

DATABASE KEY IDENTIFICATION

BACKGROUND

This description relates to data quality.

5 Many businesses have large collections of data sets, linked together by keys. These data sets may be tables in a relational database or they may be flat files in a disk storage system. The collections may number in the hundreds or thousands of data sets, with tens of keys or more. The structure may be undesigned, arising from an accumulation of data over time from multiple sources. There may once have been a design, but that design has been lost, forgotten, or is known to be out-of-date. The design
10 may be known only to specialists but is otherwise undocumented. Or, the design may be known and current, but it is not enforced.

SUMMARY

In general, one innovative aspect of the subject matter described in this
15 specification can be embodied in methods that include the actions of receiving an identification of a first field in a first data set, the first data set including a plurality of records. The methods include the actions of identifying a set of values, the set including, for each record, a value in the respective record associated with the field. The methods include the actions of generating a filter mask based on the set of values, where
20 application of the filter mask is capable of determining that a given value is not in the set of values. The methods include the actions of receiving a second data set, the second data set including a second field, the second data set including a plurality of records. The methods include the actions of determining a count of a number of records in the second data set having a value associated with the second field that passes the filter mask. The
25 methods also include the actions of storing the count in a profile.

Other embodiments of this aspect include corresponding computer systems, apparatus, and computer programs recorded on one or more computer storage devices, each configured to perform the actions of the methods. A system of one or more computers can be configured to perform particular actions by virtue of having software,

firmware, hardware, or a combination of them installed on the system that in operation causes or cause the system to perform the actions. One or more computer programs can be configured to perform particular actions by virtue of including instructions that, when executed by data processing apparatus, cause the apparatus to perform the actions.

5 The foregoing and other embodiments can each optionally include one or more of the following features, alone or in combination. The methods may include the actions of determining the Sørensen-Dice coefficient of the set of values and the records in the second data set having a value associated with the second field. Generating a filter mask may include for each value in the set of values and generating a filter key for the
10 corresponding value. Generating a filter key for the corresponding value may include generating a hash value for the corresponding value, segmenting the hash value into a predetermined number of integers, creating a filter key by setting bits in a bit vector based on the integers. Generating the filter mask may include performing a binary operation on each of the generated filter keys. Determining a count of a number of
15 records in the second data set having a value associated with the second field that passes the filter mask may include calculating a second filter key for a value associated with the second field and comparing the second filter key to the filter mask.

Aspects can include one or more of the following advantages. Confidence can be established in a data quality program. Unknown or unsuspected relationships may be
20 revealed. Information of which fields hold shared values may be discovered. Discrepancies between expected relationships and identified relationships can be discovered. An accurate data structure on which to base further data quality analysis may be determined. In particular, the discovered structure of key relationships may be used as the starting point from which to quantify the referential integrity of the collection of data
25 sets.

Other features and advantages of the invention will become apparent from the following description, and from the claims.

DESCRIPTION OF DRAWINGS

FIG. 1 shows an example of a data processing system in which the database
30 crawling techniques can be used.

FIG. 2 shows a flowchart for an exemplary filter mask creation procedure.

FIG. 3 illustrates an example of generating a filter key.

FIG. 4 illustrates creating a filter mask from the filter keys.

FIG. 5 shows a flowchart for an exemplary profiling procedure based on the filter mask.

5

DESCRIPTION

Institutions collect large amounts of data. The data is collected over a large period of time. Over time, new data sets are created and old data sets are changed. It can be difficult to identify relationships between data sets.

In general, data sets are structured sets of data. The data set is broken up into discrete records. Each record of a data set shares a common record format. For example, a relational database table includes rows of data, each record (e.g. row) of data conforms to the database schema definition of the table. A schema can include a definition of a table (or data set). Generally, the schema defines one or more fields that exist in the data set. These records can be identified by using one or more of the fields as a key. In general, “a primary key” is one or more fields whose values uniquely identify each record in the data set.

In some implementations, a primary key includes one or more fields in the record. The key may be derived from business information contained within the record, for example, a first name and last name and current address may identify a customer (for example, “John”, “Smith”, “154 Locke Lane”).

