Lab n°1

Étape 3

Démarrage de la session Cassandra:

```
$ ./cassandra
Starting a Cassandra cluster ... DONE!
Cassandra successfully started.
$ cqlsh
WARNING: cqlsh was built against 5.0.0, but this server is 4.0.14.  All features may not work!
Connected to Cassandra Cluster at 127.0.0.1:9042
[cqlsh 6.2.0 | Cassandra 4.0.14 | CQL spec 3.4.5 | Native protocol v5]
Use HELP for help.
cqlsh>
```

Création et utilisation du keyspace:

```
CREATE KEYSPACE IF NOT EXISTS ks_inserts_updates_deletes
WITH replication = {
    'class': 'NetworkTopologyStrategy',
    'DC-Houston': 1 };
USE ks_inserts_updates_deletes;
```

Création des tables:

```
CREATE TABLE IF NOT EXISTS users (
    email TEXT,
    name TEXT,
    age INT,
    date_joined DATE,
    PRIMARY KEY ((email))
);
CREATE TABLE IF NOT EXISTS ratings_by_user (
    email TEXT,
    title TEXT,
    year INT,
    rating INT,
    PRIMARY KEY ((email), title, year)
);
```

Étape 4

Insertion de l'utilisateur dans `users`:

```
INSERT INTO users (email, name, age, date_joined)
VALUES ('joe@datastax.com', 'Joe', 25, '2020-01-01');
```

Insertion des dans `ratings_by_user`:

```
INSERT INTO ratings_by_user (email, title, year, rating)
VALUES ('joe@datastax.com', 'Alice in Wonderland', 2010, 9);
INSERT INTO ratings_by_user (email, title, year, rating)
VALUES ('joe@datastax.com', 'Edward Scissorhands', 1990, 10);
```

Sélection de toutes les lignes:

```
> SELECT * FROM users;
```

```
 email            | age | date_joined | name
------------------+-----+-------------+------
 joe@datastax.com |  25 |  2020-01-01 |  Joe

(1 rows)
> SELECT * FROM ratings_by_user;

 email            | title               | year | rating
------------------+---------------------+------+--------
 joe@datastax.com | Alice in Wonderland | 2010 |      9
 joe@datastax.com | Edward Scissorhands | 1990 |     10

(2 rows)
```

Insertion du reste de l'exemple:

```
INSERT INTO users (email, name, age, date_joined)
VALUES ('jen@datastax.com', 'Jen', 27, '2020-01-01');

INSERT INTO ratings_by_user (email, title, year, rating)
VALUES ('jen@datastax.com', 'Alice in Wonderland', 2010, 10);
INSERT INTO ratings_by_user (email, title, year, rating)
VALUES ('jen@datastax.com', 'Alice in Wonderland', 1951, 8);
```

Étape 5

Mise à jour dans `users`:

```
UPDATE users SET name = 'Joseph', age = 26
WHERE email = 'joe@datastax.com';
```

Mise à jour d'un des ratings de `ratings_by_user`:

```
UPDATE ratings_by_user SET rating = 5
WHERE email = 'joe@datastax.com' AND title = 'Alice in Wonderland' AND year = 2010;
```

Étape 6

Suppression d'une valeur dans une ligne:

```
DELETE age FROM users
WHERE email = 'joe@datastax.com';
```

Suppression d''une ligne:

```
DELETE FROM ratings_by_user
WHERE email = 'joe@datastax.com'
    AND title = 'Alice in Wonderland'
    AND year  = 2010;
```

Supprimer toutes les lignes d'une partition:

```
DELETE FROM ratings_by_user
WHERE email = 'joe@datastax.com';
```

Supprimer le reste:

```
DELETE FROM users WHERE email = 'jen@datastax.com';
DELETE FROM users WHERE email = 'joe@datastax.com';
DELETE FROM ratings_by_user WHERE email = 'joe@datastax.com';
DELETE FROM ratings_by_user WHERE email = 'jen@datastax.com';
```

Étape 7

Vérifier qu'on a bien tout supprimé:

```
TRUNCATE TABLE users;
TRUNCATE TABLE ratings_by_user;
```

