# The data model

For my purposes I am happy with daily data. The data I want to store has an obvious grain of date-security ie one row for each day for each security that I have data for. This implies just two dimensions – date and security (or product, as I call it below, following ASIC’s terminology for their published data on short positions). The facts that I’m interested in are the open, high, low, close and adjusted price for each day; the volume of transactions; the short position; the total float; and the short position as a proportion of the total float. The price and volume data can come from Yahoo Finance, and the short positions data can be downloaded from the Australian Securities and Investment Commission, who collect self-reports from short sellers.

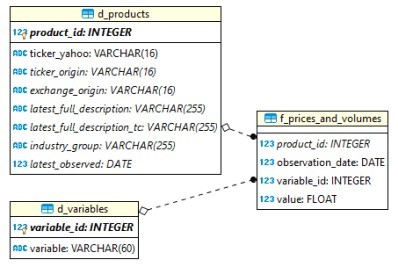
My initial plan was just two tables:

one for the product dimension with unchanging (or slowly changing) information like its ASX ticker, the ticker on Yahoo Finance, its latest name with various alternative approaches to punctuation

one for the facts, with separate columns for the nine facts (open price, volume, etc) listed in the previous paragraph.

I think this is indeed probably the correct structure for a Kimball-style dimensionally modelled analytical datamart. However, it proved unpleasant to write the extract-transform-load for my two data sources into that wide fact table. It would have involved a lot of UPDATE operations to add data from one source to columns for rows that are partly populated by the other). SQLite in particular does not really support UPDATE in combination with a JOIN and getting around this would have been awkward. So to simplify things I normalised the data one step further and made my fact table more “long and thin”. This meant adding a variable dimension so that when extra data comes in from another source I am just adding new rows to the data for a subset of variables rather than filling in empty columns for existing rows.

Here’s how the data model looks:



I used the excellent (and free) DBeaver universal database tool to make that diagram, and to develop any non-trivial SQL code used in this blog.

And this is the SQL that creates it. I’ve used GO (in the style of Microsoft T-SQL) to separate

each command, not because SQLite understands it (it doesn’t) but because I have R functions that do, which I’ll be using in a minute.

*Post continues below SQL code*

-- Drop existing version of database (careful!): DROP TABLE IF EXISTS f\_prices\_and\_volumes;

GO

DROP TABLE IF EXISTS d\_variables; GO

DROP TABLE IF EXISTS d\_products; GO

-- Define the various tables:

CREATE TABLE d\_products ( product\_id INTEGER PRIMARY KEY,

ticker\_yahoo VARCHAR(16) NOT NULL UNIQUE, ticker\_origin VARCHAR(16), exchange\_origin VARCHAR(16), latest\_full\_description VARCHAR(255), latest\_full\_description\_tc VARCHAR(255), industry\_group VARCHAR(255), latest\_observed DATE

) GO

CREATE TABLE d\_variables ( variable\_id INTEGER PRIMARY KEY,

variable VARCHAR(60) NOT NULL UNIQUE

) GO

CREATE TABLE f\_prices\_and\_volumes ( product\_id INTEGER, observation\_date DATE NOT NULL, variable\_id INTEGER NOT NULL, value FLOAT NOT NULL,

FOREIGN KEY (product\_id) REFERENCES d\_products (product\_id),

FOREIGN KEY (variable\_id) REFERENCES d\_variables (variable\_id)

) GO

CREATE UNIQUE INDEX one\_obs\_per\_var

ON f\_prices\_and\_volumes(variable\_id, product\_id, observation\_date);

GO

-- Populate the variables dimension:

INSERT INTO d\_variables (variable) VALUES ('open'),

('high'),

('low'),

('close'),

('volume'),

('adjusted'), ('short\_positions'), ('total\_product\_in\_issue'), ('short\_positions\_prop')

# Populating with data

I used R for accessing the origin data from the web and sending SQL commands to set up the database. Here’s the first chunk of code that creates the empty database, runs the SQL above to set up tables, and makes an initial dump of ASX listed companies into the d\_products table. I’m not sure how definitive is that list of ASX listed companies referred to in the below code.

