Experian – Identity As Service (IaaS)

MVP Solution – Technical Design Document

November 5, 2021

Contents

[Introduction 2](#_Toc87017985)

[Altimetrik Team Role 2](#_Toc87017986)

[Zobrist Hashing 3](#_Toc87017987)

[Double Metaphone 3](#_Toc87017988)

[UDF Creation 3](#_Toc87017989)

[Schema Details – Client Input Table 4](#_Toc87017990)

[Schema Details – Experian Identity Graph Table 4](#_Toc87017991)

[Standardization of PII Data 5](#_Toc87017992)

[First Name Standardization 5](#_Toc87017993)

[Surname Name Standardization 5](#_Toc87017994)

[Address Standardization 5](#_Toc87017995)

[Email Standardization 5](#_Toc87017996)

[Business logic on Hash-Table creation (converting PIIs to Hashes) 5](#_Toc87017997)

[How the Hashing and Metaphone work in SQLs 6](#_Toc87017998)

[Matching Algorithm & Experian Identity Key Assignment 7](#_Toc87017999)

[IaaS MVP – Process Flow Diagram 7](#_Toc87018000)

[Person Level Match 8](#_Toc87018001)

[Household Level Match 8](#_Toc87018002)

[Nameless Match 8](#_Toc87018003)

[No Match: 8](#_Toc87018004)

[Code Snippet 9](#_Toc87018005)

[JAVA Code – Zobrist Hash Algorithm 9](#_Toc87018006)

[SQL Code – DDLs and Stored Procedure 11](#_Toc87018007)

[Client Secure View and Encryption 19](#_Toc87018008)

[Tracking and Billing 19](#_Toc87018009)

# Introduction

Identity as a Service (IaaS) – is a means of matching a Client data with Experian master data for customer analysis.

The goal of the Experian Embedded ID (aka “Identity-as-a-Service”) solution is to give clients an anonymous way to join data within Snowflake, using intelligent IDs created by Experian.  Clients can use these keys to do customer analysis and to create new segments that previously would have required the sharing of customer data.

# Altimetrik Team Role

Altimetrik Team is a strategic partner in developing a customized solution for,

1. Standardization of Personally Identifiable Information (PII) using built in functions and new UDFs
2. Hashing of Client Data using new UDFs
3. Creating a unique and robust matching Algorithm for Client Data
4. Creating a Tracking and Billing system

We will use Snowflake Database for Data storage and consume the Snowflake Data processing engine to complete our development of Stored Proc based Matching Algorithm.

JAVA will be used in coding the Hashing Algorithm and creating UDFs in Snowflake.

# Zobrist Hashing

A [hash function](https://en.wikipedia.org/wiki/Hash_function) construction used in [computer programs](https://en.wikipedia.org/wiki/Computer_program) that play [abstract board games](https://en.wikipedia.org/wiki/Abstract_board_game), such as [chess](https://en.wikipedia.org/wiki/Computer_chess) and [Go](https://en.wikipedia.org/wiki/Computer_go), to implement [transposition tables](https://en.wikipedia.org/wiki/Transposition_table), a special kind of [hash table](https://en.wikipedia.org/wiki/Hash_table) that is indexed by a board position and used to avoid analyzing the same position more than once.

Zobrist hashing starts by [randomly generating](https://en.wikipedia.org/wiki/Pseudorandom_number_generator) [bit strings](https://en.wikipedia.org/wiki/Bitstring) for each possible element of a board game, i.e. for each combination of a piece and a position. Now any board configuration can be broken up into independent piece/position components, which are mapped to the random bit strings generated earlier. The final Zobrist hash is computed by combining those bit strings using bitwise [XOR](https://en.wikipedia.org/wiki/XOR).

In the UDF created first a Zobrist table is taken and is converted to the text file. The text file contains the hex values in the form of an array. Those hex values are converted into the bytes in java and stored in a variable to pass.

In the main java program the input string is taken and is converted to bytes. Then an empty variable is taken and is initialized with bytes of the random hex values. Then a set of operations are performed on the initialized variable and the input byte including the bitwise XOR operation to get the final hashed values in bytes. The byte array then is converted to long data type as final hashed output.

# Double Metaphone

The Double Metaphone phonetic encoding algorithm is the second generation of the Metaphone algorithm. It makes a number of fundamental design improvements over the original Metaphone algorithm.

It is called “Double” because it can return both a primary and a secondary code for a string. This accounts for some ambiguous cases as well as for multiple variants of surnames with common ancestry. For example, encoding the name “Smith” yields a primary code of SM0 and a secondary code of XMT, while the name “Schmidt” yields a primary code of XMT and a secondary code of SMT–both have XMT in common.

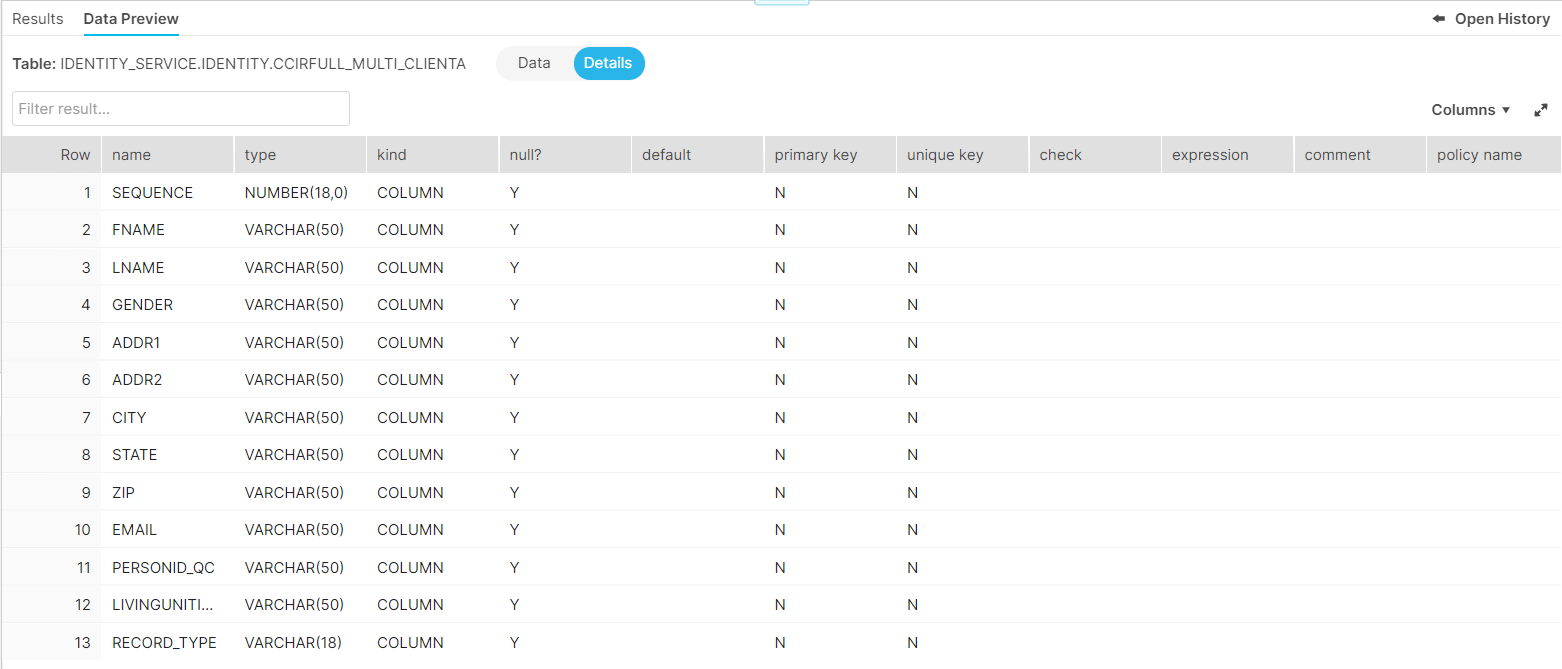
Double Metaphone tries to account for myriad irregularities in English of Slavic, Germanic, Celtic, Greek, French, Italian, Spanish, Chinese, and other origin. Thus it uses a much more complex ruleset for coding than its predecessor; for example, it tests for approximately 100 different contexts of the use of the letter C alone.

