Database Design

January 7, 2019

1 Relational Design

1.1 Relational schema design

This section will discuss some of the noteworthy decisions taken in the design of the relational schema for the given problem domain. The full SQL for the relational schema can be seen in *appendix 1.1*. Note: PostgreSQL 9.6 is used for all DML and DDL defined in this document. The online SQL compiler *https://rextester.com* was used to validate the schema and its constraints.

1.1.1 Types

The schema defines two custom enum types, namely 'ISO3166_ALPHA2' (based on the ISO3166 standard), and 'ACQUISITION_STATUS'. It is worth noting that 'ISO3166_ALPHA2' is loaded with a subset of the full standard as this is fit for the purpose of our data model.

The ACQUISITION_STATUS status enum is implement by the *status* column in the acquisition table, and the ISO3166_ALPHA2 enum is implemented

directly by the *country* table as well as all tables which contain a foreign key to this (*company*, *founder*).

The use of Enums is advantageous as it provides a means of implicit santisation to the tables that implement these types. For example it means an *acquisition* instance could never be created with an undesirable status (i.e a status outside the domain of strings defined by the enum). The same is true of the ISO3166_ALPHA2 implementation, through it we can be assured the quality of the country code in the database is accurate. Considering Date's observation [2] that a table can be thought of as a predicate for which each row is a true proposition, custom data types enable us to constrain the table to only the domain of the predicate. For example, the ISO3166_ALPHA2 implemented in this model it would be impossible to enter a country with the values 'Norway', 'NO' as the 'NO' country code does not exist in the custom defined type.

It should be noted that the tuple ('Norway', 'CN') would be a valid entry into the country table (as 'CN' does exist in the custom type). We can verify that this record is false as per the ISO3166 standard as CN is the country code for China, however there is nothing wrong with it from the data storage level as defined by this schema. Therefore although implementing the custom type has helped, it has not guarantee the quality of the data. To frame ths in the context of propositional logic as prescribed by Date; it is possible for our schema to record false propositions. Dropping the country table altogether and extending the SO3166_ALPHA2 type to include a country name and code tuple would therefore be a further improvement that could be made to this model if we wanted to introduce more rigour and reduce the error potential for such inaccuracies.

1.1.2 Schema Design

For the given problem the following schema has been defined. Sample DML statments to test the constraints and queries (including failure cases) is included in *Appendix A*).

```
--PostgreSQL 9.6
```

```
CREATE TYPE ISO3166_ALPHA2 AS ENUM ( 'DE', 'FR', 'GB', 'US',
                                     'JP', 'CN', 'SA');


CREATE TYPE AQUISITION_STATUS AS ENUM ( 'announced' 'completed',
                                       'failed');

DROP TABLE IF EXISTS country CASCADE;
CREATE TABLE country ( name varchar (255) NOT NULL,
code ISO3166_ALPHA2 NOT NULL, PRIMARY KEY (code));


DROP TABLE IF EXISTS company;
CREATE TABLE company ( id serial, name varchar (255) NOT NULL,
founded date NOT NULL, country_code ISO3166_ALPHA2 NOT NULL,
parent_company_id int DEFAULT NULL,
-- need to make this an int not a serial to allow nulls
 -- Constraints
 PRIMARY KEY (id), CONSTRAINT company_name_country UNIQUE (name,
                                                    country_code),
                      FOREIGN KEY (country_code)
                      REFERENCES country (code),
                      FOREIGN KEY (parent_company_id)
                      REFERENCES company (id));


CREATE TABLE founder ( id serial, firstname varchar (255) NOT NULL,
                       lastname varchar (255) NOT NULL,
                       dob date NOT NULL,
                       country_of_origin ISO3166_ALPHA2 NOT NULL,
                       PRIMARY KEY (id),
                       FOREIGN KEY (country_of_origin)
                       REFERENCES country (code));


CREATE TABLE founder_companies( founder_id serial, company_id serial,
                                FOREIGN KEY (founder_id)
```

```
                                REFERENCES founder (id),
                                FOREIGN KEY (company_id)
                                REFERENCES company (id));


CREATE TABLE acquisition ( parent_company_id int NOT NULL,
                           child_company_id int NOT NULL,
                           status AQUISITION_STATUS,
                           price_usd NUMERIC DEFAULT NULL,
                           announced_date DATE NOT NULL,
                           completion_date DATE DEFAULT NULL,
                           FOREIGN KEY (parent_company_id)
                           REFERENCES company (id),
                           FOREIGN KEY (child_company_id)
                           REFERENCES company (id));
```

1.2 Normalisation

A relational schema can be said to be compliant to BCNF if for all of its functional dependencies (x $\Rightarrow$ y) if one of the following statements is true:

a) y is a subset of x (trivial functional dependency)

b) x is a superkey (a subset of a candidate key for the relation). [5]

Where a candidate key is a valiable key for identifying a row. Given this definition and with the non-trivial functional dependencies highlighted below, we can judge that this schema is compliant to BCNF - this is discussed in depth in the following sections.

