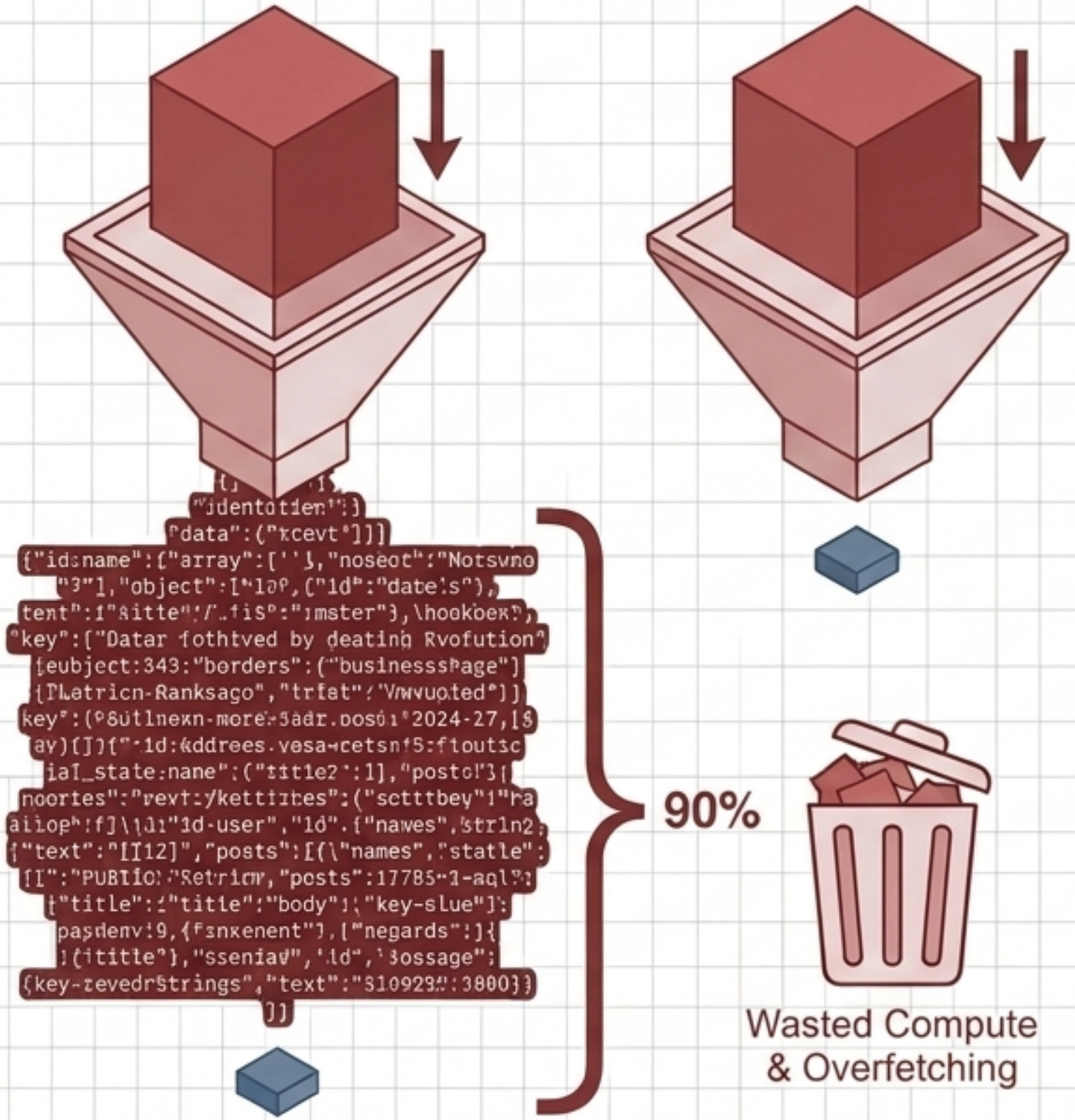
~/workday/extend/graph-api-masterclass.md

