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

```sql
-- 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);

-- 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

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

Feature	Purpose
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`;
```

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)));
```

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

INFORMATION_SCHEMA

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

-- List columns
SELECT column_name, data_type, ordinal_position
FROM catalog.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'my_table';

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

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)