In the implementation of this algorithm to resolve the string a java package is used to generate the code of the input strings.

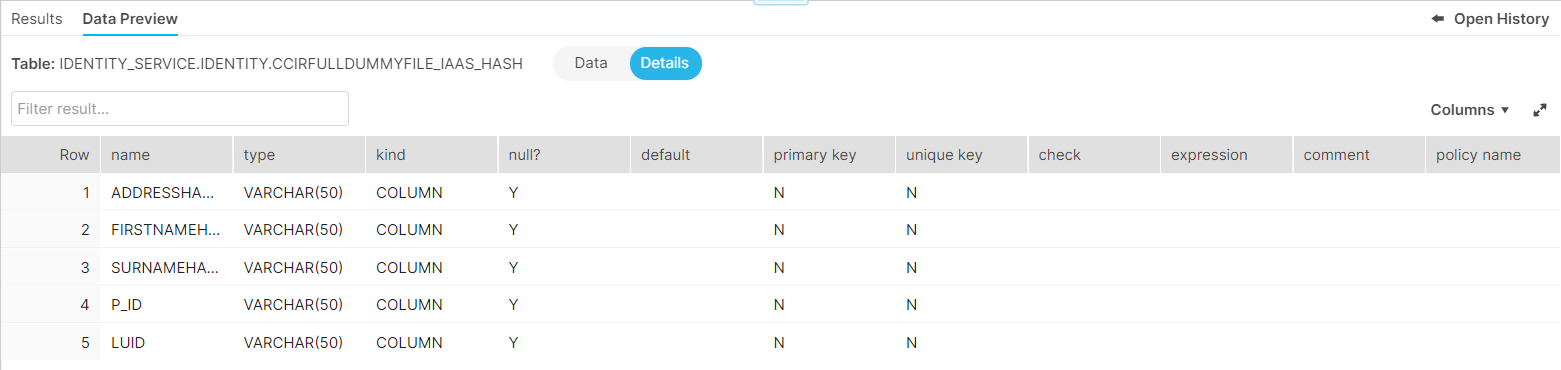
# UDF Creation

User-defined functions (UDFs) let you extend the system to perform operations that are not available through the built-in, system-defined functions provided by Snowflake. Snowflake currently supports the SQL, Java and JavaScript languages for writing UDFs. In our implementation we have used SQL. A JAVA UDF evaluates an arbitrary SQL expression and returns either scalar or tabular results.

# Schema Details – Client Input Table



# Schema Details – Experian Identity Graph Table



# Standardization of PII Data

## First Name Standardization

* If there are multiple parts in a First Name, only the first part of it is considered
* Above, first part of First Name is matched with a PROPER\_NAME table to get the real name for those colloquially used ALIAS names
* If there is no match available in the PROPER\_NAME table, the first part of First Name itself will be used in further processing

## Surname Name Standardization

* If there are multiple parts in a Surname, only the first part of it is considered
* Using DoubleMetaphone algorithm, extract the phonic for the first part of Surname
* Finally the standardized Surname outcome is the concatenation of “First Letter of Surname” + “-“ (hypen) + phonic value of the first part of Surname.

## Address Standardization

* For Address standardization, take Address\_Line\_1 attribute and remove / trim spaces between the words and other special characters like Slash (/) or Dot (.) or Hypen (-) etc.,
* Standardized Address outcome is the concatenation of above “Address\_Line\_1” value + “Zip Code” combined together

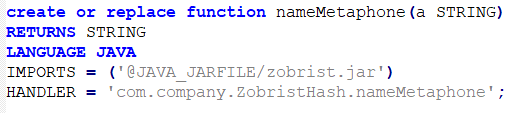
## Email Standardization

* For Email, take only the name and domain part using a pattern matching technique
* We use the Snowflake Regexp\_substr function and the following pattern is used for standardizing the emails : '\\\\w\\\\S+\@[[:alnum:]]+\.[[:alnum:]]+'

# Business logic on Hash-Table creation (converting PIIs to Hashes)

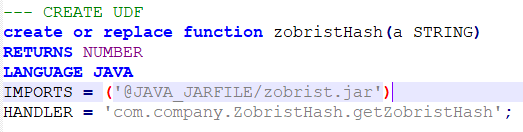
* Use the standardized First name, pass it to Zobristhash function to get FirstName\_Hash value
* Use the standardized Surname value to Zobristhash function to get the Surname\_Hash value.
* For Address, pass the standardized address value to Zobristhash function to get the Address\_Hash value
* For Email, pass the standardized email to Zobristhash function to get the Email\_Hash value
* All the above Standardized & Hashed data are loaded into the corresponding client HASH\_TABLE.

# How the Hashing and Metaphone work in SQLs



* The Metaphone value for the last name can be generated by calling the user defined function (UDF) nameMetaphone and passing the first word of last name as the input parameter to the function.

e.g., Select Namemetaphone (‘KULAS’) will give the output as ‘KLS’



* The hash values for the identity column values can be generated by calling the user defined Zobristhash function.

e.g., Select Zobristhash (‘daniel’) will give the output as ‘-1843002136784089291’

# Matching Algorithm & Experian Identity Key Assignment

Once the Primary keys or PII data shared by client is standardized based on the business logic, Zobrist hash algorithm is used to generate their corresponding hashes, which in turn matched with Experian Identity Graph (Consumer view) to get the Individual key (PID) or Household key (LUID) based on the match cases.

There are couple of business logic used to match with consumer view to generate the Experian Identity keys. SQL stored procedure has been created to get PID & LUID and below are the details on match cases.

* Person Level Match
* Household Level Match
* Nameless Match
* No Match

# IaaS MVP – Process Flow Diagram



## Person Level Match

1. FName, LName and Postal Address Match

When a combination of primary keys (FName, LName and Postal Address) match was found between Hash\_Table and Experian Identity Graph, fetch the PID and LUID from Experian Identity Graph.

