Chapter 7

October 30, 2019

```
In [1]: %reload_ext sql
```

```
In [2]: %sql postgresql://postgres:postgres@localhost:5432/analysis
```

```
Out[2]: 'Connected: postgres@analysis'
```

1 Table Design that Works for You

1.1 Primary Key Syntax

```
In [3]: %%sql

        CREATE TABLE natural_key_example (
            licence_id varchar(10) CONSTRAINT license_key PRIMARY KEY,
            first_name varchar(10),
            last_name varchar(50)
        );
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[3]: []
```

- we first use the column constraint syntax to declare **licence_id** as the primary key
- followed by aname for the primary key and then the keywords **PRIMARY KEY**
- an advantage of this is it is easy to understand at a glance which colum is the primary key

```
In [4]: %%sql

        DROP TABLE natural_key_example;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[4]: []
```

- we simply delete the table

```
In [5]: %%sql

        CREATE TABLE natural_key_example (
            licence_id varchar(10),
            first_name varchar(10),
            last_name varchar(50),
            CONSTRAINT license_key PRIMARY KEY (licence_id)
        );

 * postgresql://postgres:***@localhost:5432/analysis
Done.


Out[5]: []
```

- here we use the CONSTRAINT after listing the final column
- we also give here the column which we want to declare as the key after giving the name of the key and the keyword PRIMARY KEY
- if you want to use more than one primary key you must use this syntax
- you then declare each primary key with commas **(composite primary key)**

```
In [6]: %%sql

        INSERT INTO natural_key_example (licence_id,
                                                first_name, last_name)
        VALUES ('T229901', 'Lynn', 'Malero');

 * postgresql://postgres:***@localhost:5432/analysis
1 rows affected.


Out[6]: []
```

- we insert some data into our created tabe

```
In [7]: %%sql

        SELECT * FROM natural_key_example;

 * postgresql://postgres:***@localhost:5432/analysis
1 rows affected.


Out[7]: [('T229901', 'Lynn', 'Malero')]
```

- if we do the same operation (inserting some values into our table)
- since the id has to be unique, we will get an error
- a unique primary key protects the table from runining the integrity of the data

1.2 Creating a Composite Primary Key

```
In [10]: %%sql

         CREATE TABLE natural_key_composite_example (
             student_id varchar(10),
             school_day date,
             present boolean,
             CONSTRAINT student_key PRIMARY KEY (student_id, school_day)
         );
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[10]: []
```

- if we want to create a composition of multiple columns as one primary key we this with the *composite primary key*
- the combination of unique student id and date column the student was in school each day during a school year
- the present colum of the type boolean represents if the student was there (present) in that day

1.3 Creating an Auto-Incrementing Surrogate Key

```
In [11]: %%sql

         CREATE TABLE surrogate_key_example (
             order_number bigserial,
             product_name varchar(50),
             order_date date,
             CONSTRAINT order_key PRIMARY KEY (order_number)
         );
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[11]: []
```

```
In [12]: %%sql

         INSERT INTO surrogate_key_example (product_name, order_date)
         VALUES ('Beachball Polish', '2015-03-17'),
             ('Wrinkle De-Atomizer', '2017-05-22'),
             ('Flux Capacitor', '1985-10-26');
```

```
 * postgresql://postgres:***@localhost:5432/analysis
3 rows affected.
```

```
Out[12]: []
```

```
In [13]: %%sql

         SELECT * FROM surrogate_key_example;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
3 rows affected.
```

```
Out[13]: [(1, 'Beachball Polish', datetime.date(2015, 3, 17)),
          (2, 'Wrinkle De-Atomizer', datetime.date(2017, 5, 22)),
          (3, 'Flux Capacitor', datetime.date(1985, 10, 26))]
```

- we used bigserial for autoincrementing the primary key
- when you insert data into the table you can omit the order_number column
- it will be added automatically and incremented also automatically

