

User Queries

These are example N1QL queries that may can performed to retrieve user related data.

Users By ID

The following query will get a User by their ID.

Query

[userbydocument_id.n1ql](#)

```
1 SELECT users.user_id, users.account.username, users.account.`password`
2 FROM `flight-data` AS users
3 USE KEYS 'user_197'
```

Result

```
1 [
2   {
3     "password": "N8HERvS8btfGbmz",
4     "user_id": 197,
5     "username": "Eudora43"
6   }
7 ]
```

The following query will retrieve multiple Users by their ID.

Query

[usersbydocument_id.n1ql](#)

```
1 SELECT users.details.*
2 FROM `flight-data` AS users
3 USE KEYS ['user_197', 'user_999']
```

Result

```

1  [
2    {
3      "company": null,
4      "dob": "2015-09-24",
5      "first_name": "Albin",
6      "home_country": "GQ",
7      "job_title": "Forward Markets Director",
8      "last_name": "Price",
9      "middle_name": null,
10     "prefix": "Dr.",
11     "suffix": null
12   },
13   {
14     "company": null,
15     "dob": null,
16     "first_name": "Dallas",
17     "home_country": "KH",
18     "job_title": "Central Functionality Executive",
19     "last_name": "Kunze",
20     "middle_name": "Harriett",
21     "prefix": null,
22     "suffix": null
23   }
24 ]

```

Users By Username

The following query will get a User by their Username.

Index

[idxusersusername.n1ql](#)

```

1  CREATE INDEX idx_users_username ON `flight-data` ( account.username )
2  WHERE doc_type = 'user'
3  USING GSI

```

Query

[userbyusername.n1ql](#)

```

1 SELECT users.user_id, users.details.first_name, users.details.last_name
2 FROM `flight-data` AS users
3 WHERE users.account.username = 'Eudora43'
4       AND users.doc_type = 'user'
5 LIMIT 1

```

Result

```

1 [
2   {
3     "first_name": "Albin",
4     "last_name": "Price",
5     "user_id": 197
6   }
7 ]

```

The following index and query will retrieve a user by their `username` and `password` .

Index

We need to update our index from the previous example, to do that we need to drop and recreate it.

[idxusersusername_drop.n1ql](#)

```

1 DROP INDEX `flight-data`.idx_users_username

```

[idxusersusername_password.n1ql](#)

```

1 CREATE INDEX idx_users_username_password ON `flight-data` ( account.username, account.`pa
2 WHERE doc_type = 'user'
3 USING GSI

```

Query

[userbyusername_password.n1ql](#)

```

1 SELECT users.user_id, users.details.first_name, users.details.last_name
2 FROM `flight-data` AS users
3 WHERE users.account.username = 'Eudora43'
4       AND users.account.`password` = 'N8HERvS8btfgbmz'
5       AND users.doc_type = 'user'
6 LIMIT 1

```

Result

```
1  [
2    {
3      "first_name": "Albin",
4      "last_name": "Price",
5      "user_id": 197
6    }
7  ]
```

Users Addresses

The following query will get a users addresses by their `user_id`

Query

[useraddressesbydocumentid.n1ql](#)

```
1  SELECT users.addresses
2  FROM `flight-data` AS users
3  USE KEYS 'user_197'
```

Result

```

1  [
2    {
3      "addresses": [
4        {
5          "address_1": "98527 Tromp Light Lodge",
6          "address_2": null,
7          "iso_country": "GQ",
8          "iso_region": "GQ-CS",
9          "locality": "South Selmerhaven",
10         "postal_code": "49540-9412",
11         "primary": true,
12         "type": "Home"
13       },
14       {
15         "address_1": "5783 Mathilde Vista Parkway",
16         "address_2": "Apt. 899",
17         "iso_country": "GQ",
18         "iso_region": "GQ-CS",
19         "locality": "Schinnerside",
20         "postal_code": "78895",
21         "primary": false,
22         "type": "Home"
23       }
24     ]
25   }
26 ]

```

However, these results are not friendly to work with as there is actually only 1 record returned. This is because we only selected a single user document which has a single property `addresses` that is an array of multiple addresses. We need to flatten this array and return it as separate documents.

