

Vibe Coding Framework to accelerate end to end Data Products on Databricks

Last Updated: December 12, 2025

Created by: Prashanth Subrahmanyam

Request edit/comment access if required.

Background

This document is a comprehensive guide on an opinionated way to accelerate Data Product creation on Databricks using Cursor IDE and Claude models. This example walks through a sample use case using Wanderbricks dataset. **It is for internal use only. Please refer to public documentation for the most up-to-date information, while using this as a point-in-time reference only.**

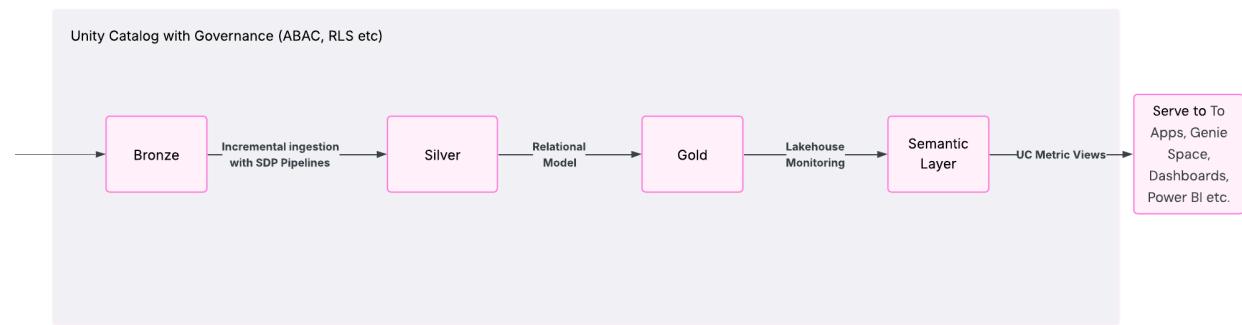


Figure 1: High-level reference architecture: *Bronze ingestion → Silver incremental processing (SDP) → Gold Layer with relational model → Lakehouse Monitoring → Semantic Layer (UC Metric Views) → Consumption via Apps/Genie/Dashboards/Power BI.*

Pre-Requisites

- Install Cursor
- [Install Databricks CLI](#)
- Install following extensions from marketplace:
 - [Markdown Preview Mermaid Support](#)
 - [IDE Support for Databricks](#)

Background

[Pre-Requisites](#)

[Step 1: Get metadata and data dictionary from the customer](#)

[Metadata Query:](#)

[iData Dictionary:](#)

[Step 2: Set-up Cursor Project with Rules and Context folders:](#)

[Start New Cursor Project -](#)

[Copy Cursor Rules and Context Folders from the framework:](#)

[Add Docs as MCP Server for Azure Databricks Docs](#)

[Step 3: Create a Gold Layer Design based on the Bronze Metadata received from the customer.](#)

[Step 4: Create a Bronze Layer](#)

[Step 5: Create Silver Layer with Centralized Data Quality Rules](#)

[Step 6: Create Gold Layer Merge Jobs for this dataset](#)

[Step 7: Create plan for use cases and supporting Artifacts for this dataset](#)

[Step 8: Create artifacts to support Genie use cases](#)

[Table Valued Functions](#)

[Metric Views](#)

[Genie Spaces](#)

[Step 9: \(Optional\) Create artifacts to support new use cases](#)

[ML Models](#)

[Lakehouse Monitoring](#)

[AI / BI Dashboards](#)

Step 1: Get metadata and data dictionary from the customer

Metadata Query:

Request the customer to run the following query on any schemas that they want to use to create the data product. In this example, we will use the Wanderbricks sample:

```
None

SELECT * FROM
samples.information_schema.columns
WHERE table_schema = 'wanderbricks'
```

The screenshot shows a database query interface with the following details:

- Query: `SELECT * FROM samples.information_schema.columns WHERE table_schema = 'wanderbricks'`
- Results (16 rows):

	table_catalog	table_schema	table_name	column_name	ordinal_position	column_default	is_nullable	full_data_type	data_type	character_maximum_length	character_octet_length
1	samples	wanderbricks	amenities	amenity_id	0	null	NO	bigint	LONG	null	null
2	samples	wanderbricks	amenities	name	1	null	YES	string	STRING	0	0
3	samples	wanderbricks	amenities	category	2	null	YES	string	STRING	0	0
4	samples	wanderbricks	amenities	icon	3	null	YES	string	STRING	0	0
5	samples	wanderbricks	booking_updates	booking_id	0	null	YES	bigint	LONG	null	null
6	samples	wanderbricks	booking_updates	booking_update_id	1	null	NO	bigint	LONG	null	null
7	samples	wanderbricks	booking_updates	check_in	2	null	YES	date	DATE	null	null
8	samples	wanderbricks	booking_updates	check_out	3	null	YES	date	DATE	null	null
9	samples	wanderbricks	booking_updates	created_at	4	null	YES	timestamp	TIMESTAMP	null	null
10	samples	wanderbricks	booking_updates	guests_count	5	null	YES	int	INT	null	null
11	samples	wanderbricks	booking_updates	property_id	6	null	YES	bigint	LONG	null	null
12	samples	wanderbricks	booking_updates	status	7	null	YES	string	STRING	0	0
13	samples	wanderbricks	booking_updates	total_amount	8	null	YES	float	FLOAT	null	null
14	samples	wanderbricks	booking_updates	updated_at	9	null	YES	timestamp	TIMESTAMP	null	null
15	samples	wanderbricks	booking_updates	user_id	10	null	YES	bigint	LONG	null	null
16	samples	wanderbricks	bookings	booking_id	0	null	NO	bigint	LONG	null	null

Figure 2: Pulling source metadata from `samples.information_schema.columns` to bootstrap the data dictionary and drive LLM-assisted modeling and code generation.

Data Dictionary:

- Request the customer for a data dictionary if available, to augment the metadata. If the metadata is complete in itself, then we can skip this. If not, this can provide additional context to the LLM to be comprehensive for the use case.
- A data dictionary will consist of a listing of all tables and columns coming from a source application, along with their descriptions, data types, nested columns etc.
- Ensure to re-concile all of this data, and preferably convert everything to CSV for easy ingestion to the LLM.

Step 2: Set-up Cursor Project with Rules and Context folders:

Start New Cursor Project -

Click on Open Project and Choose An Empty New Folder

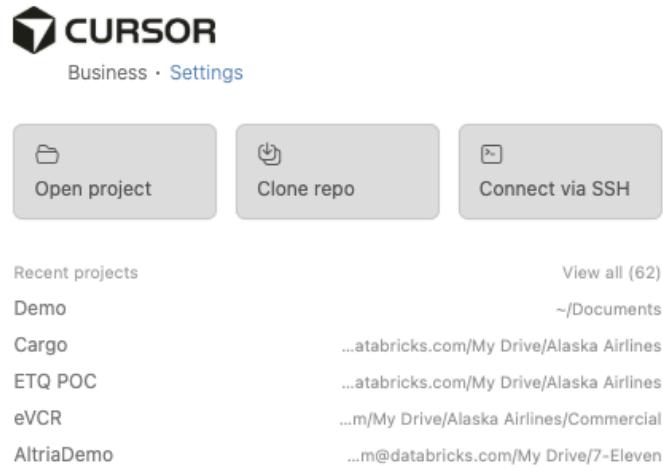


Figure 3: Starting a new Cursor project to bootstrap the Vibe Coding framework, where rules, prompts, and metadata-driven context will guide end-to-end data product generation.

Download data_product_accelerator Folders from the framework:

Ensure that the `skills` and `context` are copied into your project. Also, ensure to copy metadata and data dictionary from step 1 into the context folder.

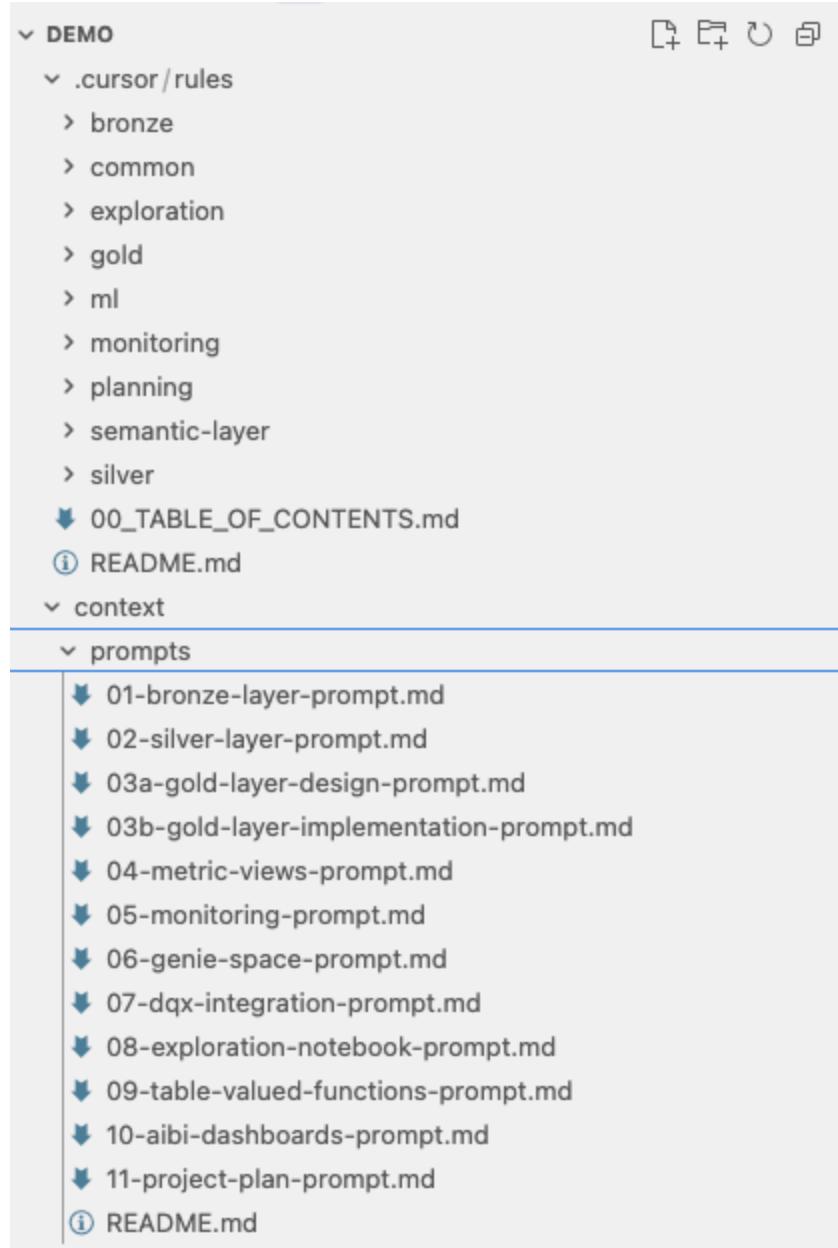


Figure 4: Cursor project structure showing standardized folders for bronze, silver, gold, monitoring, and semantic layers, along with prompt-driven context files used to ground LLM outputs.

