**Question 3**

Select father, mother and id. UNNEST the field children so that each child  
is in a separate row.

Provide your unnest query for question 3f and submit a screenshot of the results.

Your Answer:

WITH ch AS (

SELECT

father,mother,id,children

FROM `anusha-345017.Assignment3.households`)

SELECT father,mother,id,children

FROM ch, UNNEST (children) as children

Graphical user interface, text, application

Description automatically generated

**Question 4**

Create a new table called points with the following schema. In this case, create the table  
using SQL with a CREATE TABLE statement.  
a. Key is a field of type INT64.  
b. Names is an array of STRINGs.  
c. Complex is a struct with two subfields: real and imaginary. Each subfield is of  
type FLOAT64.Provide your CREATE TABLE query. Submit a screenshot of the table schema from the schema tab.

Your Answer:

CREATE TABLE anusha-345017.Assignment3.points

(

Key INT64,

Names STRING,

Complex

STRUCT<

real FLOAT64,

imaginary FLOAT64>)

Graphical user interface, text, application, email

Description automatically generated

missing ARRAY<> syntax for 'Names'

**Question 5**

What is the difference between required and nullable modes for a field? Why are  
required fields important in schema design? (For example, what could happen if a key  
field that identifies a row was created in nullable mode?)Provide your response to question 5.

Your Answer:

The difference between required and nullable modes for a field is that when required is used null values are not allowed and when nullable is used column allows NULL values which is default.If the mode is unspecified, the column defaults to NULLABLE.

Required field is important as they are  unique identifier for every row and the records can be accessed based on that and it cannot be null, more like primary key.

In our example, if the key was nullable and we would not be able to reference or extract the row based on its unique key id. hence its mandatory that the required column cannot be kept null.

**Question 6**

Explain the difference between the FLOAT and NUMERIC types. Why is the NUMERIC  
important for recording financial data? What could happen if you stored financial data  
with type FLOAT?Provide your response to Question 6.

Your Answer:

FLOAT and NUMERIC are used to stored decimal values.The real difference is when doing arithmetic operations (+,-,\*) or while doing comparison operations (>,<,+). Since, Float is an approximate datatype, there will be a leak and you may not find the exact values

We must use float datatype for storing non critical information like quantity, For all critical fields, where precision matters, we should use NUMERIC Datatype.

Approximate numeric data types (like FLOAT64) do not store the exact values specified for many numbers; they store an extremely close approximation of the value. For many applications, the tiny difference between the specified value and the stored approximation is not noticeable. At times, though, the difference becomes noticeable. Because of the approximate nature of the float and real data types, do not use these data types when exact numeric behavior is required, such as in financial applications, in operations involving rounding, or in equality checks.

**Question 7**

Modify the following SQL query so that it creates a table in your dataset.  
create or replace table sandbox.dateparse  
as  
(  
SELECT "04/22/1980" date  
UNION ALL SELECT "03/12/1991"  
UNION ALL SELECT "09/25/2006"  
)  
The table consists of dates in a column of type string. These dates are formatted as  
month/day/year. Use the PARSE\_DATE function to write a query that converts these dates to  
the DATE type.

Provide your PARSE\_DATE query for question 7 and a screenshot of the query results.

Your Answer:

Query:

with dt AS

(

SELECT "04/22/1980" date

UNION ALL SELECT "03/12/1991"

UNION ALL SELECT "09/25/2006"

)

SELECT PARSE\_DATE("%m/%d/%Y",date) as parsed\_column

FROM dt

Graphical user interface, text, application, email

Description automatically generated

**Question 8**

Use the following query to create a table.  
SELECT "IS 6850" as class\_name, 9 as section  
UNION ALL SELECT "Math 1060", 2  
UNION ALL SELECT "Math 1060", 8  
UNION ALL SELECT "IS 6850", 4  
UNION ALL SELECT "IS 6850", 5  
UNION ALL SELECT "Math 1030", 4  
UNION ALL SELECT "Math 1030", 2  
Use the ARRAY\_AGG function to create a result set with a) a row for each distinct class  
and b) an array of all sections for the class.Provide your ARRAY\_AGG query for question 8 and a screenshot of the query results.

