Duplicate Email Cleanup — OpenSearch & Neptune (Keep Latest, Nullify Others)

Owner: Data Engineering

Targets: OpenSearch indices (contacts/users/leads), Neptune vertices (Person/Contact)

Outcome: Exactly **one** record per email retains the email; all older duplicates have the email **removed/nullified**.

Pre-requisites

1. **Fields**
   - OpenSearch: email (keyword), last_updated (date).
   - Neptune: vertex label (e.g., Person), properties: email (String), lastModified (ISO 8601 or epoch).
2. **AWS Access & Network**
   - Glue / SageMaker in the same **VPC/Subnets/SGs** as OpenSearch & Neptune (if VPC-only).
   - IAM roles with least-privileged access to **S3**, **CloudWatch**, **OpenSearch**, **Neptune**.
3. **S3 Buckets**
   - s3://data-cleanup-reports/ (JSON run reports).
   - Optional: s3://data-cleanup-staging/ (dry-run exports).
4. **Python Env**
   - Use the earlier requirements.txt (opensearch-py, gremlinpython, boto3, pandas, etc.).
5. **Repo Layout**

```
infra/data_cleanup/
  config/
    cleanup_opensearch_email.json
    cleanup_neptune_email.json
  scripts/
    os_email_dedupe.py
    neptune_email_dedupe_sagemaker.py
    neptune_email_dedupe_glue.py
  README.md
  requirements.txt
```

---

1) Configuration

1.1 OpenSearch

config/cleanup_opensearch_email.json

```
{
  "host": "search-prod-opensearch.xxxxx.us-east-1.es.amazonaws.com",
  "port": 443,
  "index": "contacts",
  "auth_type": "basic",
  "username": "admin",
  "password": "REDACTED",
  "email_field": "email",
  "timestamp_field": "last_updated",
  "batch_size": 10000,
  "scroll": "2m",
  "report_bucket": "data-cleanup-reports",
  "report_prefix": "opensearch_email/",
  "dry_run": false
}
```

1.2 Neptune

config/cleanup_neptune_email.json

```
{
  "endpoint_wss":
"wss://my-neptune.cluster-xyz.us-east-1.neptune.amazonaws.com:8182/greml
in",
  "label": "Person",
  "email_prop": "email",
  "timestamp_prop": "lastModified",
  "page_limit": 5000,
  "batch_size": 500,
  "report_bucket": "data-cleanup-reports",
  "report_prefix": "neptune_email/",
  "dry_run": false
}
```

---

2) OpenSearch — Keep Latest Email, Nullify Older Duplicates

Strategy:

1. Pull minimal fields (_id, email, last_updated) via scroll in batches.
2. Compute, per email, the **latest** document by last_updated.
3. For all other docs of the same email, **set email to null**.
4. Log counts to S3.

> For very large indices, consider **Composite Aggregations + top_hits**. The scroll approach below is simpler and works well when you can batch.

scripts/os_email_dedupe.py

```
import os, json, boto3
from datetime import datetime
from collections import defaultdict

import pandas as pd
from opensearchpy import OpenSearch, helpers

CFG = json.load(open('config/cleanup_opensearch_email.json'))

client = OpenSearch(
    hosts=[{'host': CFG['host'], 'port': CFG['port']}],
    http_auth=(CFG['username'], CFG['password']),
    use_ssl=True, verify_certs=True
)

INDEX = CFG['index']
EMAIL = CFG['email_field']
TS    = CFG['timestamp_field']
SCROLL = CFG['scroll']
BATCH = CFG['batch_size']
DRY = CFG.get('dry_run', False)

q = {
  "size": BATCH,
  "_source": [EMAIL, TS],
  "query": {"exists": {"field": EMAIL}},
  "sort": [{TS: {"order": "desc"}}]
}

resp = client.search(index=INDEX, body=q, scroll=SCROLL)
scroll_id = resp.get('_scroll_id')

all_rows = []
while True:
    hits = resp['hits']['hits']
```

```
    if not hits: break
    for h in hits:
        _id = h['_id']
        src = h.get('_source', {})
        all_rows.append({
            "_id": _id,
            EMAIL: src.get(EMAIL),
            TS: src.get(TS)
        })
    resp = client.scroll(scroll_id=scroll_id, scroll=SCROLL)
    scroll_id = resp.get('_scroll_id')

df = pd.DataFrame(all_rows)
df = df.dropna(subset=[EMAIL])

# keep latest per email (we already sorted desc by TS in query, but
ensure with sort)
df_sorted = df.sort_values(TS, ascending=False)
keepers = df_sorted.drop_duplicates(EMAIL, keep='first')        # one
per email
keep_ids = set(keepers["_id"].tolist())

dupes = df_sorted[df_sorted[EMAIL].duplicated(keep='first')]     # all
but first per email
to_nullify = dupes[~dupes["_id"].isin(keepers["_id"])]

updated = 0
if not DRY:
    # Bulk update: set email to null for duplicate docs
    actions = []
    for _id in to_nullify["_id"].tolist():
        actions.append({
            "_op_type": "update",
            "_index": INDEX,
            "_id": _id,
            "script": {
                "source": f"ctx._source.{EMAIL} = null"
            }
        })
        if len(actions) == 1000:
            helpers.bulk(client, actions, request_timeout=120)
            updated += len(actions)
            actions = []
    if actions:
        helpers.bulk(client, actions, request_timeout=120)
        updated += len(actions)
```

```
report = {
  "system": "OpenSearch",
  "index": INDEX,
  "unique_emails": int(keepers.shape[0]),
  "duplicates_found": int(df.shape[0] - keepers.shape[0]),
  "nullified": int(updated if not DRY else to_nullify.shape[0]),
  "retained": int(keepers.shape[0]),
  "dry_run": DRY,
  "ts": datetime.utcnow().isoformat()
}

s3 = boto3.client('s3')
key =
f"{CFG['report_prefix']}email_dedupe_{INDEX}_{datetime.utcnow().strftime
('%Y%m%d_%H%M%S')}.json"
s3.put_object(Bucket=CFG['report_bucket'], Key=key,
Body=json.dumps(report))
print("Report:", f"s3://{CFG['report_bucket']}/{key}")
```