1. FName, LName and Email Address Match

When a combination of primary keys (FName, LName and Email Address) match was found between Hash\_Table and Experian Identity Graph, fetch the PID and LUID from Experian Identity Graph.

## Household Level Match

1. LName and Postal Address Match

When a combination of primary keys (LName and Postal Address) match was found between Hash\_Table and Experian Identity Graph, fetch the LUID from Experian Identity Graph.

1. LName and Email Address Match

When a combination of primary keys (LName and Email Address) match was found between Customer match view and Experian Identity Graph, fetch the LUID from Experian Identity Graph.

## Nameless Match

1. Email Address Match

If records are without name, but that do have an Email address, attempt to make one pass and get a PID and LUID from the Experian Identity Graph to append.

## No Match:

If a match is not found in the Experian Identity Graph through any of the above scenarios, expectation is to pass on null value for PID and LUID for the corresponding Record Id.

# Code Snippet

## JAVA Code – Zobrist Hash Algorithm

package com.company;  
  
import org.apache.commons.codec.language.DoubleMetaphone;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.nio.ByteBuffer;  
import java.util.ArrayList;  
import java.util.List;  
  
  
public class ZobristHash {  
    public static Byte[] *zTable*;  
    public static final String *filename* = "zobrist\_table.txt";  
  
    public ZobristHash() {  
        List<Byte> byteList = new ArrayList<>();  
        try {  
            InputStream stream = getClass().getClassLoader().getResource(*filename*).openStream();  
            InputStreamReader isr1 = new InputStreamReader(stream, "UTF-8");  
            BufferedReader br = new BufferedReader(isr1);  
  
            String str;  
            Integer n;  
            while ((str = br.readLine()) != null) {  
                String[] tokens = str.split(",");  
                for (String token : tokens) {  
                    //convert hex string to byte and add to the array  
                    n = Integer.*valueOf*(token.substring(2, 4), 16);  
                    byteList.add(n.byteValue());  
                }  
            }  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
        *zTable* = new Byte[byteList.size()];  
        *zTable* = byteList.toArray(*zTable*);  
    }  
  
  
    public long getZobristHash(String inputstring) {  
        byte[] inputstringptr = inputstring.getBytes();  
        int offset = 0;  
        byte[] res = new byte[8];  
        res[0] = (byte) 0x9E;  
        res[1] = (byte) 0x37;  
        res[2] = (byte) 0x79;  
        res[3] = (byte) 0xB9;  
        res[4] = (byte) 0x7F;  
        res[5] = (byte) 0x4A;  
        res[6] = (byte) 0x7C;  
        res[7] = (byte) 0x13;  
        for (int i = 0; i < inputstring.length(); i++) {  
            res[0] = (byte) (Byte.*toUnsignedInt*(res[0]) ^ Byte.*toUnsignedInt*(*zTable*[(256 \* 8) \* offset + inputstringptr[i] \* 8]));  
            res[1] = (byte) (Byte.*toUnsignedInt*(res[1]) ^ Byte.*toUnsignedInt*(*zTable*[(256 \* 8) \* offset + inputstringptr[i] \* 8 + 1]));  
            res[2] = (byte) (Byte.*toUnsignedInt*(res[2]) ^ Byte.*toUnsignedInt*(*zTable*[(256 \* 8) \* offset + inputstringptr[i] \* 8 + 2]));  
            res[3] = (byte) (Byte.*toUnsignedInt*(res[3]) ^ Byte.*toUnsignedInt*(*zTable*[(256 \* 8) \* offset + inputstringptr[i] \* 8 + 3]));  
            res[4] = (byte) (Byte.*toUnsignedInt*(res[4]) ^ Byte.*toUnsignedInt*(*zTable*[(256 \* 8) \* offset + inputstringptr[i] \* 8 + 4]));  
            res[5] = (byte) (Byte.*toUnsignedInt*(res[5]) ^ Byte.*toUnsignedInt*(*zTable*[(256 \* 8) \* offset + inputstringptr[i] \* 8 + 5]));  
            res[6] = (byte) (Byte.*toUnsignedInt*(res[6]) ^ Byte.*toUnsignedInt*(*zTable*[(256 \* 8) \* offset + inputstringptr[i] \* 8 + 6]));  
            res[7] = (byte) (Byte.*toUnsignedInt*(res[7]) ^ Byte.*toUnsignedInt*(*zTable*[(256 \* 8) \* offset + inputstringptr[i] \* 8 + 7]));  
            offset++;  
        }  
        return convertByteArrayToLong(res);  
    }  
  
    private long convertByteArrayToLong(byte[] longBytes) {  
        ByteBuffer byteBuffer = ByteBuffer.*allocate*(Long.*BYTES*);  
        byteBuffer.put(longBytes);  
        byteBuffer.flip();  
        return byteBuffer.getLong();  
    }  
  
    public String nameMetaphone(String word) {  
        DoubleMetaphone dm = new DoubleMetaphone();  
        dm.setMaxCodeLen(100);  
        return dm.doubleMetaphone(word);  
    }  
  
    public static void main(String[] args) {  
        ZobristHash zHash = new ZobristHash();  
    }  
}

## SQL Code – DDLs and Stored Procedure

1 -- One Time Activities --

2

3 create or replace sequence run\_id start = 1 increment = 1;

4

5 create or replace TABLE RUN\_SUMMARY (

6 PARTY\_ID VARCHAR(15),

7 RUN\_ID NUMBER(18,0),

8 INPUT\_TABLE VARCHAR(100),

9 OUTPUT\_TABLE VARCHAR(150),

10 OUTPUT\_TABLE\_ENCRYPTED VARCHAR(150),

11 START\_TIME TIMESTAMP\_LTZ(9),

12 END\_TIME TIMESTAMP\_LTZ(9)

13 );

14

15 create or replace TABLE PROCESS\_SUMMARY (

16 PARTY\_ID NUMBER(18,0),

17 RUN\_ID NUMBER(18,0),

18 TOTAL\_RECORDS NUMBER(18,0),

19 NAME\_AND\_ADDRESS NUMBER(13,0),

20 NAME\_AND\_EMAIL NUMBER(13,0),

21 LASTNAME\_AND\_ADDRESS NUMBER(13,0),

22 LASTNAME\_AND\_EMAIL NUMBER(13,0),

23 NONAME\_EMAIL NUMBER(13,0),

24 NOMATCH NUMBER(13,0)

25 );

26

27 create or replace TABLE PROCESS\_STATISTICS (

28 PARTY\_ID VARCHAR(15),

29 RUN\_ID NUMBER(18,0),

30 FIRSTNAME VARCHAR(1),

31 LASTNAME VARCHAR(1),

32 POSTALADDRESS VARCHAR(1),

33 EMAILID VARCHAR(1),

34 PHONE VARCHAR(1),

35 MAID VARCHAR(1),

36 IP VARCHAR(1),

37 MATCHTYPE VARCHAR(13),

38 QUANTITY NUMBER(18,0)

39 );