*Post continues below R code*

library(tidyverse) library(glue) library(lubridate)

library(frs) # for download\_if\_fresh and execute\_sql.

Available from GitHub library(janitor) library(ggrepel) library(kableExtra) library(quantmod) library(RSQLite) library(DBI) library(tools)

#=============database setup============================

#- Define database- - if(!"stocks.sqlite" %in% list.files()){

# if this is the very first run we need to create the empty database and its tables

con <- dbConnect(RSQLite::SQLite(), "stocks.sqlite")

# run the SQL script that defines the table - needs to be saved in seaprate file:

execute\_sql(con, "0199-stocks-db-setup.sql", log\_table = NULL)

} else {

con <- dbConnect(RSQLite::SQLite(), "stocks.sqlite")

}

#------------set up product dimension-----------

asx\_cos <- read.csv("<http://www.asx.com.au/asx/research/> ASXListedCompanies.csv",skip=1) %>%

mutate(ticker\_yahoo = paste0(ASX.code, ".AX"), exchange\_origin = "ASX", latest\_observed = as.Date(NA), latest\_full\_description\_tc =

tools::toTitleCase(tolower(Company.name))) %>% select(

ticker\_yahoo, ticker\_origin = ASX.code, exchange\_origin,

latest\_full\_description = Company.name, latest\_full\_description\_tc, industry\_group = GICS.industry.group, latest\_observed

)

RSQLite::dbWriteTable(con, "d\_products", asx\_cos, row.names

=FALSE, append = TRUE)

# Loading the short positions data

The short positions data from the ASIC website includes many products that aren’t in the list of listed companies on the ASX site. In general, I want to be able to update my list of products/securities. Getting data from Yahoo Finance, where I specify a security ticker code and then get the data, won’t let me do this (unless I tried codes at random). Because of all this, in my initial bulk upload I do the ASIC data first, hoping (without really checking how it happens, which of course I would for a more formal use) that this will surface new (or old) securities that aren’t in the spreadsheet I downloaded from the ASX.

The code below is in two chunks. It downloads all the CSVs of short positions data from ASIC, taking care not to re-download data it already has. Each CSV represents one day of three facts on each product. Then (somewhat more complex), it reads all the CSVs one at a time (if it hasn’t already processed this particular CSV); identifies missing products/securities which it then adds to the d\_products table; then populates the fact table with the facts for all the products it’s found in this particular CSV, having matched them to the product\_id field in the d\_products table.

Other than the three-card shuffle with adding new products to d\_products as it goes, and some annoying complications with different formats and encoding of the CSV files on the ASIC page (see comments in the code), this is fairly straightforward data-wrangling stuff for R.

This took three or four hours each for the two bits of functionality (bulk download and bulk import) to run.

#===============Get the short positions data=============

#- Download- # From:

# https://asic.gov.au/regulatory-resources/markets/short-selling/short-position- reports-table/

all\_dates <- seq(from = as.Date("2010-06-16"), to = Sys.Date(), by = 1)

dir.create("asic-shorts", showWarnings = FALSE)

i = 1

for(i in i:length(all\_dates)){ the\_date <- all\_dates[i]

# Don't bother trying to download on weekends: if(wday(the\_date, label = TRUE) %in% c("Sat", "Sun")){

next()

}

m <- str\_pad(month(the\_date), width = 2, side = "left", pad

= "0")

y <- year(the\_date)

ch <- format(the\_date, "%Y%m%d")

fn <- glue("RR{ch}-001-SSDailyAggShortPos.csv") url <- glue("https://asic.gov.au/Reports/Daily/{y}/{m}/{fn}")

