Forewords

This database application aims, through complex reports, to help an Industrialist in EVE Online to make decisions over what to build, where to sell, and from where to source any needed materials.

What we in this project consider *reports* focuses on processing data into useful information. We have deliberately overlooked all formatting goodies common to spreadsheets. (We admit, however, that such rich formatting, especially on top of the rich intelligence provided by this system to begin with, would, in EVE-lingua certainly be called “Epic”.)

Central to this project is to make a report layer, where all the necessary business data is either linked to or preprocessed and stored. This layer is the table Produce. It takes much complexity off from the actual reports, and consequently speeds them up into very convenient response times.

In this project we assume, as the already existing system the tables Part and Composite as well as the tables/views involved when fetching Market Data and Player Data. This assumption is not far off, as all these data structures do is simply mimic how data is already stored and presented in the game. “Simply”, in the sense that these features were made as slim as possible in order to provide a baseline system that then allowed to focus on the needs and details of the reports.

Even so, that baseline proved to be a lovely playground for such features as database constraints, external tables, xmldb, merge-clauses, recursive computing, and all the little details regarding accessing a resource in the internet though a secured connection.

Disclaimer

It should be obvious, but this system does not as itself automatically lead into huge amounts of cash flows (otherwise known as Interstellar Kredit, or ISK) into your EVE Online wallet. But the reports will surely give you plenty of relevant intelligence right into those moments when you are out there in the skies gaining your insight on market situation and its effects on your sourcing and production needs.

As for the source code, it helps a lot to use a virtual machine or another way that allows backup the image from a state where everything is clean and works fine. In such sandbox it is safe to play around freely and, when things turn sour, to restore the working state from the image. This is quite important as no one else is going to take responsibility of what you do with this code (should also be obvious).

EVE Online is

If you know almost everything there is to know about EVE Online and its industry gameplay, then maybe everything here is self-explanatory and you may begin from the screens of the query results and then jump into the SQLs.

EVE Online (http://www.eveonline.com) is a multiplayer online role-playing game where players acquire spaceships and equip them with useful modules and then fly into danger and have fun. The ”fun” part basically means that someone is going to loose his ship in fire and has to ”reship”, ie. acquire another ship and modules for it.

EVE Manufacturing

For an Industrialist, the constant material losses that players make, directly translate into a demand of those things. A demand that, by a very important meaning, is not constant but fluctuating. Evidently the profit-seeking Industrialist will greatly benefit from quantitative data over the market opportunities.

Almost everything that exists in EVE Online is player built. Players do market research, go through the math, and then build things to sell them in hopes for good returns. This database application will help in all that by basically integrating information from three sources: Industry information, Market data, and Player data.

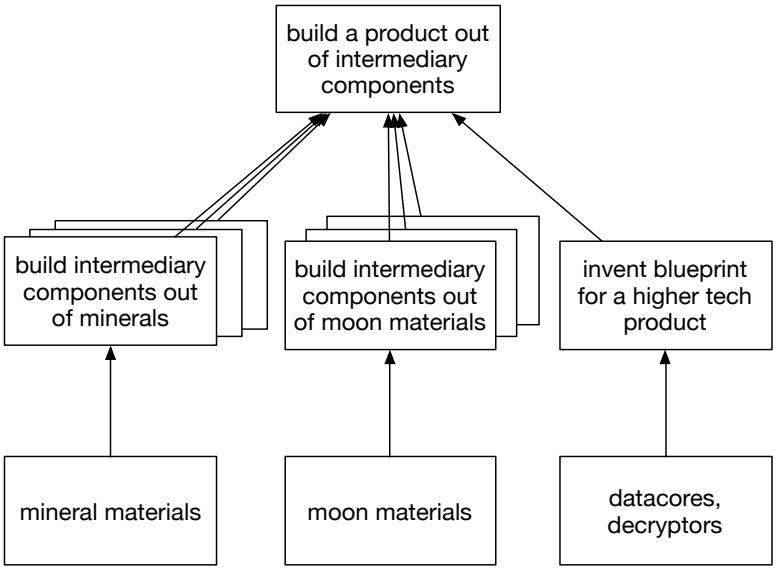

Figure 1 Hierarchy of Parts

Lets first look at the process of building an advanced ship in EVE Online. Figure 1 illustrates how you need raw materials that you build to intermediate materials or components that you then use as input to build the advanced ship. Examples in the forms of Screenshots are in the folder 0_documents/samples.

