Querying in SQL/PostgreSQL, MongoDB, Numpy, Pandas and Django

First, a little terminology…

Shortly speaking, There are differences in technical language of these frameworks as they call concepts differently:

Concept	SQL-PostgreSQL-MySQL	MongoDB	Numpy	Pandas	Django
Based on	Relational and data-based	Document based	array-based		Like sql
Language and technology	SQL	Javascript in BSON, binary representation of JSON	Python in array		Python in Sqlite3
Database	Database	Database	There's no 'db' in Pandas and Numpy as they're using 'arrays' (one dimensional or multi-dimensional) in python.		Model
Table	Table	Collection	Array	DataFrame	Model
Column	Column	Field	Column	Column	field
Row	Row	Document	Row	Row	Row
Data types	String, number, date, time, bolean	String, number, date/time, bolean	String, number, date/time, bolean	String, number, date/time, bolean	String, number, date/time, bolean

Querying:

	SQL-PostgreSQL-MySQL	MongoDB	Numpy	Pandas	Django

creating data	for creating db:
	createdb mydb
	using db: psql mydb
	for creating table:
	CREATE TABLE weather (
	city varchar(80) references cities(name),
	temp_lo int,
	temp_hi int,
	prcp real,
	date date
	);
		for creating DB, set configurations in cloud server
		for creating Table:
		db.createCollection("posts", {
		validator: {
		$jsonSchema: {
		bsonType: "object",
		required: ["title", "body"],
		properties: { title: {
		bsonType: "string",
		description: " required"
		}, body: {
		bsonType: "string",
		description: " Required."
		}, likes: {
		bsonType: "int",
		description: "Optional."
		} } } })
			vec1 = np.array([[11,12], [21,22], [31,32], [41,42]])
			# arange(start inclusive, stop exclusive, step size)
			vec2 = np.arange(0, 5, 1, dtype=float)
			vec3 = np.array([vec2, vec2])
			# linspace(start inclusive, stop inclusive, number of elements)
			vec3 = np.linspace(0, 4, 5)
			#float by default
			np.zeros((3,4))
			np.ones((2,3,4), dtype=np.int16)
			np.empty((2,3))
				dates = pnd.date_range('20130101', periods=6)
				dict = {'cat':'1', 'dog':'2'}
				arr = [1,2,3,4,5,6,7,8]
				arr2 = [[11,12], [21,22], [31,32], [41,42], [51,52], [61,62]]
				df = pnd.DataFrame(arr2)
					in models.py file:
					from django.db import models
					class Member(models.Model):
					firstname = models.CharField(max_length=255)
					lastname = models.CharField(max_length=255)

	SQL-PostgreSQL-MySQL	MongoDB	Numpy	Pandas	Django

Inserting data	INSERT INTO weather (city, temp_lo, temp_hi, prcp, date)
	VALUES ('San Francisco', 43, 57, 0.0, '1994-11-29');
	COPY weather FROM '/home/user/weather.txt' (DELIMITER ',');
	COPY (SELECT * FROM country WHERE country_name LIKE 'A%') TO '/usr1/proj/bray/sql/a_list_countries.copy';
		db.posts.insertOne({ title: "Post Title 1", body: "Body of post." })
		db.posts.insertMany({ title: "Post Title 1", body: "Body of post." }, { title: "Post Title 2", body: "Body of post." })
			# inserting at first col and third row:
			vec4[1][3] = 12
			# setting every 2nd element of the range 0:6 to -1000:
			a[:6:2] = -1000
				#setting vals by label :
				df.at[dates[0], 'A'] = 0
				#setting values by position:
				df.iat[0, 1] = 0
				s1 = pnd.Series([1, 2, 3, 4, 5, 6], index=pnd.date_range('20130102', periods=6))
				df['F'] = s1
					member1 = Member(firstname='Tobias', lastname='Refsnes')
					member2 = Member(firstname='Linus', lastname='Refsnes')
					member3 = Member(firstname='Lene', lastname='Refsnes')
					members_list = [member1, member2, member3]
					for x in members_list:
					x.save()