1.3.1 Foreign Keys

```
In [34]: %%sql

         CREATE TABLE licenses (
             license_id varchar(10),
             first_name varchar(50),
             last_name varchar(50),
             CONSTRAINT licenses_key PRIMARY KEY (license_id)
         );
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[34]: []
```

```
In [35]: %%sql

         CREATE TABLE registrations (
             registration_id varchar(10),
             registration_date date,
             license_id varchar(10) REFERENCES licenses (license_id),
             CONSTRAINT registration_key PRIMARY KEY
             (registration_id, license_id)
         );
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[35]: []
```

```
In [36]: %%sql

         INSERT INTO licenses (license_id, first_name, last_name)
         VALUES ('T229901', 'Lynn', 'Malero');

 * postgresql://postgres:***@localhost:5432/analysis
1 rows affected.


Out[36]: []
```

```
In [ ]: %%sql

        INSERT INTO registrations (registration_id, registration_date, license_id)
        VALUES ('A203391', '3/17/2017', 'T229901');
```

```
In [ ]: %%sql

        INSERT INTO registrations (registration_id, registration_date, license_id)
        VALUES ('A75772', '3/17/2017', 'T000001');
```

- the last insert statement had an licence_id which was not in the licences table
- ourforeign key was not existed so we get an error
- this is somehow good, because we can guarantee the integrity of the data

1.3.2 Automatically Deleting Related Records with CASCADE

- to delete a row in licenses and have that action automatically delete any relted rows in registrations
  - we can specify this with
    - **ON DELETE CASCADE**

1.3.3 The CHECK Constraint

```
In [43]: %%sql

         CREATE TABLE check_constraint_example (
             user_id bigserial,
             user_role varchar(50),
             salary integer,
             CONSTRAINT user_id_key PRIMARY KEY (user_id),
             CONSTRAINT check_role_in_list CHECK (user_role IN('Admin', 'Staff')),
             CONSTRAINT check_salary_not_zero CHECK (salary > 0)
         );

 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

Out[43]: []

- a **CHECK** constraint evaluates whether the data added to a column meets the expected criteria which we specify with a logical test
- if this criteria aren't met, we get an error
- it can prevent columns having nonsencial data (such as a grade 'Z')
  - **CHECK (logical expression)** after all columns are defined
- we can also combine more than one logical test with **AND**

1.3.4 The UNIQUE Constraint

In [49]:
```
%%sql

        CREATE TABLE unique_constraint_example (
            contact_id bigserial CONSTRAINT contact_id_key PRIMARY KEY,
            first_name varchar(50),
            last_name varchar(50),
            email varchar(200),
            CONSTRAINT emaiol_unique UNIQUE (email)
        );
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

Out[49]: []

In [50]:
```
%%sql

        INSERT INTO unique_constraint_example (first_name, last_name, email)
        VALUES ('Samantha', 'Lee', 'slee@example.org');
```

```
 * postgresql://postgres:***@localhost:5432/analysis
1 rows affected.
```

Out[50]: []

In [51]:
```
%%sql

        INSERT INTO unique_constraint_example (first_name, last_name, email)
        VALUES ('Betty', 'Diaz', 'bdiaz@example.org');
```

```
 * postgresql://postgres:***@localhost:5432/analysis
1 rows affected.
```

Out[51]: []

```
In [ ]: %%sql

        INSERT INTO unique_constraint_example (first_name, last_name, email)
        VALUES ('Sasha', 'Lee', 'slee@example.org');
```

- this will occur an error
- the email column we expect this to be unique
- those adresses change over time and can be null
- we use **UNIQUE** to ensure to have one email adress for one contact
- the main difference between this and the primary key is, that this value can be null

1.3.5 The NOT NUILL Constraint

```
In [55]: %%sql

         CREATE TABLE not_null_example (
             student_id bigserial,
             first_name varchar(50) NOT NULL,
             last_name varchar(50) NOT NULL,
             CONSTRAINT student_id_key PRIMARY KEY (student_id)
         );

 * postgresql://postgres:***@localhost:5432/analysis
Done.


Out[55]: []
```

- Sometimes we want columns not be null
- we declare this with the **NOT NULL** keyword
- if we attend an insert for the table and don't include values for those columns, the database will notify us the violation

1.3.6 Removing Constraints or Adding Them Later

- to remove a priamary key, foreign key, or a unique constraint you would use this stament:
  - ALTER TABLE table_name DROP CONSTRAINT constraint_name;
- to drop a NOt NULL constraint, we have to use the ALTER COLUMN statement also:
  - ALTER TABLE table_name ALTER COLUMN column_name DROP NOT NULL;

```
In [60]: %%sql

         ALTER TABLE not_null_example DROP CONSTRAINT student_id_key;

 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[60]: []
```

