Version History Q1 2024 BDT Hackathon

Team members: saraneel@, urenamar@, riachu@, edwardvo@

Background

The Version History PRFAQ was selected for the Q1 2024 BDT Hackathon. It is one of the top requested features from users during user research, as there is currently no way to easily view the history of a dataset in DataCentral. When troubleshooting, dataset owners, consumers, and internal BDT team members rely on the API to see a detailed history of what changed. In the UI, users flip back and forth between browser tabs to try to understand the differences between two versions.

[From TVDP user interviews] Users want to easily view dataset version history without relying on the API. Users frequently look at audit information and revision history, so they can understand when changes occurred and by whom. Users are currently using the API to get this information, as it's not readily available on the TVDP. Users who don't use the API to get this information are doing "a bit of an eye test" where they are manually flipping back and forth between table versions on the TVDP to view changes.

- *"One thing I would like to talk about is having an audit column here, who was the actor, that would be really helpful...Any time there's a change in description of column or inline schema change, we don't see it here. In that scenario, I have to go to endpoint and see who created the version, but can't find out which team deleted it, or when they deleted it...To compare the v12 and v11, one way is I can go to the UI, copy it, clean up the data, but it's very manual process, so what I do is I get the SDL from here [API] and I can copy the SDL into [third party tool]...I want to see the changes between the two versions. I can copy the schema, use a third party tool to compare...It's a better way to see the changes between the two versions..."* - Data engineer (watch clip)
- *"We look at revision history pretty frequently when we try to figure out what happened...DMO and customers can go edit the table at the same time, we tend to go look at that in the Andes API as well, in terms of figuring out this is the exact order of steps that happened in terms of working backwards and reconstructing a picture of what happened on a particular table...Other big pain point is the version description and audit message are not always super helpful. If we're trying to figure out what happened to a table over time, we fall back to the API to look at there's this particular revision, here's what actually went through and changed."* - SDE
- *"We're consuming data from other teams, adjusted schema updates, suddenly they're modified. For me to understand exactly what changed and why, it's a bit of an eye test where I have to toggle back and forth between versions, different tabs...to understand what's the new column here. I would like to know why there's a v7, why is there v8, what's the difference between the two...I would like to know modifications from one version to the next."* - Sr. Data engineer
- *"Sometimes I find myself looking at a version of a table and wondering if I'm the most recent one just to check if there's a more recent one. There can be multiple active ones. I don't think there's anything on the page that tells me there's a new version, you're looking at an old one."* - Sr. Data engineer

[From Manage Datasets user interviews] Users want to see what changed between versions. Multiple users described wanting to easily see the differences between versions. Right now that information is not readily available and audit history "doesn't seem to be logging anything of value," making if difficult for users to know when and why something changed. The ability to see a dif would be "really really nice to have features" that would help new team members learn about the data quickly and "show external customers what we have changed," particularly when dealing with customer concerns.

- *"What's the difference was between previous and current version. At least for our consumption, that level of information would be sufficient. If you currently see the Hoot page, if you go and see different versions, it shows all the versions. Nowhere it shows what is different from the previous version unless the provider sends an email to all subscribers, there's no option to know what has changed from the previous version. If that information could be provided, that's the only thing we need for versions...I would like to use that [version-level info] for checking what was*

published in version 2 for the dataset, why was that deprecated, who deprecated it from my team. I want to get some context on the previous version of a dataset. I want to see who created that version or deprecate that version and reach out to or interact with them. I don't see that in one place. If i want to go back and get that information, I don't have that information." - Data engineer

- *"Just showing the current version set and showing the dif between the last one...if there's a quick view that version 9 and version 10 has this dif. If a teammate comes and they're not sure what changed, anyone can see what changed. I think that information they're not present as of today, but those are really really nice to have features. As I said, nothing is a killer as of now, but these are things that can really improve the customer experience because it'll all be in one place. Help new team members learn about the data pretty quickly."* - Data engineer II
- *"We can see when we created the table, can't see when it was fully active. When you go into audit history, it doesn't seem to be logging anything of value. I added a LOB field, I can't see any information about when it was added. Basically any changes to the table we would want to be able to see when it happened, especially when dealing with customers concerns. We need support to say you were notified via health event, here's the documentation, etc. Show external customers what we have changed. I'd like to see the details of what was executed on that version. Creator, and differences between two versions. Now we have to do an arbitrary dif for schema changes. Would be ideal for consumers to be able to see changes...Documentation to point customers to for changes between versions would be highly beneficial...A field was removed, could add a description as to why. The system should be able to pick it up like we do arbitrary dif, but then you would have the opportunity to explain what changed."* - Data engineer

Problem statement

Today, dataset owners and consumer rely on the Andes API and external tools to understand the changes between versions, when they occurred, and by whom. Users call the Andes API for multiple versions, copy and paste the SDL response into a third party IDE, and use the third party tool to compare the SDLs of the different versions. In the UI, users need to manually flip back and forth between multiple versions and do "a bit of an eye test" to try to identify the differences.