# Only exists for trading days. Rather than bother to work out exactly the trading days,

# we will just skip over any 404 error try(download\_if\_fresh(url, glue("asic-shorts/{fn}")))

}

#- -Import-

all\_products <- dbGetQuery(con, "SELECT product\_id, ticker\_yahoo

'ASX'") %>%

as\_tibble()

FROM d\_products

WHERE exchange\_origin =

already\_done\_dates <- dbGetQuery(con, "SELECT DISTINCT observation\_date AS od

f\_prices\_and\_volumes AS a AS b

b.variable\_id 'short\_positions' observation\_date")$od

FROM

INNER JOIN d\_variables On a.variable\_id =

WHERE b.variable = ORDER BY

d\_variables <- dbGetQuery(con, "select variable\_id, variable

from d\_variables")

all\_csvs <- list.files("asic-shorts", pattern = "DailyAggShortPos", full.names = TRUE)

# we are going to do this backwards so product names are the latest ones

i =length(all\_csvs) for(i in i:1){

the\_csv <- all\_csvs[i]

the\_date <- as.Date(str\_extract(the\_csv, "[0-9]+"), format= "%Y%m%d")

# if we've already got short positions observations in the data for this date,

# then break out of the loop and go to the next iteration if(as.character(the\_date) %in% already\_done\_dates){next()}

# The first 1400 files are actually tab-delimited and UTF-16, the

# rest are genuine comma separated files and more standard encoding.

# Couldn't get read\_csv to work with the various encoding here.

if(i <= 1400){

d1 <- read.csv(the\_csv, fileEncoding = "UTF-16", sep = "\t")

} else {

d1 <- read.csv(the\_csv)

# sometimes this still doesn't work and we go back to the other method:

if(nrow(d1) < 10){

d1 <- read.csv(the\_csv, fileEncoding = "UTF-16", sep =

"\t")

}

}

d2 <- d1 %>%

clean\_names() %>% as\_tibble() %>%

mutate(observation\_date = the\_date,

ticker\_yahoo = paste0(str\_trim(product\_code), ".AX")) %>%

left\_join(all\_products, by = "ticker\_yahoo") %>% select(

product\_id, observation\_date,

short\_positions = reported\_short\_positions, total\_product\_in\_issue,

short\_positions\_prop = x\_of\_total\_product\_in\_issue\_ reported\_as\_short\_positions,

ticker\_yahoo, product, product\_code

) %>%

# convert from percentage to proportion: mutate(short\_positions\_prop = short\_positions\_prop / 100)

non\_match <- sum(is.na(d2$product\_id)) if(non\_match > 0){

message(glue("Found {non\_match} products in the short data not yet in the database"))

print(select(filter(d2, is.na(product\_id)), ticker\_yahoo, product, observation\_date))

new\_products <- d2 %>% filter(is.na(product\_id)) %>% mutate(exchange\_origin = 'ASX',

latest\_full\_description\_tc = tools::toTitleCase(tolower(product)),

industry\_group = NA, latest\_observed = NA,

ticker\_origin = str\_trim(product\_code)) %>% select(ticker\_yahoo,

ticker\_origin, exchange\_origin,

latest\_full\_description = product, latest\_full\_description\_tc, industry\_group,

latest\_observed)

RSQLite::dbWriteTable(con, "d\_products", new\_products, row.names =FALSE, append = TRUE)

all\_products <- dbGetQuery(con, "SELECT product\_id, ticker\_yahoo

'ASX'") %>%

as\_tibble()

FROM d\_products

WHERE exchange\_origin =

}

upload\_data <- d2 %>% select(observation\_date:ticker\_yahoo) %>% left\_join(all\_products, by = "ticker\_yahoo") %>% select(-ticker\_yahoo) %>%

gather(variable, value, -product\_id, -observation\_date)

