Polaris REST Idempotency Key Proposal

Author: Huaxin Gao

Summary

Introduce an optional Idempotency-Key HTTP header for mutation requests in the Polaris REST API (e.g., create table/namespace, commit/update table, rename/drop, property updates). When present, Polaris guarantees at-most-once execution for the same logical operation, enabling safe client retries after transient failures without side effects. The feature is backward compatible and opt-in for both servers and clients.

This document covers server-side changes. See Iceberg REST Catalog Idempotency for the companion client proposal.

Motivation

Polaris exposes REST endpoints for table/namespace management and commits that traverse multiple components (LB -> API server -> storage/persistence). Transient failures (network partitions, restarts, timeouts) can leave clients uncertain whether a mutation completed. Naïvely retrying can:

- Attempt a second commit with the same payload (duplicate effect), or
- Trigger client-side cleanup for a commit that actually succeeded (breaking referential integrity and corrupting snapshots).

Current pain point example

1. Client calls `POST /v1/tables/{name}/commit` and server commits snapshot S.
2. A timeout or 5xx leads the server to return a commit state unknown error.
3. Client retries; server rejects with `409`.
4. Client misinterprets `409` as failure and deletes manifest files, leaving metadata pointing at missing files.

Short-term (today): clients avoid retrying when commit status is unknown to prevent duplicate side effects.

Long-term (this proposal): With an Idempotency-Key, the server can safely recognize repeats (replay original result, return 409 for in-flight, 422 for mismatched payloads, or reconcile stale in-progress entries), so clients *can* retry on timeouts/5xx without risk.

Net effect: An Idempotency-Key lets Polaris detect repeated logical operations and return the original outcome (or reconcile), eliminating these failure-induced hazards and simplifying client logic.

Goals

- Guarantee at-most-once semantics for the same logical operation/payload.
- Ensure safe retries for non-validation failures (e.g., timeouts, 5xx), preventing duplicate side effects.
- By default keep behavior unchanged when the header is absent.
- Expose capability discovery so clients know whether keys are honored and for how long.

Non-Goals

- Mandate a specific persistence schema or storage engine.

API Design

Header: `Idempotency-Key`

- **Where:** HTTP request header on mutation routes (POST/PUT/DELETE).
- **Type:** string; **Pattern:** `^[a-zA-Z0-9][a-zA-Z0-9_.-]*$`; **Length:** 1–255.
- **Generation:** Clients SHOULD use a cryptographically random generator (e.g., UUID v4).
- **Reuse rule:** A key MUST be reused only when retrying the same canonical request payload for the same logical operation.

Scope

Keys are scoped to **(HTTP method, normalized resource path, tenant/catalog identifier, Idempotency-Key)** to prevent cross-endpoint collisions. Examples:

- `(POST, /v1/tables/db.tbl/commit, tenantA, K)`
- `(POST, /v1/namespaces/db, tenantA, K)`

Server Behavior

- ~~First acceptance: Bind the scoped key to the canonical payload hash.~~
- First acceptance (Normative): Associate the scoped key with a stable payload identity for the request.

  Informative (Polaris example): Compute the identity as SHA-256 over RFC 8785–canonicalized JSON.

- Duplicate key with same canonical payload (hash matches): Return the original finalized response (success 200/201/204 or the original terminal 4xx); do not re-execute. Transient 5xx MUST NOT be stored or replayed.
- Duplicate key with different canonical payload (hash differs): Return 422 Unprocessable Content (problem type: `idempotency_key_conflict`).
- In-flight duplicate: If the key is reserved but not finalized for the same payload, return 409 Conflict (problem type: `request_in_progress`; MAY include `Retry-After`).

- ~~Result storage: Only finalized outcomes (success or terminal 4xx) may be stored and replayed; transient 5xx MUST NOT be stored.~~
- Replay policy (Normative): A prior result may be replayed only if the earlier attempt reached a finalized outcome (`2xx` or terminal `4xx`). The server MUST NOT replay transient `5xx` or unknown outcomes.

  Informative (Polaris example): Implementations typically persist enough information to enable replay, but the exact mechanism is an internal concern.

Discovery (Server -> Client)

When Polaris serves the Iceberg REST Catalog API, it MUST advertise idempotency support via **GET /v1/config** (getConfig). The response contains a `properties` map (string -> string).