Add Docs as MCP Server for Azure Databricks Docs

Click on Settings → Tools & MCP → Add a Custom MCP Server and add the below code.

JSON

```
{
```

```

"mcpServers": {
    "MS Learn Docs": {
        "url": "https://learn.microsoft.com/api/mcp",
        "headers": {}
    }
}

```

This will provide the framework access to documentation that it can search for when needed. You can also explicitly prompt the LLM to look up document using MCP tool call to MS Learn Docs. You can add additional MCP servers as required for your application such as Context7 etc.

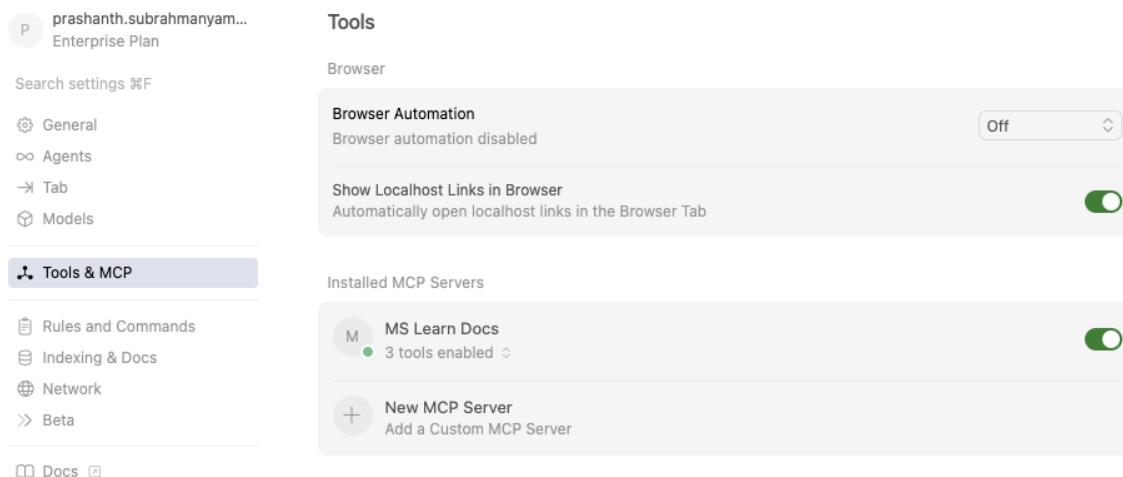


Figure 5: Configuring Microsoft Learn as an MCP server inside Cursor, enabling the LLM to dynamically retrieve and ground responses in official Azure Databricks documentation.

Step 3: Create a Gold Layer Design based on the Bronze Metadata received from the customer.

Use the following prompt to create the gold layer design. This should create the necessary YAML files that define each gold table with its metadata as well as ERD diagrams that can be rendered in mermaid.

Start a new Agent thread and use the following prompt:

None

```
I have a customer schema at  
@data_product_accelerator/context/Wanderbricks_Schema.csv.  
Please design the Gold layer using  
@data_product_accelerator/skills/gold/00-gold-layer-design/SKILL.  
md
```

What it produces :

- Schema intake report (table inventory, dimension/fact classification, FK relationships)
- Mermaid ERD diagrams
- YAML schema files in [gold_layer_design/yaml/](#)
- Business onboarding guide
- Column lineage and source table mapping

★Checkpoint★:

This is an important validation checkpoint in this process. The output of the LLM must be validated for accuracy, validity, applicability to the dataset etc before proceeding further. We should proceed to the next steps only after thorough human review.

```

! dim_host.yaml x DESIGN_SUMMARY.md ! fact_payment_daily.yaml ! fact_property_engagement
gold_layer_design > yaml > host > ! dim_host.yaml > [ ] columns > {} 9 > {} lineage > bronze_table

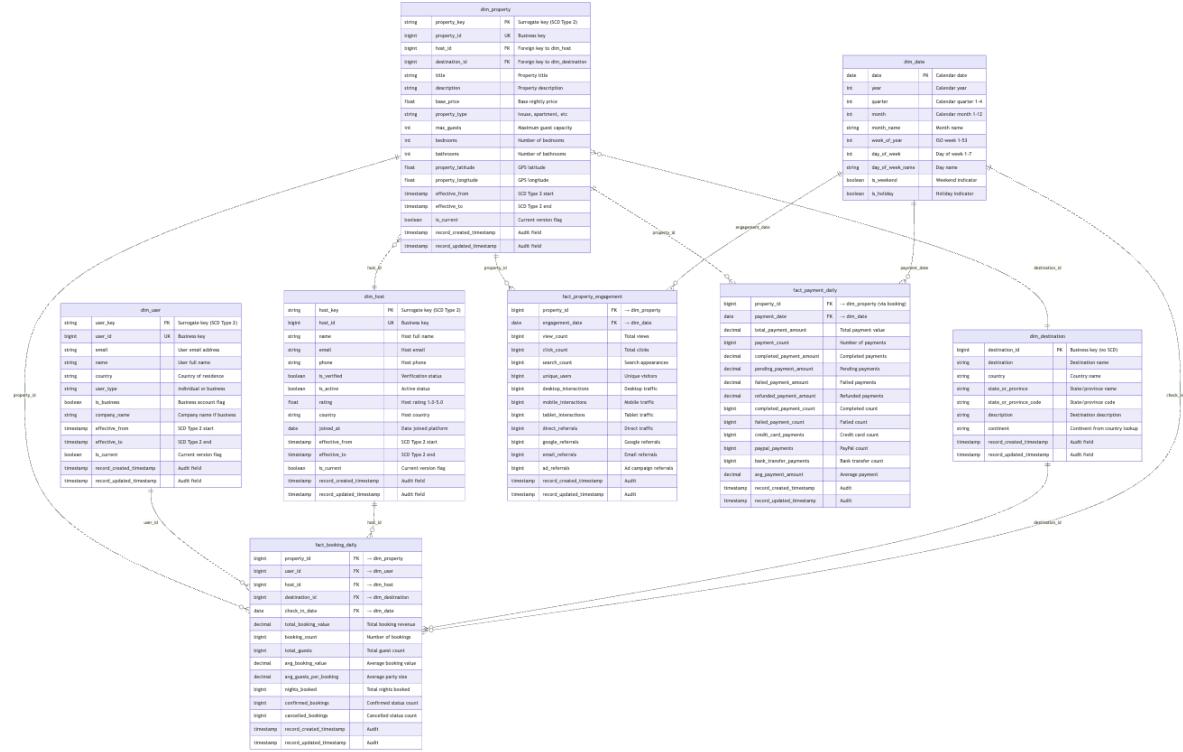
1 # Table metadata
2 table_name: dim_host
3 domain: host
4 bronze_source: wanderbricks.hosts
5
6 # Table description (dual-purpose: business + technical)
7 description: >
8   Gold layer conformed host dimension with SCD Type 2 history tracking.
9   Business: Maintains complete host account history including verification status changes,
10  rating updates, and active status modifications. Each host version tracked separately for
11  historical host performance analysis and quality assessment. Used for host relationship
12  management, portfolio analysis, host segmentation by rating tiers, and verification compliance
13  tracking. Critical for understanding host lifecycle from onboarding through maturity.
14  Technical: Slowly Changing Dimension Type 2 implementation with effective dating. Surrogate keys
15  enable proper fact table joins to historical host states. Updates via Delta MERGE from Silver
16  streaming tables. Rating and verification status changes create new versions for trend analysis.
17
18 # Grain definition
19 grain: "One row per host_id per version (SCD Type 2)"
20
21 # SCD strategy
22 scd_type: 2
23
24 # Primary key definition
25 primary_key:
26   columns: ['host_key']
27   composite: false
28
29 # Business key (unique constraint for current records)
30 business_key:
31   columns: ['host_id']
32
33 # Foreign key definitions (applied separately)
34 foreign_keys: []
35
36 # Silver lineage
37 silver_lineage:
38   source_table: silver_hosts
39   join_required: false
40   denormalization: []
41
42 # Column definitions with complete lineage
43 columns:
44   # Surrogate Key
45   - name: host_key
46     type: STRING
47     nullable: false
48     description: >
49       Surrogate key uniquely identifying each version of a host record.
50       Business: Used for joining fact tables to dimension. Enables tracking host status
51       changes over time for historical performance analysis by host characteristics and
52       rating evolution.
53       Technical: MD5 hash generated from host_id and joined_at timestamp to ensure
54       uniqueness across SCD Type 2 versions. Immutable once created.
55   lineage:
56     bronze_table: wanderbricks.hosts
57     bronze_column: host_id, joined_at
58     silver_table: silver_hosts
59     silver_column: host_id, joined_at
60     transformation: "HASH_MDS"
61     transformation_logic: "md5(concat_ws('||', col('host_id'), col('joined_at')))"
62     notes: "Ensures uniqueness across SCD Type 2 versions"
63

```

Figure 6: LLM-generated Gold layer design expressed as YAML, capturing table schemas, metadata, constraints, and documentation in a declarative, version-controlled format.

Wanderbricks Gold Layer - Entity Relationship Diagram

Complete Dimensional Model



Model Summary

Dimensional Model Type: Star Schema with Conformed Dimensions

Conformed Dimensions:

- **dim_property** - Shared across all facts
- **dim_date** - Shared across all facts
- **dim_host** - Links properties to hosts
- **dim_destination** - Geographic dimension

Figure 7: Automatically generated Mermaid ERD illustrating the complete dimensional model for the Wanderbricks Gold layer, including facts, dimensions, and their relationships.

Step 4: Create a Bronze Layer

In general, this framework is designed for taking data from Bronze and processing it through silver and gold.