Runbook (Glue/SageMaker/Local):

```
pip install -r requirements.txt
python scripts/os_email_dedupe.py
```

Validate (OpenSearch Dev Tools):

```
GET contacts/_search
{
  "size": 0,
  "aggs": {
    "dupe_emails": {
      "terms": { "field": "email.keyword", "min_doc_count": 2, "size":
100 },
      "aggs": { "docs": { "top_hits": { "size": 10, "_source":
["email","last_updated"] } } }
    }
  },
  "query": { "exists": { "field": "email" } }
}
```

---

3) Neptune — Keep Latest Email, Remove Email on Older Duplicates

Strategy:

Use Gremlin to group vertices by email, determine the latest by lastModified, and **remove** the email property on all older vertices.

Note: Instead of property('email', null), use **properties('email').drop()** (works reliably in Neptune).

3.1 SageMaker (interactive)

scripts/neptune_email_dedupe_sagemaker.py

```
import json, boto3, time
from datetime import datetime
from gremlin_python.driver import client, serializer

CFG = json.load(open('config/cleanup_neptune_email.json'))
DRY = CFG.get('dry_run', False)

g = client.Client(
    CFG['endpoint_wss'],
    'g',
    username="", password="",
    message_serializer=serializer.GraphSONSerializersV3d0()
)

LABEL = CFG['label']
EMAIL = CFG['email_prop']
TS    = CFG['timestamp_prop']
BATCH = CFG['batch_size']

# Pull candidates (id, email, lastModified)
# ValueMap(true) returns id and properties; unwrap to simple rows
q = f"g.V().hasLabel('{LABEL}').has('{EMAIL}').valueMap(true)"
rows = g.submit(q).all().result()

def unwrap(row):
    out = {"id": row["id"]}
    for k, v in row.items():
        if k == "id": continue
        out[k] = v[0] if isinstance(v, list) and v else v
    return out

records = [unwrap(r) for r in rows]
# Build map: email -> list of (id, ts)
by_email = {}
for r in records:
```

```
    e = r.get(EMAIL)
    ts = r.get(TS)
    if e is None or ts is None: continue
    by_email.setdefault(e, []).append((r["id"], ts))

# For each email: sort desc by TS, keep first, others drop email prop
to_clean = []
for e, lst in by_email.items():
    lst_sorted = sorted(lst, key=lambda x: x[1], reverse=True)
    keep = lst_sorted[0][0]
    for vid, _ in lst_sorted[1:]:
        to_clean.append(vid)

updated = 0
if not DRY and to_clean:
    for i in range(0, len(to_clean), BATCH):
        chunk = to_clean[i:i+BATCH]
        dq = "g.V(" + ",".join([f"'{vid}'" for vid in chunk]) +
").properties('" + EMAIL + "').drop()"
        g.submit(dq).all().result()
        updated += len(chunk)
        time.sleep(0.2)

report = {
  "system": "Neptune",
  "label": LABEL,
  "duplicates_found": int(sum(max(0, len(v)-1) for v in
by_email.values())),
  "nullified": int(updated if not DRY else len(to_clean)),
  "retained": int(len(by_email)),
  "dry_run": DRY,
  "ts": datetime.utcnow().isoformat()
}
boto3.client('s3').put_object(
  Bucket=CFG['report_bucket'],

Key=f"{CFG['report_prefix']}email_dedupe_{datetime.utcnow().strftime('%Y
%m%d_%H%M%S')}.json",
  Body=json.dumps(report)
)
print("Done. Report written.")
```