Objectives

- As a user, I want to see what changed, when, and by whom on a dataset, so I can quickly identify changes to troubleshoot without leaving DataCentral.
- As a user, I want to see the diff between versions, so I can easily understand the differences between versions without relying on APIs and 3rd party tools.

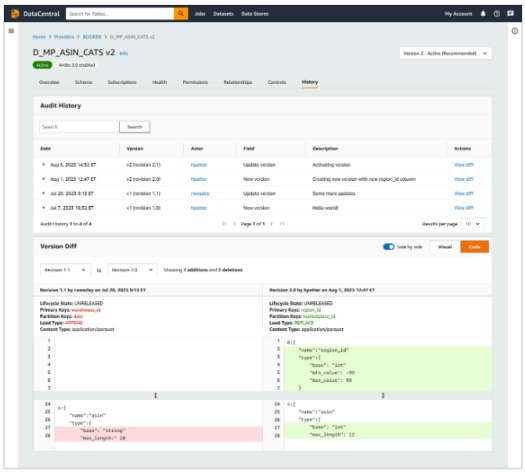
How will we measure success?

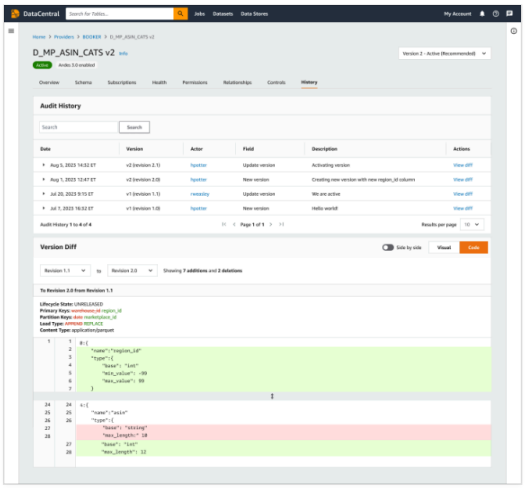
Metric	Goal	Baseline
# of clicks on History tab	Increase adoption of History tab	
# of unique users of History tab	Increase adoption of History tab	
Time on History tab	Increase adoption of History tab	
# of clicks on "View diff" action	Increase adoption of version diff feature	N/A
# of clicks on "Code" toggle	Understand user preferences	N/A
# of clicks on "Visual" toggle	Understand user preferences	N/A
# of clicks on "Side by side" turned on	Understand user preferences	N/A
# of clicks on "Side by side" turned off	Understand user preferences	N/A
# of calls to the version history API	Decrease # of calls to the version history API	

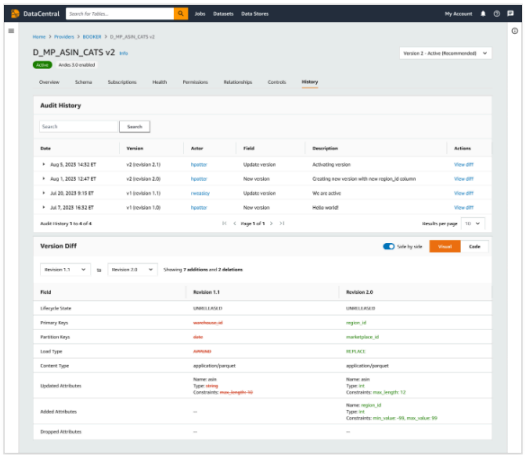
Schedule

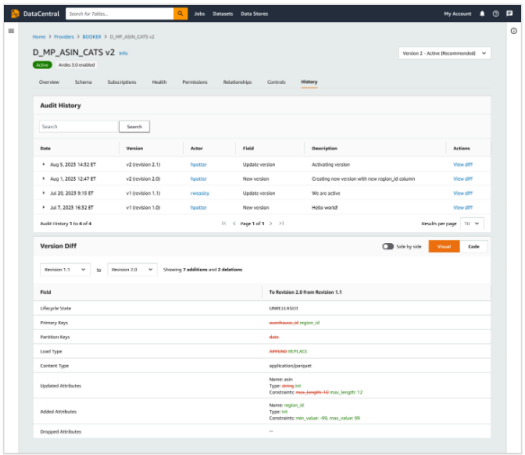
Task	Due Date	Assignee	Comments
Finalize mockups	1/22/2024	saraneel@	Initial mockups created + reviewed
Create shell for web component	1/22/2024	urenamar@	Added instruction and location here: ackage/branch:
Define interfaces	1/22/2024	riachu@, urenamar@	More or less follow these interfaces. There are a bunch of docs (Iryna, Casey) as well we can use for the row interfaces. **Version diff needs to be worked out**
Investigate 3rd party tools for code diff	1/22/2024	urenamar@	If there's a component, we could do it for hackathon
Write API/technical design doc	1/22/2024	riachu@	Input and output JSON for hackathon
Create presentation	1/26/2024	saraneel@	
[Stretch] Design realistic API	1/26/2024	riachu@	What would the API look like if we were building this for real, outside of the hackathon? How to make it scalable?
Hackathon	1/22 - 1/26	All	https://broadcast.amazon.com/videos/1012347