%>%

left\_join(d\_variables, by = "variable") %>% select(product\_id,

observation\_date, variable\_id,

value) %>%

# a small number of occasions the short\_positions\_prop is NA or Inf because

# the totla product in issue is 0 even though there are some short positions.

# we will just filter these out filter(!is.na(value)) %>%

mutate(observation\_date = as.character(observation\_date))

dbWriteTable(con, "f\_prices\_and\_volumes", upload\_data, append = TRUE)

# progress counter so we know it isn't just stuck if(i %% 100 == 0){cat(i)}

}

# Loading the price and volumes data

Next step is to get some data from Yahoo Finance on price and volumes. Overall, this is more straightforward. The quantmod R package describes the functionality I’m about to use as “essentially a simple wrapper to the underlying Yahoo! finance site’s historical data download”.

I’ve tried in the code below to make this updateable, so in future I can run the same code without downloading all the historical data again. But I haven’t fully tested this; it’s more a working prototype than production-ready code (and I wouldn’t use SQLite for production in this case). But here’s code that works, at least for now. It gets all the Australian security ticker names in the format used by Yahoo (finishing with .AX for the ASX) and their matching product\_id values for my database; finds the latest data data is available in the database; downloads anything additional to that; normalises it into long format and uploads it to the fact table in the database.

This took a couple of hours to run (I didn’t time it precisely).

#================stock price and volume information===========

all\_stocks <- dbGetQuery(con, "SELECT product\_id, ticker\_yahoo

FROM d\_products

ORDER BY product\_id") %>%

as\_tibble()

i=1

for(i in i:nrow(all\_stocks)){

# display a counter ever 20 iterations so we know we're making progress

if(i %% 20 == 0){

cat(i, " ")

}

ax\_ticker <- all\_stocks[i, ]$ticker\_yahoo latest <- as.Date(dbGetQuery(con, glue("select

max(observation\_date) as x from f\_prices\_and\_volumes

where product\_id = {all\_stocks[i,

]$product\_id}"))$x)

if(is.na(latest)){

start\_date <- as.Date("1980-01-01")

} else {

start\_date <- latest + 1

}

if(start\_date <= Sys.Date()){ tryCatch({

df\_get <- data.frame(getSymbols( ax\_ticker,

src = 'yahoo', from = start\_date, to = Sys.Date(),

auto.assign = FALSE))

if(nrow(df\_get) > 0){

df\_get$observation\_date <- row.names(df\_get)

row.names(df\_get) <- NULL

names(df\_get) <- c("open", "high", "low", "close", "volume", "adjusted", "observation\_date")

upload\_data <- df\_get %>% as\_tibble() %>%

mutate(product\_id = all\_stocks[i, ]$product\_id) %>% gather(variable, value, -observation\_date,

-product\_id) %>%

inner\_join(d\_variables, by = "variable") %>% select(product\_id,

observation\_date, variable\_id, value) %>%

filter(observation\_date >= start\_date) %>% mutate(observation\_date =

as.character(observation\_date))

dbWriteTable(con, "f\_prices\_and\_volumes", upload\_data, append = TRUE)

}

},

error=function(e){})

}

}

# Updating the product dimension with some summary data

The final steps in the extract-transform-load process are some convenience additions to the database. First, I update the d\_products table which has a latest\_observed column in it with the most recent observation for each product:

#===============Update the observation dates in the dimension table============

# This is much more complicated with SQLite than in SQL Server because of the

# apparent inability of SQLite to elegantly update a table via a join with

# another table. There may be a better way than the below but I couldn't find it:

sql1 <-

"CREATE TABLE tmp AS

SELECT product\_id, max(observation\_date) as the\_date FROM f\_prices\_and\_volumes

WHERE observation\_date IS NOT NULL GROUP BY product\_id;"

sql2 <-

"UPDATE d\_products

SET latest\_observed = (SELECT the\_date FROM tmp WHERE d\_products.product\_id = tmp.product\_id)

