SQL Technical Interview Questions and Answers

Joins Are From Descartes, Rows Are From Schemas

Part 1: Joins

- The outputs of following queries are 25 and 10, respectively.

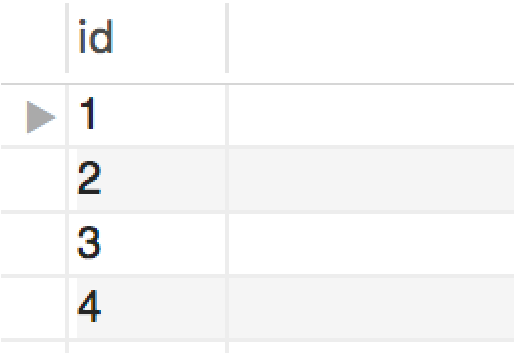
```
SELECT COUNT(*)
FROM first_table;

SELECT COUNT(*)
FROM second_table;
```

- What will be the number of rows in the output of the following query?

```
SELECT *
FROM first_table, second_table;
```

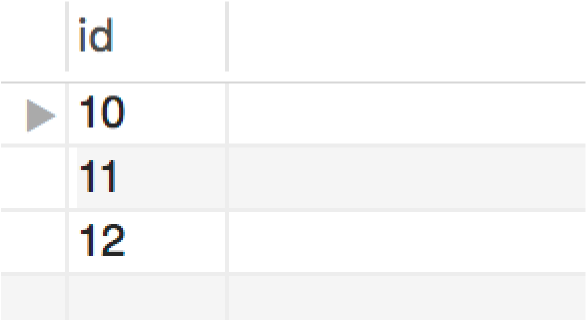
Part 2: Joins

- The query `SELECT * FROM table_one;` returns the following:



id
1
2
3
4

- And the query `SELECT * FROM table_two;` returns the following:

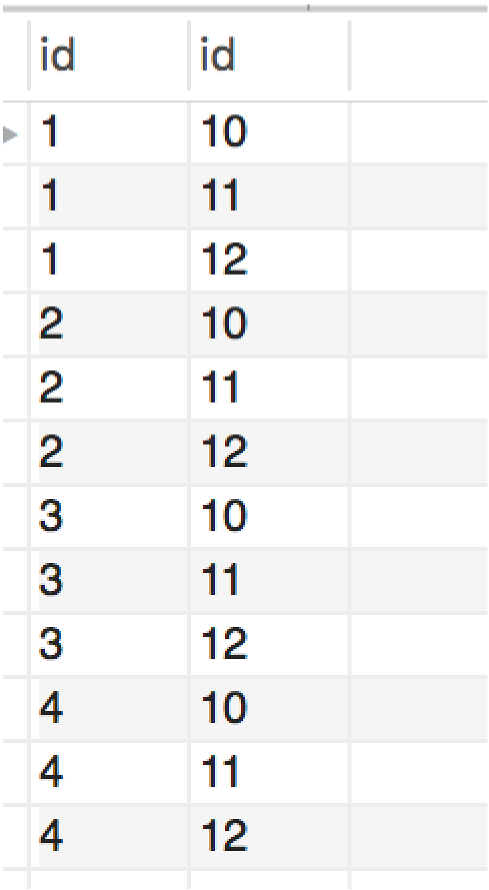


id
10
11
12

- What will the query `SELECT * FROM table_one, table_two;` look like?

Solutions: Joins

- Part 1: The query will return a Cartesian Join, and it will return 250 rows.
- Part 2:

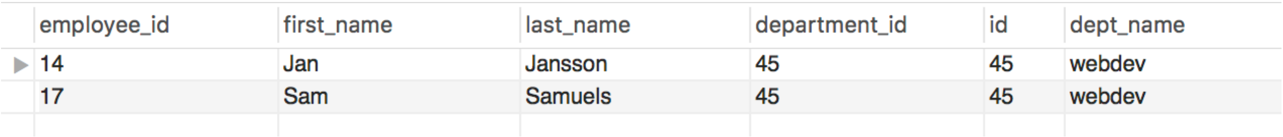


id	id
1	10
1	11
1	12
2	10
2	11
2	12
3	10
3	11
3	12
4	10
4	11
4	12

Foreign Keys

- The following are a query and output involving two tables. Notice that `department_id` and `id` columns match.

```
SELECT * FROM employees e
JOIN departments d
ON (e.department_id = d.id)
WHERE e.department_id = 45;
```



employee_id	first_name	last_name	department_id	id	dept_name
14	Jan	Jansson	45	45	webdev
17	Sam	Samuels	45	45	webdev

- Based on the above, reconstruct the table schemata for `employees` and `departments` tables.