Example getConfig response

```
{
  "properties": {
    "idempotency-key-supported": "true",
    "idempotency-key-lifetime": "PT30M"
  }
}
```

Fields

- `idempotency-key-supported` — `"true"` or `"false"` (string values, per getConfig conventions).
- `idempotency-key-lifetime` — ISO-8601 duration string (e.g., `"PT30M"`); advisory minimum retention window.

Client behavior:

- If getConfig.properties is absent, or either key is missing/invalid, clients MUST treat idempotency as unsupported and MUST NOT enable automatic same-key retries.
- Only when both keys are present and valid may clients enable idempotent retries (bounded by the advertised lifetime).

Status Codes

- `200/201/204` — Success; duplicates with same payload return the original success.
- `409 Conflict` — Duplicate for a key currently IN_PROGRESS.
- `422 Unprocessable Content` — Same key used with a different payload.
- `5xx Server error` — For mutation endpoints, clients MUST NOT retry by default. If and only if Idempotency-Key is present and the server advertises idempotency via /v1/config, clients MAY retry the same key within the advertised idempotency-key-lifetime.

OpenAPI Additions

Add an optional `Idempotency-Key` header parameter to all mutation endpoints in `openapi.yaml` and enumerate `409/422` responses where applicable.

Client-Side Changes (Polaris & Iceberg Integrations)

- Attach `Idempotency-Key` to mutation requests that may be retried (default: auto-generate UUID v4 per operation).
- Ensure client retry logic reuses the same Idempotency-Key for retries of the same logical operation and payload.
- Read discovery to decide retries and stop once elapsed time exceeds `idempotency-key-lifetime` (raise `IdempotencyWindowExpired`).
- Expose APIs for application clients to supply an Idempotency-Key per operation.

Server Design (Implementation-Flexible)

Canonicalization

- Use **RFC 8785** (JSON Canonicalization Scheme) for deterministic payload hashing across languages.
- Hash function: `SHA-256(canonicalPayload)`; store lowercase hex.

Request Flow (HTTP filter/interceptor)

- HTTP filter extracts Idempotency-Key, canonicalizes the payload (RFC 8785), and computes H(P) = SHA-256(canonicalPayload).
- Reserve: attempt to create a reservation row for *(scope, key)* bound to hash *H*. The row is in progress until a terminal HTTP status (2xx or terminal 4xx) is written. If duplicate:
    - FINALIZED & H match -> replay stored success (200/201/204).
    - IN_PROGRESS & H match ->
        - if still active -> 409 `request_in_progress`;
        - else -> Reconciliation (see below Reconciliation section).
    - H differs -> 422 `Unprocessable Content (idempotency_key_conflict)`.

- Execute the mutation handler (only when reserved).
- Finalize: set FINALIZED, persist terminal response (200/201/204 or terminal 4xx). Transient 5xx are returned but not persisted; the row remains IN_PROGRESS (retry -> reconciliation).
- Error paths: if commit succeeds but finalize fails, the row remains IN_PROGRESS; On retry, apply the reconciliation flow to verify/apply state and finalize.

Top-level request flow

```
Client        Gateway (header/hash)      Handler (idempotency)           DB
|             |                          |                                |
|-- POST(K,P) |                          |                                |
|             | K=Idempotency-Key        |                                |
|             | P*=RFC8785(P); H=SHA256  |                                |
|             |---- reserve(scope,K,H) ------------------------------>   |
|             |<----------------------------- OK(reserved) / DUP  -------|
|             |                          |                                |
|             |                          |-- [A] DUP & FINALIZED & H  --->|  (replay)
|             |                          |-- [B] DUP & IN_PROGRESS & H  --|  (active?)
|             |                          |-- [C] DUP & H differs  --------|  (422)
|             |                          |-- [D] reserved  ---------------|  (execute)
|             |                          |-- [E] error paths  ------------|
(finalize?)
|             |                          |                                |
```

