PostgreSQL

And Relational Databases

Install

- http://postgresapp.com/
- https://eggerapps.at/postico/

Post-Install

Add the line below to your `~/.bash_profile`

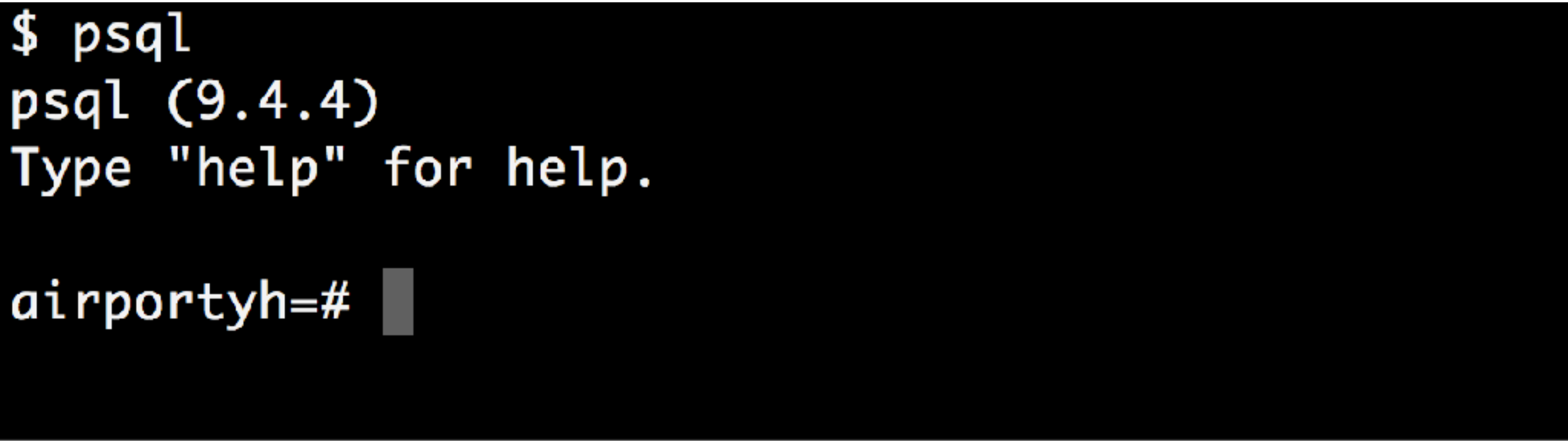
```
export PATH=$PATH:/Applications/Postgres.app/
Contents/Versions/9.6/bin
```

Restart your shell and type `psql` in your terminal

3 Ways to Execute SQL

- The `psql` shell
- Use `psql` to run a .sql file
- Use Postico

The `psql` Shell

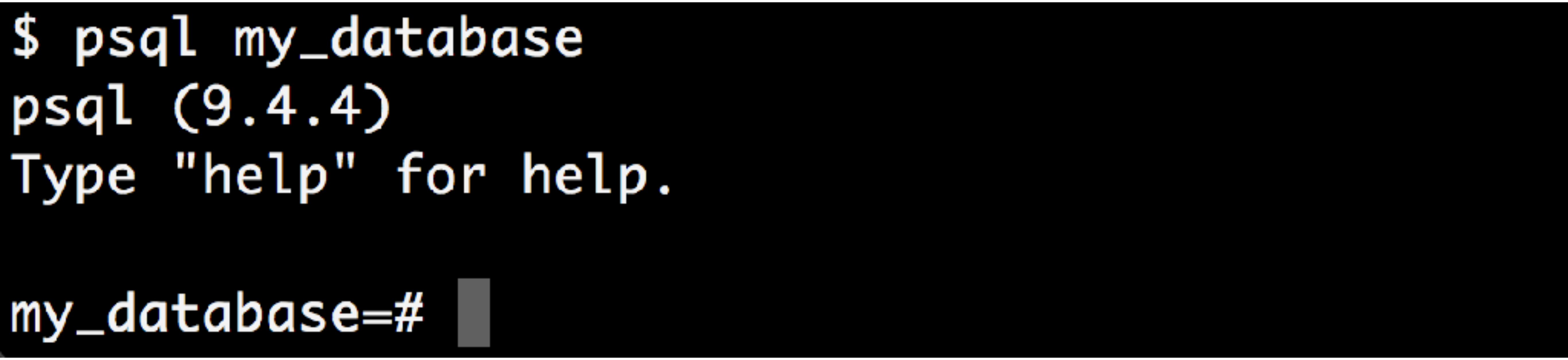


```
$ psql
psql (9.4.4)
Type "help" for help.

airportyh=#
```

Like your bash shell, or the Python shell,
but you type SQL in it

The `psql` Shell

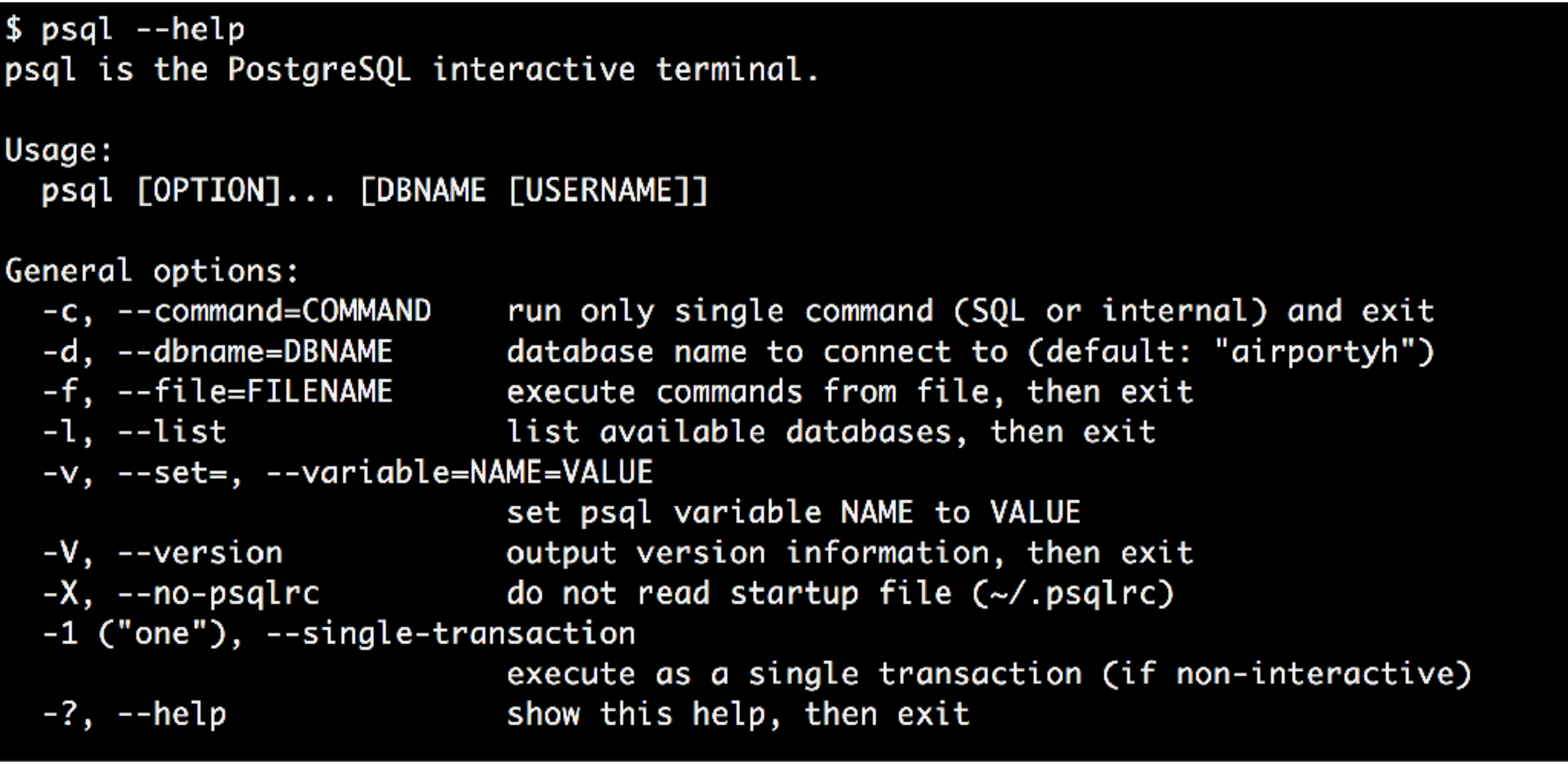


```
$ psql my_database
psql (9.4.4)
Type "help" for help.

my_database=#
```

You can put the name of your database as
a command line argument to switch that database
directly

The `psql` Shell



```
$ psql --help
psql is the PostgreSQL interactive terminal.

Usage:
  psql [OPTION]... [DBNAME [USERNAME]]

General options:
  -c, --command=COMMAND    run only single command (SQL or internal) and exit
  -d, --dbname=DBNAME      database name to connect to (default: "airportyh")
  -f, --file=FILENAME      execute commands from file, then exit
  -l, --list               list available databases, then exit
  -v, --set=, --variable=NAME=VALUE
                           set psql variable NAME to VALUE
  -V, --version            output version information, then exit
  -X, --no-psqlrc          do not read startup file (~/.psqlrc)
  -1 ("one"), --single-transaction
                           execute as a single transaction (if non-interactive)
  -?, --help               show this help, then exit
```

