

Methods, systems and computer-readable media for generating unique identifiers using hashing

Technical Field

[1] Embodiments generally relate to methods, systems and computer-readable media for creating unique identifiers from multi-field data objects, and in some embodiments, for creating unique identifiers from multi-layer, multi-field data objects. Some embodiments relate to using the unique identifiers for query matching within a data structure to return a canonical identifier related to the data object.

Background

[2] In systems which store entity information, entities may be modelled at various levels of granularity called ID types. For example, in ID type 1, an entity may be represented by only the most generic data specific to the entity within the system. In more granular ID types, additional pieces of data further identify the entity in the more specific context represented by each ID type.

[3] Notably, existing systems may not directly model real-world entities and aspects thereof. Rather, these systems may model entities as curated sets of information that reflect data captured about how users interact with entities and refer to them within a system, for example, on an accounting platform. As a result, there is no reliable domain identifier by which to identify entities. That is, there is no external system or authority in which the entities exist in the form represented within a register system and so the task of identifying entities cannot be deferred to external systems and/or authorities.

[4] However, generating and assigning universally unique identifiers to sets of entity data is complex and problematic. For example, generating random unique identifiers for entity data may result in previously identified sets of entity data being given a new unique identifier (thereby duplicating data within a system) if checks are

not first performed to see if the set of business data already exists, or sets of unrelated entity data may be assigned the same unique identifier, resulting in data pollution. Sets of entity data may also change or be updated and, in some circumstances, it is not desirable for the changed set of data to be associated with the previous unique identifier. That requires careful management in the data handling processes and careful consideration of how unique identifiers are constructed and generated.

[5] There are additional complexities and difficulties around creating an appropriate form for the generated unique identifier to take, preventing accidental cases of the unique identifier generation producing the same unique identifier for different sets of input data, allowing the unique identifier generation to account for as yet unknown future modelling of entities without invalidating existing IDs, and ensuring that new sets of unique identifiers can be created in accordance with a new process that are different from previous sets, without accidentally producing identical unique identifiers across the new and old generation policies.

[6] Any discussion of documents, acts, materials, devices, articles or the like which has been included in the present specification is not to be taken as an admission that any or all of these matters form part of the prior art base or were common general knowledge in the field relevant to the present disclosure as it existed before the priority date of each claim of this application.

Summary

[7] Some embodiments relate to a method, including: determining a set of one or more entity attributes to populate an entity data object associated with an entity; populating one or more of a plurality of fields in the entity data object with the one or more entity attributes, each entity attribute populating an associated field of the plurality of fields; transforming the entity data object into a string; and generating a hash from the string to define a unique identifier which represents the entity data object.

[8] In some embodiments, transforming the entity data object into a string may include: determining the one or more entity attributes from the entity data object; transforming each entity attribute into an entity attribute string; and concatenating each entity attribute string to form a plaintext string. Transforming each entity attribute into an entity attribute string may include prepending or appending a delimiter to each entity attribute string.

[9] Generating a hash may include applying a hash function to the plaintext string to generate a resultant hash. In some embodiments, applying the hash function to the plaintext string may include adding a pepper to the plaintext string. Applying the hash function to the plaintext string may include adding a salt.

[10] In some embodiments, generating a hash may include prepending a marker to the resultant hash to define the unique identifier. The hash function may be an MD5 hash function.

[11] In some embodiments, the unique identifier may include a marker. The marker may be positioned at the beginning of the unique identifier.

[12] In some embodiments, the method may include determining an identification type of the entity data object on the basis of the subset of the plurality of fields which are populated; and associating the entity data object with the determined identification type.

[13] The method may include writing the unique identifier to a data structure. In some embodiments, the method may include associating a first attribute of the one or more entity attributes with the unique identifier. The first attribute may be a brand or business name. The first attribute and the unique identifier may be stored within a first object in a database structure.

[14] The one or more entity attributes may include at least one of a canonical identifier, user recognisable name, brand name, business name, registration type,

business number, registration identifier, location information, country, state, suburb, contact information, address, registered name, and/or associated aliases.

[15] In some embodiments, the plurality of fields may include at least one of a brand, registered name, registration type, registration ID and location information. The method may include determining a set of one or more entity attributes to populate a set of entity data objects associated with an entity.

[16] The method may include associating the entity data object with at least one other entity data object, the entity data object and the other entity data object sharing at least one entity attribute in a common field, to create a multi-layer data object. In some embodiments, the at least one entity attribute in the common field may be a brand. In some embodiments, the at least one entity attribute in the common field may be a business number.

[17] In some embodiments, the entity data object may be stored within a data structure. Transforming the entity data object into a string may involve converting populated data in each of the plurality of fields into a string.

[18] Some embodiments relate to a method of generating a unique identifier from an entity data object, may include: determining one or more entity attributes from the entity data object; transforming each entity attribute into an entity attribute string; attaching a delimiter to each entity attribute string; concatenating each entity attribute string to form a plaintext string; applying a hash function to the plaintext string to generate a resultant hash, wherein applying a hash function includes adding a pepper to the plaintext string; and prepending a marker to the resultant hash to define the unique identifier which represents the entity data object.

[19] Some embodiments relate to a system, including: one or more processors; and memory comprising computer executable instructions, which when executed by the one or more processors, cause the system to perform any of the methods described herein.

[20] Some embodiments relate to a computer-readable storage medium storing instructions that, when executed by a computer, cause the computer to perform any of the methods described herein.

Brief Description of Drawings

[21] Various ones of the appended drawings merely illustrate example embodiments of the present disclosure and cannot be considered as limiting its scope.

[22] Figure 1A is a schematic representation of an individual entity data object, according to some embodiments;

[23] Figure 1B is a schematic representation of a plurality of associated entity data objects that form a multi-layer data object, according to some embodiments;

[24] Figure 2 is a schematic representation of the structure of a plurality of multi-layer data objects having assigned identification types, according to some embodiments;

[25] Figure 3A is a flow diagram showing a method of creating a unique identifier, according to some embodiments;

[26] Figure 3B is a flow diagram illustrating a method of generating a unique identifier from an entity data object, according to some embodiments;

[27] Figure 3C is a diagram illustrating a construction of plaintext to generate a unique identifier by hashing, according to some embodiments;

[28] Figure 4 is a schematic illustration of a business register storing a set of unique identifiers generated from entity data objects, according to some embodiments;

[29] Figure 5A is an example extract of an entity data object populated with an entity attribute, according to some embodiments;

[30] Figure 5B is a second example extract of an entity data object populated with entity attributes and associated aliases, according to some embodiments;

[31] Figure 5C is an example extract of a multi-layer entity data object comprised of a plurality of related entity data objects populated with entity attributes, according to some embodiments; and

[32] Figure 6 is a schematic illustration of a communications system, according to some embodiments.

Description of Embodiments

[33] Embodiments generally relate to methods, systems and computer-readable media for generating multi-field entity data objects that provide representations of entities and their attributes, such as businesses and brands. In some embodiments, unique identifiers are generated from entity data objects, to uniquely identify entities or brands by the attributes that form the entity data object. Some embodiments relate to storing the unique identifier in a graph data structure, business register or other data store and/or updating a graph data structure, business register or other data store with the unique identifier. Some embodiments relate to linking and/or associating multiple entity data objects to form a multi-level data object which represents the same entities or attributes to varying levels of granularity.

[34] Entity information is represented in a variety of different ways, including brands, registered business names, and identifiers such as business numbers. Each entity may use a different way of representing information and structuring their brands and organisation. In some cases, a single legal entity may hold a number of different brand names, registered business names, and identifiers. For example, a legal entity “Company A Pty Ltd” may have a brand for its range of grocery stores “Super

Market”, and a separate brand for its range of petrol stations “Petrol Stop”. In another example, a legal entity “Company A Pty Ltd” may have a single brand “Comp A” that is used to represent a telecommunications company with registered name “Comp A Telecommunications” and an energy provider company “Comp A Energy Services”. Although both have distinct registered names, they are recognised by the brand “Comp A” and they may or may not share the same business identifier.

