# Оглавление

# Введение

# 1. Системные характеристики ноутбуков.

## 1.1 На начальном этапе работы.

|  |  |
| --- | --- |
| Тип машины | Notebook |
|  |  |
| Операционная система | Linux Mint 20.3 |
|  |  |
| Процессор | AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx. (1 физический процессор, 4 ядра, 8 потоков) |
|  |  |
| Оперативная память | 6081 Мб (6081832 КиБ) |

# 1.2 Основной ноутбук.

|  |  |
| --- | --- |
| Тип машины | Notebook |
|  |  |
| Операционная система | Майкрософт Windows 11 Pro |
|  |  |
| Версия | 22H2 |
|  |  |
| Процессор | 11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz 3.11 GHz |
|  |  |
| Оперативная память | 32,0 ГБ (доступно: 31,8 ГБ) |
|  |  |
| Локальный диск | 953, 86 ГБ |

# 2. Первоначальный анализ данных.

Для того, чтобы данные, хранящиеся csv файлах изучить для дальнейшей загризки в MySQL, необходимо их исследовать:

* Узнать максимальное количество символов в одной ячейке данных, которое может быть в заданном столбце.
* Посмотреть изначальный тип данных использованный в каждом из столбцов таблиц.
* Вывести на консоль ячейку с максимальным количеством символов по заданному индексу для проверки работы кода или для ознакомления с хранящейся в этой ячейке информацией.
* Получить данные из нескольких ячеек столбца, выбранных случайным образом, для ознакомления с примерным содержимым таблицы.
* Посчитать количество всех возможных значений столбца.
* Вывести все заголовки столбцов.
* Вывести заданное количество строк таблицы.
* Узнать общее количество строк csv файла.

Следует учитывать, что данная база создается исключительно для анализа и визуализации имеющихся данных и не предполагает дальнейшей с ней работы как с действующей базой данных.

При работе с достаточно большими файлами, а в нашем случае речь идет, например, о файлах размером 20.1 GiB (vacansies) или 8.2 GiB (curricula\_vitae), могут быть сложности из-за ограничений в объеме оперативной памяти рабочего компьютера или ноутбука.

Для оптимизации работы с данными файлами можно использовать генераторы и делить файлы на чанки (chunks) для последующей работы с файлом по частям и дальнейшего объединения данных, если это необходимо.

Генератор – это удобный механизм конструирования итерируемого объекта, похожий на обычную функцию. Если обычная функция возвращает единственное значение, то генератор может возвращать последова-

тельность значений, приостанавливаться и работать дальше в ожидании запроса следующего. При создании генератора вместо return используется yield. Поскольку генераторы отдают значения по одному, а не весь список сразу, программа потребляет меньше памяти.

Для последовательного чтения файлов небольшими порциями можно:

* задать размер порций для последовательной обработки каждой (chunks);
* задать размер одной порции для дальнейшей обработки (chunksize());
* ограничить вывод количества строк таблицы для ознакомления или обработки (nrows);
* использовать метод get\_chunk для чтения кусков произвольного размера.

Для контроля за временем в течении которого в ячейке среды разработки Jupyter Notebook выполняется нужный фрагмент кода, можно использовать ipython-autotime (%load\_ext autotime или %reload\_ext autotime).

Рассмотрим далее фрагменты кода, которые можно использовать для целей описанных выше.

%load\_ext autotime

import pandas as pd

file = 'file\_path'

def read\_csv(file\_name):

    for chunk in pd.read\_csv(file\_name, sep=';', chunksize=100\_000, low\_memory=False):

        yield chunk

def process\_dataframe(dataframe):

    return dataframe['column\_name'].tolist()

m = 0

for df in read\_csv(file):

    lst = process\_dataframe(df)

    x = max([len(str(lst[i]))for i in range(len(lst))])

    if x  m:

        print(x)

        m = x

print('---' \*  20)

print(m)

Данный фрагмент кода используется для нахождения максимальной длины строки в ячейке заданного столбца.

При обработке каждого из чанков данный фрагмент кода находит максимальное значение длины строки ячейки и, при условии, что данное значение больше предыдущего, выводит его на консоль.

Завершением работы кода является вывод максимального значения длины строки и времени, в течении которого выполнялся данный фрагмент.

Следующий фрагмент задает размер чанка для обработки и выводит из него значения указанного количества случайных ячеек, для того, чтобы ознакомиться с примерным содержанием конкретного столбца таблицы.

%reload\_ext autotimer

file = 'file\_path'

reader = pd.read\_csv(file, sep=';',  iterator=True, usecols=['column\_name'])

reader.get\_chunk(1\_000\_000)['column\_name'].sample(5)

В следующим фрагменте кода производится подсчет общего количества каждого из возможных значений столбца из csv таблицы. Так же следующий фрагмент может быть использован, чтобы выявить уникальные ли

значения в заданном столбце датафрейма или нет: можно ли считать данную колонку первичным ключом таблицы или нет. Если значения в ячейках не очень длинные по количеству символов и вариантов возможных значений ориентировочно не очень много (к примеру, все значения столбца это 0 или 1), размер чанка можно не указывать (get\_chunk()). Через объект TextParser код отработает и так.

