TEST DATA MASKING
FOR Db2 on z/OS
Everybody needs to do masking, but getting it done is hard.
The challenge is to change the data in a way so that:

- You cannot derive the original data from the masked data
- The masked data looks plausible and will pass validity checks
  - Credit card numbers, IBANs, SINs have validity checks
  - SSNs cannot have all-zero in any group
  - Street name, postal code / ZIP code and city depend on each other
- The masked data does not violate database constraints
  - Unique constraints / Referential integrity constraints
• Don't just set everything to NULL or to XXX
  Cannot be used if unique constraints or RI constraints exist
  Does not give application anything plausible to work with

• Don't just shift each digit and letter
  Usually does not violate unique or foreign key constraints, but:
  Insecure, original value can be reconstructed easily
  Can result in invalid data (E.g. credit card number with invalid check digit)
• Don’t replace data with random values
  Masking needs to be repeatable
  Random values will change every time you mask
  Random values typically violate constraints
• Hash-based masking produces good results
  Derive your masked values from your source values
  Any conceivable input value can be processed
  Similar input values result in totally different hash values
  Can be designed to be practically non-invertible
  Can use hash values as lookup table index

• Alternatively: Mapping tables
  Need to be refreshed periodically
Hash-based Masking
**HASH-BASED MASKING**

**Masking function**

**Input values**
- Hash function
- Generator function

**Output values**

**Input values**
- Hash function
- Lookup function

**Output values**

**Lookup table**
(names, addresses, etc.)
• **Generator functions:**

  When the target value can be calculated based on the hash alone

  Plain numbers
  SSNs / SINs
  License plate numbers
  Data / Time values
  Credit card numbers
  UUIDs
**Lookup functions:**

When the target value needs to be from a list of valid values
- First name, last name
- Address
- Banking information
- Combination of the above
Masking primary and unique keys

- Hash functions are, by definition, not collision free
- This can lead to duplicate values in columns declared as unique
- BCV5 uses a fast hashing algorithm for INTEGER values that is collision free between 0 and 2,147,483,647 (= $2^{31} - 1$)
What to consider when masking data?
### WHAT TO_consider

- Use the current field value or a different column (e.g. ID) as seed?
- Need identical masked addresses if the source address are the same?
### What to Consider

- Use the current field value or a different column (e.g. ID) as seed?
- Need identical masked addresses if the source address are the same?
In a perfect world, everything would be at least 3NF

- There's only one table that contains names
- There's only one table that contains addresses
- There are never any typos
- Phone numbers are always stored as +12223334444
  Never as 222(333)4444 or 222-333-4444 (after 6pm call 555-6666)
- Everything is linked together through single-column primary keys
- No redundancy

But that's not how real databases look like!
<table>
<thead>
<tr>
<th>FIRST_NAME</th>
<th>LAST_NAME</th>
<th>DOB</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANNIE</td>
<td>MILLER</td>
<td>4171977</td>
<td>123-45-6789</td>
</tr>
<tr>
<td>GEORGE</td>
<td>WILSON</td>
<td>10211982</td>
<td>234-56-7890</td>
</tr>
<tr>
<td>MELISSA</td>
<td>JONSON</td>
<td>6011964</td>
<td>012-34-5678</td>
</tr>
<tr>
<td>FREDERICK</td>
<td>BROWN</td>
<td>6281967</td>
<td>001-23-4567</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FIRST_NAME</th>
<th>LAST_NAME</th>
<th>DOB</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANNIE B.</td>
<td>MILLER</td>
<td>4/17/1977</td>
<td>123456789</td>
</tr>
<tr>
<td>GEORGE F.</td>
<td>WILSON</td>
<td>10/21/1982</td>
<td>234567890</td>
</tr>
<tr>
<td>MELISSA K.</td>
<td>JOHNSON</td>
<td>6/1/1964</td>
<td>12345678</td>
</tr>
<tr>
<td>FREDERICK I.</td>
<td>BROWN</td>
<td>6/28/1967</td>
<td>1234567</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>DOB</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>MILLER, ANNIE</td>
<td>4/17/1977</td>
<td>123-45-6789</td>
</tr>
<tr>
<td>WILSON, GEORGE</td>
<td>10/21/1982</td>
<td>234-56-7890</td>
</tr>
<tr>
<td>JOHNSON, MELISSA</td>
<td>6/1/1964</td>
<td>012-34-5678</td>
</tr>
<tr>
<td>BROWN, FREDERICK</td>
<td>6/28/1967</td>
<td>001-23-4567</td>
</tr>
</tbody>
</table>

