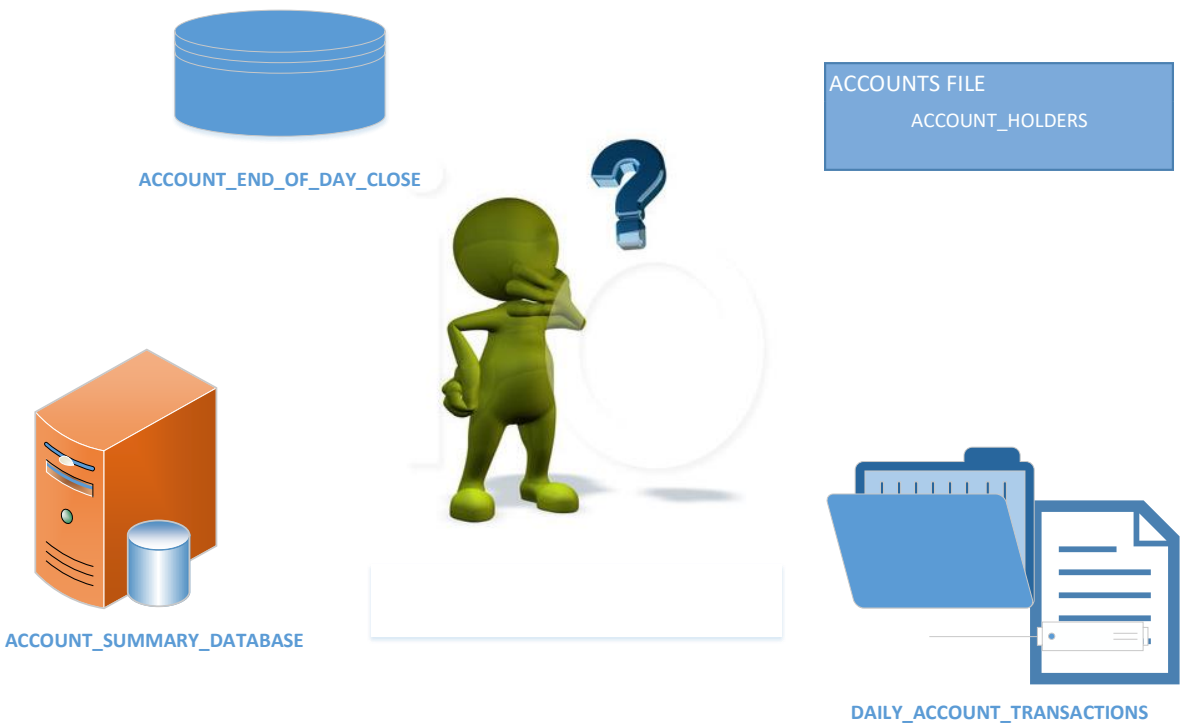
OVERVIEW:

All IT projects, application development, infrastructure upgrade or vendor package release, have one characteristic in common: the need to test. Testing validates that new functionality behaves as expected without negative consequences. Testing also validates that any introduced change will not reduce previously delivered functionality; i.e., functional regression.

The method of delivery may be traditional waterfall, DEV-OPS or Agile methodologies relying on automated or manual processes. Regardless, thorough and accurate testing relies on the availability of data: data reflecting both static data structures, such as Master files or relational databases, as well as altered data structures. Alterations could include a combination of new data fields/columns, tables or in many cases, completely new schema from new designed databases.

How do IT organizations then supply data to match new those business rules, use cases or vendor supplied schema?



HISTORIC APPROACHES:

Organizations have traditionally relied on two options for supplying testing data:

1) Create required data aligned to defined schema
2) Copy data from Production, (hopefully) de-identifying or privatizing data to obfuscate sensitive information (Personally Identifiable Information such as names, addresses, birth dates, credit card numbers, in short, data which could be used by hackers to impersonate identities or expose Protected Health Information).