Dimension / reshaping	-- 1. showing the number of columns
	SELECT COUNT(*)
	FROM INFORMATION_SCHEMA.COLUMNS
	WHERE table_catalog = 'database_name' -- the database
	AND table_name = 'table_name'
	-- 2. Showing the number of rows
	SELECT COUNT(column_name) FROM table_name;
		db.collection.aggregate([
		{"$project":{"numFields":{"$size":{"$objectToArray":"$$ROOT"}}}},
		{"$group":{"_id":null, "fields":{"$sum":"$numFields"}, "docs":{"$sum":1}}},
		{"$project":{"total":{"$subtract":["$fields", "$docs"]}, _id:0}}
		])
		First stage $project is to turn all keys into array to count fields. Second stage $group is to sum the number of keys/fields in the collection, also the number of documents processed. Third stage $project is subtracting the total number of fields with the total number of documents (As you don't want to count for _id).
			# shows the dimension of data frame --> output: int
			b.ndim
			# shows the number of elements per dimension --> e.g.: (6,3)
			b.shape
			# number of elments
			b.size
			a.resize((2,6))
			# n and m is new dimensions
			a.reshap(n,m)
				# total number of elements:
				df.size
				# per column:
				df.count()
				# there's no reshaping or resizing in data frame
					# import model
					from django.contrib.auth.models import User
					#then get the _meta data
					User._meta.get_fields()
					#also:
					def get_model_fields(model):
					return model._meta.fields

Sorting / reversing	Using ORDER BY in SELECT query	Basic syntax:
		db.COLLECTION_NAME.find().sort({KEY:1})
		example:
		db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
			b.sort()
			#reversing all items
			b[: :-1]
			# there's no partial reversing, you can do partial selection and then reverse, but you can not do this: a[1:6:1]
			# first you should run this:
			x = a[1:6:]
			# then x = x[::-1]
				df.sort_values(by='colname', ascending=false)
				df.sort_index(axis=1, ascending=False)
				#--> axis 1, horizontal and 0, vertical sorting
					This sorts lastname column ascending, but id as descending:
					mydata = Member.objects.all().order_by('lastname', '-id').values()

	SQL-PostgreSQL-MySQL	MongoDB	Numpy	Pandas	Django

Selecting data	SELECT city, (temp_hi+temp_lo)/2 AS temp_avg, date
	FROM weather
	GROUP BY city
	WHERE city = 'San Francisco' AND prcp > 0.0
	ORDER BY city, temp_lo;
	