Query

[useraddressesflattened.n1ql](#)

```

1  SELECT flattened_addresses.*
2  FROM `flight-data` AS users
3  USE KEYS 'user_197'
4  UNNEST users.addresses AS flattened_addresses

```

Result

```

1  [
2    {
3      "address_1": "98527 Tromp Light Lodge",
4      "address_2": null,
5      "iso_country": "GQ",
6      "iso_region": "GQ-CS",
7      "locality": "South Selmerhaven",
8      "postal_code": "49540-9412",
9      "primary": true,
10     "type": "Home"
11   },
12   {
13     "address_1": "5783 Mathilde Vista Parkway",
14     "address_2": "Apt. 899",
15     "iso_country": "GQ",
16     "iso_region": "GQ-CS",
17     "locality": "Schinnerside",
18     "postal_code": "78895",
19     "primary": false,
20     "type": "Home"
21   }
22 ]

```

We know that each of our users has a primary address, we need to be able to return just that address.

Query

[useraddressesflattened_primary.n1ql](#)

```

1  SELECT flattened_addresses.*
2  FROM `flight-data` AS users
3  USE KEYS 'user_197'
4  UNNEST users.addresses AS flattened_addresses
5  WHERE flattened_addresses.`primary` = true

```

Results

```

1  [
2    {
3      "address_1": "98527 Tromp Light Lodge",
4      "address_2": null,
5      "iso_country": "GQ",
6      "iso_region": "GQ-CS",
7      "locality": "South Selmerhaven",
8      "postal_code": "49540-9412",
9      "primary": true,
10     "type": "Home"
11   }
12 ]

```

Building on the previous examples, we want to return the full country and region names as part of each address.

Query

[useraddressesflattenedcountrycontinent.n1ql](#)

```

1  SELECT flattened_addresses.address_1, flattened_addresses.address_2, flattened_addresses
2     flattened_addresses.postal_code, flattened_addresses.`primary`, flattened_addresses.
3     flattened_addresses.iso_country, countries.country_name,
4     flattened_addresses.iso_region, regions.region_name
5  FROM `flight-data` AS users
6  USE KEYS 'user_197'
7  UNNEST users.addresses AS flattened_addresses
8  INNER JOIN `flight-data` AS countries
9     ON KEYS 'country_' || flattened_addresses.iso_country
10 INNER JOIN `flight-data` AS regions
11     ON KEYS 'region_' || flattened_addresses.iso_region

```

Result

```

1  [
2    {
3      "address_1": "98527 Tromp Light Lodge",
4      "address_2": null,
5      "country_name": "Equatorial Guinea",
6      "iso_country": "GQ",
7      "iso_region": "GQ-CS",
8      "locality": "South Selmerhaven",
9      "postal_code": "49540-9412",
10     "primary": true,
11     "region_name": "Centro Sur",
12     "type": "Home"
13   },
14   {
15     "address_1": "5783 Mathilde Vista Parkway",
16     "address_2": "Apt. 899",
17     "country_name": "Equatorial Guinea",
18     "iso_country": "GQ",
19     "iso_region": "GQ-CS",
20     "locality": "Schinnerside",
21     "postal_code": "78895",
22     "primary": false,
23     "region_name": "Centro Sur",
24     "type": "Home"
25   }
26 ]

```

And now with just the primary address information.

Query

[useraddressesflattenedprimarycountry_continent.n1ql](#)

```

1  SELECT flattened_addresses.address_1, flattened_addresses.address_2, flattened_addresses
2     flattened_addresses.postal_code, flattened_addresses.type,
3     flattened_addresses.iso_country, countries.country_name,
4     flattened_addresses.iso_region, regions.region_name
5  FROM `flight-data` AS users
6  USE KEYS 'user_197'
7  UNNEST users.addresses AS flattened_addresses
8  INNER JOIN `flight-data` AS countries
9     ON KEYS 'country_' || flattened_addresses.iso_country
10 INNER JOIN `flight-data` AS regions
11     ON KEYS 'region_' || flattened_addresses.iso_region
12 WHERE flattened_addresses.`primary` = true

```


Result

```
1  [
2    {
3      "address_1": "98527 Tromp Light Lodge",
4      "address_2": null,
5      "country_name": "Equatorial Guinea",
6      "iso_country": "GQ",
7      "iso_region": "GQ-CS",
8      "locality": "South Selmerhaven",
9      "postal_code": "49540-9412",
10     "region_name": "Centro Sur",
11     "type": "Home"
12   }
13 ]
```

Now we want to lookup our users by the region that they are in. To do this we will need to create an index on the `addresses[*].iso_region`. [Prior to Couchbase 4.5](#) the entire array had to be indexed and data could not be efficiently queried.

