

Phase 3 Part A

Émilie Brazeau, Nicholas Gin, Gordon Tang

Part A.1

Query Type: **Drill Down**

Query:

```
SELECT b.branch_name, d.month, rf.surrogate_key, rtd.review_text
FROM review_fact AS rf, date_dimension AS d, review_text_dimension AS rtd,
branch_dimension AS b
WHERE rf.date_key = d.date_key AND rf.review_text_key = rtd.review_text_key
AND rf.review_text_key = rtd.review_text_key AND rf.branch_key = b.branch_key
AND d.year = '2018'
GROUP BY b.branch_name, d.month, rf.surrogate_key, rtd.review_text
```

What does this query do?: Retrieve the branch name, month, surrogate key, and review text for all reviews made in the year 2018, and then group the results by branch, month, surrogate key, and review text.

This is a drill down query because it is applying a filter to the existing data to focus on reviews made in the year 2018.

Query output (first 15 rows):

	branch_name character varying (255)	month integer	surrogate_key integer	review_text text
1	Disneyland California	1	17	dreamed coming disneyland ever since remember 42 year earth finally chance bring wife 3 child 9 12 15 year due age much fun fare desire go ride diminished however look
2	Disneyland California	1	150	awesome experience spent 5 day total 2 day california adventure 2 day disneyland 4 would fine 2 day park loved frozen show car ride course peter pan
3	Disneyland California	1	286	disney get make improvement thing closed many theme park better day get charge upgrade enhance individual ride exact ride themed differently cart ride storybook boat ride
4	Disneyland California	1	572	5 year ago found trip different last time definitely atmosphere happy place longer exists crowded wait long ride even winter many stroller motorised ride constantly watch co
5	Disneyland California	1	937	disneyland much better experience california adventure little boy 2 5 year old le three year old get free still due big crowd queue weekday mid january worth also u mystery n
6	Disneyland California	1	2515	expectation high entire family disneyland disappoint disneyworld ft never disneyland really liked smaller size disneyland weather perfect day got chilly night sure thinking bri
7	Disneyland California	1	2936	busy moment disney great weather quick wait time fast pas say truly land definite fast pas indiana jones ride adventure land thunder railway frontierland bobsled venting ligh
8	Disneyland California	1	2999	beautiful like always found splash mountain closed refurbishment fast pas work inside park one one different orlando select 3 option previously also reserve restaurant like
9	Disneyland California	1	3115	amazing two different park right next door disneyland park disney california adventure park bought hopper ticket allowing access park got one day pas clearly enough time s
10	Disneyland California	1	3131	came son 3 day pas birthday disappointed even thought time year came ride closed still fantastic time every staff member came across lovely due son wearing birthday bac
11	Disneyland California	1	4163	family first time disneyland wonderful time done disney world park previous year everything clean food price reasonable park never waited longer 20 min ride gotta strategy
12	Disneyland California	1	4241	disneyland always great experience le crowded new year day able everything wanted without much waiting
13	Disneyland California	1	4730	disney truly magical especially show would say get map advance make sure catch show definitely want miss fast pas popular ride check go though lot renovation work ride
14	Disneyland California	1	5100	definitely happiest place earth grew going disneyland enjoy day love ride food people would think come ride much food must try
15	Disneyland California	1	5745	celebrated special need aunt 60th birthday park amazing time enjoyed park morning closing fantastic show spectacular would highly recommend seeing firework cancelled

Query type: Roll up

Query:

```
SELECT d.year,  
       AVG(rf.monthly_percent_positive_reviews)  
FROM review_fact AS rf, date_dimension AS d  
WHERE rf.date_key = d.date_key  
GROUP BY ROLLUP(d.year)  
ORDER BY d.year
```

What does this query do?: Calculate the average monthly percentage of positive reviews for each year.

This is a roll up query because it uses the ROLLUP function to group the data by year (which was initially monthly data); it also provides a grand total row (the very last row with a null year value, which indicates the overall average of the monthly percent of positive reviews for all years). There is aggregation involved in this query; we are summarizing monthly totals into yearly totals.

