1. Classic American names

Photo by Travis Wise on Wikimedia.

How have American baby name tastes changed since 1920? Which names have remained popular for over 100 years, and how do those names compare to more recent top baby names? These are considerations for many new parents, but the skills we'll practice while answering these queries are broadly applicable. After all, understanding trends and popularity is important for many businesses, too!

We'll be working with data provided by the United States Social Security Administration, which lists first names along with the number and sex of babies they were given to in each year. For processing speed purposes, we've limited the dataset to first names which were given to over 5,000 American babies in a given year. Our data spans 101 years, from 1920 through 2020.

`baby_names`

column	type	meaning
`year`	int	year
`first_name`	varchar	first name
`sex`	varchar	`sex` of babies given `first_name`

column	type	meaning
`num`	int	number of babies of `sex` given `first_name` in that `year`

Let's get oriented to American baby name tastes by looking at the names that have stood the test of time!

In [2]:

```
%%sql
postgresql:///names

-- Select first names and the total babies with that first_name
-- Group by first_name and filter for those names that appear in all 101 years
-- Order by the total number of babies with that first_name, descending

SELECT first_name, SUM(num)
FROM baby_names
GROUP BY first_name
HAVING COUNT(year) >= 101
ORDER BY SUM(num) DESC
```

```
8 rows affected.
```

Out[2]:

first_name	sum
James	4748138
John	4510721
William	3614424
David	3571498
Joseph	2361382
Thomas	2166802
Charles	2112352
Elizabeth	1436286

In [3]:

```
%%nose
last_output = _

def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (8, 2), \
    "The query should return eight rows and two columns."
    assert results.columns.tolist() == ["first_name", "sum"], \
    'The results should have two columns: "first_name" and "sum".'
    assert last_output.DataFrame().loc[0, 'first_name'] == 'James', \
    "The first_name in the first row should be James."
    assert last_output.DataFrame().loc[0, 'sum'] == 4748138, \
    "There should be 4,748,138 babies ever named James."
```

Out[3]:
```
2/2 tests passed
```

2. Timeless or trendy?

Wow, it looks like there are a lot of timeless traditionally male names! Elizabeth is holding her own for the female names, too.

Now, let's broaden our understanding of the dataset by looking at all names. We'll attempt to capture the type of popularity that each name in the dataset enjoyed. Was the name classic and popular across many years or trendy, only popular for a few years? Let's find out.

In [4]:
```
%%sql

-- Classify first names as 'Classic', 'Semi-classic', 'Semi-trendy', or 'Trendy'
-- Alias this column as popularity_type
-- Select first_name, the sum of babies who have ever had that name, and popularity_typ
-- Order the results alphabetically by first_name


SELECT
    first_name,
    SUM(num),
    (CASE
     WHEN count(*) > 80 THEN 'Classic'
     WHEN count(*) > 50 THEN 'Semi-classic'
     WHEN count(*) > 20 THEN 'Semi-trendy'
     ELSE 'Trendy'
     END) AS popularity_type

FROM baby_names
GROUP BY first_name
ORDER BY first_name
```

```
 * postgresql:///names
547 rows affected.
```

Out[4]:

first_name	sum	popularity_type
Aaliyah	15870	Trendy
Aaron	530592	Semi-classic
Abigail	338485	Semi-trendy
Adam	497293	Semi-trendy
Addison	107433	Trendy
Adrian	147741	Semi-trendy
Aidan	68566	Trendy
Aiden	216194	Trendy
Alan	162041	Semi-trendy
Albert	260945	Semi-trendy
Alex	158677	Semi-trendy

first_name	sum	popularity_type
Alexa	33522	Trendy
Alexander	579854	Semi-trendy
Alexandra	167122	Semi-trendy
Alexandria	5026	Trendy
Alexis	282149	Semi-trendy
Alfred	16260	Trendy
Alice	296559	Semi-trendy
Alicia	84579	Trendy
Allen	10256	Trendy
Allison	214995	Semi-trendy
Alyssa	269134	Semi-trendy
Amanda	699911	Semi-trendy
Amber	313418	Semi-trendy
Amelia	106381	Trendy
Amy	569542	Semi-trendy
Andrea	321655	Semi-trendy
Andrew	1157548	Semi-classic
Angel	157667	Trendy
Angela	541553	Semi-trendy
Angelina	11337	Trendy
Anita	44692	Trendy
Ann	336091	Semi-trendy
Anna	445496	Semi-classic
Anne	70228	Trendy
Annette	49954	Trendy
Annie	95837	Trendy
Anthony	1344352	Classic
Antonio	10097	Trendy
April	138714	Trendy
Aria	52145	Trendy
Ariana	5497	Trendy
Arianna	5270	Trendy
Ariel	5410	Trendy