# Workday Graph API Architecture and Orchestration

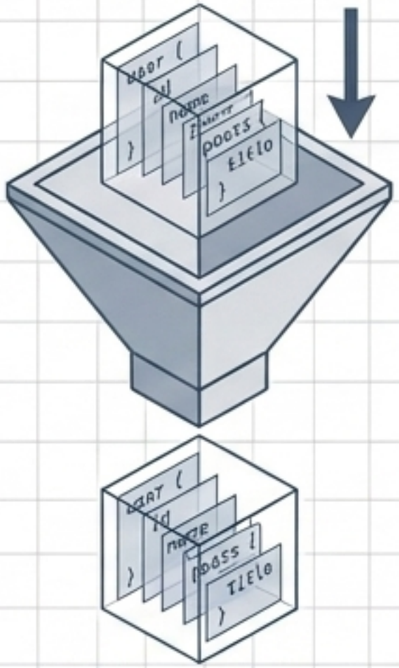
A technical reference guide for exact data fetching, serial mutations, and native orchestration.

# The Paradigm Shift: Resolving the Overfetching Problem

## REST / SOAP: The Data Dump

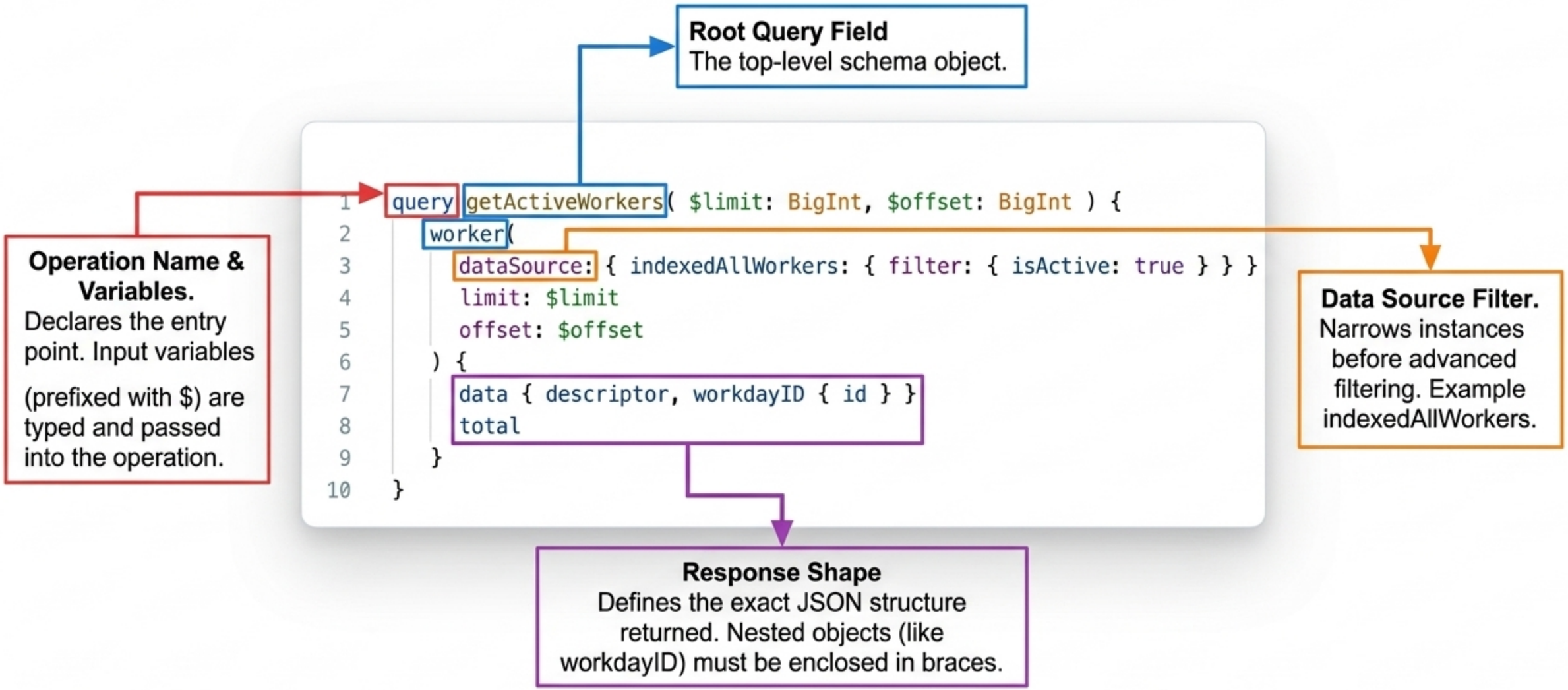



## Graph API: Exact Fetching



| Dimension | REST / SOAP | Graph API |
|---|---|---|
| Endpoints | Multiple (REST) / Single (SOAP) | Single Unified Endpoint |
| Data Fetching | Heavy payloads, over/under-fetching | Exact required shape |
| Operations | Multiple (GET, POST, PUT, DELETE) | Query (GET data), Mutation (POST/PATCH data) |

# Reading Data: Query Anatomy



# Advanced Filtering: The **Where** Argument

**where**: Criteria Object

Implicit AND

```
query getVehicles {
  vehicles(where: {
    licensePlate: { contains: "123" },
    regFee: { gt: 20.50 },
}
```

AND

**Constraint:** It vended the Explicit ctrickst.

Explicit OR / NOT

```
vehicles(where: {
  OR: [
    { model: { in: ["WRX", "Legacy"] },
    { color: { neq: "blue" } }
}
```

OR

## Full Query Example:

**Operation Name:**
Declares the entry point.