[35] Entity representations and entity attribute information which appears in raw data, documents, financial transactions and the like, may be represented in a multitude of formats with variances in how entity attributes are represented. Additionally, there are differences in how entities are represented to users, and how users recognise entity attributes including names and brands. For example, a user may recognise that they are interacting with an entity using the brand “Woolworths” without knowing whether the entity they are interacting with is using the registered name “Woolworths Metro” or “Woolworths Online”. Entity attributes representing the registered name, location and registration type or identifier are not necessarily available to users, and in some cases may not be identified from raw data. Being able to provide unique representations of entities across multiple levels of granularity may be useful in improving query matching, for example in search and reconciliation applications, and for populating and maintaining a business register or data store. Using multi-field entity data objects to generate unique identifiers for entities allows business and brands to be represented with varying levels of granularity in a business register, data store or graph data structure in a structured, consistent and clear manner.

[36] An entity data object is a data structure which includes information representing an entity. In some embodiments an entity data object includes information which represents an entity to a specific level of granularity. The data object may be any defined data structure that is used to store or reference data. For example, the data object may include, but is not limited to, tables, views, arrays, lists, clusters, sequences, and indexes. An entity may include a business, organisation and/or contact. The entity data object includes a plurality of fields, each field representing an attribute or

characteristic to be populated. In this way, the entity data object is able to represent multiple attributes associated with an entity.

[37] **Figure 1A** is a schematic representation of an individual entity data object 100, according to some embodiments. The data object has five fields 102, and an associated identification type 104. Only two of the five fields 102 are populated: the first field 106, representing a canonical identifier such as a brand, and the fifth field 108 representing location information. The fields are populated with entity attributes that correspond to the fields.

[38] **Figure 1B** is a schematic representation of a plurality of associated entity data objects 150a, 150b, 150c, according to some embodiments. The data objects 150a to 150c each have a plurality of fields 152. Each data object has a different subset of fields populated with entity attributes. Data object 150a includes the first field 154 populated. Data object 150b includes the first field 154 and the fifth field 156 populated. Data object 150c includes the first field 154, and the third to fifth fields 156, 158, 160 populated. Although each of the plurality of objects includes a different subset of populated fields, the first populated fields 154 of each of the data objects 150a, 150b, 150c all share the same first attribute that populates 154. For example, they may all share the same brand name. Accordingly, the data objects are linked together by this attribute to form a multi-layer data object 168. In some embodiments, the data objects may be linked together with an edge or relationship identifier. In some embodiments, the identification type associated with the data object is implicitly represented by an edge that exists between two data objects. For example, an edge “is more specific than” provides an indication of the relationship of IDs of different types relative to one another. In some embodiments, the data objects are associated with one another through one or more shared attributes. The first attribute may be a canonical identifier, brand, user recognisable name, or business name. Each of the data objects 150a to 150c includes an identification type 162, 164, 166 associated with the attributes and based upon the subset of populated fields. Each data object has a different subset of populated fields, and therefore each of the data objects represents a different identification type.

That is, each of the data objects 150a, 150b, 150c, represents an entity with a brand 154 with a different level of granularity.

[39] In some embodiments, the entity data object is initially empty, with each of the plurality of fields being empty, and are populated once attributes are determined from a data source and/or raw data. In some embodiments, one or more fields may be set to NULL. In some embodiments, the entity data object may be generated with one or more of the plurality of fields populated. The plurality of fields may include, but are not limited to, brand, entity identifier, location, registered name, registration type, aliases, and/or legal entity.

[40] In some embodiments the entity data object includes five fields. The five fields may represent brand (or a canonical identifier), registered name, registration type, registration ID, and location information (such as country or market). The brand field may refer to a canonical identifier or user recognisable name, and may include the marketed name that is aimed at creating a lasting impression in the minds of the public, for example, to users of an accounting platform. The brand entity attribute corresponding to the brand field may be established using a manual annotation process. The brand field may be referred to as the first field, and the brand attribute may be referred to as the first attribute. In some embodiments, the first attribute is returned upon querying the data object or its associated vertices.

[41] The registered name field may refer to a legally-registered name of an entity or business (for example, a franchise, bricks & mortar shop or a sub-specialisation of a corporation or brand) that was patronised and that is different from the brand and the legal entity name associated with the registration type/ID pair. The registration ID field refers to an identifying value or identifier, such as a national business number, which represents the legal entity and/or is associated with a unique legal entity name. The registration type field refers to the type of identifying value or identifier which appears in the registration ID field. For example, a national business number may have a type Australian Business Number (ABN). The country field refers to the location or country

within which the entity, brand or business is active. In some embodiments, the registration type may be used to infer the attribute for location information.

[42] Entity attributes may include any information or characteristics relating to an entity. In some embodiments, entity attributes may include information which represent the entity to a user. For example, entity attributes may include brand names, registration information such as business numbers, registered names, registration types, location information including addresses, legal entity names, contact information, and the like. In some embodiments, the entity attributes may comprise other publicly available information and/or information which is otherwise relevant or related to the representation of the entity. Entity attributes are used to populate the plurality of fields in an entity data object, with each individual entity attribute being assigned to an associated distinct field of the plurality of fields. In some embodiments, a single entity attribute is used to populate a single field, or a single field may be populated with a plurality of entity attributes of the same type. For example, an entity attribute “Woolworths” may be used to populate the field “Brand Name” for an entity data object. Once populated, no other entity attributes may be stored against that specific field. In some embodiments, when a change has occurred to an entity that warrants the changing or updating of one or more entity attributes, such as where a company rebrands, changes structure or changes ownership, a new unique ID (or set of unique IDs) is created. The old or outdated entity data may be stored for a historical unique identifier, deleted from the data structure, data store or data base, or be retired or marked as a “historical” identifier to avoid any associated data, such as old brand names, being returned in response to a query.

[43] Entity attributes may be determined, or inferred, from raw data. The raw data may be fragmented, inconsistently named, under documented and/or difficult to interpret. The raw data may be data collected and stored in a database. For example, the raw data may be accumulated and stored in a database accessible to an accounting system or platform for managing accounts (such as maintaining bookkeeping accounts) associated with a plurality of businesses or organisations. In some embodiments the

entity attributes may be extracted from an external source. The external source may include documents, such as financial documents including invoices or statements, which may have been generated by, or uploaded to, the accounting platform. Alternatively or additionally, the external source may include one or more databases, such as an external database in communication with the accounting platform. The entity attributes extracted from external documents or databases may come from data lines of a financial statement which represent individual transactions and their associated details. In some embodiments, entity attributes may be extracted or determined from a data store, such as a graph structure, or other database. In some embodiments, the entity attributes may be transmitted or provided by an external network, system or database.

[44] Multiple entity data objects which share at least one entity attribute may be linked together to form a multi-layer entity data object. In some embodiments, the multi-layer data object may include a plurality of entity data objects, each sharing at least one entity attribute with at least one other entity data object in the multi-layer data object. In some embodiments, a distinct identification type may be associated with each individual data object in the multi-layer data object to provide a structured representation of an entity with varying degrees of granularity.