first_name	sum	popularity_type
Arthur	309705	Semi-trendy
Asher	38156	Trendy
Ashley	798738	Semi-trendy
Ashton	5436	Trendy
Aubrey	72220	Trendy
Audrey	48341	Trendy
Aurora	5184	Trendy
Austin	365295	Semi-trendy
Ava	265126	Trendy
Avery	112293	Trendy
Ayden	34244	Trendy
Bailey	10219	Trendy
Barbara	1343901	Semi-classic
Barry	85434	Trendy
Beatrice	27983	Trendy
Bella	5127	Trendy
Benjamin	627696	Semi-trendy
Bentley	16844	Trendy
Bernice	46347	Trendy
Beth	55228	Trendy
Betty	893396	Semi-trendy
Beverly	310683	Semi-trendy
Billy	270759	Semi-trendy
Blake	48795	Trendy
Bobby	203289	Semi-trendy
Bonnie	193352	Semi-trendy
Bradley	147275	Semi-trendy
Brandi	16199	Trendy
Brandon	729832	Semi-trendy
Brandy	48762	Trendy
Brayden	93754	Trendy
Brenda	513283	Semi-trendy
Brian	1107302	Semi-classic

first_name	sum	popularity_type
Briana	5001	Trendy
Brianna	210328	Semi-trendy
Brittany	326255	Trendy
Brittney	37878	Trendy
Brody	21815	Trendy
Brooke	110847	Trendy
Brooklyn	62260	Trendy
Bruce	266549	Semi-trendy
Bryan	314250	Semi-trendy
Caden	5052	Trendy
Caitlin	38501	Trendy
Caleb	259439	Semi-trendy
Cameron	253711	Semi-trendy
Camila	51882	Trendy
Carl	334800	Semi-trendy
Carla	22276	Trendy
Carlos	118325	Trendy
Carol	740607	Semi-trendy
Carole	21186	Trendy
Caroline	5021	Trendy
Carolyn	438382	Semi-trendy
Carrie	77899	Trendy
Carson	25598	Trendy
Carter	141274	Trendy
Cassandra	42677	Trendy
Catherine	345852	Semi-trendy
Cathy	119020	Trendy
Chad	177923	Trendy
Charles	2112352	Classic
Charlotte	141540	Trendy
Chase	97684	Trendy
Chelsea	100857	Trendy
Cheryl	392691	Semi-trendy

first_name	sum	popularity_type
Chloe	187720	Semi-trendy
Chris	48878	Trendy
Christian	357457	Semi-trendy
Christina	366859	Semi-trendy
Christine	465653	Semi-trendy
Christopher	2012792	Semi-classic
Christy	10235	Trendy
Cindy	161070	Trendy
Claire	10225	Trendy
Clara	21336	Trendy
Clarence	77134	Trendy
Cody	225952	Semi-trendy
Cole	72461	Trendy
Colin	5122	Trendy
Colton	58377	Trendy
Connie	179476	Semi-trendy
Connor	203106	Semi-trendy
Cooper	31011	Trendy
Corey	70531	Trendy
Cory	29454	Trendy
Courtney	202829	Semi-trendy
Craig	190323	Semi-trendy
Crystal	239999	Semi-trendy
Curtis	48098	Trendy
Cynthia	577859	Semi-trendy
Dakota	34334	Trendy
Dale	130919	Trendy
Dana	51558	Trendy
Daniel	1824274	Classic
Danielle	299683	Semi-trendy
Danny	161078	Trendy
Darlene	89561	Trendy
Darren	5935	Trendy

first_name	sum	popularity_type
Darryl	10142	Trendy
David	3571498	Classic
Dawn	225877	Semi-trendy
Debbie	138846	Trendy
Deborah	675049	Semi-trendy
Debra	508230	Semi-trendy
Denise	285039	Semi-trendy
Dennis	492221	Semi-trendy
Derek	105026	Trendy
Destiny	100465	Trendy
Devin	70280	Trendy
Diana	172195	Semi-trendy
Diane	453135	Semi-trendy
Diego	46535	Trendy
Dillon	5062	Trendy
Dolores	101453	Trendy
Dominic	67420	Trendy
Donald	1280236	Semi-classic
Donna	762594	Semi-trendy
Doris	336062	Semi-trendy
Dorothy	791084	Semi-trendy
Douglas	426439	Semi-trendy
Dustin	138651	Trendy
Dylan	360776	Semi-trendy
Earl	74214	Trendy
Easton	21820	Trendy
Edith	53687	Trendy
Edna	63698	Trendy
Edward	1013143	Semi-classic
Elaine	100359	Trendy
Eleanor	119863	Trendy
Eli	74938	Trendy
Elias	16965	Trendy