3 approaches to Bronze Data has been laid out in the prompt:

1. **Approach A:** Generating Fake Data using the provided schema for Bronze Layer
 - a. This approach will create the necessary table DDLs and populate them with Fake data using Faker library.
 - b. If you were building this for a customer, and wanted to generate fake data to test out the data product before sharing the bundle with the customer, then in that case, you can use the following steps.
2. **Approach B:** Using existing Bronze Data:
 - a. If the customer already has a bronze layer, they could use it directly instead of creating a new bronze layer.
 - b. If the customer is building the data product themselves using this framework, they can directly connect to bronze layer, and they can skip this step.
3. **Approach C:** Copy from External Source:
 - a. If data is available as a sample set etc, we can copy it and start from that as a bronze layer.
 - b. In this example, we will use the wanderbricks sample as the bronze layer that is already landed and thus, use Approach C.

Start a new Agent thread and use the following prompt:

None

Set up the Bronze layer using
@data_product_accelerator/skills/bronze/00-bronze-layer-setup/SKILL.md with Approach C – copy data from the existing source tables in the wanderbricks schema.

Use default catalog as: your_catalog

This should create an output like below with a Databricks Asset Bundle structure created with necessary src/ files which have code as well as resources folder which have the job configurations. The framework ensures that tables created have liquid clustering enabled, change data feed enabled by default, as best practices.

What it produces:

- Bronze table DDLs with TBLPROPERTIES, CDF, and governance metadata
- Data generation or copy notebooks
- Asset Bundle job configuration

```

Bronze layer creation with Approach C
+ ○ ...
  QUICKSTART.md X
Can you please create a bronze layer using Approach C of copying sample data from samples.wanderbricks using [bronze_layer_prompt.nd] while grounding yourself in [README.md] file for asset bundles and associated artifacts?
5 of 5 To-dos Complete

Wanderbricks Bronze Layer - COMPLETE!
I successfully created a complete bronze layer implementation using Approach C (copying from samples.wanderbricks). Here's what was delivered:
  What Was Created
  1. Core Project Files
    • wanderbricks.yaml - Main Asset Bundle configuration with devprod targets
    • README.md - Complete project documentation with architecture details
    • QUICKSTART.md - 5-minute quick start guide
    • .gitignore - Git ignore pattern for clean repos
  2. Bronze Layer Source Code (src/wanderbricks_bronze/)
    • setup_tables.py - Creates 9 Bronze tables with DDL definitions (~600 lines)
    • copy_from_wanderbricks.py - Copies data from samples catalog (~250 lines)
    • __init__.py - Package initialization
  3. Asset Bundle Resources (resources/)
    • schemas.yaml - Unity Catalog schema definitions (Bronze/Silver/Gold)
    • bronze_setup.job.yaml - Serverless job to create tables
    • bronze_copy.job.yaml - Serverless job to copy data
  4. Documentation (docs/deployment/)
    • wanderbricks-bronze-deployment.md - Step-by-step deployment guide
    • wanderbricks-bronze-deployment-summary.md - Technical details
    • BRONZE_LAYER_COMPLETE.md - Success summary
  Bronze Layer Tables (9 Total)
  Dimension Tables (4)
  Table Records Domain PII
  bronze_destinations 52 destination No
  bronze_hosts 127 host Yes
  bronze_properties 324 property No
  bronze_users 1,043 user Yes
  Fact Tables (5)
  Table Records Domain
  bronze_bookings 5,247 booking
  
```

Figure 8: Validating and executing Bronze layer jobs using Databricks Asset Bundles, ensuring tables, configurations, and best practices (CDF, liquid clustering) are correctly deployed.

★Checkpoint★:

Due to the non-deterministic nature of LLMs, it may take a few tries of troubleshooting with the LLM to ensure the jobs run correctly and the tables are set-up correctly. It's of utmost importance that you verify the output before proceeding further.

We highly recommend you ask the LLM to run the bronze jobs to validate that the bronze layer creation is working as per expectations before moving ahead. We can do this by asking the LLM to run the jobs with the following prompt.

This prompt is common for all checkpoints, and essentially, it makes the LLM deploy the asset bundle and then monitor job and troubleshoot if there is an error. (**Ensure that Databricks CLI is installed for this**).

None

```
Using @data_product_accelerator/skills/common/databricks-autonomous-operations,  
authenticate, validate, deploy, run, troubleshoot until success for the Bronze  
layer jobs
```

Step 5: Create Silver Layer with Centralized Data Quality Rules

Next, we move onto creating the Silver Layer for this project. In this framework, we have chosen Spark Declarative Pipelines (SDP) for the silver layer as it's capable of incremental ingestion from bronze along with built-in quality rules. Other patterns can also be used based on project requirements, but it may require customization of other prompts for this purpose.

In this example, the Silver Layer consists of creating Spark Declarative Pipelines (SDP) with Centralized Data Quality Rules that are stored in a delta table. SDP will run on Serverless and will do incremental reads from the bronze layer from the change data feed of the Bronze Layer tables.

Start a new Agent thread and use the following prompt:

None

Set up the Silver layer using
`@data_product_accelerator/skills/silver/00-silver-layer-setup/SKILL.md`

What it produces:

- SDP pipeline notebooks with streaming ingestion
- Data quality expectations (stored in Unity Catalog Delta table)
- Quarantine patterns for failed records
- Asset Bundle pipeline configuration

Project Silver Layer creation

Note: ADLS -12 - Delta 4.45.comet

We just created the Silver Layer for this project using #02-silver-layer-prompt.mli while grounding yourself in the [silver](#) and common rules.

Silver Layer Implementation Complete!

I've successfully created a production-grade Silver Layer for Wanderbricks with Delta table-based data quality rules. Here's what was created:

Files Created (13 files)

Core Python Files (9 files)

File	Type	Lines	Purpose
src/wanderbricks_silver/_setup_dlt_rule.py	Notebook	464	Creates & populates dlt_delta table (51 rules)
src/wanderbricks_silver/_load_review.py	Pure Python	137	Loads rules at DLT runtime (NO notebook header)
src/wanderbricks_silver/_silver_dimensions.py	DLT Notebook	298	4 dimension tables with validation
src/wanderbricks_silver/_silver_bookings.py	DLT Notebook	176	Bookings fact + quarantine
src/wanderbricks_silver/_silver_reviews.py	DLT Notebook	302	Payments + reviews fact + quarantines
src/wanderbricks_silver/_silver_analytics.py	DLT Notebook	228	Clickstream + page_views tables
src/wanderbricks_silver/_silver_data_quality.py	DLT Notebook	335	5 DQ monitoring views
src/wanderbricks_silver/_silver_init__.py	Package	6	Package initialization
src/wanderbricks_silver/_silver_compliance.md	Docs	521	Comprehensive Silver layer documentation

Configuration Files (2 files)

```
resourcemanager_id: "silver_dlt_pipeline.yml"
# HeaderBlocks Silver Layer DLT Pipeline [Serverless]
# References: https://github.com/databricks/hadoop-examples/blob/main/knowledge_base/pipeline_with_schema/resources/pipeline.yml
# Delta Live Tables streaming pipeline with data quality expectations
# Uses serverless compute for automatic scaling and cost optimization
#
# resources:
#   pipelines:
#     silver_dlt_pipeline:
#       name: ${!bundle_name}.target1
#       # Pipeline root folder must be within this root folder
#       # All artifacts must be within this root folder
#       # Reference: https://docs.databricks.com/awaken/dp/multi-file-editor#root-folder
#       root_path: ./src/wanderbricks_silver
#
#       # DLT Direct Publishing Mode (Modern Pattern)
#       # schema: field (not "target" - deprecated)
#       # catalog: ${!bundle_name}.catalog
#       # schema: $var.silver.schema
#
#       # RDB Libraries (Python notebooks)
#       libraries:
#         - notebooks:
#             - paths: ./src/wanderbricks_silver/silver_dimensions.py
#             - notebooks:
#                 - paths: ./src/wanderbricks_silver/silver_bookings.py
#                 - notebooks:
#                     - paths: ./src/wanderbricks_silver/silver_payments_review.py
#                     - notebooks:
#                         - paths: ./src/wanderbricks_silver/silver_analytics.py
#                         - notebooks:
#                             - paths: ./src/wanderbricks_silver/data_quality_monitoring.py
#
#       # Pipeline Configuration (passed to notebooks)
#       # Use fully qualified table names in notebooks: (catalog).(schema).(table)
#       config:
#         catalog: ${!bundle_name}.catalog
#         bronze_schema: ${!bundle_name}.bronze_schema
#         silver_schema: ${!bundle_name}.silver_schema
#         pipelines.enableTracing: "true"
#
#       # Serverless Compute
#       serverless: true
#
#       # Photon Engine
#       photon: true
#
#       # Channel (CURRENT = latest features)
#       channel: CURRENT
#
#       # Continuous vs Triggered
#       continuous: false
# 
```

9 of 13 Files | Undo File | Keep File

DEMO

- cursor
- context
- docs
- gold_layer_design
- green_rabbit
- bronze_setup.job.yml
- schemas.yml
- silver_dlt_pipeline.yml
- silver_dq_setup.job.yml
- silver_analytics.py
- silver_bookings.py
- silver_dimensions.py
- silver_reviews.py
- silver_analytics_reviews.py
- gllignore
- databricks.yml
- QUICKSTART.md
- README.md

Figure 9: Silver layer creation using Spark Declarative Pipelines (serverless) with centralized DQ rules—shown as an LLM-driven workflow that generates pipeline code + config artifacts.