[45] Entity data objects may include an identification type. An identification type represents a level of granularity associated with the entity data object. Identification types may be assigned, and/or associated with the entity data object. Identification types may be used to structure a plurality of entity data objects. In some embodiments, identification types may be used to distinguish entity data objects which share at least one entity attribute. The identification type of an entity data object may be determined on the basis of the plurality of fields. For example, the identification type may be determined on the basis of the number of fields which are populated in an entity data object, or on the basis of the number of empty fields in an entity data object. In some embodiments, the identification types will not always utilise every one of the plurality of fields and, in some embodiments, irrelevant fields for a given identification type may be set to NULL. In some embodiments, a first field may be present in all identification

types. For example, a first field which is “Brand” or “Brand Name” may be present in all identification types for a multi-layered data object. In some embodiments, the identification types assist in distinguishing registered subsets of a particular brand to a specific level of granularity.

[46] **Figure 2** is a schematic representation of a plurality of multi-layer data objects 200, 206, 208, 210, each including a plurality of data objects having an assigned identification type structured hierarchically in order of granularity, according to some embodiments. A multi-layered data object 200, contains a plurality of entity data objects 202, 203, 204, each having an assigned identification type. In this example, the identification type of each of 202, 203, and 204 is distinct, and no two identification types are the same. However, in some embodiments, identification types may be the same where there are differences between the entity attributes which populate the entity data objects, for example, where the brand name or location information is different. The data objects 202, 203, 204 are structured based on the level of identification types to distinguish between the number and/or subset of entity attributes, that is, the representative information about the entity, which are present within each data object. In some embodiments, a multi-layered data object will always include an ID type 1. For example, if an ID type 2 exists, the ID type 2 will always ladder up to a more generic ID type 1. In some embodiments, a type 1 data object is not included in a multi-layered data object, and the multi-layered data object may contain any subset of data objects having any identification types. For example, the multi-layered data object 208 includes only an ID type 2 and an ID type 3 data object.

[47] A first identification type, or ID Type 1, may relate to a generic underspecified brand, which includes only a canonical identifier, such as a brand name. A second identification type, or ID Type 2, may relate to a region-specific representation of a generic brand or a national brand that isn’t tied to a specific registration. For example, when brands are used by multiple legal entities in a single market. ID Type 2 may be assigned when a data object includes entity information that specifies a brand that is only present in one market or country, and legal entity name is not included, and/or

when the query indicates a brand that is generic, as well as market or country information, and legal entity name information is not included. A third identification type, or ID Type 3, may relate to a generic or nationally-recognised brand that is tied to a specific registration. This is useful, for example, where there is a need to reflect legal entities within the data object and/or where the legal entity information is required to be stored. Some nationally-recognisable brands are associated with multiple legal entities that have separate but related business models. Accordingly, ID Type 3 may be assigned or associated with a data object when the entity attributes which are populated include brand, country, registration ID and registration type. ID type 3 may also include a franchisee that uses the brand in a specific location.

[48] A fourth identification type, or ID Type 4, may be the most granular type for a data object. In some embodiments, ID Type 4 may have all fields of the plurality of fields populated within the entity data object. ID Type 4 may relate to a bricks and mortar shop, franchise, or legal entity that manages multiple brands, products, business functions or sub-products. For example, brands in the convenience store and petrol industries are often used by franchisees, and these franchisees legally register only the specific shop they own, for example, “Company A Pty Ltd” may have legally registered the business name “Petrol Stop 100”, which is a franchise of the “Petrol Stop” business that is specifically located in Sydney, Australia. Additionally, legal entities may cover multiple brands (such as Woolworths and Big W) and/or multiple products, business functions, or sub-brands (for example, Woolworths Insurance, Woolworths Mobile, Woolworths Online, Woolworths Metro). ID Type 4 represents a level of granularity that is likely to be recognised and which may be present in statement lines and receipts and, as such, may be used for accurate query matching. For example, when a query matches a registered name that is not a brand or legal entity name, it is able to return information associated with the data object assigned ID Type 4. In some embodiments, there may be more or less identification types based on the structure of the data object and the number of fields in the data object.

[49] In some embodiments, the four identification types may be used as follows:

- ID Type 1: The generic brand that is international, and thus recognisable across multiple locations or markets;
- ID Type 2: A generic brand in one location;
- ID Type 3: The legal entity that owns the brand or a franchisee that uses the brand in a specific location; and
- ID Type 4: A specific geographic instance of a franchise that uses the brand.

[50] Based on the four identification types, in Figure 2 multi-layer data object 200 relates to a generic brand that has brand recognition in at least one market and for at least one legal entity. Multi-layer data object 206 includes four identification types and relates to a generic brand that has at least one legal entity in at least one market and recognised franchises, sub-brands, and/or functions. Multi-layer data object 208 relates to a market specific brand that has at least one legal entity, and multi-layer data object 210 relates to a market-specific brand that has at least one legal entity and recognised franchisees, sub-brands and/or functions.

[51] Entity data objects may be created, stored, read from and/or written to a graph data structure. The graph data structure (or “graph”) may represent a network of entities (such as people or things), for example entities associated with an accounting platform, and the information and relationships known about those entities. The graph may be an enriched information interface, which may be capable of providing a coherent and/or enhanced view of knowledge about the network of entities. The graph may be initially generated or created based on raw data derived from how the accounting platform is used by organisations to manage their accounting needs.

[52] The graph data structure may comprise a plurality of nodes or vertices, some of which are connected or interconnected with or to other vertices by edges. Each entity data object may be stored as an individual vertex. The graph data structure may include vertices of different types, including, but not limited to, entities, unique identifiers (such as business identifiers), aliases, canonical identifiers, canonical objects,

organisations, apps, employees, contacts, business types, partners, interactions such as events or transactions and the like. In some embodiments, the entity data object may be associated with vertices of the same type, such as other entity data objects, and/or vertices of different types, such as unique identifiers vertices.

[53] The vertices of the graph may be determined or generated from the raw data. One or more vertices may be associated with entity attributes of the raw data, for example, attributes or characteristics that represent a real world entity. Accordingly, vertices may be determined, in that they may be identified or obtained directly from the raw data. Where the vertex is an object, it may be mapped to an object of the raw data. This mapping may be performed by using an object identifier to map the object of the raw data to an object in the graph structure. For example, the raw data object extracted may be a transaction which includes an object identifier in the form of a brand name, such as “Woolworths”. This brand name is then used to map the raw data object to a unique identifier vertex representing an entity to a specific level of granularity, where the unique identifier vertex includes an identifier which represents the information from an associated entity data object. This enables a target entity from raw data to be readily identified in a graph structure or business register by using attributes related to how the entity is represented.

[54] In some embodiments, vertices may be inferred or deduced from the raw data, or from objects of the raw data, or from vertices and/or edges of the graph itself. Vertices and edges may be updated or inserted based on inferences or deductions made from the objects of the raw data.

[55] Each edge defines a relationship between two vertices. An edge is representative of a fact or state that holds between the two vertices. The relationship may be determined from the raw data, or maybe inferred from the raw data based on interactions or transitivity between objects of the raw data, and/or edges and/or vertices in the graph. In some embodiments, for example, an edge connecting an entity data object vertex to a unique identifier vertex may be of the type “has business ID”. In

some embodiments, an inferred relationship (or inferred edge) may not be captured in the raw data. The inference may be based, for example, on properties of the two vertices joined or connected by the inferred edge (for example, similarity in name and potentially similarity in other metadata), interactions that have been recorded (in the raw data and/or the graph) between the items of information (objects or inferred objects) represented by the specific vertices, the nature of interactions in the raw data (such as reconciling financial transactions) or logical deductions.

[56] The relationships between vertices may also be determined or inferred from data extracted from an external or associated data source. New relationships may be generated through inferences and determinations made by extracting or reading data from data sources before being updated in the graph structure. This may include creating new links and associations between vertices in the form of edge tables, or updating the nature of the associations by updating, changing or swapping edge tables, or fields within edge tables. For example, where entity attributes of an entity are extracted from a financial statement it may be determined that those entity attributes match a unique identifier for a specific brand. In this case, a new edge object may be created having a relationship type such as “is associated with” or “is represented by” between the entity or entity attributes and the unique identifier.