In creating data aligned to defined schema, each developer or tester creates data stores as necessary to test code or applications. Created data results frequently inject more defects, are limited in the bounds or conditions necessary for proper validation. To be really effective, created data must be truly random, with key attributes constrained by a combination of business rules including field level integrity, table level integrity and relational integrity.

Let's look at a "simple" example:

TRANSACTION_ DATE must be in MMDDYYYY format and be greater than or equal to today and less than or equal to tomorrow

Even spread of TRANSACTION_TYPE of values 1,2,3,4,5,6,7

Even spread of TRANSACTION_SOURCE from Transaction_Source table

ACCOUNT_NUMBER	TRANSACTION_DATE	POSTING_DATE	TRANSACTION_TYPE	TRANSACTION_AMOUNT	TRANSACTION_SOURCE	TRANSACTION_RECEIVER
9446836	10172017	10172017	3	34533.44	EFT	PAYPAL

Require 1,000 transaction records.
Even spread of records to accounts with 50% of transaction records to a single account, remainder spread across other accounts with no more 3 transactions for any one account
ACCOUNT_NUMBER must match to ACCOUNT_HOLDER Master.
Transaction records

POSTING_DATE must be in MMDDYYYY format and be greater than or equal to TRANSACTION_ DATE

Multiple TRANSACTION_AMOUNTs for any one account may not put account balance as negative

TRANSACTION_RECEIVER from Transaction_Teller _Name table if transaction type BANK_DEPOSIT:
From EFT_Source when type EFT
From ATM_Location file when type ATM

In the example above, there are seven separate data fields, with eleven separate business rules, accessing multiple external data schemas and/or tables and files.

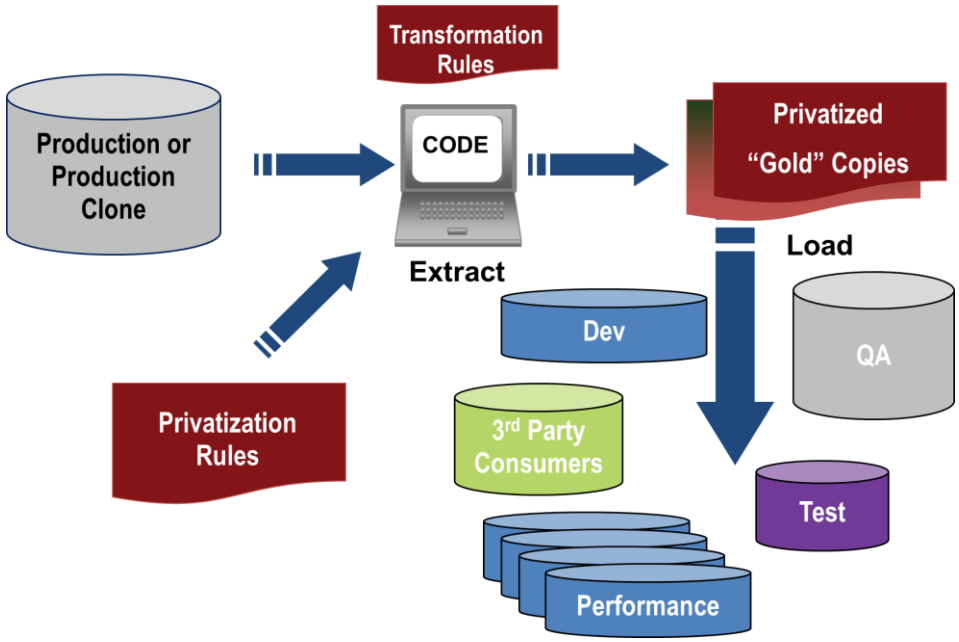
There are numerous opportunities to inject new errors along with the considerable development work required, for the “simple” transaction record. The error and complexity factors increase with more data fields, more complex business rules and more complex data targets (relational databases composed of many tables, for instance).

The second possible approach is extracting data from Production. Some organizations simply copy production to the varied test environments.