WHERE EXISTS (SELECT \* FROM tmp WHERE d\_products.product\_id

= tmp.product\_id);"

sql3 <- "DROP TABLE tmp;"

dbSendQuery(con, sql1) dbSendQuery(con, sql2) dbSendQuery(con, sql3)

Finally, I want to create a wider version of the data, closer to my original idea of a fact table with one row per product-date combination, and nine fact columns (for open price, volume, short position, etc). In another database I would use an indexed or materialized view for this sort of thing, but SQLite doesn’t support that. I tried making a view (basically a stored query) but its performance was too slow. So I created a whole new table that will need to be created from scratch after each update of the data. This isn’t as disastrous as it sounds - an indexed view does something similar in terms of disk space, and it only takes a minute or so to run this. And it is a convenient table to have.

So here’s the final step in this whole data upload and update process, creating that wide table from scratch. Note the clunky (to R or Python users who are used to things like spread() or pivot\_wider()) way that SQL pivots a table wide, with that use of the SUM(CASE WHEN

...) pattern. It looks horrible, but it works (so long as you know in advance all the column names you are trying to make in the wider version):

#============Create a denormalised (wide) version of the main fact table=======

# In another database system this would be a materialized view or indexed view or similar,

# but we don't have that in SQLite so it is a straight out table. Note that this roughly

# doubles the size of the database on disk; and duplication

means we need to remember

# to re-create this table whenevedr the original fact table updates.

sql1 <- "DROP TABLE IF EXISTS f\_prices\_and\_volumes\_w"

sql2 <- "

CREATE TABLE f\_prices\_and\_volumes\_w AS SELECT

open, high, low, close, volume,

observation\_date, c.ticker\_yahoo, c.ticker\_origin, c.latest\_full\_description,

SUM(CASE WHEN variable = 'open' THEN value END) AS SUM(CASE WHEN variable = 'high' THEN value END) AS SUM(CASE WHEN variable = 'low' THEN value END) AS SUM(CASE WHEN variable = 'close' THEN value END) AS SUM(CASE WHEN variable = 'volume' THEN value END) AS

SUM(CASE WHEN variable = 'adjusted' THEN value END)

AS adjusted

SUM(CASE WHEN variable = 'short\_positions' THEN value END) AS short\_positions,

SUM(CASE WHEN variable = 'total\_product\_in\_issue' THEN value END) AS total\_product\_in\_issue,

SUM(CASE WHEN variable = 'short\_positions\_prop' THEN value END) AS short\_positions\_prop

FROM f\_prices\_and\_volumes AS a INNER JOIN d\_variables AS b

ON a.variable\_id = b.variable\_id INNER JOIN d\_products AS c

ON a.product\_id = c.product\_id

GROUP BY observation\_date, ticker\_yahoo, ticker\_origin, latest\_full\_description"

dbSendStatement(con, sql1)

dbSendStatement(con, sql2) # takes a minute or so

# Exploratory analysis

Phew, now for the fun bit. But I’m going to leave substantive analysis of this for another post, as this is already long enough! I’ll just do two things here.

First, let’s look at a summary of how many data points we’ve got in the database

**variable n number\_products number\_dates**

|  |  |  |  |
| --- | --- | --- | --- |
| open | 880775 | 1873 | 8491 |
| high | 880775 | 1873 | 8491 |

**variable n number\_products number\_dates**

|  |  |  |  |
| --- | --- | --- | --- |
| low | 880775 | 1873 | 8491 |
| close | 880775 | 1873 | 8491 |
| volume | 880775 | 1873 | 8491 |
| adjusted | 880775 | 1873 | 8491 |
| short\_positions | 1318329 | 3048 | 2694 |
| total\_product\_in\_issue | 1318326 | 3047 | 2694 |
| short\_positions\_prop | 1318204 | 3046 | 2694 |

That all looks as expected. In total we have about 9 million observations. There are many securities with short positions reported to ASIC that I couldn’t find prices and volumes for in Yahoo Finance, which is interesting and worth looking into, but not astonishing.

