LiteDB

Collections

Documents are stored and organized in collections. `LiteCollection` is a generic class that is used to manage collections in LiteDB. Each collection must have a unique name:

- Contains only letters, numbers and `_`
- Collection names are **case insensitive**
- Collection names starting with `_` are reserved for internal storage use
- Collection names starting with `$` are reserved for internal system/virtual collections

The total size of all the collections names in a database is limited to 8000 bytes. If you plan to have many collections in your database, make sure to use short names for your collections. For example, if collection names are about 10 bytes in length, you can have ~800 collection in the database.

Collections are auto created on first `Insert` or `EnsureIndex` operation. Running a query, delete or update on a document in a non existing collection does not create one.

`LiteCollection<T>` is a generic class that can be used with `<T>` as `BsonDocument` for schema-less documents. Internally LiteDB converts `T` to `BsonDocument` and all operations use

the this generic document.

In this example, both code snippets produce the same results.

```
// Typed collection
using(var db = new LiteDatabase("mydb.db"))
{
    // Get collection instance
    var col = db.GetCollection<Customer>("customer");

    // Insert document to collection - if collection does not exist, it is creat
    col.Insert(new Customer { Id = 1, Name = "John Doe" });

    // Create an index over the Field name (if it doesn't exist)
    col.EnsureIndex(x => x.Name);

    // Now, search for your document
    var customer = col.FindOne(x => x.Name == "john doe");
}

// Untyped collection (T is BsonDocument)
using(var db = new LiteDatabase("mydb.db"))
{
    // Get collection instance
    var col = db.GetCollection("customer");

    // Insert document to collection - if collection does not exist, it is creat
    col.Insert(new BsonDocument{ ["_id"] = 1, ["Name"] = "John Doe" });

    // Create an index over the Field name (if it doesn't exist)
    col.EnsureIndex("Name");

    // Now, search for your document
    var customer = col.FindOne("$.Name = 'john doe'");
}
```

System Collections

System collections are special collections that provide information about the datafile. All system collections start with `$`. All system collections, with the exception of `$file`, are read-only.

Collection	Description

Collection	Description
$cols	Lists all collections in the datafile, including the system collections.
$database	Shows general info about the datafile.
$indexes	Lists all indexes in the datafile.
$sequences	Lists all the sequences in the datafile.
$transactions	Lists all the open transactions in the datafile.
$snapshots	Lists all existing snapshots.
$open_cursors	Lists all the open cursors in the datafile.
$dump(pageID)	Lists advanced info about the desired page. If no pageID is provided, lists all the pages.
$page_list(pageID)	Lists basic info about the desired page. If no pageID is provided, lists all the pages.
$query(subquery)	Takes a query as string and returns the result of the query. Can be used for simulating subqueries. **Experimental**.
$file(path)	See below.

$file

The `$file` system collection can be used to read from and write to external files.

- `SELECT $ INTO $FILE('customers.json') FROM Customers` dumps the entire content from the collection `Customers` into a JSON file.
- `SELECT $ FROM $FILE('customers.json')` reads the entire content from the JSON file.

There is also limited support for CSV files. Only basic data types are supported and, on writing, the schema of the first document returned by the query will define the schema of the entire CSV file, with additional fields being ignored.

- `SELECT $ INTO $FILE('customers.csv') FROM Customers` dumps the entire content from the collection `Customers` into a CSV file.

- `SELECT $ FROM $FILE('customers.csv')` reads the entire content from the CSV file.

The single parameter can be:

- `$file("filename.json|csv")` : Support filename only with file extension as format
- `$file({ options })` : Support all options

```
{
    filename: "string",
    format: "json|csv",
    enconding: "utf-8",
    overwritten: false,
    // JSON only
    ident: 4,
    pretty: false
    // CSV only
    delimiter: ",",
    header: true, // write CSV header columns
    header: [] // define header columns names when read
}
```

Made with ♥ by LiteDB team - @mbdavid - MIT License