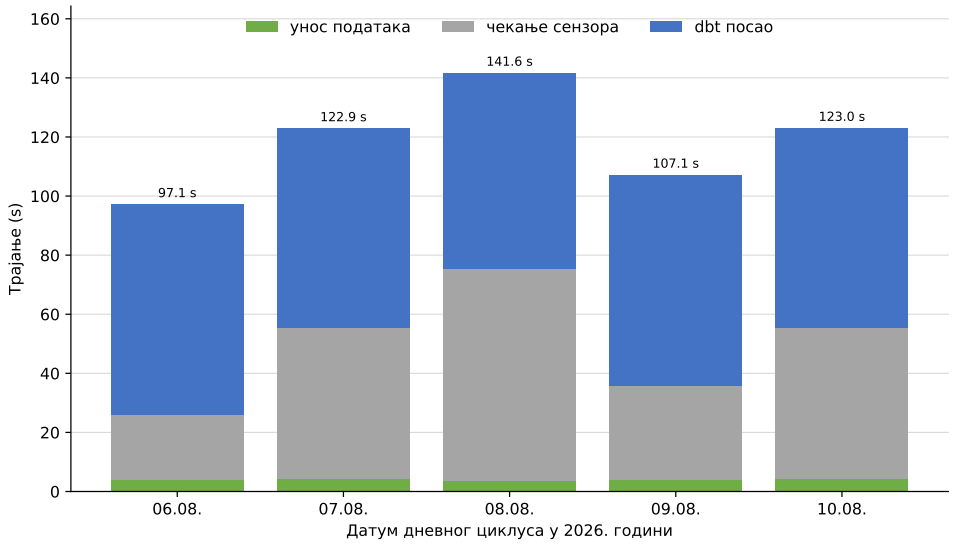
Свежина података мери колико се чека да нови подаци постану доступни за анализу: рачуна се од тренутка када је последња датотека тог дана пристигла у `S3` до завршетка `dbt` посла, када су освежене табеле у слоју `marts`. Разлика између ова два броја представља планирано чекање, будући да се дневни посао покреће тек у 06:00 часова.

У узорак су ушла само аутоматски покренута и успешно завршена дневна извршавања, сва у истој конфигурацији окружења.

Активно трајање обухвата унос изворних података, време чекања сензора и `dbt` посао. На Слици 5.1 види се да је процес уноса кратак и уједначен, док највећу разлику између дана прави време чекања да сензор покрене `dbt` посао.

Активно трајање било је од 97,1 до 141,6 секунди, са медијалном вредношћу од 122,9 секунди. На дневном нивоу додавано је између 430 и 493 реда, чиме је изворни слој на крају посматраног периода обухватао укупно 1.444.359 записа.

Подаци су у систему `S3` били доступни приближно у 00:06 часова, док се дневни циклус покретао у 06:00. Из тог разлога, свежина података за свих пет дана износила је око 5 сати и 56 минута, при чему највећи део овог периода чини планирано чекање да `dbt` сензор крене са радом.