Query output:

	year integer	avg numeric
1	2002	0.00000000000000000000
2	2003	100.000000000000000000
3	2004	65.3061224489795900
4	2005	60.000000000000000010
5	2006	56.666666666666666675
6	2007	66.666666666666666658
7	2008	62.0689655172413777
8	2009	60.9467455621301764
9	2010	65.5040755040755029
10	2011	76.9267498033928781
11	2012	81.4972645363930663
12	2013	80.8299699997865699
13	2014	80.8419860418282198
14	2015	83.1384500559771117
15	2016	82.9607906016077246
16	2017	82.7263671260408333
17	2018	79.6206311446747235
18	2019	76.8753896682429098
19	2020	73.4386365202477673
20	2021	47.0434677027683309
21	[null]	81.0475524341222742

Query type: Slice

Query:

```
SELECT rf.surrogate_key, rf.date_key, rf.branch_key, rf.review_text_key,  
rf.monthly_percent_positive_reviews, rf.monthly_percent_mixed_reviews,  
rf.monthly_percent_negative_reviews  
FROM review_fact AS rf, date_dimension as d  
WHERE rf.date_key = d.date_key AND d.year = 2010
```

What does this query do?: Retrieve all reviews in review_fact that were written in the year 2010.

This is a slice query because it filters the rows in the review_fact table based on a specific condition from the date_dimension: the year the review was written is 2010. The year = 2010 acts as a filter on the data.

Query output (first 15 rows):

	surrogate_key integer	date_key integer	branch_key integer	review_text_key integer	monthly_percent_positive_reviews numeric	monthly_percent_mixed_reviews numeric	monthly_percent_negative_reviews numeric
1	228	96	4	42698	58.333333333333336	8.333333333333332	33.33333333333333
2	754	142	6	88498	62.5	25.0	12.5
3	888	56	4	16045	61.53846153846154	7.6923076923076925	30.76923076923077
4	1078	33	4	70043	63.41463414634146	14.634146341463413	21.951219512195124
5	1206	130	4	85033	66.15384615384615	9.230769230769232	24.615384615384617
6	1449	162	4	88295	44.444444444444444	11.111111111111111	44.444444444444444
7	1494	130	2	10234	100.0	0.0	0.0
8	1532	130	4	61888	66.15384615384615	9.230769230769232	24.615384615384617
9	1659	130	4	20281	66.15384615384615	9.230769230769232	24.615384615384617
10	1680	33	4	68212	63.41463414634146	14.634146341463413	21.951219512195124
11	1737	101	4	84679	87.5	0.0	12.5
12	1759	41	1	50655	93.10344827586206	0.0	6.896551724137931
13	1904	41	3	38279	40.0	10.0	50.0
14	1975	162	4	57881	44.444444444444444	11.111111111111111	44.444444444444444
15	2192	20	1	28863	100.0	0.0	0.0

Query type: Slice

Query:

```
SELECT rf.surrogate_key, rf.date_key, rf.branch_key, rf.review_text_key,  
rf.monthly_percent_positive_reviews, rf.monthly_percent_mixed_reviews,  
rf.monthly_percent_negative_reviews  
FROM review_fact AS rf, review_text_dimension as rt  
WHERE rf.review_text_key = rt.review_text_key AND LENGTH(rt.review_text) < 100
```

What does this query do?: Retrieve all reviews in review_fact that have a review_text that is less than 100 characters long.

This is a slice query because it filters the rows in the review_fact table based on a specific condition from the review_text_dimension: the length of the review_text < 100. The length of the review_text having to be less than 100 acts as a filter on the data.

Query output (first 15 rows):

	surrogate_key integer	date_key [PK] integer	branch_key [PK] integer	review_text_key [PK] integer	monthly_percent_positive_reviews numeric	monthly_percent_mixed_reviews numeric	monthly_percent_negative_reviews numeric
1	2	155	4	84360	79.65517241379311	8.275862068965518	12.068965517241379
2	4	77	1	71513	92.16589861751152	2.7649769585253456	5.0691244239631335
3	6	32	6	16966	93.84615384615384	4.615384615384616	1.5384615384615385
4	21	148	6	61294	88.88888888888889	8.0	3.1111111111111111
5	30	135	6	19302	90.97222222222221	6.25	2.7777777777777777
6	31	126	1	9806	92.1875	5.2083333333333334	2.6041666666666667
7	32	198	3	72289	77.09923664122137	11.450381679389313	11.450381679389313
8	38	169	3	10360	79.19463087248322	14.093959731543624	6.7114093959731544
9	58	65	4	55538	85.63380281690141	7.323943661971831	7.042253521126761
10	60	126	1	54194	92.1875	5.2083333333333334	2.6041666666666667
11	61	188	6	38085	92.1875	5.2083333333333334	2.6041666666666667
12	95	202	1	51904	91.58415841584159	6.9306930693069315	1.4851485148514851
13	98	131	4	63660	73.62204724409449	12.204724409448819	14.173228346456693
14	109	168	1	29302	91.78743961352657	5.314009661835748	2.898550724637681
15	120	87	3	25006	80.8080808080808	13.131313131313133	6.0606060606060606