[57] In some embodiments, additional or aggregated information about the vertices and/or edges may be surfaced or stored as metadata annotated on the respective vertex and/or edge. For example, where two vertices are interconnected by an edge that is representative of the relationship between the two vertices, additional or aggregated information about interactions between the two vertices may be recorded or stored as metadata annotated on the respective edge. This may be stored by updating specific fields in the edge tables or vertex tables. In some embodiments, one or more entity attributes (for example, brand name) may be stored as metadata on the vertex which represents the unique identifier, or may be stored within a field of the vertex which represents the unique identifier.

[58] Entity data objects may be used to generate unique identifiers that represent the data objects and the attributes which populate them. Unique identifiers may be an object, and/or may be a value, for example, a string. In some embodiments, the unique identifier may be a string, for example, as output by a hash function. The string may have a fixed number of characters. In some embodiments, a single unique identifier may be created for an individual entity data object. For example, a single unique identifier may be created for an entity data object which is assigned ID Type 1. If a related entity data object existed that was assigned ID Type 2, this would contain different or additional entity attributes, and therefore would have its own unique identifier. In some embodiments, the unique identifier may be created from an entity data object that does not have an assigned or associated identification type. In some embodiments, the unique identifier is a business identifier, entity identifier, or business ID.

[59] **Figure 3A** is a flow diagram showing a method 300 of creating a unique identifier, according to some embodiments. The method includes determining a set of one or more entity attributes to populate an entity data object associated with an entity (310). Determining a set of one or more entity attributes may include receiving entity attributes from an external source, or extracting the entity attributes from a data source, for example from a line in a financial statement. The entity attributes are stored as a set, that is, they relate to a single entity. Next, the method populates one or more of a plurality of fields in the entity data object with the one or more entity attributes (312). Each individual entity attribute is assigned to an associated individual field of the plurality of fields within the entity data object. The method may include determining an identification type of the entity data object on the basis of the subset of the plurality of fields which are populated (314). For example, where it determines that only a single field “Brand Name” is populated with an associated entity attribute, the identification type of the entity data object may be determined to be ID Type 1. The determined identification type is then assigned to the entity data object (316). In some embodiments, this may be assigned by creating an associated identification object,

storing the identification type in the metadata of the entity data object, or by appending an additional field representing identification type.

[60] The entity data object is then transformed into a string (318). In some embodiments, the contents of the entity data object are transformed into a string, including the populated entity attributes and the NULL values of empty fields. In some embodiments, only the populated entity attributes are transformed into a string. In some embodiments, the identification type may or may not be included in the contents of the entity object which are transformed to a string. Once the entity data object has been transformed, a hash is generated from the string to define a unique identifier which represents the entity data object and its associated identification type (320). The hash may be generated from the string by applying a hash function. The generated hash defines a unique identifier.

[61] Advantageously, the methods described herein for generating unique identifiers which represent entities are derived from the sets of entity attributes that form an entity data object. This prevents unintentionally duplicating sets of entity attribute data and avoids the risk of the entity attribute data associated with a unique identifier changing, and thereby changing what the unique identifier implicitly represents.

[62] In some embodiments, the entity data object is transformed into a string by extracting each of the populated entity attributes from the data object and converting the value of each of the populated entity attributes into a string. For example, an entity attribute “*Petrol Stop*” for the brand entity attribute is converted to a string “Petrol Stop”. Once each populated attribute is transformed into a string, the strings representing each entity attribute, also referred to as entity attribute strings, are concatenated to form a concatenated string, or plaintext. The plaintext defines the string which is formed from one or more entity attribute strings. In some embodiments, the plaintext refers to unencrypted information before it is input into an encryption

algorithm, for example, a hash function. In some cases, the plaintext may also include a salt and/or a pepper.

[63] In some embodiments, the entity attribute strings are concatenated sequentially in the order in which they are transformed, in an order corresponding to the order of the plurality of fields, in an order representing identification types, or a predetermined order specified by a transformation policy. For example, the plaintext may be formed by concatenating entity attribute strings in the order of brand, country, registration type, and registration ID, in which the addition of each entity attribute string being appended is a result of the data entity object belonging to a specific identification type. The construction of the plaintext encodes the data in the entity data object representing an entity. The plaintext may be formed from at least one entity attribute string.

[64] In some embodiments, the plaintext is formed from a concatenation of entity attribute strings. For example, entity attributes may be transformed into strings, and these entity attribute strings may define string portions that make up the plain text. The plaintext may include one or more string portions representing one or more entity attributes. The number of string portions used to create the plaintext may be determined by the ID Type associated with the entity data object.

[65] In some embodiments, a unique identifier may be generated according to a particular set of predefined processes, for example, the set of predefined processes may form a policy which defines how the entity attributes are transformed into a string and hashed to result in the unique identifier.

[66] In some embodiments, only populated entity attributes are transformed into strings, and empty fields which have a NULL value are not transformed into strings. In other embodiments, fields with NULL values are transformed into string “null” and concatenated with the entity attribute strings to form the plaintext.

[67] In some embodiments, the predefined processes for generating the unique identifier avoid containing markers that indicate the non-existence of particular entity attribute data for a specified ID type. For example, an entity data object which has an ID Type 1, has null country, registration type and registration ID details. As such, that data is omitted from the plaintext. That is, instead of appending “|null|null|null”, no entity attribute string is appended for those null entity attributes. This enables flexibility to extend, change or modify the predefined processes, or policy, to represent unique identifiers of future types without causing previous unique identifiers to change. For example, a future type of identification, ID Type 4, that extends ID Type 3 with a new entity attribute of “registered name”. By definition, “registered name” would be null for ID Types 1, 2, and 3. Introducing a new entity attribute of “registered name” would mean having to append the “null” value for ID Types 1, 2, and 3 to the plaintext and regenerate a unique identifier for that entity data object, effectively rendering the old unique identifier (without the “null” value of the registered name entity attribute) invalid or deprecated. By omitting the non-existence of the entity attribute (or implying the non-existence by its absence) when transforming the entity attributes into the plaintext, the predefined processes or policy may be extended to incorporate Type 4 IDs without invalidating and/or deprecating more specific or less specific ID Types.

[68] In some embodiments, the policy for generating the plaintext is designed to substantially avoid collisions with IDs for existing types. For example, the policy which omits the non-existence of entity attributes and appends populated entity attributes to the right of the string. Concatenating the entity attributes to the plaintext is straightforward and convenient for modelling entities, such as businesses. As the entities represent more data, and the data becomes richer, transforming the entity attributes into strings may become more complex, but unintentional invalidation and/or deprecation of existing IDs may be substantially mitigated.

[69] In some embodiments, a hash is generated from the plaintext to define a unique identifier which represents the entity data object. Changes in the entity attributes of an entity data object result in the identification of a new entity which is represented

by a new unique identifier related to that entity. As the entity attributes which form the strings that make up the plaintext change, the hash of that string also changes.

[70] A “brand” entity attribute may be transformed to a string and define the first string portion that is appended to the plaintext. In some embodiments, the brand entity data attribute is the minimum amount of information that can uniquely identify an entity. For example, a brand entity attribute may be the only populated entity attribute which defines a Type 1 ID. Accordingly, a “brand” entity attribute may be the minimum required populated field to identify an entity within a business register and/or to generate a unique identifier. In some embodiments, where the plaintext may be formed from at least the brand entity data attribute. Plaintext which includes brand entity attribute as the only entity attribute may be categorised as an ID Type 1.

