Oil & Gas Query Builder (OGQB)

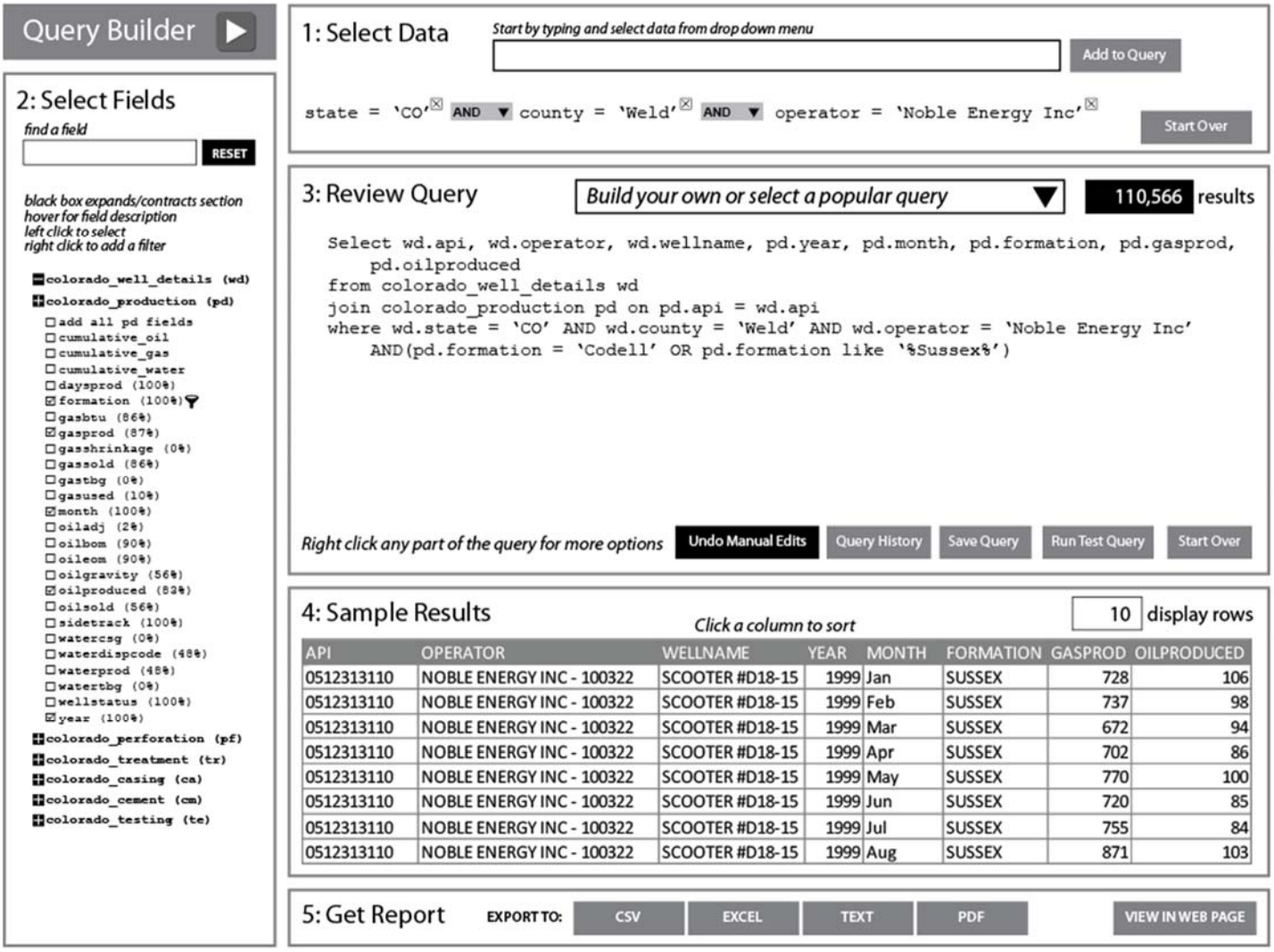
Builders will continue to add features long after it's necessary. It's human nature. Many of these new features add little to the core value of a product or service. Some argue extra features are clutter/complexity, detracting from the core offering. And it doesn't matter if it's Facebook or a BMW, our core needs were met many, many versions ago.

Now let's look at structured query language (SQL).

> SELECT emp_id AS EmployeeID, lname + ' , ' + SUBSTRING(fname,1,1) + ' . ' AS Name, ' Has been employed for ' , DATEDIFF(year, hire_date, getdate()), ' years. ' FROM employee

The above query was found in an 18 year old book (1998), called "Sams Teach Yourself Microsoft SQL Server 7.0 In 21 Days." The above query could have been written this morning. SQL is the opposite of Facebook and BMW. They got it right decades ago and then left it alone. But SQL has a major failing. By now, every human on earth should be writing basic SQL queries.

Some have tried and failed to help non-programmers create queries by building an "improved" UI on top of SQL. There is no reason to reinvent the wheel. SQL is proven, stable, robust and has vast training/help for composing queries.



I suggest we teach non-programmers how to write robust SQL queries. We stay true to the language. We don't deviate from something that has performed so well for so long. Instead we focus on bringing the humans to query language, not the other way around.

GUTCHECK: Screencaps in this document will look plain and uninspired. That's ok. This document is 100% function and 0% form (look and feel). Many graphic/web designers can take this document and make a nice interface, even with limitations of a desktop application. Button positions and label text can be decided later.

Benefits of OGQB

"Give a man a fish, and you feed him for a day. Teach a man to fish, and you feed him for a lifetime."

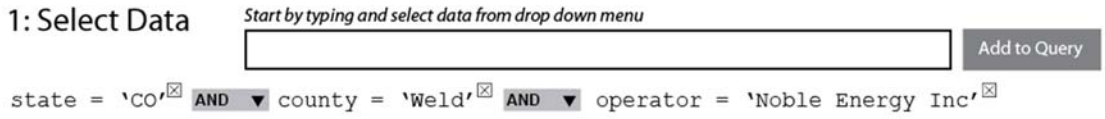
User interfaces = less than happy users. Think about your own experiences. Even if a UI is designed just for me, my needs are constantly changing. But when development teams try to satisfy thousands of users, the interface is so generic, watered down and boring, no one is satisfied. Features are tailored only to the (very) simplest of users.