Use `psql` to run a .sql file

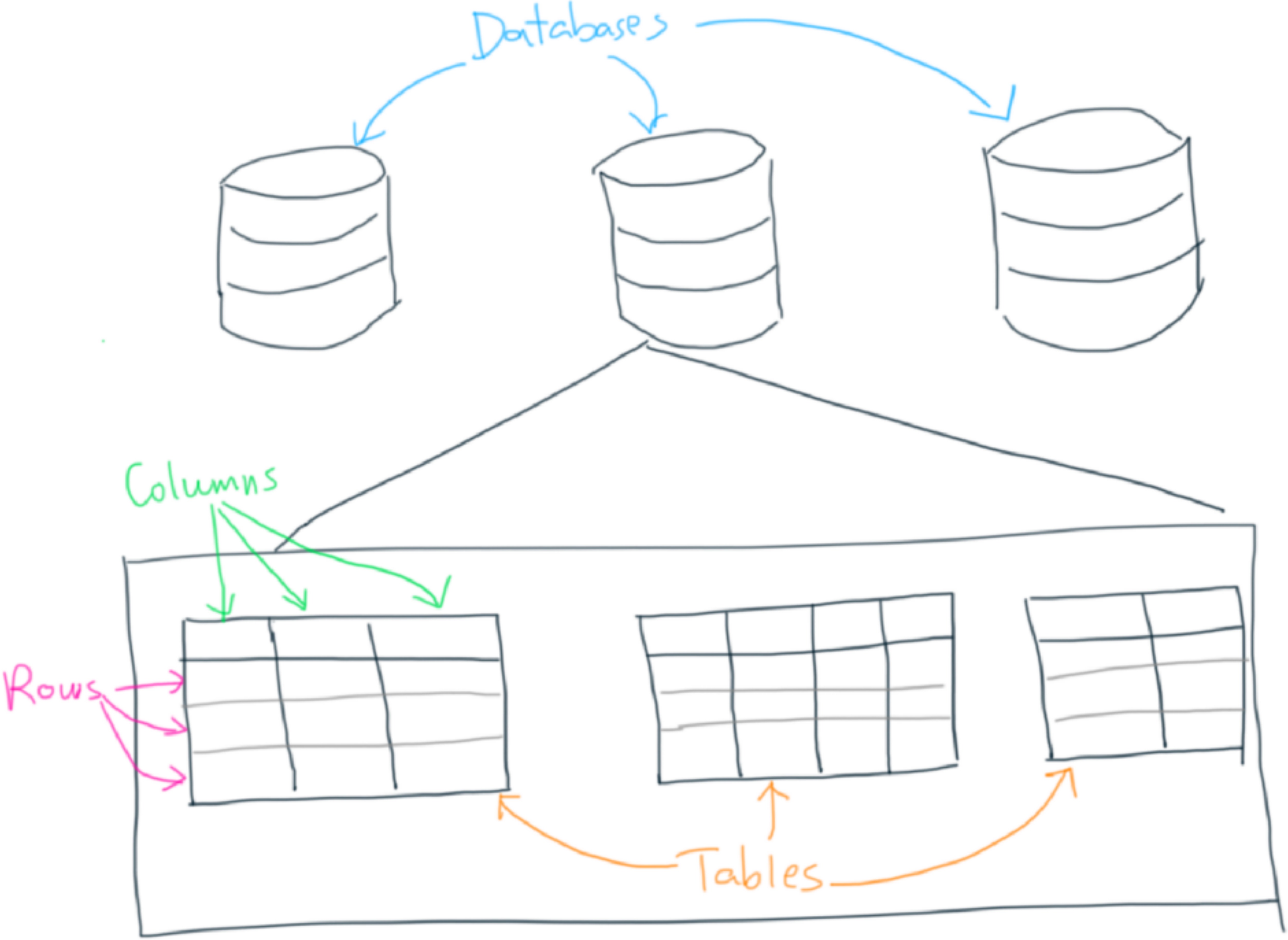
```
$ psql my_database -f my_sql_statements.sql
```

or

Copy-n-paste statements from atom into the psql shel

Database Concepts

- Databases - has many tables, like an app
- Tables - has many rows, like a class in OO
- Rows - has many columns, like an object instance in OO
- Columns / fields - individual values


Databases
Columns
Rows
Tables

SQL Statements

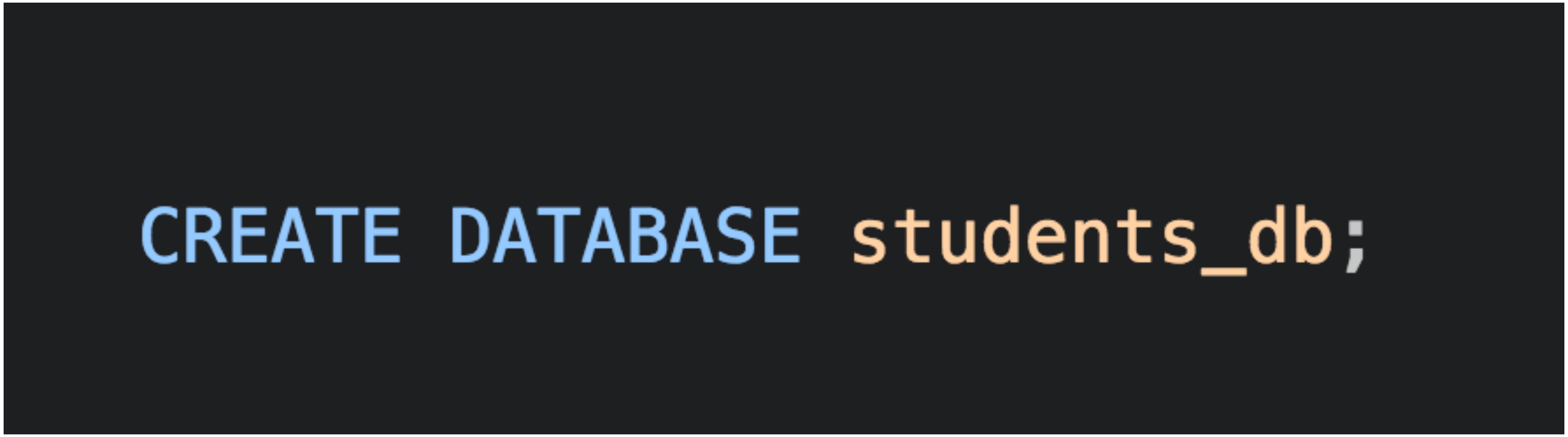
SQL Statements

- CREATE DATABASE
- CREATE TABLE
- Data Types
- INSERT
- Constraints
- Select Statement

Case Insensitivity

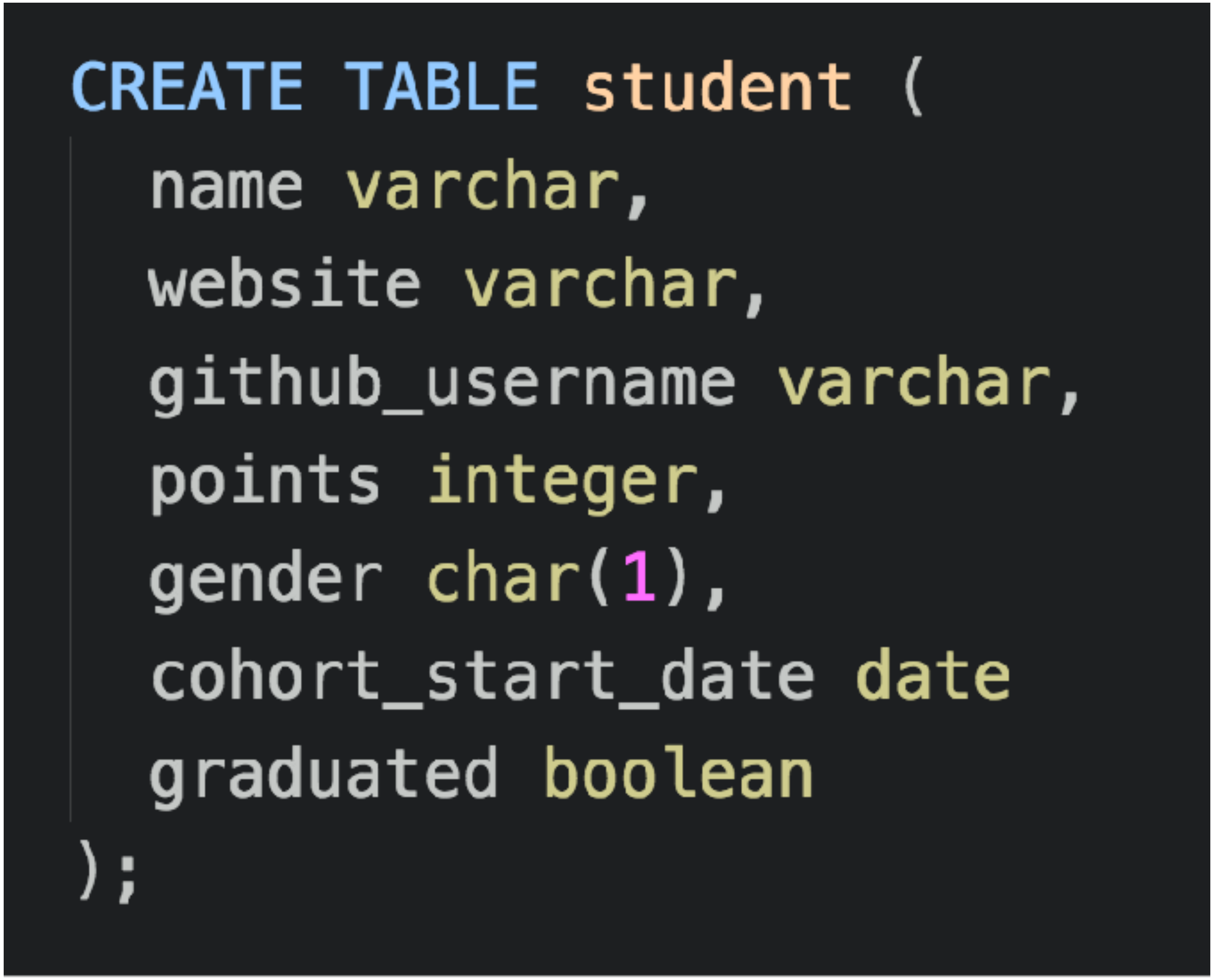
- Usually case insensitive (there are exceptions)
- `SELECT * FROM STUDENT;`
- `select * from student;`
- `sElEcT * fRoM sTuDeNt;`

