SQL select queries have two aspects that can be of a great value for exploring the utility of synthetic data to be utilized in cohort studies. First, a select query may incorporate a filtering clause (e.g. using WHERE) that reflects a specific inclusion-exclusion criterion for a cohort study. Secondly, a select query may aggregate variables in two or more distinct categories e.g. exposed and unexposed subjects. Further, the aggregation function (e.g. SUM, AVG...etc.) may serve as a workload to evaluate the synthetic data. For instance, aggregate queries are used as workloads by (Fan et al. 2020). The generated query is run on both real and synthetic datasets. The query results from both datasets are compared and the relative error is calculated to serve as a utility measure. However, the current studies assume a pre-determined query formulation such as that proposed by (Li et al. 2019) where specific categorical variables are chosen for aggregation. This deterministic approach may lead to biased conclusions. For instance, a generative model may learn the underlying distribution of some variables better than the others. These variables may happen to be the ones used in the pre-defined aggregation functions and result in unrepresentative scores. In addition, this approach requires that the query to be customized to each specific dataset. A priori knowledge of how the synthetic data will be used is also necessary. Typically, a proper solution shall be independent from the dataset selection or usage. To tackle this problem, we are inspired by the randomness introduced by SQL Fuzzing techniques also known as Fuzzers.

SQL Fuzzers are mainly used to test database management systems (DBMSs) for any bugs or vulnerabilities (Jibson 2019). Before executing an SQL query, a DBMS performs two levels of checks. First the SQL statement is checked for any syntactic error such as grammatical errors. Secondly, the query is checked semantically e.g. a call is made to a non-existent table. Once the DBMS performs the necessary checks, the SQL statement is executed according to the best execution plan (Zhong et al. 2020). Fuzzers usually generate large amount of queries that do not pass the aforementioned checks. For instance, American Fuzzy Lop (AFL) (Zalewski n.d.), a widely used fuzzer, has only 30% out of its generated queries passing the syntax check while only 4% can pass the semantic

check (Zhong et al. 2020). Some research attempts are made to focus on finding DBMS logic errors rather than semantic and syntactic errors. However, such techniques generate both the queries and the test datasets (Ghit et al. 2020). Other researchers (Rigger and Su 2020) propose an approach to generate queries that ensure fetching a randomly selected row and hence avoid syntactic and semantic error injection. Clearly, this technique assumes priori knowledge of the dataset.

To ensure unbiased representation of synthetic data utility, we propose a Fuzzy SQL technique that has the following features:

- Datasets may include categorical, continuous and date variables.
- Filtered aggregate queries shall be randomly generated, i.e. grouping may be executed using any combination of the available categorical variables.
- Aggregation may take place across any of the available continuous variables.
- A random condition may be imposed on the aggregate queries. In such case, the values used in the WHERE clause shall be randomly sampled from the real data and equally executed on both the real and synthetic data. - A proper metric shall be established to compare the results from the real and synthetic data.

1 UTILITY EVALUATION

Once a cohort is defined using a random SQL query, it needs to be evaluated. In one approach, (Fan et al. 2020) proposes to arrange the data with one of the categorical variables as a dependent variable. Then two classifiers are trained using training examples from the real and synthetic data respectively. Finally the testing examples from the real data are used to test both models using traditional metric such as F1. The F1 scores for both models are compared. Clearly,

this approach will highly depend on the selection of the dependent variable especially if the dataset mostly incudes categorical variables. In our case, since the randomly generated filtered queries can be aggregated, we use the Hellinger distance to evaluate the utility of cohorts. Whenever a continuous variable is used with an aggregation function, we measure the normalized Euclidean distances between resulting aggregate components in both the real and synthetic data. Details of our proposal and the metric calculations are given in the attached appendix.

2 APPENDIX

Fuzzy SQL is developed to enable the comparison between real and synthetic datasets that are generated by any ML technique. Fuzzy SQL will generate semantically and syntactically correct SELECT random queries that are simultaneously applied to both inputs of real and synthetic datasets. The tool returns all query results along with the corresponding query parameters such as the SQL statement. The query results may be further analyzed to measure distances between real and synthetic responses to each query.