1.2.1 Company

id ⇒{name, founded, country_code, parent_company_id}
{name, country_code} ⇒ {founded, parent_company_id, id}

N.B. At first glance a set of functional dependencies with {name, founded} on the left side seems to exist, but this does not hold due to the composite unique constraint on name and country code (see *section 1.2*). That is to say that multiple companies with the same name can exist in the relation, provided they are in different countries. Hence functional dependencies with the {name, countrycode} tuple on the left side always hold, but those with {name, founded} will only hold in some data sets.

The {name, founded} tuple is candidate key. Had this been implemented as a composite primary key there would have been no requirement for the additional 'id' attribute, however as this key is used in multiple tables as a foreign key (including the self referential foreign key'parent_company_id' in this table), it is more efficient to use a serial id field than a composite of two string columns. x is a candidate key and therefore a superkey, hence this table does conform to BCNF.

1.2.2 Founder

id ⇒ {firstname, lastname, dob, country_of_origin}

The only functional dependency present is the base dependency that all columns are functionally dependent on the 'id' primary key. Whilst it initially appears that other dependencies may exist such as

{firstname, lastname} ⇒ {dob, country_of_origin}

these dependencies are only temporal - i.e they hold for some states of the relation but not all [1]. As there are no unique constraints on any of these

attributes, it is possible (if unlikely) to have founders with the same name, date of birth and country of origin in the relation (just as it is in the real world). x is a primary key and therefore a superkey, hence this FD is BNCF compliant.

1.2.3 Acquisition

{parent_company_id, child_company_id announced_date} ⇒{status, completion_date}

This functional dependencies allows for historised records in the relation. Assuming that a parent company does not announce an attempted acquisition with the same child company more than once on the same day, if we know the attributes on the left side of the dependency we can always deduce the status and completion date of the acquisition. This enables historization, as a company could make multiple failed attempts to acquire another before they finally are successful. As with the company table, this combination could be used as a composite primary key and therefore we can evaluate this table is compliant to BCNF.

1.2.4 Founder_companies

There are no functional dependencies as this is a many to many join table, i.e. a company can have many founders, and a founder can found many companies.

1.3 Constraints

The following constraints are expressed using Relational Calculus (RC) and then again in SQL.

a) RC:

$$\{id \mid \exists\, name, founded, country\_code, \\ parent\_company\_id, (company(id, name, \\ founded, country\_code, parent\_company\_id) \wedge id = parent\_company\_id)\}$$

SQL:

```
ALTER TABLE company ADD CONSTRAINT not_own_parent
CHECK (id != parent_company_id);
```

b) RC:

$$\{(parent\_company\_id, child\_company\_id) \mid \exists\, status, price\_usd, \\ announced\_date, completion\_date, (acquisition(parent\_company\_id, \\ child\_company\_id, status, price\_usd, announced\_date, completion\_date) \\ \wedge\ completion\_date < announced\_date)\}$$

SQL:

```
ALTER TABLE acquisition ADD CONSTRAINT check_completion_date
CHECK (completion_date >= announced_date);
```

c)

$$\{(parent\_company\_id, child\_company\_id) \mid \exists\, status, price\_usd, announced\_date, \\ completion\_date, (acquisition(parent\_company\_id, child\_company\_id, status, \\ price\_usd, announced\_date, completion\_date) \\ \wedge\, status \neg\, `failed' \\ \vee\ (status = `failed' \wedge completion\_date = completion\_date)\}$$

n.b. 'completion_date = completion_date' is a means for asserting that the completion_date value is not NULL.
SQL:

```
ALTER TABLE acquisition ADD CONSTRAINT CONSTRAINT
check_no_failed_completion_dates CHECK (
        status <> 'failed'
        or status = 'failed' AND (completion_date IS NULL))
);
```

d)

$$RC : \{id \mid \exists\, name, founded, country\_code,$$
$$parent\_company\_id, parent\_company\_id, child\_company\_id,$$
$$status, price\_usd, announced\_date, completion\_date,$$
$$(company(id, name, founded, country\_code,$$
$$parent\_company\_id, parent\_company\_id)$$
$$\wedge\ acquisition(parent\_company\_id, child\_company\_id,$$
$$status, price\_usd, announced\_date, completion\_date)$$
$$\wedge\ announced\_date < founded\}$$

