Solution Design Document

SkillStorm Project: GitHub Archive

Version: 1.00

Updated: 12/31/2025

Authors: Christopher Lee, Javier Martinez, Stanley Liu

1. Overview or Introduction

- This document aims to provide an in depth breakdown of the GitHub Archive Data project. You can utilize this document to understand the high and low level design of the ETL pipeline, as well as the requirements and considerations taken for the project.

- To understand what this document provides, read the descriptions for each section below:

  - **Requirement Details:** Listing all the services the project uses and depends on.

  - **Assumptions & Prerequisites:** Listing assumptions group has, as well as what knowledge and infrastructure one should have before running this project.

  - **High Level Design:** *Broad* overview of the ETL pipeline and actions taken to achieve each layer in the Medallion Architecture. Listing aggregations that will be done with cleaned & transformed data.

  - **Low Level Design:** *Detailed* description of the ETL pipeline and actions taken to achieve each layer in the Medallion Architecture.

  - **Governance & Role Assignments:** Listing permissions given to each group member, as well as defining what each member worked on and separation of work.

  - **Out of Scope:** Stating why the actions that were taken in the pipeline were taken, and why others weren't based on the scope of the requirements.

  - **Risks & Mitigation:** Difficulties that may arise are laid out.

  - **Appendices:** Additional resources that are too detailed and of less relevance to the project.

2. Requirement Details

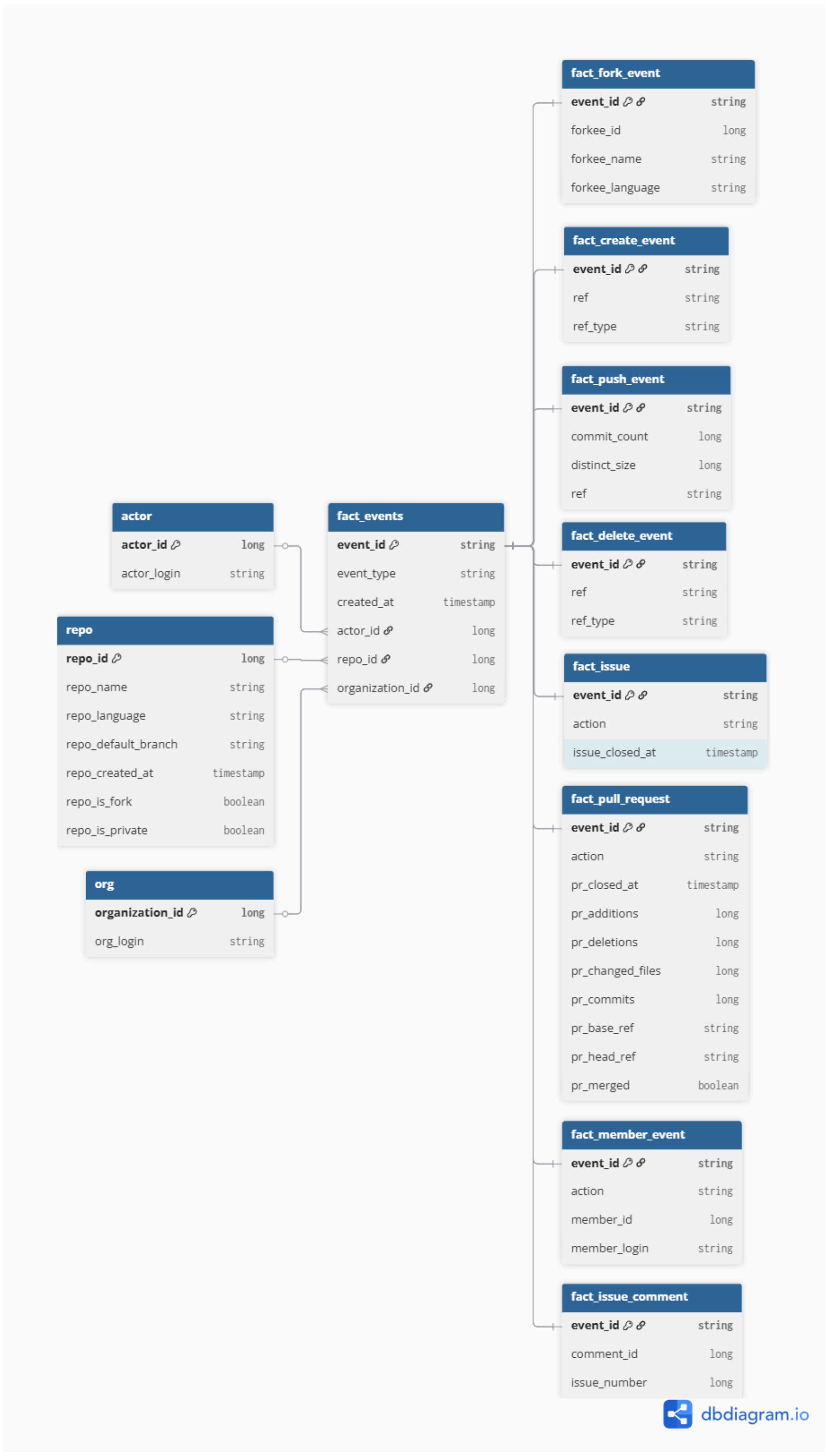
- Azure Data Lake Storage Gen2 (ADLS v2) - storing all our data (bronze, silver, gold)
- GitHub Archive Data (2015) - raw data which will be stored in ADLS and processed in Databricks
- Azure Databricks - transform and process data (ETL pipeline)
- Analytical Tools: Databricks Visualization/Power BI - showcases the usefulness of our gold layer data