- we drop the primary key

```
In [62]: %%sql

         ALTER TABLE not_null_example ADD CONSTRAINT student_id_key PRIMARY KEY (student_id);

 * postgresql://postgres:***@localhost:5432/analysis
(psycopg2.errors.InvalidTableDefinition) multiple primary keys for table "not_null_example" are

[SQL: ALTER TABLE not_null_example ADD CONSTRAINT student_id_key PRIMARY KEY (student_id);]
(Background on this error at: http://sqlalche.me/e/f405)
```

- we add a new primary key

```
In [64]: %%sql

         ALTER TABLE not_null_example ALTER COLUMN first_name DROP NOT NULL;

 * postgresql://postgres:***@localhost:5432/analysis
Done.


Out[64]: []
```

- we drop a colum which is NOT NULL

```
In [59]: %%sql

         ALTER TABLE not_null_example ALTER COLUMN first_name SET NOT NULL;

 * postgresql://postgres:***@localhost:5432/analysis
Done.


Out[59]: []
```

- we set a new column which is NOT NULL

1.4 Speeding Up Queries with Indexes

- In the same way a book indexes, SQL indexes are working the same
- the primary key can be seen as an Index too
- they are many other index types which will be handled now

1.4.1 B-Tree: PostgreSQL's Default Index

```
In [65]: %%sql

         CREATE TABLE new_york_adresses (
             longtitude numeric(9,6),
             latitude numeric(9,6),
             street_number varchar(10),
             street varchar(32),
             unit varchar(7),
             postcode varchar(5),
             id integer CONSTRAINT new_york_key PRIMARY KEY
         );

 * postgresql://postgres:***@localhost:5432/analysis
Done.


Out[65]: []
```

```
In [ ]: %%sql

        COPY new_york_adresses
        FROM '/Users/ugurtigu/Documents/Learn/Docs/SQL/city_of_new_york.csv'
        WITH (FORMAT CSV, HEADER);
```

```
In [77]: %%sql

         SELECT * FROM new_york_adresses
         LIMIT 10;

 * postgresql://postgres:***@localhost:5432/analysis
10 rows affected.


Out[77]: [(Decimal('-73.939546'), Decimal('40.725332'), '608', 'MORGAN AVENUE', None, '11222',
          (Decimal('-73.923303'), Decimal('40.692213'), '962', 'BUSHWICK AVENUE', None, '11221
          (Decimal('-73.918562'), Decimal('40.700381'), '309', 'HARMAN STREET', None, '11237',
          (Decimal('-73.923921'), Decimal('40.693764'), '1115', 'GREENE AVENUE', None, '11221'
          (Decimal('-73.918152'), Decimal('40.699605'), '1341', 'GREENE AVENUE', None, '11237'
          (Decimal('-73.916312'), Decimal('40.701468'), '1415', 'GREENE AVENUE', None, '11237'
          (Decimal('-73.937288'), Decimal('40.704293'), '185', 'MOORE STREET', None, '11206', 3
          (Decimal('-73.938597'), Decimal('40.704417'), '3', 'BUSHWICK COURT', None, '11206', 3
          (Decimal('-73.941511'), Decimal('40.703775'), '100', 'MOORE STREET', None, '11206', 3
          (Decimal('-73.960914'), Decimal('40.704585'), '135', 'RODNEY STREET', None, '11211',
```

