"Duplicate Payments and Duplicate Vendors: How to Identify All of Them"

by
Christine L. Warner
Automated Auditors, LLC

When Sarbanes-Oxley was passed in 2002, many companies were forced to take an in-depth look at internal controls. Despite efforts to tighten controls, duplicate payments still occur by the millions across all industries. Mark Van Holsbeck, Director of Enterprise Network Security for Avery-Dennison, estimates that corporations make duplicate payments at the rate of 2%.[1] Two percent may not sound like much, but if your company's A/P invoices total $75 million, duplicate payments may account for $1.5 million. Take a look at the statistics:

Medicare
The Dept of Health & Human Services' Inspector General estimated that Medicare made $89 million of duplicate payments in 1998.[2]

Cingular
" We have once again discovered that payments made online as an Electronic funds payment for TDMA accounts, have been deducted twice from the customer's checking account. "[3]

Medicaid
"We identified at least $9.7 million in such duplicate payments during our two-year audit period, and estimated that as much as $31.1 million in additional duplicate payments may have been made."[4]

Department of Veteran's Affairs
"As part of our ongoing audit of the Department of Veterans Affairs' (VA's) financial operations and reporting, the Office of Inspector General identified two duplicate payments totaling $1.08 million…"[5]

Defense Finance and Accounting Service (DFAS)
"DFAS made at least $13.2 million in duplicate payments, an estimated 30,584 payments in the wrong amounts, and other vendor payments that did not comply with 5 Code of Federal Regulations Part A."[6]

Department of Education

---

[1]Van Holsbeck, Mark and Johnson, Jeffrey Z. "Security in an ERP World" (May, 2004) as seen on www.net-security.com
[2]http://oig.hhs.gov/oei/reports/oei-03-00-00091.pdf (1998)
[3]http://forums.cingular.com/cng/board/message, online message board (March, 2005)
[4]http://www.osc.state.ny.us/audits/allaudits/093004/04f2.pdf (June, 2004), Antonia C. Novello
[5] http://www.va.gov/oig/52/reports/1997/7AF-G01-035--duppay.pdf (April, 1997)
[6] http://www.dodig.osd.mil/audit/reports/fy02/02056sum.htm, (March, 2002) DoD Office of the Inspector General

"ED has issued more than $150 million in duplicate payments to grantees and contractors in the past year alone [1999]. Recipients literally receive two checks, each worth the full amount due."[7]

In a rush to find the overpayments, many companies have emerged: A/P Recap, Automated Auditors, AP Recovery, ACL, Idea, and more. That these companies are thriving is a testament to the fact that duplicate payments still occur at an alarming rate.

Many software packages have some controls over duplicate invoices but it usually takes some in-depth querying to find them all. For example, many accounting packages do a duplicate invoice check and prevent you from keying in a duplicate invoice number for the same vendor. But just add an "A" to the invoice number or change a penny and you are on your way to a duplicate payment. Another common mistake is found in vendor files; duplicate vendor numbers for the same vendor is the number one cause of duplicate payments.

This article describes tested logic used to catch ALL of your duplicate payments and duplicate vendors. The logic outlined in this article has been used to identify millions in overpayments, for companies as large as MCI and Komatsu, but will also work for smaller corporations.

Duplicate Invoices

Common Mistakes and Solutions

There are a couple of common pitfalls that, if avoided, will strengthen your duplicate payment review process. Here are three common mistakes and some suggested solutions for avoiding them.

#1: Many people mistakenly run a query where only the invoice # is different (vendor #, invoice date, and amount are the same) but end up with a lot of junk on their reports that they have to comb through.

Solution: Try extracting only the numbers of an invoice number and matching on this. You can do this using a combination of SAS'S SUBSTR function and the INDEXC function. (See page 5 for the code to do this). This way you will catch more true duplicates and less "junk".

[7] http://interversity.org/lists/arn-l/archives/Oct2000/msg00899.html

#2: Another common query used is one where only the invoice date is different. This is a good query but often catches legitimate rent or lease payments.

Solution: Try identifying dates that are similar, for example, dates that are less than 30 days apart. They are much more likely to be duplicates than invoices paid one year apart. Also: attempt to eliminate rent/lease payments from your duplicate report by eliminating long lists of duplicates. If there are 4 or more duplicates per grouping (like the example below), the dupes are most likely legitimate rent/lease payments.