Insérer les nouvelles données:

```
UPDATE users
SET name = 'Joe', date_joined = '2020-01-01'
WHERE email = 'joe@datastax.com';
UPDATE users
SET name = 'Jen', date_joined = '2020-01-01'
WHERE email = 'jen@datastax.com';

UPDATE ratings_by_user
SET rating = -9
WHERE email = 'joe@datastax.com'
    AND title = 'Alice in Wonderland'
    AND year  = 2010;
UPDATE ratings_by_user
SET rating = -10
WHERE email = 'joe@datastax.com'
    AND title = 'Edward Scissorhands'
    AND year  = 1990;
UPDATE ratings_by_user
SET rating = -8
WHERE email = 'jen@datastax.com'
    AND title = 'Alice in Wonderland'
    AND year  = 1951;
UPDATE ratings_by_user
SET rating = -10
WHERE email = 'jen@datastax.com'
    AND title = 'Alice in Wonderland'
    AND year  = 2010;
```

Corriger les erreurs:

```
INSERT INTO users (email, age)
VALUES ('joe@datastax.com', 25);
INSERT INTO users (email, age)
VALUES ('jen@datastax.com', 27);

INSERT INTO ratings_by_user (email, title, year, rating)
VALUES ('joe@datastax.com', 'Alice in Wonderland', 2010, 9);
INSERT INTO ratings_by_user (email, title, year, rating)
VALUES ('joe@datastax.com', 'Edward Scissorhands', 1990, 10);
INSERT INTO ratings_by_user (email, title, year, rating)
VALUES ('jen@datastax.com', 'Alice in Wonderland', 1951, 8);
INSERT INTO ratings_by_user (email, title, year, rating)
VALUES ('jen@datastax.com', 'Alice in Wonderland', 2010, 10);
```

Étape 8

Mettre à jour l'email dans `users`:

```
INSERT INTO users (email, name, age, date_joined)
VALUES ('joseph@datastax.com', 'Joe', 25, '2020-01-01');
DELETE FROM users
WHERE email = 'joe@datastax.com';
```

Mettre à jour l'email dans `ratings_by_user`:

```
INSERT INTO ratings_by_user (email, title, year, rating)
VALUES ('joseph@datastax.com', 'Alice in Wonderland', 2010, 9);
INSERT INTO ratings_by_user (email, title, year, rating)
VALUES ('joseph@datastax.com', 'Edward Scissorhands', 1990, 10);
DELETE FROM ratings_by_user WHERE email = 'joe@datastax.com';
```

Étape 9

Insertion conditionnelle:

```
INSERT INTO users (email, name, age, date_joined)
VALUES ('art@datastax.com', 'Art', 33, '2020-05-04')
IF NOT EXISTS;
INSERT INTO users (email, name, age, date_joined)
VALUES ('art@datastax.com', 'Arthur', 44, '2020-05-04')
IF NOT EXISTS;
```

Mise à jour conditionnelle:

```
UPDATE users SET name = 'Artie'
WHERE email = 'art@datastax.com' IF name = 'Art';
UPDATE users SET name = 'Arthur'
WHERE email = 'art@datastax.com' IF name = 'Art';
```

Lab n°2

Étape 1:

Création et utilisation du keyspace:

```
CREATE KEYSPACE investment_data
WITH replication = {
    'class': 'NetworkTopologyStrategy',
    'DC-Houston': 1 };
USE investment_data;
```

Étape 2:

Créer `accounts_by_user`:

```
CREATE TABLE IF NOT EXISTS accounts_by_user (
    username TEXT,
    account_number TEXT,
    cash_balance DECIMAL,
    name TEXT STATIC,
    PRIMARY KEY ((username),account_number)
);
```

Créer `positions_by_account`:

```
CREATE TABLE IF NOT EXISTS positions_by_account (
    account TEXT,
    symbol TEXT,
    quantity DECIMAL,
    PRIMARY KEY ((account),symbol)
);
```

Créer `trades_by_a_d`:

```
CREATE TABLE IF NOT EXISTS trades_by_a_d (
    account TEXT,
    trade_id TIMEUUID,
    type TEXT,
    symbol TEXT,
    shares DECIMAL,
    price DECIMAL,
    amount DECIMAL,
    PRIMARY KEY ((account),trade_id)
) WITH CLUSTERING ORDER BY (trade_id DESC);
```

