# Basic Constraints Lab

1. Connect to [the RDS instance and coffeeshop db](file:///C:\Users\Gary\Dropbox%20(Personal)\CSCC_WD%20Innovations\A0-Nationwide%20Incumbent\Foundational%20Courses\DatabaseSQL\Git\psql-getting-started-master\labs\creating_rds_instance.md#connect-psql)
2. Load the lab schema with the following command...

* psql -h <AWS\_URL> -p <PORT> -U <USER\_NAME> <DB\_NAME> -a -f ./labs/resources/sql/basicConstraintsSetup.sql

1. We will create the first table, this table shows us the Check constraint.

* CREATE TABLE CONSTRAINTS.TEST\_CHECK(  
   NOPE SMALLINT,  
   CHECK (NOPE > 0)  
  );

1. By default types have some limitations. We can see that by trying to add too big of an integer.

* INSERT INTO CONSTRAINTS.TEST\_CHECK VALUES (5000000000000);
* Notice we see...
* ERROR: smallint out of range
* This is because most types have native constraints.

1. Now let's see what the custom constraint does. Let's try a valid query...

* INSERT INTO CONSTRAINTS.TEST\_CHECK VALUES (5);
* Notice we see a valid transaction INSERT 0 1.

1. What happens if we try to add an invalid record...

* INSERT INTO CONSTRAINTS.TEST\_CHECK VALUES (-5);
* Notice we see the following error...
* ERROR: new row for relation "test\_check" violates check constraint "test\_check\_nope\_check"

1. This error doesn't tell us too much detail. What if we had multiple checks on the same table or even attribute. Let's Remove the old constraint...

* ALTER TABLE CONSTRAINTS.TEST\_CHECK DROP CONSTRAINT TEST\_CHECK\_NOPE\_CHECK;
* And add a couple new ones that have better names...
* ALTER TABLE CONSTRAINTS.TEST\_CHECK   
  ADD CONSTRAINT TOO\_LOW CHECK (NOPE > 0),  
  ADD CONSTRAINT TOO\_HIGH CHECK (NOPE < 10);
* Notice the ADD/DROP CONSTRAINT. This is pretty consistent with other SQL conventions.

1. Now let's try adding a couple more test cases...

* INSERT INTO CONSTRAINTS.TEST\_CHECK VALUES (-5);
* Should now return
* ERROR: new row for relation "test\_check" violates check constraint "too\_low" DETAIL: Failing row contains (-5).
* Notice the too\_low. Now let's try too high...
* INSERT INTO CONSTRAINTS.TEST\_CHECK VALUES (15);
* ERROR: new row for relation "test\_check" violates check constraint "too\_high" DETAIL: Failing row contains (15).
* Names are an extremely important part of writing maintainable code.

1. Now let's add another attribute and create a NOT NULL constraint on that attribute.

* ALTER TABLE CONSTRAINTS.TEST\_CHECK  
  ADD COLUMN YUP SMALLINT NOT NULL;
* but...
* ERROR: column "yup" contains null values
* Thats because of the leftover value in the table, we can see this by issuing a select query.
* SELECT \* FROM CONSTRAINTS.TEST\_CHECK;

nope

5 (1 row)

In the real world we would have to jump through a bunch of hoops to make this fit without losing data, however, since we are in a dev environment let's just blow the data away.

DELETE FROM CONSTRAINTS.TEST\_CHECK;

DELETE 1

Then try running it again, this time you should see ALTER TABLE

1. Now let's try adding an both an invalid and a valid entry.

* INSERT INTO CONSTRAINTS.TEST\_CHECK (NOPE) VALUES (5);
* ERROR: null value in column "yup" violates not-null constraint DETAIL: Failing row contains (15, null).
* INSERT INTO CONSTRAINTS.TEST\_CHECK VALUES (5, 5);
* INSERT 0 1

1. Finally we will look at the UNIQUE constraint. This constraint ensures that the value is only in the table once.

* ALTER TABLE CONSTRAINTS.TEST\_CHECK ADD CONSTRAINT MUST\_BE\_UNIQUE UNIQUE (NOPE, YUP);
* Now let's try adding a second record with the same values
* INSERT INTO CONSTRAINTS.TEST\_CHECK VALUES (5, 5);
* ERROR: duplicate key value violates unique constraint "must\_be\_unique" DETAIL: Key (nope, yup)=(5, 5) already exists.

1. Since we are done with this lab let's clean up

* DROP SCHEMA IF EXISTS CONSTRAINTS CASCADE;