Failure to protect customer or patient information by obfuscating the data extract could result in the exposure of sensitive data. In fact, test databases are frequently targeted by intruders because of relatively little security protections and a lack of regular exfiltration monitoring.

Mature organizations combat those dangers by de-identifying data. Replacing PII values with “masked” or encrypted data values, (hopefully) de-identifying or privatizing data to obfuscate sensitive information, the Personally Identifiable Information (PII). PII could include names, addresses, birth dates, credit card numbers, in short, data which could be used by hackers to impersonate identities or expose Protected Health Information (PHI).

Data is extracted from source, ideally a subset of data specific to testing needs, which is then privatized, masking PII values prior to loading into one of many possible testing environments.



When de-identifying, care must be exercised to ensure new assigned data values are contextually accurate. A Social Security Number (SSN) is assigned a new valid SSN value. A Name is correct for a given gender. A credit card number is assigned a new value, capable of passing the card payer rules (VISA has a 4 in position 1, with a 16 digit number; AMEX, a 14 digit number starting with 3) and check-sum validation.

In Example 1 below, data is de-identified to a contextually accurate combination:

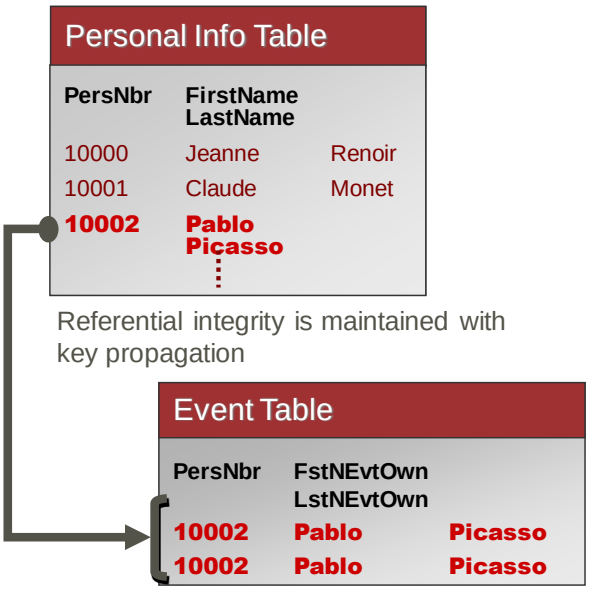
- A valid full name, with feminine first name
- A street address, which is found within the combination of City/State/Zip Code.

In Example 2, two separate tables, in a federated data structure, both containing First and Last Names, are de-identified to new data values obfuscating the actual identity from the production files.

