Bug UFC-135: Fix for Slow Transaction Tables

The following document will go over the design of the fix needed for improving the performance of the transaction tables.

Reproduction Steps

There are two main ways to see this performance issue in action: from the table in the `Adjust Transactions` step of the import process, or from the table on the main `Transactions` page.

Adjust Transactions Step Example

i. Load up a CSV file with > 50 transactions.

ii. Try to change the Target Account (or really, any field).

iii. Notice lag.

Transactions Page Example

i. Switch to `All Transactions` and have > 50 transactions.

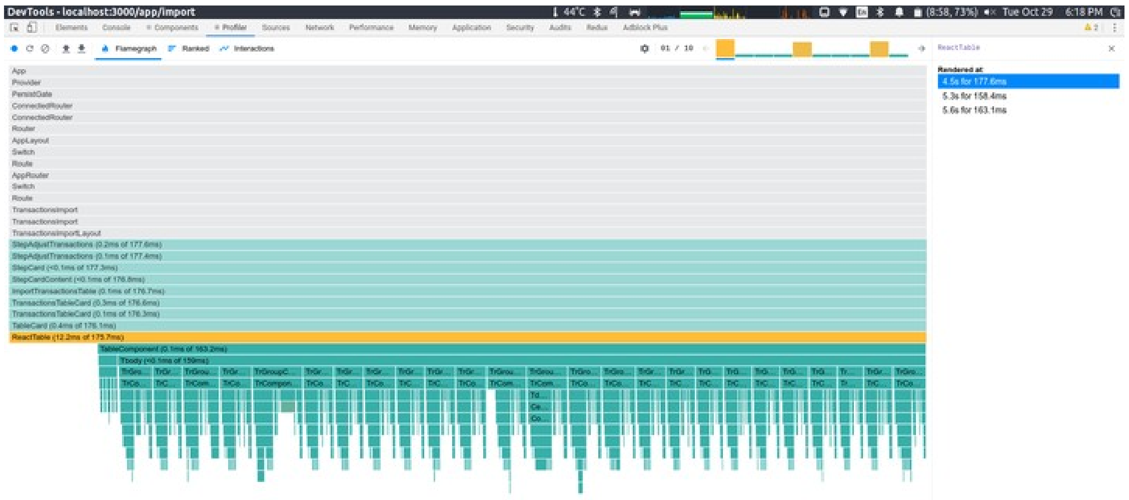
ii. Try to delete a transaction.

iii. Notice the lag between the transaction being removed from the table and the deletion toast appearing.

Bug Details

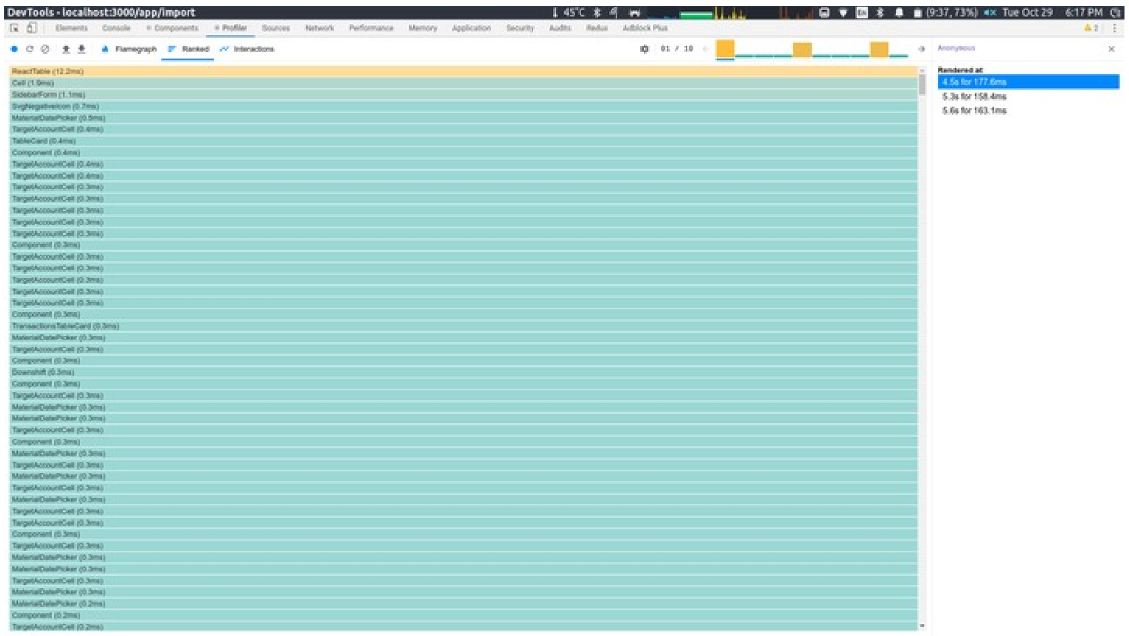
It would appear that anytime anything changes to any of the transactions, the *entire* table re-renders (i.e. the top-level ReactTable component, and then every sub component).

