Streamlining Data Migration with In-Situ Document De-Duplication

TL;DR: *We conducted a PoC to address document duplication in a client's CRM. Using Oracle PL/SQL, we identified and de-duplicated PDF files stored as BLOBs, reducing storage needs by over 90%. This approach ensures data integrity, optimizes document management, and simplifies the transition to SharePoint.*

In this post, we explore a real-world data migration challenge encountered during a client's transition from an on-premises CRM to a cloud-based solution. A critical issue was the presence of nearly 1TB of documents, far exceeding the expected volume due to widespread duplication. To address this, we implemented an in-situ de-duplication strategy using Oracle SQL. This approach allowed us to identify exact duplicates within the database, mark them efficiently, and prepare the data for migration to SharePoint. Although we briefly reference a Python-based near-duplicate detection solution, this post focuses exclusively on the Oracle SQL solution, with further details to be provided in a subsequent post.

Introduction

Background

Anante was engaged by a client to oversee a data migration project as they transitioned from an on-premises CRM to the latest cloud-based CRM from the same vendor. The migration itself was relatively straightforward, with schema changes well understood, aside from some user-customized data entities that required additional mapping and validation.

However, one major challenge emerged: document and media file storage. The on-premises database exceeded 1TB, but estimates, based on their document masters, suggested that only around 10GB of documents/media should have existed. Upon further investigation, we found that users had attached media files to individual recipient records, leading to massive duplication of identical documents across thousands of records. That's a lot of BLOBs.

Given that the cloud-based CRM's storage and ingress/egress costs are significantly higher than SharePoint, we advised the client to offload document storage to SharePoint, linking records in the CRM to the relevant documents. However, before migration, de-duplication was essential to avoid unnecessarily storing and managing redundant documents.

Scope of the Duplication Problem

An analysis of the stored media files revealed the following distribution of file types:

File Type	Percentage
PDF Documents	**90%**
Microsoft Publisher	**4%**
Microsoft Word	**3%**
Microsoft PowerPoint	**2%**
Video/Audio Files	**1%**

Since PDFs represented 90% of the stored documents, they were identified as the first target for de-duplication. The proof of concept (PoC) discussed here focuses exclusively on PDF files, specifically on their text content. The next phase of analysis (covered in a future post) will extend this approach to image-based content within PDFs, ensuring that marketing materials with identical text but different visuals are not mistakenly consolidated.