3. Assumptions and Prerequisites

- Assumptions:
  - GHArchive data format is consistent throughout when we use it.
  - Timestamps are in UTC.
  - All columns or fields with value, including columns which we will not use to answer aggregation questions, will be kept.
- Prerequisites:
  - All users have access to Azure Databricks workspace.
  - Sufficient cluster compute power.
  - GHArchive 2015 data downloaded into data lake, accessed through there.
  - Team familiarity with PySpark, SparkSQL, Medallion Architecture, and other needed concepts.
  - Team familiarity of navigating the Azure portal.

4. High-Level Design

Normalized ERD (Silver Layer)



actor

actor_id	long
actor_login	string

repo

repo_id	long
repo_name	string
repo_language	string
repo_default_branch	string
repo_created_at	timestamp
repo_is_fork	boolean
repo_is_private	boolean

org

organization_id	long
org_login	string

fact_events

event_id	string
event_type	string
created_at	timestamp
actor_id	long
repo_id	long
organization_id	long

fact_fork_event

event_id	string
forkee_id	long
forkee_name	string
forkee_language	string

fact_create_event

event_id	string
ref	string
ref_type	string

fact_push_event

event_id	string
commit_count	long
distinct_size	long
ref	string

fact_delete_event

event_id	string
ref	string
ref_type	string

fact_issue

event_id	string
action	string
issue_closed_at	timestamp

fact_pull_request

event_id	string
action	string
pr_closed_at	timestamp
pr_additions	long
pr_deletions	long
pr_changed_files	long
pr_commits	long
pr_base_ref	string
pr_head_ref	string
pr_merged	boolean