Créer `trades_by_a_td`:

```
CREATE TABLE IF NOT EXISTS trades_by_a_td (
    account TEXT,
    trade_id TIMEUUID,
    type TEXT,
    symbol TEXT,
    shares DECIMAL,
    price DECIMAL,
    amount DECIMAL,
    PRIMARY KEY ((account),type,trade_id)
) WITH CLUSTERING ORDER BY (type ASC, trade_id DESC);
```

Créer `trades_by_a_std`:

```
CREATE TABLE IF NOT EXISTS trades_by_a_std (
    account TEXT,
    trade_id TIMEUUID,
    type TEXT,
    symbol TEXT,
    shares DECIMAL,
    price DECIMAL,
    amount DECIMAL,
    PRIMARY KEY ((account),symbol,type,trade_id)
) WITH CLUSTERING ORDER BY (symbol ASC, type ASC, trade_id DESC);
```

Créer `trades_by_a_sd`:

```
CREATE TABLE IF NOT EXISTS trades_by_a_sd (
    account TEXT,
    trade_id TIMEUUID,
    type TEXT,
    symbol TEXT,
    shares DECIMAL,
    price DECIMAL,
    amount DECIMAL,
    PRIMARY KEY ((account),symbol,trade_id)
) WITH CLUSTERING ORDER BY (symbol ASC, trade_id DESC);
```

Vérfier les tables:

```
DESCRIBE TABLES;
```

Étape 3:

Insérer les données:

```
SOURCE 'assets/investment_data.cql'
```

Récupérer toutes les lignes de `accounts_by_user`:

```
SELECT * FROM accounts_by_user;
```

Récupérer toutes les lignes de `positions_by_account`:

```
SELECT * FROM positions_by_account;
```

Récupérer toutes les lignes de `trades_by_a_d`:

```
SELECT * FROM trades_by_a_d;
```

Récupérer toutes les lignes de `trades_by_a_td`:

```
SELECT * FROM trades_by_a_td;
```

Récupérer toutes les lignes de `trades_by_a_std`:

```
SELECT * FROM trades_by_a_std;
```

Récupérer toutes les lignes de `trades_by_a_sd`:

```
SELECT * FROM trades_by_a_sd;
```

Étape 4:

Trouver les informations sur les comptes d'investiment de `joe`:

```
SELECT * FROM accounts_by_user
WHERE username = 'joe';
```

Étape 5:

Récupérer toutes les positions du compte `joe001` par symbol ascendant:

```
SELECT * FROM positions_by_account
WHERE account = 'joe001'
ORDER BY symbol ASC;
```

Étape 6:

Récupérer les échanges du compte `joe001` par date d'échange descendante:

```
SELECT * FROM trades_by_a_d
WHERE account = 'joe001'
ORDER BY trade_id DESC;
```

Étape 7:

Récupérer les échanges de `joe001` entre le 2020-09-07 et le 2020-09-11 par date descendante:

```
SELECT account,
    TODATE(DATEOF(trade_id)) AS date,
    trade_id, type, symbol,
    shares, price, amount
FROM trades_by_a_d
WHERE account = 'joe001'
  AND trade_id > maxTimeuuid('2020-09-07')
  AND trade_id < minTimeuuid('2020-09-12');
```

bla:

```

```

bla:

```

```

Étape 8:

Tous les achats de `joe001` entre le 2020-09-07 et le 2020-09-11:

```
SELECT account,
    TODATE(DATEOF(trade_id)) AS date,
    trade_id, type, symbol,
    shares, price, amount
FROM trades_by_a_d
WHERE account = 'joe001'
    AND trade_id > maxTimeuuid('2020-09-07')
    AND trade_id < minTimeuuid('2020-09-12')
    AND type = 'buy'
ALLOW FILTERING;
```

Étape 9:

Trouver les achats sur AAPL de `joe001` entre le 2020-09-07 et le 2020-0907:

```
SELECT account,
    TODATE(DATEOF(trade_id)) AS date,
    trade_id, type, symbol,
    shares, price, amount
FROM trades_by_a_d
WHERE account = 'joe001'
    AND trade_id > maxTimeuuid('2020-09-07')
    AND trade_id < minTimeuuid('2020-09-12')
    AND type = 'buy'
    AND symbol = 'AAPL'
ALLOW FILTERING;
```

Étape 10:

Find all trades for account joe001, date range 2020-09-07 - 2020-09-11 and instrument symbol AAPL;