Facebook should not be adding more stuff to their user interface. Instead they should be teaching users how to write code and build personalized experiences over FB data. How does this relate to Oil & Gas? Walk through a cubicle of engineer's and all you see is Excel running on screens. This is how desperate engineers are to have control of their data. They have advanced degrees, make six figures and make important decisions using a $99 piece of software.

OGQB is a user interface. But it's a training interface. Our goal is to get users to start authoring their own SQL queries and wean them off all (data) interfaces. We want them researching on the web for new types of query syntax that goes way beyond the OGQB interface. SQL is not an interface, it's a canvas. It teaches us how to fish for data ourselves.

Select Data

As stated above, button placemat, size of boxes, etc. does not matter for this document. I am only showing the major components of a query trainer. This is important. We are not building a(nother) visual query tool. We are training non-programmers how to write their own queries. Let's unpack the training features…



Step one is to tell us what data you want to retrieve. This is part of our where clause in a SQL query.

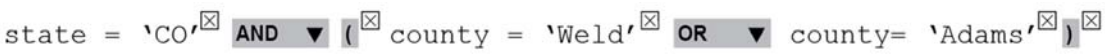
As a user types in the search box, a drop down gives them options for what to select. Choose operator: "Noble Energy Inc" and it will add Noble to the where clause in the SQL query (operator = 'Noble Energy Inc'). It automatically puts the Boolean operator AND between two parts of the query, but users can change this to OR.

```
Select wd.api, wd.operator, wd.wellname, pd.year, pd.month, pd.formation, pd.gasprod,
    pd.oilproduced
from colorado_well_details wd
join colorado_production pd on pd.api = wd.api
where wd.state = 'CO' AND wd.county = 'Weld' AND wd.operator = 'Noble Energy Inc'
    AND(pd.formation = 'Codell' OR pd.formation like '%Sussex%')
```

As users are selecting geographies, operators, fields, etc. we transfer these selections into SQL language. Over time, users begin to learn how to write their own where clauses. But this part gets tricky. We need a process to map layperson selections into SQL. For example, Weld County in Colorado. The user may select county: "Colorado/Weld" from the drop down, but we change that to state = 'CO' and county = 'Weld'.

Note the number of Noble Energy options in the above drop down. We might create another option that says… operator: "All forms of Noble Energy." This would translate into operator like 'Noble Energy%' instead of operator = 'Noble Energy Inc'. All of this would be managed behind the scenes. After a user does 40-50 queries, s/he begins to learn the difference between like and equals and how wildcards are used in where clauses.

This gets more complicated. When a user adds two of the same field, we will automatically default to OR as the operator. This means we may have a nested query and need to add parentheses. Again, ignore look and feel.



The above example will query both Weld County and Adams county. Without the parentheses, we might be looking for O&G data in Adams County, Colorado OR Adams County, Ohio. We can probably pick up 80% of the cases where parentheses are needed for nested queries, but we also need a way for the user to add or delete parentheses, as needed. And if a user makes a mistake, click the X and that part of the where clause will be eliminated.

We might remove all the X boxes and just tell the user to right click part of the where clause to delete it. This can be decided by application designers, but we need a way to select a starting point for our query.

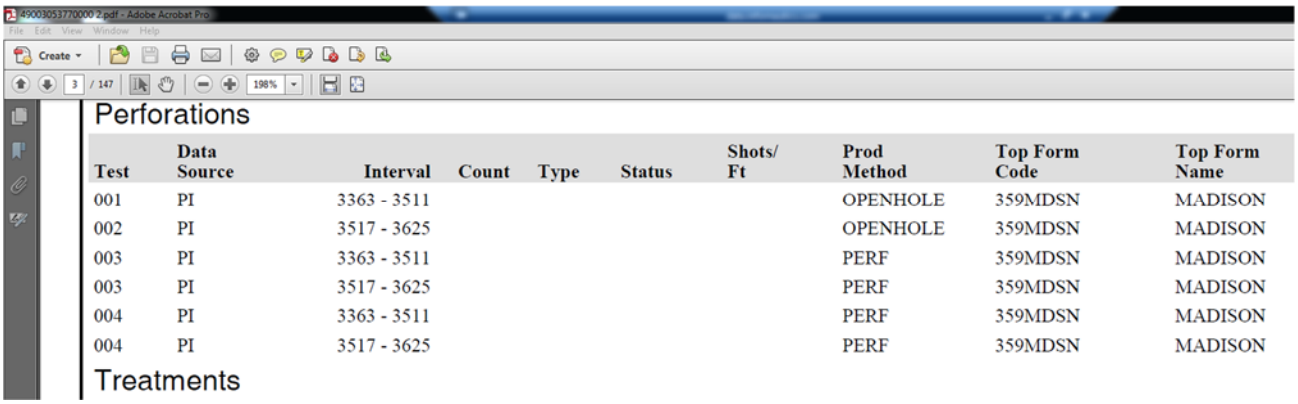
Select Data Fields

Many data providers will come up with a "national" set of fields and try to cover all oil and gas data with a single schema. This creates a huge problem for energy data providers. For example, in the case of IHS…

> 1,575 of 1,936 (81.35%) wellbore elements are filled in less than 50% of the time.
> 397 of 790 (50.25%) production elements are filled in less than 50% of the time.

Users do not like empty fields! And that's an understatement. And many data providers will only generate "canned" (Excel) reports with tons of tabs, even if a tab is completely empty…

Still others will build 100+ page PDFs on a wellbore and just fill in the data it is has for each PDF section.