Catalog Explorer - prashanth_subrahmanyam_catalog / dev_prashanth_subrahmanyam_wande... >									
dq_rules									
Overview	Sample Data	Details	Permissions	Policies	History	Lineage	Insights	Quality	
Ask your question about the sample data...									
❖ How many rules are critical severity?	❖ What are the most recent updates to dq_rules?	❖ Which tables have the most dq_rules assigned?							
Preview > Reset									
Sample									
#	table_name	rule_name	constraint_sql	severity	description	created_timestamp	updated_timestamp		
1	silver_employees	valid_employment_status	end_service_date IS NULL OR end_service_date > joined_at	warning	End date should be after start date if present	2025-12-11T04:02:14.636+00:00	2025-12-11T04:02:14.636+00:00		
2	silver_payments	valid_payment_id	payment_id IS NOT NULL	critical	Payment ID must be present (primary key)	2025-12-11T04:02:32.665+00:00	2025-12-11T04:02:32.665+00:00		
3	silver_countries	valid_country_code	country_code IS NOT NULL AND LENGTH(country_code) = 2	critical	Country code must be 2-letter ISO code	2025-12-11T04:01:58.869+00:00	2025-12-11T04:01:58.869+00:00		
4	silver_page_views	valid_timestamp	timestamp IS NOT NULL AND timestamp >= MAKE_DATE(2020, 1, 1)	critical	Timestamp must be present and reasonable	2025-12-11T04:02:34.066+00:00	2025-12-11T04:02:34.066+00:00		
5	silver_customer_support_logs	has_messages	messages IS NOT NULL AND SIZE(messages) > 0	warning	Support ticket should have at least one message	2025-12-11T04:03:03.033+00:00	2025-12-11T04:03:03.033+00:00		
6	silver_reviews	has_comment_or_rating	comment IS NOT NULL OR rating IS NOT NULL	warning	Review should have either comment or rating	2025-12-11T04:02:42.535+00:00	2025-12-11T04:02:42.535+00:00		
7	silver_bookings	reasonable Stay	DATEDIFF(check_out, check_in) BETWEEN 1 AND 365	warning	Stay duration should be reasonable (1-365 days)	2025-12-11T04:02:28.621+00:00	2025-12-11T04:02:28.621+00:00		
8	silver_payments	valid_payment_date	payment_date IS NOT NULL AND payment_date >= MAKE_DATE(2020, 1, 1)	critical	Payment date must be present and reasonable	2025-12-11T04:02:38.191+00:00	2025-12-11T04:02:38.191+00:00		
9	silver_customer_support_logs	valid_ticket_id	ticket_id IS NOT NULL AND LENGTH(ticket_id) > 0	critical	Ticket ID must be present (primary key)	2025-12-11T04:02:37.964+00:00	2025-12-11T04:02:37.964+00:00		
10	silver_properties	reasonable_guests	max_guests BETWEEN 1 AND 20	warning	Guest count should be reasonable (1-20)	2025-12-11T04:02:36.757+00:00	2025-12-11T04:02:36.757+00:00		
11	silver_booking_updates	update_after_creation	created_at IS NULL OR updated_at >= created_at	warning	Update timestamp should be after creation timestamp	2025-12-11T04:02:23.152+00:00	2025-12-11T04:02:23.152+00:00		
12	silver_bookings	reasonable_guests	guests_count BETWEEN 1 AND 20	warning	Guest count should be reasonable (1-20)	2025-12-11T04:02:23.130+00:00	2025-12-11T04:02:23.130+00:00		
13	silver_clickstream	valid_timestamp	timestamp IS NOT NULL AND timestamp >= MAKE_DATE(2020, 1, 1)	critical	Timestamp must be present and reasonable	2025-12-11T04:02:47.735+00:00	2025-12-11T04:02:47.735+00:00		
14	silver_payments	valid_booking	booking_id IS NOT NULL	critical	Booking ID must be present (FK validation)	2025-12-11T04:02:33.570+00:00	2025-12-11T04:02:33.570+00:00		
15	silver_hosts	valid_email	email IS NOT NULL AND INSTR(email, '@hr(b4)i') > 0	critical	Host email must be present and valid	2025-12-11T04:01:51.953+00:00	2025-12-11T04:01:51.953+00:00		
16	silver_booking_updates	valid_dates	check_in IS NOT NULL AND check_out IS NOT NULL AND check_out > check_in	warning	Dates must be present and check-out after check-in	2025-12-11T04:02:30.307+00:00	2025-12-11T04:02:30.307+00:00		
17	silver_properties	valid_destination	destination_id IS NOT NULL	critical	Destination ID must be present (FK validation)	2025-12-11T04:02:20.239+00:00	2025-12-11T04:02:20.239+00:00		
18	silver_bookings	valid_check_in	check_in IS NOT NULL AND check_in >= MAKE_DATE(2020, 1, 1)	critical	Check-in date must be present and reasonable	2025-12-11T04:02:19.897+00:00	2025-12-11T04:02:19.897+00:00		
19	silver_hosts	valid_rating	rating IS NOT NULL OR (rating >= 1.0 AND rating <= 5.0)	warning	Host rating should be between 1.0 and 5.0	2025-12-11T04:01:53.239+00:00	2025-12-11T04:01:53.239+00:00		
20	silver_clickstream	recent_event	timestamp >= CURRENT_DATE() - INTERVAL 90 DAYS	warning	Clickstream event should be recent (within 90 days)	2025-12-11T04:02:48.570+00:00	2025-12-11T04:02:48.570+00:00		
21	silver_booking_updates	valid_booking	booking_id IS NOT NULL	critical	Booking ID must be present (FK validation)	2025-12-11T04:02:28.888+00:00	2025-12-11T04:02:28.888+00:00		
22	silver_employees	valid_joined_date	joined_at IS NOT NULL AND joined_at >= MAKE_DATE(2015, 1, 1)	critical	Join date must be present and reasonable	2025-12-11T04:02:11.346+00:00	2025-12-11T04:02:11.346+00:00		
23	silver_properties	valid_coordinates	property_latitude BETWEEN -90 AND 90 AND property_longitude BETWEEN -180 AND 180	warning	Coordinate should be within valid geographic ranges	2025-12-11T04:02:07.897+00:00	2025-12-11T04:02:07.897+00:00		
24	silver_users	valid_created_date	created_at IS NOT NULL AND created_at >= MAKE_DATE(2020, 1, 1)	critical	Creation date must be present and after minimum valid date	2025-12-11T04:01:46.819+00:00	2025-12-11T04:01:46.819+00:00		
25	silver_booking_updates	valid_booking_update_id	booking_update_id IS NOT NULL	critical	Booking update id must be present (primary key)	2025-12-11T04:02:27.804+00:00	2025-12-11T04:02:27.804+00:00		
26	silver_clickstream	valid_event	event IS NOT NULL AND LENGTH(event) > 0	critical	Event type must be present and non-empty	2025-12-11T04:02:46.490+00:00	2025-12-11T04:02:46.490+00:00		
27	silver_bookings	reasonable_amount	total_amount BETWEEN 10 AND 100000	warning	Total amount should be within reasonable range (\$10-\$100K)	2025-12-11T04:02:23.541+00:00	2025-12-11T04:02:23.541+00:00		
28	silver_page_views	valid_page_url	page_url IS NOT NULL AND LENGTH(page_url) > 0	critical	Page URL must be present	2025-12-11T04:02:55.139+00:00	2025-12-11T04:02:55.139+00:00		
29	silver_historical_dally_metric	has_data	1=1	warning	Basic data presence check	2025-12-11T04:03:02.385+00:00	2025-12-11T04:03:02.385+00:00		
30	silver_bookings	valid_check_out	check_out IS NOT NULL AND check_out > check_in	warning	Check-out must be after check-in	2025-12-11T04:02:20.882+00:00	2025-12-11T04:02:20.882+00:00		
31	silver_bookings	valid_property	property_id IS NOT NULL	critical	Property ID must be present (FK validation)	2025-12-11T04:02:18.463+00:00	2025-12-11T04:02:18.463+00:00		
32	silver_customer_support_log	valid_created_date	created_at IS NOT NULL	critical	Creation timestamp must be present	2025-12-11T04:02:59.866+00:00	2025-12-11T04:02:59.866+00:00		
33	silver_destinations	valid_destination_id	destination_id IS NOT NULL	critical	Destination ID must be present (primary key)	2025-12-11T04:01:54.386+00:00	2025-12-11T04:01:54.386+00:00		
34	silver_bookings	valid_user	user_id IS NOT NULL	critical	User ID must be present (FK validation)	2025-12-11T04:02:17.346+00:00	2025-12-11T04:02:17.346+00:00		
35	silver_users	recent_user	created_at > CURRENT_DATE() - INTERVAL 5 YEARS	warning	User should be created within last 5 years	2025-12-11T04:01:40.412+00:00	2025-12-11T04:01:40.412+00:00		

Figure 10: Central DQ rules persisted as a table (rule name, constraint SQL, severity, timestamps), enabling consistent governance and reuse across Silver transformations.

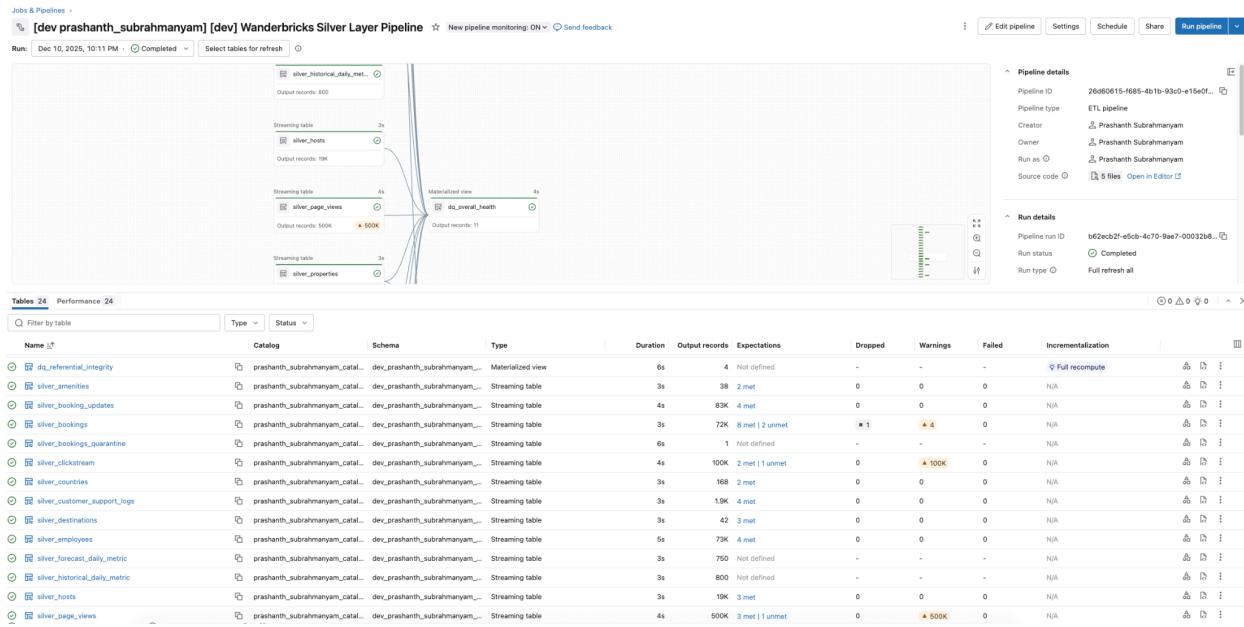


Figure 11: Silver SDP pipeline execution view showing lineage/graph and table-level outcomes (expectations, dropped/failed records, incrementalization), validating data quality enforcement.