Here's a sample performance profile of changing a target account value in the `AdjustTransactionsTable`:


DevTools - localhost:3000/app/import

App
Provider
PersistGate
ConnectedRouter
ConnectedRouter
Router
AppLayout
Switch
Route
AppRouter
Switch
Route
TransactionsImport
TransactionsImport
TransactionsImportLayout
StepAdjustTransactions (0.2ms of 177.6ms)
StepAdjustTransactions (0.1ms of 177.4ms)
StepCard (<0.1ms of 177.3ms)
StepCardContent (<0.1ms of 176.8ms)
ImportTransactionsTable (0.1ms of 176.7ms)
TransactionsTableCard (0.3ms of 176.6ms)
TransactionsTableCard (0.1ms of 176.3ms)
TableCard (0.4ms of 176.1ms)
ReactTable (12.2ms of 175.7ms)
TableComponent (0.1ms of 163.2ms)
Tbody (<0.1ms of 159ms)

Rendered at:
4.5s for 177.6ms
5.3s for 158.4ms
5.6s for 163.1ms


DevTools - localhost:3000/app/import

ReactTable (12.2ms)
Cell (1.9ms)
SidebarForm (1.1ms)
SvgNegativeIcon (0.7ms)
MaterialDatePicker (0.5ms)
TargetAccountCell (0.4ms)
TableCard (0.4ms)
Component (0.4ms)
TargetAccountCell (0.4ms)
TargetAccountCell (0.4ms)
TargetAccountCell (0.3ms)
TargetAccountCell (0.3ms)
TargetAccountCell (0.3ms)
TargetAccountCell (0.3ms)
TargetAccountCell (0.3ms)
Component (0.3ms)
TargetAccountCell (0.3ms)
TargetAccountCell (0.3ms)
TargetAccountCell (0.3ms)
TargetAccountCell (0.3ms)
TargetAccountCell (0.3ms)
Component (0.3ms)
TransactionsTableCard (0.3ms)
MaterialDatePicker (0.3ms)
TargetAccountCell (0.3ms)
Component (0.3ms)
Downshift (0.3ms)
Component (0.3ms)
TargetAccountCell (0.3ms)
MaterialDatePicker (0.3ms)
MaterialDatePicker (0.3ms)
TargetAccountCell (0.3ms)
Component (0.3ms)
MaterialDatePicker (0.3ms)
TargetAccountCell (0.3ms)
MaterialDatePicker (0.3ms)
TargetAccountCell (0.3ms)
MaterialDatePicker (0.3ms)
TargetAccountCell (0.3ms)
TargetAccountCell (0.3ms)
Component (0.3ms)
TargetAccountCell (0.3ms)
MaterialDatePicker (0.3ms)
MaterialDatePicker (0.3ms)
TargetAccountCell (0.3ms)
MaterialDatePicker (0.3ms)
MaterialDatePicker (0.2ms)
Component (0.2ms)
TargetAccountCell (0.2ms)