SQL: This check cannot be enforced using SQL as a pure constraint, as it operates on values from multiple tables in the schema. It is possible to make use of a trigger which is called on row insert or update to check for the constraint and raise an exception should a write request attempt be found to be in violation. A potential implementation is:

```
CREATE FUNCTION acquisition_date_constraint() RETURNS trigger
AS $acquisition_date_constraint$
 BEGIN
     IF NEW.announced_date < (SELECT founded FROM company
     WHERE id = NEW.child_company_id) THEN
         RAISE EXCEPTION
         '% Acquisition cannot be announced before
         company founded!', NEW.announced_date;
     END IF;

     RETURN NEW;
 END;
$acquisition_date_constraint$ LANGUAGE plpgsql;
```

```
-- Run the trigger whenever a row is
-- inserted or updated to acquisition
CREATE TRIGGER acquisition_date_constraint
BEFORE INSERT OR UPDATE ON acquisition
    FOR EACH ROW EXECUTE PROCEDURE acquisition_date_constraint();
```

Here the custom function *acquisition_date_constraint* is defined to return a trigger. The trigger uses SQL a SELECT statement and the NEW keyword to compare the ingress announced date and comparing it to the founded date of the company. An exception is raised if the constraint is violated. The function is set to be called on any insert or update to the acquisition table. [4]

2 Relations - Relational Calculus & Algebra

All of the queries in this section are expressed using Relational Algebra (RA), except for *query a* is expressed using Relational Calculus (RC).

a)

$$\{id \mid \exists\ name, founded, country\_code, parent\_company\_id, \\ (company(id, name, founded, country\_code, parent\_company\_id) \\ \wedge\ founded\ >\ `1999-12-31')\}$$

b)

$$\pi\ id(\sigma\ country\_code\ =\ `US'\ \vee country\_code\ =\ `GB'(company))$$

c)

$$\pi\ parent\_company\_id(\sigma status\ =\ `completed'(acquisition))$$

n.b. The SQL equivalent to this query (see *section 3*) uses the DISTINCT operator to remove duplicates; as RA is based on set theory there are no duplicates.

d)

$$\pi\ founder\_id(\sigma founder\_id \geq 3(founder\_id\ g\ count(founder\_id)(founder\_companies))$$

n.b. Here the aggregate operator g is used to count over the occurrences of each founder_id in the relational. This aggregate can then be selected over.

e)

$$\pi\ child\_company\_id(f \bowtie c\ with(f := (\pi\ child\_company\_id, announced\_date(\sigma\ status = `failed'(acquisition))) \\ c := (\pi\ child\_company\_id, completion\_date\ (\sigma\ status = `completed'(acquisition)))) \\ \sigma completion\_date > announced\_date)$$

In this query we select all companies with a failed acquisition and completed acquisitions and name the resulting relations f & c respectively using the *with* operator. In both relations the *child_company_id* is kept. Within f the *announced_date* attribute is kept, and within c the *completion_date* attribute is kept. The rest of the fields are projected away. The relations are then joined using a natural join which utilises the fact that both relations have a *child_company_id* attribute.We can then compare the *completion_date* of c to the *announced_date* of f to select only those in which the completed acquisition occurred after the failed attempt.

It is worth noting that this query assumes that the acquisitions are announced and resolved (i.e. fail or complete) in contiguous manner in respect of their announced dates. The schema does not record the *failed_date* for failed acquisitions. Such an addition would make this query more accurate as the completion and failed dates could be directly compared (as oppossed to comparing the announced_date and completion_date). This would significantly affect the database design as the additional attribute were to be added to the acquisitions relation then new functional dependencies would be introduced which would cause third normal form (3NF) violations. For example, {announced_date,

status, parent_company_id, child_company_id } ⇒ {completion_date, failed_date} or simply {completion_date ⇔ failed_date}. Both of these examples are 3NF violations and as both introduce functional dependencies between non-key attributes.

To strictly keep the schema in 3NF/BCNF a potential refactor could be to have an aqcuisition_dates table with just an announced_date and end_date. The acquisitions table could then have a FK to this table as well as the status as is currently indicated. Implementing this design was outside of the scope of the project requirements, but it is interesting to note that even adding one field in the aim of better analysis can drastically affect the normalisation of the database.

f) This query is not expressible in RA or RC due to the lack of a native *limit* function.

g) This query is not expressible in RA or RC due to the lack of recursion in the query languages.