first_name	sum	popularity_type
Elijah	277457	Semi-trendy
Elizabeth	1436286	Classic
Ella	154079	Trendy
Ellen	82403	Trendy
Ellie	26266	Trendy
Emily	750420	Semi-trendy
Emma	448087	Semi-trendy
Eric	797880	Semi-classic
Erica	156158	Trendy
Erin	239718	Semi-trendy
Ernest	43959	Trendy
Esther	28040	Trendy
Ethan	408918	Semi-trendy
Ethel	53359	Trendy
Eugene	239512	Semi-trendy
Evan	203165	Semi-trendy
Evelyn	310824	Semi-trendy
Ezekiel	5013	Trendy
Ezra	24632	Trendy
Faith	27204	Trendy
Florence	99171	Trendy
Frances	348520	Semi-trendy
Francis	59097	Trendy
Frank	596887	Semi-classic
Franklin	5364	Trendy
Fred	170837	Semi-trendy
Gabriel	270207	Semi-trendy
Gabriella	51486	Trendy
Gabrielle	51144	Trendy
Gail	118334	Trendy
Garrett	15976	Trendy
Gary	817491	Semi-trendy
Gavin	130460	Trendy

first_name	sum	popularity_type
George	1032513	Semi-classic
Gerald	327545	Semi-trendy
Geraldine	52850	Trendy
Gertrude	16229	Trendy
Gianna	7826	Trendy
Gina	46380	Trendy
Gladys	77627	Trendy
Glenn	94437	Trendy
Gloria	331698	Semi-trendy
Grace	254573	Semi-trendy
Grayson	61689	Trendy
Greg	15849	Trendy
Gregory	644286	Semi-trendy
Hailey	111571	Trendy
Haley	106119	Trendy
Hannah	392284	Semi-trendy
Harold	368463	Semi-trendy
Harper	86554	Trendy
Harry	182312	Semi-trendy
Hayden	34724	Trendy
Hazel	66103	Trendy
Heather	468165	Semi-trendy
Helen	569998	Semi-trendy
Henry	429656	Semi-classic
Herbert	52341	Trendy
Holly	54634	Trendy
Howard	157144	Semi-trendy
Hudson	43048	Trendy
Hunter	220439	Semi-trendy
Ian	159100	Semi-trendy
Irene	110116	Trendy
Isaac	209563	Semi-trendy
Isabella	336924	Semi-trendy

first_name	sum	popularity_type
Isaiah	200116	Semi-trendy
Jace	23233	Trendy
Jack	552411	Semi-classic
Jackson	219588	Semi-trendy
Jacob	888209	Semi-trendy
Jacqueline	223092	Semi-trendy
Jaden	21777	Trendy
Jaime	13744	Trendy
James	4748138	Classic
Jamie	157417	Trendy
Jane	195627	Semi-trendy
Janet	444842	Semi-trendy
Janice	328335	Semi-trendy
Jared	137848	Semi-trendy
Jasmine	192464	Semi-trendy
Jason	998257	Semi-trendy
Jaxon	66542	Trendy
Jaxson	5061	Trendy
Jay	20682	Trendy
Jayden	213005	Trendy
Jean	363812	Semi-trendy
Jeff	61390	Trendy
Jeffery	131241	Trendy
Jeffrey	907378	Semi-trendy
Jenna	60548	Trendy
Jennifer	1404743	Semi-trendy
Jeremiah	104792	Trendy
Jeremy	366271	Semi-trendy
Jerry	494469	Semi-trendy
Jesse	183645	Semi-trendy
Jessica	994210	Semi-trendy
Jesus	82359	Trendy
Jill	120405	Trendy

first_name	sum	popularity_type
Jim	16588	Trendy
Jimmy	135558	Semi-trendy
Jo	84238	Trendy
Joan	413286	Semi-trendy
Joanne	77323	Trendy
Jocelyn	5292	Trendy
Joe	302127	Semi-trendy
John	4510721	Classic
Johnny	138737	Semi-trendy
Jonathan	772846	Semi-classic
Jordan	426912	Semi-trendy
Jose	434805	Semi-trendy
Joseph	2361382	Classic
Josephine	69222	Trendy
Joshua	1204236	Semi-trendy
Josiah	74001	Trendy
Joyce	436267	Semi-trendy
Juan	189094	Semi-trendy
Juanita	20758	Trendy
Judith	377449	Semi-trendy
Judy	329356	Semi-trendy
Julia	112397	Trendy
Julian	137742	Trendy
Julie	411989	Semi-trendy
June	61668	Trendy
Justin	729931	Semi-trendy
Kaitlyn	121541	Trendy
Karen	892033	Semi-trendy
Katelyn	65023	Trendy
Katherine	413349	Semi-classic
Kathleen	516918	Semi-trendy
Kathryn	204173	Semi-trendy
Kathy	269922	Semi-trendy