One type of primary key is a “surrogate key”. A surrogate key is a unique identifier, or primary key, that is not derived from business information. For example, a surrogate key may be selected from a numerical sequence. Surrogate keys can be unique for each data set or may be unique across all data sets within a particular system (for example, across all the data sets for a particular company). For example, a number “10234” may identify the record for John Smith of 154 Locke Lane. Because business information may change (for example, John Smith may move from 154 Locke Lane to 275 Greenwood Ave) or may not be unique (for example, there may be a John Smith of 154 Locke Lane and a John Smith Jr. of 154 Lock Lane), surrogate keys are generally preferred. As used herein, surrogate keys that are unique for all data sets within a

particular domain are referred to as “system-wide surrogate keys”. Surrogate keys that are unique within a data set are referred to as “data set surrogate keys”.

Data sets can also be related to other data sets, and records within a data set can be related to records in other data sets. Generally, relationships between data sets and records are described by using a “foreign key”. A foreign key is a field defined in the data format of a data set whose values refer to the primary key of another field.

For example, a business may be organized to sell products to customers. A customer places an order which references one or more products. In order to support the business model, a customer record may be stored in a customer data set. When the customer places an order, an order record may be created in an order data set. The order record includes a field that references the customer who placed the order, typically by referencing the primary key of the customer record. More concretely, an Order Table may include a record for an order placed by John Smith. The order may have a primary key with a numeric value of 987,321 and include a “customer identifier” field which contains the numeric value 10,234 (which references John Smith of 154 Locke Lane, as described above). In this example, the customer data set may be referred to as a primary data set, the order data set (which references the customer data set) may be referred to as a secondary data set.

In general, the terms primary data set and secondary data set define characteristics of a relationship between two data sets and may not be absolute. For example, an order may include multiple line items; each line item references a different product. In this example, a line item record may be created for each product. The line item record may include a foreign key that references the order record. In this example, as per the order to line item relationship, the order is the primary data set and the line item is the secondary data set. A data set may also be a part of multiple relationships; for example, the line item record may also reference a product record stored in a product data set. In this example, the product data set is the primary data set and the line item data set is the secondary data set.

Many data stores have mechanisms that can be used to enforce these relationships (referred to as referential integrity). However, frequently these mechanisms are not used for operational or performance reasons. Instead, systems rely on programming logic to

enforce referential integrity. As a result, over time, the referential integrity of a system can degrade. Orphaned records (records in secondary data sets that lack a foreign key relationship to the primary data set) can be introduced; expected relationships may be missing; etc. In some cases, a new data set may be introduced into an existing data
5 system (for example, during a merger effort). Equivalent primary and foreign key relationships between databases may not be precisely known.

Determining the structure of key relationships from the data rather than from the schema can be a first step in a data-driven data quality analysis of the collection of data sets. Being able to see the actual key relations present in their data gives confidence to
10 an organization as they start a data quality program. Unknown or unsuspected relationships between data sets may be revealed; information of which fields hold shared values may be discovered. Discrepancies between expected relationships and identified relationships can be discovered. An accurate structure on which to base further data quality analysis may be determined. In general, the discovered structure of primary key –
15 foreign key relationships may be used as the starting point from which to quantify the referential integrity of the data sets in the system.

Conventional methods of reverse engineering primary key and foreign key relationships involves identifying potential foreign key fields and potential primary key fields in the different data sets. This requires comparing each value in the potential
20 foreign key to each value in the potential primary key.

An alternative approach described herein includes utilizing a filter mask which allows for false positives (that is, the filter mask may indicate that a value is a member of a set when it is not) while eliminating the possibility of false negatives (that is, the filter mask will not indicate that the value is not a member of the set when it is). In general, a
25 filter mask is created from the values stored in the records of the field (or fields) of the potential primary key. The value of the potential foreign key field of each record is then compared to the filter mask.

For example, one process utilizes a Bloom filter to characterize the distribution of values in each field of every data set from a designated collection of fields and data sets.
30 A Bloom filter is a space-efficient probabilistic data structure that can be used to test whether an element is a member of a set.

