ere's just a sampling of the many data types that SQL supports. We won't be using most of these types in this course, though.

The exact list of types differs from one database to another. For a full list of types, check the manual for your database, such as [**this one for PostgreSQL**](http://www.postgresql.org/docs/9.4/static/datatype.html).

## Text and string types

**text** — a string of any length, like Python **str** or **unicode** types.  
**char(n)** — a string of exactly n characters.  
**varchar(n)** — a string of up to n characters.

## Numeric types

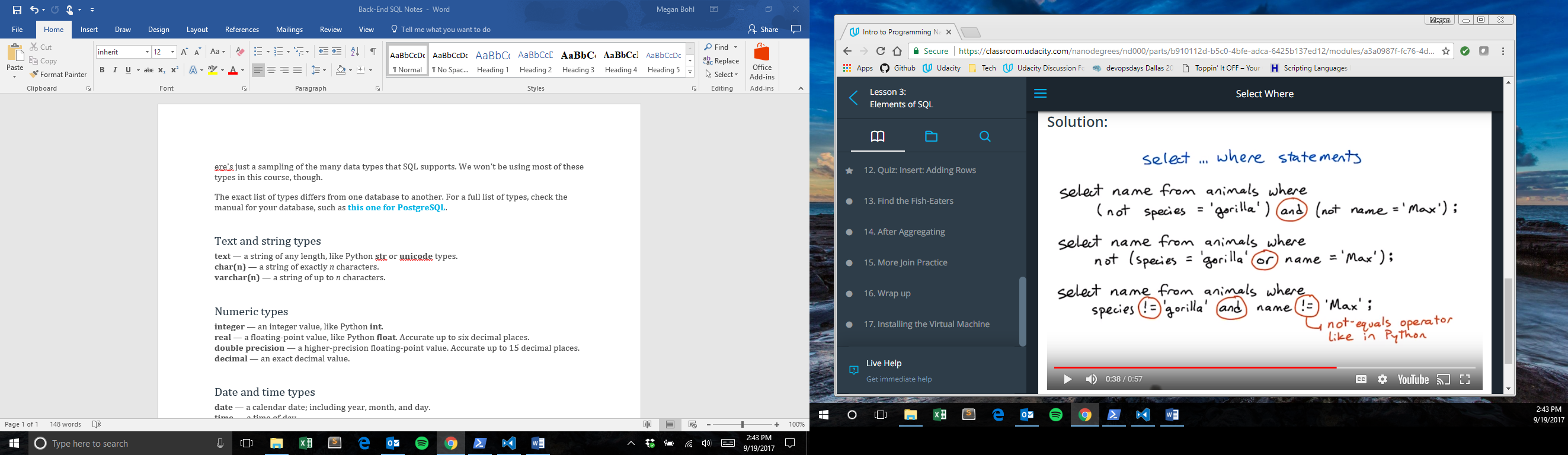
**integer** — an integer value, like Python **int**.  
**real** — a floating-point value, like Python **float**. Accurate up to six decimal places.  
**double precision** — a higher-precision floating-point value. Accurate up to 15 decimal places.  
**decimal** — an exact decimal value.

## Date and time types

**date** — a calendar date; including year, month, and day.  
**time** — a time of day.  
**timestamp** — a date and time together.

NEXT

NOT EQUAL TO



# eference

For reference, here's a list of all the tables in the zoo database:

### animals

This table lists individual animals in the zoo. Each animal has only one row. There may be multiple animals with the same name, or even multiple animals with the same name and species.

* name — the animal's name (example: 'George')
* species — the animal's species (example: 'gorilla')
* birthdate — the animal's date of birth (example: '1998-05-18')

### diet

This table matches up species with the foods they eat. Every species in the zoo eats at least one sort of food, and many eat more than one. If a species eats more than one food, there will be more than one row for that species.

* species — the name of a species (example: 'hyena')
* food — the name of a food that species eats (example: 'meat')

### taxonomy

This table gives the (partial) biological taxonomic names for each species in the zoo. It can be used to find which species are more closely related to each other evolutionarily.

* name — the common name of the species (e.g. 'jackal')
* species — the taxonomic species name (e.g. 'aureus')
* genus — the taxonomic genus name (e.g. 'Canis')
* family — the taxonomic family name (e.g. 'Canidae')
* t\_order — the taxonomic order name (e.g. 'Carnivora')

If you've never heard of this classification, don't worry about it; the details won't be necessary for this course. But if you're curious, Wikipedia articles [**Taxonomy**](http://en.wikipedia.org/wiki/Biological_classification) and [**Biological classification**](http://en.wikipedia.org/wiki/Biological_classification) may help.

### ordernames

This table gives the common names for each of the taxonomic orders in the **taxonomy** table.