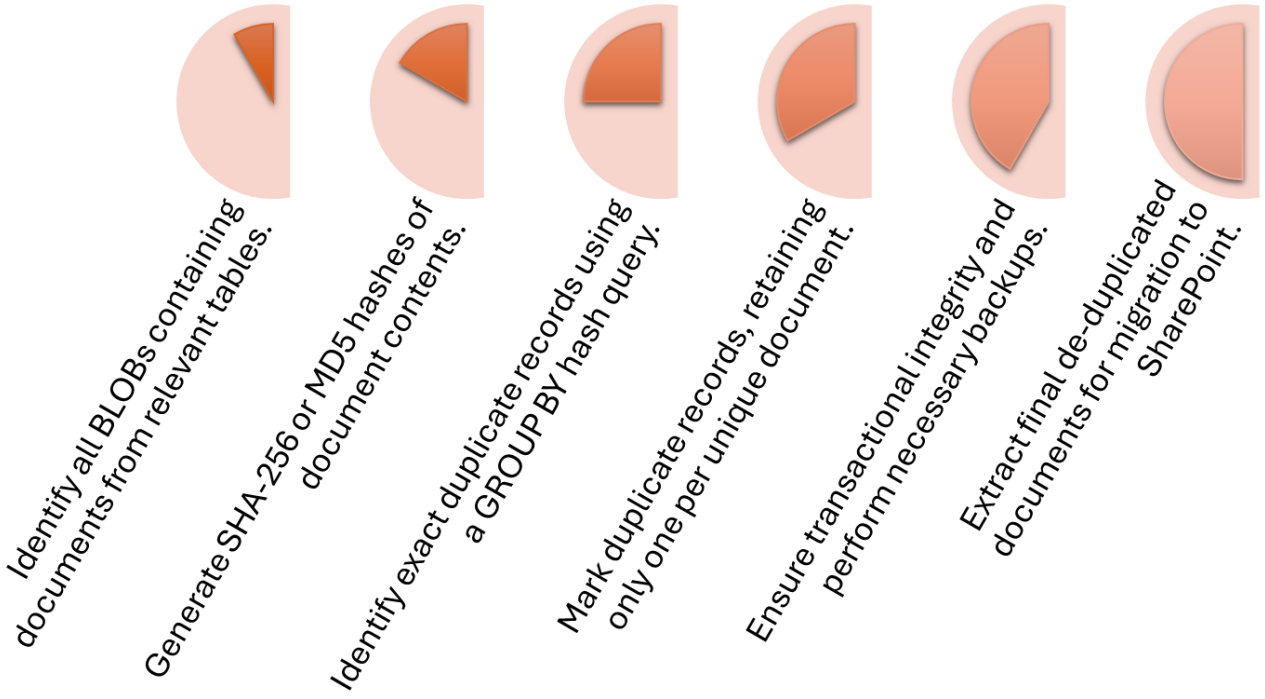
Objectives

1. Identify and remove exact duplicate documents, ensuring that only one copy is retained and linked to all relevant CRM records.
2. Detect near-duplicate documents, where text is largely identical except for personalized details (e.g., recipient names, dates, or tracking IDs).
3. Develop an efficient, scalable processing pipeline for handling thousands of documents rapidly.
4. Evaluate two approaches to de-duplication:
    a. In-situ: A simple approach performing de-duping directly within the database before extraction focussing only on exact duplicates.
    b. Externally: After extraction of all media, a Python-based text analysis approach leveraging multiple fuzzy matching techniques to not only handle exact duplicates but also near-duplicates.
5. Provide statistical reporting of the findings.
6. Determine viability of reverse-engineering templates from personalised documents.

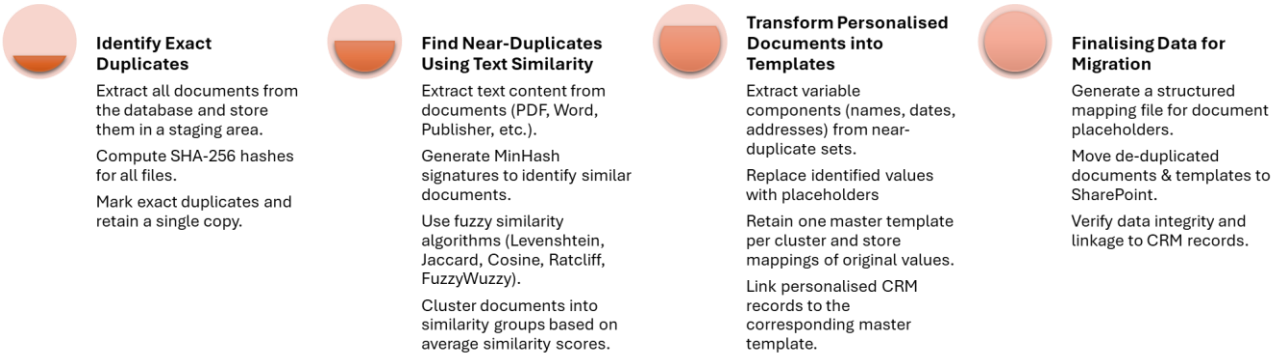
Methodology

Now let's break down the process for both approaches:

The following schematic shows the process for in-situ de-duplication (SQL-based within the database):



And next, the external de-duplication (Python-based with similarity checks and transformation back to templates where possible):



We'll explore the external, more complex approach in a subsequent post. For now, let's get into the depths of the Oracle PoC solution.

Implementation

Let's explore some of the key parts of the process for de-duplicating directly within the database.