Vendor Name:	Invoice #:	Invoice Date:	Invoice Amount:
Realty, Inc.	RENT	3/1/2003	$2,010.00
Realty, Inc.	RENT	4/2/2003	$2,010.00
Realty, Inc.	RENT	5/1/2003	$2,010.00
Realty, Inc.	RENT	6/1/2003	$2,010.00
Realty, Inc.	RENT	7/3/2003	$2,010.00

#3: Credits are not factored in to the program. If you match on EXACT amount, you may find duplicates but not the credit that reversed the duplicate. Example:

Vendor:	Amount:	Date:	Invoice #:
ABC, Inc.	$525,199.10	6/25/2002	INV001
ABC, Inc.	$525,199.10	7/22/2002	INV001A
ABC, Inc.	-$525,199.10	7/23/2003	INV001CR

Solution: Try matching on the absolute value of the amount instead. We actually take out credits and the invoice it applies to so that these are eliminated before the duplicate payment logic is run.

9 Core Algorithms

In order to maximize the accuracy of your dupe payment reports, we have outlined some tested logic that identifies all duplicate payments without adding extraneous junk to the reports.

You will need 4 fields at a minimum to identify dupe invoices:

Vendor / supplier number
Invoice number
Invoice date
Invoice amount

Our approach consists of 9 core duplicate payment detection algorithms. The table below lists the basic structure of the algorithms:

Algorithm #	Vendor #	Invoice #	Invoice Date	Invoice Amount
1	Exact	Exact	Exact	Exact

2	Different	Exact	Exact	Exact
3	Exact	Similar	Exact	Exact
4	Exact	Exact	Similar	Exact
5	Exact	Exact	Exact	Similar
6	Exact	Similar	Exact	Similar
7	Exact	Similar	Similar	Exact
8	Exact	Exact	Similar	Similar
9	Different	Exact	Similar	Exact

In the section below, we describe each algorithm and give examples of ACTUAL duplicates found using each. Please note that all vendor names have been changed to protect data confidentiality agreements.

Algorithm #1: EEEE

This is the most accurate duplicate where the vendor number, invoice number, invoice date, and invoice amount are all exactly the same.

Vendor Name	Invoice #	Invoice Date	Invoice Amount
GOOD FRAGRANCES INC	54891	4-May-01	$2,750.00
GOOD FRAGRANCES INC	54891	4-May-01	$2,750.00

Algorithm #2: DEEE

This algorithm identifies invoices where the vendor number is different but the invoice number, date, and amount are all exactly the same.

Vendor Name	Invoice #	Invoice Date	Invoice Amount
H & L PACKAGING	29869-25	19-Aug-04	$2,139.25
PETTY CASH	29869-25	19-Aug-04	$2,139.25

Algorithm #3: ESEE

This algorithm identifies invoices where the vendor number is exactly the same, the invoice number is SIMILAR but not exactly the same, and the date and amount are exactly the same.

Vendor Name	Invoice #	Invoice Date	Invoice Amount
TURNCO PACKAGING CORP.	8012306	22-Nov-02	$8,870.40
TURNCO PACKAGING CORP.	08012306A	22-Nov-02	$8,870.40

When comparing invoice numbers to see if they are similar, try extracting only the numbers 1 through 9, and then comparing this newly created field. For example, we create a field called "INVCODE" which contains only the numbers 1 through 9, eliminating all letters and all zeros. We create the INVCODE using the following logic in SAS:

```
%macro invcode(code);

/* strip leading zeros and blanks from a string like invoice number */
/* returns invcode */

length invcode $50;

invcode="X";

do i = 1 to length(&code);
    char = SUBSTR(&code,i,1);
    if indexc(char,'123456789')>0 then invcode= compress(invcode||char);
end;

%mend;
```

Then we match on INVCODE instead of invoice number using the following PROC SQL code:

```
SELECT * FROM INVOICE_TABLE A, INVOICE_TABLE B
WHERE A.INVCODE=B.INVCODE AND
      A.INVOICENUM <> B.INVOICENUM
```

Make sure you specify that the invoice numbers cannot be exactly equal or you will end up identifying duplicates you already identified in algorithm #1 (EEEE).

Algorithm #4: EESE

This algorithm identifies invoices where the vendor numbers and invoice numbers are exactly the same, the invoice dates are similar, and the amounts are exactly the same. We usually define dates as "similar" if they are less than 30 days apart, but this variable can be changed to any number of days. For a more accurate listing, we often decrease the date tolerance to 14 or 21 days.