★Checkpoint★:

1. Ensure bundle is validated and deployed successfully, and silver layer jobs run with no errors.
2. Validate the results in the UI to ensure the DQ rules show up in centralized delta table, and that the silver layer pipeline runs successfully with Expectations being checked.

None

Using [@data_product_accelerator/skills/common/databricks-autonomous-operations](#), validate, deploy, run, troubleshoot until success for the Silver layer SDP pipelines

Step 6: Create Gold Layer Merge Jobs for this dataset

Use the following prompt to create the Gold Layer for this project. This prompt will create scripts that read from the YAML files of the Gold Layer design, and setup the necessary gold tables will necessary best practices. In addition, it will ensure to create all the primary and foreign keys documented in gold layer design and create the merge scripts to read from Silver layer and write to gold layer incrementally.

Start a new Agent thread and use the following prompt:

None

```
Implement the Gold layer using  
@data_product_accelerator/skills/gold/01-gold-layer-setup/SKILL.m  
d
```

What it produces:

- Gold table creation scripts (from YAML schemas)
- Silver-to-Gold MERGE notebooks (SCD Type 1/2 dimensions, fact tables)
- FK constraint application scripts
- Asset Bundle job configuration

```

1 # Databricks notebook source
2
3 ===
4 Wonderbricks Gold Layer - Table Setup (YAML-Driven)
5
6 Creates Gold layer tables dynamically from YAML schema definitions.
7
8 YAML is the single source of truth:
9   - Column definitions
10  - Constraints
11  - Comments and descriptions
12  - Table properties
13
14 This script is generic and reusable across all domains.
15
16 Usage:
17   databricks bundle run gold_setup_job -t dev
18
19 Import yaml
20 From pythondb import Path
21 From pyspark.sql import SparkSession
22
23 def get_params():
24     return params.get("domain")
25
26 def get_params_from_databricks_widgets():
27     catalog = databricks.widgets.get("catalog")
28     gold_schema = databricks.widgets.get("gold_schema")
29     domain = get_params()
30     print(f"Catalog: {catalog}")
31     print(f"Gold Schema: {gold_schema}")
32     print(f"Domain: {domain}")
33
34     return catalog, gold_schema, domain
35
36 # Standard table properties for all Gold tables
37 standard_properties = {
38     "delta.enableChangeDataFeed": "true",
39     "delta.enableTableTracking": "true",
40     "delta.enforceTableProperties": "true",
41     "delta.automaticOptimize": "true",
42     "delta.automaticOptimize.autoCompact": "true",
43     "delta.automaticOptimize.optimizeFor": "true",
44     "logLevel": "WARN"
45 }
46
47
48 def find_yml_file(path):
49     """Find yml directory - works in Databricks and locally."""
50     possible_paths = [
51         f"{path}/yml",
52         f"{path}/yml/_files",
53         f"{path}/yml/_files/yml_designed_yml",
54         f"{path}/yml/_files/yml_designed_yml.yaml"
55     ]
56
57     for path in possible_paths:
58         if path.exists():
59             return path
60
61     raise FileNotFoundError("YML directory not found. Ensure YMLs are synced in databricks.yml")
62
63
64 def load_yml(path) -> dict:
65     """Load and parse YML file."""
66     with open(path, "r") as f:
67         return yaml.safe_load(f)
68
69
70 def escape_sql_string(s: str) -> str:
71     ...

```

★Checkpoint★:

At this stage, we can encounter some issues related to LLMs making mistakes with column names, hallucinating on new names etc.

It may take a few iterations of deployment, validation and testing to get this step right, thus be patient in this step.

Use the following prompt to start the validation process, and ensure this step is successfully completed before moving ahead.

None

Using [@data_product_accelerator/skills/common/databricks-autonomous-operations](#), validate, deploy, run, troubleshoot until success for the Gold layer setup and merge jobs

Catalog Shared Unity Catalog Serverless Serverless XL

Type to search...

For you (All)

- dev_prashanth_subrahmanyam_dev_prashanth_s...
- dev_prashanth_subrahmanyam_dev_prashanth_s...
- dev_prashanth_subrahmanyam_lakehouse_col...
- dev_prashanth_subrahmanyam_lakehouse_col...
- dev_prashanth_subrahmanyam_lakehouse_col...
- dev_prashanth_subrahmanyam_lakehouse_col...
- dev_prashanth_subrahmanyam_retail_bronze_dev
- dev_prashanth_subrahmanyam_retail_silver_de...
- dev_prashanth_subrahmanyam_system_bronze
- dev_prashanth_subrahmanyam_system_gold
- dev_prashanth_subrahmanyam_system_gold_fea...
- dev_prashanth_subrahmanyam_wanderbricks_br...
- dev_prashanth_subrahmanyam_wanderbricks_gold
- Tables (15)
 - customer_analytics_metrics
 - dim_date
 - dim_destination
 - dim_host
 - dim_property
 - dim_user
 - dim_weather_location
 - engagement_analytics_metrics
 - fact_booking_detail
 - fact_booking_daily

Catalog Explorer > prashanth_subrahmanyam_catalog > dev_prashanth_subrahmanyam_wande...

fact_booking_daily

Overview Sample Data Details Permissions Policies History Lineage Insights Quality

Description Gold layer daily aggregated booking fact table with pre-calculated KPIs at property-date grain for fast dashboard queries. Business: Primary source for dashboard KPIs, executive reporting, and high-level booking performance analysis. Pre-aggregated metrics eliminate need for transaction-level scans, enabling sub-second dashboard response times. Used for property performance dashboards, destination market analysis, and daily operational reporting. Aggregates booking counts, revenue, guest counts, and booking outcomes by property and check-in date. Ideal for time-series visualization and trend analysis. Technical: Grain is one row per property per check-in date (daily aggregate). Derived from fact_booking_detail or directly from Silver via aggregation query. Updates via Delta MERGE with additive aggregations. Composite primary key (property_id, check_in_date).

Filter columns... View relationships

Column	Type	Comment	Tags	Column masking
property_id	bigint	Property identifier (part of composite primary key). Business: Links to dim_property for property attributes, host information, and pricing details. Groups all bookings for this property on this date. Technical: FK to dim_property.property_id (business key). Part of composite PK (property_id, check_in_date).		
destination_id	bigint	Destination identifier for geographic market aggregation. Business: Links to dim_destination for geographic market analysis. Denormalized from dim_property for query performance, enables fast destination lookups without joining property dimension. Technical: FK to dim_destination.destination_id. Denormalized for performance.		
check_in_date	date	Check-in date grain of composite primary key. Business: Links to dim_date for time-based analysis and trending. Groups all bookings checking in on this date. Primary date dimension for occupancy calculations and daily trends. Technical: Date part of composite PK (property_id, check_in_date).		

Figure 13: Gold fact table definition in Unity Catalog with documented columns, primary/foreign keys, and governance tags to support discoverability and enforcement.

Step 7: Create plan for use cases and supporting Artifacts for this dataset

Now, using our gold layer design and implementation, we will come up with the necessary plans to implement supporting artifacts to operationalize various use cases from this dataset. The intention behind this step is to take the time to first identify different use cases, and then use various artifacts to realize those use cases.

Start a new Agent thread and choose only ONE of the following prompt:

None

Prompt (Workshop mode – for Learning & Enablement):

Perform project planning using

`@data_product_accelerator/skills/planning/00-project-planning/SKI
LL.md` with `planning_mode: workshop`

None

***Workshop mode** produces a minimal representative plan (3-5 TVFs, 1-2 Metric Views, 1 Genie Space) designed for hands-on workshops. It is only activated when `planning_mode: workshop` is explicitly included. See `data_product_accelerator/skills/planning/00-project-planning/references/works
hop-mode-profile.md` for details. Use following prompt for Data Product use case.*

Alternate Option: Prompt (Data Product Acceleration – default):

Perform project planning using

`@data_product_accelerator/skills/planning/00-project-planning/SKI
LL.md`

What it produces:

- Phase plan documents (TVFs, Metric Views, Monitoring, Dashboards, Genie Spaces, ML, Alerting)
- YAML manifest files in [plans/manifests/](#) (contracts for downstream stages)
- Prerequisites summary

Search Demo

New Agent

RECENT

Project plan methodology

13 files <500B · 12

Gold layer implementation

1 file · 4KB · 0

OLIVER

Project Silver Layer creation

Bronze layer creation with Apache

Gold layer design with rules

Project plan methodology

Perfect! All 13 plan documents have been successfully created! Let me provide you with a comprehensive summary:

✓ Complete Project Plans Created

I've successfully created comprehensive phased project plans for the Wandericks vacation rental analysis platform following the methodology in [./scripts/release/generate/26-project-plan-methodology.md](#). Here's what was delivered:

Plan Documents Created (13 total)

Core Planning (4 documents):

1. [README.md](#) (12 KB) - Complete index with navigation, artifact counts, instructions
2. [prerequisites.md](#) (16 KB) - Bronze layer summary (8 tables each)
3. [phase1-use-cases.md](#) (16 KB) - Master Phase 1 with implementation order
4. [PLAN_CREATION_SUMMARY.md](#) (1 KB) - Progress tracking document

Phase 1 Addendum (7 documents):

5. [phase1-addendum-1-ml-models.md](#) (20 KB) - 5 ML models with training instructions
6. [phase1-addendum-2-metrics.md](#) (12 KB) - 25+ TVs with SQL examples
7. [phase1-addendum-3-metric-views.md](#) (15 KB) - 5 Metric Views with full YAML
8. [phase1-addendum-4-lakehouse-monitoring.md](#) (13 KB) - 5 monitors with configuration
9. [phase1-addendum-5-abi-dashboards.md](#) (9 KB) - 5 dashboards with visualizations
10. [phase1-addendum-6-genie-spaces.md](#) (1 KB) - 5 Genie Spaces with instructions
11. [phase1-addendum-7-alerting.md](#) (9 KB) - 21+ SQL alerts

Additional Phases (2 documents):

12. [phase2-agent-framework.md](#) (9 KB) - 5 AI Agents with capabilities
13. [phase3-frontend-app.md](#) (13 KB) - Frontend application architecture

Total Size - 180 KB of comprehensive planning documentation

Agent Domain Framework

All 75+ artifacts are organized by these 5 domains:

Domain	Icon	Artifacts	Key Tables
Revenue	💰	MU_2, TVF_5, MU_1, Alerts_5+, fact_booking_daily, fact_payment_daily	
Engagement	📊	MU_1, TVF_1, MU_1, Alerts_4+, fact_property_engagement	
Property	🏡	TVF_6, MU_1, dim_property	

... x

QUICKSTART.md 01-databricks-expert-agent.md 26-project-plan-methodology.md 11-project-plan-prompt.md x phase3-frontend-app.md phase2-ag ...

context prompts > 01-project-plan-prompt.md > 11-project Plan Prompt

111 Project Plan Prompt

1

2

3 <create a comprehensive phased project plan for Databricks solutions (starting from Use Cases after Gold Layer is complete)>

4

5 ---

6 ## Quick Start (5 Minutes)

7

8 ## Fast Track: Create Your Project Plan

9

10 ***back

11 # 1. Verify prerequisites are complete:

12 > bronze ingestion [✓]

13 # – Silver DLT streaming [✓]

14 # – Gold dimensional model [✓]

15

16 # 2. Run the project with your project info

17 # 3. Create a project plan for project_name with:

18 # – Gold tables: (n) tables (d) dimensions + (f) facts

19 # – Use cases: (revenue analysis, marketing, operations, etc.)