Some useful products can be directly built from input materials and sold as such for profit at the market (like the ship Arbitrator or the advanced components). Others, however, may have many generations of builds through intermediary levers to the final product. Lets generalize this to our first attempt of a data model (Figure 2).

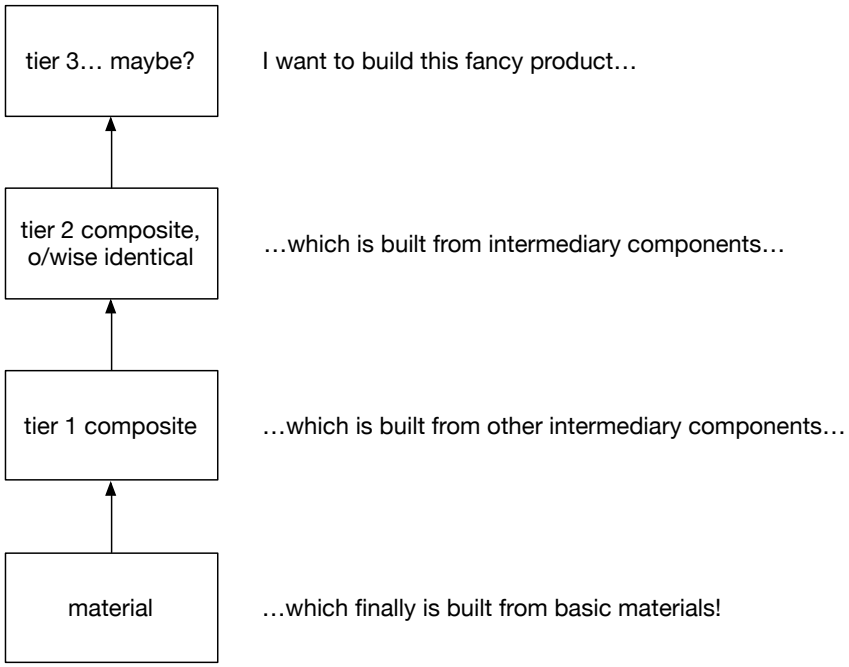

Figure 2 First degree of abstraction to the hierarchy

First thing to notice here is that there is a part that has a market, which can be somehow refined into another part, which also has a market… and so on. Ok the graph we started with had only three levels and this one has four. True, but the added fourth layer is to illustrate, that since there is already a recursion up to three generations, nothing will guarantee that there will not eventually be a fourth and fifth generation too.

Actually the long recursive chain exists already, as those moon materials must first be harvested, then "reacted" many times over and then they may enter that manufacturing process. This system just has no visibility over all that, yet. But reprocessing minerals for raw materials out of Ore sure is in the scope. The mineral reprocessing even follows a similar pattern, just the other way around: as one product is built out of many materials the one ore actually breaks down into many mineral materials.

And so the solution that will cover manufacturing, will also come to cover the appraisal of ore. Image file 4_what_is_ore_really_worth_to_me_now.png illustrates this. Note that this is the same report as in the image file 1_breakeven_of_produce_and_breakdown_to_parts.png.

Data Model

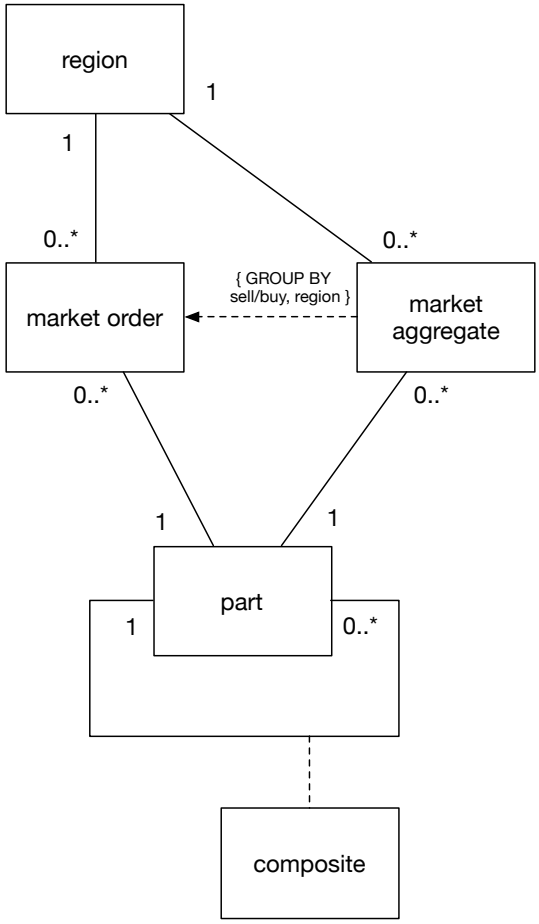

Figure 3 The Data model

Figure 3 shows the data model of this system. It is a Tree of Parts: every Part is either a root or has a parent; has max 1 parent and that parent may have * children, or it may be a child.