3.2 Glue (scheduled)

scripts/neptune_email_dedupe_glue.py

```
import json, boto3, time
from datetime import datetime
from gremlin_python.driver import client, serializer

CFG = json.load(open('config/cleanup_neptune_email.json'))
DRY = CFG.get('dry_run', False)

g = client.Client(
    CFG['endpoint_wss'],
    'g',
    username="", password="",
    message_serializer=serializer.GraphSONSerializersV3d0()
)

LABEL = CFG['label']; EMAIL = CFG['email_prop']; TS =
CFG['timestamp_prop']
PAGE = CFG['page_limit']; BATCH = CFG['batch_size']

# Page through vertices to avoid huge result sets
offset = 0
records = []
while True:
    q =
f"g.V().hasLabel('{LABEL}').has('{EMAIL}').range({offset},{offset+PAGE})
.valueMap(true)"
    page = g.submit(q).all().result()
    if not page: break
    for r in page:
        rec = {"id": r["id"]}
        for k, v in r.items():
            if k == "id": continue
            rec[k] = v[0] if isinstance(v, list) and v else v
        records.append(rec)
    offset += PAGE

by_email = {}
for r in records:
    e = r.get(EMAIL); ts = r.get(TS)
    if e is None or ts is None: continue
    by_email.setdefault(e, []).append((r["id"], ts))

to_clean = []
for e, lst in by_email.items():
    lst_sorted = sorted(lst, key=lambda x: x[1], reverse=True)
    for vid, _ in lst_sorted[1:]:
        to_clean.append(vid)
```

```
updated = 0
if not DRY and to_clean:
    for i in range(0, len(to_clean), BATCH):
        chunk = to_clean[i:i+BATCH]
        dq = "g.V(" + ",".join([f\"'{vid}'\" for vid in chunk]) +
").properties('" + EMAIL + "').drop()"
        g.submit(dq).all().result()
        updated += len(chunk)
        time.sleep(0.2)

report = {
  "system": "Neptune",
  "label": LABEL,
  "duplicates_found": int(sum(max(0, len(v)-1) for v in
by_email.values())),
  "nullified": int(updated if not DRY else len(to_clean)),
  "retained": int(len(by_email)),
  "dry_run": DRY,
  "ts": datetime.utcnow().isoformat()
}
boto3.client('s3').put_object(
  Bucket=CFG['report_bucket'],

Key=f"{CFG['report_prefix']}email_dedupe_{datetime.utcnow().strftime('%Y
%m%d_%H%M%S')}.json",
  Body=json.dumps(report)
)
print("Report stored.")
```

Validate (Gremlin):

// Find any remaining duplicate emails (count > 1)

```
g.V().has('email').groupCount().by('email').unfold().where(select(values
).is(gt(1))).limit(20)
```

// Spot check a specific email's vertices ordered by lastModified desc

```
g.V().has('email','someone@example.com').order().by('lastModified',
decr).valueMap(true)
```

---

4) Safety, Idempotency, Rollback

- **Dry Runs:** Set "dry_run": true in configs to compute targets without updates.
- **Atomicity:** Updates are per-doc/node; scripts can be re-run safely.

- **OpenSearch Snapshots:** (Optional) take a snapshot before first production run.
- **Neptune Snapshots:** Create a DB cluster snapshot before first production run.
- **Rollback:**
    - OpenSearch → restore snapshot to new index, swap alias.
    - Neptune → restore from snapshot (blue/green or maintenance window).

---

5) Scheduling

- **OpenSearch**: Glue Python Shell weekly via EventBridge (`cron(0 21 ? * SUN *)`).
- **Neptune**: Glue Python Shell monthly via EventBridge (`cron(0 21 1 * ? *)`).
- **SageMaker**: On-demand runs for review/spot fixes.

---

6) Post-Run Checks

- **OpenSearch**
    - Verify no email.keyword has doc_count > 1 (terms agg).
    - Sample search for a previously duplicated email; expect one record with email, others with email=null.
- **Neptune**
    - groupCount().by('email') has no value > 1.
    - For a known duplicated email, only **one** vertex retains it; others have no email property.