Vendor Name	Invoice #	Invoice Date	Invoice Amount
CC PACKAGES, INC.	4206	12-Dec-02	$12,716.98
CC PACKAGES, INC.	4206	9-Jan-03	$12,716.98

Algorithm #5: EEES

This algorithm identifies invoices where all fields are exactly the same except amount, but the amount is SIMILAR, not exact and not completely different.

We define amounts as "similar" if they meet one of the following criteria:

1) they are within 3% of one another, +/-
2) one is twice as much as the other one
3) they start with the same first 4 digits (e.g. $1,234.50 and $123.45)

Here are three examples of each type of amount-similar duplicate:

1) amounts are within 3% of one another:

Vendor Name	Invoice #	Invoice Date	Invoice Amount
NORWALK, INC.	970003511	24-May-01	$1,650.26
NORWALK, INC.	970003511	24-May-01	$1,620.00

2) one invoice is half of the other:

Vendor Name	Invoice #	Invoice Date	Invoice Amount
TRANSPORT SOUTHWEST	23401/2/3	14-Jan-05	$4,143.00
TRANSPORT SOUTHWEST	23401/2/3	14-Jan-05	$8,286.00

3) invoices start with the same first 4 digits:

Vendor Name	Invoice #	Invoice Date	Invoice Amount
HLC LTD.	Blank	22-Jul-02	$15,233.76
HLC LTD.	Blank	22-Jul-02	$1,523.38

We created a SAS macro to determine whether the amounts were similar or not. The macro returns a Boolean variable amtsimilar that is 1 if the amounts are similar and 0 if they are not.

```
%macro amtsim(amt1,amt2,amtsimilar);

/* checks to see if amounts are similar */
/* returns AMT_SIMILAR */

amt_similar = 0;

If &amt1 < &amt2 Then
do;
    thresh = 0.03 * &amt2;
    If (&amt1 >= &amt2 - thresh) And (&amt1 < &amt2 + thresh) Then amt_similar
= 1;
End;
```

```
If &amt2 < &amt1 Then
do;
    thresh = 0.03 * &amt1;
    If (&amt2 >= &amt1 - thresh) And (&amt2 <= &amt1 + thresh) Then amt_similar
= 1;
End;

If (&amt1 * 2 = &amt2) Or (&amt2 * 2 = &amt1) Then amt_similar = 1;
If &amt1 = &amt2 Then amt_similar = 1;
If &amt1 * (-1) = &amt2 Then amt_similar = 1;

str1 = input(&amt1,$15.);
str2 = input(&amt2,$15.);

If SUBSTR(str1, 1, 4) = SUBSTR(str2, 1, 4) Then amt_similar = 1;

&amtsimilar=amt_similar;  /* 1=true-similar, 0=false-not similar */

%mend;
```

Then we called the macro later on in the program to create the Boolean variable "similar":

```
%amtsim(invamt1, invamt2, similar);
```

The following line is executed to include only invoices whose amounts are similar. The "id1 < id2" portion includes one and only one unique combination of the duplicates. If this logic were excluded, you would end up with two identical pairs of duplicates, making your report twice as large as it needed to be.

```
if similar=1 and id1 < id2; /* take only 1 combination */
```

Algorithm #6: ESES

This logic catches duplicates with the same vendor #, similar invoice #, same date, and similar amount. In this example, the amounts are considered similar because one invoice is half of the other.

Vendor Name	Invoice #	Invoice Date	Invoice Amount
CC PACKAGES, INC.	11903	8-Sep-04	$4,290.00
CC PACKAGES, INC.	11903A	8-Sep-04	$8,580.00

Algorithm #7: ESSE

This logic identifies duplicates with the same vendor #, similar invoice #, similar date, and exact amounts.

Vendor Name	Invoice #	Invoice Date	Invoice Amount
CI SERVICES	22762	25-Jun-03	$2,725.13
CI SERVICES	22762A	17-Jul-03	$2,725.13

Algorithm #8: EESS

This logic identifies duplicates with the same vendor #, same invoice #, similar date, and similar amount.

In this example, the amounts are considered similar because they are within 3% of each other. Also, the dates are within 30 days of each other. It is possible that these could represent two separate legitimate payments, but having the same invoice number indicates they are true duplicates.

Vendor Name	Invoice #	Invoice Date	Invoice Amount
VULCAN INDUSTRIES, INC.	59282	14-Jun-04	$10,828.44
VULCAN INDUSTRIES, INC.	59282	13-Jul-04	$10,830.00

Algorithm #9: DESE

This algorithm identifies duplicates with different (not similar and not exact) vendor #s, same invoice #s, similar dates, and exact amounts.