```
query getVehicles {
  vehicles(where: {
    licensePlate: { contains: "123" },
    regFee: { gt: 20.50 },
    OR: [
      { model: { in: ["WRX", "Legacy"] } },
      { color: { neq: "blue" } }
    ]
  }) {
    id
    model
    color
    licensePlate
    regFee
  }
}
```

**Root Query Field with 'where' Argument:** Combines implicit AND and explicit OR criteria.

**Criteria Object:**
Defines filtering logic.

**Response Shape:** Defines the exact JSON structure returned.

# Handling Large Datasets: Pagination Mechanics

## Step 1: Cache Generation

Initial execution builds a cache of the query results.

## Step 2: The Sliding Window


## Step 3: Pagination Progression

| | |
|---|---|
| Page 1: | `{ "limit": 50, "offset": 0 }` |
| Page 2: | `{ "limit": 50, "offset": 50 }` |
| Page 3: | `{ "limit": 50, "offset": 100 }` |

**Takeaway**

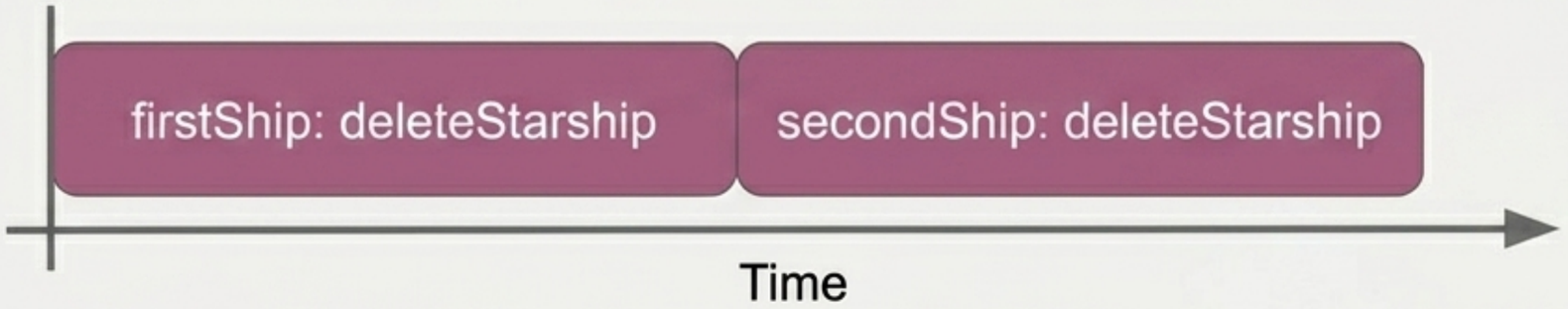
The **total** response field returns the complete dataset size, providing the ceiling for your offset iterations.

# Architectural Execution: Queries vs. Mutations

## Queries: Parallel Execution


Query fields execute in parallel, maximizing retrieval speed.