Identify Tables with Binary Objects

The first job is to identify all BLOB/CLOB columns in the schema. These are Binary Large OBjects (we are not interested in CLOBs in this project as they are Character Large OBjects

typically storing XML, JSON etc). Luckily, Oracle helpfully maintains a data dictionary view called all_lobs:

Create a View of BLOBs

The output from the "all_lobs" query will tell us which our user's tables have BLOBs. In the example code below from our test environment, we have found 2 tables: "donors" and "marketing_campaigns":

```
CREATE OR REPLACE VIEW documents AS
SELECT
    d.id AS doc_id,
    'donors' AS source_table,
    ROWID AS source_rowid,
    d.doc_blob
FROM donors d
UNION ALL
SELECT
    m.file_id AS doc_id,
    'marketing_campaigns' AS source_table,
    ROWID AS source_rowid,
    m.file_blob
FROM marketing_campaigns m
```

Create a Metadata Control Table

We have created a standard view as a static query on our underlying tables. If we wanted to allow updates to the view that propagate changes to the underlying tables, we would need to create an INSTEAD OF trigger on the view to handle those updates appropriately. However,

since we are migrating the documents out of this database, we don't need to modify the underlying data. Instead, we will manage our findings about the PDF files within a control table that maintains a 1:1 relationship with the view:

```
CREATE TABLE documents_control (
    control_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    doc_id NUMBER,                  -- Links to the original record's ID
    source_table VARCHAR2(128),     -- Original table name for traceability
    source_rowid ROWID,             -- To uniquely identify the row in the source table
    valid_pdf CHAR(1) DEFAULT 'N',  -- 'Y' if valid PDF, 'N' otherwise
    pdf_hash VARCHAR2(64),          -- SHA-256 hash (64 hex characters)
    guid RAW(16) DEFAULT SYS_GUID() -- Globally Unique Identifier
);
```

We can populate this table with the base information from our view:

```
INSERT INTO documents_control (doc_id, source_table, source_rowid)
SELECT doc_id, source_table, source_rowid
FROM documents;
```

The remaining fields, valid_pdf, pdf_hash and guid, we will populate in subsequent steps.

This is what we now have in our test environment:

```
SQL> SELECT count(*) FROM documents;

  COUNT(*)
----------
       200

SQL> SELECT count(*) FROM documents_control;

  COUNT(*)
----------
       200
```

Identify BLOBs that are PDFs

Remember that not all of the documents are PDFs, 90% are, but we still have to cater for those other types. Luckily, PDFs identify themselves with a magic number right at the start.

This is 25 50 44 46 2D in hexadecimal which equates to "%PDF-". Here's an example output from the filesystem:

```
User@AN20 MINGW64 ~/docsim/docs
$ for f in `ls -1 *.pdf`; do od -c $f | head -1; done
0000000   %   P   D   F   -   1   .   4  \n   % 307 354 217 242  \n   5
0000000   %   P   D   F   -   1   .   4  \n   % 307 354 217 242  \n   5
0000000   %   P   D   F   -   1   .   4  \n   % 307 354 217 242  \n   5
```

Hash all PDFs

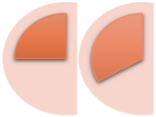
We have an additional problem. The documents are not always stored as raw PDFs. Depending on how they were added to the database, they may have additional wrapper control data and headers before we see the PDF file signature. So, we might need to read a bit more than just the first few bytes. The following PL/SQL code reads 8KB just to be sure:

```
DECLARE
  CURSOR doc_cursor IS
    SELECT ROWID AS rid, doc_id, doc_blob
    FROM documents;

  l_raw RAW(8192);
  l_hex VARCHAR2(16384);
  pdf_offset NUMBER;
  pdf_hash VARCHAR2(64);
  guid VARCHAR2(36);
BEGIN
  FOR doc_rec IN doc_cursor LOOP
    -- Read first 8KB to find PDF signature
    l_raw := DBMS_LOB.SUBSTR(doc_rec.doc_blob, 8192, 1);
    l_hex := RAWTOHEX(l_raw);
    pdf_offset := INSTR(l_hex, '255044462D'); -- Look for "%PDF-"

    IF pdf_offset > 0 THEN
      -- Valid PDF Found
      pdf_hash := LOWER(STANDARD_HASH(doc_rec.doc_blob, 'SHA256'));
      guid := SYS_GUID();  -- Generate GUID for SP

      -- Upsert into the control table
```

```
        MERGE INTO document_control dc
        USING (SELECT doc_rec.doc_id AS doc_id FROM dual) src
        ON (dc.doc_id = src.doc_id)
        WHEN MATCHED THEN
          UPDATE SET valid_pdf = 1, pdf_hash = pdf_hash, guid = guid
        WHEN NOT MATCHED THEN
          INSERT (doc_id, source_rowid, valid_pdf, pdf_hash, guid, processed_date)
          VALUES (doc_rec.doc_id, doc_rec.rid, 1, pdf_hash, guid, SYSDATE);
      ELSE
        MERGE INTO document_control dc
        USING (SELECT doc_rec.doc_id AS doc_id FROM dual) src
        ON (dc.doc_id = src.doc_id)
        WHEN MATCHED THEN
          UPDATE SET valid_pdf = 0
        WHEN NOT MATCHED THEN
          INSERT (doc_id, source_rowid, valid_pdf, processed_date)
          VALUES (doc_rec.doc_id, doc_rec.rid, 0, SYSDATE);
      END IF;
    END LOOP;
  END;
  /
```