В случае работы с данной базой данных, когда идентификаторы по размеру примерно в 36 символов и уникальных значений должно быть очень много, размер чанка (get\_chunk(50\_000)) необходимо указать, для того, чтобы не перегрузить оперативную память.

%reload\_ext autotime

file = 'file\_path'

reader = pd.read\_csv(file, sep=';', iterator=True, usecols=['column\_name’])

reader.get\_chunk(50\_000)['column\_name'].value\_counts()

Иногда в данной базе встречаются неожиданные данные. Для проверки правильно ли код находит размер самой длинной строки и для ее просмотра, можно вывести эту самую строку, указав найденную ранее длину.

Данный код сначала создает переменную, в которую сохраняет чанк, в котором находится ячейка с указанной длиной, после чего выходит из цикла и распечатывает содержимое найденной ячейки как элемент списка.

import pandas as pd

%reload\_ext autotime

file = 'file\_path'

def read\_csv(file\_name):

    for chunk in pd.read\_csv(file\_name, sep=';', chunksize=10\_000, low\_memory=False):

        yield chunk

def process\_dataframe(dataframe):

    return dataframe['metro\_station'].tolist()

for df in read\_csv(file):

    lst = process\_dataframe(df)

    x = max([len(str(lst[i])) for i in range(len(lst))])

    if x == value\_length:

        data = df['metro\_station']

        break

print([j for j in data if len(str(j)) == value\_length])

Следующий фрагмент выводит все названия колонок одним списком. Список нужен, например, для создания схемы базы данных. Количество строк ограничено (nrows=5) для того, чтобы избежать перезагрузки оперативной памяти, когда файл большой. Если .columns убрать, фрагмент распечатает указанное количество строк из csv файла.

import pandas as pd

file = 'file\_path'

pd.read\_csv(file, sep=';', nrows=5).columns

Вывод количества пустых (не имеющих значений) ячеек в указанном столбце. Размер чанка не задан, но, через объект TextParser работает и так.

%reload\_ext autotime

reader = pd.read\_csv('/media/marina/big/Диплом/diploma\_data/vacancies.csv', sep=';', iterator=True, usecols=['column\_name'])

reader.get\_chunk()['column\_name'].isnull().sum()

Находим общее количество строк в csv файле. После обработки чанка используется двухсекундное засыпание (sleep(2)).

%reload\_ext autotime

from time import sleep

import pandas as pd

file = 'file\_path'

import pandas as pd

def read\_csv(file\_name):

    for chunk in pd.read\_csv(file\_name, sep=';', chunksize=50\_000, low\_memory=False):

        yield chunk

def process\_dataframe(dataframe):

    return dataframe.shape[0]

result = 0

for df in read\_csv(file):

    result += process\_dataframe(df)

    sleep(2)

print(result)

Вывести на консоль строку по индексу.

**import** pandas as pd

filename **=** "file\_path"

df **=** pd.read\_csv(filename, sep**=**';', encoding**=**'utf-8')

show **=** [index\_number]

df.iloc[show]

1. # Посчитать одинаковые значения для поиска связей между таблицами
3. **import** pandas as pd
4. **%**reload\_ext autotime
6. **def** num\_shared\_meaning(df1, df2, col1, col2):
7. """
8. Count how many unique items exist in both
9. specified columns across two given Pandas DataFrames.
10. """
11. s1 **=** set(df1[col1].unique())
12. s2 **=** set(df2[col2].unique())
13. intersection **=** s1 & s2
14. **return** len(intersection)
16. df1 **=** pd.read\_csv("/Users/РС/Documents/diploma\_data/curricula\_vitae.csv", sep**=**';', encoding**=**'UTF-8', usecols**=**['industry\_code'])
18. df2 **=** pd.read\_csv("/Users/РС/Documents/diploma\_data/vacancies.csv", sep**=**';', encoding**=**'UTF-8', usecols**=**['industry'])
20. num\_same\_cat **=** num\_shared\_meaning(df1, df2, 'industry\_code', 'industry')
21. print(f"There are {num\_same\_cat} categories found in both datasets.")

2. Загрузка в базу данных MySql.

В данной главе мы рассмотрим:

* вариант загрузки данных из CSV в базу MySQL;
* примеры кода для очистки данных;
* вариант использования Microsoft Excel для очистки данных;
* способ добавления столбца таблицы из CSV и имеющуюся таблицу MySQL.

В связи с большим разнообразием типов данных иногда и в данных одного столбца, данные изначально загружаем как VARCHAR или TEXT, ограничивая при необходимости размер ячейки для оптимизации БД.

Следующий код позволяет подключиться к базе данных и строку за строкой загружать данные из csv файла в созданную ранее таблицу.

Строки, которые не могут быть загружены в базу данных, перенаправляются в файл CSV для дальнейшего анализа и выявления причин препятствующих загрузке.