## Mutations: Serial Execution




Top-level mutation fields run in series. The first is guaranteed to finish before the second begins, ensuring no self-induced race conditions.

# Writing Data: Mutation Anatomy

```
mutation updateReq (
  $update_input: EditRequistion_Input!,
  $reqId: IdentifierInput!
) {
  updateProcurementRequisition (
    input: $update_input,
    requisitionId: $reqId
  ) {
      descriptor
      workdayID { id }
  }
}
```

**The Input Object.** Passes structured JSON containing the specific fields to modify rather than individual scalar values.

**Target Identification.** Requires an object with id and type (e.g., {"id": "d5f07...", "type": "WID"}).

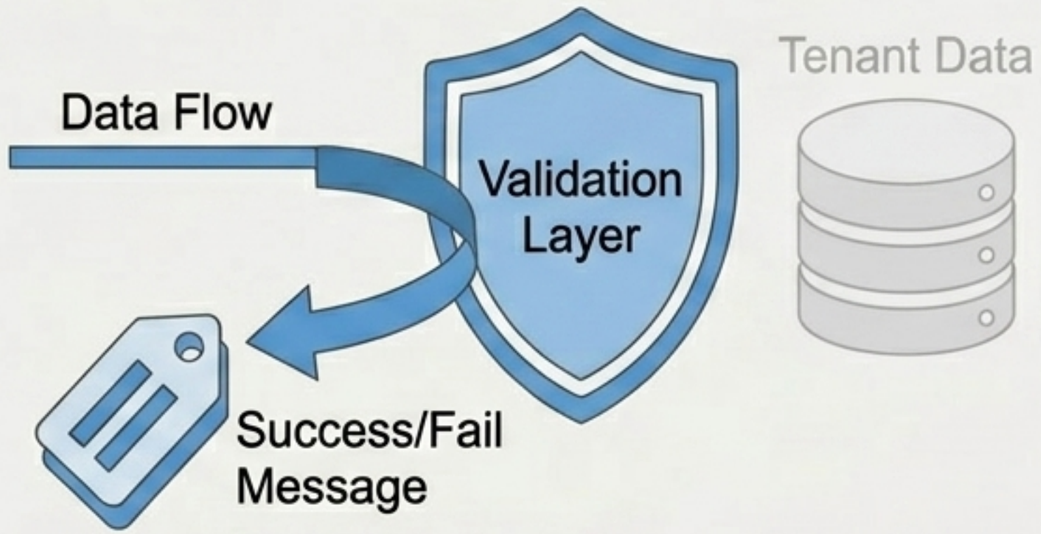
**Root Mutation Field.** Named specifically for the action (create, update, cancel, submit).

**Side-Effect Return.** Mutations return the exact shape of the updated data requested, confirming the write operation succeeded.

# Advanced Mutation Controls: HTTP Headers

## Panel 1: The Validation Sandbox

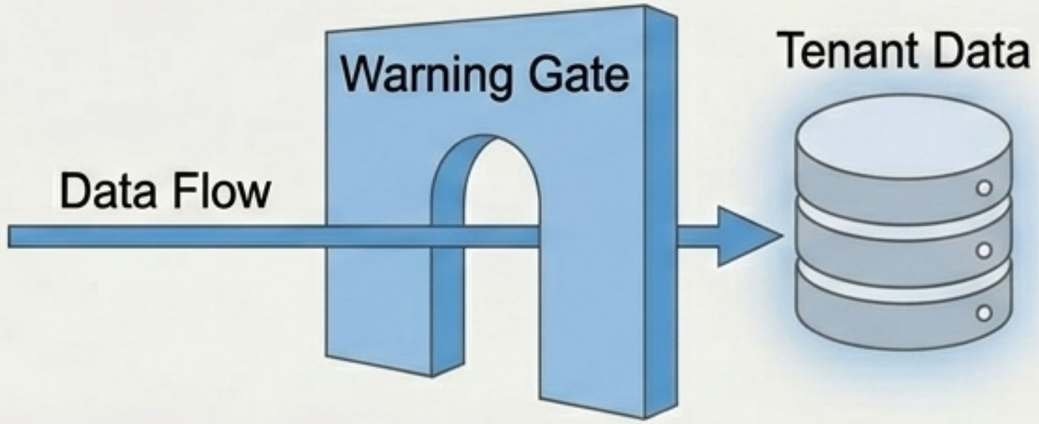
`x-validate-only: 1`




Runs full validation logic but prevents data from being saved. Returns isSuccess boolean in the extensions object. Defaults to 0 if omitted.