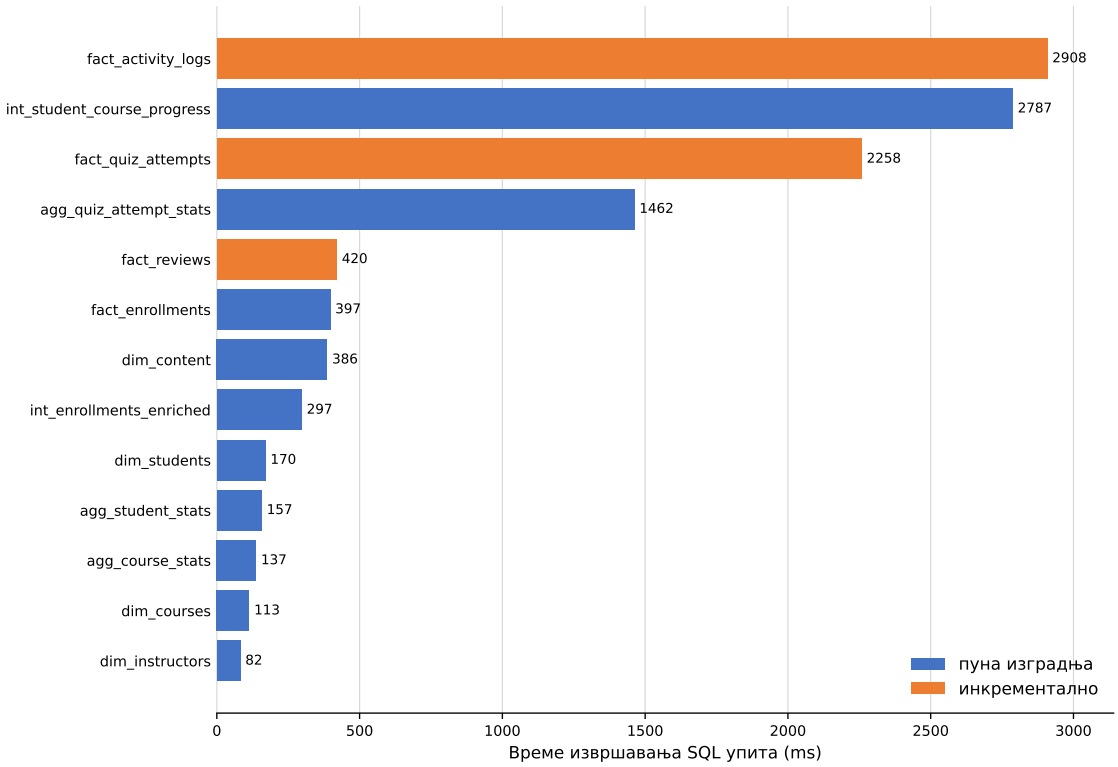


Слика 5.1: Трајање корака дневног циклуса, од 06.08. до 10.08.2026.

Трајање и потрошња ресурса по dbt моделу

За циклус од 10.08. издвојен је, из `system.query_log`, главни SQL упит сваког `dbt` модела. За моделе изграђене као табела то је упит који креира и попуни табелу, а за инкрементaлне моделе упит који уписује нове или измењене редове. За сваки такав упит забележени су време извршавања, максимално заузеће меморије, као и број прочитаних и уписаних редова. Ово време односи се само на сам SQL упит, не на цео `dbt` посао јер посао додатно обухвата кораке попут тестова и успостављања конекције.

Слика 5.2 приказује 13 модела материјализованих у форми табела, чије резултате `dbt` уписује у базу. Осам модела припремног слоја није приказано јер су материјализовани као погледи (енгл. *views*) и не врше упис података, док тестови нису обухваћени овим приказом. Плавом бојом означена је пуна изградња табеле, док наранџаста боја означава инкрементaлну обраду нових или измењених редова.



Слика 5.2: Време извршавања SQL упита по `dbt` моделу, 10.08.2026.

Табела 5.5 детаљније приказује четири упита са најдужим трајањем извршавања.

Модел	Трајање SQL упита (ms)	Највећа употреба меморије (MB)	Прочитани редови	Уписани редови
fact_activity_logs	2.908	693,6	3.093.291	473
int_student_course_progress	2.787	573,8	1.319.724	55.073
fact_quiz_attempts	2.258	556,6	2.312.609	119
agg_quiz_attempt_stats	1.462	460,5	616.789	279.900

Табела 5.5: Ресурси главних SQL упита четири најзахтевнија модела

Најдуже трајање забележено је код упита модела `fact_activity_logs`. Он је генерисао упис 473 нова реда, али је током поступка њиховог повезивања са подацима о уписима, студентима, курсевима и садржајем прочитао укупно 3.093.291 ред. Сличан поступак примењује и модел `fact_quiz_attempts`. Модели `int_student_course_progress` и `agg_quiz_attempt_stats` такође обрађују велики број података, док су сви преостали упити завршени у року од највише 420 милисекунди.

Инкрементална материјализација смањује број уписаних, али не и број прочитаних редова. Разлог је временско повезивање: да би нови догађај добио верзије студента и курса које су у том тренутку важиле, упит мора претражити историјске `SCD2` димензије и табелу уписа у пуном обиму, независно од броја пристиглих догађаја за тај дан (Одељак 3.5). Са растом историјских података расте и комплексност тог повезивања, иако дневни прираст остаје исти. У посматраном циклусу, ниједан главни упит модела није трајао дуже од 2,9 секунде.

Време извршавања аналитичких упита

У датотеци `docs/report_scripts/analytical_performance_queries.sql` налазе се два упита која проверавају време одговора аналитичког слоја: `top_courses_by_revenue` приказује десет курсева са највећим приходом, а `enrollment_summary_by_category_and_subscription` над моделима `fact_enrollments` и `dim_courses` израчунава показатеље уписа по категоријама курсева и врстама претплате. Упити су извршени 10.08, након завршетка посматраног `dbt` посла, сваки поновљен десет пута уз приказану медијалну вредност. Приказана су времена искључиво за извршавање унутар система `ClickHouse`, без времена преноса резултата до крајњег корисника.

Табела 5.6 приказује резултате оба упита. Осам пута мањи обим прочитаних редова директно се одражава на време одговора.

Упит	**Прочитани редови**	**Резултати**	**Медијана (ms)**
`top_courses_by_revenue`	7.785	10	7
`enrollment_summary_by_category_and_subscription`	59.254	17	17

Табела 5.6: Време извршавања два аналитичка упита, 10.08.2026.