20 # – Business domains: (hotels, restaurants, retail, etc.)

21 # – Agent domains: (cost, security, performance, reliability, quality)*

22

23 # 3. Output: Complete plan structure in planes/ folder

24

25

26

27 ## Key Decisions (Answer These First)

28

29 | Decision | Options | Your Choice |

30

31 | Agent Domains | 10+Time 4-4 business domains | _____ |

32 | Phase 1 Addendum | TVs, Metric Views, Dashboards, Monitoring, Genie, Alerts, MU | _____ |

33 | Phase 2 Scope | 10 Agents (optional) or skip | _____ |

34 | Phase 3 Scope | Planned App (optional) or skip | _____ |

35 | Artifact Counts | 1 Mu per domain (TVF_4+), Alerts (4+), etc. | _____ |

36

37

38

39 # Your Requirements (Fill These In First)

40

41 ## Project Information

42

43 | Field | Your Value |

44

45 | Project Name | (project_name) |

46 | Business Domains | (hotels, retail, restaurants, finance, etc.) |

47 | TVF Scopes | (marketing, sales, advertising, customer analysis, operations, etc.) |

48 | Target Stakeholders | (executives, analysts, data scientists, operations) |

49

50

51 ## Prerequisites (Must Be Complete)

52

53 | Layer | Count | Status |

54 | Bronze Tables | (0) | ✓ Complete |

55 | Silver Tables | (0) | ✓ Complete |

56 | Gold Dimensions | (0) | ✓ Complete |

57 | Gold Facts | (0) | ✓ Complete |

58

59 ## Agent Domain Framework

60

61 Define your business domains (typically 4-8):

62

63 | Domain | Icon | Focus Area | Key Gold Tables |

64

... x

DEMO

> Jupyter notebooks

> admin

> bronze

> common

> exploration

> gold

> ml

> monitoring

> pipelines

> 26-project-plan-methodology.md

> semantic-layer

> silver

01_TABLE_OF_CONTENTS.md

01 README.md

> content

> docs

> gold_layer_design

phase1-addendum-1.5-mi-models.md

phase1-addendum-1.2-vrfs.md

phase1-addendum-1.3-metric-views.md

phase1-addendum-1.4-business-domains.md

phase1-addendum-1.5-abi-dashboards.md

phase1-addendum-1.6-genie-spaces.md

phase1-addendum-1.7-alerting.md

phase1-use-cases.md

phase2-agent-framework.md

phase3-frontend-app.md

PLAN_CREATION_SUMMARY.md

prerequisites.md

README.md

resources

| bronze_copy_job.yml

| bronze_setup_job.yml

| gold_copy_job.yml

| gold_setup_job.yml

| schema.yml

| silver_dlt_pipeline.yml

| silver_dlt_setup_job.yml

> tvf

> wandericks_bronze

> wandericks_gold

> wandericks_silver

> add_ll_constraints.py

> merge_gold_tables.py

01 README.md

> silver_tables.py

wandericks_bronze_silver

> gptmore

| databricks.yml

01 QUICKSTART.md

01 README.md

Figure 14: Planning prompt executed with various plans created for use cases and associated artifacts.

Step 8: Create artifacts to support Genie use cases (Semantic Layer)

Based on the use cases and plans generated from Step 7, we will be creating a set of artifacts that support this use case in the following steps. Not every artifact is mandatory but we strongly suggest creating Table Valued Functions and Metric Views as a minimum to enable Genie use cases.

None

```
Set up the semantic layer using  
@data_product_accelerator/skills/semantic-layer/00-semantic-layer  
-setup/SKILL.md
```

What it produces:

- Metric View YAML definitions and SQL creation scripts
- TVFs optimized for Genie (STRING parameters, null safety)
- Genie Space configuration with agent instructions and benchmark questions
- Asset Bundle deployment

Table Valued Functions

Table-Valued Functions allow you to:

- Encapsulate complex joins and filters
- Expose parameterized logic
- Hide physical table complexity from consumers
- **Genie Spaces only support Table Valued Functions.**

Catalog Explorer > prashanth_subrahmanyam_catalog >

dev_prashanth_subrahmanyam_wanderbricks_gold  

Overview Details Permissions Policies

Description

Functions 26		
Name	Owner	Created at
<code>fx get_amenity_impact</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_availability_by_destination</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_booking_frequency_analysis</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_business_vs_leisure_analysis</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_cancellation_analysis</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_conversion_funnel</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_customer_geographic_analysis</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_customer_ltv</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_customer_segments</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_engagement_trends</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_host_geographic_distribution</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_host_performance</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_host_quality_metrics</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_host_retention_analysis</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_multi_property_hosts</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_payment_metrics</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_pricing_analysis</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_property_engagement</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_property_performance</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM
<code>fx get_property_type_analysis</code>	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 09:51 PM

Figure 15: Catalog view of table-valued functions published for standardized reuse of curated logic across teams and workloads.

Catalog Explorer > prashanth_subrahmanyam_catalog > dev_prashanth_subrahmanyam_wande... >

fx get_booking_frequency_analysis

[Overview](#) [Permissions](#)

Description   

LLM: Returns booking frequency distribution showing customer behavior patterns. Use this for: Frequency analysis, repeat purchase behavior, customer loyalty assessment, engagement strategies. Parameters: start_date, end_date (YYYY-MM-DD format). Example questions: "Booking frequency distribution" "How often do customers book?" "Repeat booking patterns"

Definition

```

SQL
1 WITH customer_frequency AS (
2     SELECT
3         du.user_id,
4         CASE
5             WHEN COUNT(DISTINCT fbd.booking_id) = 1 THEN '1 Booking'
6             WHEN COUNT(DISTINCT fbd.booking_id) BETWEEN 2 AND 3 THEN '2-3 Bookings'
7             WHEN COUNT(DISTINCT fbd.booking_id) BETWEEN 4 AND 6 THEN '4-6 Bookings'
8             WHEN COUNT(DISTINCT fbd.booking_id) >= 7 THEN '7+ Bookings'
9             ELSE 'No Bookings'
10        END as booking_frequency_bucket,
11        COUNT(DISTINCT fbd.booking_id) as booking_count,
12        SUM(fbd.total_amount) as revenue
13    FROM prashanth_subrahmanyam_catalog.dev_prashanth_subrahmanyam_wanderbricks_gold.dim_user du
14    LEFT JOIN prashanth_subrahmanyam_catalog.dev_prashanth_subrahmanyam_wanderbricks_gold.fact_booking_detail fbd
15        ON du.user_id = fbd.user_id
16
17 ... 38 more lines

```

Function metadata

Parameters	start_date: STRING COMMENT Start date (format: YYYY-MM-DD) end_date: STRING COMMENT End date (format: YYYY-MM-DD)
Type	TABLE
Return type	(booking_frequency_bucket STRING, customer_count BIGINT, total_bookings BIGINT, total_revenue DECIMAL(18,2), avg_booking_value DECIMAL(18,2), customer_percentage DECIMAL(5,2), revenue_percentage DECIMAL(5,2))
Language	SQL
Deterministic	True

Details

Parameter Style	S
Sql Data Access	READS_SQL_DATA
Security Type	DEFINER
Specific Name	get_booking_frequency_analysis

Figure 16: TVF definition page showing SQL implementation and interface metadata (inputs/outputs), making reusable business logic easy to consume.

Metric Views

The screenshot shows a developer interface with multiple tabs and panes. On the left, there's a sidebar with navigation links like 'Search Demo', 'New Agent', 'RECENT...', 'Metric views implementation', 'Table-valued functions implementation', 'Metric layer design with rules', 'Project plan methodology', 'Gold layer implementation', 'Project Silver Layer creation', 'Bronze layer creation with Approach...', and 'Gold layer design with rules'. The main area has a code editor with the following content:

```

1  # description: Best practices for creating Databricks Lakeview A/B dashboards with proper system table usage and layout formatting
2  # 
3  # locals:
4  #   - "$dashboard.json"
5  #   - "$dashboard.layout.json"
6  # alwaysSkip: false
7  #
8  #
9  # Databricks A/B Lakeview Dashboard Best Practices
10 #
11 # Critical: Grid System
12 #
13 #ALWAYS use a 6-column grid layout*, not 12-column!
14 #
15 # Widget Positioning Rules
16 #
17 # json
18 #
19 #   "position": {
20 #     "x": 0, // Column position: 0-5 (6-column grid)
21 #     "y": 0, // Row position: 0-6 (6-row grid)
22 #     "width": 3, // Width in columns: 1, 2, 3, 4, or 6 (must sum to 6 per row)
23 #     "height": 6 // Height in units: common values are 1, 2, 6, 9
24 #   }
25 #
26 #
27 # Grid Layout Examples
28 #
29 #   1) 3 widgets side-by-side (each 3 columns wide)
30 #     "x": 0, "y": 0, "width": 3, "height": 6 // Left
31 #     "x": 3, "y": 0, "width": 3, "height": 6 // Right
32 #
33 #   Three widgets across (each 2 columns wide)
34 #     "x": 0, "y": 0, "width": 2, "height": 2 // Left
35 #     "x": 2, "y": 0, "width": 2, "height": 2 // Center
36 #     "x": 4, "y": 0, "width": 2, "height": 6 // Right
37 #
38 #   KPI counters (1 column each, 6 across)
39 #     "x": 0, "y": 0, "width": 1, "height": 2
40 #     "x": 1, "y": 0, "width": 1, "height": 2
41 #     "x": 2, "y": 0, "width": 1, "height": 2
42 #     "x": 3, "y": 0, "width": 1, "height": 2
43 #     "x": 4, "y": 0, "width": 1, "height": 2
44 #     "x": 5, "y": 0, "width": 1, "height": 2
45 #
46 # Full-width chart
47 #     "x": 0, "y": 0, "width": 6, "height": 6
48 #
49 #
50 # Common Height Values
51 #   -> filters: height 1 or "height": 2
52 #   -> charts: height 2 or "height": 2
53 #   -> charts: "height": 6 (standard), "height": 9 (large)
54 #   -> tables: "height": 6 or more
55 #
56 #
57 # System Tables: Schema Verification

```

On the right, there's a file browser pane showing a directory structure for 'wanderbricks_gold' containing files like 'bronze_dv_job.yml', 'bronze_dv_job.yml', 'gold_merge_job.yml', 'gold_setup_job.yml', 'metric_view_setup_job.yml', 'silver_dv_pipeline.yml', 'silver_dv_pipeline.yml', 'silver_dv_setup_job.yml', 'tbl_setup.job.yml', and 'src'.