Rendered at:
4.5s for 177.6ms
5.3s for 158.4ms
5.6s for 163.1ms

And here's the saved profiler information: transactions-table-profile.json

As can be seen, it appears that just the sheer number of components being re-rendered is the problem; no single component takes an exceptionally long time (except for the top-level `ReactTable` component). The total render is roughly ~150ms per change when the table was loaded up with a mere 25 transactions; totally unreasonable when trying to change a text input, for example.

Potential Fixes

The following are a number of different options for fixing this bug.

Janky Fix Details

Given that the below 'true' fix details are starting to become much more involved than I expected, here's the janky fix:

- Turn on pagination.

Can't have bad performance with lots of transactions if there aren't lots of transactions

'Kinda Janky' Fix Brainstorming

The following are a stream of thought discussion of what would be needed to fix this bug using the existing v6 of `react-table`.

- Modify the table to connect straight to the store and grab only IDs.
    - But then this makes both sorting *and* filtering more annoying, cause both operations have to be done at the Redux level (see below for the discussions on both).
- Modify all of the cells to be connected components that just take in transaction IDs, instead of values.
- Sorting will have to be done manually (using the `onSortedChange` callback) → will probably have to have some redux state for storing the sort state of the table, since the `onSortedChange` callback will have to dispatch a 'sort' action, and then `mapStateToProps` can sort the IDs; don't forget to reset this state whenever the user changes pages.
    - Actually, no, this is a stupid idea. Just add a piece of local state that holds the sort direction/column and performs the sorting inside the component.
        - Actually, THAT is a stupid idea, cause the table would still need to be passed the whole transactions set to do the sorting locally... brainfart.
            - Wait, no, it *should* be fine if the component that wraps `ReactTable` receives all of the transactions, because we just need to cutoff the render at the `ReactTable` level.
            - The sorting will just need to be memoized using `useMemo` so that when `ReactTable` receives the list of IDs, it only gets a different set when the order *actually* changes (specifically, so that `TableCard` gets the memoized IDs -- everything above `TableCard` are the components handling the filtering/sorting).
                - But wait, the memo will break whenever the transaction data changes...
                - So this memo is useless...

    - `TableCard` will need to be wrapped in `React.memo` since it isn't a connected component.
        - Need a custom equality checker to do a 'deep' equality check of the list of IDs being passed as data, since we can't count on it being memoized properly due to the above.
        - A custom equality checker for 'deep' checking an array is way, *way*, ***way*** cheaper than doing a whole table render.
  - In order for the `onSortedChange` callback to work, I think the `manual` prop has to be set to `true`. Note that this, in the future, means that pagination will also have to be calculated manually (at least, that's what I believe the docs are saying).
- Everything that applies to sorting applies to filtering.

Problems with this Fix

- Ah shit, kinda hit a problem... Connecting the cell components seems easy, but what if we want to use a different source of transactions in the connect? i.e. the set of transactions being imported (stored in the `transactionsImport` slice) or the existing transactions set (stored in the `transactions` slice).
  - Really hacky... could pass a `getTransaction` selector *as a prop* to the table, then pass that selector down to all the cells, and have the cells use that to get the transaction information.
  - Wait wait wait... what if, and hang on there, what if instead of passing an actual (redux) selector into the table as a prop, what if the table created it's *own* selector based on the transaction data and then passed *that* down to the cells.
    - Would need a `memoize` function (could steal the `defaultMemoize` from `reselect`)...
    - Wait no, this wouldn't work because the function would be re-generated whenever the transaction data changed, which would cause a render anyways... nvm then

'Proper' Fix Details

It would seem like the 'true' fix would be to completely rewrite the table components to use the new v7 of the `react-table` library.

v7 seems to be a complete rewrite of the `react-table` library itself, transitioning the library from a fairly nice table component, to a completely headless (i.e. UI-less) library that uses hooks. That is, an API very similar to `downshift`, except using hooks instead of a render prop.