Solution: Foreign Keys

```
CREATE TABLE departments (
  id integer(11) UNIQUE NOT NULL,
  dept_name VARCHAR (255) NOT NULL,
  primary key (id)
);

CREATE TABLE employees (
  employee_id INTEGER(11) UNIQUE NOT NULL,
  first_name VARCHAR (255) NOT NULL,
  last_name VARCHAR (255) NOT NULL,
  department_id INTEGER (11) NOT NULL,
  PRIMARY KEY (employee_id),
  FOREIGN KEY (department_id) REFERENCES departments(id)
);


INSERT INTO departments (id, dept_name) VALUES (25, "data");
INSERT INTO departments (id, dept_name) VALUES (45, "webdev");

INSERT INTO employees (employee_id, first_name, last_name, department_id)
VALUES (3, "Chris", "Christian", 25);
INSERT INTO employees (employee_id, first_name, last_name, department_id)
VALUES (14, "Jan", "Jansson", 45);
INSERT INTO employees (employee_id, first_name, last_name, department_id)
VALUES (17, "Sam", "Samuels", 45);


SELECT * FROM employees e
JOIN departments d
ON (e.department_id = d.id)
WHERE e.department_id = 45;
```

ACID

- What are the ACID properties of SQL transactions? If possible, explain each property with an illustration of an example.

Solution: ACID

- ACID stands for Atomicity, Consistency, Isolation, and Durability.
- Atomicity: each transaction must be all-or-nothing. That is, a transaction takes place if it's only partly completed. In a bank transfer from person A to person B, for example, person B's account cannot be credited unless withdrawal takes place from person A's account.
- Consistency: all constraints are followed for each transaction. Constraints such as keys, data types, uniqueness are followed. The database should remain in a consistent state before and after the transaction.

- Isolation: no transaction affects another transaction. If there are multiple bank transfers, for example, only one can be carried out at the same time, before another one can begin.
- Durability: the transaction will persist in the database after a transaction. For example, after a bank transfer, even if a power outage should take place, the record of the transaction remains intact.
- For another analogy, see https://stackoverflow.com/questions/3740280/acid-and-database-transactions#3741079.

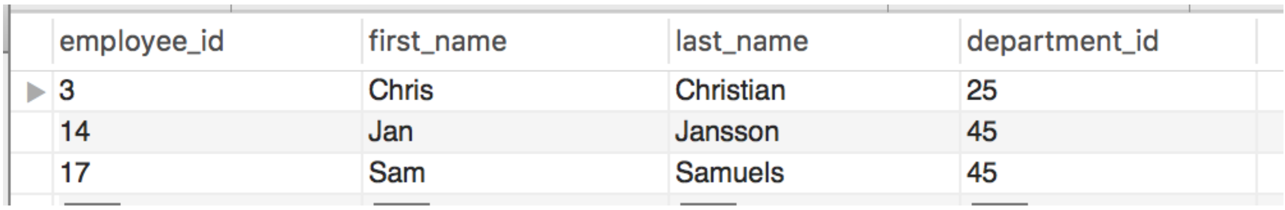
Alter vs. Update

Part 1: Alter vs. Update

- Explain the difference between `alter` and `update` in SQL statements.

