Lab 4: Basic Operations in CockroachDB

This lab guides you through some of the most essential CockroachDB SQL statements using an interactive SQL shell connected to a temporary, single-node CockroachDB cluster.

Start CockroachDB

Run the `cockroach demo` command:

```
cockroach demo
```

This starts a single-node, temporary cluster with the `movr` dataset pre-loaded.

Show tables

To see all tables in the active database, use the `SHOW TABLES` statement or the \dt shell command:

```
SHOW TABLES;
```

```
  schema_name |         table_name         | type  | estimated_row_count
--------------+----------------------------+-------+----------------------
  public      | promo_codes                | table |                 1000
  public      | rides                      | table |                  500
  public      | user_promo_codes           | table |                    0
  public      | users                      | table |                   50
  public      | vehicle_location_histories | table |                 1000
  public      | vehicles                   | table |                   15
(6 rows)
```

Create a table

Suppose that you want MovR to offer ride-sharing services, in addition to vehicle-sharing services. You'll need to add a table for drivers to the `movr` database. To create a table, use `CREATE TABLE` followed by a table name, the column names, and the [data type] and [constraint], if any, for each column:

```
CREATE TABLE drivers (
    id UUID NOT NULL,
    city STRING NOT NULL,
    name STRING,
    dl STRING UNIQUE,
    address STRING,
    CONSTRAINT "primary" PRIMARY KEY (city ASC, id ASC)
);
```

Table and column names must follow [these rules]. Also, when you do not explicitly define a [primary key], CockroachDB will automatically add a hidden `rowid` column as the primary key.

To avoid an error in case the table already exists, you can include `IF NOT EXISTS` :

```
CREATE TABLE IF NOT EXISTS drivers (
    id UUID NOT NULL,
```

```
    city STRING NOT NULL,
    name STRING,
    dl STRING UNIQUE,
    address STRING,
    CONSTRAINT "primary" PRIMARY KEY (city ASC, id ASC)
);
```

To show all of the columns from a table, use the `SHOW COLUMNS FROM <table>` statement or the `\d <table>` [shell command]:

```
SHOW COLUMNS FROM drivers;
```

```
  column_name | data_type | is_nullable | column_default | generation_expression |
indices           | is_hidden
--------------+-----------+-------------+----------------+-----------------------+----
----------------------+------------
  id          | UUID      |    false    | NULL           |                       |
{drivers_dl_key,primary} |   false
  city        | STRING    |    false    | NULL           |                       |
{drivers_dl_key,primary} |   false
  name        | STRING    |    true     | NULL           |                       |
{primary}                |   false
  dl          | STRING    |    true     | NULL           |                       |
{drivers_dl_key,primary} |   false
  address     | STRING    |    true     | NULL           |                       |
{primary}                |   false
(5 rows)
```

Insert rows

To insert a row into a table, use `INSERT INTO` followed by the table name and then the column values listed in the order in which the columns appear in the table:

```
INSERT INTO drivers VALUES
    ('c28f5c28-f5c2-4000-8000-000000000026', 'new york', 'Petee', 'ABC-1234', '101 5th
Ave');
```

If you want to pass column values in a different order, list the column names explicitly and provide the column values in the corresponding order:

```
INSERT INTO drivers (name, city, dl, address, id) VALUES
    ('Adam Driver', 'chicago', 'DEF-5678', '201 E Randolph St', '1eb851eb-851e-4800-
8000-000000000006');
```

To insert multiple rows into a table, use a comma-separated list of parentheses, each containing column values for one row:

```
INSERT INTO drivers VALUES
    ('8a3d70a3-d70a-4000-8000-00000000001b', 'seattle', 'Eric', 'GHI-9123', '400 Broad
St'),
    ('9eb851eb-851e-4800-8000-00000000001f', 'new york', 'Harry Potter', 'JKL-456',
'214 W 43rd St');
```

[Default values] are used when you leave specific columns out of your statement, or when you explicitly request default values. For example, both of the following statements create a row where the `name`, `dl`, and `address` entries each contain their default value, in this case `NULL`:

```
INSERT INTO drivers (id, city) VALUES
    ('70a3d70a-3d70-4400-8000-000000000016', 'chicago');
```

```
INSERT INTO drivers (id, city, name, dl, address) VALUES
    ('b851eb85-1eb8-4000-8000-000000000024', 'seattle', DEFAULT, DEFAULT, DEFAULT);
```

```
SELECT * FROM drivers WHERE id in ('70a3d70a-3d70-4400-8000-000000000016', 'b851eb85-
1eb8-4000-8000-000000000024');
```

```
                  id                  |  city   | name |  dl  | address
+-------------------------------------+---------+------+------+---------+
  70a3d70a-3d70-4400-8000-000000000016 | chicago | NULL | NULL | NULL
  b851eb85-1eb8-4000-8000-000000000024 | seattle | NULL | NULL | NULL
(2 rows)
```

Create an index

[Indexes] help locate data without having to look through every row of a table. They're automatically created for the [primary key] of a table and any columns with a `UNIQUE` constraint].