Your Answer:

WITH

section AS

(SELECT "IS 6850" as class\_name, 9

UNION ALL SELECT "Math 1060", 2

UNION ALL SELECT "Math 1060", 8

UNION ALL SELECT "IS 6850", 4

UNION ALL SELECT "IS 6850", 5

UNION ALL SELECT "Math 1030", 4

UNION ALL SELECT "Math 1030", 2)

SELECT class\_name, ARRAY\_AGG(section) as array\_agg

FROM section

GROUP BY class\_name;

Graphical user interface, application, table

Description automatically generated

**Question 9**

You will frequently encounter malformed data as you are working on analytics problems.  
One option for handling bad fields is to set them to null. Another is to discard rows  
containing bad entries. We’ll go through a couple of examples using the SAFE\_CAST  
function. Create a table with this query:  
select "2017-01-01" date  
UNION ALL select "2013-04-05"  
UNION ALL select "2015-07-07"  
Use the CAST function to convert the data in the string field into DATE type.  
Now, create a table with this query.  
select "2017-01-01" date, 431 transaction\_id  
UNION ALL select "2013-04-05", 278  
UNION ALL select "2015-07-07", 448  
UNION ALL select "Robert", 542  
UNION ALL select "Street", 770  
a. What happens this time when you use the CAST function to attempt to convert  
the date field to type DATE?  
b. Use the SAFE\_CAST function to convert strings to dates where possible and  
return nulls for bad dates.  
c. Wrap the query from b. In a CTE. Select from the previous results and add a  
WHERE clause so that rows with null in the date column are discarded.Provide your answers to 9a-c. (Provide queries for b and c.)

Your Answer:

9 a.

Ans: When you use the CAST function to attempt to convert the date field to type DATE we get the following error that robert is an invalid date.

Graphical user interface, text, application

Description automatically generated

9. b

Query:

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, email

Description automatically generated

9 c.

Query:

with discard as

(

select

date,

SAFE\_CAST(TRIM(date) AS DATE) AS date\_casted

FROM

(

select

"2017-01-01" AS date,431 transaction\_id,

UNION ALL

(select

"2013-04-05" AS date,278)

UNION ALL

( select

"2015-07-07" AS date,448)

UNION ALL

( select

"Robert" AS date,542)

UNION ALL

(select

"Street" AS date,770)

)

)

select date\_casted from discard where date\_casted is not NULL

Graphical user interface, text, email

Description automatically generated

**Question 10**

Explain the difference between UNION and UNION ALL in SQL. (You may need to do  
some research online.) How is this related to the DISTINCT keyword?Provide your response to question 10.

Your Answer:

Union extracts the rows that are being specified in the query while Union All extracts all the rows including the duplicates (repeated values) from both the queries

Union all will not eliminate duplicate rows, instead it just pulls all rows from all tables fitting your query specifics and combines them into a table

Duplicate rows are removed from the union results and the optional distinct keyword has the same effect but it makes it explicit.

We can mix union ll and union distinct in the same query. Mixed union types are treated such that a Distinct union over overrides any ALL union to its left. A DISTINCT union can be produced explicitly by using union distinct or implicitly by using union with no following DISTINCT or ALL keyword .

This is a copy and paste from another text. We wouldn't your own response.

**Question 11**

More date parsing.  
We can parse the string 'July 24, 2019' as follows.

select PARSE\_DATE('%B %e, %Y ','July 24, 2019')  
This returns the DATE 2019-07-24.  
Look at the Parse Date Documentation as well as Supported Format Elements for Date.  
Write statements to parse each of the following strings into dates.  
a. ‘07/29/84’  
b. ‘4/21/2021’  
c. ‘2016-04-01’  
d. ‘04-Jun-84’  
e. ‘August 28th, 2019’

Provide your select statements for parts a-e.

Your Answer:

a.

SELECT PARSE\_DATE("%m/%d/%y", "07/29/84") AS parsed;

b.

SELECT PARSE\_DATE("%m/%d/%Y", "4/21/2021") AS parsed;

c.

SELECT PARSE\_DATE("%Y-%m-%d", "2016-04-01") AS parsed;

d.

SELECT PARSE\_DATE("%d-%b-%y", "04-Jun-84") AS parsed;

e.

SELECT PARSE\_DATE("%B %dth, %Y", "August 28th, 2019") AS parsed;

**Question 12**

Now, we’ll combine your work in the previous exercise with the SAFE keyword and the  
COALESCE function. We can combine multiple SAFE.PARSE\_DATE statements in  
sequence using COALESCE to account for different date formats - COALESCE returns  
the first value in the sequence that is not null.  
Use the following query to create a table.  
select '08/12/91' date  
UNION ALL SELECT '3/01/2018'  
UNION ALL SELECT '2019-03-04'  
UNION ALL SELECT '01-Apr-92'  
UNION ALL SELECT 'July 30th, 2001'  
Apply the SAFE prefix to each of the PARSE\_DATE statements you wrote in exercise 2.  
Combine these with COALESCE and query the table you just created. Verify that each  
input date string parses to a DATE type in the results. (You shouldn’t get back any null  
values.)Provide your query and a screenshot of the results. (You should use COALESCE and SAFE.PARSE\_DATE)