Benchmarking Query Performance with Explain

- We will measure the performance of the query with the EXPLAIN keyword

```
In [82]: %%sql

         EXPLAIN ANALYSE SELECT * FROM new_york_adresses
         WHERE street = 'BROADWAY';

 * postgresql://postgres:***@localhost:5432/analysis
7 rows affected.


Out[82]: [('Bitmap Heap Scan on new_york_adresses  (cost=76.23..6368.64 rows=3072 width=46) (ac
          ("  Recheck Cond: ((street)::text = 'BROADWAY'::text)",),
          ('  Heap Blocks: exact=2157',),
          ('  ->  Bitmap Index Scan on street_idx  (cost=0.00..75.46 rows=3072 width=0) (actual
          ("        Index Cond: ((street)::text = 'BROADWAY'::text)",),
          ('Planning Time: 0.666 ms',),
          ('Execution Time: 16.249 ms',)]

In [83]: %%sql

         EXPLAIN ANALYSE SELECT * FROM new_york_adresses
         WHERE street = '52 STREET';

 * postgresql://postgres:***@localhost:5432/analysis
7 rows affected.


Out[83]: [('Bitmap Heap Scan on new_york_adresses  (cost=5.63..563.21 rows=155 width=46) (actua
          ("  Recheck Cond: ((street)::text = '52 STREET'::text)",),
          ('  Heap Blocks: exact=704',),
          ('  ->  Bitmap Index Scan on street_idx  (cost=0.00..5.59 rows=155 width=0) (actual t
          ("        Index Cond: ((street)::text = '52 STREET'::text)",),
          ('Planning Time: 0.163 ms',),
          ('Execution Time: 5.701 ms',)]

In [84]: %%sql

         EXPLAIN ANALYSE SELECT * FROM new_york_adresses
         WHERE street = 'ZWICKY AVENUE';

 * postgresql://postgres:***@localhost:5432/analysis
7 rows affected.


Out[84]: [('Bitmap Heap Scan on new_york_adresses  (cost=5.63..563.21 rows=155 width=46) (actua
          ("  Recheck Cond: ((street)::text = 'ZWICKY AVENUE'::text)",),
          ('  Heap Blocks: exact=6',),
          ('  ->  Bitmap Index Scan on street_idx  (cost=0.00..5.59 rows=155 width=0) (actual t
          ("        Index Cond: ((street)::text = 'ZWICKY AVENUE'::text)",),
          ('Planning Time: 0.160 ms',),
          ('Execution Time: 0.700 ms',)]
```

- we have a parallel seq scan which means that the table will be scanned fully

Adding the Index

```
In [81]: %%sql

         CREATE INDEX street_idx ON new_york_adresses (street);
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[81]: []
```

- similar to creating constraints we use **CREATE INDEX** keywords followed by a name we chose for the index (street_idx)
- than **ON** is added follwing by the target table and column
- this statement will scan the values in the street column and build the index from them
- when we now run the 3 statement we did with the EXPLAIN keyword we can see a better performance
- instead of a seq scan we now have a index scan on street_idx
- instead of visiting each row

– Consider the following two tables from a database you're making to keep – track of your vinyl LP collection. Start by reviewing these CREATE TABLE – statements.

– The albums table includes information specific to the overall collection – of songs on the disc. The songs table catalogs each track on the album. – Each song has a title and its own artist column, because each song might. – feature its own collection of artists.

CREATE TABLE albums (album_id bigserial, album_catalog_code varchar(100), album_title text, album_artist text, album_time interval, album_release_date date, album_genre varchar(40), album_description text);

CREATE TABLE songs (song_id bigserial, song_title text, song_artist text, album_id bigint);

– Use the tables to answer these questions:

– 1. Modify these CREATE TABLE statements to include primary and foreign keys – plus additional constraints on both tables. Explain why you made your – choices.

```
In [110]: %%sql

          CREATE TABLE albums (
              album_id bigserial,
              album_catalog_code varchar(100) NOT NULL,
              album_title text NOT NULL,
              album_artist text NOT NULL,
              album_release_date date,
              album_genre varchar(40),
              album_description text,
              CONSTRAINT album_id_key PRIMARY KEY (album_id),
              CONSTRAINT release_date_check CHECK (album_release_date > '1/1/1925')
          );
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[110]: []
```

```
In [111]: %%sql

          CREATE TABLE songs (
              song_id bigserial,
              song_title text NOT NULL,
              song_artist text NOT NULL,
              album_id bigint REFERENCES albums (album_id),
              CONSTRAINT song_id_key PRIMARY KEY (song_id)
          );

 * postgresql://postgres:***@localhost:5432/analysis
Done.


Out[111]: []
```

- Both tables get a primary key using surrogate key id values that are auto-generated via serial data types.
- The songs table references albums via a foreign key constraint. Note that the reference is a bigint (not a serial anymore)
- In both tables, the title and artist columns cannot be empty, which is specified via a NOT NULL constraint. We assume that every album and song should at minimum have that information.
- In albums, the album_release_date column has a CHECK constraint because it would be likely impossible for us to own an LP made before 1925.

```
In [ ]:
```