FIG. 1 shows an example of a data processing system 100 in which the key matching and identification techniques can be used. In general, key matching and identification refers to a process of gathering information about one or more data sets and the information to identify potential relationships between the data sets. The system 100 includes a data source 102 that may include one or more sources of data such as storage devices or connections to online data streams, each of which may store or provide data in any of a variety of formats (e.g., database tables, spreadsheet files, flat text files, or a native format used by a computer). An execution environment 104 includes a filter mask creation module 106 and a filter mask application module 112. The execution environment 104 may be hosted, for example, on one or more general-purpose computers under the control of a suitable operating system, such as a version of the UNIX operating system. For example, the execution environment 104 can include a multiple-node parallel computing environment. This may include a configuration of computer systems using multiple central processing units (CPUs) or processor cores, either local (e.g., multiprocessor systems such as symmetric multi-processing (SMP) computers), or locally distributed (e.g., multiple processors coupled as clusters or massively parallel processing (MPP) systems), or remote, or remotely distributed (e.g., multiple processors coupled via a local area network (LAN) and/or wide-area network (WAN)), or any combination thereof.

The filter mask creation module 106 reads data from the data source 102 and stores a filter mask for one or more fields of a data set in the data source 102. The filter mask can be, for example, a Bloom filter.

A conventional Bloom filter includes a number of bits and one or more hash functions. Each hash function is used to map an input value into a subset of the bits in the filter key. Filter keys are combined to create a filter mask. A Bloom filter can be created by creating filter keys using a single hash function as described below.

Filter masks can be created for a field that is identified as a potential key. The filter mask can be generated using the process described below. Storage devices providing the data source 102 may be local to the execution environment 104, for example, being stored on a storage medium connected to a computer hosting the execution environment 104 (e.g., hard drive 108), or may be remote to the execution

environment 104, for example, being hosted on a remote system (e.g., mainframe 110) in communication with a computer hosting the execution environment 104, over a remote connection (e.g., provided by a cloud computing infrastructure).

5 The filter mask application module 112 uses the filter mask generated by the filter mask creation module 106 to compare values in records of a data set. The filter mask application module may determine (among other things) for a particular field of a data set, the following: a count of the number of values that match the characteristics specified by the filter mask, as discussed below, a percentage of the number of values that match the filter mask, and / or statistics that compare the similarity of the filter mask to the
10 values, such as a Sørensen-Dice coefficient. Other statistics may also be determined that can be used to determine whether the field in the data set is a good match for the identified key field.

The filter mask application module 112 may operate in conjunction with other data profiling modules. The output data 114 may be stored back in the data source 102
15 or in a data storage system 116 accessible to the execution environment 104, or otherwise used. The data storage system 116 is also accessible to a development environment 118 in which a developer 120 is able to identify one or more fields in a data set that are potential keys.

The development environment 118 is, in some implementations, a system for
20 developing applications as dataflow graphs that include vertices (representing data processing components or data sets) connected by directed links (representing flows of work elements, i.e., data) between the vertices. For example, such an environment is described in more detail in U.S. Publication No. 2007/0011668, titled “Managing Parameters for Graph-Based Applications,” incorporated herein by reference. A system
25 for executing such graph-based computations is described in U.S. Patent 5,966,072, titled “EXECUTING COMPUTATIONS EXPRESSED AS GRAPHS,” incorporated herein by reference. Dataflow graphs made in accordance with this system provide methods for getting information into and out of individual processes represented by graph components, moving information between the processes, and defining a running order for
30 the processes. This system includes algorithms that choose interprocess communication methods from any available methods (for example, communication paths according to the

links of the graph can use TCP/IP or UNIX domain sockets, or use shared memory to pass data between the processes).

The filter mask creation module 106 can receive data from a variety of types of systems that may embody the data source 102, including different forms of database systems. The data may be organized as records having values for respective fields (also called “attributes” or “columns”), including possibly null values. When first reading data from a data source, the filter mask creation module 106 typically starts with some initial format information about records in that data source. In some circumstances, the record structure of the data source may not be known initially and may instead be determined after analysis of the data source or the data. The initial information about records can include, for example, the number of bits that represent a distinct value, the order of fields within a record, and the type of value (e.g., string, signed/unsigned integer) represented by the bits.

FIG. 2 shows a flowchart 200 for an exemplary filter mask creation procedure 200 for use in a Bloom filter. The procedure can be performed, for example, by the filter mask creation module 106 of FIG. 1. The procedure 200 includes receiving data 202. The data can be included in a data set, such as a database table or flat file. The data may include one or more fields and one or more records, each record including values for at least some of the fields. In some scenarios, a record may include a NULL value or other indication that a value is not present for a particular field.

