1. Typical type of constraints in postgresql:
2. Not NULL:

CREATE TABLE products (

product\_no integer **NOT NULL,**

name text **NOT NULL,**

price numeric

);

1. Multi constraints for one column:

CREATE TABLE products (

product\_no integer **NOT NULL,**

name text **NOT NULL,**

price numeric **NOT NULL CHECK (price > 0)**

);

1. Unique Constraints:

CREATE TABLE products (

product\_no integer **UNIQUE**,

name text,

price numeric

);

multiple column unique:

CREATE TABLE example (

a integer,

b integer,

c integer,

**UNIQUE (a, c)**

);

1. Primary Keys:

CREATE TABLE products (

product\_no integer **PRIMARY KEY,**

name text,

price numeric

);

CREATE TABLE example (

a integer,

b integer,

c integer,

**PRIMARY KEY (a, c)**

);

1. Foreign Keys:

CREATE TABLE orders (

order\_id integer PRIMARY KEY,

product\_no integer **REFERENCES** products (product\_no),

quantity integer

);

CREATE TABLE t1 (

a integer PRIMARY KEY,

b integer,

c integer,

**FOREIGN KEY (b, c) REFERENCES** other\_table (c1, c2)

);

1. Check constraints

CREATE TABLE products (

product\_no integer,

name text,

price numeric **CHECK** (price > 0)

);

CREATE TABLE products (

product\_no integer,

name text,

price numeric **CONSTRAINT positive\_price CHECK (price > 0)**

);

1. Task to do for project3:
2. parser: since peloton does not support any constraints before, the first thing we need to add a new operator: CONSTRAINT and CHECK; then build the syntax tree and get the parameters of constraints
3. handle simple task first: primary key, not null, unique
4. handle complex task: check
5. handle most complex task: foreign key
6. change file:

src/parser/sql\_scanner.l

+ CHECK TOKEN(CHECK)

+ EXCULDE TOKEN(CHECK)

src/storage/data\_table.cpp:

bool DataTable::CheckConstraints(const storage::Tuple \*tuple) const {}

bool DataTable::CheckForeignKeyConstraints(const storage::Tuple \*tuple UNUSED\_ATTRIBUTE) {}