Cheatsheet – Day 3

Medallion & Lakeflow, Orchestration, Governance & Security

Medallion Architecture

Layer	Purpose	Data Quality
Bronze	Raw ingestion, append-only	As-is from source
Silver	Cleaned, validated, deduplicated	Enforced expectations
Gold	Business aggregations, dimensions, facts	Curated for analytics

---

Lakeflow (DLT) SQL Declarations

```
-- Bronze: streaming ingestion
CREATE OR REFRESH STREAMING TABLE bronze_orders
AS SELECT * FROM STREAM read_files('/path', format =>
        'json');


-- Silver: validated + cleaned
CREATE OR REFRESH STREAMING TABLE silver_orders (
    CONSTRAINT valid_id EXPECT (id IS NOT NULL) ON VIOLATION
        DROP ROW,
    CONSTRAINT valid_amount EXPECT (amount > 0) ON VIOLATION
        FAIL UPDATE
)
AS SELECT *, current_timestamp() AS processed_at
FROM STREAM(LIVE.bronze_orders);
```

```sql
-- Gold: aggregated (materialized view)
CREATE OR REFRESH MATERIALIZED VIEW gold_daily_revenue
AS SELECT date, SUM(amount) AS revenue, COUNT(*) AS orders
FROM LIVE.silver_orders
GROUP BY date;
```

STREAMING TABLE vs MATERIALIZED VIEW

Feature	STREAMING TABLE	MATERIALIZED VIEW
Processing	Incremental (append)	Full recompute
Source	Streaming/append sources	Any query
Best for	Raw ingestion, append-only	Aggregations, joins
Reference	`STREAM(LIVE.x)`	`LIVE.x`

Expectations (Data Quality)

Violation Action	Behavior
(omitted)	Warn only, keep all rows, log metrics
`DROP ROW`	Silently drop failing rows
`FAIL UPDATE`	Abort pipeline, no data written

APPLY CHANGES (SCD / Change Data Capture)

```sql
-- SCD Type 1 (overwrite) — default
CREATE OR REFRESH STREAMING TABLE silver_customers;

APPLY CHANGES INTO LIVE.silver_customers
FROM STREAM(LIVE.bronze_customers)
KEYS (customer_id)
SEQUENCE BY updated_at
```

```
COLUMNS * EXCEPT (_rescued_data)
STORED AS SCD TYPE 1;  -- keeps only latest version

-- SCD Type 2 (history tracking)
APPLY CHANGES INTO LIVE.dim_customers
FROM STREAM(LIVE.bronze_customers)
KEYS (customer_id)
SEQUENCE BY updated_at
STORED AS SCD TYPE 2;  -- adds __START_AT, __END_AT columns
```

SCD Type	Behavior	History	Extra Columns
Type 1	Overwrite (upsert)	No	None
Type 2	Track all changes	Yes	`__START_AT`, `__END_AT`

FLOW (Multi-Source Append)

```
-- Append data from multiple sources into one table
CREATE OR REFRESH STREAMING TABLE all_orders;

CREATE FLOW append_region_eu
AS INSERT INTO LIVE.all_orders
SELECT * FROM STREAM(LIVE.bronze_eu_orders);

CREATE FLOW append_region_us
AS INSERT INTO LIVE.all_orders
SELECT * FROM STREAM(LIVE.bronze_us_orders);
```

Delta Sharing

```
-- Create share
CREATE SHARE my_share;
ALTER SHARE my_share ADD TABLE catalog.schema.gold_report;

-- Create recipient
CREATE RECIPIENT partner_org;
```

```sql
-- Grant access
GRANT SELECT ON SHARE my_share TO RECIPIENT partner_org;
```

- Open protocol (cross-platform, any client)
- Recipient gets read-only, live access
- No data copy — shared in place

Event Log (Data Quality Metrics)

```sql
SELECT timestamp,
       details:flow_progress:data_quality:expectations
FROM event_log(TABLE(catalog.schema.silver_orders))
WHERE details:flow_progress:data_quality IS NOT NULL;
```

Databricks Workflows (Jobs)

```
Job
  ├── Task 1 (notebook)
  │     └── taskValues.set("key", value)
  ├── Task 2 (depends on Task 1)
  │     └── taskValues.get("Task_1", "key")
  └── Task 3 (depends on Task 2)
```

```python
# Pass values between tasks
dbutils.jobs.taskValues.set(key="row_count", value=1000)

# Retrieve in downstream task
count = dbutils.jobs.taskValues.get(
    taskKey="upstream_task", key="row_count")
```

Orchestration Patterns

Feature	Purpose
Task dependencies	DAG execution order
Retry policy	Auto-retry on failure
Alerts (email/webhook)	Notify on success/failure
Repair Run	Re-run only failed tasks
Parameters	Pass runtime values to jobs

Governance: GRANT / REVOKE

```
-- Grant read access
GRANT USE CATALOG ON CATALOG my_catalog TO `analysts`;
GRANT USE SCHEMA ON SCHEMA my_catalog.silver TO `analysts`;
GRANT SELECT ON SCHEMA my_catalog.silver TO `analysts`;

-- Revoke
REVOKE MODIFY ON SCHEMA my_catalog.bronze FROM `analysts`;
```