The procedure 200 includes identifying a potential key 206. The potential key may be made up of one or more fields. The potential key may be identified by a user or may be determined based on the data or metadata (data that describes the data). For example, a potential key may be identified by a primary key constraint on a relational database table. In general, a constraint specifies rules for data in a data set, such as a relational database table. A primary key constraint indicates that a particular field (or combination of fields) is a primary key such that the particular field (or combination of fields) uniquely identifies a record in the data set.

For each record, the procedure 200 creates a filter key 208. In general, a filter key is a sparsely populated key that can be created based on the value stored in the field of the record. In order to generate a filter mask that does not allow for false negatives, a filter

key should be generated for each unique value in the field. In some implementations, unique values may be identified prior to generating the key and a key may be generated for each unique value.

FIG. 3 illustrates an example of generating a filter key 350. In a conventional Bloom filter, a series of hash functions are performed on a value to identify a filter key. Each of the hash functions maps some elements to one or more bits in a bit array. In an alternative implementation, a multi-step hashing function can be used, as described below.

In this example, the field for the record includes the value 302, in this example “John Smith.” A hashing algorithm 304 is applied to the value 302 to generate a hash value 306, in this example the hash value has a hexadecimal representation 6117323D2CABBC17D44C2B44587F682C_{hex}. In general, the hashing algorithm may be any hashing algorithm that has a sufficiently low potential for collisions. In this example, an MD5 message-digest algorithm is used. The MD5 hash produces a 128-bit (16-byte) hash value. Other hashing algorithms may also be used such as SHA-1, which generates a 160-bit (20 byte) hash value, or SHA-2, which generates a 224, 256, 384, or 512 bit hash value. In general, the hashing algorithm is selected to minimize the likelihood of collisions within the data while maintaining desired performance characteristics. SHA-1 returns a 20 byte string which can be interpreted as 5 unsigned 4 byte integers. A SHA-256 returns a 32 byte string which can be interpreted as 8 unsigned 4 byte integers. In general, the size of the key and consequently the algorithm is selected based on the size of data to be analyzed. The size of the key is increased to avoid collisions in the key space. For example, a 64MB key and a 25 million row candidate key table produces a false positive ratio of approximately 0.0004 with 5 keys (SHA-1), 0.0001 with 8 keys (SHA-256).

The procedure segments 308 the hash value 306 into numeric values, in this example, four byte integer values 310, 320, 328, and 330. For example, the md5 hash value 6117323D2CABBC17D44C2B44587F682C_{hex} is divided into integer values 6117323D_{hex}, 2CABBC17_{hex}, D44C2B44_{hex}, 587F682C_{hex}. Other sizes may be used; for example, the hash value may be divided into eight- byte integer values.

For illustrative purposes, FIG.3 shows that the hexadecimal representation of the integer value 310 (6177323D_{hex}) can also be represented as a decimal value (1,628,910,141_{decimal}).

5 In scenarios where the length of a filter key is less than the maximum value of the integer (e.g. 4,294,967,295 for an unsigned four byte integer) the value of the integer may be reduced using a modulo operation. In this example, when the filter key 350 is 4096 bits in size each integer value 310, 320, 328, and 330 is adjusted by modulo 4096 to produce values 316, 324, 332, and 334.

10 The size of a filter key may vary depending on the system. For example, the filter key may be 8 megabytes (MB) in size, or 67,108,864 bits. The exact size is chosen based on the well understood statistics of Bloom filters for optimal performance and precision. In general, the size of the Bloom filter is selected to ensure that the filter mask is sparsely populated, that is, has a large number of unset bits ('0') with respect to set bits ('1').

15 The procedure maps each of the integer values into a bit in the filter key 350. In some implementations, the filter key 350 is a one dimensional array of bits. A location in the array may be determined based off of a numerical offset from the head of the array, referred to as an index. Depending on the programming language, the first bit in the array can be referenced using the index '0' (zero-based numbering) or the index '1'. In this example, the array is zero-based. Each of the integer values results in setting a single
20 bit within the array. A bit is set to the value of '1' in index value position. In this example, the bits at index position 573, 2092, 2884, and 3095 are set to '1' and the remaining bits are set to (or remain) '0'.