## Panel 2: Warning Override

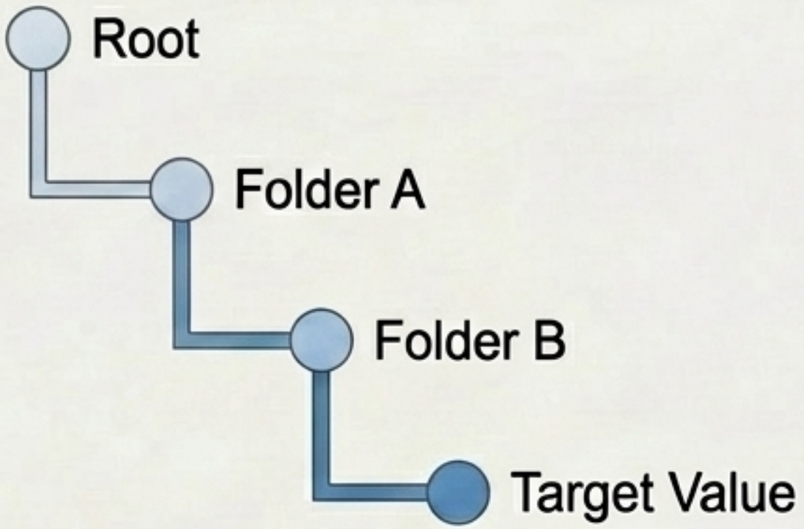
`wd-warning-action: updateonwarning`




By default, validation warnings block mutation saves. This header allows business logic to bypass non-critical warnings while still enforcing strict blocks on critical errors.

# Reading Data: Finding Workday IDs

## Method 1: Directory Drilling (`promptPath`)




Passes an array of IDs to iteratively step down through nested categories to reach leaf-node values.

## Method 2: Flat Search (`search`)


```
query {
  companyReference(search: "gms ireland") {
    id } }
```

Performs a string search across descriptors at all levels.

**Constraint Note:** If both arguments are provided, Graph API ignores promptPath and prioritizes search.
**Clarification Note:** This slide describes read operations and belongs in the 'Reading Data' section of this presenta

# Advanced Capabilities: Handling Attachments

**Step 1: Multipart Upload**

Include a field of File type (e.g., attachmentContent) in the mutation input via multipart/form-data. Graph API virus-scans and saves the file, returning the parent object's Workday ID.

