N1QL

N1QL

To issue N1QL queries, you should create a *N1qlQuery* object, and pass it to *the query(N1qlQuery q)* method in the *Bucket* class

The return value from `query()` is the object `N1qlQueryResult`

Query Types

- Simple
  - Executes a single N1QL statement, no options available.
- Parameterized
  - Executes a parameterized query with positional or named params
- Prepared
  - Executed a previously prepared statement

Simple Query

```
1  bucket
2      .query(Query.simple(select("*").from(i("travel-sample")).limit(10)))
3      .flatMap(AsyncQueryResult::rows)
4      .forEach(System.out::println);
```


```
1  {"travel-sample":{"country":"United States","image":null,"hours":"Daily 11:30AM-10PM","address":"545 Haight Stree
2  {"travel-sample":{"country":"United Kingdom","image":null,"hours":null,"address":"50 Borough Road Altrincham WA15
3  {"travel-sample":{"airportname":"Twin Hills Airport","geo":{"alt":82,"lon":-160.275,"lat":59.074444},"country":"U
```


```
// raw string query
QueryResult queryResult =
   bucket.query(Query.simple("SELECT * FROM beer-sample LIMIT 10"));
```

Parameterized Query

- Named Params

```
bucket
    .query(Query.parametrized(
        select("*").from(i("travel-sample")).where(x("type").eq("$type")).limit(10),
        JsonObject.create().put("type", "airline")
    ))
    .flatMap(AsyncQueryResult::rows)
    .forEach(System.out::println);
```

Parametrized Query

- Positional Params

```
bucket
   .query(Query.parametrized(
     select("*").from(i("travel-sample")).where(x("type").eq("$1")),
     JsonArray.from("airline")
   ))
   .flatMap(AsyncQueryResult::rows)
   .forEach(System.out::println)
```

N1QL Query with placeholders

```
import static com.couchbase.client.java.query.Select.select;
import static com.couchbase.client.java.query.dsl.Expression.*;

// ...
Statement statement = select("fname", "lname", "age").from(i("default")).where(x("age").gt(x("$age")));
JsonObject placeholderValues = JsonObject.create().put("age", 22);
q = N1qlQuery.parameterized(statement, placeholderValues);
for (N1qlQueryRow row : bkt.query(q)) {
    System.out.println(row);
}
```

Prepared Query

```
1  Observable<QueryPlan> prepare = bucket.prepare("SELECT * FROM `travel-sample` LIMIT 10");
2
3  prepare
4      .flatMap(plan -> bucket.query(Query.prepared(plan)))
5      .doOnNext(res -> res.errors().subscribe(System.err::println))
6      .flatMap(res -> res.rows())
7      .subscribe(System.out::println);
```

MapReduce Views

MapReduce Views

```
function (doc, meta) {
    if (doc.type && doc.type == "beer" && doc.name) {
        emit(doc.name, null);
    }
}
```

definition of a `by_name` view in a *"beer"* design document

Querying Views

```
Bucket bkt = CouchbaseCluster.create("192.168.33.101").openBucket("beer-sample");
ViewResult result = bkt.query(ViewQuery.from("beer", "by_name");
for (ViewRow row : result) {
    System.out.println(row); //prints the row
    System.out.println(row.document().content()); //retrieves the doc and prints content
}
```

```
DefaultViewRow{id=harvey_son_lewes-a_lecoq_imperial_extra_double_stout_1999, key=A. LeCoq Imperial
Extra Double Stout 1999, value=null}
DefaultViewRow{id=harvey_son_lewes-a_lecoq_imperial_extra_double_stout_2000, key=A. LeCoq Imperial
Extra Double Stout 2000, value=null}
DefaultViewRow{id=mickey_finn_s_brewery-abana_amber_ale, key=Abana Amber Ale, value=null}
DefaultViewRow{id=brasserie_lefebvre-abbaye_de_floreffe_double, key=Abbaye de Floreffe Double,
value=null}
DefaultViewRow{id=brasserie_de_brunehaut-abbaye_de_saint_martin_blonde, key=Abbaye de Saint-Martin
Blonde, value=null}
```