Query type: Dice

Query:

```
SELECT rt.review_text, b.branch_name
FROM review_fact AS rf, review_text_dimension AS rt, branch_dimension AS b
WHERE rf.branch_key = b.branch_key AND rf.review_text_key = rt.review_text_key
AND rf.branch_key = 1
```

What does this query do?: Retrieve all review texts and the branch names for the branch_key = 1 in the review fact table (the branch is Disneyland California).

This is a dice query because it is creating a subset of the data by selecting specific values from more than one dimension (in this case, the review_text dimension and the branch dimension). We are looking only at the subset of the data containing reviews written for Disneyland California.

Query output (first 15 rows):

	branch_name character varying (255)	review_text text
1	Disneyland California	love everything disneyland many time excited many time planning
2	Disneyland California	husband took three teenager disneyland christmas gift besides nasty case food poisoning flo dinner trip fine
3	Disneyland California	place iid rather original lived closer nearly every day disneyland must go worth every penny could write book place stop
4	Disneyland California	disneyland happy place plan waiting line bag check front gate food parade ride etc need hire help stream line process
5	Disneyland California	never seen many people one place hard get ride long queue visit week day still magical place
6	Disneyland California	line great fun photo tinkerbelle firework way better suspected
7	Disneyland California	everything pretty detailed staff helpful friendly
8	Disneyland California	3 great day usual 3 adult 3 child many attraction closed due upcoming change constructive progress regular visitor 3 day pass
9	Disneyland California	3 1 2 year old husband loved truly magical
10	Disneyland California	surprisingly line long expected july 3 good time go wrong disney
11	Disneyland California	say disneyland awesome even kid wife visit every year october redecorate lot park spooky halloween stuff decoration outside haunted mansion awesome
12	Disneyland California	seriously say happiest place earth try avoid crowd enjoyable day
13	Disneyland California	always dislike line always well worth hate problem ride shutting especially one ride time disappointing happened trip
14	Disneyland California	bank account hate body exhausted see smile kid face well worth
15	Disneyland California	nothing dislike magical place earth love disneyland

Query type: Dice

Query:

```
SELECT d.quarter, rt.review_text
FROM review_fact AS rf, review_text_dimension AS rt, date_dimension AS d,
branch_dimension AS b
WHERE rf.date_key = d.date_key AND rf.review_text_key = rt.review_text_key
AND rf.branch_key = b.branch_key AND b.branch_key = 6
```

What does this query do?: Retrieve all review texts and the quarter the review texts were written in as long as they are reviews written for the branch with branch_key = 6 (the branch is Universal Studios Singapore).

This is a dice query because it is creating a subset of the data by selecting specific values from more than one dimension (in this case, the review_text dimension and the date dimension – we are also selecting values that correspond to a particular branch from the branch dimension). We are looking only at the subset of the data containing reviews_texts and the quarter they were written in for the branch with branch_key = 6.

Query output (first 15 rows):

	quarter integer	review_text text
1	1	loved way wish main roller coaster running though strongly recommended first timer thumb
2	1	make sure prepared stay whole day worth take hat poncho drink bottle thing expensive
3	3	child wanted see universal studio okay child much fun plenty stuff
4	3	much fun traveled two kid aged 5 8 great time line long place definitely cleared afternoon
5	2	even think going weekend crowded every ride waiting time 1 2 hour
6	4	average dated setting nothing great maybe kid like young people used much adrenaline
7	3	third trip universal studio always great place highly recommended anyone
8	3	great place despite expensive price ticket fare reach venue even first timer traveller quite nice see
9	2	would much better two red blue roller coaster ride working ride seem go quick ideal waiting line 30 minute
10	4	ultimate place earth ride ambience high quality right start end everything well managed organised
11	3	great kid long ques highlight 1 transformer 2 mummy 3 scifi 4 jurassic world 5 waterworld show 6 steven spielberg show
12	2	experience wow coming across castle feel really nice ride fun must visit people visit sentosa island
13	1	loved universal studio character street lot space crowded gain access quickly onto ride clean professional loved
14	3	one best entertainment park world wide range activity enjoy clean imaginative environment fun show ride date newest technology trend
15	3	went one day organized clean however crowded wait long time line use different attraction next time go sure buy seasonal pas express access believe worth