Trouver les échanges de `joe001` entre le 2020:

```
SELECT account,
    TODATE(DATEOF(trade_id)) AS date,
    trade_id, type, symbol,
    shares, price, amount
FROM trades_by_a_d
WHERE account = 'joe001'
    AND trade_id > maxTimeuuid('2020-09-07')
    AND trade_id < minTimeuuid('2020-09-12')
    AND type = 'buy'
    AND symbol = 'AAPL'
ALLOW FILTERING;
```

Lab n°3

Étape 1:

Créer et utiliser le keyspace:

```
CREATE KEYSPACE messaging_data
WITH replication = {
    'class': 'NetworkTopologyStrategy',
    'DC-Houston': 1 };
USE messaging_data;
```

Étape 2:

Créer la table `folders_by_user`:

```
CREATE TABLE IF NOT EXISTS folders_by_user (
    username TEXT,
    label TEXT,
    color TEXT,
    PRIMARY KEY ((username),label)
);
```

Créer la table `unread_email_stats`:

```
CREATE TABLE IF NOT EXISTS unread_email_stats (
    username TEXT,
    label TEXT,
    num_unread COUNTER,
    PRIMARY KEY ((username),label)
);
```

Créer la table `emails_by_user_folder`:

```
CREATE TABLE IF NOT EXISTS emails_by_user_folder (
    username TEXT,
    label TEXT,
    id TIMEUUID,
    "from" TEXT,
    subject TEXT,
    is_read BOOLEAN,
    PRIMARY KEY ((username,label),id)
) WITH CLUSTERING ORDER BY (id DESC);
```

Créer la table `emails`:

```
CREATE TABLE IF NOT EXISTS emails (
    id TIMEUUID,
    "to" LIST<TEXT>,
    "from" TEXT,
    subject TEXT,
    body TEXT,
    attachments MAP<TEXT,INT>,
    PRIMARY KEY ((id))
);
```

Créer la table `attachments`:

```
CREATE TABLE IF NOT EXISTS attachments (
```

```
    email_id TIMEUUID,
    filename TEXT,
    chunk_number INT,
    type TEXT,
    value BLOB,
    PRIMARY KEY ((email_id,filename,chunk_number))
);
```

Vérifier les tables:

```
DESCRIBE TABLES;
```

Étape 3:

Insérer les données d'exemple:

```
SOURCE 'assets/messaging_data.cql'
```

Lire la table `folders_by_user`:

```
SELECT * FROM folders_by_user;
```

Lire la table `unread_email_stats`:

```
SELECT * FROM unread_email_stats;
```

Lire la table `emails_by_user_folder`:

```
SELECT * FROM emails_by_user_folder;
```

Lire la table `emails`:

```
SELECT id, "to", "from" FROM emails;
SELECT id, subject, body FROM emails;
SELECT id, attachments FROM emails;
```

Lire la table `attachments`:

```
SELECT * FROM attachments;
```

Étape 4:

Trouver les dossier et couleurs de `joe@datastax.com`:

```
SELECT * FROM folders_by_user
WHERE username = 'joe@datastax.com';
```

Trouver les dossier et le nombre de mails non-lus de `joe@datastax.com`:

```
SELECT * FROM unread_email_stats
WHERE username = 'joe@datastax.com';
```

Étape 5:

Trouver les ids, sujets, émetteurs, statuts de lecture, et horodatage des mails dans la boîte de réception de `joe@datastax.com` par horodatage descendant:

```
SELECT id, subject, "from", is_read, toTimestamp(id) as timestamp
FROM emails_by_user_folder
WHERE username = 'joe@datastax.com'
    AND label = 'inbox';
```

Étape 6:

Trouver toutes les infos sur le mail `8ae31dd0-d361-11ea-a40e-5dd6331dfc45`:

```
SELECT id, attachments, body, "from", subject, "to", toTimestamp(id) as timestamp
FROM emails
WHERE id = 8ae31dd0-d361-11ea-a40e-5dd6331dfc45;
```

Étape 7:

Trouver la pièce jointe `Budget.xlsx` du mail `8ae31dd0-d361-11ea-a40e-5dd6331dfc45` avec `chunk_number` 1:

```
SELECT * FROM attachments
WHERE email_id = 8ae31dd0-d361-11ea-a40e-5dd6331dfc45
    AND filename = 'Budget.xlsx'
    AND chunk_number = 1;
```