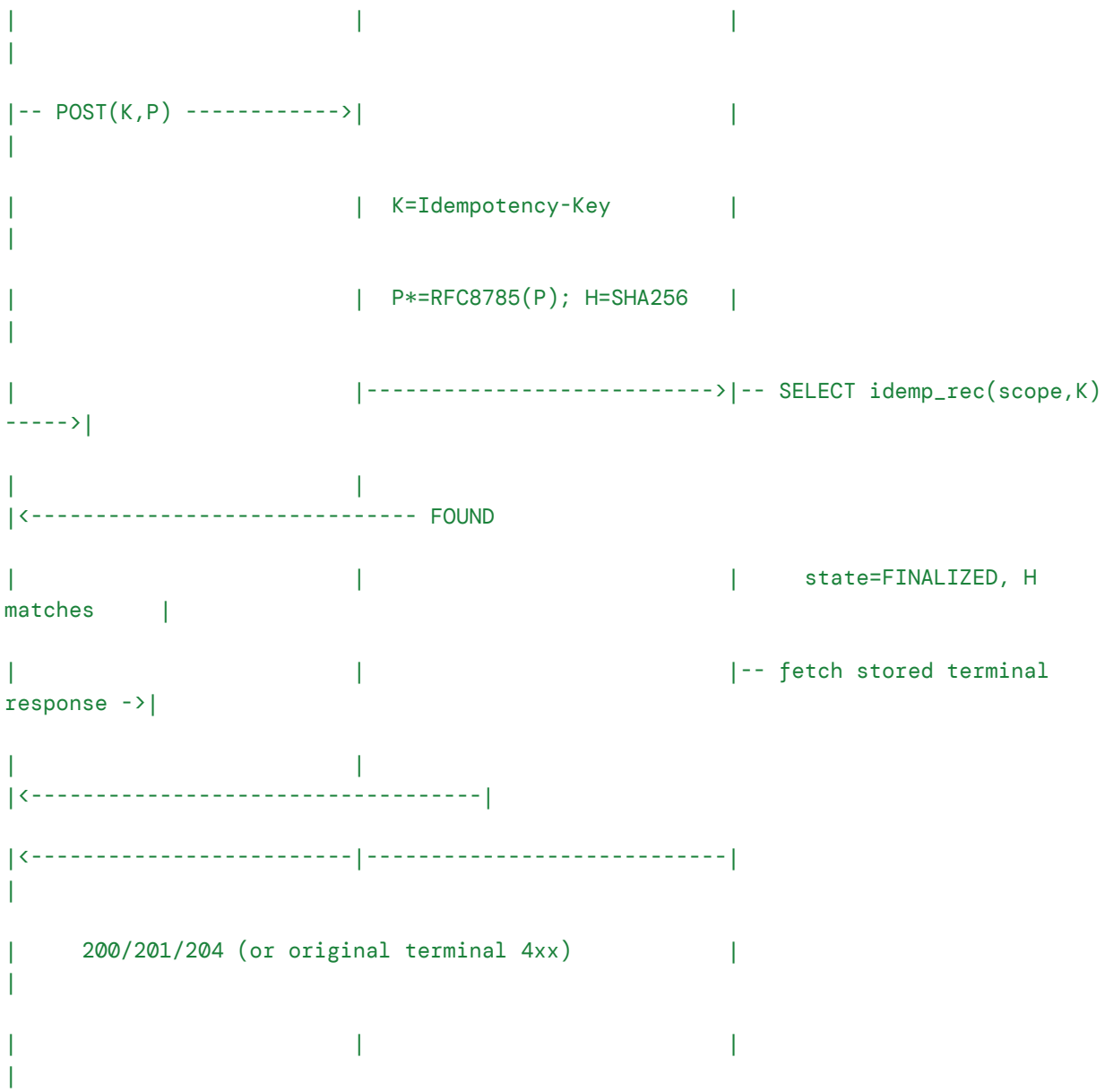
Legend

- **Gateway** = pre-matching request filter/interceptor (parse header, canonicalize JSON, compute hash).
- **Handler** = idempotency lookup/reserve/replay + mutation execution.
- **DB** = idempotency table + catalog state.