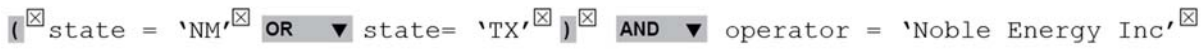
Perforations

Test	Data Source	Interval	Count	Type	Status	Shots/ Ft	Prod Method	Top Form Code	Top Form Name
001	PI	3363 - 3511					OPENHOLE	359MDSN	MADISON
002	PI	3517 - 3625					OPENHOLE	359MDSN	MADISON
003	PI	3363 - 3511					PERF	359MDSN	MADISON
003	PI	3517 - 3625					PERF	359MDSN	MADISON
004	PI	3363 - 3511					PERF	359MDSN	MADISON
004	PI	3517 - 3625					PERF	359MDSN	MADISON

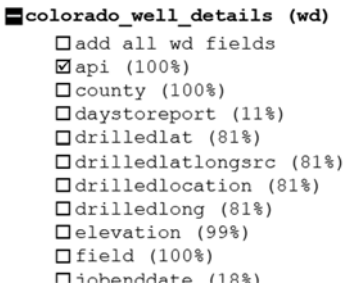
Treatments

This is the disconnect between non-programmers writing their own queries and the data provider spitting out Excel or a PDF. We must teach users to work directly with the data. They will never be (completely) satisfied otherwise.

When a user selects state = 'CO', we list every Colorado data field available. For example, Colorado has a treatment field called, "Green completions techniques utilized." Here, we see API# 05-045-15422 has stated they use green techniques. Is this field valuable? Probably not, but why eliminate any field, just because Ohio, California and Louisiana don't have it? There is huge competitive advantage to making every field, provided by each state, available to the user. And we avoid some of the IHS empty field issues because green completions will not be a field in Texas or New Mexico, etc. It only appears when the user is querying data in Colorado.



There will be some cases where a user wants to query Noble across state lines. This, too, is not a problem. There is a national set of fields (less robust) that activates if the user is querying more than one state. But we believe interstate queries will account for less than 10% of all usage. Most users are digging into a very specific area.



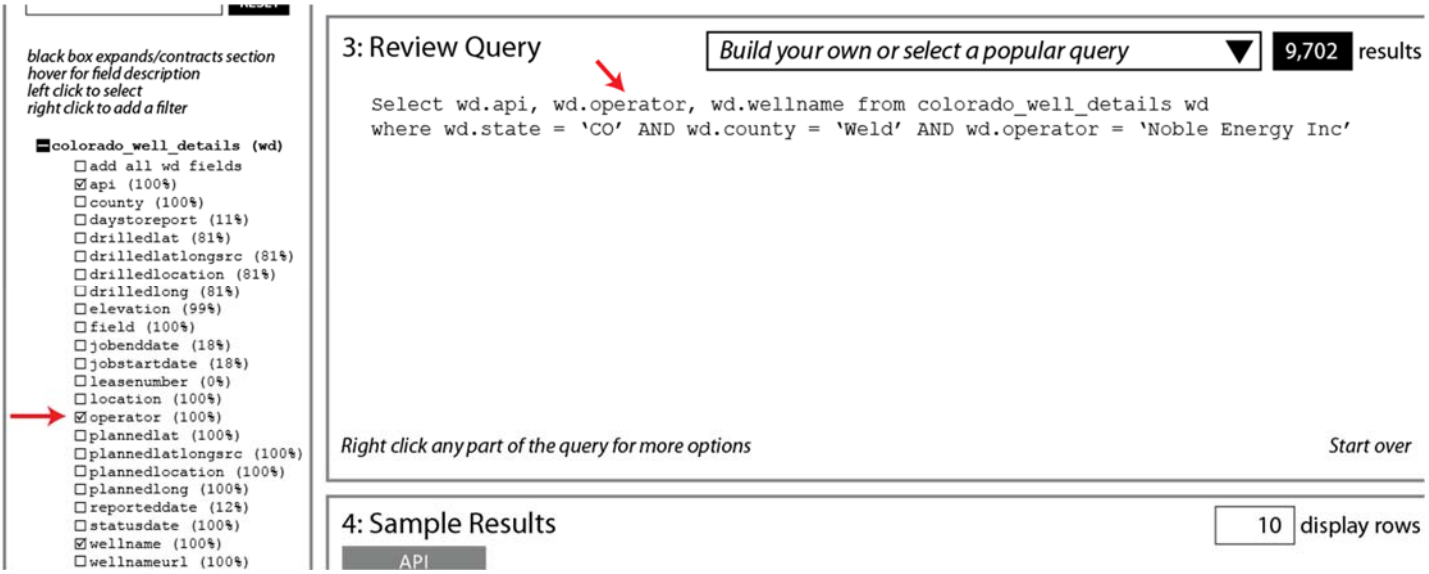
As the user works on step one, the only data field we add to the SQL query (text) is API. You can never have a query without API, since it's our primary key to connect different tables. But note the percentages (above) next to each field. When the user selects Noble Energy in Weld County, we see that daystoreport is only populated in 11% of the Noble/Weld rows. User may not want to select it. These percents are constantly updating based on the user's step one.

```
state = 'CO' AND county = 'Weld' AND ( operator= 'Noble%' OR operator= 'Kerr McGee%' )
```

If we run operator = Noble or operator = Kerr McGee, our total count jumps to 17,594 rows. Our days to report field only increases to 14%. We get the impression days to report is not a very popular (CO) field. This is a critical feature of the query trainer. The user is alerted that if an 11% or 14% field is selected, do not expect a lot of data in those columns. Do we kill the field? You decide. Maybe fields always under 10% never appear in step two. But I would err on the side of including as many fields as possible. Let's look at an example we will see later in this document…

```
☑month (100%)
☐oiladj (2%)
☐oilbom (90%)
```

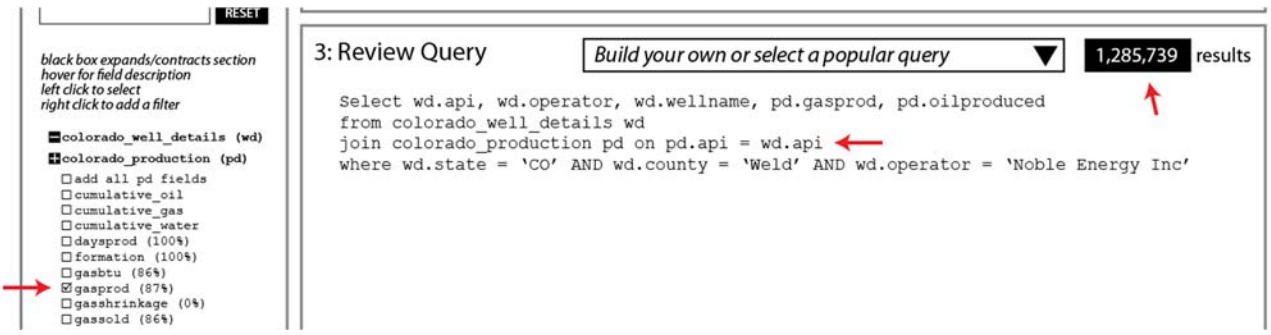
Oil adjusted is defined as "oil spilled, lost or that needs to be accounted for in the month." No surprise it happens only 2% of the time, but does that mean oiladj is a bad field? Engineers may find this field very valuable, when populated.