To create an index for non-unique columns, use `CREATE INDEX` followed by an optional index name and an `ON` clause identifying the table and column(s) to index. For each column, you can choose whether to sort ascending (`ASC`) or descending (`DESC`).

```
CREATE INDEX name_idx ON users (name DESC);
```

You can create indexes during table creation as well; just include the `INDEX` keyword followed by an optional index name and the column(s) to index:

```
CREATE TABLE IF NOT EXISTS drivers (
    id UUID NOT NULL,
    city STRING NOT NULL,
    name STRING,
    dl STRING,
    address STRING,
    INDEX name_idx (name),
    CONSTRAINT "primary" PRIMARY KEY (city ASC, id ASC)
);
```

Show indexes

To show the indexes on a table, use `SHOW INDEX FROM` followed by the name of the table:

```
SHOW INDEX FROM users;
```

```
  table_name | index_name | non_unique | seq_in_index | column_name | direction |
storing | implicit | visible
```

```
-------------+------------+------------+--------------+-------------+-----------+-----
----+----------+----------
  users      | name_idx   |     t      |            1 | name        | DESC      |    f
|    f     |    t
  users      | name_idx   |     t      |            2 | city        | ASC       |    f
|    t     |    t
  users      | name_idx   |     t      |            3 | id          | ASC       |    f
|    t     |    t
  users      | users_pkey |     f      |            1 | city        | ASC       |    f
|    f     |    t
  users      | users_pkey |     f      |            2 | id          | ASC       |    f
|    f     |    t
  users      | users_pkey |     f      |            3 | name        | N/A       |    t
|    f     |    t
  users      | users_pkey |     f      |            4 | address     | N/A       |    t
|    f     |    t
  users      | users_pkey |     f      |            5 | credit_card | N/A       |    t
|    f     |    t
(8 rows)
```

Query a table

To query a table, use `SELECT` followed by a comma-separated list of the columns to be returned and the table from which to retrieve the data. You can also use the `LIMIT` clause to restrict the number of rows retrieved:

```
SELECT name FROM users LIMIT 10;
```

```
        name
+-------------------+
  William Wood
  Victoria Jennings
  Tyler Dalton
  Tony Ortiz
  Tina Miller
  Taylor Cunningham
  Susan Morse
  Steven Lara
  Stephen Diaz
  Sarah Wang DDS
(10 rows)
```

To retrieve all columns, use the `*` wildcard:

```
SELECT * FROM users LIMIT 10;
```

```
                  id                  |   city    |        name        |
address             | credit_card
+--------------------------------------+-----------+--------------------+-------------
------------------+-------------+
  c28f5c28-f5c2-4000-8000-000000000026 | amsterdam | Maria Weber        | 14729 Karen
Radial              | 5844236997
  c7ae147a-e147-4000-8000-000000000027 | amsterdam | Tina Miller        | 97521 Mark
```

```
Extensions          | 8880478663
  cccccccc-cccc-4000-8000-000000000028 | amsterdam | Taylor Cunningham  | 89214
Jennifer Well            | 5130593761
  d1eb851e-b851-4800-8000-000000000029 | amsterdam | Kimberly Alexander | 48474 Alfred
Hollow            | 4059628542
  19999999-9999-4a00-8000-000000000005 | boston    | Nicole Mcmahon     | 11540 Patton
Extensions        | 0303726947
  1eb851eb-851e-4800-8000-000000000006 | boston    | Brian Campbell     | 92025 Yang
Village             | 9016427332
  23d70a3d-70a3-4800-8000-000000000007 | boston    | Carl Mcguire       | 60124 Palmer
Mews Apt. 49      | 4566257702
  28f5c28f-5c28-4600-8000-000000000008 | boston    | Jennifer Sanders   | 19121
Padilla Brooks Apt. 12   | 1350968125
  80000000-0000-4000-8000-000000000019 | chicago   | Matthew Clay       | 49220 Lisa
Junctions           | 9132291015
  851eb851-eb85-4000-8000-00000000001a | chicago   | Samantha Coffey    | 6423 Jessica
Underpass Apt. 87 | 9437219051
(10 rows)
```