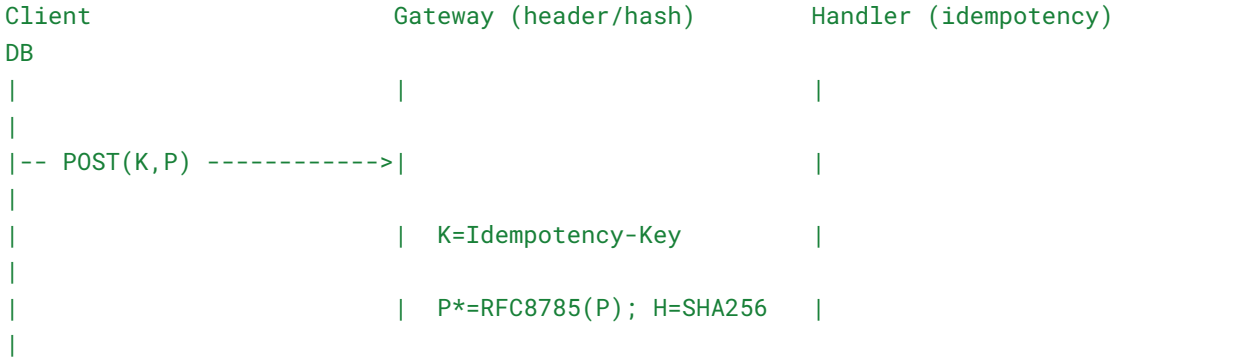
Zoom-ins (one per branch)

[A] DUP & FINALIZED & H -> replay

```
Client                    Gateway (header/hash)        Handler (idempotency)
DB
```



[B] DUP & IN_PROGRESS & H -> active?

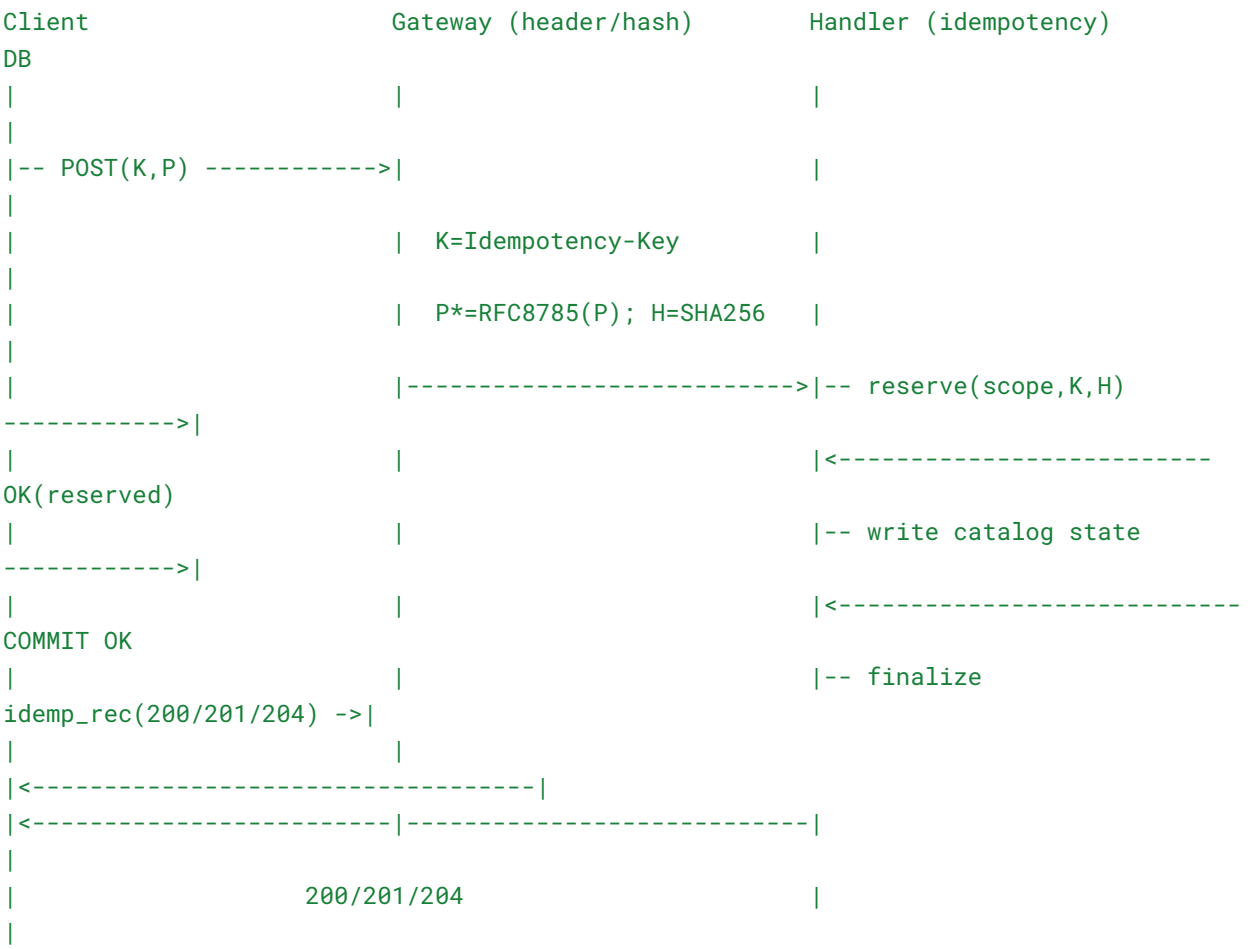


```
|                          |---------------------------->|-- SELECT idemp_rec(scope,K)
----->|
|                          |
|<------------------------------ FOUND
|                          |                             |     state=IN_PROGRESS, H
matches   |
|                          |                             |-- check "still active?"
---------->|
|                          |                             |<---------- YES
--------------------|
|<-------------------------|-----------------------------|
|
|            409 request_in_progress                     |
|
|                          |                             |
|
|                          |                             |-- check "still active?"
---------->|
|                          |                             |<---------- NO
----------------------|
|                          |                             |---> go to Reconciliation
(R1/R2)  |
```