Find an attachment file with name Presentation.pptx for an email with id 8ae31dd0-d361-11ea-a40e-5dd6331dfc45, assuming that the three file chunks are stored across three partitions with chunk numbers 1, 2 and 3:

Trouver la pièce jointe `Presentation.pptx` du mail `8ae31dd0-d361-11ea-a40e-5dd6331dfc45` séparé en trois chunks 1 2 et 3:

```
SELECT * FROM attachments
WHERE email_id = 8ae31dd0-d361-11ea-a40e-5dd6331dfc45
    AND filename = 'Presentation.pptx'
ALLOW FILTERING;
```

Lab n°4

Étape 2

Créer les tables:

```
CREATE TABLE IF NOT EXISTS performers (
    name TEXT,
    type TEXT,
    country TEXT,
    born INT,
    died INT,
    founded INT,
    PRIMARY KEY ((name))
);

CREATE TABLE IF NOT EXISTS albums_by_performer (
    performer TEXT,
    year INT,
    title TEXT,
    genre TEXT,
    PRIMARY KEY ((performer),year,title)
    ) WITH CLUSTERING ORDER BY (year DESC, title ASC);

CREATE TABLE IF NOT EXISTS albums_by_title (
    title TEXT,
    year INT,
    performer TEXT,
    genre TEXT,
    PRIMARY KEY ((title),year)
) WITH CLUSTERING ORDER BY (year DESC);

CREATE TABLE IF NOT EXISTS albums_by_genre (
    genre TEXT,
    year INT,
    title TEXT,
    performer TEXT,
    PRIMARY KEY ((genre),year,title)
) WITH CLUSTERING ORDER BY (year DESC, title ASC);

CREATE TABLE IF NOT EXISTS tracks_by_title (
    title TEXT,
    album_year INT,
    album_title TEXT,
    number INT,
    length INT,
    genre TEXT,
    PRIMARY KEY ((title),album_year,album_title,number)
) WITH CLUSTERING ORDER BY (album_year DESC, album_title ASC, number ASC);

CREATE TABLE IF NOT EXISTS tracks_by_album (
    album_title TEXT,
    album_year INT,
    number INT,
    title TEXT,
    length INT,
    genre TEXT STATIC,
  PRIMARY KEY ((album_title,album_year),number)
);

CREATE TABLE IF NOT EXISTS users (
    id UUID,
    name TEXT,
    PRIMARY KEY ((id))
);

CREATE TABLE IF NOT EXISTS tracks_by_user (
    id UUID,
    month DATE,
    timestamp TIMESTAMP,
    album_title TEXT,
```

```
    album_year INT,
    number INT,
    title TEXT,
    length INT,
    PRIMARY KEY ((id,month),timestamp,album_title,album_year,number)
) WITH CLUSTERING ORDER BY (timestamp DESC, album_title ASC, album_year ASC, number ASC);
```

Vérifier les tables:

```
DESCRIBE TABLES;
```

Étape 3

Charger les données de `performers`:

```
export PATH=/home/gitpod/dsbulk/bin:$PATH
dsbulk load -url assets/performers.csv -k music_data -t performers -header true -logDir /tmp/logs
```

Afficher des lignes de `performers`:

```
SELECT * FROM music_data.performers LIMIT 10;
```

Charger les données de `albums_by_performer`, `albums_by_title` et `albums_by_genre`:

```
dsbulk load -url assets/albums.csv -k music_data -t albums_by_performer -header true -logDir /tmp/logs
dsbulk load -url assets/albums.csv -k music_data -t albums_by_title -header true -logDir /tmp/logs
dsbulk load -url assets/albums.csv -k music_data -t albums_by_genre -header true -logDir /tmp/logs
```

Lire des lignes de `albums_by_performer`, `albums_by_title` et `albums_by_genre`:

```
SELECT * FROM music_data.albums_by_performer LIMIT 5;
SELECT * FROM music_data.albums_by_title LIMIT 5;
SELECT * FROM music_data.albums_by_genre LIMIT 5;
```