Part 2: Alter vs. Update

- You are given the following table:



employee_id	first_name	last_name	department_id
3	Chris	Christian	25
14	Jan	Jansson	45
17	Sam	Samuels	45

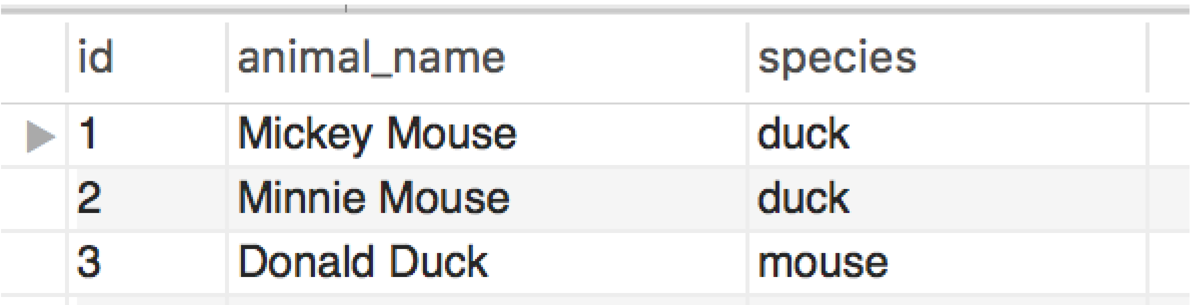
- Change the name of the column from `department_id` to `dept_id`.
- Add a column named `annual_salary` to the table.

Solutions: Alter vs. Update

```
-- MySQL
ALTER TABLE employees
CHANGE department_id dept_id VARCHAR(125);

ALTER TABLE employees
ADD annual_salary INT(11);

UPDATE employees
SET annual_salary = 80000
WHERE employee_id = 17;
```

The Thrill of the Case

- Change each animal's species to the correct species.



	id	animal_name	species
▶	1	Mickey Mouse	duck
	2	Minnie Mouse	duck
	3	Donald Duck	mouse

Solution: The Thrill of the Case

```
-- MySQL code
CREATE TABLE animals (
  id integer(11) auto_increment not null,
  animal_name varchar(255) not null,
  species varchar(255),
  primary key(id)
);

INSERT INTO animals (animal_name, species) VALUES ("Mickey Mouse",
"duck");
INSERT INTO animals (animal_name, species) VALUES ("Minnie Mouse",
"duck");
INSERT INTO animals (animal_name, species) VALUES ("Donald Duck",
"mouse");

UPDATE animals
SET species =
CASE species
    WHEN "duck" THEN "mouse"
    WHEN "mouse" THEN "duck"
END;
```

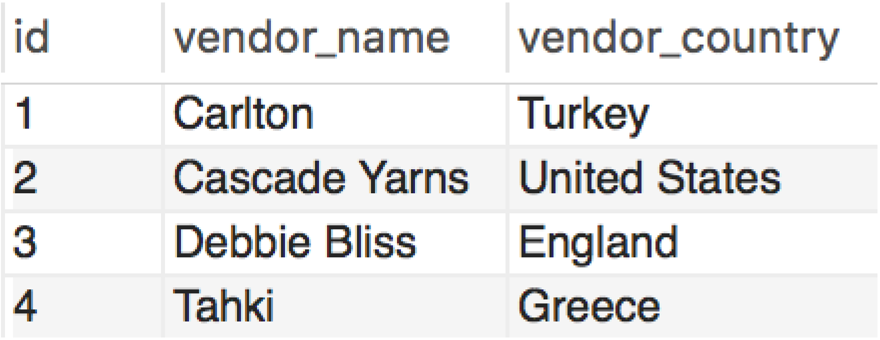
SQL Joins

Part 1: SQL Joins

- Describe the different types of join clauses supported in SQL.