first_name	sum	popularity_type
Katie	97378	Trendy
Kayla	294192	Semi-trendy
Kaylee	65209	Trendy
Keith	313978	Semi-trendy
Kelly	417352	Semi-trendy
Kelsey	95434	Trendy
Kenneth	1153846	Semi-classic
Kevin	1140092	Semi-classic
Khloe	5406	Trendy
Kim	143648	Trendy
Kimberly	767543	Semi-trendy
Kristen	127223	Trendy
Kristin	81495	Trendy
Kristina	10585	Trendy
Kristy	5331	Trendy
Krystal	5935	Trendy
Kyle	394877	Semi-trendy
Kylie	16309	Trendy
Landon	129558	Trendy
Larry	700521	Semi-trendy
Latoya	5051	Trendy
Laura	587161	Semi-trendy
Lauren	401513	Semi-trendy
Laurie	87568	Trendy
Lawrence	307238	Semi-trendy
Layla	74474	Trendy
Leah	63600	Trendy
Leo	32643	Trendy
Leonard	54127	Trendy
Leslie	73075	Trendy
Levi	91814	Trendy
Liam	213059	Trendy
Lillian	185120	Semi-trendy

first_name	sum	popularity_type
Lily	115354	Trendy
Lincoln	43147	Trendy
Linda	1361021	Semi-trendy
Lindsay	69178	Trendy
Lindsey	88669	Trendy
Lisa	920119	Semi-trendy
Logan	316927	Semi-trendy
Lois	220781	Semi-trendy
Lori	289439	Semi-trendy
Lorraine	27338	Trendy
Louis	115731	Trendy
Louise	106456	Trendy
Lucas	191033	Trendy
Lucille	59379	Trendy
Luis	144176	Semi-trendy
Luke	207795	Semi-trendy
Luna	27822	Trendy
Lynn	97059	Trendy
Mackenzie	46972	Trendy
Madeline	26888	Trendy
Madison	378127	Semi-trendy
Makayla	55446	Trendy
Marc	5013	Trendy
Marcia	15571	Trendy
Marcus	53788	Trendy
Margaret	806838	Semi-trendy
Maria	417502	Semi-classic
Mariah	15723	Trendy
Marie	249462	Semi-trendy
Marilyn	286722	Semi-trendy
Marion	50545	Trendy
Marissa	29003	Trendy
Marjorie	114386	Trendy

first_name	sum	popularity_type
Mark	1265910	Semi-classic
Marlene	10368	Trendy
Marsha	21303	Trendy
Martha	359762	Semi-trendy
Martin	56023	Trendy
Mary	3215850	Classic
Mason	263609	Semi-trendy
Mateo	45440	Trendy
Matthew	1567204	Semi-classic
Maverick	16863	Trendy
Maya	5047	Trendy
Megan	384668	Semi-trendy
Melanie	43995	Trendy
Melissa	666250	Semi-trendy
Mia	216167	Trendy
Michael	4278824	Classic
Michele	139690	Trendy
Michelle	736097	Semi-trendy
Mike	97902	Trendy
Mila	28047	Trendy
Mildred	195666	Trendy
Miles	5249	Trendy
Miranda	11434	Trendy
Misty	34935	Trendy
Mitchell	5370	Trendy
Monica	111143	Trendy
Morgan	157320	Trendy
Nancy	854761	Semi-trendy
Natalie	266634	Semi-trendy
Nathan	493746	Semi-trendy
Nathaniel	103671	Trendy
Nevaeh	42926	Trendy
Nicholas	777269	Semi-trendy

first_name	sum	popularity_type
Nicole	533803	Semi-trendy
Noah	389490	Semi-trendy
Nolan	38147	Trendy
Nora	34285	Trendy
Norma	144522	Semi-trendy
Norman	47596	Trendy
Oliver	107511	Trendy
Olivia	429118	Semi-trendy
Owen	151569	Trendy
Paige	48894	Trendy
Paisley	5085	Trendy
Pamela	524481	Semi-trendy
Parker	27453	Trendy
Patricia	1479802	Semi-classic
Patrick	559661	Semi-classic
Paul	1218996	Semi-classic
Paula	196090	Semi-trendy
Pauline	64073	Trendy
Peggy	220586	Semi-trendy
Penelope	43409	Trendy
Penny	10128	Trendy
Peter	388795	Semi-classic
Peyton	5315	Trendy
Philip	100415	Trendy
Phillip	86811	Trendy
Phyllis	251517	Semi-trendy
Rachel	434626	Semi-trendy
Ralph	273663	Semi-trendy
Randall	89055	Trendy
Randy	215094	Trendy
Raymond	541922	Semi-classic
Rebecca	638458	Semi-classic
Regina	10003	Trendy

first_name	sum	popularity_type
Renee	61185	Trendy
Rhonda	157706	Trendy
Richard	2414838	Classic
Rick	5462	Trendy
Ricky	119547	Trendy
Riley	73607	Trendy
Rita	125877	Semi-trendy
Robert	4495199	Classic
Robin	210806	Trendy
Rodney	125500	Trendy
Roger	314531	Semi-trendy
Ronald	974343	Semi-classic
Ronnie	45564	Trendy
Rose	248527	Semi-trendy
Roy	227920	Semi-trendy
Ruby	93528	Trendy
Russell	128647	Semi-trendy
Ruth	475908	Semi-trendy
Ryan	926995	Semi-trendy
Sabrina	11589	Trendy
Sally	30713	Trendy
Samantha	514826	Semi-trendy
Samuel	539556	Semi-classic
Sandra	783878	Semi-trendy
Santiago	5036	Trendy
Sara	226696	Semi-trendy
Sarah	777519	Semi-trendy
Savannah	134405	Semi-trendy
Scarlett	54329	Trendy
Scott	704468	Semi-trendy
Sean	372082	Semi-trendy
Sebastian	130244	Trendy
Seth	35423	Trendy