**Step 2: Retrieve Download ID**

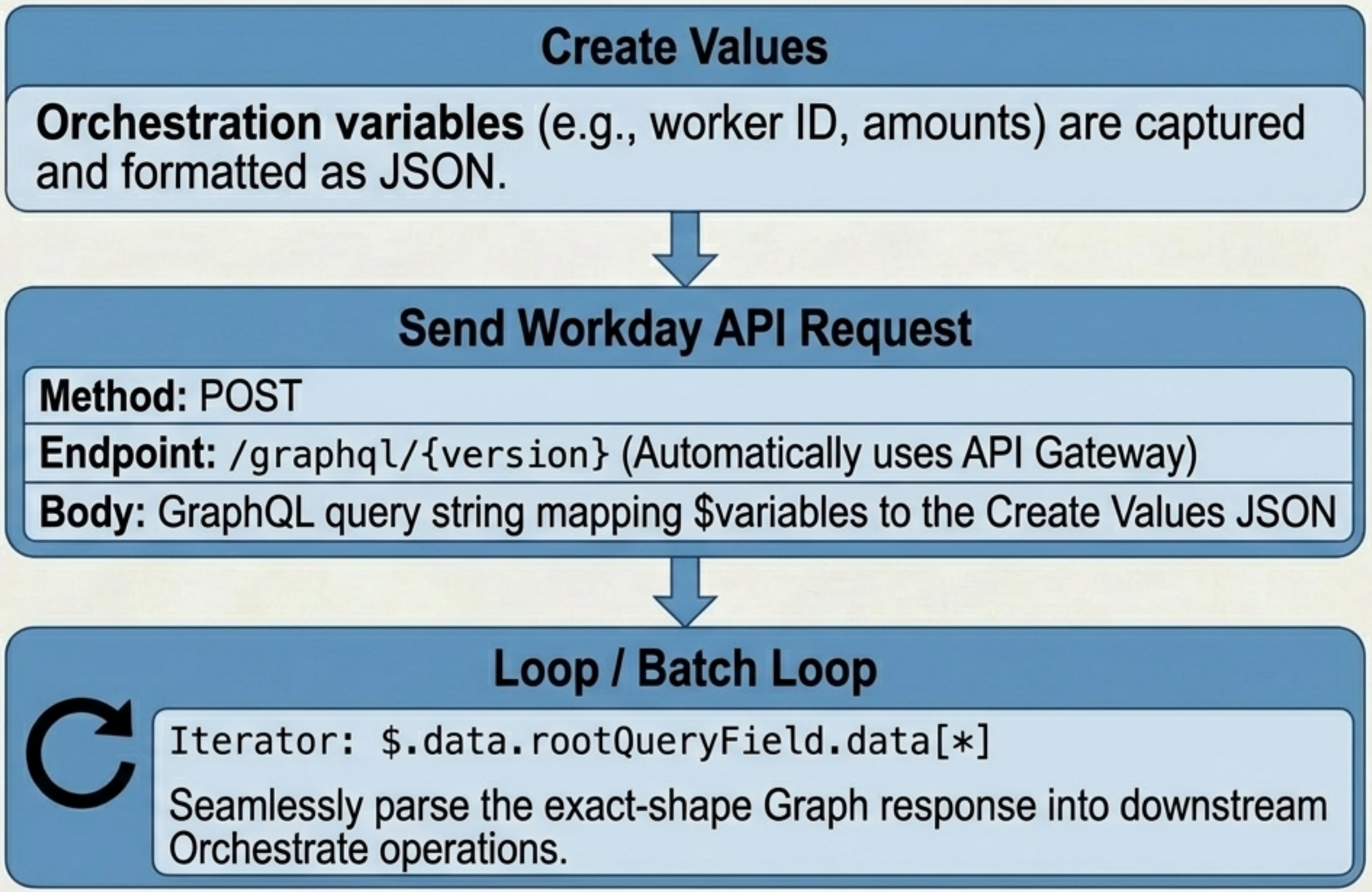
Query the parent object using the Workday ID to retrieve the downloadID key.

Crucial constraint: This ID has a strict 1-hour expiration.

**Step 3: REST API Download**

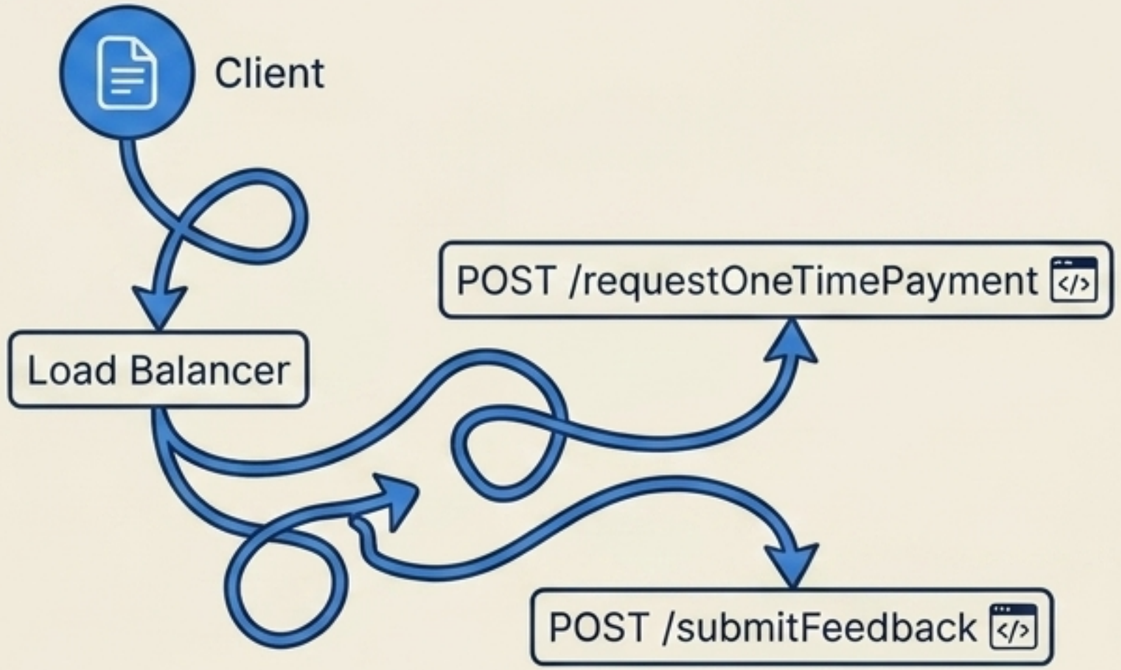
Pass the downloadID to the Attachments REST API via a GET request to retrieve the binary payload: https://api.workday.com/attachments/v1/graphql/{downloadID}