Referring again to FIG. 2, the procedure combines the filter keys to create a filter mask 212. FIG. 4 illustrates creating a filter mask from the filter keys. The data set
25 includes records, each record includes values in fields 402a-j identified as a potential key. The procedure creates a filter key 404a-j for each value (for example, value Jonas Demers 402a is used to produce filter key Key A 404a, value Sofia Votaw 402b is used to produce filter key Key B 404b, etc.). The procedure combines the filter keys 404a-j to create a filter mask 406. The filter keys 404a-j can be combined, for example, by using a
30 binary OR operation. An OR operation accepts two Boolean values (a Boolean value contains either a true or false) and returns true if either of the two values is true and false

if both of the two values are false. Conventionally, a binary ‘1’ represents a true value and a binary ‘0’ represents a false value. A binary OR operation takes two bit patterns of equal length and performs the logical inclusive OR operation on each pair of corresponding bits. The result in each position is 1 if the first bit is 1 or the second bit is 1 or both bits are 1; otherwise, the result is 0.

For example, a binary OR of the number 10011B and the number 00010B produces:

10011B

00010B

10011B

The resulting filter mask 406 is a “fingerprint” of the contents of the potential key for the data set.

Referring to FIG. 2, the procedure can store 214 the filter mask in a persistent data store, with meta-data associated with the data set, or in any other storage medium.

FIG. 5 shows a flowchart for an exemplary profiling procedure 500 based on the filter mask. The procedure 500 receives a filter mask 501. The filter mask may be generated using the process described above. In some implementations, the procedure 500 may receive one or more filter masks. For example, a system or user may identify two data sets to be compared to attempt to identify any primary key-foreign key relationships between the two data sets. Alternatively a data set may be compared to multiple different data sets to attempt to identify any key relationships between the initial data set and all the other data sets in the system. In some implementations, a filter mask may be generated for each field in a data set. Each of the filter masks may be compared to the fields in other data sets.

The procedure 500 receives data 502 to profile. The data may include one or more data sets, each data set containing one or more fields, as well as one or more records that each includes values for the one or more fields. In some implementations, one or more fields of the data set may be optional. A record may or may not include a value for an optional field.

The procedure 500 may analyze data sets sequentially or in parallel. Each parallel process can be provided its own copy of the filter mask. While filter masks are large

enough to be relatively sparsely populated, they remain relatively small when compared to the data being processed. For each record, the procedure applies the filter mask to the value for a field. In some implementations, the procedure may process a single field, for example, a predetermined row in a database table. In other implementations, the system may compare each value in the record the one or more filter masks.

In some implementations, comparing a value to the filter mask can include generating a filter key for the value 504. The filter key may be generated using the process described above with respect to FIGS 2 and 3. In general, each filter key can be generated using the same process used to generate the keys combined to create the filter mask. Using the same process ensures that all keys are the same length as the filter mask.

The generated filter key can be compared to the filter mask by performing a binary AND operation 506. In general, an AND operation accept two values, produces an affirmative response if both of the values are true and negative response otherwise. Conventionally, a binary ‘1’ represents a true value and a binary ‘0’ represents a false value. A binary AND operation is a function that accepts two bit patterns of equal length and performs the AND operation on each pair of corresponding bits. The result in each position is 1 if the first bit is 1 and the second bit; otherwise, the result is 0.

For example, a binary AND of the number ‘10011B’ and the number ‘00010’B produces:

10011B

00010B

00010B

The result of the binary AND operation can be compared to the filter key 506. If the result of the binary AND operation matches the filter key, then the value passes the filter and is a potential match. If the result of the binary AND operation does not match the filter key, then the value does not pass the filter and is not a match.

For example, a filter mask of 10011B is compared to a filter key of 00010B. As described above, performing a binary AND operation on the two binary numbers results in a value of 00010B. Because the result value (00010B) is the same as the filter key (00010B) the filter key passes the filter.

As a contrary example, the filter mask 10011B is compared to a filter key 01001B. A binary AND operation formed on the mask and key is shown below.

10011B

01001B

00001B

Because the result, 00001B, does not match the filter key, 01001B, the filter key does not pass the filter.

Comparing a filter key to a filter mask is a relatively quick operation. Therefore, a large number of records can be processed more quickly and efficiently than through conventional key identification methods, which usually include comparing each value in the potentially primary key field with the each value in the potential foreign key field.

As discussed above, the filter masking algorithm can be selected to conclusively determine that an element is not a member of the set used to generate the filter mask. The results of the comparison can be stored in a persistent data store, such as a relational database or flat file.