WHAT TO CONSIDER: **GOTCHAS**
WHAT TO CONSIDER
DATA STANDARDIZATION

If we want to hash our actual source names, source addresses, etc., we need to tidy it up

• Technically

Remove leading and trailing blanks
Cast numeric data to actual numeric data type if stored in text column
If processing outside of Db2: Convert to common code page
If we want to hash our actual source names, source addresses, etc., we need to tidy it up

- Functionally
  Truncate to shortest representation that actually exists in the database

String normalization: Remove diacritics, spaces and punctuation

Can you guarantee that all of these will be identically after masking?

Does your application care?

St. Louis, Missouri
St. Louis (MO)
St. Louis, MO
St. Louis / MO
St. Louis, MO
St Louis MO
St Louis MO
StLouis MO
St. Louis
Saint Louis
St. Luis
Getting consistent masking
• Sometimes valid values for a column depend on other columns

Most prominent example: address

• Street, city, zip code and state are dependent on each other

<table>
<thead>
<tr>
<th>ID</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3750982</td>
<td>1100 Congress Ave</td>
<td>Austin</td>
<td>TX</td>
<td>78701</td>
<td>US</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
- Address masking uses lookup tables

<table>
<thead>
<tr>
<th>ID</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>500 N. Church St.</td>
<td>Palestine</td>
<td>TX</td>
<td>75081</td>
</tr>
<tr>
<td>2</td>
<td>215 East Lufkin Avenue</td>
<td>Lufkin</td>
<td>TX</td>
<td>75902</td>
</tr>
<tr>
<td>3</td>
<td>2840 TX-35 BUS</td>
<td>Rockport</td>
<td>TX</td>
<td>78382</td>
</tr>
<tr>
<td>4</td>
<td>100 S Center St.</td>
<td>Archer City</td>
<td>TX</td>
<td>76351</td>
</tr>
<tr>
<td>5</td>
<td>100 Trice Street</td>
<td>Claude</td>
<td>TX</td>
<td>79019</td>
</tr>
<tr>
<td>6</td>
<td>1 Courthouse Circle Dr.</td>
<td>Jourdanton</td>
<td>TX</td>
<td>78026</td>
</tr>
<tr>
<td>7</td>
<td>1 East Main Street</td>
<td>Bellville</td>
<td>TX</td>
<td>77418</td>
</tr>
<tr>
<td>8</td>
<td>300 S 1st St.</td>
<td>Muleshoe</td>
<td>TX</td>
<td>79347</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
## WHAT TO CONSIDER

### DATA STANDARDIZATION | CONSISTENT DATA

<table>
<thead>
<tr>
<th>ID</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3750982</td>
<td>1100 Congress Ave</td>
<td>Austin</td>
<td>TX</td>
<td>78701</td>
<td>US</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

---

**MASKING**

**MASK_STREET**

**MASK_CITY**

**MASK_STATE**

**MASK_ZIP**

---

<table>
<thead>
<tr>
<th>ID</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3750982</td>
<td>3750982</td>
<td></td>
<td></td>
<td></td>
<td>US</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### WHAT TO CONSIDER

**DATA STANDARDIZATION | CONSISTENT DATA**

<table>
<thead>
<tr>
<th>ID</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3750982</td>
<td>1100 Congress Ave</td>
<td>Austin</td>
<td>TX</td>
<td>78701</td>
<td>US</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### Hash input