CREATE DATABASE



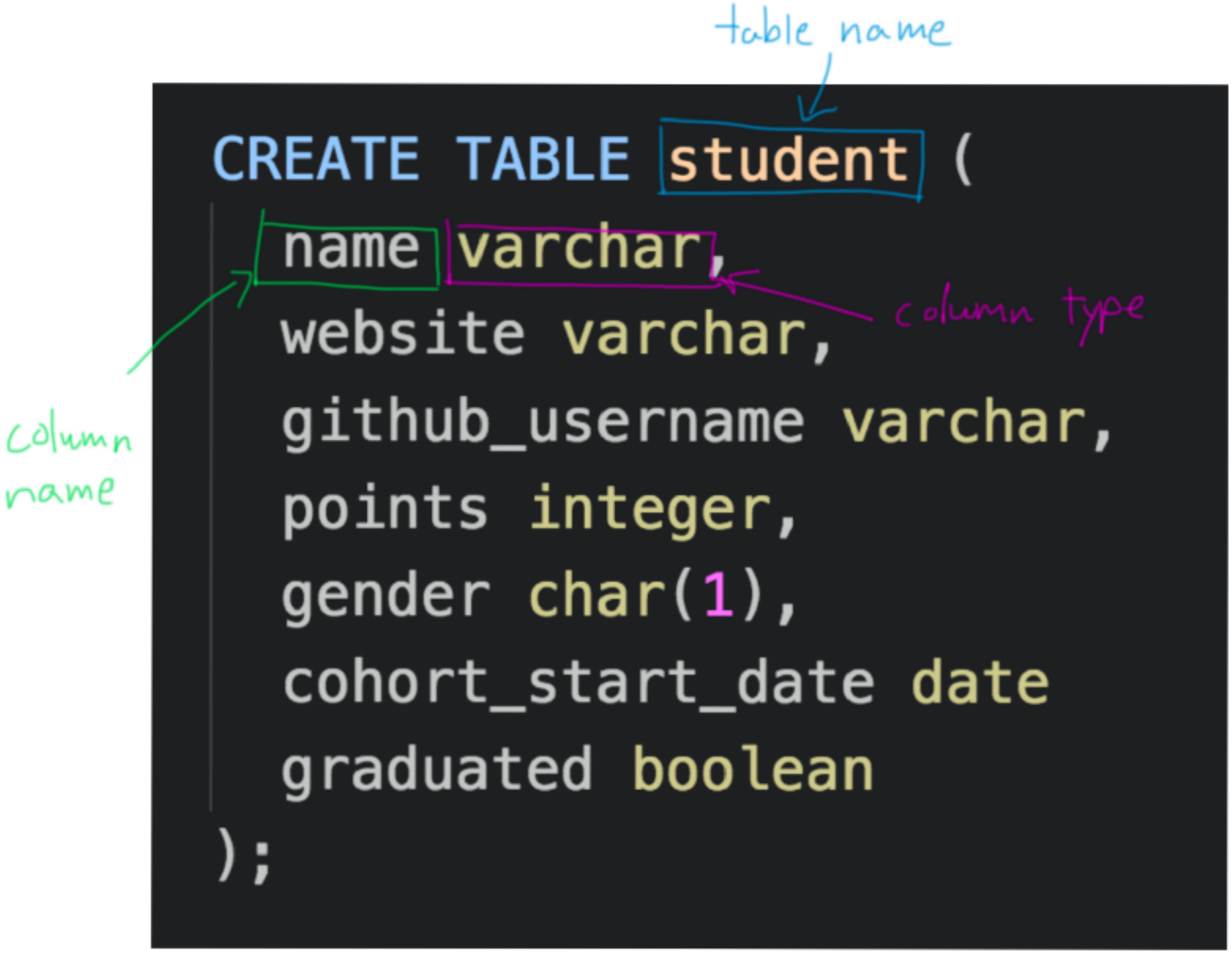
```sql
CREATE DATABASE students_db;
```

CREATE TABLE



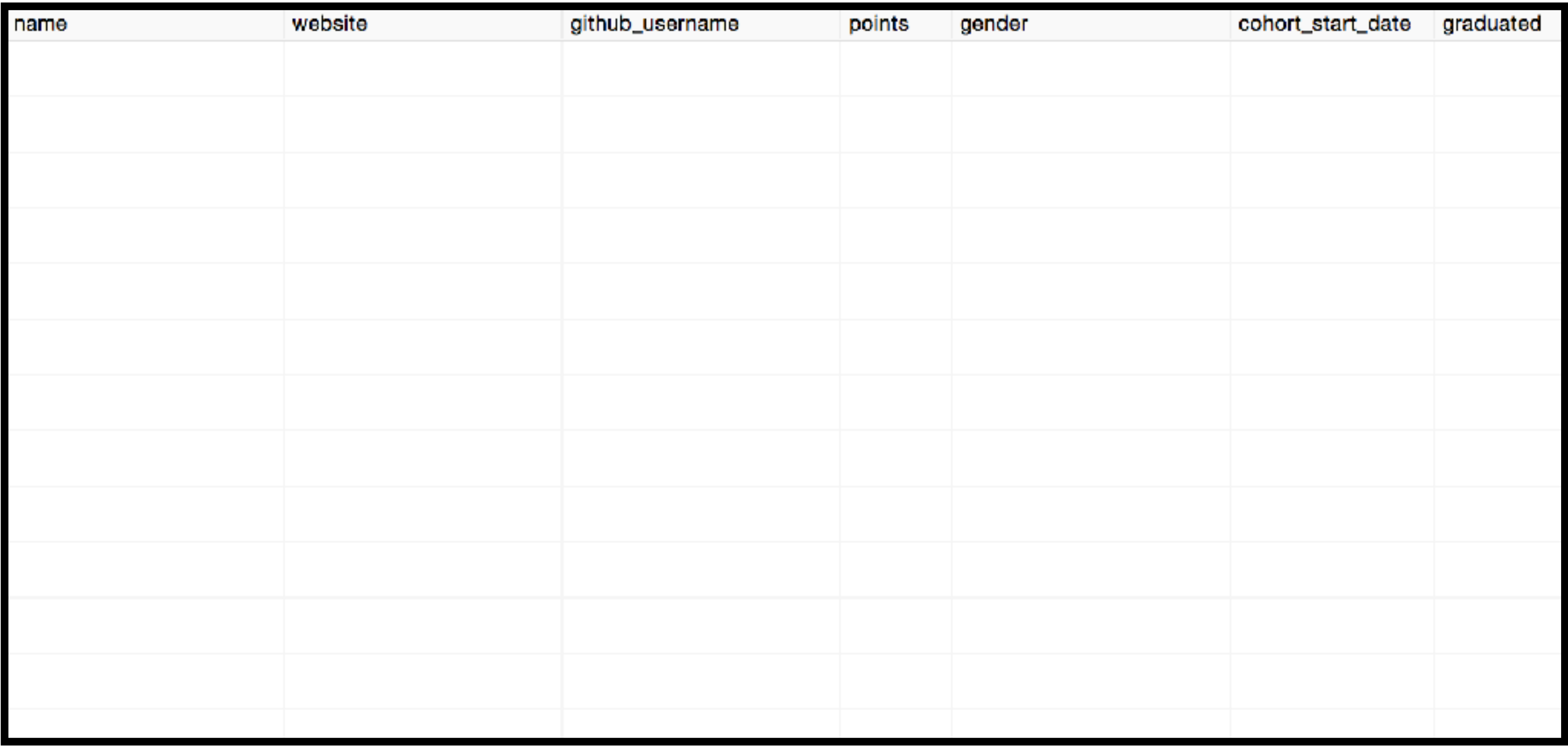
```sql
CREATE TABLE student (
  name varchar,
  website varchar,
  github_username varchar,
  points integer,
  gender char(1),
  cohort_start_date date
  graduated boolean
);
```

CREATE TABLE



```
CREATE TABLE student (
  name varchar,
  website varchar,
  github_username varchar,
  points integer,
  gender char(1),
  cohort_start_date date
  graduated boolean
);
```

Result



name	website	github_username	points	gender	cohort_start_date	graduated

Data Types

String Types

- char(n) or character(n) - fixed length strings
- varchar(n) or character varying(n) - variable length strings with length limit
- varchar or text - variable length strings with no limit
- Recommendation: just use varchar

Number Types

- integer
- numeric - precise decimal (good for currency)
- real - floating point numbers

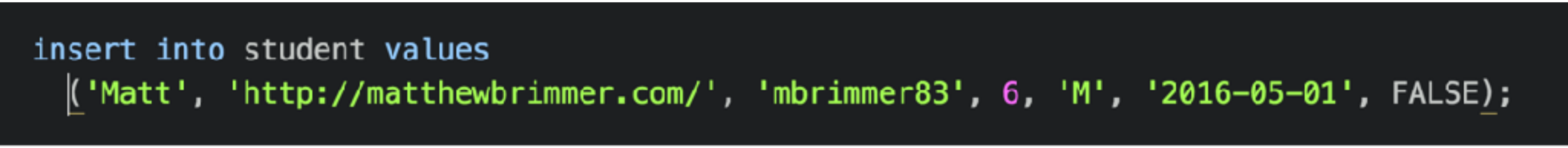
More Data Types

- date, timestamp
- boolean

Notes about Syntax

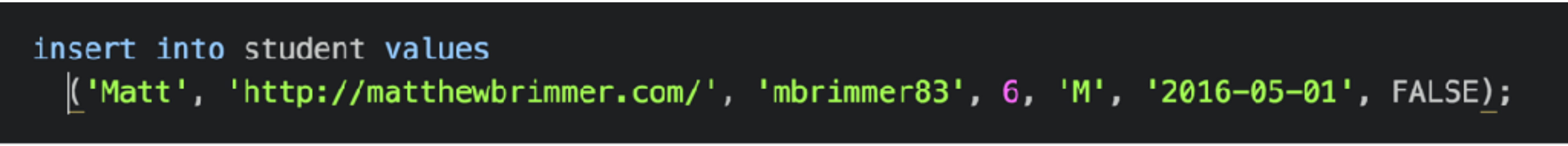
- Strings - use single quotes
- Booleans - use TRUE vs FALSE or ’t’ vs ‘f’
- Table names - use double quotes when needed (if has a dash in table name)
- Nulls - PostgreSQL’s representation of the empty value (None in Python), use NULL

Inserting Data



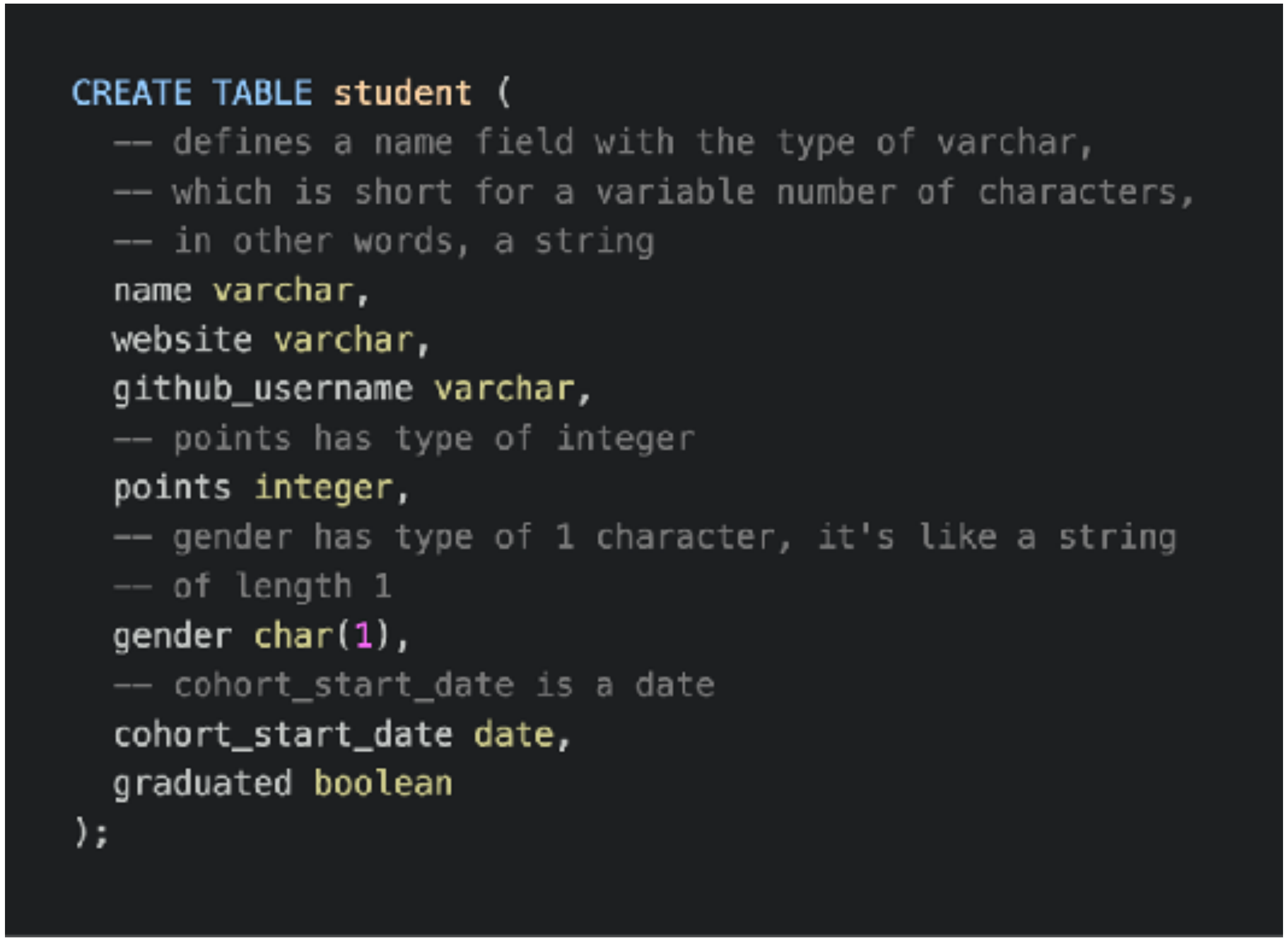
```
insert into student values
  ('Matt', 'http://matthewbrimmer.com/', 'mbrimmer83', 6, 'M', '2016-05-01', FALSE);
```