Figure 17: Authoring Metric Views using a prompt-driven workflow, where LLM-generated semantic definitions (measures, dimensions, metadata) are validated and refined before registration in Unity Catalog.

The screenshot shows the 'Catalog Explorer' interface for the catalog 'prashanth_subrahmanyam_catalog'. The main view displays a table of registered tables:

Name	Owner	Created at
fact_weather_daily	prashanth.subrahmanyam@databricks.com	Dec 10, 2025, 07:25 PM
dim_weather_location	prashanth.subrahmanyam@databricks.com	Dec 10, 2025, 07:25 PM
revenue_analytics_metrics	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 10:06 PM
engagement_analytics_metrics	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 10:06 PM
property_analytics_metrics	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 10:06 PM
host_analytics_metrics	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 10:06 PM
customer_analytics_metrics	prashanth.subrahmanyam@databricks.com	Dec 09, 2025, 10:06 PM

At the top, there are tabs for 'Overview', 'Details', 'Permissions', and 'Policies'. Below the table, there are buttons for 'AI generate' and 'Add'. The top navigation bar shows 'Catalog Explorer > prashanth_subrahmanyam_catalog > dev_prashanth_subrahmanyam_wanderbricks_gold'.

Figure 18: Metric Views registered on top of Gold to standardize KPI definitions and ensure consistent analytics consumption.

Catalog Explorer > prashanth_subrahmanyam_catalog > dev_prashanth_subrahmanyam_wande... >

 revenue_analytics_metrics 

Ask Genie P

Overview Details Permissions Lineage Insights

Description  Comprehensive revenue and booking analytics for vacation rental platform. Optimized for Genie natural language queries and AI/BI dashboards. Provides revenue trends, booking volumes, cancellation rates, and payment metrics with geographic and time-based dimensions for executive reporting and performance analysis.

Measures (10)

Name	Comment	Tags
.00 Total Revenue total_revenue	Total booking revenue aggregated across all bookings. Primary revenue KPI for business reporting, financial analysis, and executive dashboards. Represents total value of confirmed and completed bookings.	
.2 Booking Count booking_count	Total number of bookings. Volume metric for demand analysis, capacity planning, and understanding booking patterns across properties and destinations.	
.00 Avg Booking Value avg_booking_value	Average booking value per reservation. Pricing effectiveness indicator showing typical transaction size. Used for pricing strategy and revenue optimization.	
.2 Total Guests total_guests	Total guest count across all bookings. Occupancy planning metric showing capacity utilization and guest volume patterns for operational planning.	
.00 Avg Nights avg_nights	Average length of stay in nights. Shows typical booking duration patterns for inventory management and understanding guest behavior.	
.2 Cancellations cancellation_count	Number of cancelled bookings. Volume metric for understanding cancellation patterns and impact on revenue forecasting.	
.2 Confirmed Bookings confirmed_count	Number of confirmed bookings. Represents finalized reservations used for capacity planning and revenue recognition.	
.00 Cancellation Rate cancellation_rate	Percentage of bookings cancelled. Key operational metric for understanding cancellation patterns and impact on revenue. Used for forecasting accuracy.	
.00 Payment Rate payment_rate	Average payment completion rate showing percentage of bookings with completed payments. Indicates payment process effectiveness and revenue realization.	
.2 Property Count property_count	Number of unique properties with bookings. Shows active inventory utilization and helps understand portfolio breadth.	

Dimensions (8)

Name	Comment	Tags
Check-in Date check_in_date	Check-in date for the booking used for time-based revenue analysis and trending	
Property ID property_id	Property identifier linking bookings to property listings	
Property Title property_title	Property listing title for user-friendly property identification in reports	
Property Type property_type	Type of property (house, apartment, villa) for category-based	

Figure 19: Metric View semantic definition (measures, dimensions, descriptions/metadata) used to drive governed, repeatable BI semantics.

Genie Spaces

Figure 20: Genie Spaces implementation prompt, showing how the framework generates a document that can guide you through everything needed to set-up a Genie Space in few minutes, constrained to approved semantic artifacts (TVFs + Metric Views).

Figure 21: Genie Spaces implementation workflow, showing how approved semantic artifacts like Table-Valued Functions can be added for an improved quality genie space.

None

Using
@data_product_accelerator/skills/common/databricks-autonomous-operations, validate, deploy, run, troubleshoot until success for the Semantic layer jobs

Step 9: (Optional) Create artifacts to support new use cases

ML Models

Machine learning artifacts in this framework are built directly on top of the curated Gold and Semantic layers, ensuring that features are business-aligned, well-documented, and consistent with analytical use cases.

The framework leverages MLflow for experiment tracking, model versioning, and lifecycle management, and Unity Catalog for governance and access control. Models are treated as first-class assets alongside tables, views, and functions, enabling discoverability, auditability, and controlled deployment.

The following prompt will create the necessary feature stores, training pipelines as well as batch inference jobs for ML. Further customization for real-time model serving can be done, but this has not been implemented in this example.

None

Set up the ML pipeline using
@data_product_accelerator/skills/ml/00-ml-pipeline-setup/SKILL.md

```

monitoring-implementation-summary.md
phase1-addendum-1.5-abi-dashboards.md
ml-implementation-summary.md
README.md
dbnabrics.yml
...

```

ML models implementation plan

Using # 12-ml-models-primer.md while grounding in # 12-ml-models-primer.md, please implement # phase1-addendum-1.5-ml-models.

④ 9 of 9 To-dos Complete

Implementation Completed!

I've successfully implemented Phase 1 Addendum 1.5. ML Models with all production-ready ML models following the cursor rule patterns.

What Was Created (13 Items)

- Feature Store (1 script)
- Metric Views Implementation (28)
- Project plan methodology (6)
- Gold layer implementation (6)
- Project Silver Layer creation (1)
- Show more

Table-valued Functions Implementation (20)

Metric Views Implementation (28)

Project plan methodology (6)

Gold layer implementation (6)

Project Silver Layer creation (1)

Show more

ML models implementation plan

Using # 12-ml-models-primer.md while grounding in # 12-ml-models-primer.md, please implement # phase1-addendum-1.5-ml-models.

④ 9 of 9 To-dos Complete

Implementation Completed!

I've successfully implemented Phase 1 Addendum 1.5. ML Models with all production-ready ML models following the cursor rule patterns.

What Was Created (13 Items)

- Feature tables (property, user, destination) (1)
- Training Pipelines (5 models):

 - 1. src/wanderbricks_ml/models/demand_predictor/train.py - XGBoost for booking demand forecasting
 - 2. src/wanderbricks_ml/models/pricing_optimizer/train.py - XGBoost for optimal pricing
 - 3. src/wanderbricks_ml/models/conversion_predictor/train.py - Logistic regression for conversion prediction
 - 4. src/wanderbricks_ml/models/customer_ltv_predictor/train.py - XGBoost for customer lifetime value prediction
 - 5. src/wanderbricks_ml/models/churn_predictor/train.py - XGBoost for churn prediction

- Batch Inference (1 unified pipeline):

 - src/wanderbricks_ml/inference/churn_inference.py - Supports all 5 models with model-specific preprocessing

- Asset Bundle Jobs (3 orchestrated):

 - resources/ml/feature_store_setup.job.yml - Feature table creation
 - resources/ml/training_arachitecture.job.yml - Parallel training (all 5 models)
 - resources/ml/assets_inference.job.yml - Daily batch scoring

Configuration & Documentation:

 - src/wanderbricks_ml/_dashboards/_dashboards.yaml - Updated with _dashboards variable and includes
 - src/wanderbricks_ml/_README.md - Comprehensive ML documentation
 - docs/deployment/ml-implementation-summary.md - Implementation summary

Key Features

AI Critical Patterns Entered:

 - Experiment Path: /databricks/wanderbricks_ml/_dashboards/_dashboards.yaml
 - Dataset Logging: Inside #!flow_start() context
 - Helper Functions: ALWAYS inferred (no imports)
 - Exit Signal: dbutils.notebook.exit("SUCCESS") explicit
 - UC Signature: BOTH input AND output specified

Schemas-Grounded: All features verified against Gold layer YAML schemas

Production-Ready: Serverless compute, Unity Catalog model registry.

10 files

Figure 22: End-to-end ML model development workflow, combining LLM-assisted code generation, MLflow experiment tracking, and model registration as governed assets in Unity Catalog.

Experiments

GenAI apps & agents

Build a Generative AI application or agent

Foundation Model Fine-tuning Preview

Fine-tune a foundation model

MLflow 3 is available!

Featuring unified ML and GenAI experiment tracking, improved model logging, prompt versioning, enhanced LLM support, and much more.

wanderbricks

X
Q
 Only my experiments
Reset filters

Name	Created by
wanderbricks_ml_customer_segmentation	Prashanth Subrahmanyam
wanderbricks_ml_demand_predictor	Prashanth Subrahmanyam
wanderbricks_ml_conversion_predictor	Prashanth Subrahmanyam
wanderbricks_ml_customer_ltv_predictor	Prashanth Subrahmanyam
wanderbricks_ml_pricing_optimizer	Prashanth Subrahmanyam

Figure 23: MLflow experiments for the Wanderbricks use cases, tracking training runs and enabling reproducibility across ML workflows.

Figure 24: Unity Catalog schema that organizes ML-related tables/assets (feature-like tables and supporting datasets) alongside governed ownership metadata.

Figure 25: Published/registered models visible in Unity Catalog, enabling governed discovery and consistent deployment practices.