Да би се измерило колико сам аналитички слој доприноси, други од та два упита је извршен и непосредно над слојем `src`. Тај упит сачуван је у датотеци `docs/report_scripts/src_layer_equivalent_query.sql` и мора самостално да реализује трансформације које `stg` и `marts` модели већ обављају (дедупликацију уписа, конверзију типова и `SCD2` повезивање). Оба упита враћају идентичан резултат, а Табела 5.7 приказује разлику у трајању и потрошњи меморије, приближно шест пута спорије извршавање и четири и по пута већу меморијску потрошњу над слојем `src`. Ово мерење је спроведено 12.08.2026, а појединачни резултати извршавања сачувани су у датотеци `docs/evaluation/performance/src_vs_marts_query_2026-08-12.csv`.

Слој	**Медијана (ms)**	**Меморија (MB)**
`marts`	15	15,2
`src`	95,5	68,5

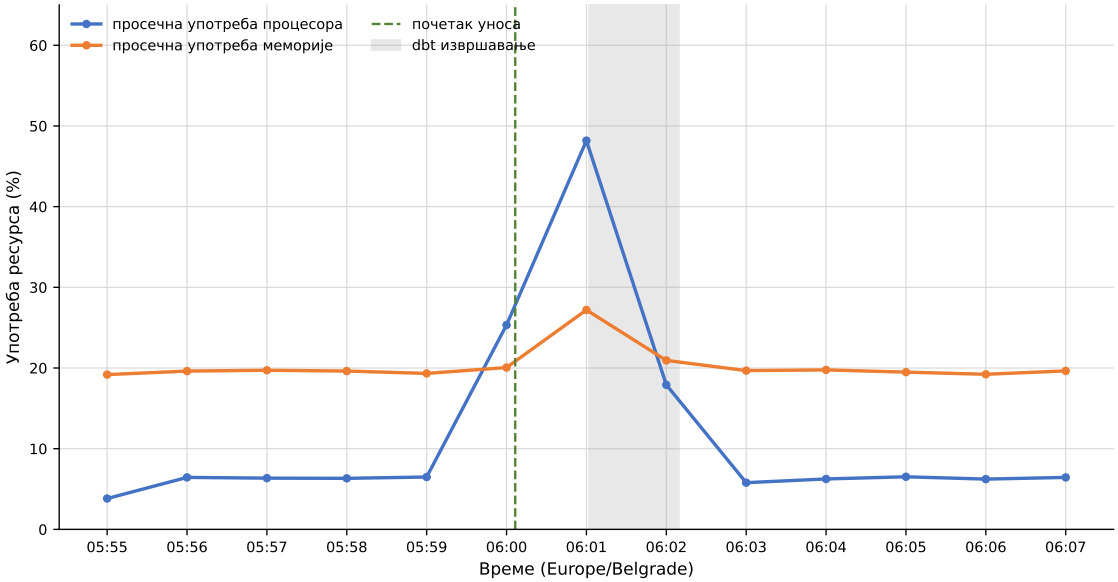
Табела 5.7: Упит `enrollment_summary_by_category_and_subscription` над `marts` и `src` слојем, 12.08.2026.

Искоришћеност ресурса `EC2` инстанце

Оптерећење `EC2` инстанце праћено је путем `Amazon CloudWatch Agent` сервиса, који на сваких 10 секунди прикупља параметре искоришћености процесора и меморије, чекања на диск, као и мрежног саобраћаја. Интервал обухвата период од пет минута пре почетка уноса до пет минута након завршетка `dbt` посла.

За анализу је изабран дневни циклус од 10.08.2026. године. Унос података започео је у 06:00:06 и завршио се у 06:00:10, док је `dbt` посао трајао од 06:01:01

до 06:02:09. Због једноминутног груписања вредности у систему `CloudWatch`, приказани интервал обухвата период од 05:55 до 06:08. Слика 5.3 приказује просечну употребу процесора и меморије по минутима.



Слика 5.3: Употреба процесора и меморије `EC2` инстанце током дневног циклуса, 10.08.2026. (извор: `Amazon CloudWatch`)

Метрика	**Просечна вредност**	**Највећа или укупна вредност**
употреба процесора	11,7%	59,1%
употреба меморије	20,3%	27,8%
чекање процесора на диск	2,0%	37,0%

Табела 5.8: Искоришћеност ресурса `EC2` инстанце током посматраног интервала

Табела 5.8 приказује просечне и максималне вредности мерења у посматраном интервалу. Као највећа вредност узета је максимална забележена појединачна вредност узорака унутар интервала, коју систем `CloudWatch` чува паралелно са просечном вредношћу за сваки минут.