Index

[idxusersaddresses_regions.n1ql](#)

```
1  CREATE INDEX idx_users_addresses_region ON `flight-data` (
2    DISTINCT ARRAY address.iso_region
3    FOR address IN addresses
4    WHEN address.iso_region IS NOT NULL
5    END
6  )
7  WHERE doc_type = 'user';
```

Query

For our query, we do not want to return the entire `addresses` property, we want to omit the `primary` and `type` fields. The results should be sorted by `iso_region DESC` and have the `primary` address listed first.

[useraddressesby_region.n1ql](#)

```

1 SELECT users.details.first_name ||
2     IFNULL(' ' || users.details.last_name, '') AS name,
3     (
4         ARRAY {
5             "address_1": address.address_1,
6             "address_2": address.address_2,
7             "iso_region": address.iso_region,
8             "iso_country": address.iso_country,
9             "postal_code": address.postal_code,
10            "locality": address.locality
11        } FOR address IN users.addresses END
12    ) AS addresses
13 FROM `flight-data` AS users
14 WHERE users.doc_type = 'user'
15 AND (
16     ANY address IN users.addresses
17     SATISFIES address.iso_region IN [
18         'US-AK', 'US-MN', 'US-NC'
19     ]
20     END
21 )
22 ORDER BY users.addresses[*].iso_region DESC,
23     users.addresses[*].`primary` DESC

```

Result

```

1 [
2   {
3     "addresses": [
4       {
5         "address_1": "38582 Sigrid Terrace Cape",
6         "address_2": null,
7         "iso_country": "US",
8         "iso_region": "US-NC",
9         "locality": "Sengermouth",
10        "postal_code": "76275-0205",
11        "primary": false,
12        "type": "Work"
13      },
14      {
15        "address_1": "1844 Krajcik Unions Garden",
16        "address_2": "Apt. 925",
17        "iso_country": "US",
18        "iso_region": "US-NC",
19        "locality": "Macieborough",
20        "postal code": "17581-8835".

```

```

21     "primary": true,
22     "type": "Other"
23   }
24 ],
25   "first_name": "Marianna",
26   "last_name": "Labadie"
27 },
28 {
29   "addresses": [
30     {
31       "address_1": "82985 Angus Garden Mountain",
32       "address_2": null,
33       "iso_country": "US",
34       "iso_region": "US-NC",
35       "locality": "Swiftfort",
36       "postal_code": "26911-4639",
37       "primary": true,
38       "type": "Other"
39     }
40   ],
41   "first_name": "Yasmeen",
42   "last_name": "Ripplin"
43 },
44 {
45   "addresses": [
46     {
47       "address_1": "8913 Rodriguez Gardens Fords",
48       "address_2": "Apt. 764",
49       "iso_country": "US",
50       "iso_region": "US-MN",
51       "locality": "Bonniestad",
52       "postal_code": "07379",
53       "primary": true,
54       "type": "Home"
55     },
56     {
57       "address_1": "69403 Cleora Ports Shores",
58       "address_2": null,
59       "iso_country": "US",
60       "iso_region": "US-MN",
61       "locality": "Randiside",
62       "postal_code": "91175",
63       "primary": false,
64       "type": "Work"
65     }

```

```
66     ],
67     "first_name": "Winnifred",
68     "last_name": "Koepp"
69 },
70 {
71     "addresses": [
72         {
73             "address_1": "354 Susanna Row Falls",
74             "address_2": null,
75             "iso_country": "US",
76             "iso_region": "US-MN",
77             "locality": "Champlinchester",
78             "postal_code": "26170",
79             "primary": true,
80             "type": "Other"
81         },
82         {
83             "address_1": "38849 Brakus Divide Keys",
84             "address_2": null,
85             "iso_country": "US",
86             "iso_region": "US-MN",
87             "locality": "Barrowston",
88             "postal_code": "05343-0841",
89             "primary": false,
90             "type": "Home"
91         }
92     ],
93     "first_name": "Lola",
94     "last_name": "Emmerich"
95 },
96 {
97     "addresses": [
98         {
99             "address_1": "90856 Stark Streets Manors",
100             "address_2": null,
101             "iso_country": "US",
102             "iso_region": "US-MN",
103             "locality": "Port Carlie",
104             "postal_code": "56282-1062",
105             "primary": true,
106             "type": "Work"
107         }
108     ],
109     "first_name": "Marie",
110     "last_name": "Marks"
111 },
```

```

112 {
113   "addresses": [
114     {
115       "address_1": "6136 Kuhlman Isle Crossroad",
116       "address_2": null,
117       "iso_country": "US",
118       "iso_region": "US-MN",
119       "locality": "Emmittshire",
120       "postal_code": "41017-8748",
121       "primary": true,
122       "type": "Other"
123     }
124   ],
125   "first_name": "Emmie",
126   "last_name": null
127 }
128 ]