40

41 create or replace TABLE ERROR\_LOG (

42 PARTY\_ID VARCHAR(15),

43 RUN\_ID NUMBER(18,0),

44 INPUT\_TABLE VARCHAR(100),

45 ERROR\_MESSAGE VARCHAR

46 );

47

48 -- stored procedure -- proc

49 CREATE OR REPLACE PROCEDURE create\_customer\_view(CLIENT\_DB VARCHAR, CLIENT\_SCHEMA

VARCHAR, IDENTITY\_GRAPH VARCHAR, CLIENT\_TABLE VARCHAR, CUSTOMER\_ID VARCHAR, RUNID

VARCHAR)

50 returns string not null

51 language javascript

52 execute as owner

53 AS

54 $$

55 var result = "";

56 var queryText = "";

57

58 var qry\_run\_summary\_1 = `INSERT INTO ${CLIENT\_DB}.${CLIENT\_SCHEMA}.RUN\_SUMMARY (

party\_id, run\_id, input\_table, start\_time)

59 SELECT ${CUSTOMER\_ID} AS party\_id,

60 ${RUNID} AS run\_id,

61 '${CLIENT\_TABLE}' AS input\_table,

62 current\_timestamp() AS start\_time

63 FROM dual;`

64

65 var hash\_sql\_command = `CREATE OR REPLACE TABLE ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${

CLIENT\_TABLE}\_HASH\_TABLE AS

66 SELECT a.sequence AS sequence\_number,

67 a.record\_type,

68 CASE

69 WHEN coalesce(b.proper\_name, a.first\_name) IS NULL THEN ''

70 ELSE cast(Zobristhash(Lower(coalesce(b.proper\_name, a.first\_name))) AS

varchar(50))

71 END AS firstname\_hash,

72 CASE

73 WHEN a.last\_name IS NULL THEN ''

74 ELSE cast(Zobristhash(a.last\_name) AS VARCHAR(50))

75 END AS surname\_hash,

76 CASE

77 WHEN a.address\_line\_1 IS NULL THEN ''

78 ELSE cast(Zobristhash(a.address\_line\_1) AS VARCHAR(50))

79 END AS address\_hash,

80 CASE

81 WHEN a.email\_address IS NULL THEN ''

82 ELSE cast(Zobristhash(a.email\_address) AS VARCHAR(50))

83 END AS email\_hash

84 FROM

85 (SELECT sequence,

86 record\_type,

87 upper(Split\_part(fname, ' ', 1))

AS first\_name,

88 lower(concat(substr(lname, 1, 1), '-', namemetaphone(split\_part(

lname, '-', 1)))) AS last\_name,

89 concat(lower(regexp\_replace(addr1, '[^a-zA-Z0-9]', '')), zip)

AS address\_line\_1,

90 lower(regexp\_substr(email, '\\\\w\\\\S+\@[[:alnum:]]+\.[[:alnum:]]+'

)) AS email\_address

91 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${CLIENT\_TABLE}) a

92 LEFT JOIN

93 (SELECT alias\_name,

94 proper\_name

95 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.PROPER\_NAME

96 WHERE c\_order = 1 ) b

97 ON a.first\_name = b.alias\_name;`

98

99 var funnel\_sql\_command\_1 = `CREATE OR REPLACE TABLE ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${

CLIENT\_TABLE}\_FUNNEL\_STEP\_A AS

100 SELECT ${CUSTOMER\_ID} AS party\_id,

101 a.sequence\_number AS sequence\_number,

102 a.record\_type AS record\_type,

103 CASE

104 WHEN length(trim(a.firstname\_hash)) = 0 THEN NULL

105 ELSE a.firstname\_hash

106 END AS hfname,

107 CASE

108 WHEN length(trim(a.surname\_hash)) = 0 THEN NULL

109 ELSE a.surname\_hash

110 END AS hlname,

111 CASE

112 WHEN length(trim(a.address\_hash)) = 0 THEN NULL

113 ELSE a.address\_hash

114 END AS hpostal,

115 CASE

116 WHEN length(trim(a.email\_hash)) = 0 THEN NULL

117 ELSE a.email\_hash

118 END AS hemail,

119 b.p\_id AS hpersonid,

120 b.luid AS hhouseholdid,

121 CASE

122 WHEN b.name\_addr\_email\_hash IS NOT NULL THEN 'Name\_Addr'

123 ELSE 'NoMatch'

124 END AS matchtype

125 FROM

126 (SELECT \*,

127 Concat(address\_hash, surname\_hash, firstname\_hash) AS name\_addr\_hash

128 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${CLIENT\_TABLE}\_HASH\_TABLE) a

129 LEFT JOIN

130 (SELECT p\_id,

131 luid,

132 concat(addresshash, surnamehash, firstnamehash) AS

name\_addr\_email\_hash,

133 row\_number() OVER (PARTITION BY addresshash, surnamehash,

firstnamehash ORDER BY p\_id) AS ind

134 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${IDENTITY\_GRAPH} qualify ind = 1) b

135 ON a.name\_addr\_hash = b.name\_addr\_email\_hash;`

136

137 var funnel\_sql\_command\_2 = `CREATE OR REPLACE TABLE ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${

CLIENT\_TABLE}\_FUNNEL\_STEP\_B AS

138 SELECT ${CUSTOMER\_ID} AS party\_id,

139 a.sequence\_number AS sequence\_number,

140 a.record\_type AS record\_type,

141 a.hfname AS hfname,

142 a.hlname AS hlname,

143 a.hpostal AS hpostal,

144 a.hemail AS hemail,

145 b.p\_id AS hpersonid,

146 b.luid AS hhouseholdid,

147 CASE

148 WHEN b.name\_addr\_email\_hash IS NOT NULL THEN 'Name\_Email'

149 ELSE 'NoMatch'

150 END AS matchtype

151 FROM

152 (SELECT \*,

153 concat(hemail, hlname, hfname) AS name\_email\_hash

154 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${CLIENT\_TABLE}\_FUNNEL\_STEP\_A

155 WHERE matchtype = 'NoMatch') a

156 LEFT JOIN

157 (SELECT p\_id,

158 luid,

159 concat(addresshash, surnamehash, firstnamehash) AS

name\_addr\_email\_hash,

160 row\_number() OVER (PARTITION BY addresshash, surnamehash,

firstnamehash ORDER BY p\_id) AS ind

161 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${IDENTITY\_GRAPH} qualify ind = 1) b

162 ON a.name\_email\_hash = b.name\_addr\_email\_hash;`

163

164 var funnel\_sql\_command\_3 = `CREATE OR REPLACE TABLE ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${

CLIENT\_TABLE}\_FUNNEL\_STEP\_C AS

165 SELECT ${CUSTOMER\_ID} AS party\_id,

166 a.sequence\_number AS sequence\_number,

167 a.record\_type AS record\_type,

168 a.hfname AS hfname,

169 a.hlname AS hlname,

170 a.hpostal AS hpostal,

171 a.hemail AS hemail,

172 NULL AS hpersonid,

173 b.luid AS hhouseholdid,

174 CASE

175 WHEN b.surname\_addr\_email\_hash IS NOT NULL THEN 'Surname\_Addr'

176 ELSE 'NoMatch'

177 END AS matchtype

178 FROM

179 (SELECT \*,

180 concat(hpostal, hlname) AS surname\_addr\_hash

181 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${CLIENT\_TABLE}\_FUNNEL\_STEP\_B