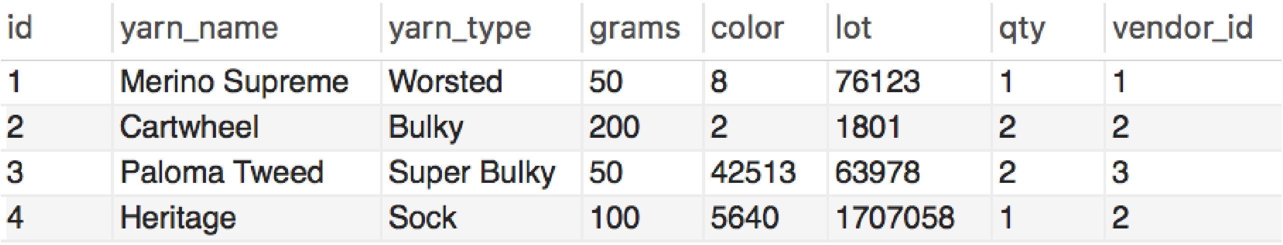
Part 2: SQL Joins

- Consider the following tables:
  - vendor_table



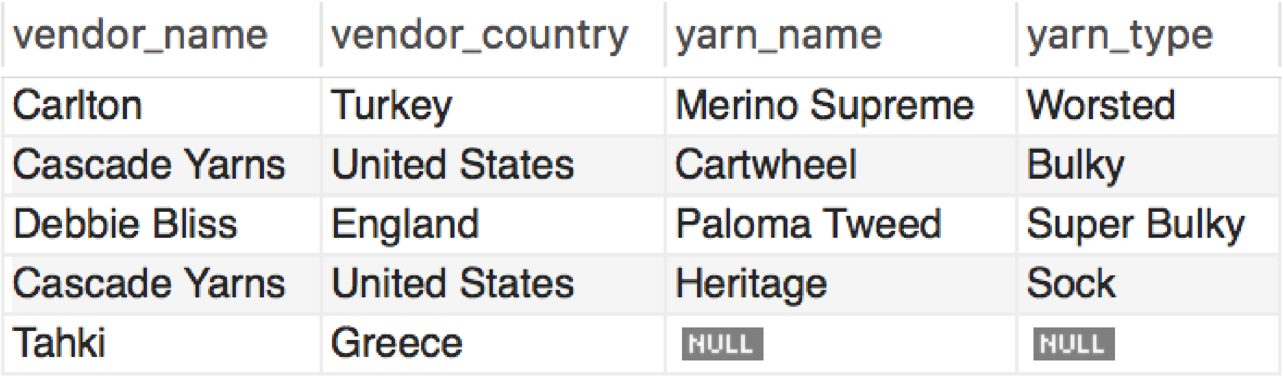
id	vendor_name	vendor_country
1	Carlton	Turkey
2	Cascade Yarns	United States
3	Debbie Bliss	England
4	Tahki	Greece

  - yarn_table



id	yarn_name	yarn_type	grams	color	lot	qty	vendor_id
1	Merino Supreme	Worsted	50	8	76123	1	1
2	Cartwheel	Bulky	200	2	1801	2	2
3	Paloma Tweed	Super Bulky	50	42513	63978	2	3
4	Heritage	Sock	100	5640	1707058	1	2

- Which join was used to create the final view below?



vendor_name	vendor_country	yarn_name	yarn_type
Carlton	Turkey	Merino Supreme	Worsted
Cascade Yarns	United States	Cartwheel	Bulky
Debbie Bliss	England	Paloma Tweed	Super Bulky
Cascade Yarns	United States	Heritage	Sock
Tahki	Greece	NULL	NULL

Solutions: SQL Joins

- Part 1:
  - An inner join will return records at the intersection between two tables.
  - A left join will return all records from Table A and the matching records from Table B.
  - A right join returns all records from Table B and the matching records from Table A.
  - A full join returns a list of all records from both tables.
- Part 2: There are two ways to join these tables.
  - A `Left Join` is performed with the vendor table as the first table referenced (Table A in the explanation above) and the yarn table as the second (Table B).

```
SELECT vendors.vendor_name, vendors.vendor_country,
yarn.yarn_name, yarn.yarn_type
FROM vendors
```

```
LEFT JOIN yarn ON
vendors.id = yarn.vendor_id;
```

- Alternatively, a `Right Join` can also be performed. The difference is that the order of the tables referenced is reversed.

```
SELECT vendors.vendor_name, vendors.vendor_country,
yarn.yarn_name, yarn.yarn_type
FROM yarn
RIGHT JOIN vendors ON
yarn.vendor_id = vendors.id;
```