Designs (Figma)

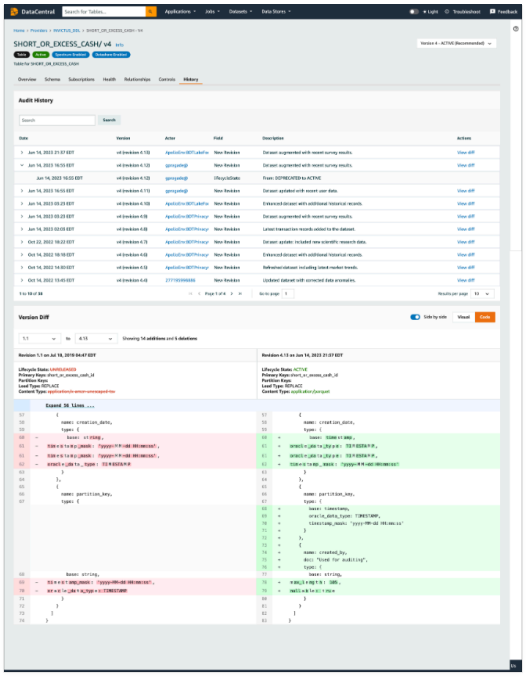


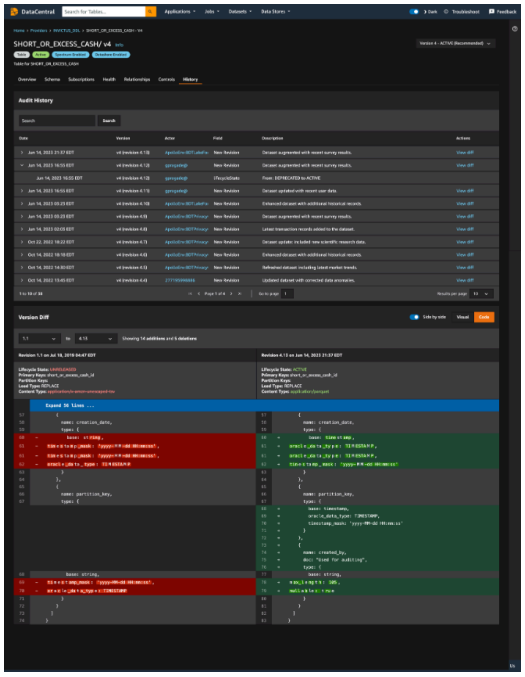


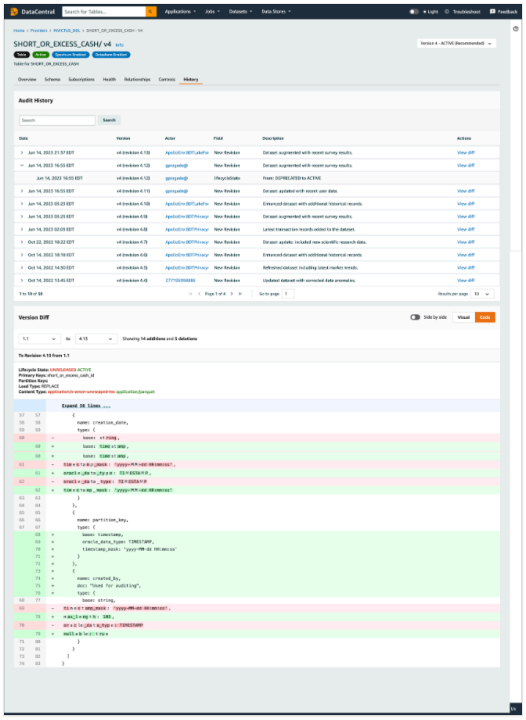


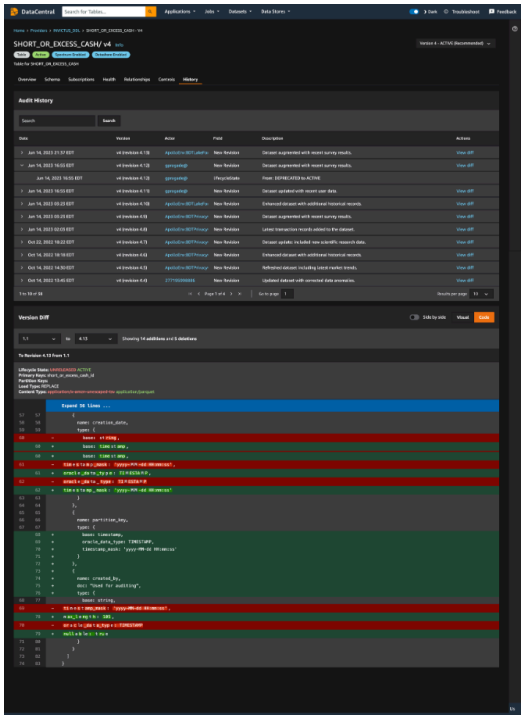


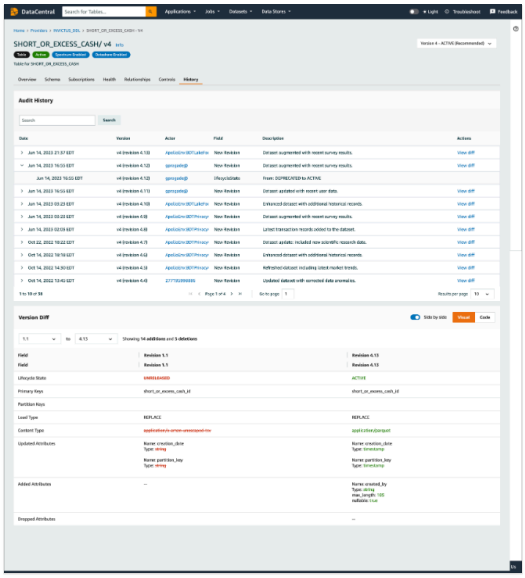