A rewrite of the table components at this level would also enable us much greater control of the table.

Finalized Fix Details Design

This is somewhat finalized design of what should be implemented to fix this bug.

- Create a connected version for each of the cells used by `TransactionsTableCard` that:
  - Takes in a `transactionId` prop.
  - Takes in a `getTransaction` 'selector' prop.
  - `mapStateToProps` pulls out the above props from `ownProps` and extracts the necessary data for the cell.
    - If no `getTransaction` prop provided, default to the selector for the `transactions` slice.
- Add a `getTransaction` 'selector' prop to `TransactionsTableCard`.
  - Pass it down to `generateColumns` and then pass it down to all of the cells.
- Wrap `TableCard` in `React.memo` with a custom equality function for checking that the data (list of IDs) are the same.
- Add state to `TransactionsTableCard`for holding the sorting state `(column, direction)`.
  - Remove the custom `sortMethod` from all of the columns in `generateColumns`.
  - Add a `manual` prop to `TableCard` that gets passed down to `ReactTable`.
    - Disable the `defaultSorted` prop when `manual` is true.
  - Add a callback to `TableCard` for `onSortedChange` to be able to propagate the sort info back up to `TransactionsTableCard`.
- Modify `TransactionsTableCard` to have a (memoized) pipeline for taking the `transactions` data prop, filtering it, sorting it, and mapping it to IDs for passing down to the `TableCard`.

Postmortem Fix Details

This is what actually was implemented to fix this bug. These are ordered to correspond with their related files as they would be found in the pull request.

- Increased the max allowed request payload on the backend Feathers API to 10 MB to allow importing many hundreds or thousands of transactions at once.
- Upgraded various packages:
  - Upgraded `react` and `react-dom` from v16.9.0/v16.8.1 to v16.11.0.
    - Somehow, `react-dom` was working fine as a different version from `react`.
    - This upgrade caused a number of the other package components to start throwing warnings because of the deprecated lifecycle methods -- that's why they also had to be upgraded.
    - The React upgrades also seemed to bring some subtle performance improvements, but those might have been more attributed to the upgrades in the other packages.
  - Upgraded `react-redux` from v5.1.1 to v7.1.3.
    - The jump over v6 was necessary since that whole release was a complete mess performance-wise.
    - Upgrading to v7 also seemed to fix some subtle `connect` related performance problems that I was seeing.
      - Most notable, when the `ImportTransactionsTable` was loaded with hundreds of transactions, adding a new Account object would cause each individual `TransactionTargetAccountCell` to render *separately*, instead of all at once (i.e. they each had their own commit in the React profiling tools, instead of just one). The result was that any change to the `accounts` slice would cause a huge lock-up since each component rendered separately; particularly when doing the final import, since the `accounts` slice would change as a result of adding all of the transactions to all of the accounts.
      - I don't know *why* this was happening, but since it was fixed with the `react-redux` upgrade, I can only assume it was a bug with the package instead of our code.
  - Upgraded `connected-react-router` from v5.0.1 to v6.5.2.
    - Fixed unsafe lifecycle method warnings.
  - Upgraded `react-router-dom` from v4.3.1 to v5.1.2.
    - Fixed unsafe lifecycle method warnings.