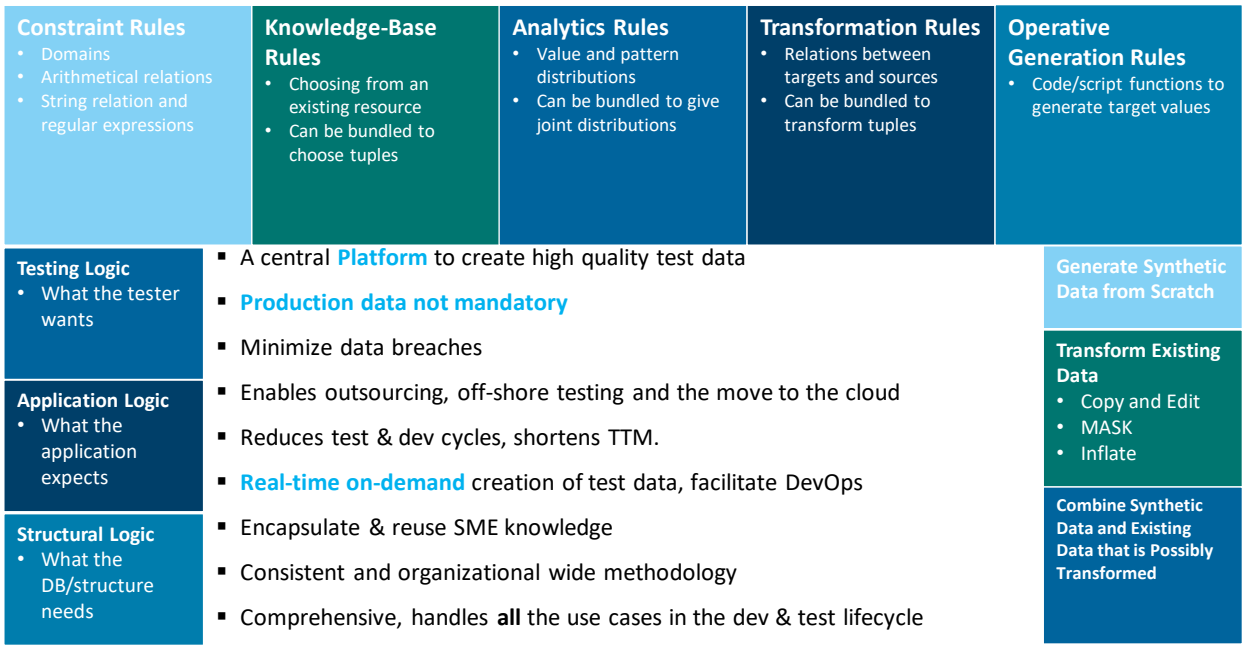
Example 1

Data is masked with contextually correct data to preserve integrity of test data

Example 2



Fabricating data requires the ability to quickly add data to both databases and files; data added in accordance with business rules and schema design:



The complexity of today's schemas make data fabrication more difficult and time consuming, especially if added to each individual developer or tester workload.

In fact, considering the technical demands of extraction, de-identification and fabrication. Organizations increasingly must rely on automation tools expediting the process of creating test data.

PROCESS TOOLING:

Tools exist in the marketplace for rapid solution response:

- Test Data Management (Extract and De-Identification): IBM InfoSphere Optim
- Test Data Fabrication : IBM InfoSphere Test Data Fabricator (TDF)

Optim is a mature product, over 25 years in the marketplace. TDF is a brand-new product, yet, the underlying black box logic (interpreting the business rules and generating random values) is the proprietary Constraint Satisfaction Problem Solver initially developed by IBM supporting the Apollo moon missions of the 1970s.

Constraint satisfaction problems (CSPs) are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations.

Together, the two products are able to satisfy customer testing needs, efficiently and rapidly.

Example Scenario 1:

Customer Need: Customer is testing a new application change, the solution design creating a new schema expanding a current RDBMS with additional tables and columns, satisfying new business process rules.

Solution:

1) Extract and de-identify data into the new schema from Production using IBM Optim.
2) Update the data using TDF, fabricating data into the new defined tables and columns (regression testing existing business rules)
3) Create new records matching the complete schema appending into the database satisfying testing of the new business process rules.

Example Scenario 2:

Customer Need: Customer is replacing core technology solution with a brand new solution developed using an Agile framework.

Solution:

1) As the data schema is designed, TDF loads sample records, following PK/FK constraints
2) Within each successive sprint, as the data design is expanded, the TDF project is expanded with the new target data fields, continually appending into the schema as new business rules are designed.

Example Scenario 3:

Customer Need: Banking customer is regression testing a retail bank application. The desire is to continually age records to match 30/60/90 day processes.

Solution:

1) Extract and de-identify Account and program specific translation tables from Production using IBM Optim.
2) With TDF, customer fabricates aged data, following specific business rules updating database.
3) Alternatively, customer fabricates 30, 60, 90 transaction records updating the database enabling customer to simulate reruns at any point in the 90 day lifecycle.

SUMMARY:

Customer expectations continue to expand as do requirements and speed -to-market on solutions. IT organizations need to continue to improve overall quality delivery methods to meet schedule and quality expectations.

Test Data Fabricator is a welcome tool in the market assisting IT Clients in meeting targets by simplifying the development of test data bases, while reducing the risk of exposure of PII.