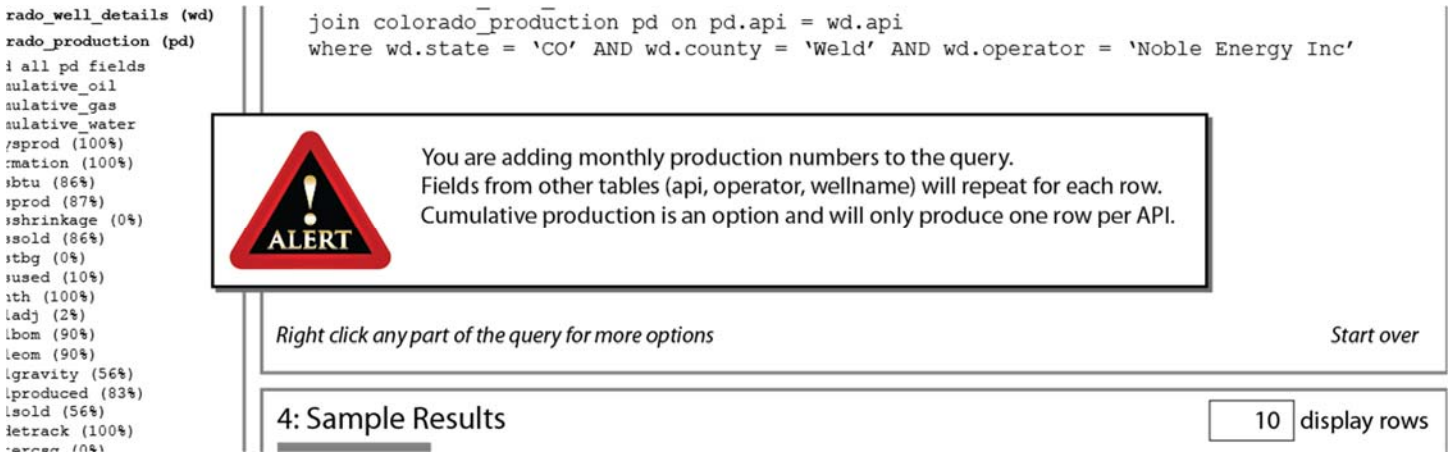
As the user checks off fields (step two), the query automatically updates. Again, users will start to understand how a select clause differs from a where clause and will someday be able to write (simple) queries manually. It should also be noted that we are already putting in things like "wd." in anticipation that joins are coming.

Joins

Assume we want to bring in some production data. We expand the colorado_production (pd) section (step two) and select gasprod. We see the query change on the fly and a new join is added. But the row count has exploded from 9K rows to 1.2MM rows.



We may add an alert screen when row counts change significantly or some fields will have lots of repeated data.



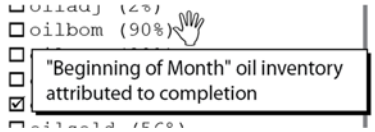
GUTCHECK: OGQB is not a web site, it's a desktop application. When the user installs/opens it, a small set of tables is downloaded to the local machine. The user is alerted when updates (new data downloads) are being made to the application. All the joins, rows counts, percent field coverages take place locally with no round trip to the server. Only when the user runs the query or exports data is the final query sent to the server for data. < Note: this may be wishful thinking, but believe it's worth a shot to move as much data to the user's desktop as feasible.

All SQL functionality does not have to roll out at one time. Inner joins may be a good starting point with the ability to add left joins, etc. down the road. Left joins may never be added. Most (99%) of user queries will be basic. But there is nothing more satisfying to a user than to have *all the data available at her/his fingertips*.

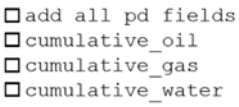
Use me as the perfect example. I have been a user of many interfaces, but as soon as I learned SQL, it was like the handcuffs came off. The FracFocus analysis would have not been possible without direct access to the data (tables). No one could have built an interface that would have satisfied me during that analysis project.

Fields

You may have noticed some of these field names are cryptic. Let's assume users will not understand that oilbom is really "oil beginning of month inventory." 1) We can rename the field at the database level. 2) When the user hovers over the field, s/he gets a popup (alt-text) that describes the field.

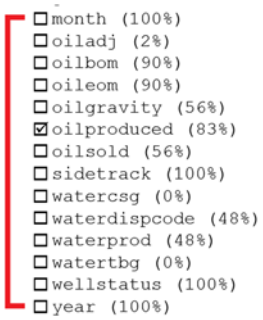


We can also add more than raw fields. As data is refreshed each month, a series of calculations are performed on the data. We can offer users cumulative production, either overall (life of well) totals or by year with a single click. And there is a single click option if the user wants to add every field from that specific table (add all pd fields).



This is where a data provider can really shine. Since they are exiting the user interface business, all that energy and time can be spent looking for new fields (hiding in state-site PDFs), second source data or cool calculated fields. This is the ultimate differentiator for a data provider – we provide more (value-add) than raw data (a commodity).

Dates are very interesting. Should month and year be combined into a single field? If the user selects oil produced, there should (probably) be a trigger to add month and year automatically.