    - Upgraded `react-table` from v6.9.2 to v6.10.3.
        - Fixed unsafe lifecycle method warnings.
    - Removed the `react-scrollspy` package and forked it into the repo to fix it.
        - Had to modify it to use `UNSAFE_componentWillReceiveProps` since no version had been published to address it.
    - Had to bring in the `prop-types` package so that we could monkey-patch the prop-types of `ReactTable` so that we could pass a `React.memo` version of `TrGroupComponent`; seems like result of `React.memo` isn't compatible with the existing prop-types that `ReactTable` used (see facebook/react#15752).
- Modified the `Dropdown` component to conditionally render instead of simply being hidden using CSS.
    - Performance optimization so that its DOM elements don't have to be rendered all the time; relevant when importing large amounts of transactions since we don't want the dropdown for every `Type` cell to be 'rendered' but hidden.
- Modified the `ImportTransactionsTable` to connect directly to get the dispatch function for `onTransactionChange`, just so that the `Step` component didn't have to do it.
    - Note that it's not connected to grab the transactions directly, since those are different depending on the step. However, since the `onTransactionChange` only corresponds to the adjustment step, we can connect the component directly.
    - Obviously, this might change with future requirements.
- Modified the `MaterialMoneyInput` to support being wrapped in the new `changeOnBlur` HOC. This will be explained later.
- `TableCard` had a number of modifications to optimize performance:
    1. The table now supports using ID data. That is, where the `data` prop is just an array of string IDs.
    2. This means that we can now add keys to the `TrGroup` components with the IDs. This means that each row of the table now has a proper stable identity to React and React can optimize accordingly.
    3. Using the ID data as a key on `TrGroup` also means that we can memoize the `TrGroupComponent` so that it (and it's many children) only re-render when the ID changes. Since IDs don't change, this means that the `TrGroupComponent` only has to render once: the first render. This particularly important when using the table for editing (ala the `ImportTransactionsTable`) or when

sorting, since changing Transaction values only affects the individual cell, and sorting just re-arranges the order of the rows, instead of re-rendering the whole table.

4. As a result of making the table support ID data, now the table has to use `manual` mode -- that is, sorting, filtering, and pagination have to be handled by us instead of by `react-table`. This makes things a bit more cumbersome to implement/maintain, but since we were already doing filtering ourselves, adding sorting to that isn't that much more work (we'll see more of this in `TransactionsTableCard`).

5. The `ActionsCell` also got wrapped in `React.memo`, again to prevent unnecessary renders.

6. In other changes, the `onRowClick` handler now takes in the whole data object for the row, instead of trying to pass along just the `id` property of the row object. This is to account for the fact that the data can now be just IDs.

    - The `AccountsTableCard` has been updated accordingly.

7. `TransactionsTableCard` was where the bulk was done. This is gonna be a long one...

8. The most important change is that now, while the `TransactionsTableCard` itself still receives a full list of transactions as data (i.e. its API hasn't changed in that regard), it now passes only IDs down to `TableCard`. This is important for a couple of reasons:

    - This means that the individual cells only receive an ID as a prop. This means that they have to connect directly to the store to get the data that they need to render (e.g. date, description, amount, etc).

    - Since the cells connect directly to the store for their data, and only receive an ID as a prop, they only have to render whenever the particular piece of their data of their transaction changes; i.e. no longer do all cells in the table have to render whenever any piece of transaction data changes.

    - As a result, there is a *massive* performance improvement when editing transactions. No longer is there a multiple second delay when typing in the `description` field; everything is now (basically) instant when editing (there is a further performance optimization for the `description` and `amount` fields, but that is explained below).

    - Passing IDs also, like mentioned previously, allows us to memoize the `TrGroupComponent` so that *nothing* re-renders unless the IDs change.

9. Using IDs also means that we can memoize at the top-level table level (i.e. at the `TableCard` level).

    - As will be expanded on in a second, since the transformation process for turning the raw transaction data into an array of IDs isn't immutable (i.e. isn't memoized to the point where the same transaction data just returns the same instance of an array of IDs), we have to do a 'deep' comparison of the ID arrays as part of `React.memo`'s `areEqual` function. That is, instead of a shallow reference equality check, we actually check that all of the IDs are the same between old and new props.

        - While this is obviously less performant than a shallow equality check, it is *orders of magnitude* more performant than letting the table actually re-render when it shouldn't. So, the tradeoff is more than worth it.