У периоду између 06:00 и 06:02 часова, током уноса и извршавања `dbt` посла, просечна искоришћеност процесора порасла је на 48%, док ван тог

периода инстанца мирује на око 6% процесора и 19% меморије. Чекање на диск је краткотрајно порасло, али је његов просек у целом интервалу био 2,0%, што указује на то да није било сталног застоја.

Утицај броја истовремено покренутих партиција на трајање уноса

Одељак 3.6.2 наводи да `backfill` захтев покреће онолико паралелних извршавања колико је партиција затражено, без унапред постављеног ограничења. Да би се ово понашање проверило, изведен је поступак уноса над 20, потом додатних 50 и на крају додатних 200 већ постојећих партиција (укупно 270), сваки пут покренутих једним `backfill` захтевом. Коришћена је изолована, унапред испражњена `ClickHouse` инстанца.

Табела 5.9 приказује број партиција и три мере: **зидно време** – протекло време од почетка првог до завршетка последњег извршавања; **збир трајања** – збир трајања свих појединачних извршавања, колико би укупно трајало да су извршена једно за другим; и **степен паралелизма** – однос ове две вредности, тј. колико пута је стварно извршавање брже него да су партиције обрађене редом, једна по једна.

Број партиција	„Зидно” време (s)	Збир трајања (s)	Степен паралелизма	CPU у прозору (%)
20	82,1	111,3	1,36	–
50	218,9	236,4	1,08	65,4
200	873,5	1.163,5	1,33	99,9

Табела 5.9: Унос над већим бројем истовремено покренутих партиција

Свих 270 покренутих извршавања успешно је завршено, без ниједног отказивања. Степен паралелизма остаје између 1,08 и 1,36 — знатно испод теоријског убрзања које би одговарало броју истовремено покренутих партиција — што показује да фиксни капацитет инстанце (2 `vCPU`) ограничава стварно постигнуто убрзање, иако сама оркестрација не поставља експлицитно ограничење броја паралелних извршавања.

Ово потврђује и оптерећење процесора: у последњем кораку, при 200 истовремено покренутих партиција, вредности забележене унутар прозора извршавања крећу се од 83,2% до 99,9%, наспрам 65,4% при 50 партиција. Меморија током читавог теста није представљала уско грло — остала је у опсегу од 14% до 28%, знатно испод расположивог капацитета инстанце.

Утицај броја истовремених упита на време одговора

Понашање аналитичког слоја при већем оптерећењу проверено је контролисаним тестом: више корисника који имају сопствену везу ка бази истовремено постављају исти упит, а прати се како се мењају време одговора и проток. Коришћен је упит који чита појединачне уписе и групише их према категорији курса и врсти претплате.

Тест је извршен 10.08.2026. године са једним, пет и десет извршилаца, при чему је сваки од њих поновио упит по 50 пута. Скрипта је покренута на `EC2` инстанци, мерећи време од тренутка слања упита до пријема резултата. Забележени су и време извршавања у бази, број прочитаних редова и максимална меморијска потрошња по упиту. Појединачна мерења сачувана су у датотеци `docs/evaluation/scalability/analytical_load_test_runs_2026-08-10.csv`.

Табела 5.10 приказује медијално време одговора и 95. перцентил - вредност испод које се завршило 95% упита. Број упита у секунди, односно проток, израчунат је на основу броја успешно завршених упита и укупног трајања сваког нивоа теста.

Истовремени извршиоци	**Број упита**	**Медијана (ms)**	**95. перцентил (ms)**	**Упити у секунди**	**Грешке**
1	50	18,5	28,8	48,0	0
5	250	21,9	77,9	134,4	0
10	500	72,5	99,6	141,2	0

Табела 5.10: Резултат аналитичког упита при већем броју истовремених извршавања

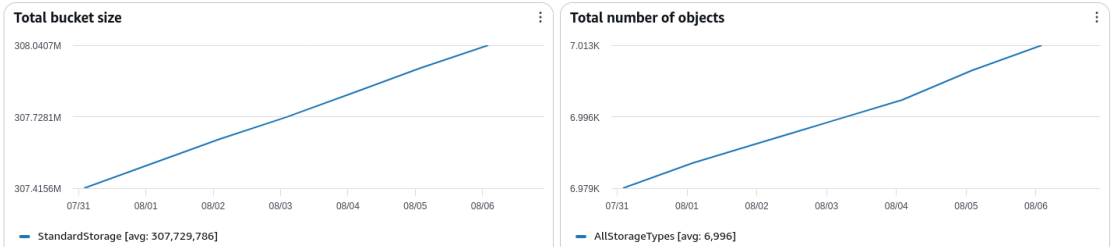
Свих 800 тестираних упита успешно је завршено. Проток остварује значајан раст до нивоа од пет паралелних извршилаца, након чега долази до засићења: прелазак са пет на десет доноси још свега седам упита у секунди, док се медијана времена одговора више него утростручује.