The input datasets may be either tabular of longitudinal.

We define two basic types of random SELECT queries, namely, 'filter' and 'aggregate' queries. We further combine these into a third type and we call it 'filter-aggregate' query. Metrics, such as the Hellinger distance, can be applied to the results of any aggregate query in the same manner they are applied to the original datasets.

If the input data includes continuous variables, various 'aggregate functions' may be randomly applied in addition to counting the resulting number of records. These functions are limited to AVG, SUM, MIN and MAX. If the input does not include any continuous variable, only the COUNT aggregate function is applied. The types of each variable are typically defined by the user and inputted along with the real and synthetic datasets.

3 DATA TYPE OF VARIABLES

To ensure the validity of the SQL select statement as interpreted by the database engine, Fuzzy SQL makes a distinction among three basic data types, namely: Categorical, Continuous and Date. Accordingly, if a dataset includes a variable with a different data type, it will be mapped to the proper type as per the table below:

Input Data Type	Output
'qualitative', 'categorical', 'nominal', 'discrete', 'ordinal', 'dichotomous', 'TEXT', 'INTEGER'	'categori
'quantitative', 'continuous', 'interval', 'ratio', 'REAL'	'continuo
'date', 'time', 'datetime'	'date'

The distinction arises from their intrinsic properties as summarized in the following table:

Property	'categorical'	'continuous'	'date'
Can be used to aggregate data across it	Yes	No	Yes
Can be used with the aggregate functions: SUM, AVG, MIN and MAX	No	Yes	No
Can be used with the 'IN' operation	Yes	No	Yes
Can be used with the 'BETWEEN' operation	No	Yes	Yes

[1] The term 'nominal' may be interchangeably used throughout this document.

The three basic data types can be equally used for the rest of operations.

4 TABULAR DATA TEMPLATES

Without loss of generality, and to simplify the mathematical constructs, we herein ignore the logical operation 'NOT' and the value comparison operations BETWEEN, LIKE and IN. We further consider that the same set of value comparison operations is applicable to all types of variables. In practice, a distinction in their applicability is made and all the aforementioned operations are considered. Further, please note that, for our purpose, the terms categorical and nominal are used interchangeably. Define:

$\mathcal{T}^r$: Database table for real data.

$\mathcal{T}^s$: Database table for synthetic data

N : The number of records in both $\mathcal{T}^r$ and $\mathcal{T}^s$.

$\mathbb{A}^n = \left\{A_1^n, A_2^n, \cdots, A_{|\mathbb{A}^n|}^n\right\}$ is the set of nominal variables in both $\mathcal{T}^r$ and $\mathcal{T}^s$ where $|\mathbb{A}^n|$ indicates the number of these variables.

$\mathbb{A}^c = \left\{A_1^c, A_2^c, \cdots, A_{|\mathbb{A}^c|}^c\right\}$ is the set of continuous variables in both $\mathcal{T}^r$ and $\mathcal{T}^s$.

$\mathbb{A}^d = \left\{A_1^d, A_2^d, \cdots, A_{|\mathbb{A}^d}^d\right\}$ is the set of date variables in both $\mathcal{T}^r$ and $\mathcal{T}^s$.

For any member A_j in the above sets, it may assume a value given in the real dataset $\mathcal{T}^r$ such that: $V(A_j)$ is the the set of all values that A_j may take. The length of $V(A_j)$ is $|V(A_j)| = N$.

We further define:

$LO = \{AND, OR\}$ is the set of logical operations.

$CO = \{=, \neq, <, \leq, >, \geq\}$ is the set of value comparison operations.

$AG = \{SUM, AVG, MIN, MAX\}$ is the set of aggregate functions.

