Couchbase NoSQL Developer Workshop

ラボハンドブック

ラボ 4: N1QL オペレーション

手順

このラボの目的は、特定の顧客（つまり、ログインしているユーザー）の、注文 (新規ないし保留中の注文または全ての注文のいずれか)を取得するロジックを作成することです。

ドキュメント

- エイリアシング
- 日付関数
- Query パラメータ化
- メタデータ関数

ステップ 1: API にロジックを追加する

注文の取得

API リポジトリ ディレクトリで *repository.js* ファイルを開きます(付録の API プロジェクト構造を参照)。 *getOrders( )*メソッドを検索します。 顧客の注文ドキュメントを取得するために必要なロジックを追加して、*getOrders*() メソッドを編集します。

考慮点がいくつかあります。

1. ドキュメント内にどんなフィールドがあるか。
2. N1QL でのエイリアス利用。
3. N1QL 日付形式での作業
    a. エポックタイムスタンプから日時(およびその逆)に取得する方法?
4. クエリパラメータ化

getOrders() 入力:

- 顧客 ID: 整数
- コールバック

getOrders() 出力:

- エラー オブジェクト (該当する場合)
- 注文ドキュメントのサブセット:注文ドキュメントのサンプルについては、付録を参照してください。
    - 必要なプロパティ:
        - 注文ドキュメント ID
            - ヒント: ドキュメントのメタデータ関数を参照してください。
            - META() orderStatus を参照してください。
        - shippedTo というエイリアスを持つ shippingInfo.name
        - grandTotal
        - lineItems
        - orderDate, 人間が判読可能な日付に変換され、同じ名前(orderDate)のエイリアスを持つ
            - ヒント: ドキュメントの日付関数を参照してください。

*getOrders()*メソッドの実装について、以下のコード スニペットを参照ください。
注意: NOP コード行を、コメントアウトするか、削除します。

```
0:   async getOrders(customerId) {
1:     try {
2:       /**
3:        * Lab 4:  N1QL operations
4:        *  1. Get orders for customerId
5:        *     - WHERE order.orderStatus != 'created'
6:        *     - Document properties needed (more can be provided):
7:        *         id,
8:        *         orderStatus,
9:        *         shippingInfo.name aliased as shippedTo,
10:       *         grandTotal,
11:       *         lineItems,
12:       *         orderDate (hint use MILLIS_TO_STR())
13:       *
14:       */
15:      let n1qlQuery = `
16:        SELECT
17:          META(o).id,
18:          o.orderStatus,
19:          o.shippingInfo.name AS shippedTo,
20:          o.grandTotal,
21:          o.lineItems,
22:          MILLIS_TO_STR(o.orderDate) AS orderDate
23:        FROM \`${this.bucketName}\` o
24:        WHERE o.doc.type = 'order' AND o.custId=$1
25:        ORDER BY o.orderDate DESC NULLS FIRST`;
26:
27:      let options = {
28:        parameters: [customerId],
29:      };
30:
31:      //TODO:  prepared queries
32:      // if (adhoc) {
33:      //   options.adhoc = true;
34:      // }
35:
36:      let qResult = await this.cluster.query(n1qlQuery, options);
37:      if (!qResult.rows || qResult.rows.length == 0) {
38:        return { orders: null, error: null };
39:      }
40:
41:      return { orders: qResult.rows, error: null };
42:    } catch (err) {
43:      //Optional - add business logic to handle error types
44:      outputMessage(err, "repository.js:getOrders() - error:");
45:      return { orders: null, error: err };
46:    }
47:  }
```

コードに関する注意事項:

- async/wait 構文を使用します。
- 15-25 行: N1QL クエリ文字列。 N1QL では、次の機能を利用します。
    - エイリアス化 (例: "o.shippingInfo.name shippedTo"など)

    - 注文日付エポックタイムスタンプを人間が読み取り可能な日付に変換 (例:MILLIS_TO_STR())
    - 位置パラメータ (例: o.custId = $1)
    - クエリ結果の順序付け
- 27-29 行:クエリオプションオブジェクトの作成
- 36 行: N1QL 操作は、クラスター レベルで実行されます。(SDK 3.x 以降の仕様)
    - クエリ操作パラメータ:
        - N1QL クエリ文字列
        - クエリオプション ob の検索
- 37-41 行: クエリ結果行を返すだけで、生の行情報を注文として返すことができます。
- *outputMessage()*: コンソールに情報を簡単に出力するためのヘルパーメソッドは、/library ディレクトリにあります (付録の API プロジェクト構造を参照)。
- try/catch: err オブジェクトの処理は、意図的に汎用的な方法で行われています。 ラボ参加者は、エラーを処理するさまざまな方法をテストするために、目的に応じて自由にロジックを追加できます。

完了したら、*repository.js* ファイルが保存されていることを確認します。 API *Docker* コンテナは API の作業ディレクトリにマップされるため、API コードに対して行われたすべての更新はコンテナに反映される必要があります。 コンテナの状態の詳細については、*docker logs api* コマンドを使用します。 コードが保存されると、顧客の以前の注文 (新規/未決注文ではない) を取得する機能は、Web UI 内でアクティブにする必要があります。
次の手順に従って、*getOrders()*ロジックを確認します。

