Gatsby keeps a record of used nodes for each query result. This makes it possible to cache and reuse results from previous runs if used nodes didn't change and, conversely, is used to determine which query results are stale and need to be rerun.

How dependencies are recorded

`CREATE_COMPONENT_DEPENDENCY` action and `createPageAction` action creator

The internal `CREATE_COMPONENT_DEPENDENCY` action handles the recording of Page -> Node dependencies. It takes the `path` (page path for page queries or internal id for static queries), and either a `nodeId`, or `connection`.

Passing `nodeId` tells Gatsby that the page depends specifically on this node. So, if the node is changed, then the page's query needs to be re-executed.

`connection` is a Type string. E.g. `MarkdownRemark`, or `File`. Calling `createPageDependency` with a `path` and a `connection` tells Gatsby that this page depends on all nodes of this type. Therefore if any node of this type changes (e.g. a change to a markdown node), then this page must be rebuilt. Using connection fields (e.g. `allMarkdownRemark`) is one of the cases when this variant is used.

`CREATE_COMPONENT_DEPENDENCY` action is conditionally dispatched by the internal `createPageAction` action creator. Action creator checks if we already have given dependencies stored to avoid emitting no-op actions. `createPageAction` is a low level internal API that is then used by higher level APIs.

Higher level abstractions

Node Model

Node Model is an API used in GraphQL resolvers to retrieve nodes from the data store. It's used internally in resolvers provided by Gatsby core and it can be used in resolvers provided by plugins via `context.nodeModel`. It calls `createPageAction` under the hood because Node Model is aware of the path of the query as well as the nodes being retrieved.

`getNodeAndSavePathDependency` helper

`getNodeAndSavePathDependency` is a convenience wrapper around `getNode` and `createPageDependency`. It is not used internally. It's a legacy API for plugins to record data dependencies and is equivalent to `nodeModel.getNodeById`. The Node Model variant should be used instead as its API is less error-prone. (Node Model is `path` aware and doesn't require you to pass it.)

How dependencies are stored

Page -> Node dependencies are tracked via the `componentDataDependencies` Redux namespace.

```
{
  nodes: { ... }, // mapping nodeId -> pages
  connections: { ... } // mapping of type names -> pages
}
```

Nodes is a map of nodeID to the set of pages that depend on that node. E.g

```
// state.componentDataDependencies.nodes
{
  `ID of Some MarkdownRemark node`: [
    `blogs/my-blog1`,
    `blogs/my-blog2`
  ],
  `otherId`: [ `more pages`, ...].
  ...
}
```

Connections is a map of type name to the set of pages that depend on that type. e.g

```
// state.componentDataDependencies.connections
{
  `MarkdownRemark`: [
    `blogs/my-blog1`,
    `blogs/my-blog2`
  ],
  `File`: [ `more pages`, ... ],
  ...
}
```

How dependency information is used

Page -> Node dependencies are used entirely during query execution to figure out which nodes are "dirty" and to figure out which queries don't have any dependencies yet. "Dirty" nodes are used to determine which query results are stale and need to be re-executed. Finding queries without dependencies is used as a heuristic to determine which queries haven't run yet and therefore need to run.