Inserting Data



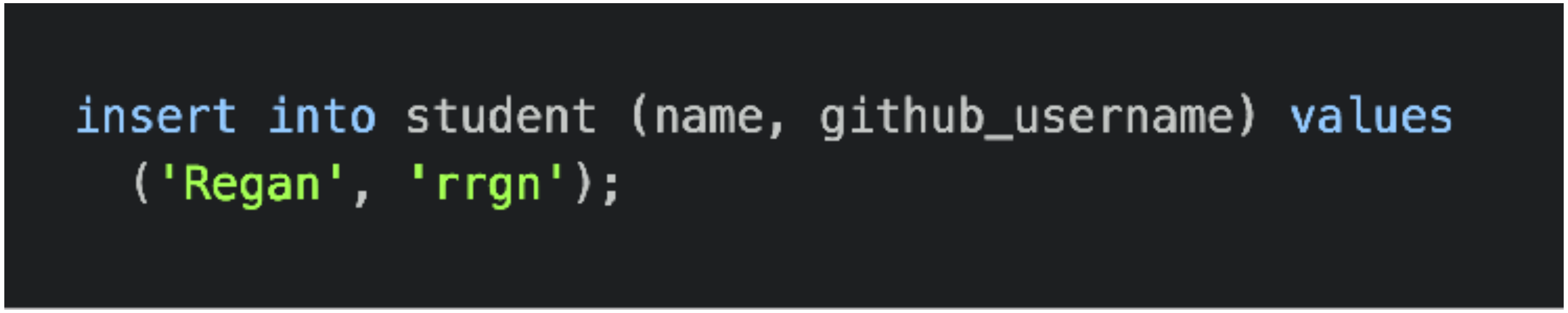
```
insert into student values
  ('Matt', 'http://matthewbrimmer.com/', 'mbrimmer83', 6, 'M', '2016-05-01', FALSE);
```

order has to match field definition:



```
CREATE TABLE student (
  -- defines a name field with the type of varchar,
  -- which is short for a variable number of characters,
  -- in other words, a string
  name varchar,
  website varchar,
  github_username varchar,
  -- points has type of integer
  points integer,
  -- gender has type of 1 character, it's like a string
  -- of length 1
  gender char(1),
  -- cohort_start_date is a date
  cohort_start_date date,
  graduated boolean
);
```

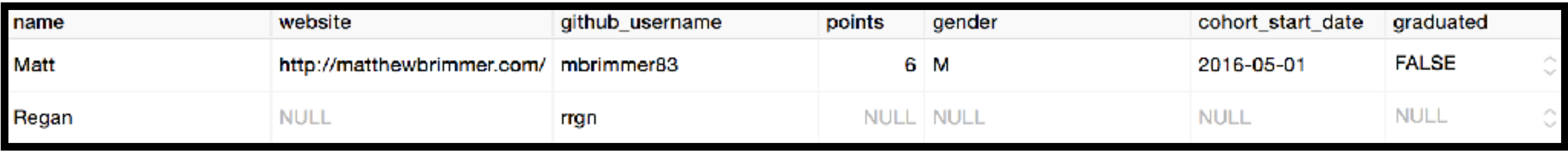
Insert specifying some fields



```
insert into student (name, github_username) values
  ('Regan', 'rrgn');
```

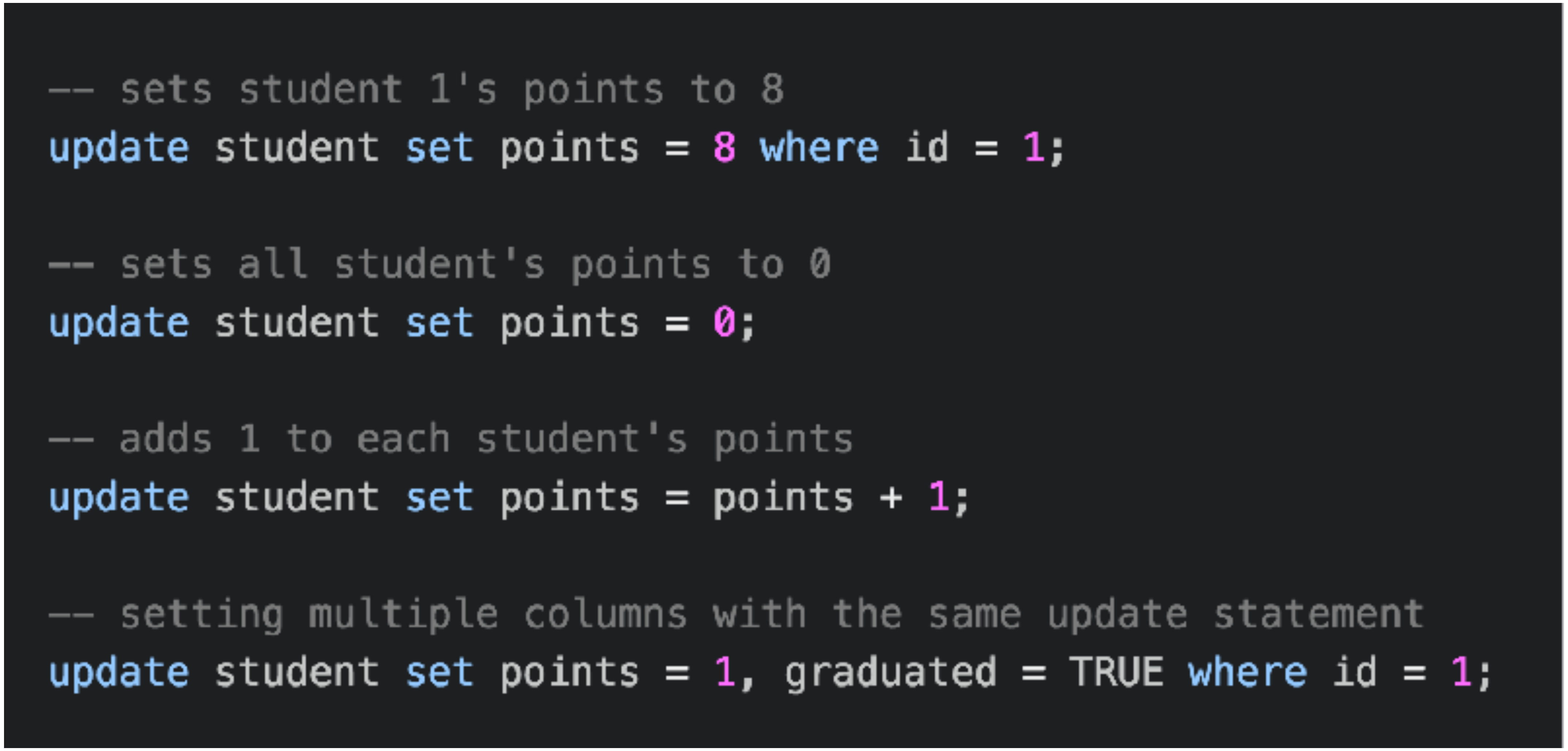
Specifies the value order in the column name list.
Leaves rest of fields empty or default values.

Insert Data



name	website	github_username	points	gender	cohort_start_date	graduated
Matt	http://matthewbrimmer.com/	mbrimmer83	6	M	2016-05-01	FALSE
Regan	NULL	rrgn	NULL	NULL	NULL	NULL

Update Statement



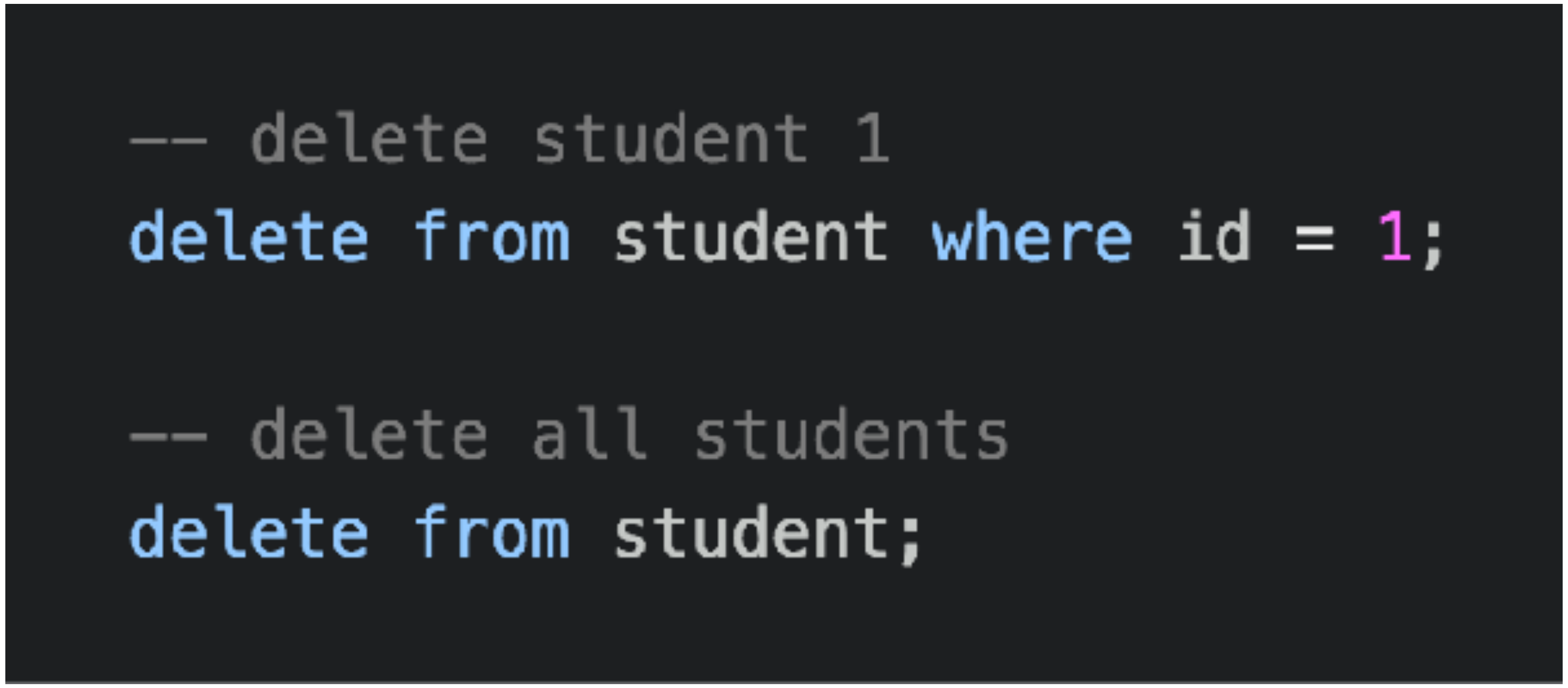
```sql
-- sets student 1's points to 8
update student set points = 8 where id = 1;

-- sets all student's points to 0
update student set points = 0;

-- adds 1 to each student's points
update student set points = points + 1;

-- setting multiple columns with the same update statement
update student set points = 1, graduated = TRUE where id = 1;
```