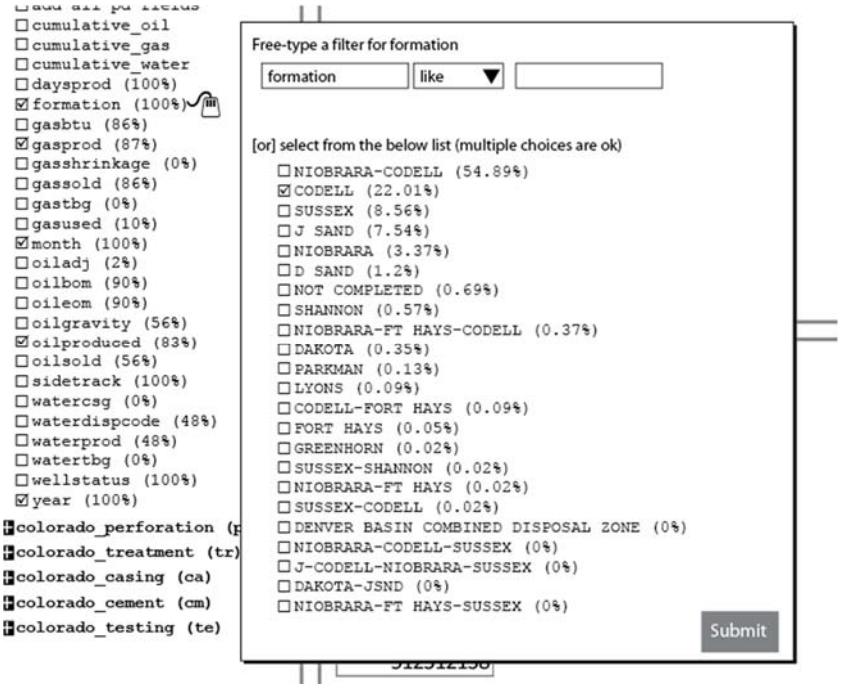
We have zero idea (below) what month and year produced 292 MCF for API# 05-123-16784.

```
select api, operator, wellname, gasprod, oilproduced from temp_noble_weld_prod
```

api	operator	wellname	gasprod	oilproduced
0512316784	NOBLE ENERGY INC - 100322	GIBBS #4	292	0
0512316784	NOBLE ENERGY INC - 100322	GIBBS #4	292	0
0512316784	NOBLE ENERGY INC - 100322	GIBBS #4	317	26
0512316784	NOBLE ENERGY INC - 100322	GIBBS #4	317	26
0512316784	NOBLE ENERGY INC - 100322	GIBBS #4	212	12

There needs to be a long list of rules that work behind the scenes. And I would avoid group by at all cost. Instead, try to pre-calculate as many of these options in advance. Cumulative_oil is a sum(oil_produced) group by api. Another field can be added for yearly production numbers. You can decide whether to have the cumulative numbers repeat many times in a production table [or] introduce a brand-new table called colorado_cumulative (cu) to the query interface. Repeating rows is bad for DBAs, but great for end users. Another table may be confusing for users, but is normalized (better) for DBAs.

Filters



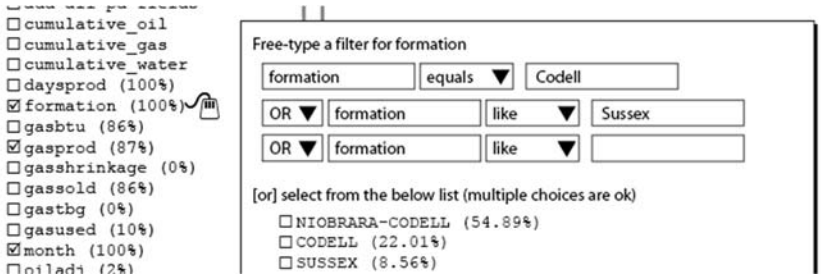
When a user right clicks on a field, an option can be selected to add a filter. If the field content options are few, they can be listed. If there are more than 50 choices for a field, it's recommended not to include the second option (list of available fields). And, of course, all numerical fields would need to be filtered using the first (free-type) option.

Note: I was working with some GIS data during the recent hurricane and came across an interesting idea – display sample content. This gives the user a "flavor" of the type of data to expect in any single field. I think this idea can be expanded. Show the top 50 choices (by count) and the check boxes. User, here are the most common entries for this field, but there are a total of X options. 9 times out of 10, the top 50 will represent 90% of the field's usage and work fine for the end user.

Once a filter is added, the query automatically updates.

```
Select wd.api, wd.operator, wd.wellname, pd.gasprod, pd.oilproduced
from colorado_well_details wd
join colorado_production pd on pd.api = wd.api
where wd.state = 'CO' AND wd.county = 'Weld' AND wd.operator = 'Noble Energy Inc'
→ and pd.formation = 'CODELL'
```

If the user decides to type in a filter, versus checking boxes, additional rows are added as soon as the user starts entering text. This allows the user to set more than one filter for formation.

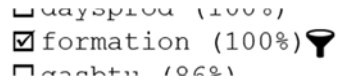


The above selections would look like this (below) in the review query section.

```
Select wd.api, wd.operator, wd.wellname, pd.year, pd.month, pd.formation, pd.gasprod,
    pd.oilproduced
from colorado_well_details wd
join colorado_production pd on pd.api = wd.api
where wd.state = 'CO' AND wd.county = 'Weld' AND wd.operator = 'Noble Energy Inc'
→ AND(pd.formation = 'Codell' OR pd.formation like '%Sussex%')
```

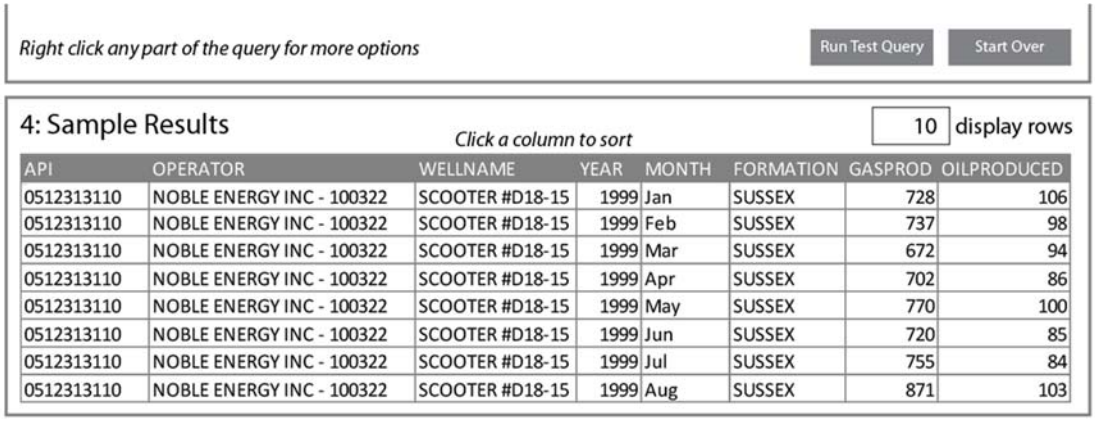
Even reading this document, do you feel yourself already learning how to compose simple queries? Imagine doing 50 or 100 of these queries. Soon, you won't need the OGQB interface at all. But the underlying data is now tied to hours and hours of an engineer's work product. You can't take away their data – all their queries will fail. They are used to field names and boutique fields within a state. Switching costs to change out the database are extremely high. And the loyalty users will feel to the "one" that taught them SQL cannot be measured in dollars.