Vendor Name	Invoice #	Invoice Date	Invoice Amount
SMITH LABORATORIES, INC.	90796747	18-Jul-02	$5,367.32
ASBURY CHEMICALS	90796747	31-Jul-02	$5,367.32

Algorithm #10: Matching with blank invoice numbers:

Another algorithm that we have recently added to our repertoire is matching with blank invoices. Normally, A/P systems will not accept invoices without invoice numbers. Apparently, some checks still are cut without invoice numbers – look at the example below:

Vendor Name	Invoice #	Invoice Date	Invoice Amount
CHEMCUT SOUTHWEST, L.	3316983	1-Jun-05	$10,641.00
CHEMCUT SOUTHWEST, L.	Blank	1-Jun-05	$10,641.00

In another example, BOTH invoices were left blank, leading to an $84,000 duplicate payment:

Vendor Name	Invoice #	Invoice Date	Invoice Amount
C&C LABS, INC.	Blank	29-Apr-05	$84,246.60
C&C LABS, INC.	Blank	2-May-05	$84,246.60

Duplicate Vendors

What is a "duplicate vendor"? This occurrence really should be called "multiple vendor numbers for the same vendor", because that is what we are referring to when we say "duplicate vendors". Having multiple vendor ids for the same vendor is a problem because it directly leads to duplicate payments. Many accounting packages do have dupe payment controls, but they do a quick check on vendor number and invoice number, and if there is an exact match, the system will prohibit you from creating that invoice for that vendor number. If the vendor has another number, however, the duplicate invoice will pass through the system.

We recommend implementing at least three duplicate vendor search algorithms:

1) address matching: identify vendors with same/similar addresses
2) tax ID matching: identify vendors with same/similar tax IDs
3) bank routing # matching: identify vendors with same/similar bank routing #s

Address Matching

There are several address matching algorithms out there. There are fancy "sound-ex" functions that compare individual words in the address with other words. There is the filter-approach where you take out the noise words like "Road", "Drive", "Suite", etc. In our experience, we have found that filtering the street address first helps tremendously. We filter out noise words such as "North", "South", "Road", "Highway", "Suite", and many more. Then we rely on one of two well-tested algorithms: 1) the number approach and/or 2) the first 6 digit approach.

Filtering Logic

Filtering the street address field is not extremely complicated. The best algorithms are those that have been tested on several different sets of data so that every noise word you could encounter has been eliminated. Here is a list of common noise words that we systematically eliminate before conducting address-matching. This list is by no means exhaustive; we are constantly adding to it to improve our algorithms.

Noise Words

St.
Street
Ave.
Avenue
Dr.
Drive
Rd.
Road
Blvd
Boulevard
Ct.
Court
Way
P.O. Box
PO Box
P O Box
P. O. Box
Hwy
Highway
RR.
Rural Route
South
North
West
East
S.
N.
E.
W.
SE
SW
NE
NW
Lane
Ln.
Ste
Suite

We rely on SAS's TRANWRD function to perform the word replacements. The syntax for this function is:

TRANWRD(*source,target,replacement*)

Where you replace "target" with "replacement" in the field called "source". For example, applying this procedure to the noise words above, we want to change these words to blanks. So we would call the TRANWRD function with the following parameters:

```
TRANWRD(address,=Suite=,= ||);
```

The following are changed to their language equivalent: 1st is changed to "First" using the TRANWRD function: 1st 2nd 3rd 4th 5th 6th 7th 8th 9th 10th

After the noise words are eliminated, take out all vowels + Y, spaces, and all punctuation. Here are some suggestions for eliminating punctuation: #&*,-./\'. You can do this using SAS's COMPRESS function:

```
addresscode = COMPRESS(address,'#&*,-./\');
```

Number Approach

With the number approach, you basically create a new field called "addresscode" that has all of the numbers that are in the street address. Then you concatenate (add) the zip code to this field so that you are only comparing addresses in the same zip code area. For example:

Street:	***Zip Code:***	***Address Code:***
4300 Fair Lakes Court	22033	430022033
4300 Fair Lakes Ct. Ste 101	22033	430010122033
1578 S. Main St, Ste 420	16803	157842016803

Then, once the address code field is created, match the records on address code to find your duplicates.

Pros/Cons with the Number Approach

The advantage to using the number approach is it is easy and quick to implement. It does not eat up a lot of system time, either. When you use the number approach, you do not have to run the address filter for noise words because words are not matched upon.