first_name	sum	popularity_type
Shane	52869	Trendy
Shannon	231132	Semi-trendy
Sharon	647989	Semi-trendy
Shaun	6107	Trendy
Shawn	215326	Semi-trendy
Sheila	154361	Semi-trendy
Shelby	68474	Trendy
Sherri	10819	Trendy
Sherry	173913	Semi-trendy
Shirley	615887	Semi-trendy
Sierra	38980	Trendy
Skylar	10408	Trendy
Sofia	117208	Trendy
Sophia	318523	Semi-trendy
Stacey	67483	Trendy
Stacy	86835	Trendy
Stanley	95152	Trendy
Stella	10217	Trendy
Stephanie	651976	Semi-trendy
Stephen	753958	Semi-classic
Steve	114750	Trendy
Steven	1216819	Semi-classic
Sue	10450	Trendy
Susan	1025728	Semi-trendy
Suzanne	109387	Trendy
Sydney	117279	Trendy
Tammy	296905	Trendy
Tanya	22407	Trendy
Tara	107987	Trendy
Taylor	323699	Semi-trendy
Teresa	298059	Semi-trendy
Terri	80961	Trendy
Terry	346213	Semi-trendy

first_name	sum	popularity_type
Thelma	74017	Trendy
Theodore	29464	Trendy
Theresa	225262	Semi-trendy
Thomas	2166802	Classic
Tiffany	283969	Semi-trendy
Tim	36165	Trendy
Timothy	1001771	Semi-classic
Tina	227252	Semi-trendy
Todd	207137	Trendy
Tom	5061	Trendy
Tony	96417	Trendy
Tonya	58234	Trendy
Tracey	16979	Trendy
Tracy	199320	Trendy
Travis	218731	Semi-trendy
Trevor	76138	Trendy
Trinity	16217	Trendy
Tristan	27212	Trendy
Troy	82294	Trendy
Tyler	548624	Semi-trendy
Valerie	70039	Trendy
Vanessa	119596	Trendy
Vicki	94504	Trendy
Vickie	49252	Trendy
Victoria	347794	Semi-trendy
Vincent	23419	Trendy
Violet	10471	Trendy
Virginia	441418	Semi-trendy
Walter	378194	Semi-trendy
Wanda	125458	Trendy
Warren	13290	Trendy
Wayne	211347	Semi-trendy
Wendy	159446	Trendy

first_name	sum	popularity_type
Whitney	43759	Trendy
William	3614424	Classic
Willie	274564	Semi-trendy
Wyatt	128168	Trendy
Xavier	51892	Trendy
Zachary	483955	Semi-trendy
Zoe	78773	Trendy
Zoey	70140	Trendy

In [5]:

```
%%nose
last_output = _

def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (547, 3), \
    "The query should return 547 rows and three columns."
    assert results.columns.tolist() == ["first_name", "sum", "popularity_type"], \
    'The results should have three columns: "first_name", "sum", and "popularity_type".
    assert last_output.DataFrame().loc[0, 'first_name'] == 'Aaliyah', \
    "The first_name in the first row should be Aaliyah. Did you sort first names alphab
    assert last_output.DataFrame().loc[0, 'sum'] == 15870, \
    "There should be 15,870 babies ever named Aaliyah."
    assert last_output.DataFrame().loc[0, 'popularity_type'] == "Trendy", \
    "The name Aaliyah should be classified as 'Trendy'."
```

Out[5]:

```
2/2 tests passed
```

3. Top-ranked female names since 1920

Did you find your favorite American celebrity's name on the popularity chart? Was it classic or trendy? How do you think the name Henry did? What about Jaxon?

Since we didn't get many traditionally female names in our classic American names search in the first task, let's limit our search to names which were given to female babies.

We can use this opportunity to practice window functions by assigning a rank to female names based on the number of babies that have ever been given that name. What are the top-ranked female names since 1920?