Random samples are drawn from the above sets to construct the three major queries defined below. The basic sampling functions can be defined as:

$f_s : S_m \rightarrow S_s$ where f_s is a sampling function that maps any set S_m into a single element set S_s. For instance, the set AG may be mapped by f_s into $\{AVG\}$ $f_m : S_{m1} \rightarrow S_{m2}$ where f_m is a sampling function that maps any set S_{m1} into a multiple element set S_{m2}. For instance, the set $\mathbb{A}^n$ may be mapped by f_m into $\left\{A_1^n, A_{|\mathbb{A}^n|}^n\right\}$

5 AGGREGATE QUERIES

If $\mathbb{A}^c = \phi$, an aggregate query takes the form:

SELECT $f_m(\mathbb{A}^n), \text{COUNT}(^*)$

FROM $\mathcal{T}^r$

GROUP BY $f_m(\mathbb{A}^n)$

However, if $\mathbb{A}^c \neq \phi$, an aggregate query takes the form:

SELECT $f_m\left(\mathbb{A}^n\right), f_s(AG)\left(f_s\left(\mathbb{A}^c\right)\right), \text{COUNT}(*)$

FROM $\mathcal{T}^r$

GROUP BY $f_m\left(\mathbb{A}^n\right)$

Similar queries are constructed for $\mathcal{T}^{\int}$.

6 FILTER QUERIES

If $\mathbb{A}^c = \phi$, a filter query takes the form:

SELECT $*$

FROM $\mathcal{T}^r$

WHERE $\left[f_s\left(\mathbb{A}^n \cup \mathbb{A}^c \cup \mathbb{A}^d\right) \quad f_s(CO) \quad f_s\left(V\left(f_s\left(\mathbb{A}^n \cup \mathbb{A}^c \cup \mathbb{A}^d\right)\right)\right)\right]$
$[f_s(LO)]$
$\left[f_s\left(\mathbb{A}^n \cup \mathbb{A}^c \cup \mathbb{A}^d\right) \quad f_s(CO) \quad f_s\left(V\left(f_s\left(\mathbb{A}^n \cup \mathbb{A}^c \cup \mathbb{A}^d\right)\right)\right)\right]$
$[f_s(LO)]$
$\ldots$

The WHERE clause comprises basic expressions denoted by [ ]. Say if the sampled number of variables equals to 1 (i.e. $\left|f_m\left(\mathbb{A}^n \cup \mathbb{A}^c \cup \mathbb{A}^d\right)\right| = 1$), then the above expression will reduce to:

If $\mathbb{A}^c \neq \phi$, a filter query takes the form:

SELECT $f_s(AG)\left(f_s\left(\mathbb{A}^c\right)\right), \text{COUNT}(*$

FROM $\mathcal{T}^r$

WHERE $\left[f_s\left(\mathbb{A}^n \cup \mathbb{A}^c \cup \mathbb{A}^d\right) \quad f_s(CO) \quad f_s\left(V\left(f_s\left(\mathbb{A}^n \cup \mathbb{A}^c \cup \mathbb{A}^d\right)\right)\right)\right]$
$[f_s(LO)]$
$\left[f_s\left(\mathbb{A}^n \cup \mathbb{A}^c \cup \mathbb{A}^d\right) \quad f_s(CO) \quad f_s\left(V\left(f_s\left(\mathbb{A}^n \cup \mathbb{A}^c \cup \mathbb{A}^d\right)\right)\right)\right]$
$[f_s(LO)]$

Similar queries are constructed for $\mathcal{T}^s$.

7 FILTER-AGGREGATE QUERIES

Filter-Aggregate queries are the most important for comparing real and synthetic datasets. The query is constructed by combining the above two forms. Hence, if $\mathbb{A}^c = \phi$, a filter-aggregate query takes the form:

and if $\mathbb{A}^c \neq \phi$, a filter-aggregate query takes the form:

Similar queries are constructed for $\mathcal{T}^s$.

8 LONGITUDINAL DATA TEMPLATES

Fuzzy SQL supports the generation of valid random queries for joint tables in parent-child relationship. One-to-many relationship for single or multiple-child is supported. In continuation to the definitions given in Tabular Data Templates, and for simplicity, we will drop the distinction between the real and synthetic tables since the generated random queries are always identical. Define:

$\mathcal{T}$: Any randomly selected parent table.

