```
   ...

   # Read the staged claim records into memory
   curated_policies = dlt.read(“curated_policies”)
   # Evaluate the validity of the claim
   curated_claims = curated_claims \
       .alias(“a”) \
       .join(
           curated_policies.alias(“b”),
           on  = F.col(“a.policy_number”) == F.col(“b.policy_number”),
           how = “left”
       ) \
       .select([F.col(f”a.{c}”) for c in curated_claims.columns] + [F.col(f”b.
{c}”).alias(f”policy_{c}”) for c in (“effective_date”, “expiry_date”)]) \
       .withColumn(
           # Calculate the number of months between coverage starting and the
claim being filed
           “months_since_covered”, F.round(F.months_between(F.col(“claim_date”),
F.col(“policy_effective_date”)))
       ) \
       .withColumn(
           # Check if the claim was filed before the policy came into effect
           “claim_before_covered”, F.when(F.col(“claim_date”) < F.col(“policy_
effective_date”), F.lit(1)).otherwise(F.lit(0))
       ) \
       .withColumn(
           # Calculate the number of days between the incident occurring and the
claim being filed
           “days_between_incident_and_claim”, F.datediff(F.col(“claim_date”),
F.col(“incident_date”))
       )

   # Return the curated dataset
   return curated_claims
```

One significant advantage of DLT is the ability to specify and enforce data quality standards. We can set expectations for each DLT table with detailed data quality constraints that should be applied to the contents of the table. Currently, DLT supports expectations for three different scenarios:

Decorator	Description
`expect`	Retain records that violate expectations
`expect_or_drop`	Drop records that violate expectations
`expect_or_fail`	Halt the execution if any record(s) violate constraints

Expectations can be defined with one or more data quality constraints. Each constraint requires a description and a Python or SQL expression to evaluate. Multiple constraints can be defined using the `expect_all`, `expect_all_or_drop`, and `expect_all_or_fail` decorators. Each decorator expects a Python dictionary where the keys are the constraint descriptions, and the values are the respective expressions. The example below shows multiple data quality constraints for the retain and drop scenarios described above.

```
@dlt.expect_all({
   “valid_driver_license”: “driver_license_issue_date > (current_date() -
cast(cast(driver_age AS INT) AS INTERVAL YEAR))”,
   “valid_claim_amount”: “total_claim_amount > 0”,
   “valid_coverage”: “months_since_covered > 0”,
   “valid_incident_before_claim”: “days_between_incident_and_claim > 0”
})
@dlt.expect_all_or_drop({
   “valid_claim_number”: “claim_number IS NOT NULL”,
   “valid_policy_number”: “policy_number IS NOT NULL”,
   “valid_claim_date”: “claim_date < current_date”,
   “valid_incident_date”: “incident_date < current_date”,
   “valid_incident_hour”: “incident_hour between 0 and 24”,
   “valid_driver_age”: “driver_age > 16”,
   “valid_effective_date”: “policy_effective_date < current_date()”,
   “valid_expiry_date”: “policy_expiry_date <= current_date()”
})
def curate_claims():
    ...
```

databricks