3 SQL

a)
```
SELECT id
FROM company
WHERE founded > '1999-12-31';
```

b)
```
SELECT id
FROM company
WHERE country_code IN ('UK', 'US');
```

c)
```
SELECT distinct(parent_company_id)
FROM acquisition
WHERE status = 'completed';
```

d)
```
 SELECT founder_id,
       COUNT (founder_id)
FROM founder_companies
GROUP BY founder_id HAVING COUNT(founder_id) >= 3;
```

e)
```
SELECT f.child_company_id
FROM
  (SELECT *
   FROM acquisition
   WHERE status = 'failed')f
JOIN
  (SELECT *
   FROM acquisition
   WHERE status = 'completed')c
   ON c.child_company_id = f.child_company_id
WHERE c.completion_date > f.failed_date;
```

f)
```
SELECT founder_id,
       count(founder_id)
FROM founder_companies
GROUP BY founder_id
ORDER BY count(founder_id) DESC LIMIT 10 ;
```

g)
```
WITH RECURSIVE comp AS
  (SELECT parent_company_id,
          id,
          name
   FROM company
   WHERE name = 'Tesla'
   UNION SELECT c.parent_company_id,
                c.id,
                c.name
   FROM company c
   INNER JOIN comp p
   ON p.id = c.parent_company_id)
SELECT comp.name
FROM comp;
```

Of note here is query *(g)* which implements a recursive query. Here the *WITH* operator is used to signify a Common Table Expression (CTE) - affectively a temporary table to be used for the duration of the query. The *RECURSIVE* operator is used to signify that the CTE is to be used recursively, and the

UNION operator creates the recursive call by joining a SELECT statement to the CTE [3].

4 Data Mining

The focus of this data mining activity is to predict the following two questions: Q1) Whether an acquisition is likely to succeed or fail Q2) If an acquisition is to succeed, how long will the process take?

4.1 Data Understanding

The schema has almost certainly evolved since it was first designed. We must understand structural changes to the schema and data within the database, especially in the key tables, 'company', 'founder' and 'acquisition'. Cardinality, sparseness and volume can aid us in understanding the data.

The original schema allowed NULL attributes which could impact on modelling. The attribute 'acquisition.status' is used for both questions (either as a target attribute (Q1), or as an attribute to subset on (Q2)). For Q2 the target attribute is the delta $d :=$ ('completion_date' -' announced_date'). We need to assess the quality and sparseness of these attributes. In the original schema 'announced_date' had a NOT NULL constraint; if this constraint has since been lifted both elements of our composite d could be sparse.

We can now judge whether our data is sufficient to model on, or if enrichment with external data is required. Information omitted from the original schema includes industry sectors and financials. The schema may have evolved to include this information, however if not, we should consider finding relevant external data sources. This presents new complexity, including data ingestion and mapping companies to industry sectors whilst ensuring veracity.

4.2 Data Preparation

The subset of completed acquisitions with NULL completed_dates may require attention. If the subset is small (0-5% of the superset) it may be acceptable to omit these records from the model, otherwise we may consider filling null values. Statistical methods i.e. using the mean delta /emphd, and adding this value to the announced_date of each record could be considered. As this data forms a target attribute, filling the missing values from research may be more appropriate, though time and resource consuming.

4.3 Data Modelling

Both cases use supervised learning methods. As Q1 is a binary classification problem a decision tree (DT) is a suitable model. Further data preparation may be required to aid modelling. For example, we could define a 'hot' industry sector as one with high growth, or one which has seen voluminous acquisitions. Further decision points are formulated on company age, size and revenue.

Q2 models a continuous numerical variable; making a regression model appropriate. Features modelled in Q1 can be reused, but from the subset of data containing only completed acquisitions.

4.4 Evaluation

Binary classification problems use ROC curves to validate error rate. The models can be fine tuned using information gain and error reduction rate respectively.

We can also evaluate whether the models selected were appropriate. Could the DT be improved with a random forest algorithm? Perhaps the probability of an acquisition completing is of now interest (i.e. no longer a binary classification), in this instance we would restart the cycle, addressing structural

questions of the data (e.g. is it acyclic) to define what probabilistic models can be applied.

4.5 Deployment

Whether the business are expecting a report or a deployable microservice will greatly affect the delivery. If the latter, does the organisation understand the maintenance burden as the schema evolves? If the former, then what is the update frequency of the research? Establishing these factors will inform the deployment strategy.