Once a filter is set, an icon appears next to the field name.



If the user right clicks on the field or filter icon, choices are modify the filter or remove filter. If no filter has been added yet, option is add a filter.

At any time, the user can click on "run test query" and view sample results. Right now, it says 10 rows, but the default will be 100 rows and have scroll bars. Sort can be done here in the sample results section or we can add sort to the right click for each field – or we could do both?



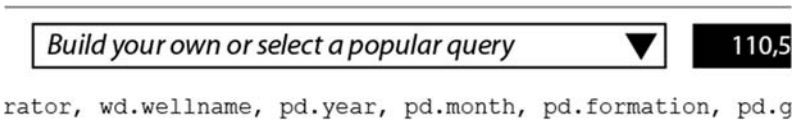
API	OPERATOR	WELLNAME	YEAR	MONTH	FORMATION	GASPROD	OILPRODUCED
0512313110	NOBLE ENERGY INC - 100322	SCOOTER #D18-15	1999	Jan	SUSSEX	728	106
0512313110	NOBLE ENERGY INC - 100322	SCOOTER #D18-15	1999	Feb	SUSSEX	737	98
0512313110	NOBLE ENERGY INC - 100322	SCOOTER #D18-15	1999	Mar	SUSSEX	672	94
0512313110	NOBLE ENERGY INC - 100322	SCOOTER #D18-15	1999	Apr	SUSSEX	702	86
0512313110	NOBLE ENERGY INC - 100322	SCOOTER #D18-15	1999	May	SUSSEX	770	100
0512313110	NOBLE ENERGY INC - 100322	SCOOTER #D18-15	1999	Jun	SUSSEX	720	85
0512313110	NOBLE ENERGY INC - 100322	SCOOTER #D18-15	1999	Jul	SUSSEX	755	84
0512313110	NOBLE ENERGY INC - 100322	SCOOTER #D18-15	1999	Aug	SUSSEX	871	103

Note: “Run Test Query” is the first time app makes a round trip to the server. So far, all the counts, percents, field definitions, filter options, etc. are residing locally. The local database (on user’s computer) has the bare minimum to build queries. Because filters are available, some fields will need complete data. For example, formations will need to be included for every row. Numbers, like oil produced, will also need to be present for < or >, etc. filters. Personally, I think some of this can be done with persistent lists and numbers in place of values in the database.

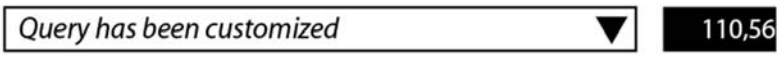
GUTCHECK: Making a round trip to the server to build the filter lists and overall queries is not a deal killer. And some of these state databases are very large. We might start with round trips and slowly move some of the critical data over to the user’s computer. Certainly, things like field lists (step 2) can be cached locally.

Review Query

The query (text) portion and results count are constantly changing as data, fields and filters are added to the query.