[71] A “location” entity attribute may be transformed to a string and define a second string portion that is appended to the plaintext. In some embodiments, the country or location entity attribute represents the location of operation of an entity, for example, where a business (defined by a brand entity attribute) is operating. The location entity attribute may include a country code. In some embodiments, the location entity attribute is converted to a country code when it is transformed into a string. In some embodiments, the country code is an ISO 3166-1 alpha-2 country code. The plaintext may be formed from at least the brand entity attribute and the location entity attribute. In some cases, the location entity attribute being included in the plaintext is optional. Plaintext which includes brand entity attributes and a location entity attribute as the only entity attributes may be categorised as an ID Type 2.

[72] A “registration type” entity attribute and/or a “registration identifier” entity attribute may be transformed to a string to define a third string portion that is appended to the plaintext. In some embodiments, the registration type and/or registration identifier represent the registration details relating to an entity. For example, the business registration details that are specific to a particular authority, such as a registration type being an Australian Business Number (ABN) and a registration ID

being the number “12345678987”. Although this data is unique to the entity within individual registration schemes, the plaintext is also assembled from the brand and country. This ensures that entity data objects which have registration types and registration identifier entity attributes which are the same (such as the examples of Super Market and Petrol Stop which have the same registration ID shown in Figure 5C) are still able to generate a unique identifier in light of differences between other entity attributes. Thus, it allows the entity registration details associated with multiple brands to create unique identifiers. The plaintext may be formed from at least the brand entity attribute, the location entity attribute, the registration type entity attribute and the registration identifier entity attribute. In some cases, the registration type and/or registration identifier entity attribute being included in the plaintext is optional. Plaintext which includes brand, location, registration type and registration identifier entity attributes may be categorised as an ID Type 3.

[73] Once the plaintext representing the entity attributes of the entity data object is formed, a hash is generated from the plaintext. In some embodiments, generating a hash from the plaintext may include applying a hash function to the plaintext (or hashing the plaintext). Hashing may refer to a deterministic process of reducing input, such as plaintext, to an output value which is compressed representation of the input. That is hashing is the process of taking plaintext and transforming it into another value. This value may be shorter and have a fixed length. This process cannot be reversed to obtain the original data, making the output value something of a “fingerprint” of the input. The output value is a unique representation of the plaintext. Any changes to the plaintext will result in a different output value.

[74] Applying hashing to the plaintext provides a number of unique advantages. Hashing avoids undesirable variability in the unique identifiers, for example, the hash provides a fixed length value, as opposed to widely varying lengths of unique identifiers dependent on what data is captured or included in the plaintext. Hashing also avoids exposing entity data through the unique identifier (and may do so in combination with a pepper, which is described later). Additionally, hashing prevents

unintentional dependencies on the context of the unique identifier, which is unknowable in the resultant hash.

[75] In some embodiments, the hash function applied to the plaintext to generate the hash may include, but is not limited to, SHA (including SHA-1, SHA-2, and SHA-3), CRC, MurmurHash, BLAKE2, Argon2 and/or MD5. In some embodiments, the hash function used is SHA-512 or SHA-256.

[76] In some embodiments, the hash function used to generate the hash from the plaintext is an MD5 hashing algorithm. The MD5 hashing algorithm provides the advantages described above, and in particular, generates a relatively short string as the output hash compared to some of the more cryptographically secure algorithms. A relatively short string as the output hash means that the resulting unique identifier can be shorter, and this can be beneficial for improving user and developer ergonomics. For example, in functionalities of a platform, such as a business register, which enable a user to search and/or lookup an entity by a unique identifier. As users have the capability to interact with URLs, it is also more convenient to have shorter unique identifiers in URLs which are generated by the MD5 hashing algorithm. In some embodiments, the hash function supports the embedding of the unique identifier in REST URL paths.

[77] The number of representable inputs to the hash function is larger than the number of representable outputs, which means that there is a possibility of collisions where different plaintexts result in the same hash, thus compromising uniqueness and presenting an attack vector in hashing for cryptographic purposes. In light of the process used to construct the plaintext, the chance of natural collisions is extremely low, and the threat impact is relatively low, so the MD5 hashing algorithm is sufficient for generating unique identifiers for the methods described herein despite known security weaknesses. That is, since the hashing is being used for abstraction, rather than obfuscation, the benefits of the MD5 algorithm are utilised while the security deficiencies of the MD5 algorithm can be substantially managed and mitigated.

[78] In some embodiments, the construction of the plaintext includes adding one or more delimiters to the plaintext. A delimiter may define one or more characters that specifies the boundary between independent portions in the plaintext. A delimiter may be a character including, but not limited to, a comma (,), semicolon (;), quotation marks (“or ’), braces ({ or }), pipes (|), or slashes (/ or \). The delimiter may be prepended to the left of an entity attribute string, thereby separating the entity data string from what comes before it. Delimiters are particularly advantageous for the hashing methods being used, particularly to avoid accidental collisions between hashing plain text. For example, a delimiter may prevent an accidental collision between entities. As an example, consider two separate brands: “Spackle”, which is operating in Australia, and “SpackleAU”. The plaintext generated from the ID Type 1 (that is, only including brand name in the entity data object) for the globally unique brand “SpackleAU” would be “SpackleAU”. The plaintext generated from the ID Type 2 (that is, including brand name and location data) for brand “Spackle” which operates in Australia, would also be “SpackleAU” (with AU being the ISO-3166-1 Alpha 2 country code). Thus, applying a hashing algorithm to both the plaintexts which are the same would result in the same output hash, and therefore a collision.

[79] However, by prepending a delimiter to the entity attribute string “AU” when it is transformed from an entity attribute, the resultant plaintext would be “Spackle|AU” which is distinguished from “SpackleAU”, and therefore is very likely to provide a differing resultant hash from “SpackleAU”.

[80] In some cases, the delimiter may be prepended at the time of transforming an entity data attribute into a string. In some embodiments, the delimiter may be appended to the right of the entity attribute string. The delimiter may be on some, but not all entity data attributes, and/or may be appended or prepended to one or more string portions. In some embodiments, more than one delimiter may be concatenated to an entity attribute string or a string portion.

[81] In addition to hashing the plaintext, in some embodiments the hashing algorithm may be applied to the plaintext in combination with a pepper and/or a salt. A pepper (also known as a secret salt) is a secret value that is added to the plaintext during generation of the hash. The pepper is a single value which is the same for each application of the hashing algorithm. The pepper may be stored separately to the resultant hash, for example, in a secure database. The pepper may be in the form of a randomised value.

[82] A pepper is used to obfuscate the hashing algorithm, so that an attacker that has access to the data associated with unique identifiers cannot easily determine from the data the entities to which it relates. For example, entity attribute data may be publicly available information (such as brand, location and registration type/ID). If only entity attributes were hashed, an attacker with access to publicly available entity attributes may discover the real-world entities to which unique identifiers are associated, for example, through brute force attacks. The addition of a pepper makes this type of attack much less feasible. Including a pepper when generating the unique identifier for an entity helps to obfuscate the entity associated with the unique identifier where it is being used in platforms, such as an accounting platform. For example, it assists in obfuscating which real-world entities and brands a user of an accounting platform may have interacted with, or where the entity data is being used.

[83] The pepper is prepended to the plaintext during hashing with the hash function. For example, the unique identifier may apply the hash function to the plaintext "<pepper>SpackleAU" which would result in a different unique identifier to only applying the hash function to plaintext "SpackleAU".

[84] In some embodiments, the pepper may be rotated. For example, when implementing a new policy for generating unique identifiers which implies no semantic equivalent between the entities represented by unique identifiers generated by previous policies and the entities represented by unique identifiers generated under the new policy. In some embodiments, rotating the pepper at implementation of a new policy is

a desirable break in continuity between the generated unique identifiers because there is no continuity between what the old and new unique identifiers represent. As such, an intentional change to a new policy provides an opportunity to rotate the pepper since there is already an intention to change the unique identifiers. In some embodiments, the pepper may be rotated and/or updated without a policy change, and without migrating the old unique identifiers, and instead all known entities are re-entered with freshly generated unique identifiers from the new pepper. For example, this may be implemented where there is a need to generate new identifiers but there is no requirement to maintain continuity for all data referring to the same underlying entity.