$\mathcal{T}.\mathbb{A}^n = \left\{\mathcal{T}.A_1^n, \mathcal{T}.A_2^n, \cdots, \mathcal{T}.A_{|\mathcal{T}.\mathbb{A}^n|}^n\right\}$ is the set of nominal variables in $\mathcal{T}$ where $|\mathcal{T}.\mathbb{A}^n|$ indicates the number of these variables. Similarly, we define the continuous and date variables pertaining to the parent table using the superscripts 'c' and 'd' instead on 'n'.

$\mathbb{R} = \{\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_{|\mathbb{R}|}\}$ where $|\mathbb{R}|$ is the number of child tables related to $\mathcal{T}$.

$\mathcal{R}_k.\mathbb{A}^n = \left\{\mathcal{R}_k.A_1^n, \mathcal{R}_k.A_2^n, \cdots, \mathcal{R}_k.A_{|\mathcal{R}_k.\mathbb{A}^n|}^n\right\}$ is the set of nominal variables in $\mathcal{R}_k$ $\forall k$ where $|\mathcal{R}_k.\mathbb{A}^n|$ indicates the number of these variables. Similarly, we define the continuous and date variables pertaining to the k^{th} child table using the superscripts 'c' and 'd'.

$\mathcal{T}.A_{\mathcal{R}_k}^n$: is the parent's joining key between $\mathcal{T}$ and $\mathcal{R}_k$ $\forall k$ whereas $\mathcal{T}.A_{\mathcal{R}_k}^n \in \mathcal{T}.\mathbb{A}^n$.

$\mathcal{R}_k.A_{\mathcal{T}}^n$: is the child's joining key between $\mathcal{T}$ and $\mathcal{R}_k$ $\forall k$ whereas $\mathcal{R}_k.A_{\mathcal{T}}^n \in \mathcal{R}_k.\mathbb{A}^n$.

To avoid repetition, we show herein the construct for a filter-aggregate query.

9 FILTER-AGGREGATE QUERIES

If $f_m(\mathcal{T}.\mathbb{A}^c \cup \mathcal{R}_k.\mathbb{A}^c) = \phi, \forall k$, and let the random sampling of the related tables be $f_m : \mathbb{R} = \{\mathcal{R}_1, \mathcal{R}_2, \cdots\} \rightarrow \mathbb{R}_s = \{\mathcal{R}_{s1}, \mathcal{R}_{s2}, \cdots\}$, then a filter-aggregate query takes the form:

SELECT $f_m(\mathcal{T}.\mathbb{A}^n \cup \mathcal{R}_{s1}.\mathbb{A}^n \cup \cdots \cup \mathcal{R}_{|\mathbb{R}_s|}.\mathbb{A}^n)$, COUNT(*)

FROM $\mathcal{T}$

JOIN $\mathcal{R}_{s1}$ ON $\mathcal{T}.A^n_{\mathcal{R}_{s1}} = \mathcal{R}_{s1}.A^n_{\mathcal{T}}$

JOIN $\mathcal{R}_{s2}$ ON $\mathcal{T}.A^n_{\mathcal{R}_{s2}} = \mathcal{R}_{s2}.A^n_{\mathcal{T}}$

...

WHERE

$[f_s(\mathcal{T}.\mathbb{A}^n \cup \mathcal{T}.\mathbb{A}^c \cup \mathcal{T}.\mathbb{A}^d \cup \mathcal{R}_{s1}.\mathbb{A}^n \cup \mathcal{R}_{s1}.\mathbb{A}^c \cup \mathcal{R}_{s2}.\mathbb{A}^d \cup \mathcal{R}_{s2}.\mathbb{A}^n \cup \cdots)$

$f_s(CO)$

$f_s(\mathcal{T}.\mathbb{A}^n \cup \mathcal{T}.\mathbb{A}^c \cup \mathcal{T}.\mathbb{A}^d \cup \mathcal{R}_{s1}.\mathbb{A}^n \cup \mathcal{R}_{s1}.\mathbb{A}^c \cup \mathcal{R}_{s2}.\mathbb{A}^d \cup \mathcal{R}_{s2}.\mathbb{A}^n \cup \cdots)]$

$[f_s(LO)]$

...