In [6]:

```
%%sql
```

```
-- RANK names by the sum of babies who have ever had that name (descending), aliasing a
-- Select name_rank, first_name, and the sum of babies who have ever had that name
-- Filter the data for results where sex equals 'F'
-- Limit to ten results


SELECT
    RANK() OVER(ORDER BY SUM(num) DESC) AS name_rank,
    first_name,
    SUM(num)
FROM baby_names
WHERE sex = 'F'
GROUP BY first_name
LIMIT 10
```

```
 * postgresql:///names
10 rows affected.
```

Out[6]:

name_rank	first_name	sum
1	Mary	3215850
2	Patricia	1479802
3	Elizabeth	1436286
4	Jennifer	1404743
5	Linda	1361021
6	Barbara	1343901
7	Susan	1025728
8	Jessica	994210
9	Lisa	920119
10	Betty	893396

In [7]:

```
%%nose
last_output = _

def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (10, 3), \
    "The query should return ten rows and three columns."
    assert set(results.columns.tolist()) == set(["first_name", "sum", "name_rank"]), \
    'The results should have three columns: "name_rank", "first_name", and "sum".'
    assert last_output.DataFrame().loc[0, 'first_name'] == 'Mary', \
    "The first_name in the first row should be Mary. Did you order so that names given
    assert last_output.DataFrame().loc[0, 'sum'] == 3215850, \
    "There should be 3,215,850 babies ever named Mary."
    assert last_output.DataFrame().loc[0, 'name_rank'] == 1, \
    "The name Mary should be ranked number one."
```

```
Out[7]: 2/2 tests passed
```

4. Picking a baby name

Perhaps a friend has heard of our work analyzing baby names and would like help choosing a name for her baby, a girl. She doesn't like any of the top-ranked names we found in the previous task.

She's set on a traditionally female name ending in the letter 'a' since she's heard that vowels in baby names are trendy. She's also looking for a name that has been popular in the years since 2015.

Let's see what we can do to find some options for this friend!

```
In [8]:
%%sql
-- Select only the first_name column
-- Filter for results where sex is 'F', year is greater than 2015, and first_name ends
-- Group by first_name and order by the total number of babies given that first_name

SELECT first_name
FROM baby_names
WHERE sex = 'F'
    AND year > 2015
    AND first_name LIKE '%a'
GROUP BY first_name
ORDER BY SUM(num) DESC
```

```
 * postgresql:///names
19 rows affected.
```

Out[8]:

first_name
Olivia
Emma
Ava
Sophia
Isabella
Mia
Amelia
Ella
Sofia
Camila
Aria
Victoria
Layla
Nora
Mila
Luna

first_name
Stella
Gianna
Aurora

In [9]:
```
%%nose
last_output = _

def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (19, 1), \
    "The query should return 19 rows and one column."
    assert results.columns.tolist() == ["first_name"], \
    'The results should have one column: "first_name".'
    assert last_output.DataFrame().loc[0, 'first_name'] == 'Olivia', \
    "The first_name in the first row should be Olivia."
```

Out[9]:
```
2/2 tests passed
```

5. The Olivia expansion

Based on the results in the previous task, we can see that Olivia is the most popular female name ending in 'A' since 2015. When did the name Olivia become so popular?

Let's explore the rise of the name Olivia with the help of a window function.

In [10]:
```
%%sql

-- Select year, first_name, num of Olivias in that year, and cumulative_olivias
-- Sum the cumulative babies who have been named Olivia up to that year; alias as cumul
-- Filter so that only data for the name Olivia is returned.
-- Order by year from the earliest year to most recent

SELECT
    year,
    first_name,
    num,
    SUM(num) OVER(ORDER BY year) cumulative_olivias
FROM baby_names
WHERE first_name = 'Olivia'
ORDER BY year
```

```
 * postgresql:///names
30 rows affected.
```

Out[10]:

year	first_name	num	cumulative_olivias

year	first_name	num	cumulative_olivias
1991	Olivia	5601	5601
1992	Olivia	5809	11410
1993	Olivia	6340	17750
1994	Olivia	6434	24184
1995	Olivia	7624	31808
1996	Olivia	8124	39932
1997	Olivia	9477	49409
1998	Olivia	10610	60019
1999	Olivia	11255	71274
2000	Olivia	12852	84126
2001	Olivia	13977	98103
2002	Olivia	14630	112733
2003	Olivia	16152	128885
2004	Olivia	16106	144991
2005	Olivia	15694	160685
2006	Olivia	15501	176186
2007	Olivia	16584	192770
2008	Olivia	17084	209854
2009	Olivia	17438	227292
2010	Olivia	17029	244321
2011	Olivia	17327	261648
2012	Olivia	17320	278968
2013	Olivia	18439	297407
2014	Olivia	19823	317230
2015	Olivia	19710	336940
2016	Olivia	19380	356320
2017	Olivia	18744	375064
2018	Olivia	18011	393075
2019	Olivia	18508	411583
2020	Olivia	17535	429118