[85] In some embodiments, a salt may be used in combination with the hashing algorithm, and/or in combination with the hashing algorithm and the pepper. In some embodiments, the hash function used is MD5 with a salt or a pepper, or a salt and a pepper.

[86] In some embodiments, a marker may be prepended to the resultant hash of the plaintext. The marker may be in the form of a character prepended to the resultant hash that represents the process by which the unique identifier has been generated. For example, the marker may represent the version of a process or policy which has been applied to generate the unique identifier. The marker may be a single character, or it may be multiple characters, and may be formed from alphanumeric value and/or symbol or sign. The marker may consist of a single letter, or a single number, or it may consist of a combination of letters, numbers and/or symbols, for example: "0", "1", "2"... / "a", "b", "c"... / "v1", "v2" "v3"... / "1.1", "1.2", "1.3"... and the like. The marker may be selected from the character set [a-zA-F0-9]. In some embodiments, the marker has a fixed position, and may be prepended in front of all resultant hashes generated from entity attributes. In some embodiments, the marker may be appended to the end of the resultant hash. The marker may form part of the unique identifiers to which it is prepended. In some embodiments, the marker may represent the set of predefined processes, or the policy, by which the unique identifier was generated. That is, the marker links the unique identifier to the policy by which the unique identifier

was generated. The marker is a simple and unambiguous way of marking how the unique identifier was generated.

[87] Advantageously, the use of a marker enables provenance of a unique identifier to be known. In a dataset of unique identifiers that may be generated from multiple policies, it would otherwise not be possible to tell which unique identifiers represent the same or similar entities generated by different policies without trying every entity attribute-policy combination. For example, this can be difficult for unique identifiers derived from the same set of entity attributes. A marker enables such a distinction between unique identifiers, for example, a first unique identifier “0abc123d4e5” (generated from a plaintext) has a prepended marker “0” and a second unique identifier “1abc123d4e5” (generated from the same plaintext as the first unique identifier) has a prepended marker “1” which indicates that both of the unique identifiers, despite being a unique set of characters, were generated through a different set of processes. In some embodiments, the marker can provide temporal information such as when the IDs are created, for example, where a marker represents generation by a policy that was in place during a specific timeframe.

[88] Another advantage of using a prepended marker is that it allows for an explicit breaking change to be made across all unique identifiers generated by a policy. For example, the introduction of a new policy may only occur when the modelling of entities within a database, such as a Business Register, changes. It is possible that in both an old and new policy, identical plaintexts are generated through coincidence or through the respective, underlying models of entities depending on the same or similar entity attribute data in plaintext assembly. As such, the resultant unique identifiers would be identical, but the conceptual representations of the entities represented by the underlying entity models and expressed through the unique identifier generation policy would be very different. By prepending a marker and changing it as the policy changes, coincidental collisions between policies can be substantially mitigated or avoided.

[89] Collisions in the unique identifiers are undesirable for various reasons. For example, collisions may cause ambiguity as to which unique identifier should be identified from services configured to return entity information, for example, it may be unclear which unique identifier to return where two of the exact same unique identifiers exist for different entities. Further, there may be ambiguity surrounding which entity was served when service exhaust data is used for other purposes. This may result in data pertaining to the colliding entities (such as links to other data) being grouped together, thereby polluting the data, and making it difficult to extract accurately.

[90] In some embodiments, the marker may not change, and may be a constant character, or set of characters, appended to the resultant hash. The marker may not change where the policy never changes, or where there is no need to change it. In some embodiments, the marker may be kept consistent to avoid breaking changes, for example, when used in a client facing accounting platform, and accepting the costs. However, unique identifiers generated by the first (and potentially only) policy and/or process still benefit from having a prepended unambiguous marker that defines by which process they were generated in order to have the option to update or change the marker, if needed. This is advantageous because once the unique identifiers are generated, they cannot be modified after the fact, since they will be served to and stored by external systems, and any changes would invalidate the existing unique identifiers that are already in use, or require the “old” identifiers to be deprecated.

[91] In some embodiments, the marker may be in the form of another signal to mark the unique identifier generated by a particular policy. In some embodiments, the fixed length hash is used and unique identifiers from different policies are inferred by prepending a marker and then producing longer hashes.

[92] In some embodiments, the resultant unique identifiers are 128 bits plus the marker. For example, the unique identifiers may start with a marker in the form of a version character, “0”, followed by an apparently random sequence of 32 characters from the character set [a-zA-F0-9].

[93] **Figure 3B** is a flow diagram illustrating a method 330 of generating a unique identifier from an entity data object, according to some embodiments. At 332, the entity attributes are determined from the populated fields of an entity data object. In some embodiments, the entity attributes may be determined by extraction from the entity data object. Each entity attribute is then transformed into an entity attribute string at 334, which is a string that represents the content of the entity attribute. In some embodiments, the entity attributes which populate the fields may already be in the form of a string. At 336, a delimiter is prepended to each entity attribute string, and at 338 each entity attribute string is concatenated to form a plaintext. As a result, a delimiter exists between each portion of the constructed hash. A pepper is then determined at 340. Determining the pepper may include receiving the pepper or extracting the pepper to apply to the plaintext at the time of applying a hash function. In some embodiments, determining the pepper at 340 and applying the hash function at 342 are undertaken simultaneously. In some embodiments, determining the pepper is undertaken as part of applying the hash function at 342. In some embodiments, the pepper may be prepended or appended to the plaintext.

[94] At 342, a hash function is applied to the plaintext to generate a hash. The hash function includes applying the pepper to the plaintext and taking a hash of the plaintext plus the pepper. After the hash is generated, at 344 a marker is prepended to the hash, to thereby define a unique identifier representing an entity. The marker may be in the form of a version character. The marker is used to identify the process by which the unique identifier has been generated.

[95] **Figure 3C** is a diagram illustrating the construction of the plaintext to generate the unique identifier by hashing, according to some embodiments. The hash 350 is constructed by determining the entity attributes from an entity data object. The entity attributes are the number of attributes that populate fields in the data object. For example, if only one of a plurality of fields in a data object is populated, then only one entity attribute will be determined to construct the hash. In Figure 3C, the number of entity attributes determined is also based on the ID type of the entity data object, that is,

the level of granularity with which the entity is represented. The hash includes plaintext 352. The plaintext represents a string that will be hashed, the string being constructed from one or more entity attribute strings. The hash function 350 includes a pepper 354 which is prepended to the plaintext formed from the concatenated entity attribute strings at the time of application. The number of entity attribute strings that form the plaintext may be based on the ID type. For example, an ID type 1 plaintext may consist of the pepper 354 and a brand entity attribute string 356. An ID Type 2 plaintext may consist of the pepper 354, a brand entity attribute string 356, and a location entity attribute string 358. An ID type 3 plaintext may consist of the ID Type 2 plaintext, plus a registration type entity attribute string 360 and registration ID entity attribute string 362. In some embodiments, each of the entity attribute strings may include a delimiter, for example, in the form of a pipe “|”. After the hash function 350 is applied to the plaintext, including adding in the pepper 354, a resultant hash 364 is generated. A marker 366 is prepended to the resultant hash 364 to generate a unique identifier 368 that represents an entity with a specific ID Type.

[96] In some embodiments, unique identifiers which are considered “old” (for example, which may have been generated by a previous or outdated policy), outdated, or which have been superseded by a new unique identifier may be deprecated. Unique identifiers may be deprecated by marking the unique identifiers as no longer being in use or having been superseded. For example, this may include tagging or annotating the unique identifier to indicate deprecation. In some embodiments, the deprecated unique identifiers are not removed from the database so as to maintain data continuity and mitigate regression errors. In some embodiments, the deprecated unique identifiers may still be used and/or accessed.