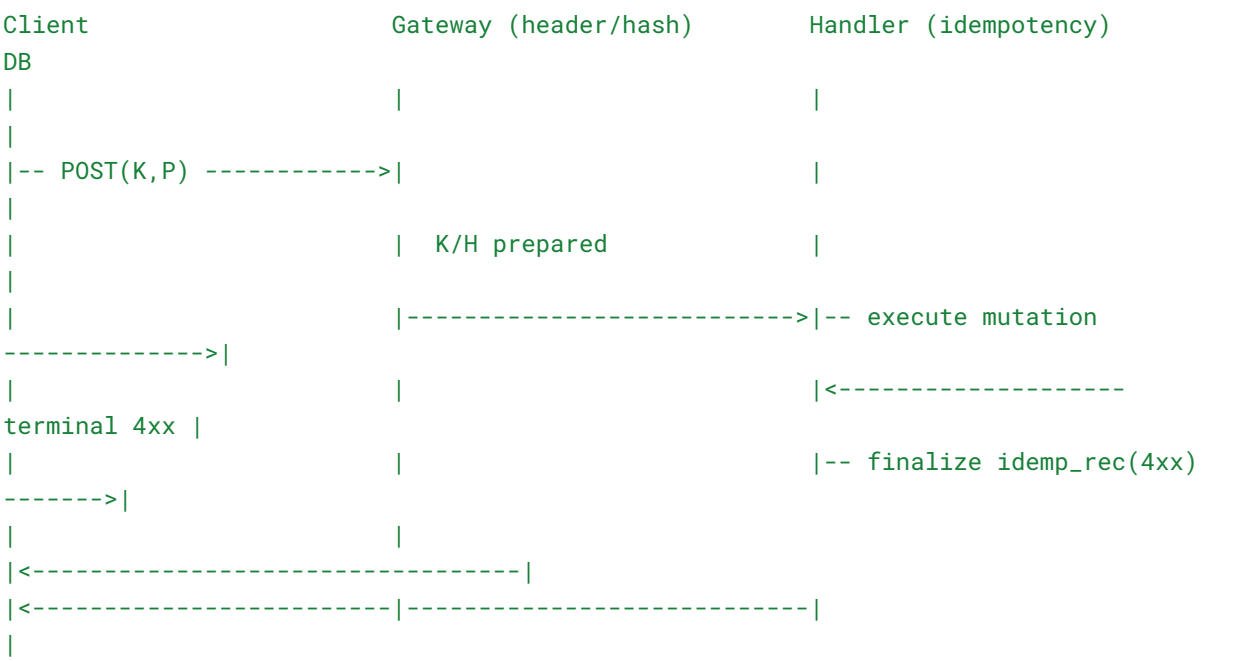
[C] DUP & H differs -> 422

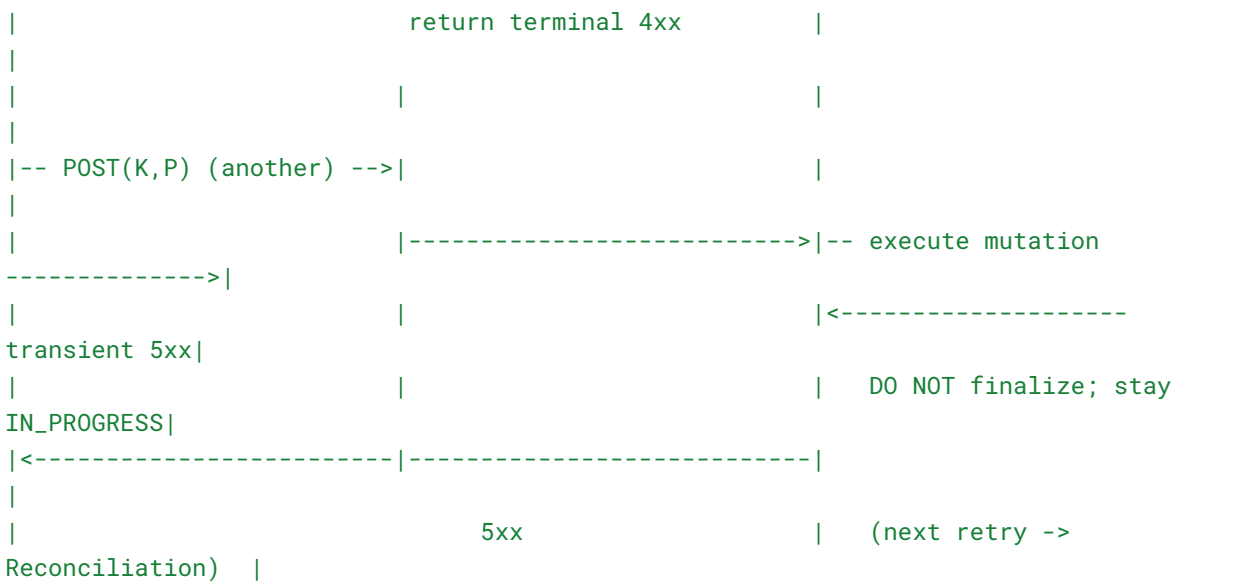
```
Client                     Gateway (header/hash)        Handler (idempotency)
DB
|                          |                             |
|
|-- POST(K,P) ------------>|                             |
|
|                          |  K=Idempotency-Key          |
|
|                          |  P*=RFC8785(P); H=SHA256    |
|