Query type: Composite Query

Query:

```
SELECT b.branch_key, d.year, AVG(rf.monthly_percent_positive_reviews) AS  
avg_monthly_percent_positive_reviews  
FROM review_fact AS rf, date_dimension AS d, branch_dimension AS b  
WHERE rf.date_key = d.date_key AND rf.branch_key = b.branch_key  
AND b.branch_key = 1 AND (d.year = 2015 OR d.year = 2016)  
GROUP BY ROLLUP (b.branch_key, d.year)  
ORDER BY b.branch_key
```

What does this query do?: Compare the average monthly percentage of positive reviews for Disneyland California (the branch with branch_key = 1) for the years of 2015 and 2016.

This is a composite query because it is both dicing and rolling up. The query dices the data by selecting the subset of the data where the branch_key = 1 and the year = 2015 or 2016 (it selects specific values from more than one dimension: the branch and date dimensions). The query rolls up to generate averaged subtotals for all possible combinations of branch_key and year. We are summarizing monthly totals into yearly totals.

Query output:

	branch_key integer	year integer	avg_monthly_percent_positive_reviews numeric
1	[null]	[null]	84.9809571121046520
2	1	2015	85.1806112997838828
3	1	2016	84.7499999999999990
4	1	[null]	84.9809571121046520

Query type: Composite Query

Query:

```
SELECT b.branch_key, d.quarter, AVG(rf.monthly_percent_negative_reviews) AS  
avg_monthly_percent_negative_reviews  
FROM review_fact AS rf, date_dimension AS d, branch_dimension AS b  
WHERE rf.date_key = d.date_key AND rf.branch_key = b.branch_key
```

```
AND (b.branch_key = 2 OR b.branch_key = 3) AND d.quarter = 2
GROUP BY ROLLUP (b.branch_key, d.quarter)
ORDER BY b.branch_key;
```

What does this query do?: Compare the average monthly percentage of negative reviews for Disneyland Hong Kong (the branch with branch_key = 2) and Disneyland Paris (the branch with branch_key = 3) for quarter 2.

This is a composite query because it is both dicing and rolling up. The query dices the data by selecting the subset of the data where the branch_key = 2 or 3 and the quarter = 2 (it selects specific values from more than one dimension: the branch and date dimensions). The query rolls up to generate averaged subtotals for all possible combinations of branch_key and quarter. We are summarizing monthly totals into quarterly totals.

Query output:

	branch_key integer	quarter integer	avg_monthly_percent_negative_reviews numeric
1	2	2	3.9566847147022073
2	2	[null]	3.9566847147022073
3	3	2	14.3999264016227546
4	3	[null]	14.3999264016227546
5	[null]	[null]	10.0790194069137706

Part A.2

Query type: Iceberg



Query:

```
SELECT b.branch_name, COUNT(*) AS count_5_star_reviews
FROM review_fact AS rf, branch_dimension AS b, review_text_dimension AS rt
WHERE rf.branch_key = b.branch_key AND rf.review_text_key = rt.review_text_key
AND rt.rating = 5
GROUP BY b.branch_name
ORDER BY count_5_star_reviews DESC
LIMIT 5;
```

What does this query do?: Retrieve the top 5 branches in terms of how many 5 star reviews they have.

This is an iceberg query because we are getting the top 5 branches.