Views Comparison

Feature	Temp View	Global Temp View	Permanent View
Scope	Session only	All sessions (cluster)	Persistent (UC)
SQL prefix	`view_name`	`global_temp.view_name`	`catalog.schema.view`
Survives restart?	No	No	Yes
Data stored?	No (query only)	No (query only)	No (query only)
	No	No	

Feature	Temp View	Global Temp View	Permanent View
Governance (UC)			Yes (GRANT/ REVOKE)

```
CREATE OR REPLACE TEMP VIEW my_temp AS SELECT ...;
CREATE OR REPLACE GLOBAL TEMP VIEW my_global AS SELECT ...;
CREATE OR REPLACE VIEW catalog.schema.my_perm AS SELECT ...;
```

Row Filters & Column Masks

```
-- Row Filter: restrict visible rows
CREATE FUNCTION schema.row_filter(store_id STRING)
RETURN IF(is_account_group_member('admins'), true,
          store_id = current_user());

ALTER TABLE t SET ROW FILTER schema.row_filter ON (store_id);
ALTER TABLE t DROP ROW FILTER;  -- remove

-- Column Mask: redact sensitive data
CREATE FUNCTION schema.mask_email(email STRING)
RETURN IF(is_account_group_member('admins'), email,
          CONCAT('***@', SUBSTRING_INDEX(email, '@', -1)));

ALTER TABLE t ALTER COLUMN email SET MASK schema.mask_email;
ALTER TABLE t ALTER COLUMN email DROP MASK;  -- remove
```

INFORMATION_SCHEMA

```
-- List all tables
SELECT table_schema, table_name, table_type
FROM catalog.INFORMATION_SCHEMA.TABLES;

-- List columns
SELECT column_name, data_type, ordinal_position
```

```sql
FROM catalog.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'my_table';

-- List privileges
SELECT * FROM catalog.INFORMATION_SCHEMA.TABLE_PRIVILEGES;
```

Monitoring & Observability (System Tables)

System Table	Purpose
`system.billing.usage`	DBU consumption by SKU, user, workspace
`system.lakeflow.job_run_timeline`	Job run history, success/ failure rates
`system.lakeflow.pipeline_event_log`	Lakeflow pipeline events, update health
`system.query.history`	SQL Warehouse query performance
`system.compute.clusters`	Cluster uptime, node types, DBU usage
`system.compute.warehouse_events`	SQL Warehouse

System Table	Purpose
	scaling events
`system.storage.predictive_optimization_operations_history`	OPTIMIZE/ VACUUM auto-runs
`system.access.audit`	Who did what, when (security audit)
`system.access.table_lineage`	Data flow: source -> target tables
`system.access.column_lineage`	Column-level dependency tracking

```sql
-- Daily DBU cost by SKU (last 30 days)
SELECT usage_date, sku_name,
       ROUND(SUM(usage_quantity), 2) as total_dbus
FROM system.billing.usage
WHERE usage_date >= current_date() - INTERVAL 30 DAYS
GROUP BY usage_date, sku_name
ORDER BY usage_date DESC;

-- Failed jobs (last 24h)
SELECT job_name, result_state,
       period.start_time, error_message
FROM system.lakeflow.job_run_timeline
WHERE period.start_time >= current_timestamp() - INTERVAL 24
        HOURS
  AND result_state != 'SUCCESS';

-- Slowest queries (> 60s, last 7 days)
SELECT executed_by,
       SUBSTRING(statement_text, 1, 100) as query,
       ROUND(total_duration_ms / 1000, 1) as seconds
```

```sql
FROM system.query.history
WHERE start_time >= current_timestamp() - INTERVAL 7 DAYS
  AND total_duration_ms > 60000
ORDER BY total_duration_ms DESC;
```

Key Exam Points

- `LIVE.x` = pipeline-scoped reference (not in Unity Catalog)
- `STREAM(LIVE.x)` for STREAMING TABLE sources
- `DROP ROW` = silent removal; `FAIL UPDATE` = abort pipeline
- Expectations without `ON VIOLATION` = warn only
- `taskValues` = pass data between Job tasks
- Unity Catalog = deny-by-default
- Row Filter = SQL UDF returning BOOLEAN
- Column Mask = SQL UDF returning same type as column
- `is_account_group_member()` = dynamic security function
- `INFORMATION_SCHEMA` = metadata audit (read-only)
- `APPLY CHANGES` = Lakeflow CDC (SCD Type 1 or 2)
- SCD Type 2 adds `__START_AT` and `__END_AT` columns
- `FLOW` = append from multiple sources into one streaming table
- Delta Sharing = open protocol, no data copy, read-only
- Temp View dies with session; Global Temp dies with cluster
- Permanent View = stored in Unity Catalog, has governance
- `system.billing.usage` = DBU cost tracking
- `system.access.audit` = security audit log (who/what/when)
- `system.access.table_lineage` = automatic data flow tracking