10. As was noted previously, using IDs as data means that we can't rely on `ReactTable`'s built in features for sorting/fitering/pagination (not that we're doing pagination yet). As such, there is now a new `useTransformTransaction` hook for the table that handles doing all of these things (and, obviously, transforming the transactions into the IDs that will be fed to the `TableCard`).

    - This hook alone deserves its own discussion, but I think the comment in the code explains it adaquately.

        - tl;dr we defer rendering the rows so that other things (like clicking on filters or sorting headers) can finish rendering first and give the user better *perceived* performance.

    - Obviously, this also means that we need to keep track of sorting state in local state.

11. Since the table needs to operate using IDs, it now takes in an optional `getTransactionSelector` prop.

    - This prop is, quite literally, a Redux selector of the form `(id) => (state) => transaction`.

    - It gets passed down directly to all of the table cells to enable them to find the correct transaction data.

    - Since we have multiple places where transaction data can be stored (i.e. the `transactions` and `transactionsImport` slices), the cells need to know where to find their transaction data; this prop is what enables that.

        - If not passed, the cells default to just using the `transactions` slice.

    12. Also, the `TransactionsFilterControls` and `TransactionTableCardTitle` components were moved to their own files to clean up the main component file.

- The `columns` for the `TransactionsTableCard` also deserve their own discussion:
    - All columns now make use of custom connected cells that only receive `id` as a prop.
    - All of the cells have been wrapped with a wrapper that makes sure that the derived `onEditingChange` callback is memoized to eliminate extraneous renders.
    - The adding of the type class to the amount cell for coloring was moved from the columns declaration and into the `TransactionAmountCell` itself, since obviously the columns no longer have the transaction information necessary to determine the type (this seemed to have been broken for a while...).
- All cells have now been customized just for the `TransactionsTableCard` so that they take in `id` as a prop and connect to the store to grab the necessary transaction data.
    - The `Description` and `Amount` cells are extra noteworthy because they use an optimization technique (the `changeOnBlur` HOC) whereby the actual text in the input is stored as local state to the component, and the changes are only propagated to the Redux store on blur (hence, `changeOnBlur`).
        - This makes it so that typing in these inputs is basically as fast as possible.
    - All other cells emit changes directly to the store, since they don't require typing.
    - The `Target Account` cell is also noteworthy because it had some selector optimizations done to prevent unecessary re-renders.
- As a summary of the main things being memoized, take note that we're memoizing at three different levels: at the top-level table level (i.e. `TableCard`), at the row level (i.e. `TrGroupComponent`), and at the cell level (i.e. `Transaction...Cell`).
    - Table level memoization prevents anything from re-rendering unless the order or number of transactions changes.
    - Row level memoization prevents rows from re-rendering, even if the order of number changes (i.e. their position just changes, but their children don't re-render).
    - Cell level memoization prevents cells from re-rendering at all, unless the data for their transaction changes.

- In other optimizations:
  - The `EditAccountForm` was fixed to not re-render anytime anything happened, by fixing an empty object into a constant.
  - The `TransactionAutocompleteInput` had some regular memoization added for generating the options; otherwise it too would render too often.
    - It also had to have some ref forwarding reworked due to the React upgrades.
  - The `ImportStepper` also some regular memoization added to fix unecessary re-renders.

Future Optimizations

As was mentioned previously, rebuilding the whole table component from scratch using v7 of `react-table` is probably the most optimal solution, at this point.

Additionally, I had the idea of "what if we progressively rendered the list of transactions in chunks?". Not like how `react-window` or `react-virtualized` does it, with a scrolling list, but where the whole list *eventually* gets rendered, just that it renders it chunks of, say, 100 items at a time, so that the user can see the results and start interacting faster.

I kind of put together a proof of concept of this here:

TransactionsTableCard_hooks_withProgressiveRendering.js