Once the procedure has processed the determined fields for the records of the data set, the procedure can determine if there are more data sets to process 508. If there are more data sets to process, the next data set is received 502.

In some implementations, a filter mask generated from a potential primary key in a primary table can be compared to each field in a potential secondary table.

In some implementations many filter masks from different data sets may be generated at the start of the procedure 502. As candidate data sets are processed, all key masks may be loaded into computer memory. The value of each field in the candidate can be tested against each of these Bloom filters with minimal additional overhead.

Once all values are processed, the statistics are recorded and analyzed 510. Statistics can be recorded for each field that contains values that are compared to the filter mask.

Additional statistics can be calculated; for example, the procedure may determine statistics may be gathered for each column, including the number of records, the number of records where the value in the field is found in the filter mask, the number of distinct values in the field, and the Sørensen-Dice coefficient (a statistic used to compare the

similarity of two samples). Other statistics used to compare the similarity between the potential key field and the target field may be determined. For example, the range (hi/lo), mean, geometric mean and standard deviation of numeric keys values of the primary table.

5 In some implementations, the comparing count, number of distinct values and Percent Intersection (that is, the percentage of values that occur in both the field being tested and the field being compared) and Sørensen-Dice Coeff are computed. For numeric keys the standard deviation can be computed.

10 In some implementations, the statistics may be formatted into a report and presented to a user.

15 In some implementations, the report may identify one or more fields that are matches for a foreign key primary key relationship. For example, by comparing the percentage of records where the value in the field is found in the filter mask to a threshold, for example, 60%, and comparing the Sørensen-Dice coefficient to a threshold, for example 80%, and the percent intersection to a threshold, for example 95%. Fields which match all the thresholds are presented as potential matches to a user.

 In scenarios in which the data sets use system-wide surrogate keys, potential matches may be determined automatically.

20 In some implementations, a report may be presented to a user to enable the user to identify potential foreign key matches.

25 The database analysis approach described above can be implemented using a computing system executing suitable software. For example, the software may include procedures in one or more computer programs that execute on one or more programmed or programmable computing system (which may be of various architectures such as distributed, client/server, or grid) each including at least one processor, at least one data storage system (including volatile and/or non-volatile memory and/or storage elements), at least one user interface (for receiving input using at least one input device or port, and for providing output using at least one output device or port). The software may include one or more modules of a larger program, for example, that provides services related to
30 the design, configuration, and execution of dataflow graphs. The modules of the program

(e.g., elements of a dataflow graph) can be implemented as data structures or other organized data conforming to a data model stored in a data repository.

The software may be provided on a tangible, non-transitory medium, such as a CD-ROM or other computer-readable medium (e.g., readable by a general or special purpose computing system or device), or delivered (e.g., encoded in a propagated signal) over a communication medium of a network to a tangible, non-transitory medium of a computing system where it is executed. Some or all of the processing may be performed on a special purpose computer, or using special-purpose hardware, such as coprocessors or field-programmable gate arrays (FPGAs) or dedicated, application-specific integrated circuits (ASICs). The processing may be implemented in a distributed manner in which different parts of the computation specified by the software are performed by different computing elements. Each such computer program is preferably stored on or downloaded to a computer-readable storage medium (e.g., solid state memory or media, or magnetic or optical media) of a storage device accessible by a general or special purpose programmable computer, for configuring and operating the computer when the storage device medium is read by the computer to perform the processing described herein. The inventive system may also be considered to be implemented as a tangible, non-transitory medium, configured with a computer program, where the medium so configured causes a computer to operate in a specific and predefined manner to perform one or more of the processing steps described herein.

A number of embodiments of the invention have been described. Nevertheless, it is to be understood that the foregoing description is intended to illustrate and not to limit the scope of the invention, which is defined by the scope of the following claims.

Accordingly, other embodiments are also within the scope of the following claims. For example, various modifications may be made without departing from the scope of the invention. Additionally, some of the steps described above may be order independent, and thus can be performed in an order different from that described.