To filter the results, add a `WHERE` clause identifying the columns and values to filter on:

```
SELECT id, name FROM users WHERE city = 'san francisco';
```

```
                  id                  |       name
--------------------------------------+-------------------
  75c28f5c-28f5-4400-8000-000000000017 | William Wood
  7ae147ae-147a-4000-8000-000000000018 | Alfred Garcia
  80000000-0000-4000-8000-000000000019 | Matthew Clay
  851eb851-eb85-4000-8000-00000000001a | Samantha Coffey
  8a3d70a3-d70a-4000-8000-00000000001b | Jessica Martinez
(5 rows)
```

To sort the results, add an `ORDER BY` clause identifying the columns to sort by. For each column, you can choose whether to sort ascending (`ASC`) or descending (`DESC`).

```
SELECT city, type, current_location FROM vehicles ORDER BY city, type DESC;
```

```
      city      |    type    |        current_location
+---------------+------------+-------------------------------+
  amsterdam     | skateboard | 19202 Edward Pass
  boston        | scooter    | 19659 Christina Ville
  chicago       | skateboard | 69721 Noah River
  detroit       | scooter    | 43051 Jonathan Fords Suite 36
  los angeles   | skateboard | 49164 Anna Mission Apt. 38
  minneapolis   | scooter    | 62609 Stephanie Route
  minneapolis   | scooter    | 57637 Mitchell Shoals Suite 59
  new york      | skateboard | 64110 Richard Crescent
  new york      | scooter    | 86667 Edwards Valley
  paris         | skateboard | 2505 Harrison Parkway Apt. 89
  rome          | bike       | 64935 Matthew Flats Suite 55
  san francisco | skateboard | 81472 Morris Run
  san francisco | scooter    | 91427 Steven Spurs Apt. 49
```

```
  seattle       | bike       | 37754 Farmer Extension
  washington dc | scooter    | 47259 Natasha Cliffs
(15 rows)
```

Update rows

To update rows in a table, use `UPDATE` followed by the table name, a `SET` clause identifying the columns to update and their new values, and a `WHERE` clause identifying the rows to update:

```
UPDATE promo_codes SET (description, rules) = ('EXPIRED', '{"type":
"percent_discount", "value": "0%"}') WHERE expiration_time < '2019-01-22
03:04:05+00:00';
```

```
SELECT code, description, rules FROM promo_codes LIMIT 10;
```

```
             code             | description |                     rules
------------------------------+-------------+-----------------------------------------
----
  0_explain_theory_something | EXPIRED     | {"type": "percent_discount", "value":
"0%"}
  100_address_garden_certain | EXPIRED     | {"type": "percent_discount", "value":
"0%"}
  101_system_skin_night      | EXPIRED     | {"type": "percent_discount", "value":
"0%"}
  102_card_professional_kid  | EXPIRED     | {"type": "percent_discount", "value":
"0%"}
  103_now_project_focus      | EXPIRED     | {"type": "percent_discount", "value":
"0%"}
  104_long_become_prove      | EXPIRED     | {"type": "percent_discount", "value":
"0%"}
  105_republican_guess_arm   | EXPIRED     | {"type": "percent_discount", "value":
"0%"}
  106_court_especially_plan  | EXPIRED     | {"type": "percent_discount", "value":
"0%"}
  107_she_matter_ten         | EXPIRED     | {"type": "percent_discount", "value":
"0%"}
  108_wind_marriage_for      | EXPIRED     | {"type": "percent_discount", "value":
"0%"}
(10 rows)
```

If a table has a primary key, you can use that in the `WHERE` clause to reliably update specific rows; otherwise, each row matching the `WHERE` clause is updated. When there's no `WHERE` clause, all rows in the table are updated.

Delete rows

To delete rows from a table, use `DELETE FROM` followed by the table name and a `WHERE` clause identifying the rows to delete:

```
DELETE FROM promo_codes WHERE description = 'EXPIRED';
```

Just as with the `UPDATE` statement, if a table has a primary key, you can use that in the `WHERE` clause to reliably delete specific rows; otherwise, each row matching the `WHERE` clause is deleted. When there's no `WHERE` clause, all rows in the table are deleted.

Remove a table

When you no longer need a table, use `DROP TABLE` followed by the table name to remove the table and all its data:

```
DROP TABLE drivers;
```

You can exit the CockroachDB shell now.

```
exit
```