This is the heart of our SQL PoC: we determine whether a BLOB is a PDF, generate a standard SHA256 hash and a unique GUID (which we could use as a document identifier from hereon).

Note the use of the "Upsert" operation via the MERGE INTO command. This might seem unnecessary since we have pre-populated the control table to be 1:1 mapped with the documents view. However, this is good defensive programming. It allows for new data to be added to the underlying table and the control table not getting populated to reflect these (though we ought to be doing data migration during a freeze). And it provides idempotency so re-running the script does not cause duplication issues or errors.



Identify and Mark Duplicates

And now for the all-important de-duplication step. What we will do is mark all but one document that has the same hash (i.e. are identical) as duplicates. We can then extract only the ones that are not, i.e. extract unique PDFs only, for loading into SharePoint.

First, let's extend the control table to have this new flag:

```
ALTER TABLE document_control ADD (duplicate_flag NUMBER(1));
-- 1 = Duplicate, 0 = Unique, NULL = Not evaluated yet
```

And now let's do the marking of duplicates:

```
BEGIN
  FOR rec IN (
    SELECT pdf_hash, COUNT(*) AS occurrences
    FROM document_control
    WHERE valid_pdf = 1
    GROUP BY pdf_hash
    HAVING COUNT(*) > 1
  ) LOOP
    -- Mark all but one as duplicates
    UPDATE document_control
    SET duplicate_flag = 1
    WHERE pdf_hash = rec.pdf_hash
      AND ROWNUM > 1; -- Keep the first occurrence as unique
  END LOOP;
END;
/
```

Ensuring Transactional Integrity

Although we have used defensive programming techniques In our PoC, we really haven't bothered much about transactional integrity. To put this into a production environment, we would have to take additional steps like locking tables, backing up the DB, disabling triggers, using transactions and savepoints, log our actions and have a rollback procedure pre-planned.