What is claimed is:

1. A method including:
receiving an identification of a first field in a first data set, the first data set including a plurality of records;
identifying a set of values, the set including, for each record, a value in the respective record associated with the field;
generating a filter mask based on the set of values, where application of the filter mask is capable of determining that a given value is not in the set of values;
receiving a second data set, the second data set including a second field, the second data set including a plurality of records;
determining a count of a number of records in the second data set having a value associated with the second field that passes the filter mask; and
storing the count in a profile.
2. The method of claim 1, further including determining the Sørensen-Dice coefficient of the set of values and the records in the second data set having a value associated with the second field.
3. The method of claim 1, wherein generating a filter mask includes for each value in the set of values, generating a filter key for the corresponding value.
4. The method of claim 3, wherein generating a filter key for the corresponding value includes:
generating a hash value for the corresponding value;
segmenting the hash value into a predetermined number of integers;
creating a filter key by setting bits in a bit vector based on the integers.

5. The method of claim 3, wherein generating the filter mask further includes performing a binary operation on each of the generated filter keys.

6. The method of claim 5, wherein determining a count of a number of records in the second data set having a value associated with the second field that passes the filter mask includes:

- calculating a second filter key for a value associated with the second field;
- comparing the second filter key to the filter mask.

7. A non-transitory computer storage medium encoded with computer program instructions that when executed by one or more computers cause the one or more computers to perform operations comprising:

- receiving an identification of a first field in a first data set, the first data set including a plurality of records;
- identifying a set of values, the set including, for each record, a value in the respective record associated with the field;
- generating a filter mask based on the set of values, where application of the filter mask is capable of determining that a given value is not in the set of values;
- receiving a second data set, the second data set including a second field, the second data set including a plurality of records;
- determining a count of a number of records in the second data set having a value associated with the second field that passes the filter mask; and
- storing the count in a profile.

8. The medium of claim 7, further including determining the Sørensen-Dice coefficient of the set of values and the records in the second data set having a value associated with the second field.

9. The medium of claim 7, wherein generating a filter mask includes for each value in the set of values, generating a filter key for the corresponding value.

10. The medium of claim 9, wherein generating a filter key for the corresponding value includes:

- generating a hash value for the corresponding value;
- segmenting the hash value into a predetermined number of integers;
- creating a filter key by setting bits in a bit vector based on the integers.

11. The medium of claim 9, wherein generating the filter mask further includes performing a binary operation on each of the generated filter keys.

12. The medium of claim 11, wherein determining a count of a number of records in the second data set having a value associated with the second field that passes the filter mask includes:

- calculating a second filter key for a value associated with the second field;
- comparing the second filter key to the filter mask.

13. A system comprising:
one or more computers and one or more storage devices storing instructions that are operable, when executed by the one or more computers, to cause the one or more computers to perform operations comprising:

- receiving an identification of a first field in a first data set, the first data set including a plurality of records;
- identifying a set of values, the set including, for each record, a value in the respective record associated with the field;
- generating a filter mask based on the set of values, where application of the filter mask is capable of determining that a given value is not in the set of values;
- receiving a second data set, the second data set including a second field, the second data set including a plurality of records;
- determining a count of a number of records in the second data set having a value associated with the second field that passes the filter mask; and
- storing the count in a profile.

14. The system of claim 13, further including determining the Sørensen-Dice coefficient of the set of values and the records in the second data set having a value associated with the second field.

15. The system of claim 13, wherein generating a filter mask includes for each value in the set of values, generating a filter key for the corresponding value.

16. The system of claim 15, wherein generating a filter key for the corresponding value includes:

- generating a hash value for the corresponding value;
- segmenting the hash value into a predetermined number of integers;
- creating a filter key by setting bits in a bit vector based on the integers.

17. The system of claim 15, wherein generating the filter mask further includes performing a binary operation on each of the generated filter keys.

18. The system of claim 17, wherein determining a count of a number of records in the second data set having a value associated with the second field that passes the filter mask includes:

- calculating a second filter key for a value associated with the second field;
- comparing the second filter key to the filter mask.

19. A system comprising:

- means for receiving an identification of a first field in a first data set, the first data set including a plurality of records;

- means for identifying a set of values, the set including, for each record, a value in the respective record associated with the field;

- means for generating a filter mask based on the set of values, where application of the filter mask is capable of determining that a given value is not in the set of values;

means for receiving a second data set, the second data set including a second field, the second data set including a plurality of records;
means for determining a count of a number of records in the second data set having a value associated with the second field that passes the filter mask;
and
means for storing the count in a profile.