182 WHERE matchtype = 'NoMatch') a

183 LEFT JOIN

184 (SELECT luid,

185 concat(addresshash, surnamehash) AS surname\_addr\_email\_hash,

186 row\_number() OVER (PARTITION BY addresshash, surnamehash ORDER BY

luid) AS ind

187 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${IDENTITY\_GRAPH} qualify ind = 1) b

188 ON a.surname\_addr\_hash = b.surname\_addr\_email\_hash;`

189

190 var funnel\_sql\_command\_4 = `CREATE OR REPLACE TABLE ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${

CLIENT\_TABLE}\_FUNNEL\_STEP\_D AS

191 SELECT ${CUSTOMER\_ID} AS party\_id,

192 a.sequence\_number AS sequence\_number,

193 a.record\_type AS record\_type,

194 a.hfname AS hfname,

195 a.hlname AS hlname,

196 a.hpostal AS hpostal,

197 a.hemail AS hemail,

198 NULL AS hpersonid,

199 b.luid AS hhouseholdid,

200 CASE

201 WHEN b.surname\_addr\_email\_hash IS NOT NULL THEN 'Surname\_Email'

202 ELSE 'NoMatch'

203 END AS matchtype

204 FROM

205 (SELECT \*,

206 concat(hemail, hlname) AS surname\_email\_hash

207 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${CLIENT\_TABLE}\_FUNNEL\_STEP\_C

208 WHERE matchtype = 'NoMatch') a

209 LEFT JOIN

210 (SELECT luid,

211 concat(addresshash, surnamehash) AS surname\_addr\_email\_hash,

212 row\_number() OVER (PARTITION BY addresshash, surnamehash ORDER BY

luid) AS ind

213 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${IDENTITY\_GRAPH} qualify ind = 1) b

214 ON a.surname\_email\_hash = b.surname\_addr\_email\_hash;`

215

216 var funnel\_sql\_command\_5 = `CREATE OR REPLACE TABLE ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${

CLIENT\_TABLE}\_FUNNEL\_STEP\_E AS

217 SELECT ${CUSTOMER\_ID} AS party\_id,

218 a.sequence\_number AS sequence\_number,

219 a.record\_type AS record\_type,

220 a.hfname AS hfname,

221 a.hlname AS hlname,

222 a.hpostal AS hpostal,

223 a.hemail AS hemail,

224 b.p\_id AS hpersonid,

225 b.luid AS hhouseholdid,

226 CASE

227 WHEN b.noname\_email\_hash IS NOT NULL THEN 'NOname\_Email'

228 ELSE 'NoMatch'

229 END AS matchtype

230 FROM

231 (SELECT \*,

232 hemail AS noname\_email\_hash

233 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${CLIENT\_TABLE}\_FUNNEL\_STEP\_D

234 WHERE matchtype = 'NoMatch') a

235 LEFT JOIN

236 (SELECT p\_id,

237 luid,

238 addresshash AS noname\_email\_hash,

239 row\_number() OVER (PARTITION BY addresshash ORDER BY p\_id) AS ind

240 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${IDENTITY\_GRAPH} qualify ind = 1) b

241 ON a.noname\_email\_hash = b.noname\_email\_hash;`

242

243 var funnel\_sql\_command\_union = `CREATE OR REPLACE TABLE ${CLIENT\_DB}.${CLIENT\_SCHEMA}.

${CLIENT\_TABLE}\_CUSTOMER\_VIEW AS

244 (

245 SELECT party\_id,

246 sequence\_number AS sequence,

247 record\_type AS record\_type,

248 cast(hfname AS VARCHAR(50)) AS hfname,

249 cast(hlname AS VARCHAR(50)) AS hlname,

250 cast(hpostal AS VARCHAR(50)) AS hpostal,

251 cast(hemail AS VARCHAR(50)) AS hemail,

252 cast(hpersonid AS VARCHAR(50)) AS hpersonid,

253 cast(hhouseholdid AS VARCHAR(50)) AS hhouseholdid,

254 matchtype

255 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${CLIENT\_TABLE}\_FUNNEL\_STEP\_A WHERE

matchtype = 'Name\_Addr'

256 UNION ALL

257 SELECT party\_id,

258 sequence\_number AS sequence,

259 record\_type AS record\_type,

260 cast(hfname AS VARCHAR(50)) AS hfname,

261 cast(hlname AS VARCHAR(50)) AS hlname,

262 cast(hpostal AS VARCHAR(50)) AS hpostal,

263 cast(hemail AS VARCHAR(50)) AS hemail,

264 cast(hpersonid AS VARCHAR(50)) AS hpersonid,

265 cast(hhouseholdid AS VARCHAR(50)) AS hhouseholdid,

266 matchtype

267 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${CLIENT\_TABLE}\_FUNNEL\_STEP\_B WHERE

matchtype = 'Name\_Email'

268 UNION ALL

269 SELECT party\_id,

270 sequence\_number AS sequence,

271 record\_type AS record\_type,

272 cast(hfname AS VARCHAR(50)) AS hfname,

273 cast(hlname AS VARCHAR(50)) AS hlname,

274 cast(hpostal AS VARCHAR(50)) AS hpostal,

275 cast(hemail AS VARCHAR(50)) AS hemail,

276 cast(hpersonid AS VARCHAR(50)) AS hpersonid,

277 cast(hhouseholdid AS VARCHAR(50)) AS hhouseholdid,

278 matchtype

279 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${CLIENT\_TABLE}\_FUNNEL\_STEP\_C WHERE

matchtype = 'Surname\_Addr'

280 UNION ALL

281 SELECT party\_id,

282 sequence\_number AS sequence,

283 record\_type AS record\_type,

284 cast(hfname AS VARCHAR(50)) AS hfname,

285 cast(hlname AS VARCHAR(50)) AS hlname,

286 cast(hpostal AS VARCHAR(50)) AS hpostal,

287 cast(hemail AS VARCHAR(50)) AS hemail,

288 cast(hpersonid AS VARCHAR(50)) AS hpersonid,

289 cast(hhouseholdid AS VARCHAR(50)) AS hhouseholdid,

290 matchtype

291 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${CLIENT\_TABLE}\_FUNNEL\_STEP\_D WHERE

matchtype = 'Surname\_Email'

292 UNION ALL

293 SELECT party\_id,

294 sequence\_number AS sequence,

295 record\_type AS record\_type,

296 cast(hfname AS VARCHAR(50)) AS hfname,

297 cast(hlname AS VARCHAR(50)) AS hlname,

298 cast(hpostal AS VARCHAR(50)) AS hpostal,

299 cast(hemail AS VARCHAR(50)) AS hemail,

300 cast(hpersonid AS VARCHAR(50)) AS hpersonid,

301 cast(hhouseholdid AS VARCHAR(50)) AS hhouseholdid,

302 matchtype

303 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${CLIENT\_TABLE}\_FUNNEL\_STEP\_E