Screenshots (hackathon)

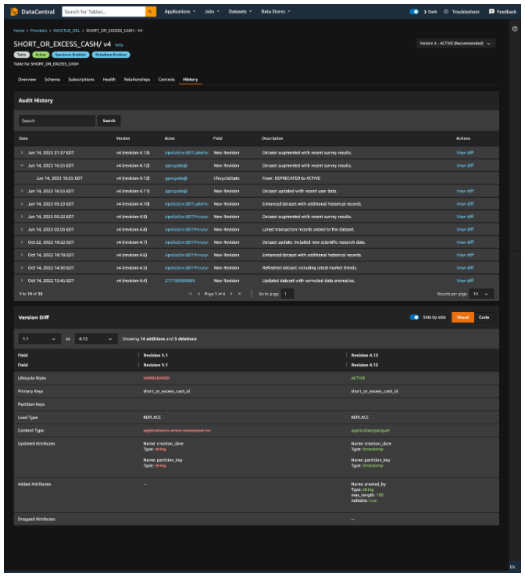


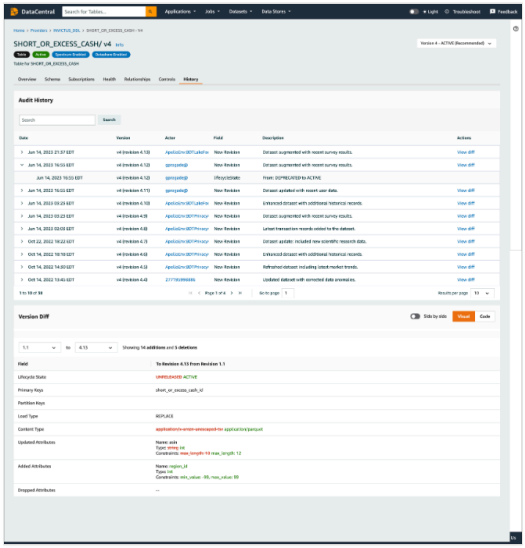


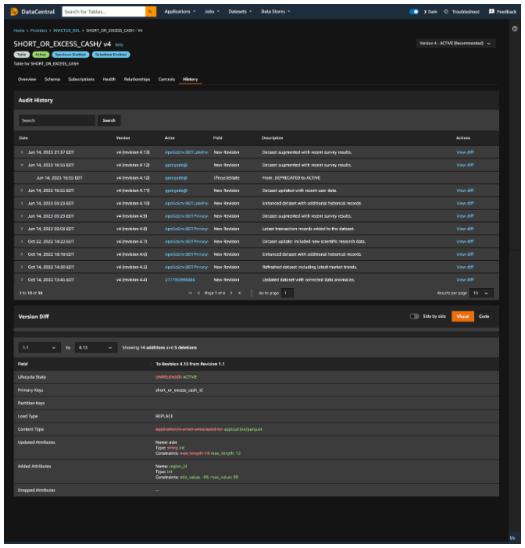












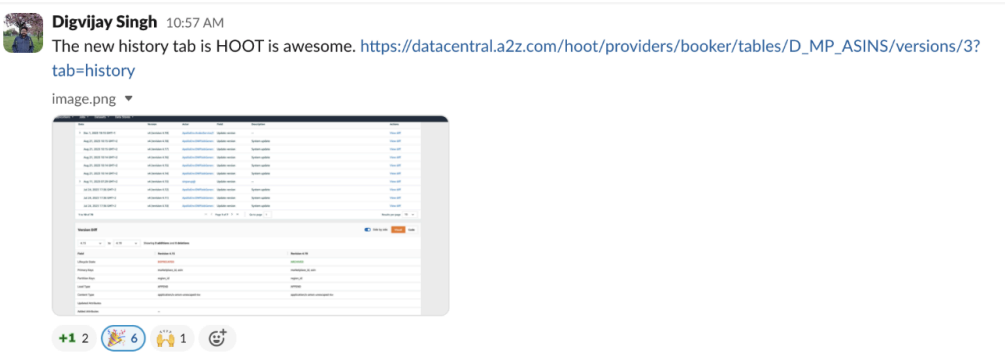
User feedback (via Slack)