# Integration Workflow: Orchestrate & Graph API



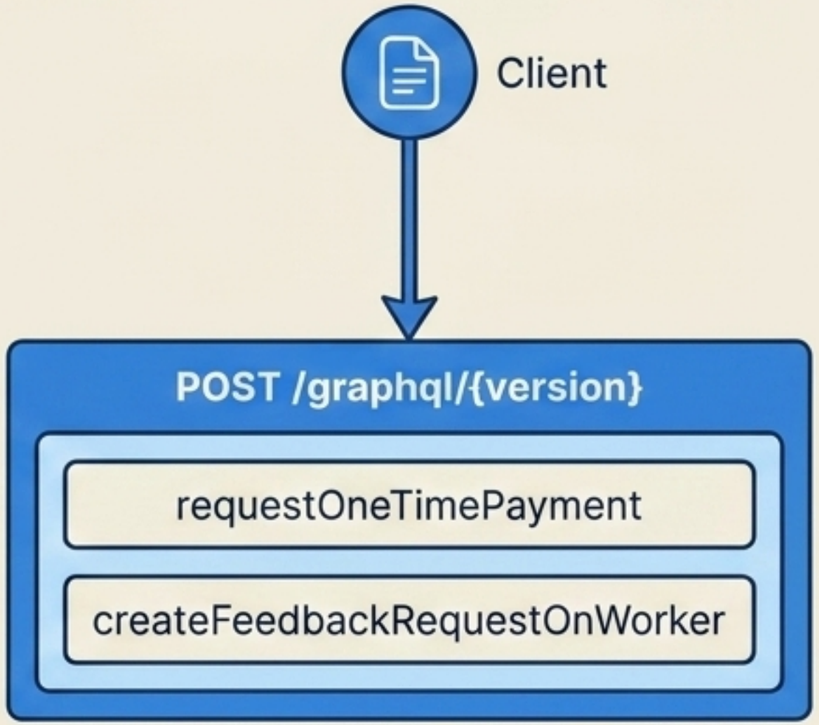
# Synthesis: Consolidation Power

## Before: REST API




**Drawback:** Two separate network trips, heavier orchestration logic, and a higher ri risk of partial failure.

## After: Graph API




**Advantage:** One network trip, unified error handling, and streamlined orchestration logic executing in perfect series.

# Masterclass Synthesis: Best Practices & Cheat Sheet

## Optimization Rules

- **Control Depth:**
  Deeply nested queries increase latency. Fetch only the exact needed fields.
- **Filter Order:**
  Always apply a Data Source Filter (dataSource) to narrow instances before applying advanced where criteria logic.
- **Parallel vs Serial:**
  Group independent queries in one request for parallel speed; group writes in one mutation for serial safety.

## Error Handling & Security

- **Beware the 200 OK:**
  Graph API returns HTTP 200 even if the query fails. Always parse the errors[] array in the response body.
- **Security Alignment:**
  A user needs the 'Workday Graph API Applications' security domain PLUS the specific domain securing the requested business object (e.g., 'Worker Data').