	→ refer to query operators below
		db.posts.find({category= "News", likes: { $gt: 1 }}, {_id: 0, title: 1, date: 1})
		// searching and indexing:
		{ $search: { index: "default",
		// optional unless you named your index something other than "default"
		text: { query: "star wars", path: "title" }, }, }
		//for filtering data in MongoDB, use 'aggregate' function like this:
		db.posts.aggregate([
		// This only finds docs that have more than 1 like
		{ $match: { likes: { $gt: 1 } } },
		// Group docs by category and sum each categories likes
		{ $group: { _id: "$category", totalLikes: { $sum: "$likes" } } }
		// limits the no of docs passed to the next stage.
		{ $limit: 1 }
		// passes only the specified fields along to the next stage. consider two brackets
		{ $project: {"name": 1, "cuisine": 1, "address": 1 } }
		//This aggregation sorts desc
		{ $sort: { "accommodates": -1 } }
		// It behaves like 'find'. It will filter docs that match the query.
		{ $match : { property_type : "House" } }
		// This adds new fields to docs.
		{ $addFields: { avgGrade: { $avg: "$grades.score" } } }
		// counts total no of all docs passed from the previous stage
		{ $count: "totalChinese" }
		// left outer join in the same db.
		{ $lookup: { from: "movies",
		localField: "movie_id",
		foreignField: "_id",
		as: "movie_details", }, }])
		→ refer to query operators below
			in np, cols and rows are located transposed, cols horizontally and rows vertically
			array([[0, 1, 2, 3],
			[4, 5, 6, 7],
			[8, 9, 10, 11],
			[12, 13, 14, 15]])
			b[2:4]
			# shows col no. 2 and 3, not 4
			b[0:4][0:2]
			# showing first two results ([0:2]) of b[0:4]
			b[0:4][0:2][0][0]
			# getting first element of last query
			b[:6:2]
			# every 2nd elements of range 0:6
				#col1 and col2 rows 4to8
				df[['col1', 'col2']][4:8]
				ddf[0][1:4]
				# elements 1 to 3 (first to third, not index value of 1 to 3), from first col
				df[2:]
				# shows from 2 to the end not from second row
				df[::2] # show rows in even pos
				df.groupby('colname').count()
				df['colname'].apply(lambda x:x <= 5000)
				# returns True/False for each row
				df.loc[[0,12], :]
				# select all cols for rows of index values 0 and 12 (not 0 to 12)
				df.iloc[0:8, 0:12]
				# slicing 0to8 from rows and 0to12 from cols
				df.head(n=2)
				# show first 2 rows, the same as the next , also like this:
				df.head(2)
				df.tail(4)
				# shows the last 4 rows
				# for conditional selection, use df[condition]:
				df[df[1] == 22]
				df.loc[df[1].isin(['val1', 'val2'])]
				# locating in df[1] to find rows with val1 and val2
				df[(df[1] == 'val') & (df2[2] == 'val')]
				#val in first col of two data frame: df and df2
					from members.models import Member
					x = Member.objects.all()[4]
					# this filters values by condition:
					mydata = Member.objects.filter(firstname ='Emil').values()

	SQL-PostgreSQL-MySQL	MongoDB	Numpy	Pandas	Django

Updating data	UPDATE weather
	SET temp_hi = temp_hi - 2, temp_lo = temp_lo - 2
	WHERE date > '1994-11-28';
		db.posts.updateOne({ title: "Post Title 1" }, { $set: { likes: 2 } })
		insert if not found by upsert:
		db.posts.updateOne(
		{ title: "Post Title 5" },
		{ $set:
		{
		title: "Post Title 5",
		body: "Body of post.",
		likes: 5,
		} },
		{ upsert: true }
		)
		// Update likes on all documents by 1. For this we will use the $inc (increment) operator:
		db.posts.updateMany({}, { $inc: { likes: 1 } })
		→ refer to update operators below
			b[1] = val	df.at[dates[0], 'A'] = 0
				#setting vals by label
				df.iat[0, 1] = 0
				#setting values by position
					from members.models import Member
					x = Member.objects.all()[4].field_name = 'val'
					x.save()

Joining data	SELECT * FROM weather JOIN cities ON city = name;
	--> refer to pdf from another repository
		1. Using $lookup aggragate function:
		Model_name.aggregate([
		{ $lookup: {
		from: <collection to join>,
		localField: <field from the input documents>,
		foreignField: <field from the documents of the "from" collection>,
		as: <output array field>
		} }])
		2. using View to join data:
		db.createView("sales", "orders", [
		{ $lookup: {
		from: "inventory",
		localField: "prodId",
		foreignField: "prodId",
		as: "inventoryDocs"
		} },
		{ $project: {
		_id: 0,
		prodId: 1,
		orderId: 1,
		numPurchased: 1,
		price: "$inventoryDocs.price"
		} },
		{ $unwind: "$price" }])
			b = a + c	pnd.concat([df1, df2])
				#attach to each other, one on top of the other, if cols are diff, NaN will be put
				pnd.concat([df1, df2], join="inner")
				#only common cols are joined together
				#or this
				pieces = [df[:3], df[3:7], df[7:]]
				pnd.concat(pieces)
				pnd.merge(df1, df2, on='colname')
				#if values in df1.col1 are the same as values in df2.col1, df2 merges to the botom of df1
				key lval rval
				0 foo 1 4
				1 foo 1 5
				2 foo 2 4
				3 foo 2 5
				#if values in df1.col1 are diffferent from vals in df2.col1, df2 merges to the right of df1
				key lval rval
				0 foo 1 4
				1 bar 2 5
					1. Creating new model with foreign key to some models:
					2. // Using Django ORM's select_related method
					a1 = model_.objects.select_related('reporter')
					3. using Django ORM's filter method:
					a2 = model_name.objects.filter(reporter__username='John')
					4. using union methon:
					q1.union(q2)