304 );`

305

306 var drop\_sql\_command\_1 = `DROP TABLE IF EXISTS ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${

CLIENT\_TABLE}\_FUNNEL\_STEP\_A;`

307 var drop\_sql\_command\_2 = `DROP TABLE IF EXISTS ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${

CLIENT\_TABLE}\_FUNNEL\_STEP\_B;`

308 var drop\_sql\_command\_3 = `DROP TABLE IF EXISTS ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${

CLIENT\_TABLE}\_FUNNEL\_STEP\_C;`

309 var drop\_sql\_command\_4 = `DROP TABLE IF EXISTS ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${

CLIENT\_TABLE}\_FUNNEL\_STEP\_D;`

310 var drop\_sql\_command\_5 = `DROP TABLE IF EXISTS ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${

CLIENT\_TABLE}\_FUNNEL\_STEP\_E;`

311 var drop\_sql\_command\_6 = `DROP TABLE IF EXISTS ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${

CLIENT\_TABLE}\_HASH\_TABLE;`

312

313 var sql\_process\_summary = `INSERT INTO ${CLIENT\_DB}.${CLIENT\_SCHEMA}.PROCESS\_SUMMARY

314 SELECT party\_id,

315 ${RUNID} AS run\_id,

316 COUNT(\*) AS total\_records,

317 SUM(CASE WHEN matchtype = 'Name\_Addr' THEN 1 ELSE 0 END) AS

Name\_And\_Address,

318 SUM(CASE WHEN matchtype = 'Name\_Email' THEN 1 ELSE 0 END) AS

Name\_And\_Email,

319 SUM(CASE WHEN matchtype = 'Surname\_Addr' THEN 1 ELSE 0 END) AS

LastName\_And\_Address,

320 SUM(CASE WHEN matchtype = 'Surname\_Email' THEN 1 ELSE 0 END) AS

LastName\_And\_Email,

321 SUM(CASE WHEN matchtype = 'NOname\_Email' THEN 1 ELSE 0 END) AS

Noname\_Email,

322 SUM(CASE WHEN matchtype = 'NoMatch' THEN 1 ELSE 0 END) AS Nomatch

323 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${CLIENT\_TABLE}\_CUSTOMER\_VIEW

324 GROUP BY 1;`

325

326 var sql\_process\_statistics = `INSERT INTO ${CLIENT\_DB}.${CLIENT\_SCHEMA}.

PROCESS\_STATISTICS

327 (

328 party\_id,

329 run\_id,

330 firstname,

331 lastname,

332 postaladdress,

333 emailid,

334 matchtype,

335 quantity

336 )

337 SELECT party\_id,

338 ${RUNID} AS run\_id,

339 CASE WHEN Hfname is not null THEN 'V' ELSE 'B' END AS Firstname,

340 CASE WHEN Hlname is not null THEN 'V' ELSE 'B' END AS Lastname,

341 CASE WHEN Hpostal is not null THEN 'V' ELSE 'B' END AS PostalAddress,

342 CASE WHEN Hemail is not null THEN 'V' ELSE 'B' END AS EmailID,

343 matchtype,

344 count(matchtype) AS Quantity

345 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${CLIENT\_TABLE}\_CUSTOMER\_VIEW

346 GROUP BY 1, 2, 3, 4, 5, 6, 7

347 ORDER BY matchtype;`

348

349 var encrypt\_sql\_command = `CREATE OR REPLACE TABLE ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${

CLIENT\_TABLE}\_CUSTOMER\_VIEW\_ENCRYPTED AS

350 SELECT party\_id,

351 sequence AS sequence,

352 record\_type,

353 encrypt(Hfname, concat(${CUSTOMER\_ID}, reverse(${CUSTOMER\_ID}))) AS

Hfname,

354 encrypt(Hlname, concat(${CUSTOMER\_ID}, reverse(${CUSTOMER\_ID}))) AS

Hlname,

355 encrypt(Hpostal, concat(${CUSTOMER\_ID}, reverse(${CUSTOMER\_ID}))) AS

Hpostal,

356 encrypt(Hemail, concat(${CUSTOMER\_ID}, reverse(${CUSTOMER\_ID}))) AS

Hemail,

357 encrypt(HpersonId, concat(${CUSTOMER\_ID}, reverse(${CUSTOMER\_ID}))) AS

HpersonId,

358 encrypt(HhouseholdId, concat(${CUSTOMER\_ID}, reverse(${CUSTOMER\_ID}))) AS

HhouseholdId,

359 matchtype

360 FROM ${CLIENT\_DB}.${CLIENT\_SCHEMA}.${CLIENT\_TABLE}\_CUSTOMER\_VIEW;`

361

362 var qry\_run\_summary\_2 = `UPDATE ${CLIENT\_DB}.${CLIENT\_SCHEMA}.RUN\_SUMMARY

363 SET OUTPUT\_TABLE = '${CLIENT\_TABLE}\_CUSTOMER\_VIEW' ,

364 OUTPUT\_TABLE\_ENCRYPTED = '${CLIENT\_TABLE}\_CUSTOMER\_VIEW\_ENCRYPTED' ,

365 END\_TIME = current\_timestamp()

366 WHERE run\_id = ${RUNID}

367 AND Party\_id = ${CUSTOMER\_ID};`

368

369 // error\_log

370 var qry\_error\_log=`INSERT INTO ${CLIENT\_DB}.${CLIENT\_SCHEMA}.ERROR\_LOG values (${

CUSTOMER\_ID}, ${RUNID}, '${CLIENT\_TABLE}', :1);`

371

372 var qry\_statement = snowflake.createStatement( {sqlText: qry\_run\_summary\_1} );

373 try

374 {

375 queryText = qry\_statement.getSqlText();

376 qry\_statement.execute(); // Update start timestamp

377

378 // create hash table

379 var qry\_statement = snowflake.createStatement( {sqlText: hash\_sql\_command} );

380 queryText = qry\_statement.getSqlText();

381 qry\_statement.execute();

382

383 // create funnel tables

384 var qry\_statement = snowflake.createStatement( {sqlText: funnel\_sql\_command\_1} );

385 queryText = qry\_statement.getSqlText();

386 qry\_statement.execute(); // create funnel\_table\_1

387

388 var qry\_statement = snowflake.createStatement( {sqlText: funnel\_sql\_command\_2} );

389 queryText = qry\_statement.getSqlText();

390 qry\_statement.execute(); // create funnel\_table\_2

391

392 var qry\_statement = snowflake.createStatement( {sqlText: funnel\_sql\_command\_3} );

393 queryText = qry\_statement.getSqlText();

394 qry\_statement.execute(); // create funnel\_table\_3

395

396 var qry\_statement = snowflake.createStatement( {sqlText: funnel\_sql\_command\_4} );

397 queryText = qry\_statement.getSqlText();