- **Hash input ↓ 46340**
  - Get street from row 46340

- **Hash input ↓ 103873**
  - Get city from row 103873

- **Hash input ↓ 86233**
  - Get state from row 86233

- **Hash input ↓ 58237**
  - Get zip from row 58237

#### Address lookup table
### WHAT TO CONSIDER

#### DATA STANDARDIZATION | CONSISTENT DATA

<table>
<thead>
<tr>
<th>ID</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3750982</td>
<td>1100 Congress Ave</td>
<td>Austin</td>
<td>TX</td>
<td>78701</td>
<td>US</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**MASKING:**
- MASK_STREET
- MASK_CITY
- MASK_STATE
- MASK_ZIP

<table>
<thead>
<tr>
<th>ID</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3750982</td>
<td>6233 Hollywood Blvd</td>
<td>New York City</td>
<td>FL</td>
<td>60622</td>
<td>US</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
• Use the same input value when hashing different columns with related info
• Good candidate: ID column, any other primary key
• Or concatenate all related columns
  
  Street column:  MASK_STREET(STREET || CITY || ZIP || STATE)
  City column:  MASK_CITY (STREET || CITY || ZIP || STATE)
  Zip column:  MASK_ZIP (STREET || CITY || ZIP || STATE)
  State column:  MASK_STATE (STREET || CITY || ZIP || STATE)
## WHAT TO CONSIDER

### DATA STANDARDIZATION | CONSISTENT DATA

<table>
<thead>
<tr>
<th>ID</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>3750982</td>
<td>1100 Congress Ave</td>
<td>Austin</td>
<td>TX</td>
<td>78701</td>
<td>US</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

**Example:**

- ID: 3750982
- Street: 1100 Congress Ave
- City: Austin
- State: TX
- Zip: 78701
- Country: US

**Masking:**

- **MASK_STREET**
- **MASK_CITY**
- **MASK_STATE**
- **MASK_ZIP**
## WHAT TO CONSIDER

**DATA STANDARDIZATION | CONSISTENT DATA**

<table>
<thead>
<tr>
<th>ID</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3750982</td>
<td>1100 Congress Ave</td>
<td>Austin</td>
<td>TX</td>
<td>78701</td>
<td>US</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **Hash input** ↓ 18925
- **Hash input** ↓ 18925
- **Hash input** ↓ 18925
- **Hash input** ↓ 18925

- **Get street from row 18925**
- **Get city from row 18925**
- **Get state from row 18925**
- **Get zip from row 18925**

**Address lookup table**
## WHAT TO CONSIDER

**DATA STANDARDIZATION | CONSISTENT DATA**

<table>
<thead>
<tr>
<th>ID</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>3750982</td>
<td>1100 Congress Ave</td>
<td>Austin</td>
<td>TX</td>
<td>78701</td>
<td>US</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>316 West Main Street</td>
<td>Lafayette</td>
<td>LA</td>
<td>70501</td>
<td>US</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
Masking names, addresses, banking info

- Hash ID
- Retrieve masked value from lookup table
- Banking information can be a bit tricky
  
  Routing number and bank name from lookup table
  Not all possible account numbers are valid – depends on the bank
  Hash source value to a number between 1 and \((10^{\log(x)}) - 1\)
  where \(x\) is the original bank account number
  Applications rarely check for this
Masking email addresses

- Hash ID
- Retrieve first and last name from lookup table
- Combine first name, last name and domain name
Masking SSNs

- Hash the original SSN to a value between 0 and 999,999,999
- Check for validity
  - No group of digits can be all-zero
  - Area number cannot be 666 or 900 - 999
  - If invalid, re-hash and generate a new number
- Since June 25, 2011 the number is not tied to a location anymore
Masking credit card numbers

- Hash original credit card number to a value between 0 - 9,999,999,999,999,999
  theoretical maximum: 19 digits, in practice 15 or 16 digits
- Keep first $n$ digits
  typically 6 for the Issuer Identification Number
- Calculate check digit using Luhn algorithm
Masking date values

- **Either**
  - Decide on a minimum and maximum date
  - Determine the number of days $x$ between the two dates
  - Hash source value to a number between 0 and $x$
  - Add $x$ to minimum date

- **Or**
  - Hash source value to a number between 0 and 365
  - Add $x$ to the date 01/01/xxxx, where xxxx is the year of the source date
Can you mask any column with PII?