GROUP BY $f_m(\mathcal{T}.\mathbb{A}^n \cup \mathcal{R}_{s1}.\mathbb{A}^n \cup \cdots \cup \mathcal{R}_{|\mathbb{R}_s|}.\mathbb{A}^n)$

If $f_m(\mathcal{T}.\mathbb{A}^c \cup \mathcal{R}_k.\mathbb{A}^c) \neq \phi, \forall k$, the SELECT term in the above expression is replaced by:

SELECT $f_m(\mathcal{T}.\mathbb{A}^n \cup \mathcal{R}_{s1}.\mathbb{A}^n \cup \cdots \cup \mathcal{R}_{|\mathbb{R}_s|}.\mathbb{A}^n)$,

$f_s(AG)(f_s(\mathcal{T}.\mathbb{A}^c \cup \mathcal{R}_{s1}.\mathbb{A}^{\mathfrak{c}} \cup \mathcal{R}_{s2}.\mathbb{A}^{\mathfrak{c}} \cup \cdots))$,

COUNT(*)

In practice, the pre-aggregation filter condition is applied either using WHERE or alternatively using AND after the JOIN clause. Our implementation make a random pick.

10 METRICS

HELLINGER DISTANCE FOR DATASETS

The Hellinger distance may be used to measure the quality of synthetic data. First we consider the calculation of the Hellinger distance between the real and the synthetic tabular datasets $\mathcal{T}^r$ and $\mathcal{T}^{\int}$ respectively. Define:

$\mathbb{A} = \left\{A_1, \cdots, A_i, \cdots, A_{|\mathbb{A}|}\right\}$ is the set of nominal variables in both $\mathcal{T}^r$ and $\mathcal{T}^s$ where $|\mathbb{A}|$ indicates the number of these variables.

$o^j_{A_i}$ is the number of occurrences (i.e. counts) of the j^{th} class for the nominal variable A_i in $\mathcal{T}^r$. The discrete probability of the j^{th} class can be calculated

as:

$$r^j_{A_i} = \frac{o^j_{A_i}}{\sum_{\forall j} o^j_{A_i}}$$

For instance, consider the nominal variable A_1 = "income" with two classes '<=50k' and '>50k'. Then the first class may have $o^1_{A_1} = 1200$ occurrences and the second may have $o^2_{A_1} = 2000$ occurrences with discrete probabilities of $r^1_{A_1} = 0.375$ and $r^2_{A_1} = 0.625$ respectively.

Similarly, for the synthetic data $\mathcal{T}^s$ we can calculate the discrete probabilities $s^j_{A_i}$

The Hellinger distance for the nominal variable A_i is calculated as:

$$\mathcal{H}^{A_i} = \frac{1}{\sqrt{2}} \left(\sum_{\forall j} \left(\sqrt{r^j_{A_i}} - \sqrt{s^j_{A_i}} \right)^2 \right)^{1/2}$$

The Hellinger distance between $\mathcal{T}^r$ and $\mathcal{T}^s$ can be calculated by taking the mean across all nominal variables:

$$\mathcal{H}^{\mathcal{T}} = \frac{1}{|\mathbb{A}|} \sum_{i=1}^{|\mathbb{A}|} \mathcal{H}^{A_i}$$

11 HELLINGER DISTANCE FOR QUERIES

In aggregate queries, grouping is done by randomly selected nominal variables. In this sense, measuring the Hellinger distance for the datasets as explained above is just a special case where grouping is done by a single nominal variable at a time. So, for $|\mathbb{A}|$ number of nominal variables in the original datasets, we may execute $|\mathbb{A}|$ number of queries with each query grouped by a single variable. Then by averaging the Hellinger distances of these queries, we reach the same results in eq_hlngr_T

If grouping is done by more than a single variable, it is as we are defining a new nominal variable A^q where A^q may be any combination of two or more dataset variables $A^i \quad \forall A^i \in \mathbb{A}$ as defined in hellinger distance for datasets. The query will result in specific number of classes for A^q. Using the superscript j to

indicate the j^{th} class of A^q, we calculate the Hellinger distance for the query by:

$$\mathcal{H}^{Q} = \frac{1}{\sqrt{2}} \left(\sum_{\forall j} \left(\sqrt{r^j_{A^q}} - \sqrt{s^j_{A^q}} \right)^2 \right)^{1/2}$$

Both discrete probabilities r and s were defined earlier in hellinger distance for datasets.