[97] Unique identifiers may be written to a graph data structure, for example, as a vertex. The vertex which contains a unique identifier may be a unique identifier object, or a business identifier object. Alternatively or additionally, a generated unique identifier may be written to a data store or a business register. The unique identifier may be associated with the entity data object from which it is generated. In some

embodiments, the unique identifier may be linked with multiple entity data objects, linked with an individual attribute such as a first attribute or brand, a canonical identifier, or an associated business. In some embodiments, unique identifiers which are generated from entity data objects which share one or more entity attributes may themselves be linked or associated with one another. For example, the unique identifiers may be linked with other data objects through edges, defined relationships, or through metadata applied to the unique identifier. In some embodiments, unique identifiers may be stored in a field of a business identifier object. The business identifier object may also store an associated canonical identifier of brand name and/or an associated identification type, which may also be linked to an entity data object. In some embodiments, the identification type may be associated directly with the unique identifier, for example, through associated metadata or an associated object, field or vertex.

[98] In some embodiments, unique identifiers may be linked to aliases, variations to canonical names, and/or legal entities. Unique identifiers may be pulled from a graph data structure and stored in the business register so as to define a business register of unique identifiers representing canonical brand names, so as to allow improved query matching and searching within the business register. In some embodiments, associated entity data objects are stored in a graph data structure in communication with a business register.

[99] **Figure 4** is a schematic illustration of a business register representing a set of unique identifiers generated from respective entity data objects, according to some embodiments. Entity data objects 400, 402 and 404 each have different identification types, but are grouped together in a multi-layer data object 401 as they have one or more overlapping entity attributes. In Figure 4, entity data objects 400, 402, and 404 each have an individually generated unique identifier, 412, 414 and 416 respectively. Unique identifiers 412, 414 and 416 are represented in the business register 430, however, the entity data objects are stored external to the business register. For example, they may be stored in a graph data structure. Each of the unique identifiers

412, 414 and 416 relate to the same brand 424. Accordingly, they are all linked to a brand name, or canonical identifier 424. Entity data object 406 has an associated unique identifier 418, which is stored in the business register along with its associated brand name 426. In some embodiments, the associated canonical identifier or brand name may be the first entity attribute of the entity data object 406. Entity data objects 408 and 410 are grouped together in a multi-layer data object 407. In this example, they may share an entity attribute which is not brand name, for example, the entity attribute that they share may be a registration ID. In this case, whilst the registration ID is enough to link the entity data objects in a multi-layer data object 407, they will each be linked to separate brands stored in the business register 430. Each of 408 and 410 have associated unique identifiers 420 and 422 respectively. However, unique identifier 420 is associated with a brand 428, and unique identifier 422 is associated with a separate and distinct brand 429. This situation may be the case, for example, where an entity has identical registration IDs for two different brands, such as “Woolworths” and “Big W”.

[100] The business register may comprise a plurality of unique identifiers, representing an entity or organisation and its associated information, such as a brand name, business number, or other identifier, which may be unique to an entity and to the business register. In some embodiments, the business register may also comprise a plurality of canonical names that may be associated or linked with each unique identifier. Each unique identifier may be associated with one or more sets of associated names or identifiers. Each set of associated names may comprise a canonical name, that is, a representative name for the entity or a brand. In some embodiments, each set of associated names may comprise one or more associated aliases or variants. The associated aliases or variants may be names and/or name formats other than the canonical name by which the organisation may be known or referred to. Each set of names may relate to names which are associated with a unique identifier, for example, a set of associated names may refer to registered names or legal entities that are entity attributes populating an entity data object associated with the unique identifier.

[101] The business register may be configured to store aliases relating to entities. This enables related entity names to be searched to return a single unique identifier and/or a canonical identifier associated with an entity. Once a match is determined upon querying the business register, additional queries and requests can be made to the graph structure based on the one or more identifiers in the business register. Alternatively, if no identifiers or brand name is found or identified, it can be determined that there is no entity data object matching the entity identifier, and therefore that the entity data object does not exist. From there, a request can either be made to write a new entity data object to the graph structure, write a new unique identifier to the business register, or to expand search parameters for additional queries. Further discussion of such an embodiment is described in Australian provisional patent application 2023900913, in the name of Xero Limited, filed on 31 March 2023, the entire contents of which is incorporated herein by reference.

[102] In some embodiments, the business register may be used to respond to requests for entity, business, organisation and/or contact details from a user, such as a user of an accounting platform associated with a particular organisation. For example, such a request may be received from a transaction reconciliation module configured to make reconciliation suggestions to a user to enable the user to reconcile transactions in the accounting platform.

[103] In such an example, the request may comprise one or more target entity attributes, for example, in the form of a candidate brand name extracted from a piece of raw data, such as a data line on a financial statement. In response to receiving a query with a target brand name, the brand name may be used to populate a target entity data object, which is then transformed into a string to generate a target hash. The target hash is then used to query the business register to determine a match for the target hash which represents the one or more candidate attributes in order to take further actions, such as returning a brand name and/or a unique identifier associated with the entity attributes.

[104] **Figure 5A** is an example extract of an entity data object 500 populated with a single entity attribute which is the brand name “Super Market”, according to some embodiments. Entity data object 500 has been assigned ID Type 1, since only the brand field is populated. **Figure 5B** is another example extract of an entity data object 510 populated with multiple entity attributes. In this case, brand name “Super Market”, country information “AU”, registration ID and registration type have also been populated. An associated alias “Company A Group Limited” has been identified in the search index, indicating that this may be a legal entity by which the company may also be referred. Entity data object 510 has been assigned ID Type 3, as it includes populated entity attributes: brand, market or country, and registration information.

[105] **Figure 5C** is an example extract of a multi-layer data object 520 comprised of a plurality of entity data objects 530 populated with entity attributes, according to some embodiments. Each of the entity data objects 530 includes different subsets of populated entity attributes, and varying identification types. Multi-layer entity data object 520 includes entity data objects which represent two distinct brand names “Super Market” and “Petrol Stop” to varying degrees of granularity. Each entity data object shares at least one attribute with at least one other entity data object in the multi-layer data object 520.

[106] **Figure 6** is a schematic illustration of a communications system 600, according to some embodiments. The communications system 600 comprises a system 602 generating entity data objects and/or unique entity identifiers, such as an accounting system (or a system providing an accounting platform) in communication with one or more computing devices 608 across a communications network 606. Examples of a suitable communications network 606 may include a cloud server network, wired or wireless internet connection, BluetoothTM or other near field radio communication, and/or physical media such as USB.

[107] The system 603 comprises one or more processor(s) 614, and memory 616. The processor(s) 614 may include integrated electronic circuits that perform

calculations, and may include a microprocessor, for example. The processor(s) 614 may comprise one or more microprocessors, central processing units (CPUs), application specific instruction set processors (ASIPs), application specific integrated circuits (ASICs) or other processors capable of reading and executing instruction code.

[108] Memory 616 may comprise one or more volatile or non-volatile memory types. For example, memory 616 may comprise one or more of random access memory (RAM), read-only memory (ROM), electrically erasable programmable read-only memory (EEPROM) or flash memory. Memory 616 comprises program code (for example, configured to store executable code modules or engines), accessible by the processor(s). When executed by the processor(s) 614, the program code provides the various computational capabilities and functionality of the system 602, causing the system to perform certain functionalities, which are described herein. In some embodiments, memory 616 stores instructions (such as program code) which when executed by the processor(s) 614 causes the system 602 to manage accounting aspects for a business or entity, provide accounting functionality to the one or more computing devices 608 and/or to function according to the described methods.