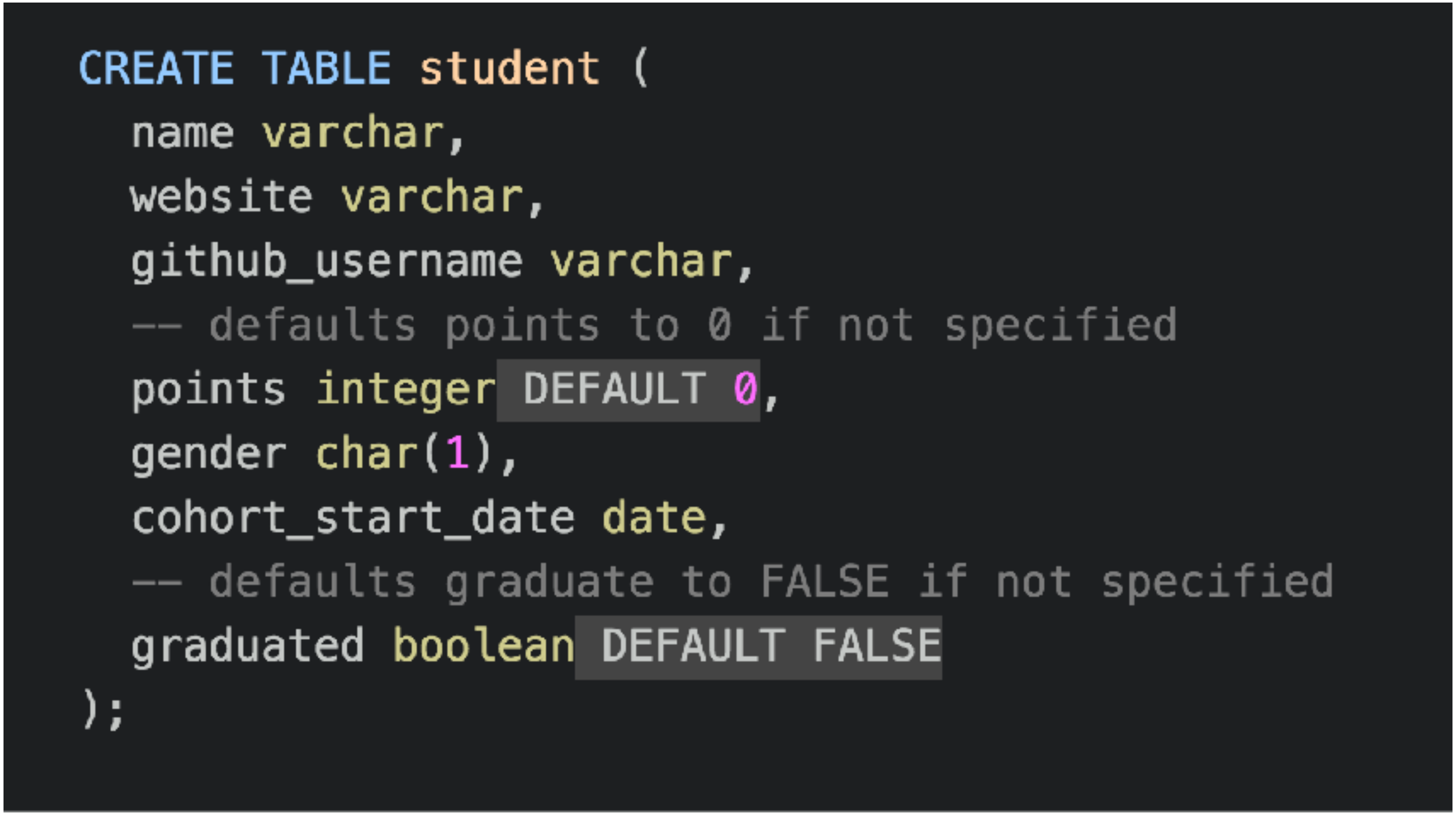
Delete Statement



```sql
-- delete student 1
delete from student where id = 1;

-- delete all students
delete from student;
```

Default Values



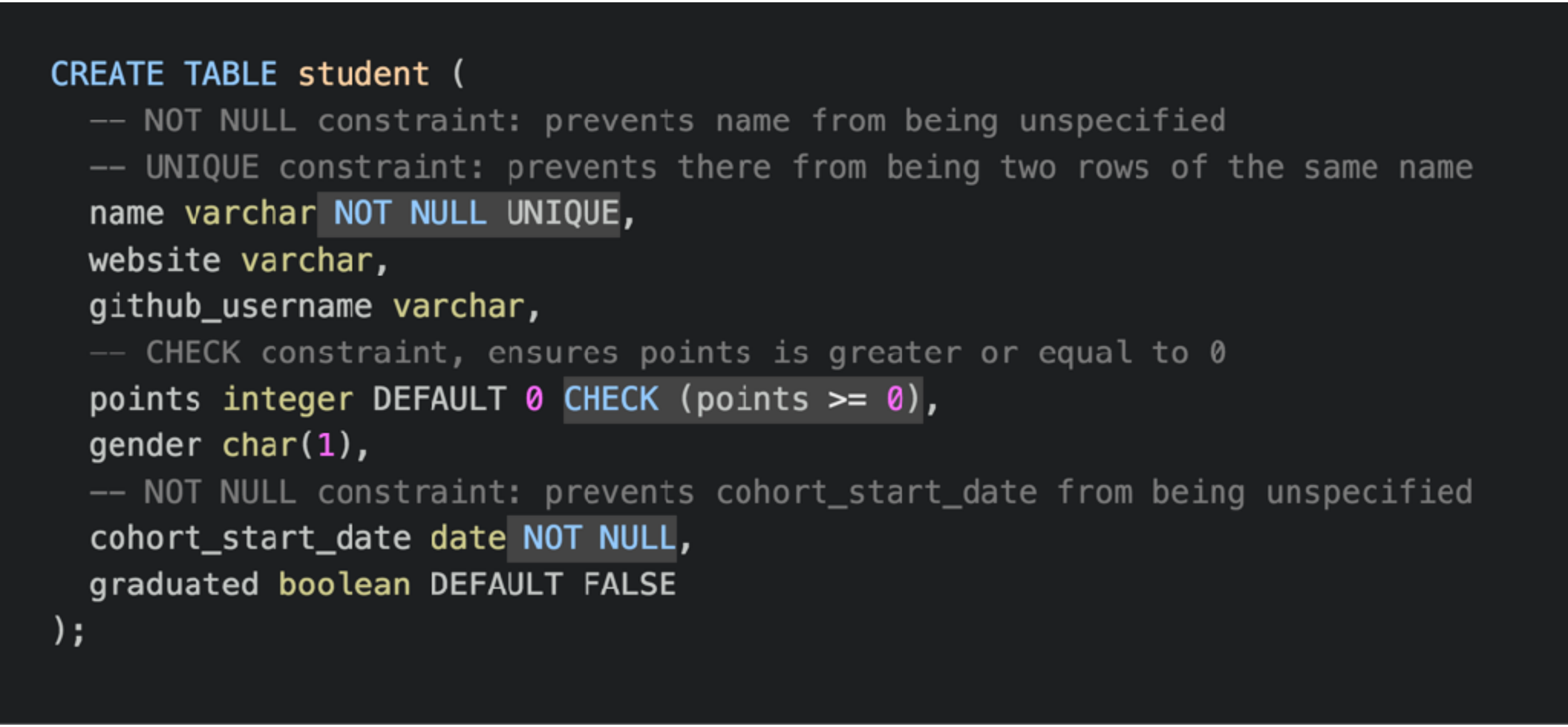
```
CREATE TABLE student (
  name varchar,
  website varchar,
  github_username varchar,
  -- defaults points to 0 if not specified
  points integer DEFAULT 0,
  gender char(1),
  cohort_start_date date,
  -- defaults graduate to FALSE if not specified
  graduated boolean DEFAULT FALSE
);
```

The default default value is `NULL`

Constraints

- NOT NULL - prevents a column value from being NULL
- UNIQUE - prevents any 2 rows in the table from having the same value in this column
- CHECK - allows number range checking and more powerful checking

Constraints



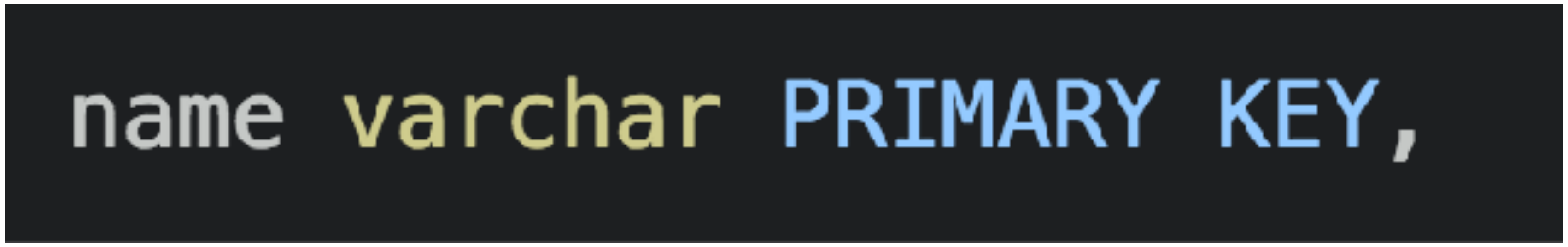
```
CREATE TABLE student (
  -- NOT NULL constraint: prevents name from being unspecified
  -- UNIQUE constraint: prevents there from being two rows of the same name
  name varchar NOT NULL UNIQUE,
  website varchar,
  github_username varchar,
  -- CHECK constraint, ensures points is greater or equal to 0
  points integer DEFAULT 0 CHECK (points >= 0),
  gender char(1),
  -- NOT NULL constraint: prevents cohort_start_date from being unspecified
  cohort_start_date date NOT NULL,
  graduated boolean DEFAULT FALSE
);
```