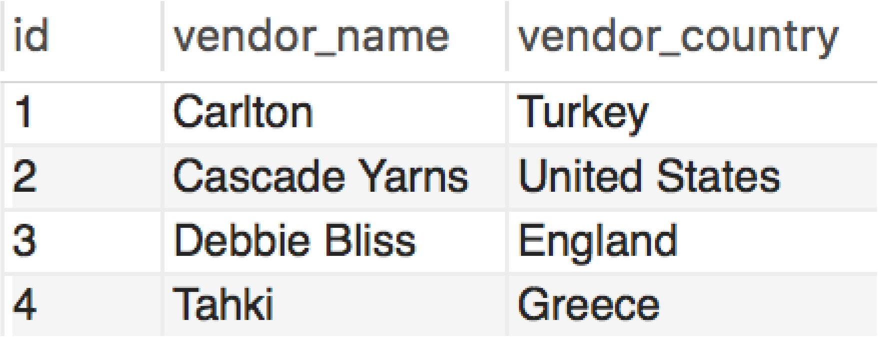
DML & DDL

Part 1: DML & DDL

- What is the difference between `DML` and `DDL` in SQL?

Part 2: DML & DDL

- Demonstrate the use of `DDL` in the following table:
  - vendor_table



id	vendor_name	vendor_country
1	Carlton	Turkey
2	Cascade Yarns	United States
3	Debbie Bliss	England
4	Tahki	Greece

Solutions: DML & DDL

- Part 1: `DML` refers to `Data Manipulation Language`. There are several DML statements used to update, insert, or delete data in a table. `DDL` stands for `Data Definition Language`, which deals with the structure of the data.
  - Examples of `DDL` include `CREATE`, `ALTER`, `DROP`, `TRUNCATE`, `COMMENT`, and `RENAME`.
  - `DML` commands include `SELECT`, `INSERT`, `UPDATE`, and `DELETE`.
- Part 2: several different `DDL` commands are available for use on the table.
  - `DROP` to drop the table altogether.

  - `ALTER` will add one or more columns, modify an existing column, drop a column, rename a column, or rename a table.
  - `CREATE` creates a new table.
  - `DROP` will drop or remove an existing table from a database.
  - `TRUNCATE` will remove all records from an existing table within a database.
  - `COMMENT` is used to add a comment to a data dictionary.
  - `RENAME` can be used to rename existing tables and columns in a database.

Index

Part 1: Index

- Explain an index in SQL.

Part 2: Index

- What are the different types of `index`? If possible, explain each type with an illustration.

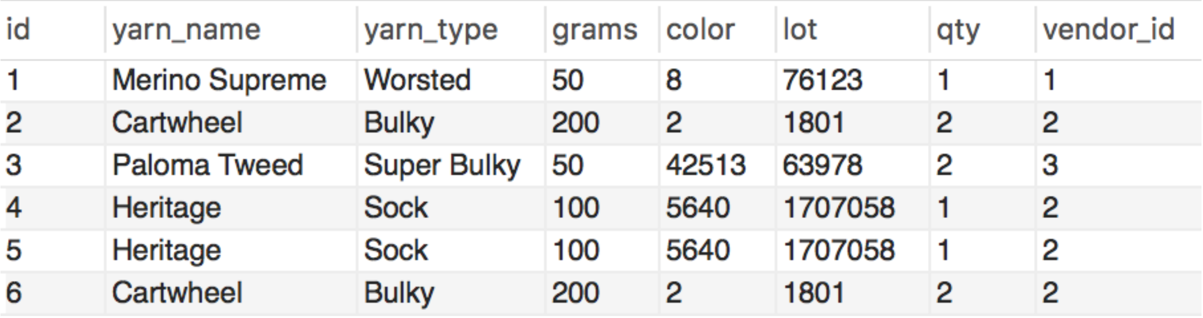
Solutions: Index

- Part 1: an `index` is used to create indexes in tables. They aid in quick data retrieval by speeding up searches and queries. When creating an index, table data is not changed. Instead, a new data structure is created that refers to the table.
- Part 2: There are three types of index:
  - Unique: if a column is uniquely indexed, no duplicate values will be allowed in the field. If a primary key is already defined, then a unique index is automatically applied.
  - Clustered: this type of index will reorder the physical order of the table and searches based off the key values. Only one clustered index is allowed per table.
  - Non-Clustered: a non-clustered index will not alter the physical form of the table. More than one non-clustered indexes are allowed per table.