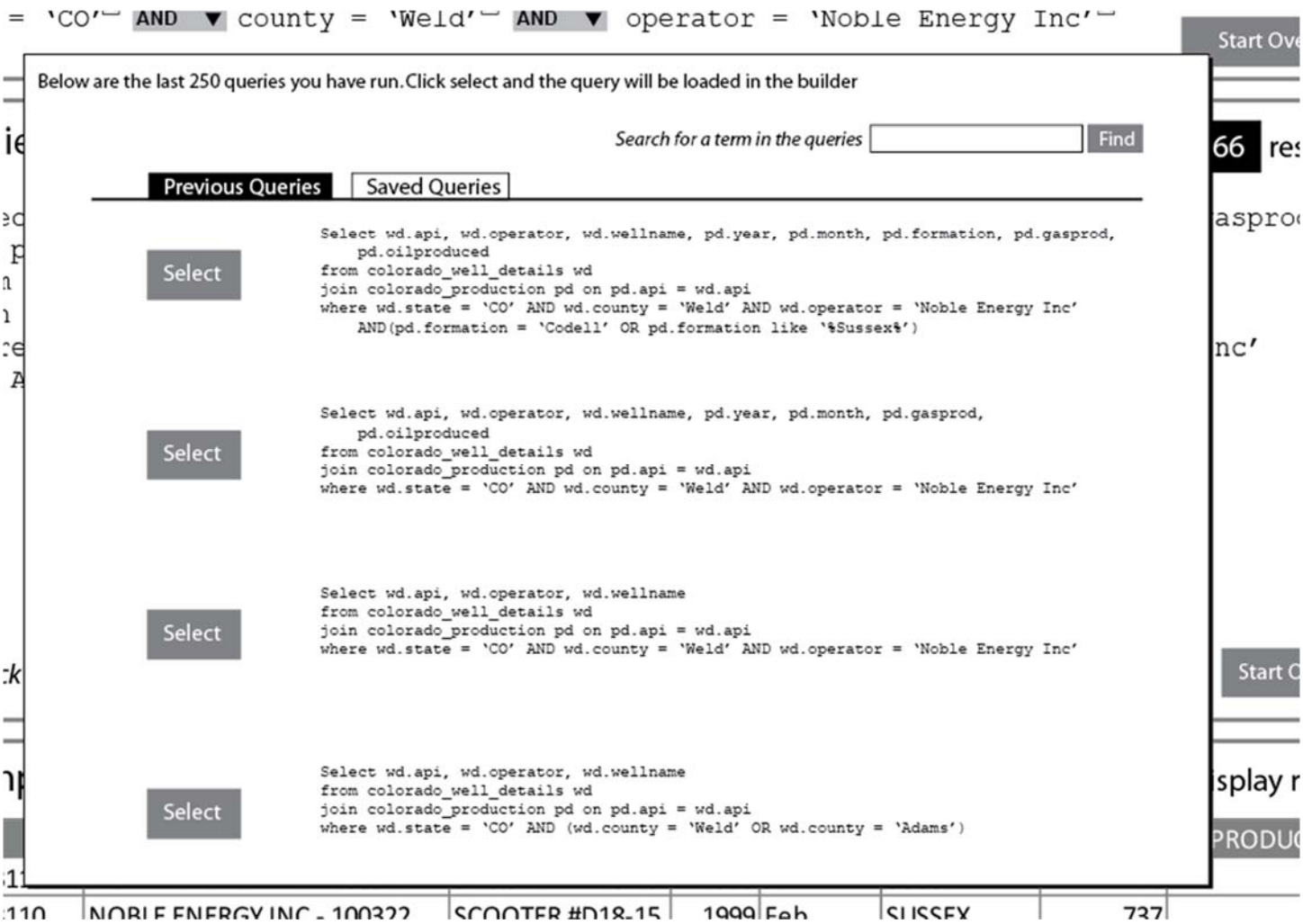


Some users will select from pre-built queries. There may be a cement query or a cumulative production query or a frac chemical query. When a query is chosen from the drop down all fields in step 2 (from that pre-defined query) are checked off. This way the pre-built can be a starting point and the user can continue to refine the query. As soon as fields are added or changed, the menu changes to “custom query.”



Other options are to start over. This clears out the query string and allows the user to choose fields or a pre-built, drop down, query. If query text exists and start over or the menu is used, an alert will pop up warning the user that the current query will be deleted. But the queries are not really deleted. They are saved in a history file and can be accessed by clicking “Query History.”

Clicking history brings up a popup of all previous queries run by this user. This is a KILLER feature and one I use every single day in our own query engine. Note there are “saved” queries in this popup too. The only difference is the user clicked save my query and gave the query a customized name. Users can also search for a term in a previous query. For example, I might search for Adams if I want to find the old Weld AND Adams query. This, too, is a very useful feature.



A query is automatically logged to history as soon as the “run” button is clicked.

The Big Win

It’s going to take users, especially engineers, about 5 minutes to start messing with the actual query text. They will see how to add fields to the query, why we start certain fields with pd., how to apply filters, do joins and use nested queries. We want them to do this and fast. Remember, this is not a tool (UI) or replacement for building queries. This is a trainer that helps users learn how to create SQL queries. The faster they learn, the more powerful queries they create.

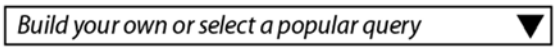
The only thing users cannot modify is the state part of the where clause, because this loads specific fields (step two) for each state. It’s better for the user to start with step one, so state-specific fields can be loaded. However, if the user changes state = ‘CO’ to state = ‘TX’, it is possible to change out the step two field selections. But an alert should popup that all fields and filters will need to be revisited because the user has changed states.

As soon as the query builder detects the user has started manually editing the query, a new button appears at the bottom of the review query section. At any time, the user can remove all manual edits and go back to what was generated using step one and two.

Another option is to right click on any area of the query text and get additional options. One might be to remove a section of the query. This will need to be smart and know how much of the query text needs to be removed to keep the query working.

Error Detection
SQL engines do have error detection, but it's often cryptic. Something like this tells us we are missing the apostrophe at the end of Sussex. This error tells us we are missing a Boolean connector near formation. This one tells us we mistyped the field month (made it plural). It's possible to write code that will make these errors more self-explanatory. SQL expects certain features in each query. Start with select, then from, then joins, then where, etc.… Code can walk through these manual queries, detect problems and alert the user.

In the beginning, users will make every mistake in the book. But over time, when they see an error, the fix will be obvious to the user. Building queries will always include mistakes (mistyping, etc.), but users will find those mistakes within seconds, like it's second nature.



Being able to choose a starter query will also reduce the learning curve substantially. The custom queries will be tuned to each state. If Colorado has a treatment summary section, colorado_treatment (tr), pre-built treatment queries will be included in the drop down. Having starter queries will be a huge help for learning how to build your own queries.

Still another option may be to have a popup "prompt" when the user chooses a pre-built query. If a formation query is selected, the query engine may ask the user, via a popup, if they wish to filter the query to a specific formation(s). This, too, is a nice feature (one we have built before). That single (drop down) query can satisfy a ton of variations through the popup prompts.

The goal of this entire effort is to train users to build their own queries. This will create huge opportunities for users. It will create a huge dependency on the provider of the data. All saved/historical queries are built around that single data model. The switch to another database would be next to impossible. All queries would have to be re-written.

A bonus feature would be to map client-specific data into the OGQB. Internal and external data can be blended into a single database (by state). Imagine an IT department waking up one morning and learning that 250+ staff were now SQL query builders, interacting directly with the data layer. This is the way it should have always been… teach users to fish!

Get Report

After the query is finished, the user can export the results using several formats. S/He can also view it inside a web page (with paging enabled), but have no idea why anyone would prefer a web page to CSV. Excel is listed just because it's the go to tool for most engineers at this point. But we would hope that Excel gets replaced by the OGQB (i.e. manually building SQL queries) and users are getting knowledge directly from (fine-tuned) queries.

The user will also be prompted at this moment to save the query. Queries are automatically added to previous history as the query text changes and test runs are executed. But when a user is pleased enough to generate an export, it's a good time to suggest saving the query for later use. A bonus feature would be to share or email queries between engineers.

Each time a query is sent to the server, it's logged. This is incredibly valuable feedback for the OGQB builder. New capabilities (software releases) are driven by analyzing the types and popularity of queries. When a user generates a report, those queries are held and monitored for any data changes. For example, if a user does a production query for Noble in Weld County, an email will be sent to the user when new data is available. This is not one email per query. All query updates will be aggregated into a single weekly or monthly email. Single click from the email and the query re-

exports data with updates. Another single click (directly from email) and the query is automatically loaded in the builder for editing.