- *"That is...wow, that's pretty much exactly what I was envisioning! I didn't even consider one thing that's in there too, which is the version "1.1" naming convention, presumably for minor version updates (like inline changes and whatnot). The changelog at the top and the side by side comparison are really clean, and they also visually look very familiar (which I assume is on purpose!) because they remind me of the CRUX tool for CRs."* - Sr. Data engineer
- *"This is awesome. One request would be that it can be used before submitting a new version for code reviews as well... possibly growth idea for the project :) so when we use ANDES API to do new version the schema format is heinous and then trying to line it up for code review before creating new version is a pain. Also when we create a new version there is a communication created, it would be good to see it here to understand why it was created. I struggle because there is multiple times i've created the new version (or inline schema change) but can't see what was send to downstream consumers to reference when they say they weren't notified"* - Data engineer

Post-launch user feedback

- *"I really like the option to expand each row! It's nice to have the description and the "diff" window...oh, and I just*

noticed the drop down that lets you toggle and show diffs among various versions, not just the two latest ones. Ok, yeah, I see why this won 1st place." - Sr. Data engineer

- *"I like the fancy new dataset/version history."* - Data engineer III
- *"Very exciting update and it looks great!"* - Sr. Data engineer
- *"I didnt realize version history works across table versions. This is soooo dope"* - SDE II
- 

Coding & development

PACKAGE/BRANCH:

- code.amazon: https://code.amazon.com/packages/DataCentralDatasetMicrofrontendsWebsite/trees/heads/hackathon-version-history (see README for setup, summarized below).
- Setup brazil workspace `cd ~/your_workspace`
- Make it `bws create –name Hackathon_or_whatever`
- `cd Hackathon_or_whatever` then `bws use -p DataCentralDatasetMicrofrontendsWebsite`
- Then `cd src/DataCentralDatasetMicrofrontendsWebsite`
- I usually start with `bb` and `bb i`
- local server: `bb run server` & modify App.tsx to see changes locally only.
- beta.datacentral: sync to your dev desktop, and run `bb run mons-server-module` .
    - You will also need to run `HootWebsite` from you dev-desktop by running `bb run mons-server` .
    - In Hoot, you will need to add the `<dc-fragment/>` loader wrapper by following the same structure/formats as the other dataset MFES: e.g. `<dc-schema-attributes/>`. ← try searching for that to see how it's imported and displayed on Hoot. (**TODO**: let's setup a Hoot branch where this will already be setup.)

```
Setup mons cookie overrides:
{
  "addedRights": [],
  "removedRights": [],
  "claims": [],
  "urlOverrides": {
    "DataCentralDatasetMicrofrontendsWebsite": "http://dev-dsk-urenamar-1d-f1427c8f.us
    "Hoot:HootUIOpenID": "http://dev-dsk-urenamar-1d-f1427c8f.us-east-1.amazon.com:432
  },
  "translationKeys": [],
  "iamRoleOverrides": {},
  "scopedRightsToken": "",
  "customerId": "",
```

```
  "merchantId": "",
  "marketplaceId": ""
}
```

DatasetMicrofrontend Code Branch

Presentation

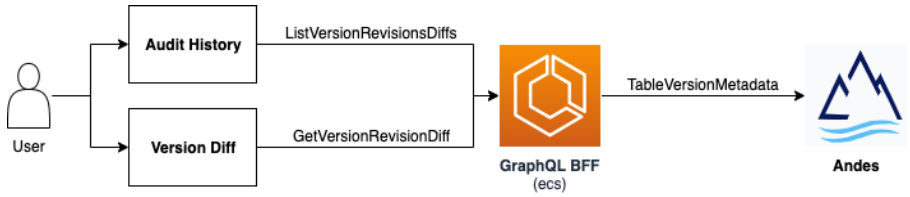
Broadcast: https://broadcast.amazon.com/videos/1012347