Duplicates

Part 1: Duplicates

- How do you locate a duplicate record with one field? Using the table below, write a query to demonstrate.
  - Yarn table with duplicates:



id	yarn_name	yarn_type	grams	color	lot	qty	vendor_id
1	Merino Supreme	Worsted	50	8	76123	1	1
2	Cartwheel	Bulky	200	2	1801	2	2
3	Paloma Tweed	Super Bulky	50	42513	63978	2	3
4	Heritage	Sock	100	5640	1707058	1	2
5	Heritage	Sock	100	5640	1707058	1	2
6	Cartwheel	Bulky	200	2	1801	2	2

Part 2: Duplicates

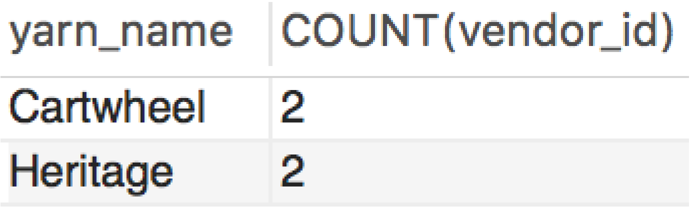
- How do you find duplicate records using more than one field? Using the table from Part 1, write a query to demonstrate.

Solutions: Duplicates

- Part 1:

```
SELECT yarn_name, COUNT(vendor_id)
FROM yarn
GROUP BY yarn_name
HAVING COUNT (vendor_id) > 1;
```

  - Result:

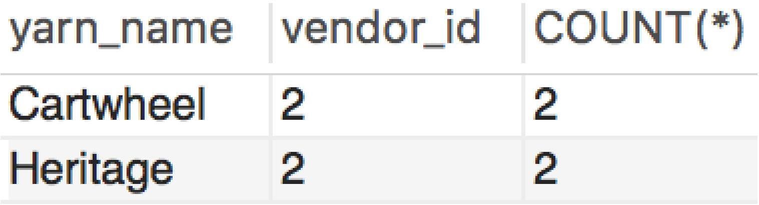


yarn_name	COUNT(vendor_id)
Cartwheel	2
Heritage	2

- Part 2:

```
SELECT yarn_name, vendor_id, COUNT(*)
FROM yarn
GROUP BY yarn_name, vendor_id
HAVING COUNT(*) > 1;
```

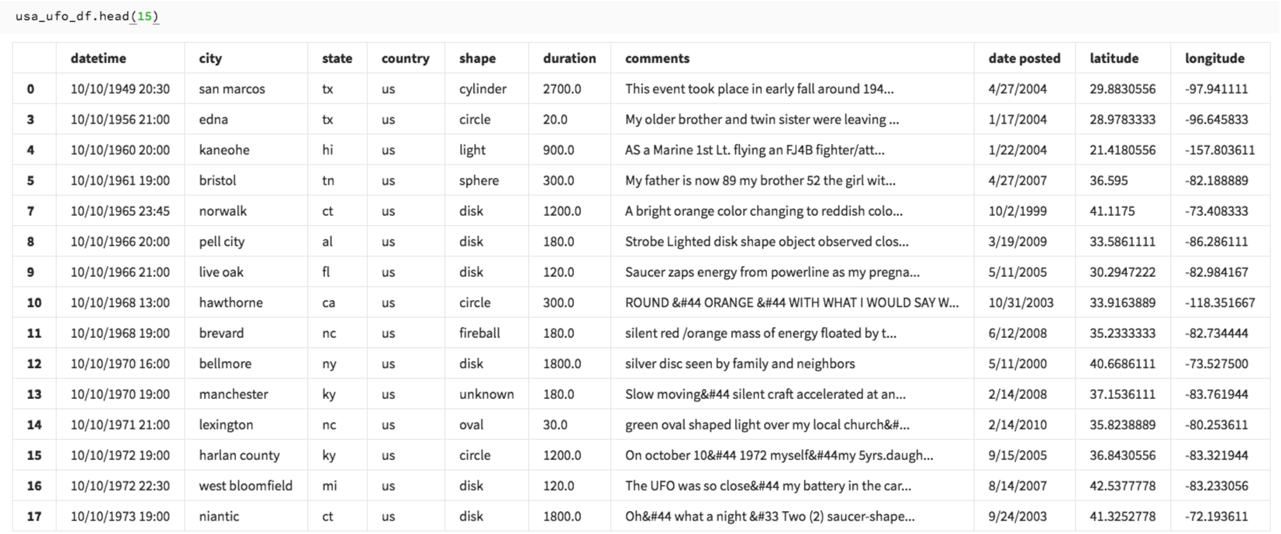
  - Result:



yarn_name	vendor_id	COUNT(*)
Cartwheel	2	2
Heritage	2	2

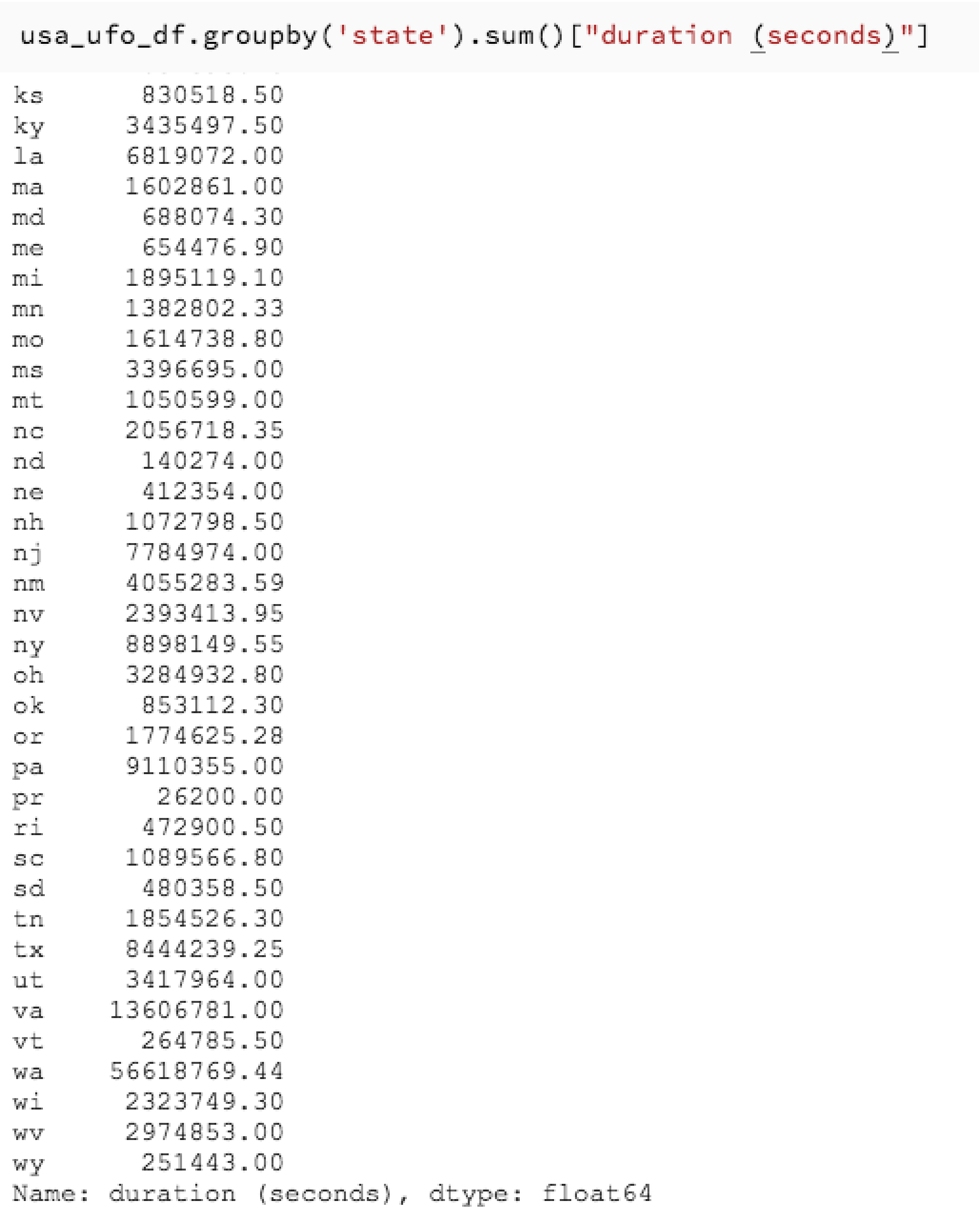
Groupby, Don't Cry

- The below are a pandas data frame preview and a query for the total duration (in seconds) of UFO sightings by state, respectively.



```
usa_ufo_df.head(15)
```