```

Building on the previous index and query, we want to retrieve all of the users, a unique array of each region the user is in, as well as the total # of addresses.

Query

[useraddressesbyregiondistinct.n1ql](#)

```

1  SELECT users.details.first_name ||
2     IFNULL(' ' || users.details.last_name, '') AS name,
3     ARRAY_LENGTH(users.addresses) AS addresses,
4     ARRAY_DISTINCT(
5         ARRAY address.iso_region FOR address IN users.addresses END
6     ) AS regions
7  FROM `flight-data` AS users
8  WHERE users.doc_type = 'user'
9     AND (
10     ANY address IN users.addresses
11         SATISFIES address.iso_region IN ['US-SC']
12     END
13 )
14 ORDER BY users.addresses[*].iso_region DESC

```

Result

```
1  [
2    {
3      "addresses": 2,
4      "name": "Lenny Borer",
5      "regions": [
6        "US-KS",
7        "US-SC"
8      ]
9    },
10   {
11     "addresses": 2,
12     "name": "Jasper Donnelly",
13     "regions": [
14       "US-SC"
15     ]
16   },
17   {
18     "addresses": 1,
19     "name": "Cindy Thiel",
20     "regions": [
21       "US-SC"
22     ]
23   },
24   {
25     "addresses": 2,
26     "name": "Zoila Koepp",
27     "regions": [
28       "US-NM",
29       "US-SC"
30     ]
31   }
32 ]
```

Users Phones

The following query will get a users phones by their `user_id`

Query

[userphonesbydocumentid.n1ql](#)

```
1 SELECT users.phones
2 FROM `flight-data` AS users
3 USE KEYS 'user_197'
```

Result

```
1 [
2   {
3     "phones": [
4       {
5         "extension": null,
6         "phone_number": "(570) 615-4605",
7         "primary": false,
8         "type": "Home"
9       },
10      {
11        "extension": "1735",
12        "phone_number": "(923) 578-2435",
13        "primary": true,
14        "type": "Mobile"
15      }
16    ]
17  }
18 ]
```

Just like the addresses, we need to flatten these results to make them more useful.

Query

[userphonesflattened.n1ql](#)

```
1 SELECT flattened_phones.*
2 FROM `flight-data` AS users
3 USE KEYS 'user_197'
4 UNNEST users.phones AS flattened_phones
```

Result

```

1  [
2    {
3      "extension": null,
4      "phone_number": "(570) 615-4605",
5      "primary": false,
6      "type": "Home"
7    },
8    {
9      "extension": "1735",
10     "phone_number": "(923) 578-2435",
11     "primary": true,
12     "type": "Mobile"
13   }
14 ]

```

We know that each of our users has a primary phone, we need to be able to return just that phone.

Query

[userphonesflattened_primary.n1ql](#)

```

1  SELECT flattened_phones.*
2  FROM `flight-data` AS users
3  USE KEYS 'user_197'
4  UNNEST users.phones AS flattened_phones
5  WHERE flattened_phones.`primary` = true

```

Results

```

1  [
2    {
3      "extension": "1735",
4      "phone_number": "(923) 578-2435",
5      "primary": true,
6      "type": "Mobile"
7    }
8  ]

```

Users Emails

The following query will get a users emails by their `user_id`

Query

[useremailsbydocumentid.n1ql](#)

```
1 SELECT users.emails
2 FROM `flight-data` AS users
3 USE KEYS 'user_1997'
```

Result

```
1 [
2   {
3     "emails": [
4       {
5         "email_address": "Chase.Kohler63@gmail.com",
6         "primary": false,
7         "type": "Home"
8       },
9       {
10        "email_address": "Judah66@gmail.com",
11        "primary": true,
12        "type": "Home"
13      },
14      {
15        "email_address": "Creola_Little34@gmail.com",
16        "primary": false,
17        "type": "Work"
18      }
19    ]
20  }
21 ]
```

Just like the addresses and phones, we need to flatten these results to make them more useful.