Figure 26: Operational monitoring view capturing run history and evaluation-style metrics, supporting observability for ML/data-product behavior over time.

Step 10: (Optional) Create monitoring and observational artifacts

Set up Lakehouse Monitors, AI/BI Dashboards, and SQL Alerts.

- Lakehouse Monitors with custom business metrics
- AI/BI Dashboard JSON definitions
- Asset Bundle deployment

None

Set up observability using
[`@data_product_accelerator/skills/monitoring/00-observability-setup/SKILL.md`](#)

Lakehouse Monitoring

Lakehouse Monitoring closes the loop between data ingestion, transformation, and consumption by providing continuous visibility into time-series profiles for normal tables as well as inference tables. Because downstream analytics, BI, and ML workloads depend on stable and trustworthy data, observability is treated as a core requirement rather than an afterthought.

In this framework, monitoring artifacts are generated declaratively and integrated directly with Gold layers, allowing teams to detect issues early and maintain confidence in the data product over time. The lakehouse monitoring includes Custom Business Metrics that are generated by the LLM and can be used to monitor data slices.

```

# MONITORING_DEPLOYMENT_CHECKLIST.md
monitoring-implementation-summary.md
monitoring-quickstart.md
phasel-addendum-14-lakehouse-monitoring.md

docs/deployment/monitoring-implementation-summary.md
monitoring-implementation-summary.md
monitoring-quickstart.md
phasel-addendum-14-lakehouse-monitoring.md

1 # WanderBricks Lakehouse Monitoring - Implementation Summary
2
3 docs/deployment/monitoring-implementation-summary.md
4 monitoring-implementation-summary.md
5 monitoring-quickstart.md
6 phasel-addendum-14-lakehouse-monitoring.md
7
8
9
10 # What Was Implemented
11
12
13 ## 5 Production Monitors
14
15 | # | Monitor | Table | Type | Custom Metrics |
16 | 1 | Booking Revenue Monitor | fact_booking_daily | Time Series | 9 metrics | Daily 2 AM UTC |
17 | 2 | Property Data Monitor | fact_property | Time Series | 4 metrics | Daily 2 AM UTC |
18 | 3 | Property Data Monitor | dim_property | Snapshot | 4 metrics | Daily 2 AM UTC |
19 | 4 | Host Quality Monitor | dim_host | Snapshot | 4 metrics | Daily 2 AM UTC |
20 | 5 | Customer Data Monitor | dim_user | Snapshot | 3 metrics | Daily 2 AM UTC |
21
22 **Total Custom Metrics**== 28 (17 AGGREGATE + 11 DERIVED)
23
24
25 ## File Created
26
27
28 ## Production Code
29
30
31 | File | Purpose | Lines | Key Features |
32 | --- | --- | --- | --- |
33 | `src/wanderbricks_gold/monitoring/setup_monitors.py` | Monitor setup script | 729 | - Complete cleanup pattern<br>- Graceful error handling<br>- SDK compatibility checks<br>- 5 monitor creation functions<br>- Table-level metrics pattern [!row-column!=""table"]| |
34 | `src/wanderbricks_gold/monitoring/monitoring_metrics.sql` | Monitor setup script | 729 | - Complete cleanup pattern<br>- Graceful error handling<br>- SDK compatibility checks<br>- 5 monitor creation functions<br>- Async initialization handling<br>- Asset Build job | 58 | - Serverless compute<br>- Email notifications<br>- Parameter passing<br>- Timeout handling |
35 | `resources/monitoring_setup.job.yaml` | Asset Build job | 50 | |
36 | `resources/monitoring_setup.job.yml` | Asset Build job | 50 | |
37
38 ## Query Examples & Documentation
39
40 | File | Purpose | Lines |
41 | --- | --- | --- |
42 | `src/wanderbricks_gold/monitoring/query_monitoring_metrics.sql` | Query patterns | 322 | |
43 | `docs/deployment/monitoring/query-setup-side.md` | Complete setup guide | 40+ | |
44 | `docs/deployment/monitoring-quickstart.md` | Quick start | 280+ | |
45 | `docs/deployment/monitoring-implementation-summary.md` | This file | 300+ | |
46
47 **Documentation Total**== 1,272+ Lines
48
49
50 ## Custom Metrics Detail
51
52 ## 1. Booking Revenue Monitor (fact_booking_daily)
53
54
55 ## Purpose<br> Track booking performance, revenue, and conversion rates
56
57 | Metric Name | Type | Definition | Business Value |
58 | --- | --- | --- | --- |
59 | `avg_booking_value` | AGGREGATE | AVG(book.booking_value) | Average booking size tracking |
60 | `total_bookings` | AGGREGATE | SUM(book.booking_count) | Volume metric |
61 | `total_Confirmed` | AGGREGATE | SUM(book.booking_is_confirmed) | Confirmed volume |
62 | `total_Confirmed_Conversion` | AGGREGATE | SUM(book.booking_is_confirmed * book.booking_is_converted) | Conversion tracking |
63 | `total_revenue` | AGGREGATE | SUM(book.booking_value) | Revenue |

```

Figure 27: Lakehouse Monitoring implementation prompt and workflow, generating monitoring assets to track data quality, freshness, volume, and distribution signals for ongoing observability of the data product.

AI / BI Dashboards

AI / BI Dashboards represent the primary human-facing consumption layer of the data product. These dashboards are intentionally built on top of Metric Views and semantic artifacts rather than raw tables, ensuring that visualizations remain thin, maintainable, and consistent.

By minimizing embedded business logic in dashboards, teams can evolve metrics and rules centrally without requiring widespread rework across reports.

Databricks AI/BI dashboards are serialized JSON files, and hence can be generated by an LLM. The generated files may have some minor issues (such as referencing right catalog, schema in queries, and choosing right parameters on visuals etc), which can be fixed in the UI, but in general, this can be a tremendous accelerator to dashboard development.

The screenshot shows a code editor with several tabs open, illustrating the AI/BI dashboard generation process. The tabs include:

- customer_insights_dashboard\dashboard\livedash.json**: A JSON file defining the dashboard structure.
- hf-implementation-summary.md**, **phase1-addendum-12-rvls.md**, **hf-deployment-guide.md**, **validate_rvls.sql**: Documentation and deployment files.
- customer_insights_dashboard\livedash.json**: Another view of the dashboard definition.
- customer_insights_dashboard\livedash.json**: A third view of the dashboard definition.
- DEMO**: A folder containing cursor, context, data, and dashboards.
- dashboards**: A folder containing customer_insights_dashboard.livedash.json.
- customer_insights_dashboard.livedash.json**: The final dashboard definition.
- IMPLEMENTATION_SUMMARY.md**: A summary document.
- QUICKSTART.md**: A quickstart guide.
- README.md**: A general README.
- gold_layer_design**: A folder containing gold_layer_design\plan, gold_layer_design\resources, gold_layer_design\scripts, and gold_layer_design\sql.
- gold_layer_design\plan**: A plan file.
- gold_layer_design\resources**: Resource files.
- gold_layer_design\scripts**: Script files.
- gold_layer_design\sql**: SQL files.
- gignore**: A ignore file.
- databricksm.yml**: A configuration file.
- METRIC_VIEWS_COMPLETE.md**: A documentation file.
- QUICKSTART.md**: Another quickstart file.
- README.md**: Another README file.

The code editor displays a large JSON object representing the dashboard structure, including sections like "Revenue Performance Dashboard", "Engagement Analytics Dashboard", and "Customer Insights Dashboard". It also includes "Key Features" and "File Location" sections.

Figure 28: AI/BI Dashboard generation prompt, showing how dashboards are created on top of governed Metric Views/semantic definitions to keep BI logic centralized and consistent.

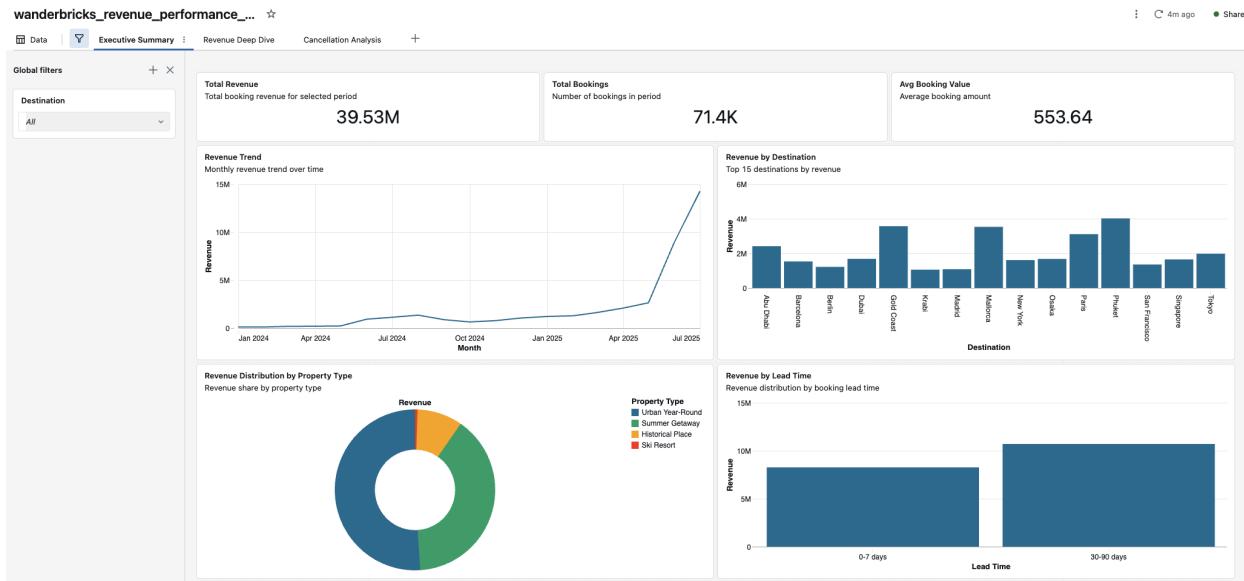


Figure 29: Executive BI dashboard built on the semantic layer, showcasing business KPIs and trends powered by the Gold model and governed metrics.

★Checkpoint★:

At this stage, we can encounter some issues related to LLMs making mistakes with column names, hallucinating on new names etc.

It may take a few iterations of deployment, validation and testing to get this step right, thus be patient in this step.

Use the following prompt to start the validation process, and ensure this step is successfully completed before moving ahead.

None

Using

```
@data_product_accelerator/skills/common/databricks-autonomous-operations, validate, deploy, run, troubleshoot until success for  
the Monitoring layer jobs
```