In [11]:
```
%%nose
last_output = _

def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (30, 4), \
    "The query should return thirty rows and four columns."
    assert set(results.columns.tolist()) == set(["year", "first_name", "num", "cumulati
    'The results should have four columns: "year", "first_name", "num", and "cumulative
    assert last_output.DataFrame().loc[0, 'first_name'] == 'Olivia', \
    "The first_name in the first row should be Olivia. Did you filter so that results o
    assert last_output.DataFrame().loc[0, 'num'] == 5601, \
    "In 1991, there should have been 5,601 female babies named Olivia."
    assert last_output.DataFrame().loc[0, 'year'] == 1991, \
    "1991 should be the first year that Olivia appears in the results. Did you sort by
    assert last_output.DataFrame().loc[1, 'cumulative_olivias'] == 11410, \
    "In 1992, the cumulative_olivias column should read 11,410."
```

Out[11]:
```
2/2 tests passed
```

6. Many males with the same name

Wow, Olivia has had a meteoric rise! Let's take a look at traditionally male names now. We saw in the first task that there are nine traditionally male names given to at least 5,000 babies every single year in our 101-year dataset! Those names are classics, but showing up in the dataset every year doesn't necessarily mean that the timeless names were the most popular. Let's explore popular male names a little further.

In the next two tasks, we will build up to listing every year along with the most popular male name in that year. This presents a common problem: how do we find the greatest X in a group? Or, in the context of this problem, how do we find the male name given to the highest number of babies in a year?

In SQL, one approach is to use a subquery. We can first write a query that selects the `year` and the maximum `num` of babies given any single male name in that year. For example, in 1989, the male name given to the highest number of babies was given to 65,339 babies. We'll write this query in this task. In the next task, we can use the code from this task as a subquery to look up the `first_name` that was given to 65,339 babies in 1989... as well as the top male first name for all other years!

In [12]:
```
%%sql

-- Select year and maximum number of babies given any one male name in that year, alias
-- Filter the data to include only results where sex equals 'M'

SELECT
    year,
    MAX(num) AS max_num
FROM baby_names
WHERE sex = 'M'
```

```
GROUP BY year
ORDER BY year DESC
```

```
 * postgresql:///names
101 rows affected.
```

Out[12]:

year	max_num
2020	19659
2019	20555
2018	19924
2017	18824
2016	19154
2015	19650
2014	19319
2013	18266
2012	19088
2011	20378
2010	22139
2009	21184
2008	22603
2007	24292
2006	24850
2005	25837
2004	27886
2003	29643
2002	30579
2001	32554
2000	34483
1999	35367
1998	36616
1997	37549
1996	38365
1995	41399
1994	44472
1993	49554
1992	54397
1991	60793
1990	65302

year	max_num
1989	65399
1988	64150
1987	63654
1986	64224
1985	64924
1984	67745
1983	68010
1982	68244
1981	68776
1980	68704
1979	67742
1978	67157
1977	67609
1976	66947
1975	68451
1974	67580
1973	67842
1972	71401
1971	77599
1970	85291
1969	85201
1968	81995
1967	82440
1966	79990
1965	81021
1964	82642
1963	83778
1962	85041
1961	86917
1960	85933
1959	85224
1958	90564
1957	92718

year	max_num
1956	90665
1955	88372
1954	88576
1953	86247
1952	87063
1951	87261
1950	86229
1949	86865
1948	88589
1947	94764
1946	87439
1945	74460
1944	76954
1943	80274
1942	77174
1941	66743
1940	62476
1939	59653
1938	62269
1937	61842
1936	58499
1935	56522
1934	55834
1933	54223
1932	59265
1931	60518
1930	62149
1929	59804
1928	60703
1927	61671
1926	61130
1925	60897
1924	60801

year	max_num
1923	57469
1922	57280
1921	58215
1920	56914

In [13]:

```
%%nose
import numpy as np
last_output = _

def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (101, 2), \
    "The query should return 101 rows and two columns."
    assert set(results.columns.tolist()) == set(["year", "max_num"]), \
    'The results should have two columns: "year" and "max_num".'
    assert last_output.DataFrame().loc[list(np.where(last_output.DataFrame() == 1964)[0
    "In 1964, the name given to the most babies was given 82,642 times."
```

Out[13]:

```
2/2 tests passed
```

7. Top male names over the years

In the previous task, we found the maximum number of babies given any one male name in each year. Incredibly, the most popular name each year varied from being given to less than 20,000 babies to being given to more than 90,000!

In this task, we find out what that top male name is for each year in our dataset.

In [14]:

```
%%sql

-- Select year, first_name given to the largest number of male babies, and num of babie
-- Join baby_names to the code in the last task as a subquery
-- Order results by year descending

SELECT
    a.year,
    a.first_name,
    a.num
FROM baby_names AS a
JOIN (SELECT
        year,
        MAX(num) AS max_num
    FROM baby_names
    WHERE sex = 'M'
    GROUP BY year
    ORDER BY year) AS b
    ON a.year = b.year AND a.num = b.max_num
ORDER BY year DESC
```

```
 * postgresql:///names
101 rows affected.
```