1. **Introduction (15 seconds)**: Standard intro – quick, to the point.
2. **Background and User Needs (1 minute)**: Emphasize the problem and support it with user quotes. Can introduce the need for a dynamic approach to handling revisions and versions.
3. **Video demo (2 minutes)**
4. **Technical Innovation and Architecture (1 minute)**:
    a. **On the APIs:** For the hackathon implementation, we have a list API with a paginated interface as well as limited filtering that works by dynamically generating diff logs based on the current Andes table version revisions APIs by comparing each revision with the next.
    b. **In developing our components (25 seconds):**
        i. When building these components, we leveraged DataCentral's MicroFrontEnd architecture to enable them to work seamlessly across DataCentral.
        ii. We've also taken DataCentral's grid solution to the next level by implementing true server-side pagination through a GraphQL data source.
        iii. This approach significantly enhances performance and scalability while introducing a "low javascript code" version of the grid.
    c. **Dark Theme Enhancement (10 seconds)**:
        i. We also innovated by introducing a DataCentral's dark theme.
        ii. We accomplished this by adding an 'alias token' layer to our existing CSS tokens.
5. **Production readiness: Feasibility and Scalability (45 seconds)**:
    a. Components were built in production code package - one merge away from pipelines.
    b. Dark theme is already working on most complex Hoot page which says a lot regarding its' viability.
    c. This dynamic approach will work for metadata persisted in Andes itself, and the functionality will be providing value for the user. However, users will also want visibility into how other aspects of an "Andes" dataset, and currently, the data and audit logs are incompletely and spread out across multiple BDT subsystems like DMO, Tagging, and ORP.
    d. There is also a need to consistently attribute actions to a human actor, and the adoption of Transitive Auth across BDT systems will push us in that direction.
    e. Once the audit data is generated and collected, we can extend the UX presented here with logs from these additional systems, and users will be able to view them ordered by time, and filter by actor, or specific attributes.
6. **Where it can be used (15 seconds):** Edward's slide
7. **What's next (15 seconds)**: Conclude by summarizing the main points, emphasizing the potential for future expansion, such as integrating Transitive Auth for consistent attribution to human actors and extending the UX with logs from additional systems.
8. **Customer Impact and Innovation (45 seconds)**: Connect the technical aspects to user benefits. Discuss how these features provide valuable insights into Andes datasets and future possibilities of extending this functionality to other

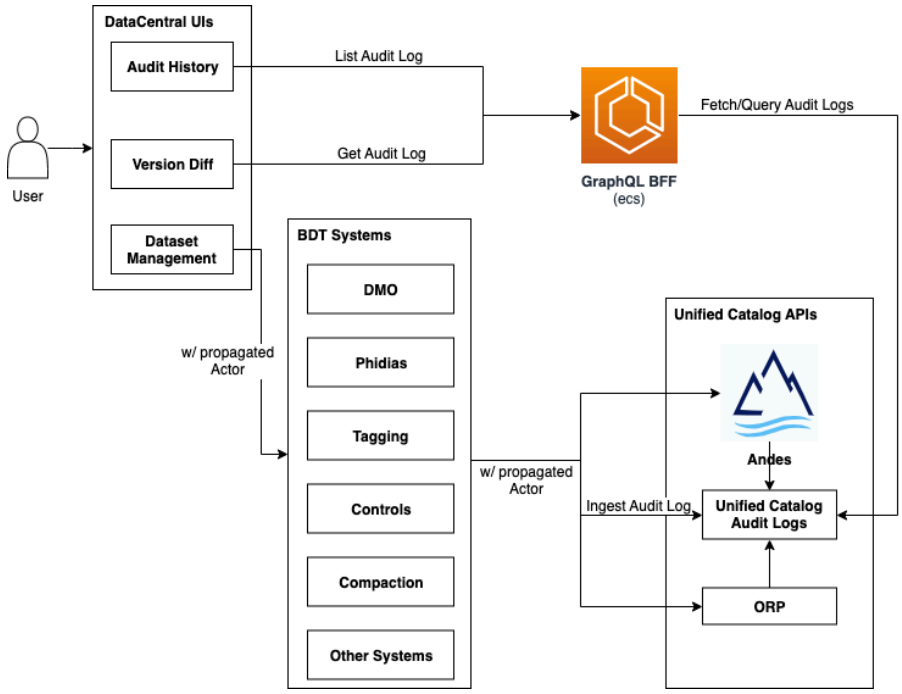
BDT subsystems for a more holistic view. Fundamentally changes the customer mindset and problem domain, opening new avenues of usability that were completely unavailable before, “No way, with this I can completely redo X/Y and it’ll be so much easier/faster/better!” “Oh wow, I could never even consider doing this this before, now it’s so simple”

Architecture

For the API, we have a GraphQL query with paginated interface as well as filtering. The API works by dynamically generating diff logs based on the current Andes table version revisions APIs by comparing each revision with the next.