Query

[useremailsflattened.n1ql](#)

```
1 SELECT flattened_emails.*
2 FROM `flight-data` AS users
3 USE KEYS 'user_1997'
4 UNNEST users.emails AS flattened_emails
```

Result

```

1  [
2    {
3      "email_address": "Chase.Kohler63@gmail.com",
4      "primary": false,
5      "type": "Home"
6    },
7    {
8      "email_address": "Judah66@gmail.com",
9      "primary": true,
10     "type": "Home"
11   },
12   {
13     "email_address": "Creola_Little34@gmail.com",
14     "primary": false,
15     "type": "Work"
16   }
17 ]

```

We know that each of our users has a primary email address, we need to be able to return just that email.

Query

[useremailsflattened_primary.n1ql](#)

```

1  SELECT flattened_emails.*
2  FROM `flight-data` AS users
3  USE KEYS 'user_1997'
4  UNNEST users.emails AS flattened_emails
5  WHERE flattened_emails.`primary` = true

```

Results

```

1  [
2    {
3      "email_address": "Judah66@gmail.com",
4      "primary": true,
5      "type": "Home"
6    }
7  ]

```

The following query will retrieve only the email address from the array of objects, omitting the `type` and `primary` attributes.

Query

[useremailsflattenedemailonly.n1ql](#)

```
1 SELECT email
2 FROM `flight-data` AS users
3 USE KEYS 'user_1997'
4 UNNEST users.emails[*].email_address AS email
```

Results

```
1 [
2   {
3     "email": "Chase.Kohler63@gmail.com"
4   },
5   {
6     "email": "Judah66@gmail.com"
7   },
8   {
9     "email": "Creola_Little34@gmail.com"
10  }
11 ]
```

Building on the previous query, what if we wanted to ensure that the first email listed was the primary email address. We can perform the following query.

[useremailsflattenedprimaryemail_only.n1ql](#)

```
1 SELECT emails.email_address AS email
2 FROM `flight-data` AS users
3 USE KEYS 'user_1997'
4 UNNEST users.emails AS emails
5 WHERE emails.`primary` = true
```

```
1 [
2   {
3     "email": "Judah66@gmail.com"
4   },
5   {
6     "email": "Chase.Kohler63@gmail.com"
7   },
8   {
9     "email": "Creola_Little34@gmail.com"
10  }
11 ]
```

User By ID as Flat Object

Our user model uses nested attributes, we need to retrieve a users record as a single level object with just the primary address, phone and email.

Query

[userbydocumentidflat.n1ql](#)

```
1  SELECT users.account.*, users.details.*,
2     primary_email.email_address,
3     primary_address.address_1, primary_address.address_2, primary_address.iso_country,
4     primary_address.iso_region, primary_address.locality, primary_address.postal_code,
5     primary_phone.phone_number, primary_phone.extension AS phone_extension
6  FROM `flight-data` AS users
7  USE KEYS 'user_1997'
8  UNNEST users.emails AS primary_email
9  UNNEST users.addresses AS primary_address
10 UNNEST users.phones AS primary_phone
11 WHERE primary_email.`primary` = true
12        AND primary_address.`primary` = true
13        AND primary_phone.`primary` = true
```

Result

```
1  [
2    {
3      "address_1": "07363 Trantow Garden Crossroad",
4      "address_2": "Suite 108",
5      "company": null,
6      "created_on": 1447034465570,
7      "dob": "2016-02-17",
8      "email_address": "Judah66@gmail.com",
9      "first_name": "Donnell",
10     "home_country": "VC",
11     "iso_country": "VC",
12     "iso_region": "VC-01",
13     "job_title": null,
14     "last_login": 1463565402328,
15     "last_name": "Ortiz",
16     "locality": "South Leland",
17     "middle_name": "Winifred",
18     "modified_on": 1463561681928,
19     "password": "cbEsvC1RxKRv0gi",
20     "phone_extension": null,
21     "phone_number": "(569) 409-9444",
22     "postal_code": "26561",
23     "prefix": null,
24     "suffix": null,
25     "username": "Garett31"
26   }
27 ]
```