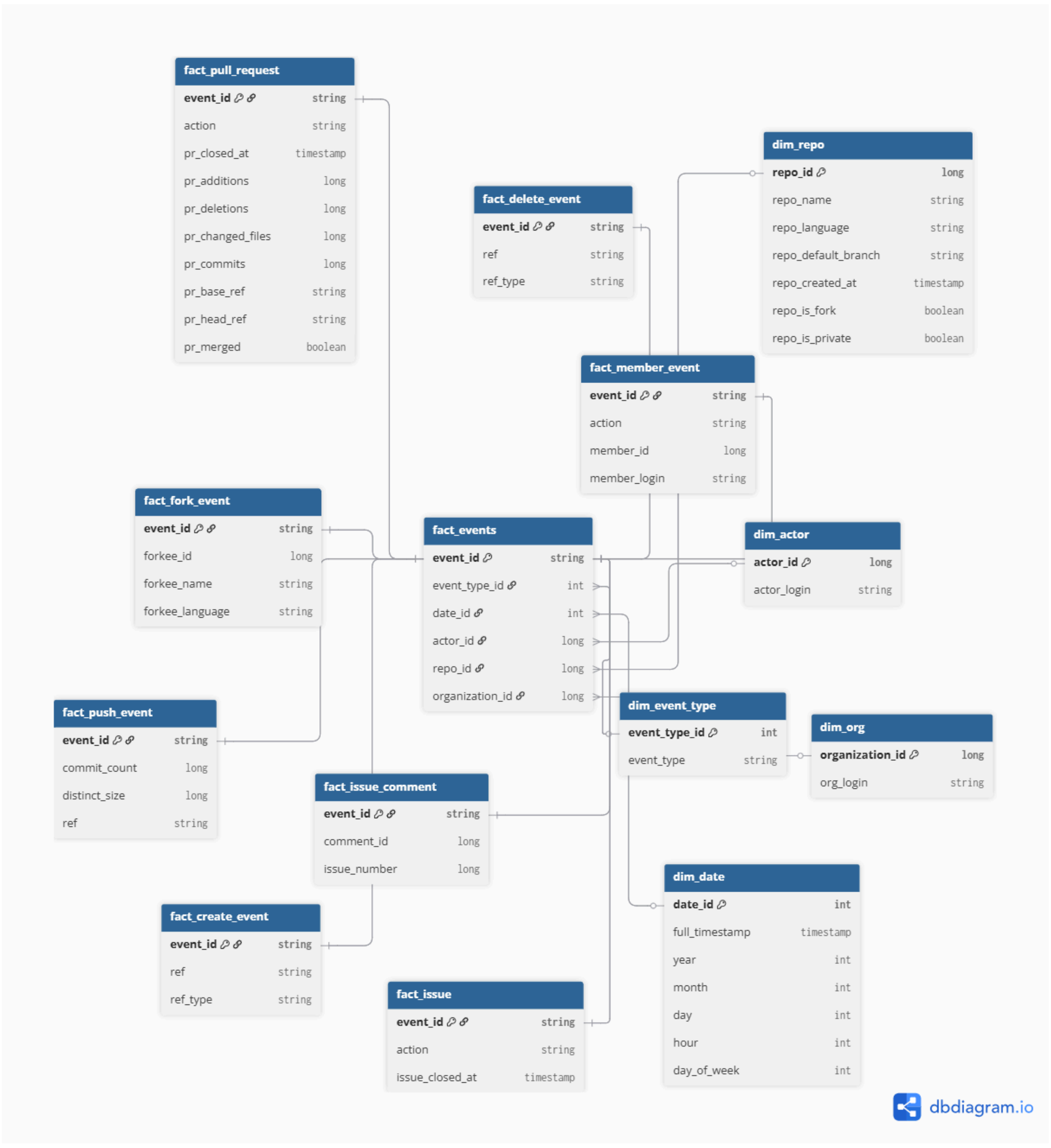
fact_member_event

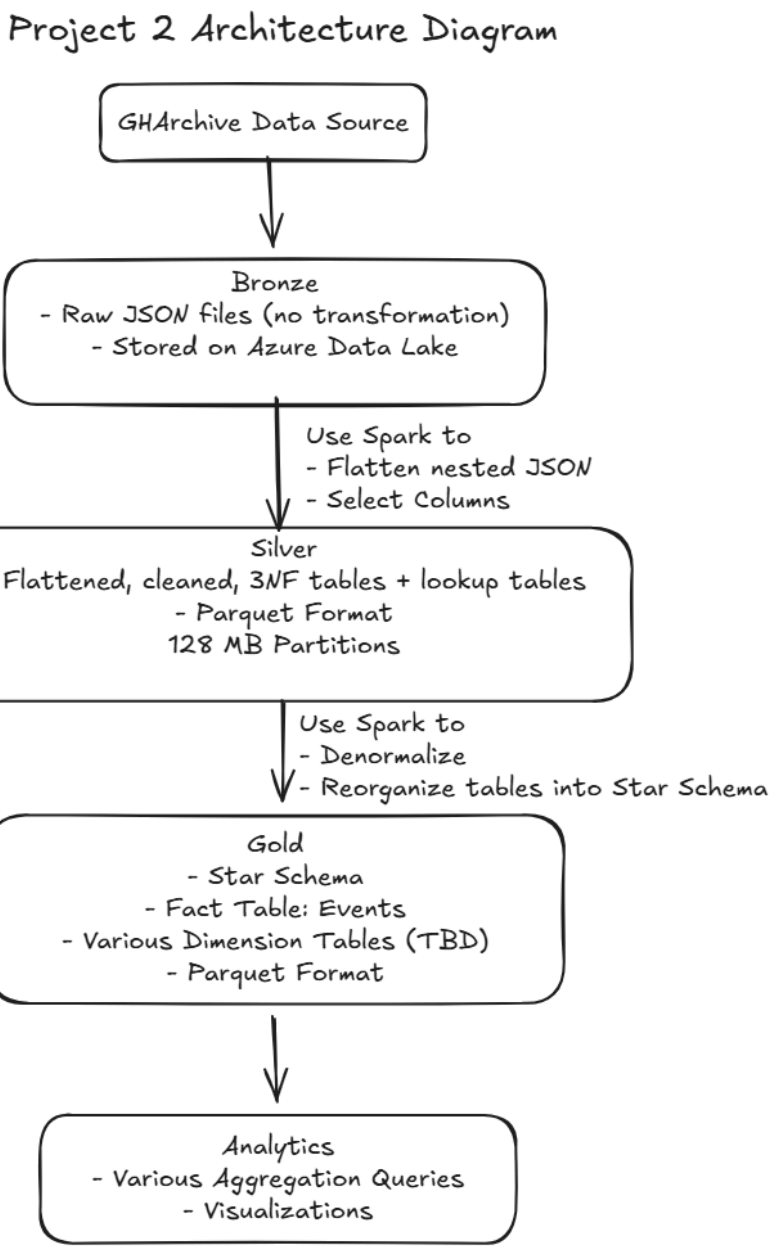
event_id	string
action	string
member_id	long
member_login	string

fact_issue_comment

event_id	string
comment_id	long
issue_number	long

Denormalized ERD (Gold Layer)




Project 2 Architecture Diagram
GHArchive Data Source
Bronze
- Raw JSON files (no transformation)
- Stored on Azure Data Lake
Use Spark to
- Flatten nested JSON
- Select Columns
Silver
- Flattened, cleaned, 3NF tables + lookup tables
- Parquet Format
128 MB Partitions
Use Spark to
- Denormalize
- Reorganize tables into Star Schema
Gold
- Star Schema
- Fact Table: Events
- Various Dimension Tables (TBD)
- Parquet Format
Analytics
- Various Aggregation Queries
- Visualizations

Bronze layer, with raw JSON.gz files.

- Data shared w/ other teams in Azure Data Lake.
- Consists of compressed JSON files.

Silver layer, with cleaned Parquet files

- Nested JSON files turned into flat tables.
- Extract useful and valuable fields.
- Discard garbage fields.
- Convert timestamps, if needed.
- Partition to 128 MB files.