|                          |---------------------------->|-- SELECT idemp_rec(scope,K)
----->|
|                          |
|<------------------------------ FOUND
|                          |                             |     H differs from stored H
|
|<-------------------------|-----------------------------|
|
|         422 idempotency_key_conflict                   |
|
```

[D] Reserved -> execute + finalize



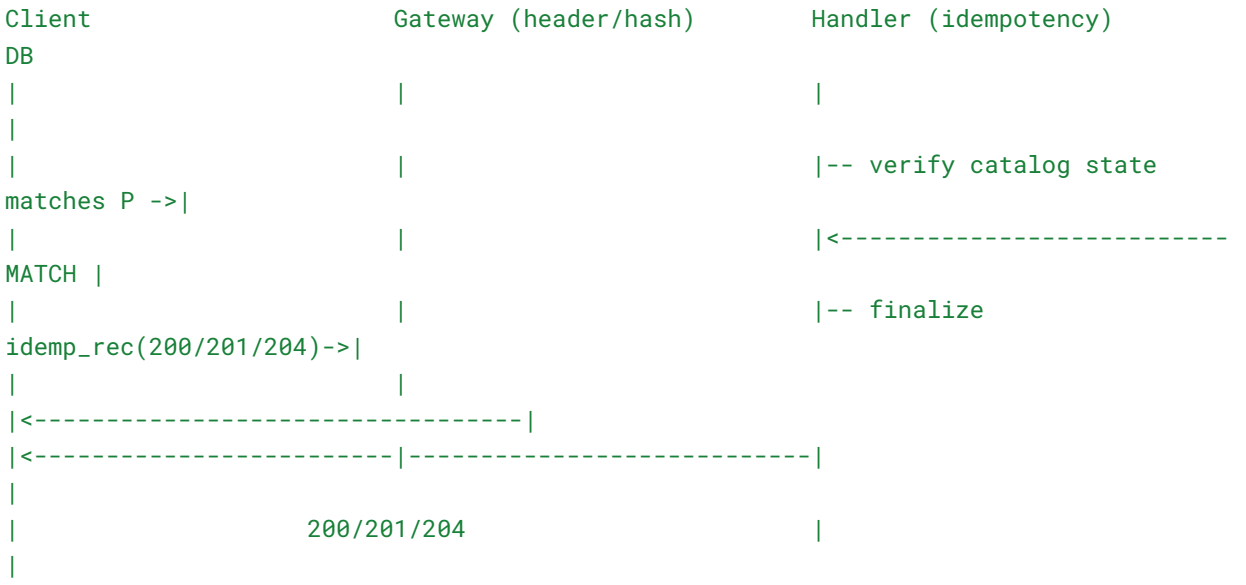
[E] Error paths (no transient replay)