Showing Sources
We call this the "Bobby Syndrome." There will always be an engineer throwing a fit the data is wrong. Somehow this is always the data provider's fault. But the provider is sourcing all this data directly from state sites. We will include a link to scout ticket/production/etc. source HTML pages for every single row in the database. If Bobby does not like a specific number, a single click confirms it's not the data provider, but the data submitter who screwed up.

Bonus Features

The upside for OGQB is limitless. While the goal is for engineers to author basic queries manually (free-type), OGQB can stay one step ahead. While we want users weaned off OGQB and directly writing SQL queries against the database, there are more advanced features we can introduce to power users. This may be a totally different training tool that teaches advanced options within a manual authoring interface?

SQL Resources
Because OGQB is staying true to structured query language, there are immense resources on the internet. Immense! For example, query examples/tutorials/etc. can be wired directly into the OGQB interface. Want to learn about using substrings? Click here. And these links would be curated to bring only the best content to the end users.

Color-coded Anomalies
Obviously, I am not a fan of UIs. We prefer data engineering, which bolts onto an existing data stream and adds new fields before the data reaches its destination (often a UI). Anomaly detection is a great example of data engineering. The code is constantly monitoring data for numbers/values that seem out of place. Sometimes, it's someone entering 1/1/1900 as a date. Other times, it's finding a frac record where the operator used methanol for 25% of an overall frac.

When the user exports a final report, a link can be included as part of the export to view all the outliers in a (popup) web page. Outliers can be given color-codes to make it easier for a user to judge the quality of the exported data. I suspect some users will find this feature very compelling and want it baked into the test query section of the main OGQB application.

Automatic Narratives
This is a derivative from a company called Narrative Science. They automatically write news articles (mostly stat-driven ones) and will describe charts with short narratives. There are claims of machine learning, etc., but I suspect this is Mad Libs on steroids.

Assume a list of thousands of descriptive sentences just waiting for the numbers/values to be plugged in. User creates a query and sees a narrative that includes, "In Weld County, 2 fracs were completed using propane, which is highly unusual." The Mad Libs sentence is this: In ____ County, __ frac_ were completed using _____, which is _____. And these sentences can be very advanced.

Can a user deduce that Mad Libs is generating the narratives? Perhaps, if there are one or two sentences. But if a user gets 1+ paragraphs of narrative, it will appear as if a person analyzed the data in real-time and wrote a story about it. Frankly, I think we might end up using auto-generated narratives to explain advanced (drop down) queries in lay speak.

Suspicious Data
This comes from our Frac work. Yes, 1/1/1900 is obviously bad data, where the submitter could not submit the record without something in that field. Is this really an anomaly or just bad data? There will be cases where the data just doesn't seem correct. There is a discrepancy between normal behavior (data ranges) and what the submitter is reporting. We can mark/alert on these questionable rows. A great example is a formation that is used only one time across an entire state. Yes, this is an anomaly, but more likely, it was just poorly entered data (misspelled formation?).

Graphing/Charting
We can't have a document about O&G data without throwing in Tableau, Spotfire and other visual tools. I still think little to zero value comes from an engineer staring at a pre-canned chart. When was the last time you made a major discovery/decision based on a graph? But the query builder can (has to) be built to support these tools.

There may be a drop down query specifically for generating a Tableau graph? It would include a time series component. There may be some apps that will take a direct interface (data stream) from OGQB. Save the export with a certain file extension/format and it will open directly in a third-party tool. I would not emphasize this feature, but users will ask about connections to 3rd party software. What's shocking is that Tableau/Spotfire/Excel/etc. are not even O&G apps.

System Tray Icon
This is the holy grail of OGQB. Just talk to the user's desk/laptop in the middle of the night. When installing the application, all queries and data exports are stored in a specific folder. Executed queries (text) are also stored at the vendor. When NM data is refreshed, the user's desktop wakes up and begins the data download into the right folder. The user gets a notification the next morning (via tray icon) that NM data refreshed overnight. Just like a tray icon tells us whether we are connected to the internet or have a new email (MS Outlook), an icon is feeds us O&G data.

And those weekly/monthly emails should not be discounted. Yes, they will be white-labeled (which the client will love). Being able to communicate directly with end users is mind blowing for client retention. We supported 600+ recruiters at Microsoft for 8+ years and only two at MSFT knew we weren't employees. Our SPOC and the person who paid us. We setup a vanity URL, msft.talent-labs.com and everyone thought it was part of the intranet!

Conclusion

The easiest vendor to kick to the curb is a SaaS vendor. Flip a switch and the entire thing turns off one night. And SaaS is totally overrated. Compare Outlook to Gmail, Word to Google Docs. There is zero comparison. HTML is not for work product (or frankly decision making, in my opinion).

Second, UIs baked for all satisfy none. Google could be epic. Instead it basically looks the same as it did in 2000. Google Search must cater to Grandmas and PhDs with a single interface. Facebook or Twitter makes a change and its front-page news with calls for executive's heads. How did we end up with 300 million+ people and only a handful know how to code? The vendor that teaches its users to build SQL queries will have loyalty Facebook can only dream about.

Decouple the backend from the frontend. Build a rich application and let the end users author hundreds, if not thousands of, one-of-a-kind, queries. Alert users when data is refreshed. Build a story around each query. Now, take away the backend one night. There will be mutiny the next day. All queries will cease to work. Thousands of hours building queries lost. Data updates gone. And no vendor can slip in and replace the data. Users must start over.

This is what we call "data penetration." Take O&G data and penetrate every orifice of the organization. Have the application loaded on hundreds of client machines. A user authors a query today and is still depending on that same exact query (data refreshes) two years from now. Instead of the service being centralized, it's spidered across an entire organization. Offer clients a way to integrate their internal data at zero cost. It will only create a deeper bond/dependence and pay handsomely in the long run.

Finally, here is a feature near and dear to me. We scrape a lot of data across a lot of states. There are fields that get dropped by every data provider out there, because they are not "nationally" agreed upon. I would bring every field to every user. If Colorado has a special treatment section, include it all. If North Dakota has all survey data plotted, include it. We will all be surprised to learn these dropped fields hold the real value-add for end users.