398 qry\_statement.execute(); // create funnel\_table\_4

399

400 var qry\_statement = snowflake.createStatement( {sqlText: funnel\_sql\_command\_5} );

401 queryText = qry\_statement.getSqlText();

402 qry\_statement.execute(); // create funnel\_table\_5

403

404 // create a single customer view table by combining all the funnel tables

405 var qry\_statement = snowflake.createStatement( {sqlText: funnel\_sql\_command\_union} );

406 queryText = qry\_statement.getSqlText();

407 qry\_statement.execute();

408

409 // drop all temporary funnel tables

410 var qry\_statement = snowflake.createStatement( {sqlText: drop\_sql\_command\_1} );

411 queryText = qry\_statement.getSqlText();

412 qry\_statement.execute();

413

414 var qry\_statement = snowflake.createStatement( {sqlText: drop\_sql\_command\_2} );

415 queryText = qry\_statement.getSqlText();

416 qry\_statement.execute();

417

418 var qry\_statement = snowflake.createStatement( {sqlText: drop\_sql\_command\_3} );

419 queryText = qry\_statement.getSqlText();

420 qry\_statement.execute();

421

422 var qry\_statement = snowflake.createStatement( {sqlText: drop\_sql\_command\_4} );

423 queryText = qry\_statement.getSqlText();

424 qry\_statement.execute();

425

426 var qry\_statement = snowflake.createStatement( {sqlText: drop\_sql\_command\_5} );

427 queryText = qry\_statement.getSqlText();

428 qry\_statement.execute();

429

430 var qry\_statement = snowflake.createStatement( {sqlText: drop\_sql\_command\_6} );

431 queryText = qry\_statement.getSqlText();

432 qry\_statement.execute();

433

434 // create process summary table

435 var qry\_statement = snowflake.createStatement( {sqlText: sql\_process\_summary} );

436 queryText = qry\_statement.getSqlText();

437 qry\_statement.execute();

438

439 // create process statistics table

440 var qry\_statement = snowflake.createStatement( {sqlText: sql\_process\_statistics} );

441 queryText = qry\_statement.getSqlText();

442 qry\_statement.execute();

443

444 // encrypt the customer view table

445 var qry\_statement = snowflake.createStatement( {sqlText: encrypt\_sql\_command} );

446 queryText = qry\_statement.getSqlText();

447 qry\_statement.execute();

448

449 // update run-summary with end timestamp

450 var qry\_statement = snowflake.createStatement( {sqlText: qry\_run\_summary\_2} );

451 queryText = qry\_statement.getSqlText();

452 qry\_statement.execute();

453

454 result = "Succeeded"; // Return a success/error indicator

455 }

456 catch (err) // Return a success/error indicator

457 {

458 result = "Failed:\n Error Code: " + err.code + "\n Error State: " + err.state;

459 result += "\n Error Message: " + err.message;

460 result += "\n Stack Trace: " + err.stackTraceTxt;

461 result += "\n Query Text: \n" + queryText;

462

463 // insert into error\_log table

464 var qry\_statement = snowflake.createStatement( {sqlText: qry\_error\_log, binds:[

result]} ).execute();

465 }

466

467 return result;

468 $$;

469

470 -- Sample SP call

471

472 call create\_customer\_view('IDENTITY\_SERVICE', 'IDENTITY', 'CCIRFULLDUMMYFILE\_IAAS\_HASH',

'CCIRFULL\_MULTI\_CLIENTA', '3811', (select run\_id.nextval));

473 call create\_customer\_view('IDENTITY\_SERVICE', 'IDENTITY', 'CCIRFULLDUMMYFILE\_IAAS\_HASH',

'CCIRFULL\_MULTI\_CLIENTB', '3812', (select run\_id.nextval));

474

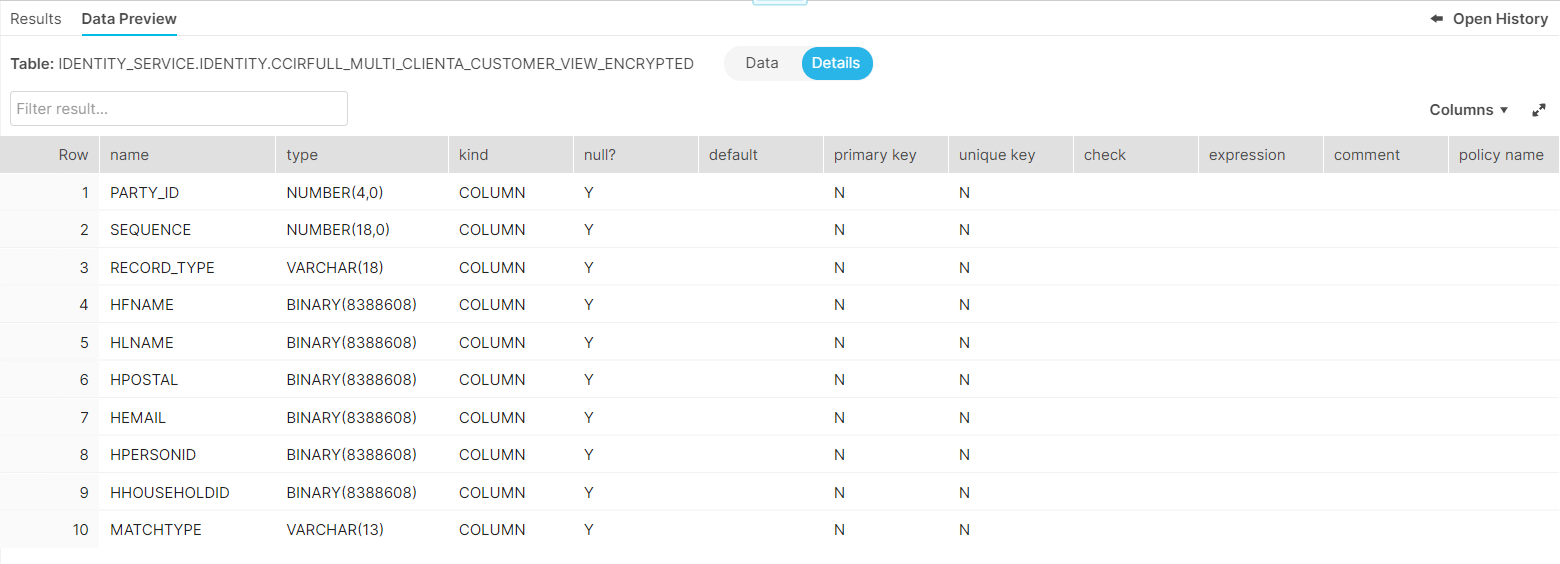
475

# Client Secure View and Encryption

When the Experian Identity key assignment is done, the hashed Primary keys, and their respective PID and/or LUID are loaded into Client Secure View (${CLIENT\_TABLE}\_CUSTOMER\_VIEW). Post that, they are encrypted as per the Client specific salt key and shared back to the respective Client along with the Party Id or Customer Id.

Below is the schema details which are being sent back to the client as encrypted data.