	datetime	city	state	country	shape	duration	comments	date posted	latitude	longitude
0	10/10/1949 20:30	san marcos	tx	us	cylinder	2700.0	This event took place in early fall around 194...	4/27/2004	29.8830556	-97.941111
3	10/10/1956 21:00	edna	tx	us	circle	20.0	My older brother and twin sister were leaving ...	1/17/2004	28.9783333	-96.645833
4	10/10/1960 20:00	kaneohe	hi	us	light	900.0	AS a Marine 1st Lt. flying an FJ4B fighter/att...	1/22/2004	21.4180556	-157.803611
5	10/10/1961 19:00	bristol	tn	us	sphere	300.0	My father is now 89 my brother 52 the girl wit...	4/27/2007	36.595	-82.188889
7	10/10/1965 23:45	norwalk	ct	us	disk	1200.0	A bright orange color changing to reddish colo...	10/2/1999	41.1175	-73.408333
8	10/10/1966 20:00	pell city	al	us	disk	180.0	Strobe Lighted disk shape object observed clos...	3/19/2009	33.5861111	-86.286111
9	10/10/1966 21:00	live oak	fl	us	disk	120.0	Saucer zaps energy from powerline as my pregna...	5/11/2005	30.2947222	-82.984167
10	10/10/1968 13:00	hawthorne	ca	us	circle	300.0	ROUND &#44 ORANGE &#44 WITH WHAT I WOULD SAY W...	10/31/2003	33.9163889	-118.351667
11	10/10/1968 19:00	brevard	nc	us	fireball	180.0	silent red /orange mass of energy floated by t...	6/12/2008	35.2333333	-82.734444
12	10/10/1970 16:00	bellmore	ny	us	disk	1800.0	silver disc seen by family and neighbors	5/11/2000	40.6686111	-73.527500
13	10/10/1970 19:00	manchester	ky	us	unknown	180.0	Slow moving&#44 silent craft accelerated at an...	2/14/2008	37.1536111	-83.761944
14	10/10/1971 21:00	lexington	nc	us	oval	30.0	green oval shaped light over my local church&#...	2/14/2010	35.8238889	-80.253611
15	10/10/1972 19:00	harlan county	ky	us	circle	1200.0	On october 10&#44 1972 myself&#44my 5yrs.daugh...	9/15/2005	36.8430556	-83.321944
16	10/10/1972 22:30	west bloomfield	mi	us	disk	120.0	The UFO was so close&#44 my battery in the car...	8/14/2007	42.5377778	-83.233056
17	10/10/1973 19:00	niantic	ct	us	disk	1800.0	Oh&#44 what a night &#33 Two (2) saucer-shape...	9/24/2003	41.3252778	-72.193611



```
usa_ufo_df.groupby('state').sum()["duration (seconds)"]
```

```
ks       830518.50
ky      3435497.50
la      6819072.00
ma      1602861.00
md       688074.30
me       654476.90
mi      1895119.10
mn      1382802.33
mo      1614738.80
ms      3396695.00
mt      1050599.00
nc      2056718.35
nd       140274.00
ne       412354.00
nh      1072798.50
nj      7784974.00
nm      4055283.59
nv      2393413.95
ny      8898149.55
oh      3284932.80
ok       853112.30
or      1774625.28
pa      9110355.00
pr        26200.00
ri       472900.50
sc      1089566.80
sd       480358.50
tn      1854526.30
tx      8444239.25
ut      3417964.00
va     13606781.00
vt       264785.50
wa     56618769.44
wi      2323749.30
wv      2974853.00
wy       251443.00
Name: duration (seconds), dtype: float64
```

- What is an equivalent SQL query? Instead of the sum, find the mean duration by state.

Solution: Groupby, Don't Cry

```sql
SELECT state, AVG(duration)
FROM usa_ufo
GROUP BY state;
```