For instance, consider an aggregate query grouped by the two nominal variables A_1 = "income" and A_2 = "marital status" with each having two distinct classes. The query will result in the variable A^q having four distinct classes with a discrete probability $r^j_{A^q}$ for each resulting class j.

12 EUCLIDEAN DISTANCE FOR QUERIES

Once the aggregate query is executed, the variable A^q, as defined in Hellinger Distance for Queries, will result in the classes: $1, 2..j..J$. If the data includes a continuous variable A^c, an aggregate function, say AGG, may be applied to that variable. For each class j, an aggregation value $[AGG\,(A^c)]_j$ of the continuous variable can be calculated. For instance, let A^q be a combination of two nominal variables A_1 = "income" and A_2 = "marital status". Let A^c = "age" be a continuous variable, then for each of the four distinct classes, we can calculate the AVG(age). Define:

v^r_j is the aggregate value (e.g. [AVG(age)]$_j$) corresponding to the j^{th} class of an arbitrary continuos variable A^c in $\mathcal{T}^r$.

v^s_j is the aggregate value corresponding to the j^{th} class of the same continuos variable A^c in $\mathcal{T}^s$

From the above components, we can find the difference components:

$d_j = v^r_j - v^s_j \quad \forall j$

We further find the mean and standard deviation across all the classes:

$\mu^d = \frac{1}{J} \sum_{j=1}^{J} d_j$

$\sigma^d = \sqrt{\frac{1}{J} \sum_{j=1}^{J} (d_j - \mu^d)^2}$

and we compute the standardized aggregate values:

$z_j = \frac{d_j - \mu^d}{\sigma^d}$

Finally, we compute the norm and normalize it to reflect the normalized Euclidean distance between the real and synthetic queries :

$\mathcal{E}^Q = \frac{\|z_j\|}{J}$

Normalizing the distance by the number of resulting classes for the random query enables us to average the Euclidean distance across multiple queries since each of them may result in different number of classes.

13 REFERENCES

Fan, Ju, Tongyu Liu, Guoliang Li, Junyou Chen, Yuwei Shen, and Xiaoyong Du. 2020.

"Relational Data Synthesis Using Generative Adversarial Networks: A Design Space Exploration." arXiv. http://arxiv.org/abs/2008.12763.

Ghit, Bogdan, Nicolas Poggi, Josh Rosen, Reynold Xin, and Peter Boncz. 2020. "SparkFuzz: Searching Correctness Regressions in Modern Query Engines." In Proceedings of the Workshop on Testing Database Systems, 1-6. Portland Oregon: ACM. https://doi.org/10.1145/3395032.3395327.

Jibson, Matt. 2019. "SQLsmith: Randomized SQL Testing in CockroachDB." Cockroach Labs. June 27, 2019. https://www.cockroachlabs.com/blog/sqlsmith-randomized-sqltesting/.

Li, Kaiyu, Yong Zhang, Guoliang Li, Wenbo Tao, and Ying Yan. 2019. “Bounded Approximate Query Processing." IEEE Transactions on Knowledge and Data Engineering 31 (12): 2262-76. https://doi.org/10.1109/TKDE.2018.2877362.

Rigger, Manuel, and Zhendong Su. 2020. "Testing Database Engines via Pivoted Query Synthesis." https://doi.org/10.48550/ARXIV.2001.04174.

Zalewski, Michal. n.d. "American Fuzzy Lop." Accessed October 20, 2022. https://lcamtuf.coredump.cx/afl/.

Zhong, Rui, Yongheng Chen, Hong Hu, Hangfan Zhang, Wenke Lee, and Dinghao Wu. 2020. "SQUIRREL: Testing Database Management Systems with

Language Validity and Coverage Feedback." https://doi.org/10.48550/ARXIV.2006.02398.