* t\_order — the taxonomic order name (e.g. 'Cetacea')
* name — the common name (e.g. 'whales and dolphins')

## The SQL for it

And here are the SQL commands that were used to create those tables. We won't cover the **create table**command until lesson 4, but it may be interesting to look at:

**create** **table** animals (

**name** text,

species text,

birthdate date);

**create** **table** diet (

species text,

food text);

**create** **table** taxonomy (

**name** text,

species text,

genus text,

family text,

t\_order text);

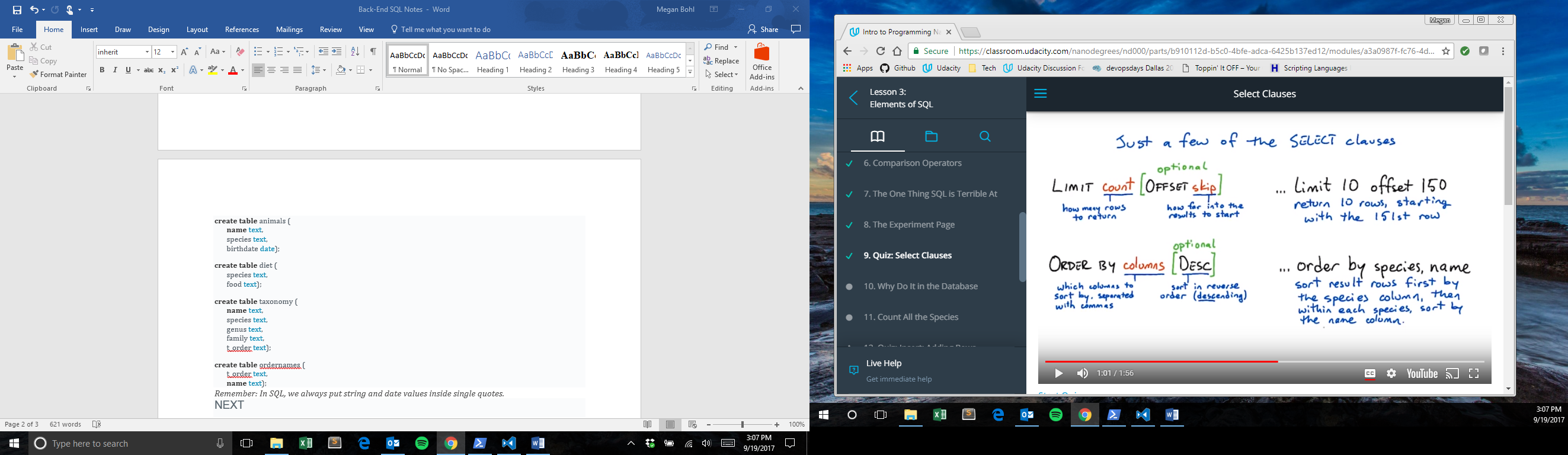
**create** **table** ordernames (

t\_order text,

**name** text);

Remember: In SQL, we always put string and date values inside single quotes.

NEXT



**Quiz: Select Clauses**

Here are the new **select** clauses introduced in the previous video:

... **limit** *count*  
Return just the first *count* rows of the result table.

... **limit** *count* **offset** *skip*  
Return *count* rows starting after the first *skip* rows.

... **order by** *columns*  
... **order by** *columns* **desc**  
Sort the rows using the *columns* (one or more, separated by commas) as the sort key. Numerical columns will be sorted in numerical order; string columns in alphabetical order. With **desc**, the order is reversed (**desc**-ending order).

... **group by** *columns*  
Change the behavior of aggregations such as **max**, **count**, and **sum**. With **group by**, the aggregation will return one row for each distinct value in *columns*.

These are all the select clauses we've seen in the lesson so far.

**where**

The **where** clause expresses *restrictions* — filtering a table for rows that follow a particular rule. **where**supports equalities, inequalities, and boolean operators (among other things):

* **where species = 'gorilla'** — return only rows that have 'gorilla' as the value of the species column.
* **where name >= 'George'** — return only rows where the name column is alphabetically after 'George'.
* **where species != 'gorilla' and name != 'George'** — return only rows where species isn't 'gorilla' and name isn't 'George'.

**limit / offset**

The **limit** clause sets a limit on how many rows to return in the result table. The optional **offset** clause says how far to skip ahead into the results. So **limit 10 offset 100** will return 10 results starting with the 101st.

**order by**

The **order by** clause tells the database how to sort the results — usually according to one or more columns. So **order by species, name** says to sort results first by the species column, then by name within each species.

Ordering happens before limit/offset, so you can use them together to extract pages of alphabetized results. (Think of the pages of a dictionary.)

The optional **desc** modifier tells the database to order results in descending order — for instance from large numbers to small ones, or from Z to A.

**group by**

The **group by** clause is only used with aggregations, such as **max** or **sum**. Without a **group by** clause, a select statement with an aggregation will aggregate over the whole selected table(s), returning only one row. With a **group by** clause, it will return one row for each distinct value of the column or expression in the **group by** clause.





1

2

3

4

5

6

7

8

9

10

-- Write a query that returns all the species in the zoo, and how many

-- animals of each species there are, sorted with the most populous

-- species at the top.

--

-- The result should have two columns: species and number.

--

-- The animals table has columns (name, species, birthdate) for each animal.

select ...

**Quiz: Insert: Adding Rows**

The basic syntax for the **insert** statement:

**insert into** *table* **(** *column1, column2, ...* **) values (** *val1, val2, ...* **);**

If the values are in the same order as the table's columns (starting with the first column), you don't have to specify the columns in the **insert** statement:

**insert into** *table* **values (** *val1, val2, ...* **);**

For instance, if a table has three columns **(a, b, c)** and you want to insert into **a** and **b**, you can leave off the column names from the **insert** statement. But if you want to insert into **b** and **c**, or **a** and **c**, you have to specify the columns.

A single **insert** statement can only insert into a single table. (Contrast this with the **select** statement, which can pull data from several tables using a join.)