[109] In some embodiments, the system 602 may be implemented as a distributed system comprising multiple server systems configured to communicate over a network to provide the functionality of the system 602. For example, one or more of the program code (for example modules or engines) may be deployed on one or more disparate or remote servers, which may cooperate to provide the functionality of the system 602 described.

[110] The system 602 further comprises a network interface 618 to facilitate communications with components of the system 600 across the communications network 606, such as computing device(s) 608, and data stores 620, 621 and/or other servers, such as a financial institute or banking server 612. The network interface 618 may comprise a combination of network interface hardware and network interface

software suitable for establishing, maintaining and facilitating communication over a relevant communication channel.

[111] The system 600 further comprises data stores 620 and 621, which may form part of or be local to the system 602, or may be remote from and accessible to the system 602. The data stores 620 or 621 may be relational or non-relational databases. In some embodiments, the data store 620 may be a graph data structure, and the data store 621 may be a business register. In some embodiments, the data store 621 may be incorporated within, or in communication with, data store 620.

[112] The data store 620 and/or data store 621 may be configured to store business records, banking records, accounting documents and/or accounting records associated with entities having user accounts with the system 602, availing of the services and functionality of the system 602, or otherwise associated with the system 602. The data store 620 and/or data store 621 may be configured to store entity attributes, entity data objects and/or unique identifiers about a plurality of entities or organisations having accounts with or otherwise managed by or manageable using the system 602 or accounting platform provided by the system 602.

[113] The system 602 may also be arranged to communicate with financial institute server(s) 612 or other third party financial systems (not shown) to receive financial records and/or financial documents associated with transactions for reconciliation. For example, in some embodiments, the system 602 may be arranged to receive bank feeds having line items (or financial data items) associated with transactions to be reconciled by the system 602. The financial institute server(s) 612 may provide an Application Programming Interface (API) to securely extract transaction information in relation to one or more bank accounts of a business. The APIs may be secured using authentication and encryption mechanisms and the extracted transaction data may be referred to as bank feed data. Bank feed data may comprise information regarding one or more transactions, including transaction data, transaction amount, transaction reference, for example.

[114] The data store 621 may comprise a business register, which may include entity or organisation identifiers, such as an organisation name, ABN, or other identifiers, and associated contact names, contact name aliases and/or contact name variants. The business register may include unique entity identifiers which represent the entity data objects, or are linked to the entity data objects.

[115] The system may further include a builder module 622. The builder module 622 may be configured to determine the vertices and/or edges of a graph structure, such as data store 620 or 621, or may be configured to create objects based on information extracted from raw data. The builder module may include an entity data object creation module 623, configured to create entity data objects which represent brands or entities based on determined entity attributes extracted from raw data. The builder module may also include a unique identifier module 624 configured to generate unique identifiers from the entity data objects created by the entity data creation module 623. The builder module 622 is configured to create or update objects in a data store or a business register through a business register module 626 and/or a graph interface module 628. The builder module may be configured to perform method 300 described herein.

[116] The unique identifier module 624 is configured to generate a unique identifier associated with an entity data object. For example, the unique identifier module may be configured to perform method 300 described herein. In some embodiments, unique identifier module and the entity data creation module 623 may be configured to perform the method 300 in conjunction with each other. In some embodiments, the unique identifier module 624 may be configured to receive entity data objects from an external data source and generate unique identifiers for each entity data object. The unique identifier module 624 may be in communication with the graph interface module 628 and/or the business register module 626, for example, to transmit unique identifiers for read requests or write unique identifiers to the business register. In some embodiments, the unique identifier module 624 may be used to query the business register through the business register module 626 with target unique identifiers or target hashes.

[117] It will be appreciated by persons skilled in the art that numerous variations and/or modifications may be made to the above-described embodiments, without departing from the broad general scope of the present disclosure. The present embodiments are, therefore, to be considered in all respects as illustrative and not restrictive.

CLAIMS:

1. A method, including:

determining a set of one or more entity attributes to populate an entity data object associated with an entity;

populating one or more of a plurality of fields in the entity data object with the one or more entity attributes, each entity attribute populating an associated field of the plurality of fields;

transforming the entity data object into a string; and

generating a hash from the string to define a unique identifier which represents the entity data object.

2. The method according to claim 1, wherein transforming the entity data object into a string includes:

determining the one or more entity attributes from the entity data object;

transforming each entity attribute into an entity attribute string; and

concatenating each entity attribute string to form a plaintext string.

3. The method according to claim 2, wherein transforming each entity attribute into an entity attribute string includes prepending or appending a delimiter to each entity attribute string.

4. The method according to any one of claims 1 to 3, wherein generating a hash includes:

applying a hash function to the plaintext string to generate a resultant hash.

5. The method according to claim 4, wherein applying the hash function to the plaintext string includes adding a pepper to the plaintext string.

6. The method according to claim 4 or claim 5, wherein applying the hash function to the plaintext string includes adding a salt.
7. The method according to any one of claims 4 to 6 wherein generating a hash includes prepending a marker to the resultant hash to define the unique identifier.
8. The method according to any one of claims 4 to 7, wherein the hash function is an MD5 hash function.
9. The method according to any one of claims 1 to 8, wherein the unique identifier includes a marker.
10. The method according to claim 9, wherein the marker is positioned at the beginning of the unique identifier.
11. The method according to any one of claims 1 to 10, wherein the method includes:
 - determining an identification type of the entity data object on the basis of the subset of the plurality of fields which are populated; and
 - associating the entity data object with the determined identification type.
12. The method according to any one of claims 1 to 11, wherein the method includes writing the unique identifier to a data structure.
13. The method according to any one of claims 1 to 12, wherein the method includes associating a first attribute of the one or more entity attributes with the unique identifier.
14. The method according to claim 13, wherein the first attribute is a brand or business name.

15. The method according to claim 13 or claim 14, wherein the first attribute and the unique identifier are stored within a first object in a database structure.
16. The method according to any one of claims 1 to 15, wherein the one or more entity attributes include at least one of a canonical identifier, user recognisable name, brand name, business name, registration type, business number, registration identifier, location information, country, state, suburb, contact information, address, registered name, and/or associated aliases.
17. The method according to any one of claims 1 to 16, wherein the plurality of fields includes at least one of a brand, registered name, registration type, registration ID and location information.
18. The method according to any one of claims 1 to 17, wherein the method includes determining a set of one or more entity attributes to populate a set of entity data objects associated with an entity.
19. The method according to any one of claims 1 to 18, wherein the method includes associating the entity data object with at least one other entity data object, the entity data object and the other entity data object sharing at least one entity attribute in a common field, to create a multi-layer data object.
20. The method according to claim 19, wherein the at least one entity attribute in the common field is a brand.
21. The method according to claim 19, wherein the at least one entity attribute in the common field is a business number.
22. The method according to any one of claims 1 to 21, wherein the entity data object is stored within a data structure.

23. The method according to any one of claims 1 to 22, wherein transforming the entity data object into a string involves converting populated data in each of the plurality of fields into a string.

24. A method of generating a unique identifier from an entity data object, including:

determining one or more entity attributes from the entity data object;

transforming each entity attribute into an entity attribute string;

attaching a delimiter to each entity attribute string;

concatenating each entity attribute string to form a plaintext string;

applying a hash function to the plaintext string to generate a resultant hash, wherein applying a hash function includes adding a pepper to the plaintext string; and

prepending a marker to the resultant hash to define the unique identifier which represents the entity data object.

25. A system, including:

one or more processors; and

memory comprising computer executable instructions, which when executed by the one or more processors, cause the system to perform the method of any one of claims 1 to 24.

26. A computer-readable storage medium storing instructions that, when executed by a computer, cause the computer to perform the method of any one of claims 1 to 24.