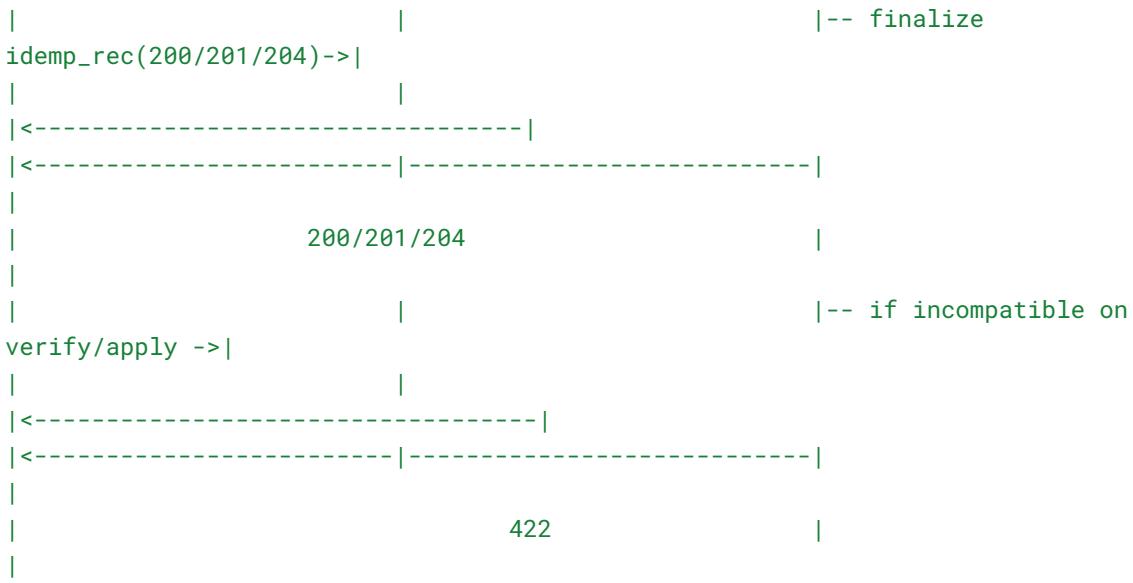


Reconciliation branches (when [B] says “inactive”)

R1. Finalize-gap (state already applied)



R2. Takeover (state not applied)



A Persistence Schema Example

```
CREATE TABLE idempotency_records (
  realm_id         VARCHAR   NOT NULL,
  method           VARCHAR   NOT NULL,
  resource_path    VARCHAR   NOT NULL,
  idempotency_key  VARCHAR   NOT NULL,
  payload_hash     CHAR(64)  NOT NULL,
  http_status      INTEGER,
  created_at       TIMESTAMP NOT NULL,
  updated_at       TIMESTAMP NOT NULL,
  expires_at       TIMESTAMP,
  PRIMARY KEY (realm_id, method, resource_path, idempotency_key)
);

CREATE INDEX idx_idemp_expires
  ON idempotency_records (expires_at);
```

Expiration & Cleanup

- TTL based on `idempotency-key-lifetime` + buffer; background job to purge expired rows.
- Configuration knobs: enable/disable, lifetime.