Primary Keys

Primary Keys

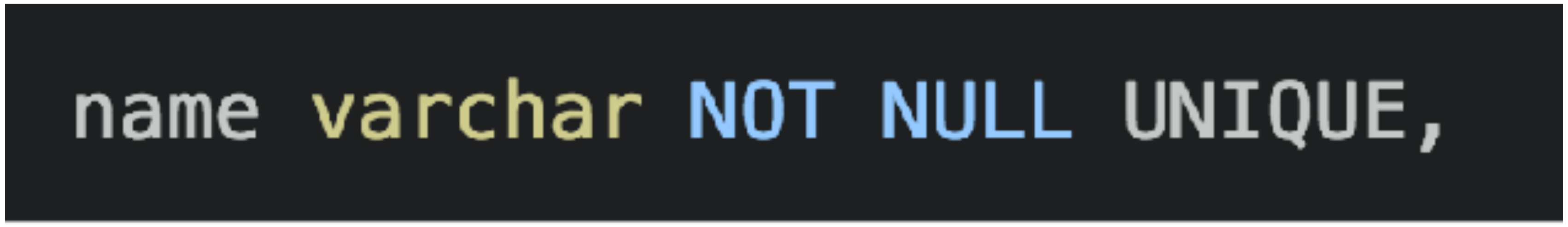
- A column or columns that uniquely identify a row
- Used for lookups (like keys in a dictionary)
- In reality is simply the combination of the constraints: NOT NULL and UNIQUE

Primary Keys



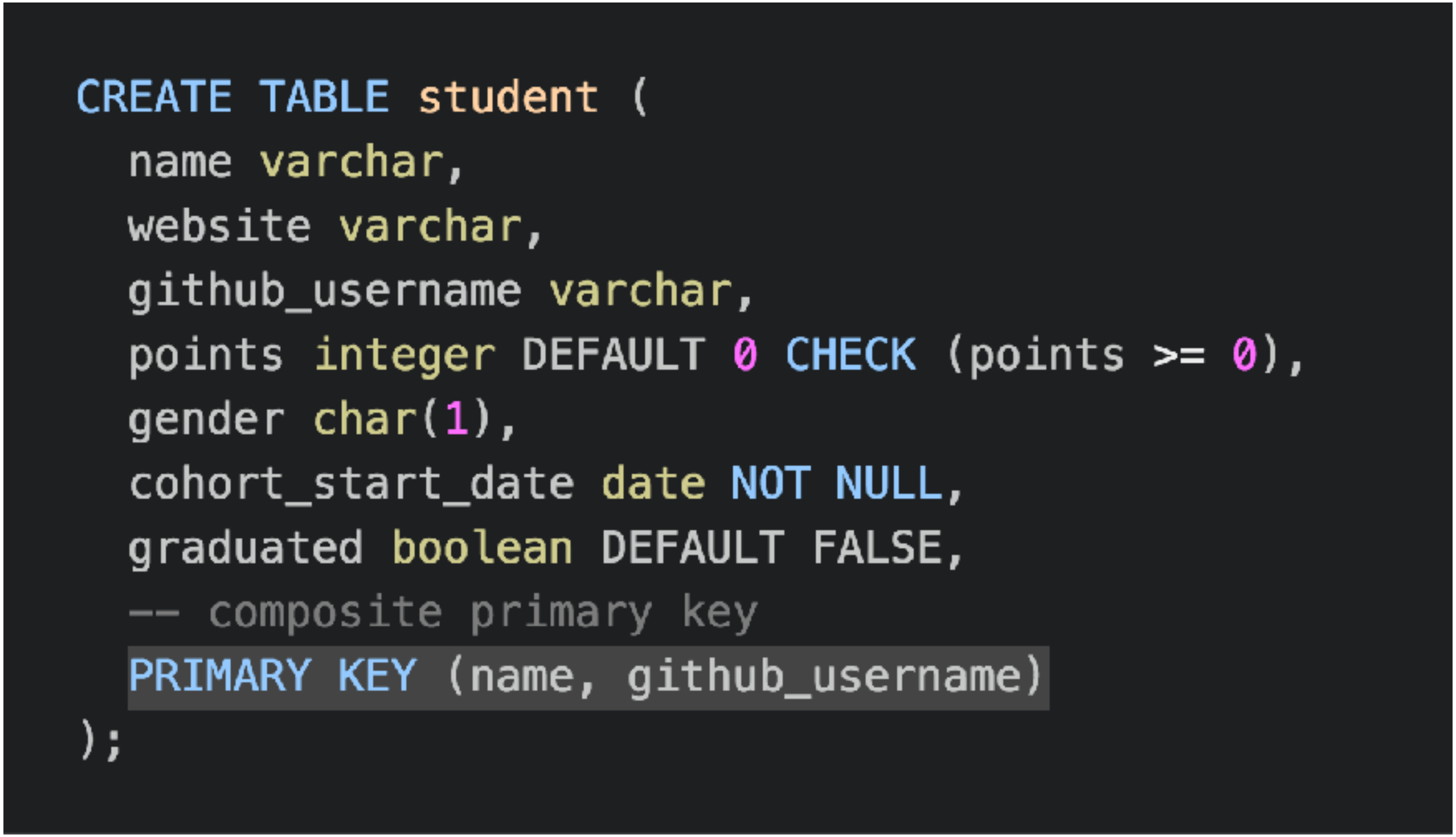
```
name varchar PRIMARY KEY,
```

Same as:



```
name varchar NOT NULL UNIQUE,
```

Composite Primary Keys



```sql
CREATE TABLE student (
  name varchar,
  website varchar,
  github_username varchar,
  points integer DEFAULT 0 CHECK (points >= 0),
  gender char(1),
  cohort_start_date date NOT NULL,
  graduated boolean DEFAULT FALSE,
  -- composite primary key
  PRIMARY KEY (name, github_username)
);
```

Serial IDs

auto-incrementing IDs

serial

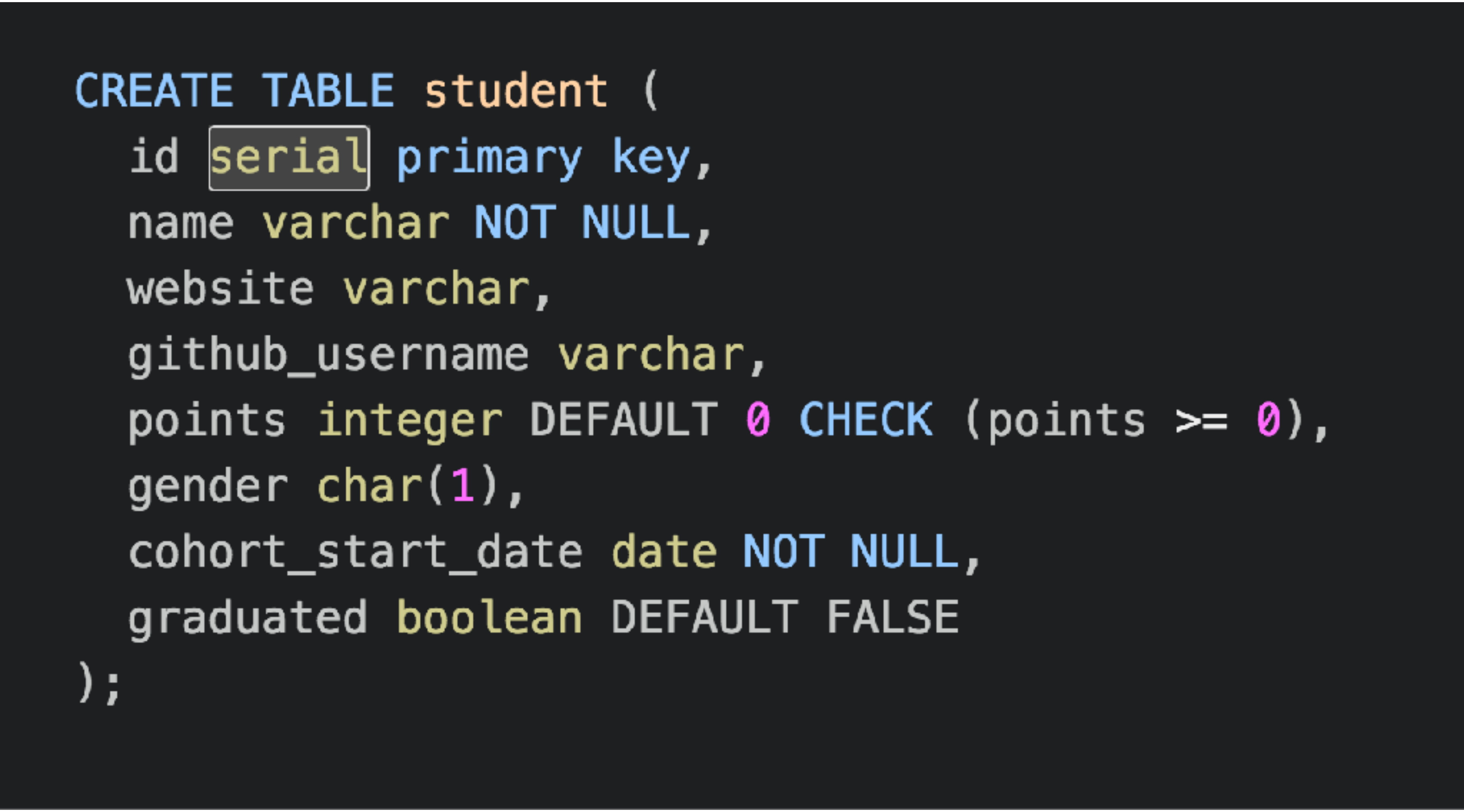
- `serial` is like a type, but not a real one, it is an alias to the integer type, but…

- that integer is auto-incremented by 1 for each new row when you insert it

- Using a combination of `serial` and `primary key` is common

- Do ***not*** specify the value of a serial column!