Gold layer, Star Schema + Aggregations

- Fact table, star schema, dimension tables
- Aggregations to answer these questions:

        - Events by type per hour - What types of events happen most often? What is the proportion of events per hour? Which event types peak at which hours?
        - pushevent by branch type - How many pushes or what proportion go to main/master vs other? What percent of commits are direct to main?
        - Events by type and commit count - Which event types involve commits, how many commits do people push at once, etc.?
        - User activity by week or month - the most active users? How does user activity change over weeks and months? Which repos have the most activity?

5. Low-Level Design

- Source code files:
    - gharchive-raw for initial data ingestion.
    - **01_bronze_to_silver notebook for flattening JSON to Parquet**
        - To begin, we set a general schema to optimize the reads of our files. All github events have the same general columns, so we make a schema with that in mind:

```
schema = StructType([

  StructField('actor',StructType(), True),

  StructField('created_at', StringType(), True),

  StructField('id', StringType(), True),

  StructField('org', StructType(), True),

  StructField('payload',StructType(), True),

  StructField('public', BooleanType(), True),

  StructField('repo',StructType(),True),

  StructField('type',StringType(), True)

])
```

        - It’s guaranteed that all entries from each file will have this data. Since nested data does not have bounds as to how far it can be nested, it’s impossible to make a schema for all the rows.
        - After specifying the schema, we read in the files. For each batch of files we read in, we do the following:
        - Keep only the columns we deem hold any analytical value. Something to note is, if there is very little actual data in this column, it can be dropped because it’ll mostly be null.
        - Following our normalized entity relationship diagram, we create the tables.

- While creating lookup tables for the different columns, we ensure that ids don’t appear twice. To give an example, the actor_id column has repeated ids, because the value related to that id changes because people change their login information so over time you’ll get a difference. Because that was the case, we kept the latest login based on timestamp.
- After that, we write our files in parquet format. The folders in our silver layer will be structured by table type, and in each folder, the file names will have information pertaining to the month.

-

02_silver_to_gold for building star schema

- To begin, we start with reading in our silver layer files. We specify a schema for every single table type we have, this is for fast reading of our files.
- After reading in our files and storing them in their appropriate table, we create our aggregation tables. Each aggregation table answers one question specified in the project:

1. **Aggregation 1:** Data aggregated by type of github event per hour. We use the date dimension table we created and the event types table. To make this table, we group by event type, year, and hour.
2. **Aggregation 2:** Push event data aggregated by ref type. We use the push event table we created in our bronze to silver notebook and our repo dimension table as well. For any column that contains master or main, that will be our main branch. Any other column would be classified as not our main branch. That way, we can see the amount of commits/pushes by the type of branch.
3. **Aggregation 3:** Breakdown of events by type and number of commits per event. For this aggregation, we group by commit count, and count all the amounts of push count. This can be done by utilizing the push event table we made in our bronze to silver layer notebook.
4. **Aggregation 4:** User activity should be aggregated so a filterable chart can be populated with breakdowns of user activity by week or month. For this we use the date dimension table, the actor id and lookup table to group them together to make a week by week analysis of the amount of activity by user. We can also do a similar strategy to get the commit count for each week.
5. **Aggregation 5:** Breakdown of activity by project. For this part, we use the events, repo, and organization table to perform an aggregation to see which repositories are for individual developers and which ones are for actual companies.

- After this has been done, we can write the files to our gold layer, and use these aggregated tables for visualizations.

6. Governance and Role Assignments

Note: We all have the same access/permissions. Do we still create a role map?

- All team members were given contributor access within Azure Databricks
- All team members were given read/write access to ADLS
- Work in each layer in Medallion Architecture is split between the team members.

7. Risks and Mitigation

Risks can arise in the form of:

- Data redundancy - as our data is stored locally using LRS in an Azure data center, any outage or issues within that data center will be problematic. A solution would be to switch to GRS where if an availability zone is down, we have a backup zone to rely on.