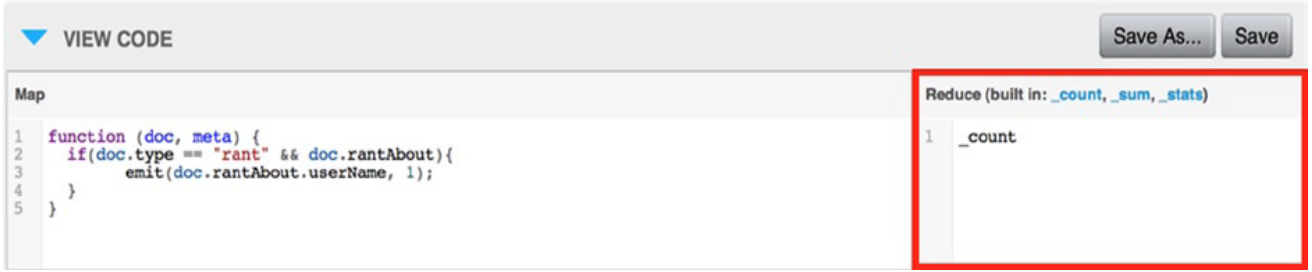
Querying Views

```
Bucket bkt = CouchbaseCluster.create("192.168.33.101").openBucket("beer-sample");
ViewQuery q = ViewQuery.from("beer", "by_name")
    .limit(5) // Limit to 5 results
    .startKey("A")
    .endKey("A\u0fff");

ViewResult result = bkt.query(q);
for (ViewRow row : result) {
    System.out.println(row);
}
```

```
DefaultViewRow{id=harvey_son_lewes-a_lecoq_imperial_extra_double_stout_1999, key=A. LeCoq Imperial
Extra Double Stout 1999, value=null}
DefaultViewRow{id=harvey_son_lewes-a_lecoq_imperial_extra_double_stout_2000, key=A. LeCoq Imperial
Extra Double Stout 2000, value=null}
DefaultViewRow{id=mickey_finn_s_brewery-abana_amber_ale, key=Abana Amber Ale, value=null}
DefaultViewRow{id=brasserie_lefebvre-abbaye_de_floreffe_double, key=Abbaye de Floreffe Double,
value=null}
DefaultViewRow{id=brasserie_de_brunehaut-abbaye_de_saint_martin_blonde, key=Abbaye de Saint-Martin
Blonde, value=null}
```

Example



tell Couchbase that you would like to group the view by key

```
String findAllBeersByAgregrate(CouchbaseClient client) {
        View view = client.getView("dev_brewery", "Specific_Attribute");
        Query query = new Query();
        query.setIncludeDocs(true).setLimit(20);
        query.setStale( Stale.FALSE );
        query.setReduce(true);
        ViewResponse result = client.query(view, query);
        StringBuffer res = new StringBuffer();
        for(ViewRow row : result) {
         res.append(row.getKey()).append("---").append(row.getValue()).append("<br>")
        }
        return res.toString();
}
```

Use Reduce

Creating design documents

```
CouchbaseCluster cluster = CouchbaseCluster.create("127.0.0.1");
Bucket bucket = cluster.openBucket("travel-sample");
// Get bucket manager
BucketManager bucketManager = bucket.bucketManager();

// Initialize design document
DesignDocument designDoc = DesignDocument.create(
    "landmarks",
    Arrays.asList(
        DefaultView.create("by_country",
            "function (doc, meta) { if (doc.type == 'landmark') { emit([doc.country, doc.city], null); }
}"),
        DefaultView.create("by_activity",
            "function (doc, meta) { if (doc.type == 'landmark') { emit(doc.activity, null); } }",
            "_count"),
        SpatialView.create("by_coordinates",
            "function (doc, meta) { if (doc.type == 'landmark') { emit([doc.geo.lon, doc.geo.lat], null); }
}")
    )
);

// Insert design document into the bucket
bucketManager.insertDesignDocument(designDoc);
```

Error Handling

Preparing to Fail

- Things will go wrong, so better plan for it
- Do not trust integration points (including the SDK)
- Synchronous retry & fallback is too damn hard

- RxJava provides (almost) everything you need to
  - fallback
  - retry
  - fail fast

Timeouts: Simple

```
1  bucket
2      .get("id")
3      .timeout(5, TimeUnit.SECONDS)
4      .subscribe(System.out::println);
```

Timeouts: Synchronous API

```
1  @Override
2  public <D extends Document<?>> D get(D document, long timeout, TimeUnit timeUnit) {
3    return asyncBucket
4         .get(document)
5         .timeout(timeout, timeUnit)
6         .toBlocking()
7         .singleOrDefault(null);
8  }
```

Coordinated Fallback

```
1  bucket
2      .get("beer") // fetch doc
3      .onErrorResumeNext(bucket.getFromReplica("beer", ReplicaMode.ALL)) // fallback to replica
4      .first() // grab the first doc to arrive
5      .map(doc -> doc.content().getString("name")) // extract the name
6      .timeout(2, TimeUnit.SECONDS) // only wait 2 secs
7      .doOnError(System.err::println) // error log if needed
8      .onErrorReturn(error -> "Not found!"); // if failure, return a default string
```

Coordinated Retry: Builder

- Declarative API instead of complicated retryWhen

```
observable.retryWhen(
  anyOf(BackpressureException.class, TemporaryFailureException.class)
    .delay(Delay.exponential(TimeUnit.SECONDS), 5)
    .max(100)
    .build()
);
```

Lab: Indexes Using Java API – Standalone App