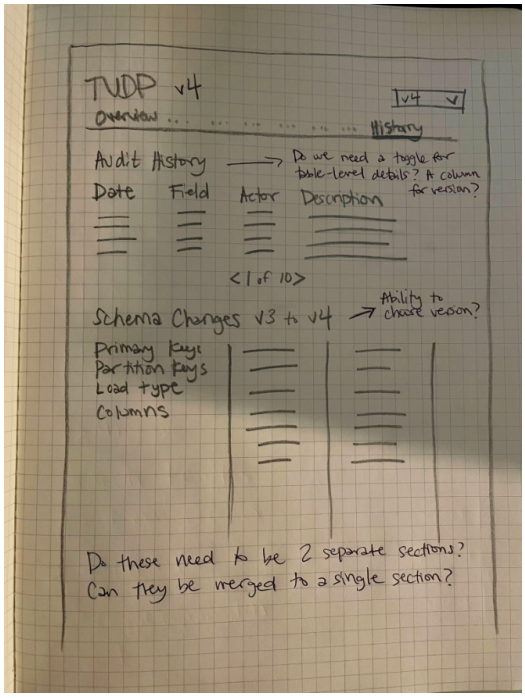
The API approach that we have will work with existing Andes Datasets at the P99 level, with 60 revisions per version, and 10 versions per dataset). However the existing audit data is often missing a link to a human actor, especially when the original user request goes through proxy systems before reaching Andes. This diagram is how a complete audit logging solution might look like, with the actor propagated all the way through.

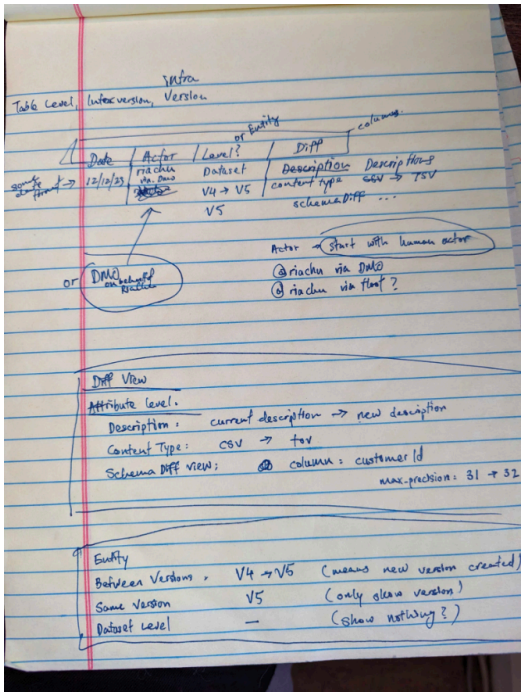


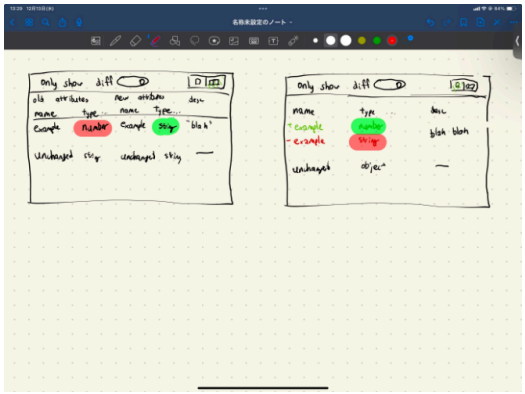
Appendix

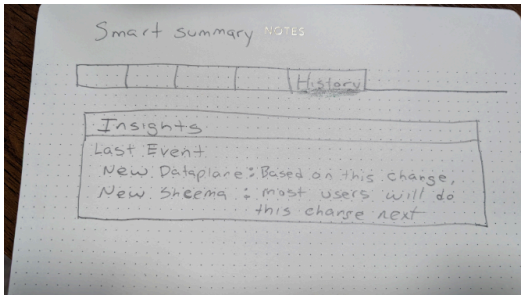
Design studio

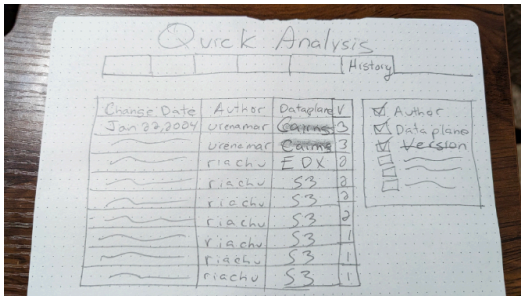
A design studio (parallel design) was conducted on 12/13 to generate ideas for the version history feature.