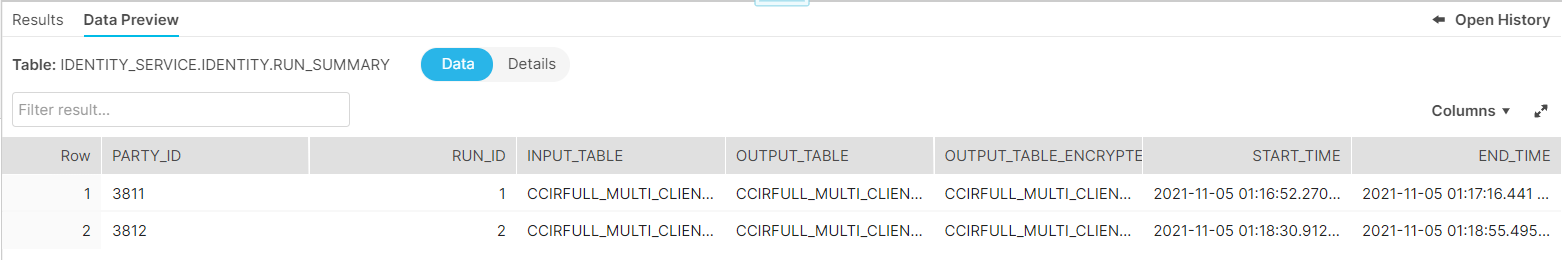
(${CLIENT\_TABLE}\_CUSTOMER\_VIEW\_ENCRYPTED).



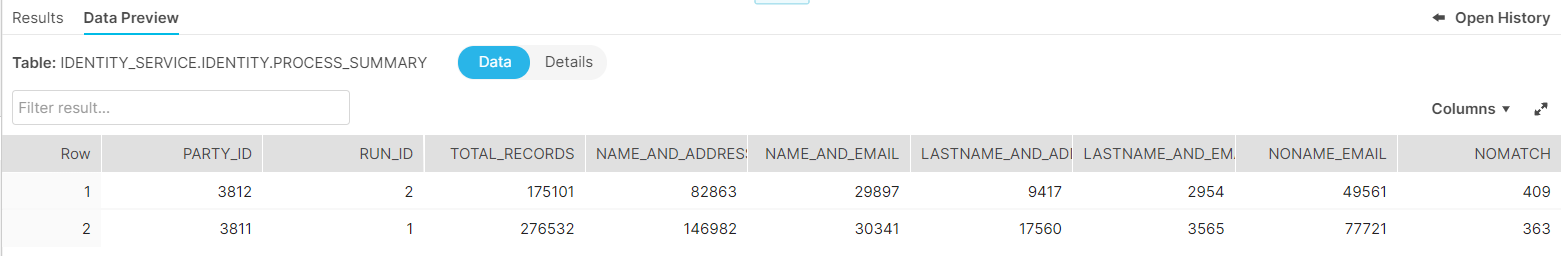
# Tracking and Billing

In Order to get the statistics of the data volume which we received from the Client & Statistics, three tables are created (RUN\_SUMMARY, PROCESS\_SUMMARY, PROCESS\_STATISTICS) to track the total number of records based on the match type with the record insertion timestamp.

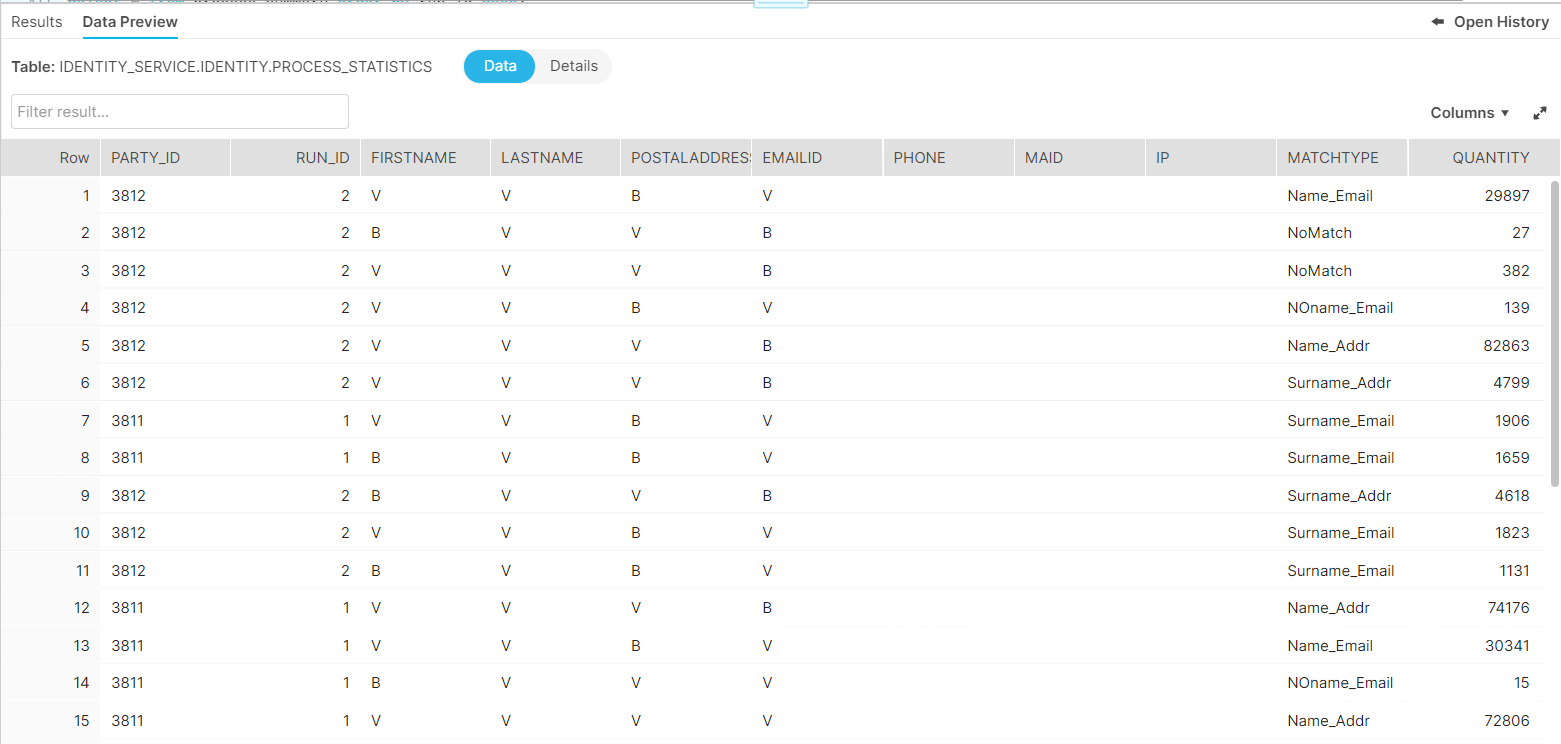
**RUN\_SUMMARY**



**PROCESS\_SUMMARY**



**PROCESS\_STATISTICS**



**ERROR\_LOG**