Charger les données de `tracks_by_title` et `tracks_by_album`:

```
dsbulk load -url assets/tracks.csv -k music_data -t tracks_by_title -header true -m "0=album_title,
1=album_year, 2=genre, 3=number, 4=title" -logDir /tmp/logs
dsbulk load -url assets/tracks.csv -k music_data -t tracks_by_album -header true -m "0=album_title,
1=album_year, 2=genre, 3=number, 4=title" -logDir /tmp/logs
```

Lire des lignes de `tracks_by_title` et `tracks_by_album`:

```
SELECT * FROM music_data.tracks_by_title LIMIT 5;
SELECT * FROM music_data.tracks_by_album LIMIT 5;
```

Étape 4

Insérer des lignes dans `users`:

```
INSERT INTO users (id, name)
VALUES (12345678-aaaa-bbbb-cccc-123456789abc, 'Joe');
INSERT INTO users (id, name)
VALUES (UUID(), 'Jen');
```

```
INSERT INTO users (id, name)
VALUES (UUID(), 'Jim');
```

Insérer des lignes dans `tracks_by_user`:

```
INSERT INTO tracks_by_user (id, month, timestamp, album_title, album_year, number, title)
VALUES (12345678-aaaa-bbbb-cccc-123456789abc, '2020-01-01', '2020-01-05T11:22:33', '20 Greatest Hits',
1982, 16, 'Hey Jude');

INSERT INTO tracks_by_user (id, month, timestamp, album_title, album_year, number, title)
VALUES (12345678-aaaa-bbbb-cccc-123456789abc, '2020-09-01', '2020-09-15T09:00:00', '20 Greatest Hits',
1982, 16, 'Hey Jude');

INSERT INTO tracks_by_user (id, month, timestamp, album_title, album_year, number, title)
VALUES (12345678-aaaa-bbbb-cccc-123456789abc, '2020-09-01', '2020-09-15T16:41:10', 'Legendary Concert
Performances', 1978, 6, 'Johnny B. Goode');

INSERT INTO tracks_by_user (id, month, timestamp, album_title, album_year, number, title)
VALUES (12345678-aaaa-bbbb-cccc-123456789abc, '2020-09-01', '2020-09-15T16:44:56', 'The Beatles 1967-1970',
1973, 17, 'Come Together');

INSERT INTO tracks_by_user (id, month, timestamp, album_title, album_year, number, title)
VALUES (12345678-aaaa-bbbb-cccc-123456789abc, '2020-09-01', '2020-09-15T21:13:13', 'Dark Side Of The Moon',
1973, 3, 'Time');
```

Étape 5

Trouver l'artiste `The Beatles`:

```
SELECT * FROM music_data.performers
WHERE name = 'The Beatles';
```

Étape 6

Trouver les albums par `The Beatles` par année descendante:

```
SELECT * FROM music_data.albums_by_performer
WHERE performer = 'The Beatles'
ORDER BY year DESC;
```

Étape 7

Trouver l'album `Magical Mystery Tour` sorti en 1967:

```
SELECT * FROM albums_by_title
WHERE title = 'Magical Mystery Tour'
    AND year = 1967;
```

Étape 8

Lister les albums nommés `20 Greatest Hits` par année descendante:

```
SELECT * FROM albums_by_title
WHERE title = '20 Greatest Hits'
ORDER BY year DESC;
```

Étape 9

Lister les albums du genre `Classical` par année descendante:

```
SELECT * FROM albums_by_genre
WHERE genre = 'Classical'
ORDER BY year DESC;
```

Étape 10

Trouver les pistes nommées `Let It Be`:

```
SELECT * FROM tracks_by_title
WHERE title = 'Let It Be';
```

Étape 11

Trouver les pistes de `Magical Mystery Tour` sorti en 1967 par numéro de piste ascendant:

```
SELECT * FROM tracks_by_album
WHERE album_year = 1967
    AND album_title = 'Magical Mystery Tour'
ORDER by number ASC;
```

Étape 12

Trouver l'utilisateur `12345678-aaaa-bbbb-cccc-123456789abc`:

```
SELECT * FROM users
WHERE id = 12345678-aaaa-bbbb-cccc-123456789abc;
```

Étape 13

Trouver toutes les pistes jouées par l'utilisateur `12345678-aaaa-bbbb-cccc-123456789abc` en septembre 2020 par horodatage descendant:

```
SELECT * FROM tracks_by_user
WHERE id = 12345678-aaaa-bbbb-cccc-123456789abc
    AND month = '2020-09-01';
```