Out[14]:

year	first_name	num
2020	Liam	19659
2019	Liam	20555
2018	Liam	19924
2017	Liam	18824
2016	Noah	19154
2015	Noah	19650
2014	Noah	19319
2013	Noah	18266
2012	Jacob	19088
2011	Jacob	20378
2010	Jacob	22139
2009	Jacob	21184
2008	Jacob	22603
2007	Jacob	24292
2006	Jacob	24850
2005	Jacob	25837
2004	Jacob	27886
2003	Jacob	29643
2002	Jacob	30579
2001	Jacob	32554
2000	Jacob	34483
1999	Jacob	35367
1998	Michael	36616
1997	Michael	37549
1996	Michael	38365
1995	Michael	41399
1994	Michael	44472
1993	Michael	49554
1992	Michael	54397
1991	Michael	60793

year	first_name	num
1990	Michael	65302
1989	Michael	65399
1988	Michael	64150
1987	Michael	63654
1986	Michael	64224
1985	Michael	64924
1984	Michael	67745
1983	Michael	68010
1982	Michael	68244
1981	Michael	68776
1980	Michael	68704
1979	Michael	67742
1978	Michael	67157
1977	Michael	67609
1976	Michael	66947
1975	Michael	68451
1974	Michael	67580
1973	Michael	67842
1972	Michael	71401
1971	Michael	77599
1970	Michael	85291
1969	Michael	85201
1968	Michael	81995
1967	Michael	82440
1966	Michael	79990
1965	Michael	81021
1964	Michael	82642
1963	Michael	83778
1962	Michael	85041
1961	Michael	86917
1960	David	85933
1959	Michael	85224
1958	Michael	90564

year	first_name	num
1957	Michael	92718
1956	Michael	90665
1955	Michael	88372
1954	Michael	88576
1953	Robert	86247
1952	James	87063
1951	James	87261
1950	James	86229
1949	James	86865
1948	James	88589
1947	James	94764
1946	James	87439
1945	James	74460
1944	James	76954
1943	James	80274
1942	James	77174
1941	James	66743
1940	James	62476
1939	Robert	59653
1938	Robert	62269
1937	Robert	61842
1936	Robert	58499
1935	Robert	56522
1934	Robert	55834
1933	Robert	54223
1932	Robert	59265
1931	Robert	60518
1930	Robert	62149
1929	Robert	59804
1928	Robert	60703
1927	Robert	61671
1926	Robert	61130
1925	Robert	60897

year	first_name	num
1924	Robert	60801
1923	John	57469
1922	John	57280
1921	John	58215
1920	John	56914

In [15]:

```
%%nose
last_output = _

def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (101, 3), \
    "The query should return 101 rows and three columns."
    assert set(results.columns.tolist()) == set(["year", "first_name", "num"]), \
    'The results should have three columns: "year", "first_name", and "num".'
    assert last_output.DataFrame().loc[0, 'year'] == 2020, \
    "The first year should be 2020. Did you sort so that the most recent years appear f
    assert last_output.DataFrame().loc[0, 'first_name'] == "Liam", \
    "In 2020, the name given to the most male babies was Liam."
    assert last_output.DataFrame().loc[0, 'num'] == 19659, \
    "In 2020, the name Liam was given to 19,659 babies."
```

Out[15]:

```
2/2 tests passed
```

8. The most years at number one

Noah and Liam have ruled the roost in the last few years, but if we scroll down in the results, it looks like Michael and Jacob have also spent a good number of years as the top name! Which name has been number one for the largest number of years? Let's use a common table expression to find out.

In [16]:

```
%%sql

-- Select first_name and a count of years it was the top name in the last task; alias a
-- Use the code from the previous task as a common table expression
-- Group by first_name and order by count_top_name descending

SELECT
    first_name,
    COUNT(*) AS count_top_name
FROM (SELECT
        a.year,
        a.first_name,
        a.num
    FROM baby_names AS a
    JOIN (SELECT
            year,
            MAX(num) AS max_num
        FROM baby_names
        WHERE sex = 'M'
        GROUP BY year
        ORDER BY year) AS b
        ON a.year = b.year AND a.num = b.max_num
    ORDER BY year DESC) AS subquery
GROUP BY first_name
ORDER BY COUNT(*) DESC
```

```
 * postgresql:///names
8 rows affected.
```

Out[16]:

first_name	count_top_name
Michael	44
Robert	17
Jacob	14
James	13
Noah	4
John	4
Liam	4
David	1

In [17]:

```
%%nose
last_output = _

def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (8, 2), \
    "The query should return eight rows and two columns."
    assert set(results.columns.tolist()) == set(["first_name", "count_top_name"]), \
    'The results should have two columns: "first_name" and "count_top_name".'
    assert last_output.DataFrame().loc[0, 'first_name'] == 'Michael', \
    "The name that spent most years at number one should be Michael. Did you order from
    assert last_output.DataFrame().loc[0, 'count_top_name'] == 44, \
    "Michael was the number one male name 44 times. It doesn't look like your results r
```

Out[17]:

```
2/2 tests passed
```