Query output:

	 branch_name character varying (255)	 count_5_star_reviews bigint
1	Universal Studios Florida	17958
2	Disneyland California	11797
3	Universal Studios Singapore	8137
4	Disneyland Paris	5763
5	Disneyland Hong Kong	4331

Query type: **Windowing**

Query:

```
SELECT
    b.branch_name,
    COUNT(*) AS count_1_star_reviews,
    brand_averages.avg_1_star_reviews,
    RANK() OVER (
        PARTITION BY
            CASE
                WHEN b.branch_name LIKE '%Universal Studios%' THEN 'Universal
Studios'
                WHEN b.branch_name LIKE '%Disneyland%' THEN 'Disneyland'
                ELSE 'Other'
            END
        ORDER BY COUNT(*) DESC
    ) AS rank
FROM review_fact AS rf
JOIN branch_dimension AS b ON rf.branch_key = b.branch_key
JOIN review_text_dimension AS rt ON rf.review_text_key = rt.review_text_key
JOIN (
    SELECT
        CASE
            WHEN b.branch_name LIKE '%Universal Studios%' THEN 'Universal Studios'
            WHEN b.branch_name LIKE '%Disneyland%' THEN 'Disneyland'
            ELSE 'Other'
        END AS brand,
        AVG(review_counts.count_1_star_reviews) AS avg_1_star_reviews
    FROM (
        SELECT
            b.branch_name,
            COUNT(*) AS count_1_star_reviews
        FROM review_fact AS rf
        JOIN branch_dimension AS b ON rf.branch_key = b.branch_key
        JOIN review_text_dimension AS rt ON rf.review_text_key = rt.review_text_key
        WHERE rt.rating = 1 AND (b.branch_name LIKE '%Disneyland%' OR
b.branch_name LIKE '%Universal Studios%')
        GROUP BY b.branch_name
    ) AS review_counts
```

```

JOIN branch_dimension AS b ON review_counts.branch_name LIKE '%' ||
b.branch_name || '%'
WHERE b.branch_name IN ('Disneyland Paris', 'Disneyland California', 'Disneyland
Hong Kong', 'Universal Studios Florida', 'Universal Studios Singapore', 'Universal
Studios Japan')
GROUP BY brand
) AS brand_averages ON
CASE
WHEN b.branch_name LIKE '%Universal Studios%' THEN 'Universal Studios'
WHEN b.branch_name LIKE '%Disneyland%' THEN 'Disneyland'
ELSE 'Other'
END = brand_averages.brand
WHERE rt.rating = 1 AND (b.branch_name LIKE '%Disneyland%' OR b.branch_name
LIKE '%Universal Studios%')
GROUP BY b.branch_name, brand_averages.avg_1_star_reviews
ORDER BY b.branch_name;

```

What does this query do?: Compare each branch's count of 1 star reviews with the average count of 1 star reviews for its brand ("Disneyland" or "Universal Studios") and provide a rank. We partition and rank based on the brand.

Query output:

	branch_name character varying (255)	count_1_star_reviews bigint	avg_1_star_reviews numeric	rank bigint
1	Disneyland California	450	446.0000000000000000	2
2	Disneyland Hong Kong	152	446.0000000000000000	3
3	Disneyland Paris	736	446.0000000000000000	1
4	Universal Studios Florida	1368	655.6666666666666667	1
5	Universal Studios Japan	194	655.6666666666666667	3
6	Universal Studios Singapore	405	655.6666666666666667	2



Query type: Window Clause

Query:

```
SELECT DISTINCT
  make_date(d.year, 1, 1) AS year,
  AVG(rt.rating) OVER W AS movavg
FROM review_fact AS rf
JOIN date_dimension AS d ON rf.date_key = d.date_key
JOIN review_text_dimension as rt ON rf.review_text_key = rt.review_text_key
JOIN branch_dimension as b ON rf.branch_key = b.branch_key
WHERE b.branch_key = 3
WINDOW W AS (
  PARTITION BY make_date(d.year, 1, 1)
  ORDER BY make_date(d.year, 1, 1)
  RANGE BETWEEN INTERVAL '1' YEAR PRECEDING
  AND INTERVAL '1' YEAR FOLLOWING
)
ORDER BY make_date(d.year, 1, 1);
```

What does this query do?: Calculate the moving average of ratings for the branch with branch_key = 3 (the branch is Disneyland Paris) over a rolling window of 3 years (1 year before, the current year and 1 year after the current year), and display the result for each year.

Query output:

	 year date	 movavg numeric
1	2010-01-01	3.8500000000000000
2	2011-01-01	3.8522167487684729
3	2012-01-01	3.9665399239543726
4	2013-01-01	3.8551495016611296
5	2014-01-01	4.0189834660134721
6	2015-01-01	4.0304990757855823
7	2016-01-01	3.9984639016897081
8	2017-01-01	4.0864055299539171
9	2018-01-01	3.9505754908598510
10	2019-01-01	3.7539062500000000