Your Answer:

create table `anusha-345017.Assignment3.colas`

AS

SELECT(COALESCE(SAFE.PARSE\_DATE("%m/%d/%y", '08/12/91'),

SAFE.PARSE\_DATE("%m/%d/%Y", '3/01/2018'),

SAFE.PARSE\_DATE("%Y-%m-%d", '2019-03-04'),

SAFE.PARSE\_DATE("%d/%b/%y", '01-Apr-92'),

SAFE.PARSE\_DATE("%B %dth, %Y", 'July 30th, 2001')

)) dt

Graphical user interface, application

Description automatically generated

**Question 13**

**2 / 2 pts**

Provide your query. Make sure to use COUNTIF and STARTS\_WITH

Your Answer:

with messages as

(

select 'Hello, I am learning SQL.' message

UNION ALL select '2019-04-08'

UNION ALL select 'Hello world!'

UNION ALL SELECT 'This is a prerecorded message.'

UNION ALL SELECT 'This call will be recorded for quality assurance.'

UNION ALL SELECT 'Hello Cleveland!'

UNION ALL SELECT 'Hello?'

UNION ALL SELECT '2642 West 360 South'

)

SELECT countif( STARTS\_WITH(message, "Hello")) from messages

Graphical user interface, text, application, email

Description automatically generated

**Question 14**

nstead of importing JSON directly into a schema, we can alternatively import JSON  
objects into STRING fields and then query embedded JSON using the JSON\_EXTRACT  
function. (See the JSON Extract Documentation.)  
Start with the following query.  
WITH json as  
(  
SELECT '{"name": "Gilbert", "wins": [["straight", "7♣"], ["one pair", "10♥"]]}' data  
UNION ALL SELECT '{"name": "Alexa", "wins": [["two pair", "4♠"], ["two pair", "9♠"]]}'  
UNION ALL SELECT '{"name": "May", "wins": []}'  
UNION ALL SELECT '{"name": "Deloise", "wins": [["three of a kind", "5♣"]]}'  
)  
SELECT JSON\_EXTRACT(data, '$.name') from json  
This query selects the name field from the JSON, i.e., it returns Gilbert, Alexa, May and  
Deloise.  
a. Modify the query to return the wins field.  
b. Modify the query to return the first win from each array of wins. (This will return  
null in one case.)  
c. Modify the query to return the second element of each win selected in b., i.e., 7♣,  
4♠ and 5♣.Provide your queries and screenshots of the results for a-c.

Your Answer:

14 a.

Query

WITH json as  
(  
SELECT '{"name": "Gilbert", "wins": [["straight", "7♣"], ["one pair", "10♥"]]}' data  
UNION ALL SELECT '{"name": "Alexa", "wins": [["two pair", "4♠"], ["two pair", "9♠"]]}'  
UNION ALL SELECT '{"name": "May", "wins": []}'  
UNION ALL SELECT '{"name": "Deloise", "wins": [["three of a kind", "5♣"]]}'  
)  
SELECT JSON\_EXTRACT(data, '$.wins') from json

Graphical user interface, text, application, email

Description automatically generated

14 b.

WITH json as

(

SELECT '{"name": "Gilbert", "wins": [["straight", "7♣"], ["one pair", "10♥"]]}' data

UNION ALL SELECT '{"name": "Alexa", "wins": [["two pair", "4♠"], ["two pair", "9♠"]]}'

UNION ALL SELECT '{"name": "May", "wins": []}'

UNION ALL SELECT '{"name": "Deloise", "wins": [["three of a kind", "5♣"]]}'

)

SELECT JSON\_EXTRACT(data, '$.wins[0]') from json

Graphical user interface, text, application, email

Description automatically generated

14 c.

WITH json as

(

SELECT '{"name": "Gilbert", "wins": [["straight", "7♣"], ["one pair", "10♥"]]}' data

UNION ALL SELECT '{"name": "Alexa", "wins": [["two pair", "4♠"], ["two pair", "9♠"]]}'

UNION ALL SELECT '{"name": "May", "wins": []}'

UNION ALL SELECT '{"name": "Deloise", "wins": [["three of a kind", "5♣"]]}'

)

SELECT JSON\_EXTRACT(data, '$.wins[0][1]') from json

Graphical user interface, text, application, email

Description automatically generated