References

[1] Date, C.J *An Introduction to Database Systems*. (2003). An Introduction to Database Systems. 8th ed. London: Pearson. 334-336

[2] Date, C.J *An Introduction to Database Systems*. (2003). An Introduction to Database Systems. 8th ed. London: Pearson. 783-789

[3] PostgreSQL documentation. (2016). *Triggers on Data Changes* Available: https://www.postgresql.org/docs/9.6/plpgsql-trigger.htmlPLPGSQL-TRIGGER-EXAMPLE. [Accessed 6th Jan 2019.]

[4] PostgreSQL. (2016). *WITH Queries (Common Table Expressions).* Available: https://www.postgresql.org/docs/9.6/queries-with.html. [Accessed 6th Jan 2019.]

[5] Agnieszka Kozubek. (2014). *The Boyce-Codd Normal Form (BCNF).* Available: https://www.vertabelo.com/blog/technical-articles/boyce-codd-normal-form-bcnf. [Accessed 6th Jan 2019.]

A
seed.sql

```
-- Dummy data for tests

-- Countries
INSERT INTO country(name, code)
VALUES (‘Germany’,‘DE’),
        (‘France’, ‘FR’),
        (‘Great Britan’,‘GB’),
        (‘United States of America’, ‘US’),
        (‘Japan’, ’‘P’),
        (‘China’,‘CN’),
        (‘South Africa’,‘SA’);

 -- Fails - breaks PK constraint
INSERT INTO country(name, code)
VALUES (‘Jupiter’,
        ‘CN’);

 -- Founders
INSERT INTO founder(firstname, lastname, dob, country_of_origin)
VALUES (‘Elon’,
        ‘Musk’,
        ‘1971-06-28’,
        ‘SA’),
        (‘Bobby’,
        ‘George’,
        ‘1971-06-28’,
        ‘GB’);

 -- Companies
INSERT INTO company(name, founded, country_code, parent_company_id)
VALUES (‘Tesla’,
        ‘2003-01-01’,
        ‘US’,
```

```
        NULL),
        (‘Perbix’,
         ‘1976-01-01’,
          ‘US’,
          1),
        (‘SolarCity’,
         ‘2006-06-04’,
          ‘US’,
           1),
           (‘Grohmann Engineering’,
           ‘1963-01-01’,
            ’DE’,
            1),
            (‘Riviera Tool LLC’,
             ‘2007-01-01’,
              ‘US’,
               1)

 -- Fails - not_own_parent constraint
INSERT INTO company(name, founded, country_code, parent_company_id)
VALUES (‘Foo’,
        ‘2018-12-15’,
        ‘US’,
        6);

 -- Acquisitions
INSERT INTO acquisition (parent_company_id, child_company_id, status,
                         announced_date, completion_date)
VALUES(1, 2,
       ‘completed’,
       ‘2017-11-06’,
       ‘2017-11-06’),
      (1,3,
      ‘completed’,
      ‘2016-06-22’,
      ‘2017-11-06’),
      (1, 4,
     ‘completed’,
```

```
     '2016-11-08',
     '2017-11-06'),
      (1, 5,
     'completed',
     '2015-06-08',
     '2017-11-06');

 -- Test data for query e
INSERT INTO acquisition (parent_company_id, child_company_id, status,
                         announced_date, failed_date)
VALUES(1, 5,
       'failed',
       '2015-06-07',
       '2015-06-07');

 -- Fails - no_self_acquisitions
INSERT INTO acquisition (parent_company_id, child_company_id, status,
                           announced_date)
VALUES(1,1,
       'failed',
       '2015-06-08');

 -- Fails - check_no_failed_completion_dates
-- >> new row for relation "acquisition" violates check constraint
"check_no_failed_completion_dates"
INSERT INTO acquisition (parent_company_id, child_company_id,
                         status, announced_date, completion_date)
VALUES(1,6,
       'failed',
       '2015-06-08',
       '2015-06-09');

 --> 23514: new row for relation "acquisition" violates
    check constraint "check_no_failed_completion_dates"
 -- Fails - acquisition_date_constraint
INSERT INTO acquisition (parent_company_id, child_company_id,
                         announced_date)
VALUES(2,5,
```

```
       '2006-12-31');
-- << P0001: 2006-12-31 Acquistion cannot be
      announced before company founded!

-- Test Data for f
INSERT INTO founder_companies(founder_id, company_id)
VALUES (2, 2);

 -- Test Data for g - test_true should be returned, but not test_false
INSERT INTO company(name, founded, country_code, parent_company_id)
VALUES ('test_false',
        '2003-01-01',
        'US',
        NULL),
        ('test_true',
        '2003-01-01',
        'US',
        2),
        ('test_true_child',
        '2003-01-01',
        'US',
        7);
```