注 ***

SwaggerUI ページを使用してロジックをテストする場合は、承認が必要な *saveOrder()* を使用する場合は、付録に記載されている承認手順に従ってください。

1. http://localhost:8080 に移動
2. ログインしていない場合は、ログインします。
3. 右上隅にある*[Hello {名}*] をクリックし、ドロップダウン メニューの[*Order*] をクリックし、Web UI が *[Order]*ページにリダイレクトされます。
4. *[Order]* ページには、ログインしている顧客に関連付けられている以前の注文が表示されます。

新規/保留中の注文の取得

API リポジトリ ディレクトリで *repository.js* ファイルを開きます (付録の API のプロジェクト構造を参照)。 *getNewOrder* メソッドを検索します。 最新の新規/保留中の注文ドキュメントを取得するために必要なロジックを追加して *getNewOrder*() メソッドを編集します。

考慮する点がいくつかあります。

1. ドキュメント内にどんなフィールドがあるか。
2. クエリのパラメータ化

getOrders() 入力:

- 顧客 ID: 整数
- コールバック

getOrders() 出力:

- エラー オブジェクト (該当する場合)
- 注文ドキュメントのサブセット:注文ドキュメントのサンプルについては、付録を参照してください。
    - 必要なプロパティ:
        - ドキュメント, custId, 注文状況 ,注文日付, 請求情報 , 出荷情報, 配送 ,税金, 品目, 総計,注文番号, _id

*getNewOrder()*メソッドの実装については、以下のコード スニペットを参照してください。 これは、または同様のソリューションを使用して、*getNewOrder()*メソッド ロジックを実装できます。

注意: NOP コード行を、コメントアウトするか、削除します。

```
1:  async getNewOrder(customerId) {
2:    try {
3:      /**
4:       * Lab 4:  N1QL operations
5:       *  1. Get latest order for customerId
6:       *      - WHERE order.orderStatus = 'created'
7:       *      - Document properties needed (more can be provided):
8:       *          doc, custId, orderStatus,
9:       *          billingInfo, shippingInfo, shippingTotal,
10:      *          tax, lineItems, grandTotal, orderId, _id
11:      *
12:      */
13:     let n1qlQuery = `
14:       SELECT o.doc, o.custId, o.orderStatus,
15:       o.billingInfo, o.shippingInfo, o.shippingTotal,
16:       o.tax, o.lineItems, o.grandTotal, o.orderId, o._id
17:       FROM \`${this.bucketName}\` o
18:       WHERE o.doc.type = 'order'
19:         AND o.custId=$1 AND o.orderStatus = 'created'
20:       ORDER BY o.orderDate DESC NULLS FIRST
21:       LIMIT 1;`;
22:
23:     let options = {
24:       parameters: [customerId],
25:     };
26:
27:     //TODO:  prepared queries
28:     // if (adhoc) {
29:     //   options.adhoc = true;
30:     // }
31:
32:     let qResult = await this.cluster.query(n1qlQuery, options);
33:     if (!qResult.rows || qResult.rows.length == 0) {
34:       return { orders: [], error: null };
35:     }
36:
37:     return { orders: qResult.rows, error: null };
38:   } catch (err) {
39:     //Optional - add business logic to handle error types
40:     outputMessage(err, "repository.js:getNewOrder() - error:");
41:     return { orders: null, error: err };
42:   }
43: }
```

コードに関する注意事項:

- async/wait 構文を使用します。
- 12-20 行: N1QL クエリ文字列。 N1QL では、次の機能を利用します。
    - 特定の custI を見つけるための SDK の位置パラメータ化(例: o.custId = $1)。
    - クエリ結果の順序付け
    - クエリ結果の制限
- 22-24 行:クエリオプションオブジェクトを作成します。
- 31 行: N1QL のオペラは、3.x SDK のクラスター レベルで行われます。
    - クエリ操作パラメータ:
        - N1QL クエリ文字列
        - オブジェクト
- 32-36 行: クエリ結果の行を返すだけで、最新の順序として生の行情報を返すことができます。
- *outputMessage()*:コンソールに情報を簡単に出力するためのヘルパーメソッドは、/library ディレクトリにあります (付録で詳しく説明されている API のプロジェクト構造を参照してください)。
- try/catch & err オブジェクトの処理は、汎用的な方法で意図的に行 われます。 ラボ参加者は、エラーを処理するさまざまな方法をテストするために、それに応じてロジックを自由に追加できます。

完了したら、*repository.js* ファイルが保存されていることを確認します。 API *Docker* コンテナは API の作業ディレクトリにマップされるため、API コードに対して行われたすべての更新はコンテナに反映される必要があります。 コードが保存されると、顧客の新しい注文を取得する機能は、Web UI 内でアクティブにする必要があります。 ロジックが正しく機能しているかどうかを確認するには、注文が正常に実行されているかどうかを確認するには(Lab 3 を参照)、左上のカートアイコンに、ページの更新後、セッションがまだアクティブな場合、またはログイン後に、保留中の注文の項目が表示されます。

注 ***

getNewOrder () を使用するには承認が必要です。

注 ***

このロジックが作成されるまでは、*"/user/checkForNewOrder operation not built yet* "と言うメッセージが、さまざまなページの先頭に表示されます。これは、Web UI が新しいログイン後、またはページの更新時に、このロジック呼び出しを使用してカート ページを設定するためです。