That table was created with this SQL (and a bit of R sugar around using knitr and

kableExtra, not shown):

SELECT

variable, COUNT(1) AS n,

COUNT(DISTINCT(product\_id)) AS number\_products, COUNT(DISTINCT(observation\_date)) AS number\_dates

FROM f\_prices\_and\_volumes AS a INNER JOIN d\_variables AS b

ON a.variable\_id = b.variable\_id GROUP BY variable

ORDER BY b.variable\_id

Finally, some real analysis. What can we do with this database? Here’s an example of the sort of thing that’s possible with this asset that wasn’t earlier. This is an answer to my hypothetical question I started with - what are the most traded and fastest growing (in price) securities on the ASX?

That chart was created with this code, which has three substantive bits:

an SQL query (and R code to send it to the databse) that grabs the data we need, averaged by year for each product, from the wide table defined above

a little function purely to change the upper/lower case status of product names for the chart

ggplot2 code to draw the chart.

#- Recent growth-

sql <- "

WITH annual\_vals AS

(SELECT

CAST(STRFTIME('%Y', observation\_date) AS INT) AS

year,

AVG(adjusted) AS avg\_adjusted, sum(volume \* adjusted) AS vol\_val, latest\_full\_description, ticker\_origin

FROM f\_prices\_and\_volumes\_w

GROUP BY latest\_full\_description, ticker\_yahoo, STRFTIME('%Y', observation\_date))

SELECT

SUM(CASE WHEN year = 2020 THEN avg\_adjusted END) AS adj\_2020,

SUM(CASE WHEN year = 2021 THEN avg\_adjusted END) AS adj\_2021,

SUM(CASE WHEN year = 2021 THEN vol\_val END) AS

vol\_val\_2021,

ticker\_origin, latest\_full\_description

FROM annual\_vals WHERE year >= 2020

GROUP BY ticker\_origin, latest\_full\_description

HAVING SUM(CASE WHEN year = 2020 THEN avg\_adjusted END) > 0" recent\_growth <- dbGetQuery(con, sql) %>% as\_tibble()

d <- recent\_growth %>%

mutate(gr = adj\_2021 / adj\_2020 - 1) %>% filter(vol\_val\_2021 > 1e6) %>% arrange(adj\_2021)

#' Convert upper case security names to title case #'

#' @param x a character vector

#' @param rm\_ltd whether or not to strip "Limited" and "Ltd" from titles as unwanted clutter

#' @details A convenience function for labelling securities on a chart, which

#' converts to title case, keeps ETF (Exchange Trade Fund) in capitals, and

#' can remove 'Limited' altogether better\_case <- function(x, rm\_ltd = TRUE){

x <- toTitleCase(tolower(x)) x <- gsub("Etf ", "ETF ", x) if(rm\_ltd){

x <- gsub("Limited", "", x)

x <- gsub("Ltd", "", x)

}

x <- str\_trim(x) return(x)

}

set.seed(123) d %>%

ggplot(aes(x = vol\_val\_2021 / 1e6, y = gr, colour = ticker\_origin)) +

geom\_point() +

geom\_text\_repel(data = filter(d, gr > 2 | vol\_val\_2021 > 100e6),

aes(label = better\_case(latest\_full\_

description)),

alpha = 0.9, size = 2.6) +

scale\_x\_log10(label = dollar\_format(suffix = "m")) + scale\_y\_continuous(label = percent) + theme(legend.position = "none") +

labs(x = "Value of transactions in 2021, to 14 February", y = "Growth in price\nfrom average in 2020 to average

in 2021",

title = "High volume and growth securities in the ASX,

2021",

subtitle = "Labelled securities are those with volume

of trades > $100m or growth >200%",

caption = "Source: Yahoo Finance via

[http://freerangestats.info.](http://freerangestats.info/) This is not financial advice.")