serial



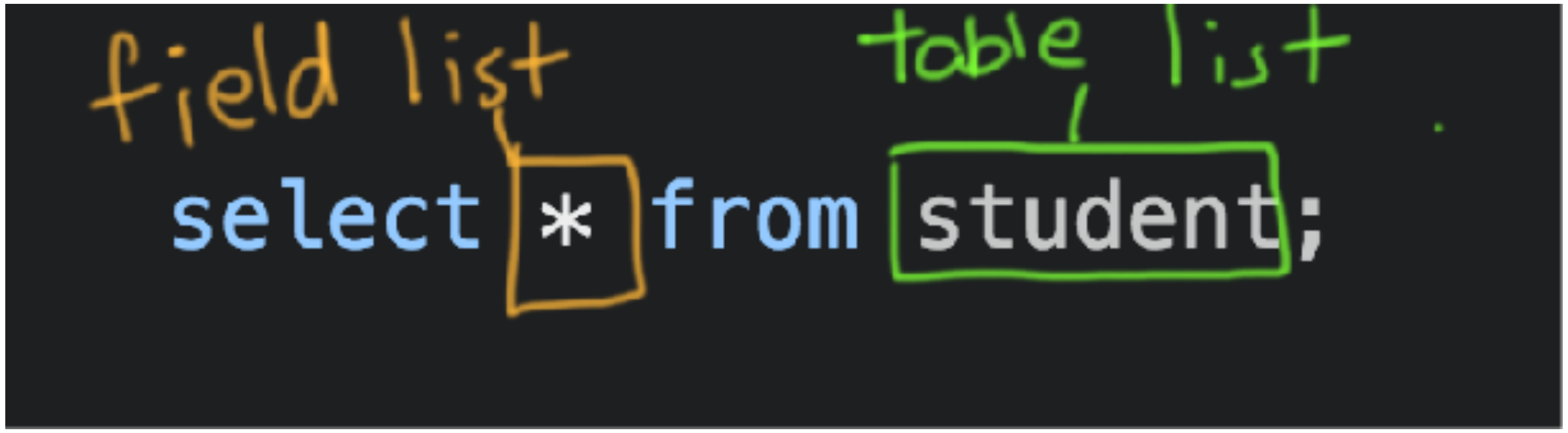
```
CREATE TABLE student (
  id serial primary key,
  name varchar NOT NULL,
  website varchar,
  github_username varchar,
  points integer DEFAULT 0 CHECK (points >= 0),
  gender char(1),
  cohort_start_date date NOT NULL,
  graduated boolean DEFAULT FALSE
);
```

Select Statement

Select Statement

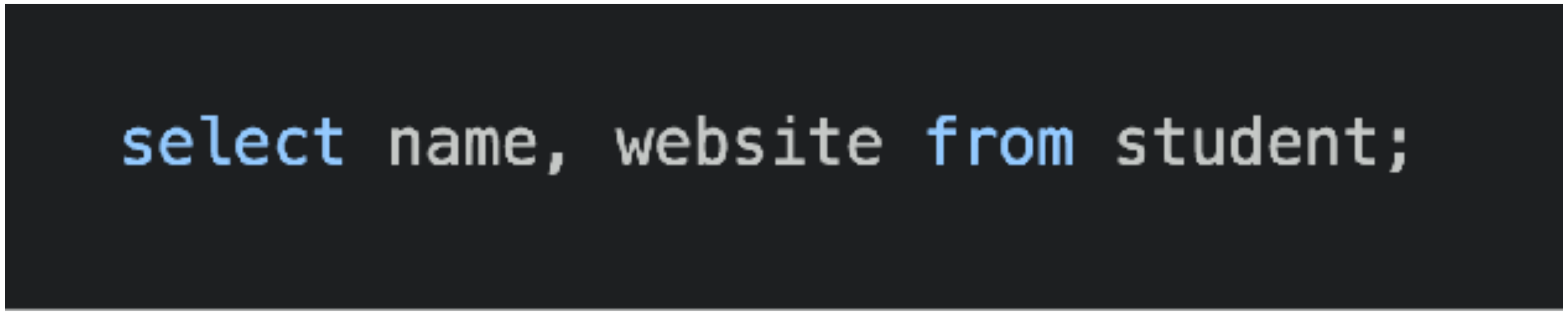
```sql
select * from student;
```

Select Statement



```
select * from student;
```

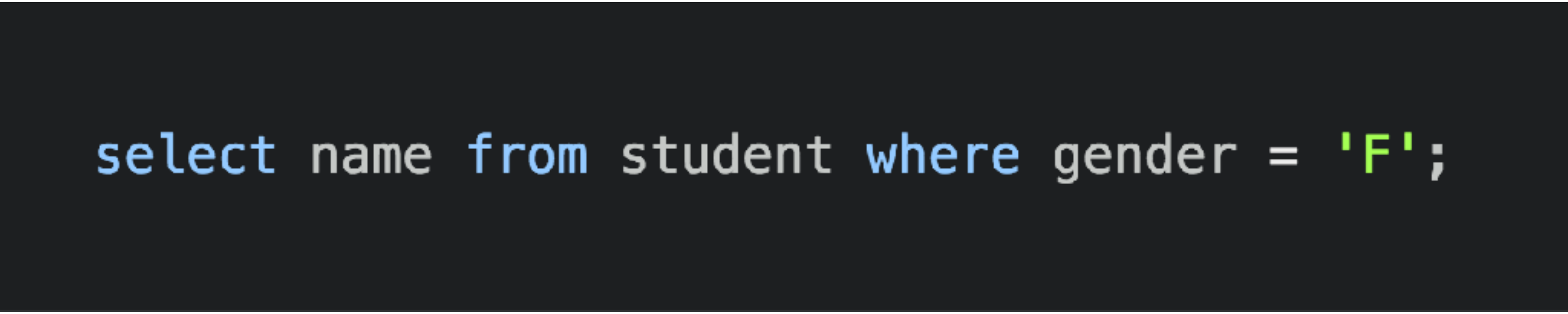
Select Statement



```
select name, website from student;
```

Where Clause

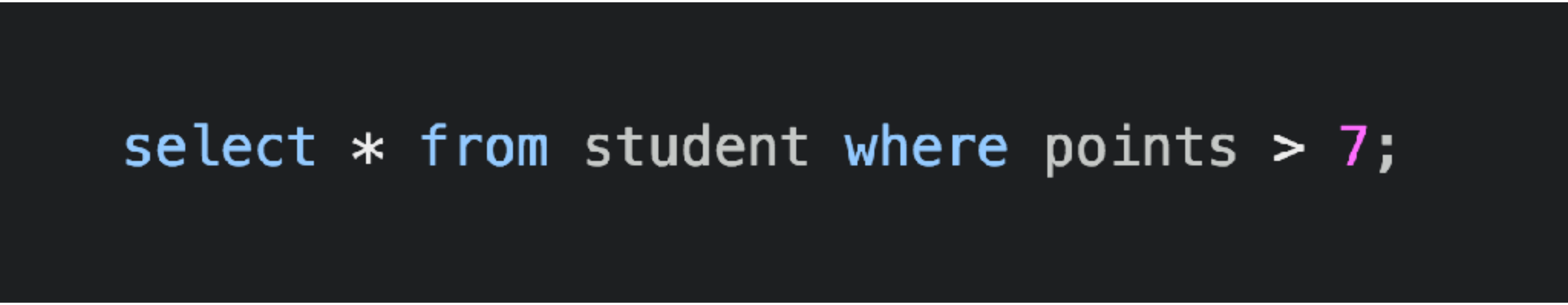
Where Clause



```
select name from student where gender = 'F';
```

single equal for comparisons (***not*** ==)

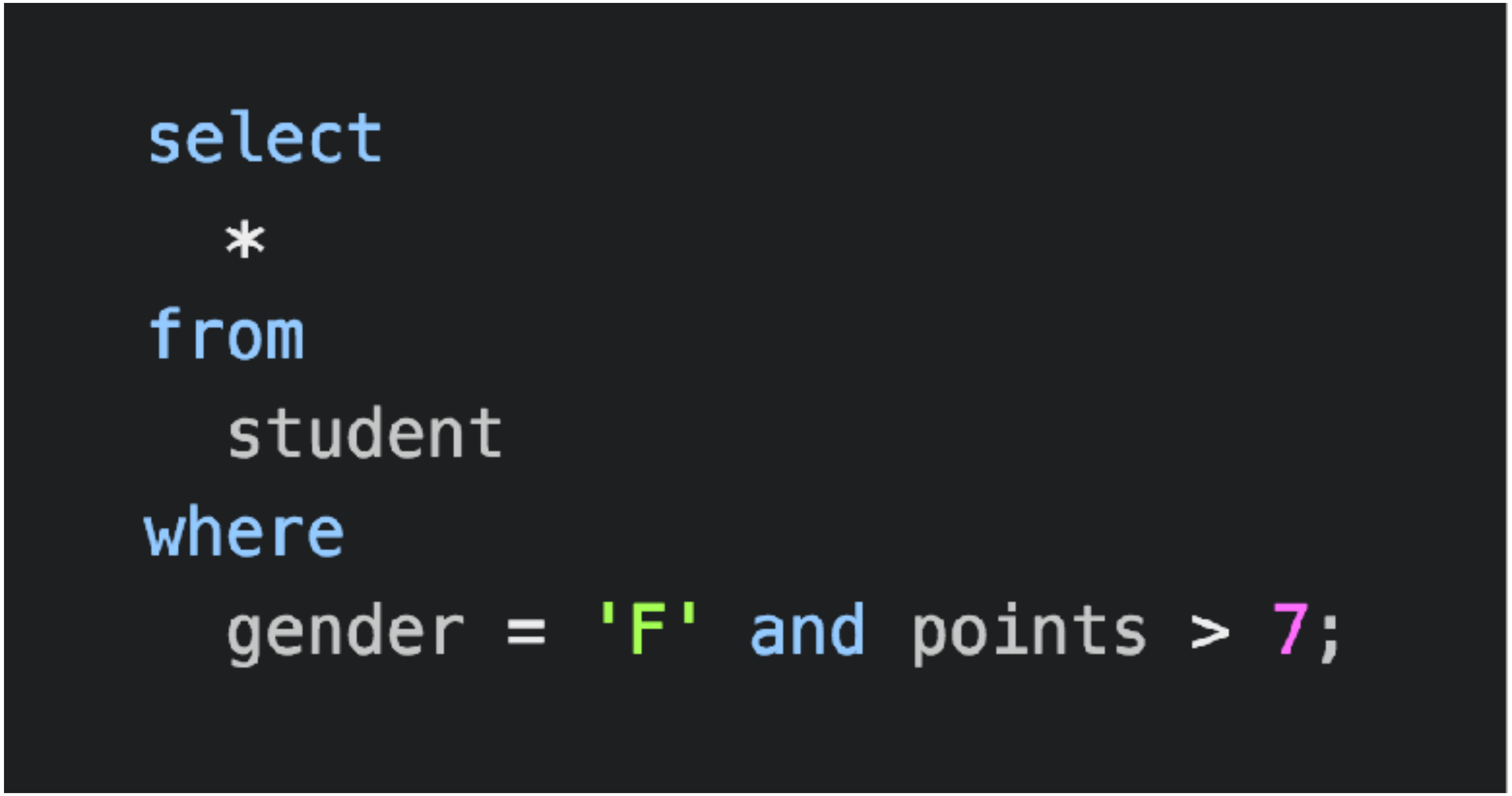
Where Clause



```
select * from student where points > 7;
```