- Applications depend on the PII to varying degrees:
- Changing first name, last name or SSN should never affect the outcome of a test
- Changing the address can have consequences, depending on your industry:
  - Testing a bookkeeping application for web shop: address does not matter
  - Testing premium calculation for flood insurance: address is essential
- You may want to put constraints on the masked data, i.e. stay in zip code area
Additional considerations

- PII may hide in unstructured text fields
- Or in a JPG stored in a BLOB column
- Or in an XML column
- Or in an external file whose path is stored in the database
Implementation

• Consider implementing your algorithms as UDFs
• Functions “live” in Db2
• Fast access to Db2 tables with lookup data
• Can pull masked data from Db2 through DSNTIAUL or other tools
• No unmasked data in flat files
• May be zIIP eligible if called through DDF
Considerations for UDFs

• **Write UDFs in SQL/PL**
  - No external load modules
  - No WLM required, no task switching

• **Performance considerations**
  - Avoid calling other UDFs from within a UDF
  - Avoid recursion
  - Avoid casting
  - Prefer integer processing over string processing
  - Design your UDFs as inline functions
UDF performance comparison

- Inline functions:
  Function consists of a single RETURN statement

- Only use these UDF attributes
  LANGUAGE SQL, SPECIFIC, PARAMETER CCSID, NOT DETERMINISTIC, DETERMINISTIC, NO EXTERNAL ACTION, EXTERNAL ACTION, READS SQL DATA, CONTAINS SQL, CALLED ON NULL INPUT, STATIC DISPATCH

CPU usage for 10 million invocations
Considerations for lookup tables

- Use unique indexes with include columns
  
  Use large index page size to minimize number of levels

- Hash organized table spaces OK, not great

  Also, hash organization will not be supported in future Db2 releases
Masking is not a one-time project

- It is not sufficient to mask data once and keep using the data for a decade
- Changing structures in production force a test data refresh
- Over time, test environments become less useful and "stale"
  - People's age may affect tests
  - Date and timestamp based calculations: Due dates, expiration dates, late fees
  - Masked data needs to be refreshed, too!
- Masking should be a building block in your test data management strategy
Embedding masking in a TDM concept
Several points in the process where you can apply masking:

• Option 1: Create a masked "base" environment

Example: Create a clone of your productions, then mask the data in the cloned environment
EMBEDDING MASKING:
CREATE A MASKED BASE ENVIRONMENT
EMBEDDING MASKING:
CREATE A MASKED BASE ENVIRONMENT
Several points in the process where you can apply masking:

- **Option 1:** Create a masked "base" environment
  
  Example: Create a clone of your productions, then mask the data in the cloned environment

- **Option 2:** Masked data on the fly every time it is copied
  
  Example: Copy directly from production to test, but mask data in-memory while it is being copied
BCV5 TEST DATA MASKING FOR Db2 on z/OS

BCV5 SAVES UP TO 90% IN 3 KEY AREAS:
CREATE A MASKED BASE ENVIRONMENT | MASK DATA ON THE FLY
Several points in the process where you can apply masking:

• Option 1: Create a masked "base" environment
  Example: Create a clone of your productions, then mask the data in the cloned environment

• Option 2: Masked data on the fly every time it is copied
  Example: Copy directly from production to test, but mask data in-memory while it is being copied

• Option 3: Mask transparently upon data access
  Example: Use Db2 column masks to conditionally return modified values for certain users
BCV5 SAVES UP TO 90% IN 3 KEY AREAS:
CREATE A MASKED BASE ENVIRONMENT | MASK DATA ON THE FLY | MASK UPON DATA ACCESS
BCV5 Masking Tool

- Delivers over 30 powerful hashing and data masking functions
- Can be used out of the box
- No customization necessary unless you want to
- Existing hashing & masking functions can also serve as a basis for your algorithms
- Comes with pre-defined lookup tables
  - Millions of names in different languages
  - Millions of banks in different countries
  - Millions of addresses in different countries
Thank you for your attention!

Your UBS Hainer Team

www.ubs-hainer.com
info@ubs-hainersoftware.com