Исто понашање уочава се и код времена забележеном унутар `ClickHouse` система: медијана серверског времена износила је 14, 17 и 65 милисекунди. Максимална меморијска потрошња по појединачном упиту била је 14,6 MB и није расла са повећањем броја извршилаца. То је меморија коју троши један упит, а не сумирана потрошња свих паралелних процеса.

5.3 Трошкови решења

Током развојне фазе платформе коришћени су `AWS Free Tier` и пробни период услуге `ClickHouse Cloud`. Пошто су те погодности временски ограничене, процена месечних трошкова је извршена на основу редовних цена, за садашњи обим података.

Процена је заснована на пресеку стања од 6. августа 2026. године. У изворном слоју се у том тренутку налазило 1.442.556 редова, док је у `S3` складишту било сачувано приближно 0,3 GB података, распоређених у око 7.000 објеката.



Слика 5.4: Промена укупне величине и броја објеката у `S3` складишту

Стање складишта проверено је на два начина. Путем алата командне линије `AWS CLI` излистани су сви објекти унутар `S3` складишта, на основу чега су сабрани њихов број и укупна величина. Добијене вредности затим су упоређене са метрикама које `AWS S3` сервис приказује у конзоли (Слика 5.4).

За оркестрацију процеса користи се `EC2` инстанца типа `m7i–flex.large`, која је стално активна. За разлику од `EC2` инстанце, `ClickHouse Cloud` се активира само током учитавања и обраде података, а затим се аутоматски зауставља након периода неактивности [16] од 15ак минута. У основном прорачуну рачуна се са приближно 30 минута активног рада дневно: сервис се активира у 05:50 ради провере доступности, `dbt` посао се завршава у 06:02, а сервис се успављује 15 минута после последњег упита. То одговара приближно 15 сати активног рада током једног месеца. У наведено време није урачунато произвољно извршавање аналитичких упита корисника, већ само дневни рад платформе. Сваки кориснички упит продужава активан рад сервиса и повећава тај део трошка.

Процена трошкова по услугама приказана је у Табели 5.11.

Услуга	**Основа за процену**	**Трошак**
`Amazon EC2`	`m7i–flex.large`, 730 сати по цени од приближно 0,1147 USD по сату	83,74 USD
`Amazon EBS`	30 GiB `gp3` простора, по цени од приближно 0,0952 USD по GiB месечно	2,86 USD
`ClickHouse Cloud`	приближно 15 сати активног рада месечно и мање од 2 GB података	око 5 USD
`Amazon S3`	приближно 0,3 GB података и око 300 операција уписа и 300 операција читања месечно	мање од 0,05 USD
`AWS Lambda` и `EventBridge Scheduler`	30 извршавања месечно, 512 MB меморије и трајање од приближно 30 секунди	мање од 0,01 USD
Укупно		**приближно 92 USD**

Табела 5.11: Процењени месечни трошкови платформе

Највећи удео процењених трошкова односи се на `EC2` инстанцу. Иако дневно учитавање и трансформације података трају релативно кратко, инстанца је стално активна, па цена оркестрационог окружења не зависи непосредно од трајања дневне обраде.

Код услуге `ClickHouse Cloud` постоји директна веза између трајања обраде и укупних трошкова. При претпостављених 15 сати рада месечно, рачунарски део трошка износи приближно 4,95 USD, уз додатак накнаде за складиштење података. Уколико би сервис у просеку био активан један сат дневно уместо планираних 30 минута, рачунарски трошак би порастао на приближно 9,90 USD, а укупан месечни трошак платформе на око 97 USD.

Избор управљане варијанте значајно утиче и на укупну цену. Ако би се `ClickHouse` систем покретао на засебној `EC2` инстанци типа `m7i.large`, активној током целог месеца, трошак инстанце и диска износио би приближно 91 USD, чиме би укупна цена платформе порасла са око 92 на приближно 178 USD [15].

При тренутном обиму података било би могуће покренути `ClickHouse` и на постојећој инстанци, чиме би се директан трошак смањио. Међутим, у том случају база би делила процесорске и меморијске ресурсе са осталим компонентама система, уз мању изолацију и могућу потребу за већом инстанцом са

растом количине података. Управљана варијанта, из тог разлога, представља повољније решење у посматраном сценарију са малим и повременим оптерећењем, с тим што се тај однос исплативости може променити са дужим трајањем обраде или растом количине података.

Трошкови `S3` складишта, `Lambda` функције и `EventBridge Scheduler` услуге занемарљиви су при посматраном обиму коришћења. Функција током једног месеца користи приближно 450 GB-секунди рачунарског времена, што се уклапа унутар бесплатног месечног пакета `Lambda` сервиса. Међутим, чак и без њега трошак извршавања био би мањи од једног цента. Једно дневно позивање услуге `EventBridge Scheduler` сервиса такође остаје далеко испод дефинисаног бесплатног месечног лимита [15].