greater than and less than operators

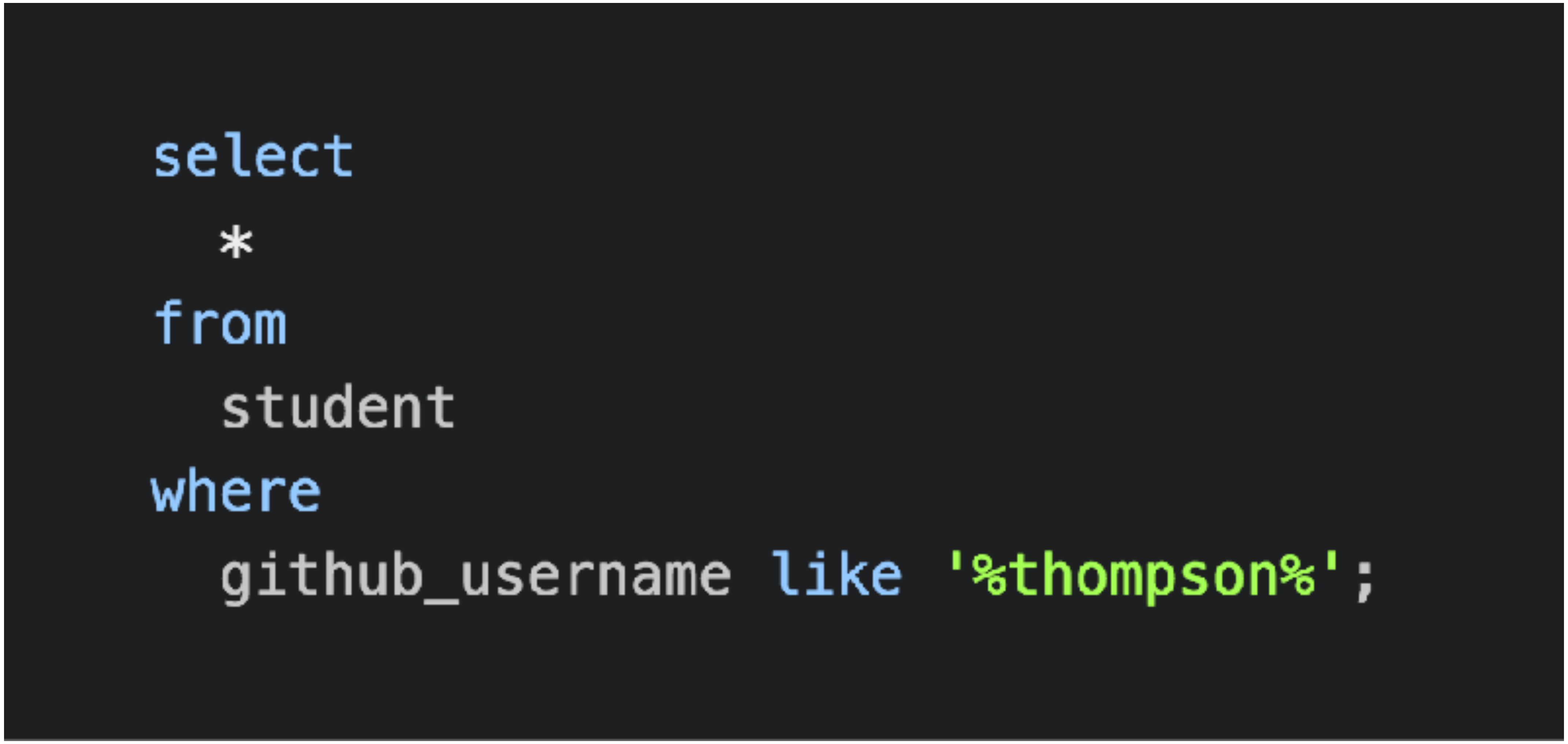
Where Clause



```
select
  *
from
  student
where
  gender = 'F' and points > 7;
```

and operator

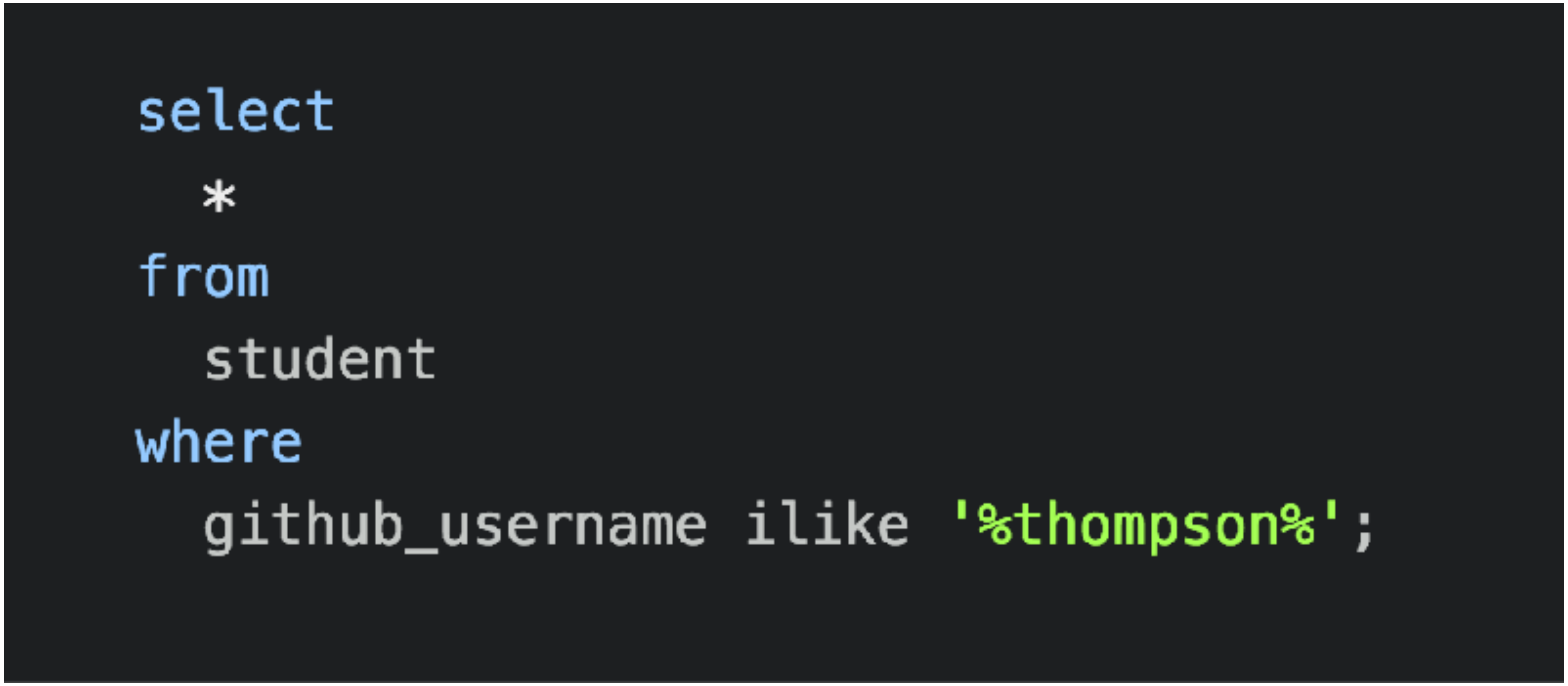
Where Clause



```
select
  *
from
  student
where
  github_username like '%thompson%';
```

like operator for substring comparison

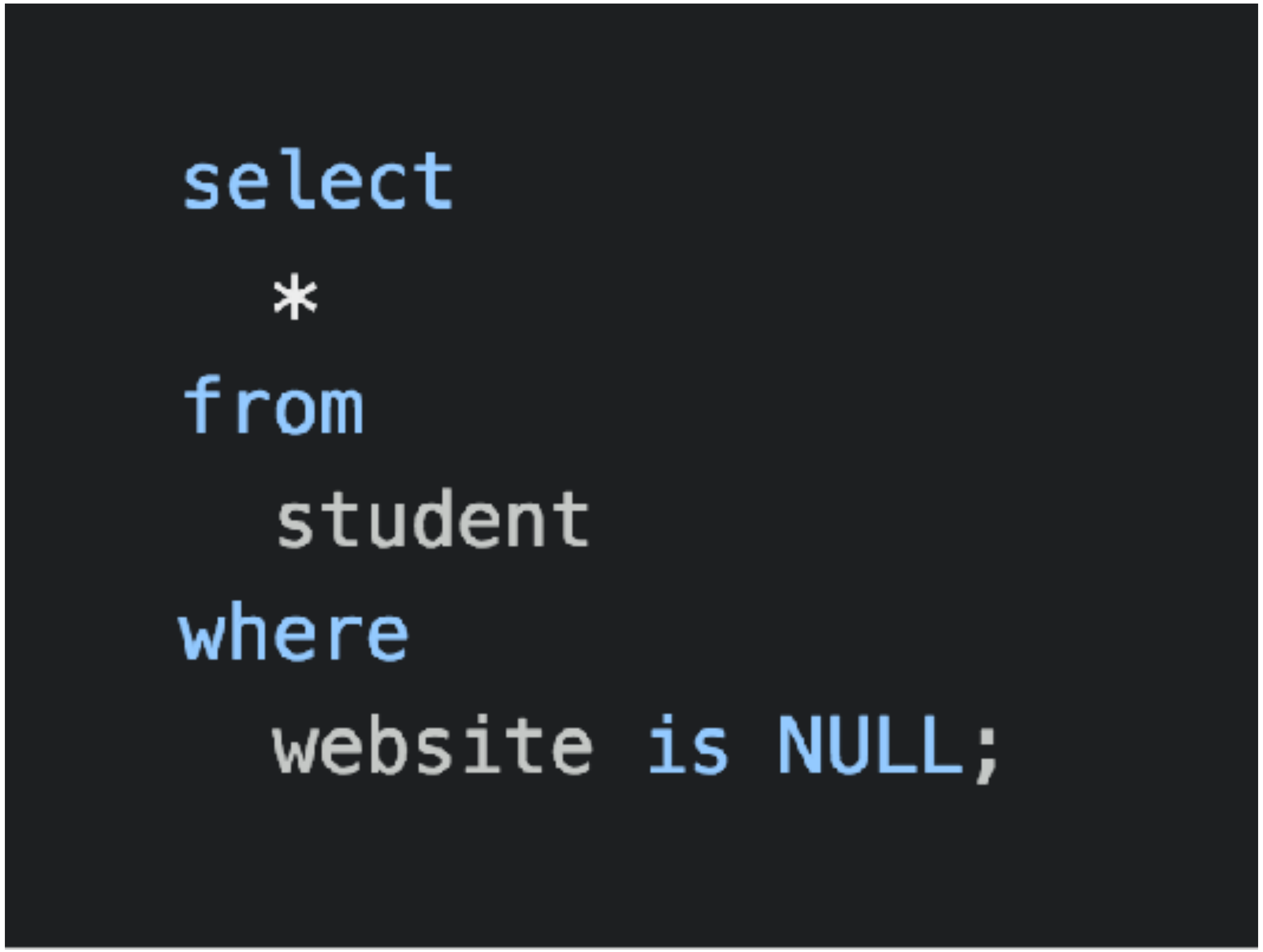
Where Clause



```
select
  *
from
  student
where
  github_username ilike '%thompson%';
```

ilike operator for case insensitivity

Where Clause



```
select
  *
from
  student
where
  website is NULL;
```

is NULL for null checks