Project scoring

	A	B
1	**Tech Innovation (0-5)**	
2	Doesn't actually propose building anything, unclear what will actually be built	
3	Used an existing technology in a standard way, e.g. service + backend, standard stuff	
4	Combines existing technology in a novel way, e.g. combining results from two or more systems, using 'off the shelf' libraries and combining them together in interesting ways.	
5	Novel extension of existing technologies, or proposes independent novel technologies, e.g. extends models/architecture of existing services or showcases how novel off the shelf tech can be applied to a customer problem space.	
6	Implements novel extensions to existing internal or external technologies (or implements entirely new technologies) and combines them with existing systems in novel ways.	
7		
8	**Feasibility (Delivery) (0-5)**	
9	Impossible to build as described	
10	Could potentially be built, but undefined as to how (e.g. 'magic model would go here', 'system to do infeasible thing would go here')	
11	Some evidence supports theory it could be built, but no/little attempt at proving function	
12	System could be reasonably delivered, clearly outlined major risks/gaps	
13	Most major risks/gaps solved, model/system performance and scale is proven, needs improvements/extensions before it is ready to launch	
14	System would clearly be able to launch at scale, all fundamental risks are mitigated, missing components are mundane launch tasks (tests, alarms, pipelines, non-essential components, etc)	
15		
16	**Customer Innovation (0-8)**	
17	Customer would not notice	
18	Customer would notice, but not particularly care	
19	Customer would notice, but would have fairly small measurable impact on day to day work	
20	Customers would appreciate the improvement 'hey, that's a nice improvement'	
21	Customers would be impressed, may choose to leave positive feedback ("Wow, this is neat!")	
22	Customers would be delighted, 'Oh thank goodness, this is exactly what I needed!', solves critical customer problems Bonus Round (Above/beyond customer obsession)	
23	Resolves a critical customer ask that they didn't realize could be solved 'Oh wow, I don't even need to have/do 'X' anymore? How did you do that?'	
24	Combines and simultaneously resolves many critical customer concerns, "Wait, you mean it handles registration, ingestion and scaling all at once? Hot dang!"	
25	Fundamentally changes the customer mindset and problem domain, opening new avenues of usability that were completely unavailable before, "No way, with this I can completely redo X/Y and it'll be so much easier/faster/better!" "Oh wow, I could never even consider doing this this before, now it's so simple"	
26		

Final email announcement about the hackathon

Dear Finalists -

We wanted to lowdown on our much awaited Hackathon event start from Monday, 22 Jan to Friday, 26 Jan at The Summit - SEA47: 2122 7th Ave., Seattle, WA 98121, (Rooms 02.300& 02.400). Hackathon committee has prepared an exciting schedule for all of us to enjoy as we work on our ideas. See below for list of activities throughout the week.

	A	B	C	D	E	F	G	H	I
1			**8.30 - 9.30 PST**	**9.30 - 10.00 PST**	**10.00 - 12.30 PST**	**12.30 - 1.30 PST**	**1.30 - 5.30 PST**		**6.30 PST**
2	**Monday**	**Day1**	Breakfast catered	Kickoff by Anas Fattahi Keynote by G2 Krishnamoorthy	Hack	Lunch catered	Hack	Snacks	Dinner catered
3	**Tuesday**	**Day2**	Breakfast catered	Hack	Hack	Lunch catered	Hack	Snacks	Dinner - on your own
4	**Wednesday**	**Day3**	Breakfast catered	Hack	Hack	Lunch - on your own	Hack	Snacks	Dinner - on your own
5	**Thursday**	**Day4**	Breakfast catered	Hack	Hack	Lunch - on your own	Hack	Snacks	Dinner catered @Spheres
6	**Friday**	**Day5**	Breakfast starts at 8.30am PST followed by presentations to judges starting at sharp 9am PST. Winners will be announced at 1pm EST (Lunch will be catered)						

FAQs:

Will I have access to the SEA buildings during the week of Hackathon ?

If you are located in SEA you will have access to all the Seattle buildings. If you are not, you must requestbadge access here to Spheres and the Summit buildings. If you need help or have questions reach out to Mrugesh Gharat or Rohini Kotamreddy

What are the food options

There will be catered Vegetarian, Non-Veg options with kosher/halal/vegan available around the venue and food boxes will be ordered and delivered as needed. Pls message Anne Driscoll if you have any dietary restrictions.

Can we show recorded presentations and answer questions instead of doing a final live presentation

Final presentations to the judges will be done Live, but you will have an option to show a recorded session of your idea.

What is the format of the final presentations before the judges on Friday, 1/26

On Friday 26 Jan, each team will present their final product to the judges for 6 mins followed by 5 mins of Q&A. There will be a setup time of 3 mins for new teams to get on stage to present their idea. https://quip-amazon.com/REGoAkpXE3pE

Who are the judges

Ippokratis Pandis, Molly Brown, Terk Terkowits, Jeff Carter, Jim Schmid and Seema Degwekar

What are the prizes for winners and are they tax deductible (*It will be winners responsibility to declare the gifts on their 2024 tax returns)*

Yes, there are prizes for winners and for the idea with most people's choice votes

- **First place** - Amazon fire televisions for each participant from the winning team **OR** $250 gift certificates
- **Second place** - $50 gift certificates
- **Third place** - $25 gift certificates
- **People's Choice award** - Kindle paper white **OR** $100 gift certificate

We look forward to seeing you all! (Idea owners, pls forward to newly added team members if I missed any)

Thank You,
Hackathon Committee