Реализованој платформи приступа само особа која је покреће и надгледа. Платформа не поседује јавни кориснички интерфејс нити јој већи број корисника приступа преко интернета. За такву употребу било би потребно додати пријављивање и управљање корисничким правима, конфигурисати и заштитити мрежни приступ и обезбедити да сервис остане доступан при већем броју захтева или отказу неке компоненте. Пошто ти захтеви зависе од броја корисника и количине саобраћаја, њихови трошкови нису укључени у процену.

5.4 Ограничења

- За евалуацију платформе коришћен је контролисан синтетички скуп података, уместо стварних продукционих података.
- Редовни дневни циклус посматран је током периода од пет дана. Детаљна мерења `dbt` модела, аналитичких упита и ресурса односе се на један изабрани дан. Наведени узорак не показује дугорочну стабилност нити понашање при знатно већој историји података.
- Поређење изворног и аналитичког слоја спроведено је над једним пословним упитом, те измерена разлика важи искључиво за посматрани прорачун и тренутну количину података, а не за произвољан упит нити за већу историју.
- `Dagster`, унос података и `dbt` клијент покренути су на једној `EC2` инстанци, па нема хоризонталног скалирања ни високе доступности.

– Процењени трошкови засновани су на јавним ценовницима и претпостављеном времену активности, а не на дугорочним трошковима продукционог система. Трошак може да се промени са ценама услуга, трајањем обраде и количином података.

– Платформа није јавно доступна, па није испитан приступ великог броја спољних корисника, висока доступност, додатни механизми заштите и опоравак целог окружења након отказа `EC2` инстанце.

Глава 6

Закључак

Реализована је аналитичка платформа која податке система за учење на даљину трансформише од изворних датотека у облик погодан за напредну анализу. Дневни скупови података пристижу у `S3` складиште, одакле их `Dagster` по дневним партицијама учитава у `ClickHouse`, док их `dbt` кроз своје слојеве претвара у димензионални модел у финалном слоју. Ток се покреће сам: распоред покреће процес уноса, док сензор чека да сви скупови података буду учитани и тек тада покреће трансформације.

Мерења су показала да такав циклус траје око два минута, од партиције до изграђених модела и извршених тестова. Праћени пословни пример показао је да платформа подржава праћење историјских верзија: цена и тип претплате у табелама чињеница потичу од верзија курса и студенткиње које су важиле у тренутку уписа. Поузданост тока потврђена је на неколико начина: провере квалитета, њих 489 у једном извршавању, открише су стварну грешку у редоследу догађаја коју производи генератор, а не унапред уграђено одступање; после стварног прекида дневног уноса, пет пропуштених партиција надокнађено је једним захтевом, без ручног мењања садржаја табела; поновно покретање већ учитане партиције потврдило је да се тада упис у потпуности прескаче, без промене броја редова; а тест капацитета, спроведен у три узастопна корака до укупно 270 партиција, показао је да инстанца овакво оптерећење обрађује без иједног отказивања, уз пораст употребе процесора до готово 100% и меморију која остаје далеко испод расположивог капацитета. Процењени месечни трошак платформе износи око 92 USD, највећим делом због инстанце која је стално активна.

Над изграђеним финалним слојем извештаји се добијају једним упитом: приход по курсевима и показатељи уписа према категорији и врсти претплате

израчунати су директно из њега, без додатне припреме података. Како би се измерила корист овог слоја исти пословни извештај изведен је на два начина: над моделима финалног слоја и директно над изворним табелама, где упит мора сам да одбаци поновљене верзије, претвори текстуалне колоне у бројеве и датуме и реконструише верзије студената важеће у тренутку уписа. Оба пута резултат је идентичан, при чему је упит над аналитичким слојем трајао 15 милисекунди наспрам 95,5 милисекунди, уз чак четири и по пута мању потрошњу меморије.

Добијени резултати важе за контролисан синтетички скуп података и конкретно извршно окружење током ограниченог временског периода. Оркестрација и трансформације се извршавају на једној инстанци, без хоризонталног скалирања и високе доступности.

Даље унапређење платформе могло би да иде у неколико праваца. Исход сваке провере података имплементираних кроз ток види се у `Dagster` интерфејсу, а редови који нису прошли остају записани у посебним табелама у бази. Збирни приказ резултата и систем за обавештавање електронском поштом при паду критичне провере омогућили би бржу реакцију и лакшу могућност праћења стабилности платформе. Сам скуп провера може се проширити новим правилима, укључујући проверу садржајног `checksum`-а партиције уместо само броја редова, чиме би се препознале и ретроактивне исправке које не мењају број редова. За рад са више корисника историју извршавања требало би преместити са локалне базе на засебан сервис, увести пријављивање и контролу приступа и обезбедити наставак рада при отказу инстанце.