Endpoints in Scope

- `POST /v1/namespaces` (create), `DELETE /v1/namespaces/{ns}` (drop)
- `POST /v1/tables` (create), `POST /v1/tables/{name}/commit` (update/commit), `DELETE /v1/tables/{name}` (drop), `POST /v1/tables/{name}/rename` (rename)
- Property/metadata updates that mutate catalog state

(Non-mutating GET/HEAD routes are out of scope.)

Backward Compatibility

- **Default behavior (no header):** When the `Idempotency-Key` header is absent, servers behave exactly as today.
- **Client adoption:** Clients may adopt the header incrementally. Capability discovery prevents futile same-key retries where unsupported.

Testing Plan

- Discovery gating: enable/disable auto same-key retries based on `getConfig` fields.
- Duplicate handling: replay success (same key+payload), 422 on payload mismatch, 409 in-flight.
- Finalize-gap recovery: stale lease + state matches -> finalize + replay success.
- No 5xx caching: transient errors aren't stored.
- Lifetime/expiry: retries stop after advertised lifetime; expired records behave as unknown.
- Canonicalization: RFC 8785 cross-lang vectors yield identical hashes.

Full scenarios will be exercised in the Catalog Compatibility Test Kit and server integration tests.

Appendix A — OpenAPI Sketch

```
idempotency-key:
  name: Idempotency-Key
  in: header
  required: false
  schema:
    type: string
    pattern: '^[a-zA-Z0-9][a-zA-Z0-9_.-]*$'
    minLength: 1
    maxLength: 255
    example: "550e8400-e29b-41d4-a716-446655440000"
  description: |
    Optional client-provided idempotency key for safe request retries.
    When provided, the server guarantees at-most-once execution for requests with the same key. If a request with this key has already been processed
    successfully, the server returns the original result instead of reprocessing.
    Key Requirements:
    - Must be unique per client mutation operation (e.g., updateTable, createTable)
    - Should be generated randomly (e.g., UUID v4)
    - Scoped to operation type and resource path
    - Catalogs may expire keys according to the advertised token life time.
    Best Practices:
    - Use UUID.randomUUID() or equivalent
    - Reuse the same key for retries of the same logical operation
```

```
    - Generate new keys for new operations
```

Appendix B — Server Config Knobs (example)

```
polaris.idempotency.enabled=true
polaris.idempotency.lifetime=PT30M
polaris.idempotency.cleanup.enabled=true
```

Appendix C — Example Client Snippet (Java)

```
String key = UUID.randomUUID().toString();
Request req = Request.post(url)
    .header("Idempotency-Key", key)
    .bodyString(payloadJson, ContentType.APPLICATION_JSON);
Response resp = httpClient.execute(req);
```

Appendix D — Example with Retry

```
public static HttpResponse<String> postWithIdempotencyRetry(
    HttpClient client,
    URI url,
    String payloadJson,
    Duration lifetime,
    int maxAttempts
) throws Exception {

  String key = UUID.randomUUID().toString();
  Instant firstAttempt = Instant.now();

  for (int attempt = 1; attempt <= maxAttempts; attempt++) {
    HttpRequest request = HttpRequest.newBuilder(url)
        .header("Content-Type", "application/json")
        .header("Idempotency-Key", key)
        .POST(HttpRequest.BodyPublishers.ofString(payloadJson))
        .build();

    HttpResponse<String> response =
        client.send(request, HttpResponse.BodyHandlers.ofString());

    int code = response.statusCode();
```

```
    if (code == 200 || code == 201 || code == 204) {
      return response; // success or finalized replay
    }

    if (code == 409) { // request_in_progress
      Thread.sleep(200L);
      continue;
    }

    if (code == 422) { // idempotency_key_conflict
      throw new IllegalStateException(
          "Idempotency key conflict (422) — payload differs");
    }

    if (code >= 500
        && Duration.between(firstAttempt, Instant.now()).compareTo(lifetime)
<= 0) {
      continue; // retry with the SAME key
    }

    throw new RuntimeException(
        "Request failed: " + code + " body=" + response.body());
  }

  throw new RuntimeException("Max attempts exceeded or idempotency window
elapsed.");
}
```