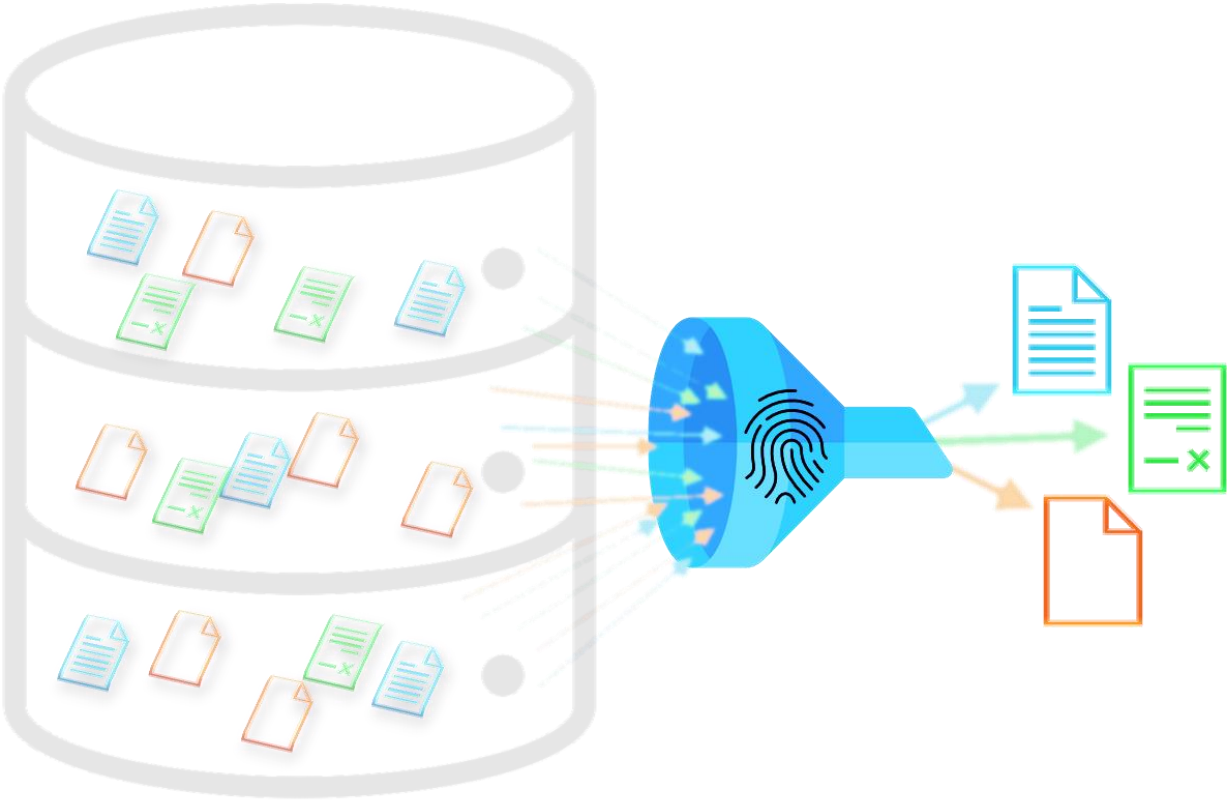
Extract Unique Documents

We can extract all the unique documents now ready for moving into SharePoint. This can be done in PL/SQL or externally. Let's do it with Lua for its simplicity and speed:


```
local luasql = require "luasql.oracle"
```

```
local env = luasql.oracle()
local conn = env:connect("ORCL", "username", "password")

local output_dir = "extracted_pdfs"
os.execute("mkdir " .. output_dir)

local cur = conn:execute([[
    SELECT d.doc_blob, dc.guid
    FROM documents d
    JOIN documents_control dc
      ON d.original_id = dc.doc_id
     AND d.source_table = dc.source_table
    WHERE dc.valid_pdf = 1 AND NVL(dc.duplicate_flag, 0) = 0
]])

local row = cur:fetch({}, "a")
while row do
    local file = io.open(output_dir .. "/" .. row.guid .. ".pdf", "wb")
    file:write(row.doc_blob)
    file:close()
    print("Extracted:", row.guid .. ".pdf")
    row = cur:fetch(row, "a")
end

cur:close()
conn:close()
env:close()
```

We now have all the PDF documents extracted from the old CRM into a local folder ready to the uploaded to SharePoint. They are named using the generated GUID which will be the key to the document within the new CRM. We don't need to use meaningful names as this SharePoint site will only be accessible to the new CRM and documents available only through the CRM.



Conclusion

For the in-situ de-duplication, we used Oracle PL/SQL to identify and manage duplicate documents directly within the database. We created a unified view that consolidated all BLOBs across relevant tables. A companion control table was introduced to track metadata such as document validity, hash values, and duplication status, ensuring a one-to-one mapping with the view.

The de-duplication process involved extracting the first few kilobytes of each BLOB to verify file signatures, specifically for PDFs, and computing SHA-256 hashes to detect exact duplicates. Only a single copy of every document was extracted and retained thus paving the way to continue this efficient practice of no more redundant copies in the new CRM. The extracted PDFs were named using GUIDs, ensuring consistency and traceability when linking documents in the new system.

Next Up

We will explore in depth the PoC based on external programs, specifically Python scripts, that will not only de-duplicate our PDF documents but also use advanced algorithms to identify and extract templates from personalized documents. While this phase will focus on near-duplicate detection using fuzzy matching and text-based similarity algorithms, future enhancements may draw inspiration from advanced redundancy detection techniques, such as those outlined by Policroniades & Pratt (2004). Their work on content-defined chunking and

similarity detection at the storage level offers promising methodologies for identifying partial overlaps and structural redundancies in large document datasets. This approach could significantly enhance our ability to detect near-duplicates, even when documents differ only in small, personalized sections.

References

Policroniades, C., & Pratt, I. (2004, June). Alternatives for Detecting Redundancy in Storage Systems Data. In *USENIX Annual Technical Conference, General Track* (pp. 73-86).