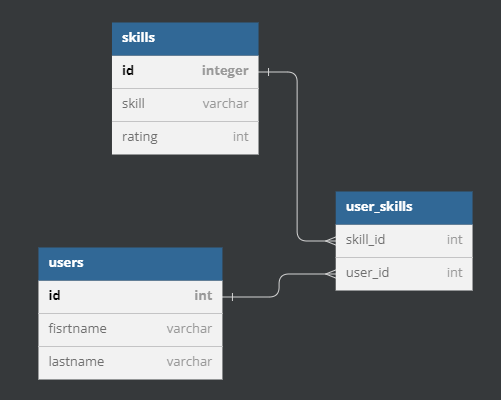
**Nombre:** Fernando Vásquez Choquehuanca

Data Engineer Part 1:

**DB model:**



**DB Create Script (PostgreSQL):**

CREATE TABLE `users` (

`id` integer PRIMARY KEY,

`fisrtname` varchar(255),

`lastname` varchar(255),

);

CREATE TABLE `skills` (

`id` integer PRIMARY KEY,

`skill ` varchar(255)

"rating" int

);

CREATE TABLE `user\_skills` (

`skill\_id` int,

`user\_id` int

);

ALTER TABLE `user\_skills` ADD FOREIGN KEY (`user\_id`) REFERENCES `users` (`id`);

ALTER TABLE `user\_skills` ADD FOREIGN KEY (`skill\_id`) REFERENCES `skills` (`id`);

**Query:**

1. select id from (
2. select u.id, count(skill\_id) total\_top\_skill from users u
3. join user\_skills us on us.user\_id = u.id
4. where skill\_id in (select id from skills where rating <= 10)
5. group by u.id) as t1
6. where total\_top\_skill >= 5;

**Explanation**:

Assuming rating field in skills helps define the best skills. The most important filter is in line 4, which filters only the users with top 10 skills. Then data is aggregated by user in line 5, so we can count skills. Finally, in line 6, we filter the user that only has more than 5 skill.

The lines 2 and 3 are just getting whole data needed from the different tables involved, using the id as foreign key.

**Performance:**

Join is critical point the query, could harm performance when not having filters. Skill\_id filter is important as you get less rows and help join with less data.

It’s just important to have index in proper fields, like id fields in every table, that will help to make the join fast.

Indexes should be in user.id, skills.id, user\_skills.user\_id and user\_skills.skill\_id. Because, joins work on those fields.

**Problem 2**

SELECT \*

FROM work\_experience

ORDER BY id

LIMIT 1, 1;

I am using id instead of date\_added, because of date\_added seems to be date when the row was added (audit field) which might not be the real second experience. Also, id seems not to be the best as it is just an ID, however in the question table, it takes the ID as reference.

**Explanation**:

Query is ordering by id, as I have said, I am assuming it’s the order in which the person registered their experience. In the case, id is not the correct field, we as developer should ask to the client or data provider, which field is most proper to use. We as developer don’t define that, and is a must to ask, and not assume.

**Performance**

Performance is good enough, as we are using LIMIT clause, and assuming id (or date\_added in case we use it for job order) is indexed, because we are using ORDER BY on this field.

PySpark

from pyspark.sql import SparkSession

from pyspark.sql import SQLContext

from pyspark import SparkContext

from pyspark.sql.types import StructType, StructField, StringType, IntegerType

from pyspark.sql import functions as f

spark = SparkSession.builder.enableHiveSupport().getOrCreate()

#Defining Schema

csv\_schema = StructType([

        StructField("product\_id", StringType(), False),

        StructField("user\_id", StringType(), True),

        StructField("user\_name", StringType(), True),

        StructField("rating", StringType(), True),

        StructField("review\_date", StringType(), True),

        StructField("review\_title", StringType(), True),

        StructField("review\_text", StringType(), True)])

# read csv from s3 using pyspark

s3\_path = "s3://aws-a0122-use1-00-d-s3b-bing-rdm-input01/Latam/Inbound/Test\_JK/dummy\_d.csv"

df = spark.read.option("badRecords", "false").csv(s3\_path, header=True, schema=csv\_schema)

#Dropping Rows where more than 90% of the values are null

threshold = 6 # 90% of 7 columns

df = df.filter((sum(f.isnull(f.col(c)).cast("integer") for c in df.columns) < threshold))

#Removing Special Characters

df=df.withColumn("user\_name",f.regexp\_replace("user\_name","[^0-9a-zA-Z\_\s]+", ""))

df=df.withColumn("review\_title",f.regexp\_replace("review\_title","[^0-9a-zA-Z\_\s]+", ""))

df=df.withColumn("review\_text",f.regexp\_replace("review\_text","[^0-9a-zA-Z\_\s]+", ""))

#Adding Primary Key

df=df.withColumn("uuid",f.concat\_ws("","product\_id","user\_id","user\_name"))

# Convert to Date Format

df = df.withColumn('review\_date', f.to\_date(f.col('review\_date'), 'dd-MM-yyyy'))

# Create temp table

table\_name='produdcts\_reviews'

df.createOrReplaceTempView(table\_name)

# Average Rating of Each Product

query\_1 = spark.sql(f"select product\_id,avg(rating) as average\_rating from {table\_name} group by 1")

#Top 10 products with highest average rating

query\_2 = spark.sql(f"select product\_id,avg(rating) as average\_rating from {table\_name} group by 1 order by 2 desc limit 10")

#Top 3 users with most reviews

query\_3 = spark.sql(f"select user\_id,count(1) as ttl\_reviews from {table\_name} group by user\_id having count(distinct review\_title)>=1 order by ttl\_reviews desc limit 3")

#print(query\_3)