Overview of `LMDB` datastore behavior changes

In Gatsby 3 we introduced a new data store called `LMDB` (enabled with `LMDB_STORE` and/or `PARALLEL_QUERY_RUNNING` flags). In Gatsby 4 it became the default data store. It allows Gatsby to execute data layer related processing outside of the main build process and enables Gatsby to run queries in multiple processes as well as support additional rendering strategies (DSG and SSR).

In a lot of cases that change is completely invisible to users, but there are cases where things behave differently.

Direct node mutations in various API lifecycles are not persisted anymore. In previous Gatsby versions it did work because source of truth for the data layer was directly in Node.js memory, so mutating node was in fact mutating source of truth. Now Gatsby edits data it receives from the database, so unless it explicitly upserts data to this database after edits, those edits will not be persisted (even for same the same build).

Common errors when doing swap to `LMDB` will be that some fields don't exist anymore or are `null` / `undefined` when trying to execute GraphQL queries.

Diagnostic mode

Gatsby (starting with version 4.4) can detect those node mutations. Unfortunately it adds measurable overhead so it isn't enabled by default. You can opt into it when you see data related issues (particularly when you didn't have this issue before using `LMDB`). To enable diagnostic mode:

- Use truthy environment variable `GATSBY_DETECT_NODE_MUTATIONS`:

  ```
  GATSBY_DETECT_NODE_MUTATIONS=1 gatsby build
  ```

- Or use `DETECT_NODE_MUTATIONS` config flag:

  ```
  module.exports = {
    flags: {
      DETECT_NODE_MUTATIONS: true,
    },
  }
  ```

Example diagnostic message you might see:

```
warn Node mutation detected

File <rootDir>/plugins/transform/gatsby-node.js:4:20
  2 |   if (node.internal.type === `Test`) {
  3 |     // console.log(`[Mutating node in onCreateNode]`)
> 4 |     node.nested.a2 = `edited`
    |                    ^
  5 |   }
  6 | }
  7 |

Stack trace:
    at Object.exports.onCreateNode (<rootDir>/plugins/transform/gatsby-node.js:4:20)
    at runAPI (<rootDir>/node_modules/gatsby/src/utils/api-runner-node.js:462:22)
```

```
    at Promise.catch.decorateEvent.pluginName
(<rootDir>/node_modules/gatsby/src/utils/api-runner-node.js:613:13)
    at Promise._execute
    ... Rest of Stacktrace
```

It will point you to the code that is trying to mutate nodes. Note that it might also point to installed plugins in which case you should notify the plugin maintainer about it.

Please Note: Be sure to stop using this mode once you find and handle all problematic code paths as it will decrease performance.

Migration

Mutating a node in `onCreateNode`

Instead of mutating nodes directly, `createNodeField` action should be used instead. This way Gatsby will update the source of truth (to actually update the node in the datastore). `createNodeField` will create that additional field under `fields.fieldName`. If you want to preserve schema shape, so that additional field is on the root of a node, you can use schema customization.

```
const { createRemoteFileNode } = require(`gatsby-source-filesystem`)

exports.onCreateNode = async ({
  node, // the node that was just created
-  actions: { createNode },
+  actions: { createNode, createNodeField },
  createNodeId,
  getCache,
}) => {
  if (node.internal.type === `SomeNodeType`) {
    const fileNode = await createRemoteFileNode({
      // the url of the remote image to generate a node for
      url: node.imgUrl,
      parentNodeId: node.id,
      createNode,
      createNodeId,
      getCache,
    })

    if (fileNode) {
-      node.localFile___NODE = fileNode.id
+      createNodeField({ node, name: 'localFile', value: fileNode.id })
    }
  }
}
+
+exports.createSchemaCustomization = ({ actions }) => {
+  const { createTypes } = actions
+
+  createTypes(`
+    type SomeNodeType implements Node {
+      localFile: File @link(from: "fields.localFile")
```

```
+      }
+    `)
+}
```