	SQL-PostgreSQL-MySQL	MongoDB	Numpy	Pandas	Django

Deleting data	dropdb mydb
	DROP TABLE tablename;
	DELETE FROM weather WHERE city = 'Hayward';
		db.posts.deleteOne({ title: "Post Title 5" })
		db.posts.deleteMany({ category: "Technology" })
			a = np.arange(12).reshape(3, 4)
			# a = [[0 1 2 3]
			# [4 5 6 7]
			# [8 9 10 11]]
			
			a_del = np.delete(a, 1, 0)
			# a_del = [[0 1 2 3]
			# [8 9 10 11]]
				removing repeating / missing data (nan)
				u = df1.unique()
				l = df1.len()
				if l > u:
				df1.dropna(how='any')
				df1.fillna(value=5)
				#replace nan with 5
				pnd.isna(df1)
				#returns true or false
					x = Member.objects.all()[5]
					x.delete()

Copying data	SELECT column FROM table INTO new_table;	Creating a new empty collection, selecting data from a collection and insert data to the new collection	df2 = df1.copy()
			c = a.view()
			# c is not a, but c.base is a, however their shape is different
			# if c changes, a does TOO
			d = a.copy()
			# d is not a, d.base is not a, and their shape is different
			# if d changes, a does NOT also
				Creating a new empty table, selecting data from a table and insert data to the new table	There is no built-in method for copying model instances, it is possible to create new instance with all fields values copied. The solution is filtering data and put the resulting queryset to a new model

Foreign keys	first when you create a table with REFERENCES keyword CREATE TABLE cities (name text, population real, elevation int -- (in ft)); CREATE TABLE capitals (state char(2) UNIQUE NOT NULL) INHERITS (cities);	However MongoDB is not relational. There is no standard "normal form" and you should model your database appropriate to the data you store and the queries you intend to run, but refer to this link for more info: https://www.mongodb.com/docs/manual/reference/database-references/	there's no partial inheritance like what's happening in SQL (i.e. referencing one column from a table to one column from another column). In pandas and numpy, you can just reference the whole dataframe (table), not a column of it. i.e. even if you reference a column from a dataframe to a column from another dataframe, it's not working but temporary copying...		Use this syntax: class this_model(models.Model): … location_id = models.ForeignKey(another_model)

Notes:

- You can always run raw SQL queries in Django by using raw() method.

Update Operators used in MongoDB:

1. Fields

The following operators can be used to update fields:
$currentDate: Sets the field value to the current date
$inc: Increments the field value
$rename: Renames the field
$set: Sets the value of a field
$unset: Removes the field from the document

2. Array

The following operators assist with updating arrays.
$addToSet: Adds distinct elements to an array
$pop: Removes the first or last element of an array
$pull: Removes all elements from an array that match the query

$push: Adds an element to an array

Query Operators can be used in aggregate function in MongoDB:

-A- Comparison
$eq, $ne, $gt, $gte, $lt, $lte, $in
-B- Logical
$and, $or, $nor, $not
-C- Evaluation
$regex, $text, $where

Query Operators can used in PostgreSQL query statements:

-A- Comparison
<, >, <>, =, !=, >=, <=
-B- Logical
AND, OR, NOT
-C- Evaluation