Литература

[1] Reis, J., Housley, M. (2022). *Fundamentals of Data Engineering: Plan and Build Robust Data Systems.* O'Reilly Media.

[2] Kleppmann, M. (2017). *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems.* O'Reilly Media.

[3] dbt Labs. (2026). *dbt Developer Hub: Models, Materializations, Tests, References, and Artifacts.* Доступно на: `https://docs.getdbt.com`.

[4] Databricks. (2026). *What is the Medallion Lakehouse Architecture?* Доступно на: `https://docs.databricks.com/aws/en/lakehouse/medallion`.

[5] Kimball, R., Ross, M. (2013). *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling.* 3rd Edition, John Wiley & Sons.

[6] Abadi, D. J., Madden, S. R., Hachem, N. (2008). *Column-Stores vs. Row-Stores: How Different Are They Really?* Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, 967--980. Доступно на: `https://doi.org/10.1145/1376616.1376712`.

[7] Docker. (2026). *Docker Documentation: Containers and Docker Compose.* Доступно на: `https://docs.docker.com`.

[8] Amazon Web Services. (2026). *AWS Documentation: Amazon S3, Amazon EC2, AWS Lambda, EventBridge Scheduler, IAM, and Amazon CloudWatch.* Доступно на: `https://docs.aws.amazon.com`.

[9] Mell, P., Grance, T. (2011). *The NIST Definition of Cloud Computing.* NIST Special Publication 800-145. Доступно на: `https://doi.org/10.6028/NIST.SP.800-145`.

[10] Dagster Labs. (2026). *Dagster Documentation: Assets, Schedules, Sensors, Partitions, and Backfills.* Доступно на: `https://docs.dagster.io`.

[11] Apache Software Foundation. (2026). *Apache Airflow Documentation: DAGs*. Доступно на: https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html.

[12] ClickHouse. (2026). *dbt-clickhouse: ClickHouse dbt Adapter*. Доступно на: https://github.com/ClickHouse/dbt-clickhouse.

[13] ClickHouse. (2026). *ClickHouse Documentation: Introduction and System Tables*. Доступно на: https://clickhouse.com/docs.

[14] Kaggle. (2020). *Udemy Courses*, скуп података. Доступно на: https://www.kaggle.com/datasets/andrewmvd/udemy-courses.

[15] Amazon Web Services. (2026). *AWS Pricing - EC2, EBS, S3, Lambda, EventBridge*. Доступно на: https://aws.amazon.com/pricing/.

[16] ClickHouse Cloud. (2026). *ClickHouse Cloud Pricing*. Доступно на: https://clickhouse.com/pricing.

Понављање експеримента

A.1 Предуслови

- на машини са које се покреће експеримент мора бити инсталиран `Docker` са додатком `Compose` и `Python` са верзијама из Табеле 1;
- доступан `S3 bucket`, са правом излиставања, читања, уписа и брисања објеката, и `ClickHouse` налог који може да креира базе и табеле и да мења њихов садржај;
- приступ `GitHub` репозиторијуму и јавним сликама `cdp-dagster:1.1.1` и `cdp-dbt:1.1.0` на `GitHub Container Registry`.

Верзије алата унутар слика и `dbt` пројекта дате су у Табели 1.

Алат	**Верзија**
`Docker`	28.1.1
`Docker Compose`	2.35.1
`Python` (host)	3.13.5
`Python` (контејнери)	3.11
`Dagster`	1.13.13
`dagster-dbt`	0.29.13
`dbt Core`	1.11.12
`dbt-clickhouse`	1.10.1
`dbt_utils`	1.4.1
`ClickHouse Cloud`	26.2.1.525

Табела 1: Верзије алата коришћене у експерименту

A.2 Конфигурација

Вредности се задају у датотеци `.env`, насталој копирањем `.env.example`. Поред приступних података за `S3` и `ClickHouse`, за понављање су значајне

следеће поставке:

```
COMPOSE_PROJECT_NAME=cdp
COMPOSE_PROFILES=local
CLICKHOUSE_SECURE=true
DATA_START_DATE=2024-01-01
DATA_END_DATE=2026-07-23
```

`CLICKHOUSE_SECURE=true` бира `dbt cloud` профил потребан за повезивање на `ClickHouse Cloud`, а `COMPOSE_PROJECT_NAME` одређује називе контејнера и мреже.

Две временске променљиве одређују само почетни, историјски скуп и приликом понављања морају остати фиксиране на наведене вредности: исти опсег и иста *seed* вредност дају исти скуп података. `DATA_END_DATE` је зато дан када је експеримент извршен, а не дан понављања. Подаци после тог датума не настају поновним генерисањем историје, већ дневним генератором, који сваки завршен дан наставља већ започете процесе. Понављање обухвата исту историју, закључно са 23.07.2026, и онолико дневних прираштаја колико је дана прошло од тог датума до дана понављања.