Then we have the table Composite with details on how a parent is connected to its children (parent = good, child = part). Namely what parts and how many of them each are needed to build that good, or a more advanced part. These are default quantities, which the Industrialist can bring down a bit though Research on Material Efficiency.

Also, very importantly, we have market data for *every* part. (Well, not exactly, because some items in the game only sell outside the market mechanism, but that's fixed into the game, we cannot do much about that in this system).

(The association between part and Region is squeezed out (1-0-1) as there might be no market order up to connect these. But this is ok, because there is currently no need to maintain identities between parts that sell at different regions; we just want to acquire those parts from somewhere.)

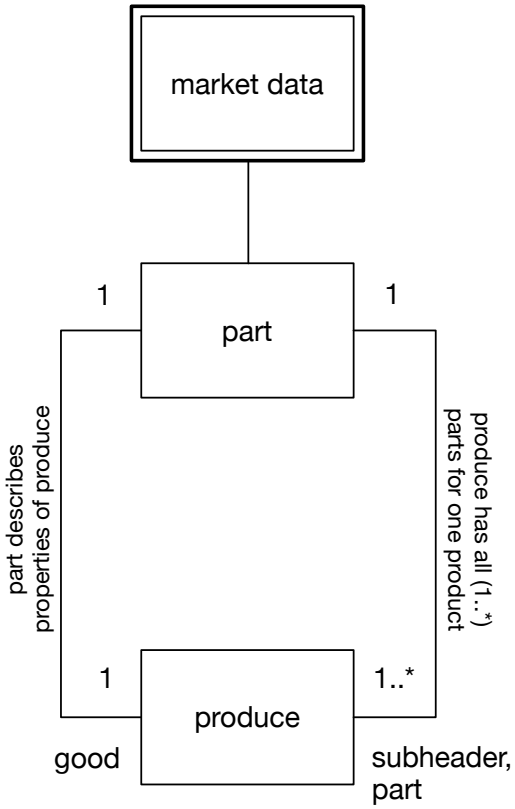

Figure 4 the table Produce as our Report Layer

Finally we make a manageable structure out of that tree of parts, as shown in Figure 4. This table is called Produce and the system will populate it automatically out of the tables: part and composite.

In the table produce we don't have those default quantities but instead the *actual* quantities where material efficiencies are already applied, including the extra 2% that comes from operating our factory at a Player Owned Starbases (POS). Further, the population is intelligent enough to 'know' the effects on the material efficiencies that are intrinsic to the recursive process. Image file 1B_the_tree_of_parts_to_build_a_pilgrim.png illustrates this.

An example (see the file pilgrim_composite.txt and the screenshots in folder 0_documents/samples):

Manufacturing a Pilgrim requires (ia.) some Nanoelectrical Microprocessors.

Building those 1,800 units of those processors require 30,600 units of Tungsten Carbide:

1800 * **17** = 30,600

Except that our blueprint copy for the Pilgrim has a material efficiency of 4%, which, combined with POS bonuses +2% material efficiency, takes the required amount of those processors down to 1,694 units.

Also our blueprint for the processor itself has a material efficiency of 10%, which, at out POS, brings the total required amount of tungsten carbides down to 25,400 units:

CEIL(**1800** * 0.96 * 0.98) * 17 * 0.9 * 0.98

or

1694*14,994 = 25,400 tungsten carbides needed (ia.) (approx)

Finally in EVE Online all the quantities are rounded up, or ceiled. But only at build time and so the final CEIL is done at the reports (that SELECTs FROM produce).

Populating the loader tables

For a demo on how to populate the loader files (part.txt, composite.txt) see the files pilgrim_part.txt and pilgrim_composite.txt in folder 0_documents/samples. If you compare those loader files with the screenshots in the same folder, you will eventually comprehend the mapping.

Final thoughts

Hopefully, in the end, you feel and/or think that the complexity at those loader files as well as this full system, are actually helpful to understand the Industry aspect of EVE Online. And that we have a far cry from how everything is actually presented in the game itself: scattered all over the place into info-boxes, as visualized by those screenshots.