The only problem with the number approach is that suite numbers and apartment numbers get added to the address code and prevent some duplicates from being caught. In the above example, the first two addresses should be considered duplicates but were not because of the suite number 101.

To avoid this problem, we sometimes use the first 6 digit approach.

First 6 Digit Approach

To implement this heuristic, first apply the filter logic described above to take out noise words. Then, just take the first six digits of the filtered street address and then concatenate the zip code to it to create your address code. For example:

Street:	***Zip Code:***	***Address Code:***
4300 Fair Lakes Court	22033	4300FA
4300 Fair Lakes Ct. Ste 101	22033	4300FA
1578 S. Main St, Ste 420	16803	1578MA

Note that for the "S. Main St." address, the address code does NOT contain the "S." in the address code. This is because we filtered out the "S." prior to creating the address code.

Pros/Cons with the First 6 Digit Approach

The advantage to using the first 6 digit approach is that two addresses will be considered similar if one has a suite # / apartment # and the other does not. This is a "less exact" match and will produce more duplicates than the numbers-only approach. So in the above example, the "Fair Lakes Court" addresses will be considered similar.

The disadvantage if there is one is that you must first apply the filtering logic to eliminate noise words. This usually does not take a substantial amount of time but does require that you maintain the noise-word list.

Tax ID Matching

Most vendor files have a vendor/supplier EIN or Tax ID for each vendor. If you have this in your vendor file, use it to capture duplicate vendors. We recommend implementing two search algorithms: 1) exact and 2) one-digit off.

Exact

This approach is very easy; just sort your vendor file by the tax ID field and you can visually identify your dupes. Or, you can also use some simple PROC SQL to catch your duplicate vendors:

```
PROC SQL;
CREATE TABLE TAXID1 AS
SELECT * FROM VENDOR_TABLE A, VENDOR_TABLE B
WHERE A.TAXID=B.TAXID AND
      A.VENDORID <> B.VENDORID;
QUIT;
```

You must specify "a.vendorid not equal to b.vendorid" so you do not label a record as a dupe of itself.

One-Digit Off

The one-digit off approach is slightly more complicated and more time-consuming for your computer's CPU, but is worth it. If all of the digits in the vendor tax ID are equal except for one, we consider this a duplicate. For example, the following pair of tax IDs would be considered duplicates in this algorithm because they are only 1 digit off:

```
Vendor Tax ID:
123456789
123556789
```

We rely heavily on SAS'S SUBSTR function to carry out this procedure. Here is some suggested PROC SQL code to catch tax IDs that have the first digit unequal but the rest equal. We are assuming in this example that your tax ID is length 9.

```
PROC SQL;
CREATE TABLE TAXID2 AS
SELECT * FROM VENDOR_TABLE A, VENDOR_TABLE B
WHERE SUBSTR(A.TAXID,1,1) <> SUBSTR(B.TAXID,1,1) AND
      SUBSTR(A.TAXID,2,8) = SUBSTR(B.TAXID,2,8) AND
      A.VENDORID <> B.VENDORID;
QUIT;
```

Then you repeat this code for each digit in the tax ID, changing the parameters of the SUBSTR function as you go.

Bank Routing # Matching

In most cases, each vendor has one and only one bank routing number. If you have this field available to you in your vendor file, use it to conduct duplicate vendor searches. When we conduct duplicate bank routing number searches, we use exactly the same algorithm as we do for the tax ID search described above. The only item we change is the length of the field. For example, if the length of the bank routing # is 8, then our PROC SQL code would change accordingly (note the 7 in the 3rd line instead of an 8):

```
PROC SQL;
CREATE TABLE BANK1 AS
SELECT * FROM VENDOR_TABLE A, VENDOR_TABLE B
WHERE SUBSTR(A.BANKID,1,1) <> SUBSTR(B.BANKID,1,1) AND
      SUBSTR(A.BANKID,2,7) = SUBSTR(B.BANKID,2,7) AND
      A.VENDORID <> B.VENDORID;
QUIT;
```

The only problem with this approach may occur when you have differing lengths. This is easily remedied by adding a leading zero if you need one, however. Also, if your tax IDs or Bank IDs have alpha characters in them, eliminate them first using SAS's COMPRESS function so that you are only comparing numbers.

About the Author

Christy Warner is the president of Automated Auditors, LLC, located in the Washington DC area. Automated Auditors is a data mining corporation that specializes in identifying duplicate payments, duplicate vendors, and financial fraud. For more information, you can contact her at cwarner@autoaudit.com or visit www.autoaudit.com.