Кораци

Понављање обухвата генерисање истог скупа података, његово учитавање и изградњу аналитичких модела. Генерисање података и оркестрација тока обраде покрећу се локално помоћу `Docker Compose` фајла, док `Amazon S3` и `ClickHouse Cloud` нису део те конфигурације и морају бити доступни унапред.

1. **Припрема.** Копирати `.env.example` у `.env` и попунити приступне податке, формирати виртуелно окружење за `Python` и применити миграције базе скриптом `dwh/run_migrations.py`.
2. **Историјски скуп.** Покренути `generate_csv.py` са фиксираним почетним и крајњим датумом. Скрипта објављује изворне датотеке у `S3`, организоване по дану.
3. **Покретање сервиса.** Повући слике и покренути `dbt` контејнер ради инсталације зависности. Затим покренути `Dagster daemon` и веб-сервер, и у корисничком интерфејсу искључити сензор `dbt_trigger_sensor`. Сензор остаје искључен до краја уноса, како трансформације не би биле покренуте после сваке учитане партиције (Одељак 4.5).

4. **Унос историје.** Историјске партиције учитавају се `backfill` поступком, који обухвата 935 дневних партиција. Пре наставка треба сачекати да све партиције у `Dagster` интерфејсу добију статус `SUCCESS`.
5. **Дневни прираштаји.** Скриптом `build_pool_snapshot.py` формирати снимак стања, која чита историјске `src` табеле, па се покреће тек после завршеног уноса историје. Затим генерисати прираштаје за дане од 24.07.2026. до дана пре понављања и учитати их другим `backfill` поступком. Дани се генеришу хронолошки, јер сваки корак ажурира снимак стања на основу ког настаје следећи.
6. **Трансформације.** Покренути `dbt build`, који формира све моделе и извршава њихове тестове. Уз команду се задаје `as_of_date`, датум који користе мере зависне од текућег дана; тако те вредности остају исте и при поновној изградњи.

Након завршеног понављања сензор `dbt_trigger_sensor` се поново укључује, чиме платформа прелази на редован дневни рад.

Команде у наставку прате исте кораке:

Припрема окружења, генерисање историјског скупа и покретање сервиса (кораци 1--3):

```
# u .env uneti pristupne podatke
cp .env.example .env
python3.13 -m venv .venv && source .venv/bin/activate
python -m pip install -r infra/requirements.experiment.txt
python dwh/run_migrations.py

python data/generate_csv.py --start-date 2024-01-01 --end-date
    2026-07-23

docker compose pull
docker compose up -d dbt
docker compose exec dbt dbt deps
docker compose up -d dagster-daemon dagster-webserver
# u interfejsu iskljuciti dbt_trigger_sensor
```

Унос историје и дневних прираштаја (кораци 4--5), уз чекање на статус `SUCCESS` свих партиција после сваког `backfill` поступка:

```
YESTERDAY=$(date -d yesterday +%F)

docker compose exec dagster-webserver dagster job backfill \
  -j daily_ingestion_job --noprompt \
  --from 2024-01-01 --to 2026-07-23

python data/build_pool_snapshot.py

d=2026-07-24
until [[ ''$d'' > ''$YESTERDAY'' ]]; do
  python data/generate_daily_increment.py --date ''$d''
  d=$(date -d ''$d + 1 day'' +%F)
done

docker compose exec dagster-webserver dagster job backfill \
  -j daily_ingestion_job --noprompt \
  --from 2026-07-24 --to ''$YESTERDAY''
```

Трансформације (корак 6):

```
TODAY=$(date +%F)

docker compose exec dbt dbt build --full-refresh \
    --vars ''{as_of_date: '$TODAY'}''
```

A.3 Провера успешности

Понављање је успешно ако се `dbt` изградња заврши без грешака, чиме су задовољени сви тестови нивоа `error`.

Кориснички интерфејс система `Dagster`, у коме се могу пратити појединачна извршавања, доступан је на адреси `http://localhost:3000`.

Биографија аутора

Сара Живковић (*Пирот, 11.април 1999.*) завршила је Гимназију у Пироту 2018. године, а затим, исте године уписала Математички факултет Универзитета у Београду, на студијском програму Математика и рачунарство. Основне академске студије завршила је 2023. године, након чега је исте године уписала мастер академске студије на истом студијском програму Математичког факултета Универзитета у Београду.

Крајем 2022. кренула је да се усавршава у области `DevOps` инжињерства, а од 2023. године ради као `DevOps` инжењер у компанији `Fincore`, где се бави инфраструктуром у облаку и аутоматизацијом процеса. Поред тога, почела је да се интересује и за инжењерство података, што је био један од мотива за избор теме овог мастер рада.