Здесь важно учитывать, что в исходной базе, хранящейся в CSV файлах, очень много двойных кавычек, поэтому, собирая строку для загрузки в MySQL, используются одинарные кавычки для строк, как типов данных, а не наоборот.

Я выбрала данный способ загрузки, так как он позволяет отследить какие конкретно строки нужно дополнительно обработать. Так же построчная загрузка помогает при работе с большими данными и не перегружает оперативную память.

import sys, time

import MySQLdb as mysql

host = '127.0.0.1'

user = 'root'

password = '\*\*\*\*\*'

db\_name = 'diploma'

from mysql.connector import IntegrityError, Error

file = "source\_file\_path"

file\_e = "erros\_file\_path"

try:

    conn = mysql.connect(host=host, database=db\_name, user=user, password=password)

    if conn.open:

        cursor = conn.cursor()

        cursor.execute("select database();")

        record = cursor.fetchone()

        print("You're connected to database: ", record)

       cursor.execute

        n = 0

        src\_file = open(file, mode="r", encoding="utf-8")

        err\_file = open(file\_e, mode="w", encoding="utf-8")

        run = True

        head = src\_file.readline()

        err\_file.writelines(head)

        while run:

            row1 = src\_file.readline()

            if row1 == '':

                run = False

                break;

            if not row1[:6] == '\*\*\*\*\*\*':

                row2 = row1.replace('\n', '')

                row2 = row2.split(';')

                s = "('" + "','".join(row2) + "');"

                sql = '''INSERT INTO `db\_name`.`table\_name` VALUES ''' + s

                try:

                    cursor.execute(sql)

                    n += 1

                    conn.commit()

                except:

                    err\_file.writelines(row1)

        try:

            conn.commit()

        finally:

            print('fin')

        src\_file.close()

        err\_file.close()

except Error as e:

    print("Error while connecting to MySQL", e)

    conn.close()

Здесь важно подчеркнуть, что данный код создает файл с ошибками, сохраняя первую строку в качестве строки с заголовками столбцов. Это позволяет использовать данный файл для дальнейшей успешной работы через библиотеку Pandas, конкретно: для успешного распознавания библиотекой строк и разделителей столбцов.

Ниже показан пример, как можно дополнить код, чтобы на консоли конкретизировать в чем именно была проблема с загрузкой конкретной строки. На консоли будет видно какая строка не загрузилась и в чем предположительно заключается ошибка. Используется это код только один раз на файле, в котором все ошибки. По первой же строке будет понятно, что нужно исправить для дальнейшей загрузки. Далее данный файл после исправлений нужно будет заново отправить в MySQL, создав следующий файл с ошибками.

**import** MySQLdb as mysql

host **=** '127.0.0.1'

user **=** 'root'

password **=** '\*\*\*\*\*\*\*'

db\_name **=** 'diploma'

**from** mysql.connector **import** IntegrityError, Error

file = "source\_file\_path"

file\_e = "erros\_file\_path"

**try**:

    conn **=** mysql.connect(host**=**host, database**=**db\_name, user**=**user, password**=**password)

**if** conn.open:

        cursor **=** conn.cursor()

        cursor.execute("select database();")

        record **=** cursor.fetchone()

        print("You're connected to database: ", record)

       cursor.execute

        # loop through the data frame

        n **=** 0

        src\_file **=** open(file, mode**=**"r", encoding**=**"utf-8")

        err\_file **=** open(file\_e, mode**=**"w", encoding**=**"utf-8")

        run **=** True

        head **=** src\_file.readline()

**while** run:

            row1 **=** src\_file.readline()

**if** row1 **==** '':

                run **=** False

**break**;

**if** **not** row1[:5] **==** 'date\_':

                row2 **=** row1.replace('\n', '')

                row2 **=** row2.split(';')

                s **=** "('" **+** "','".join(row2) **+** "');"

                sql **=** '''INSERT INTO `db\_name`.`table\_name` VALUES ''' + s

**try**:

                    print(sql)

                    cursor.execute(sql)

                    n **+=** 1

                    conn.commit()

**except** Error as e1:

                    print(e1.msg)

                    err\_file.writelines(row1)

                # print(sql)

**try**:

            conn.commit()

**finally**:

            print('final')

        src\_file.close()

        err\_file.close()

**except** Error as e:

    print("Error while connecting to MySQL", e)

    conn.close()

Поскольку исследование возможных ошибок загрузки происходит на примере конкретной базы данных, все возможные подобные ошибки можно перечислить:

* Наличие точек с запятыми, одинарных кавычек, введенных пользователями и сохранившихся в имеющихся CSV файлах.
* Большое количество эмодзи разных видов создающих ошибки связанные с распознаванием кодировки.
* Наличие символов обратной косой черты «обратный слэш» непосредственно в конце строки перед разделителем точкой с запятой. Происходит экранирование и точка с запятой не является для программы разделителем.
* Кодировка, отличная от UTF-8
* “\n” в текстовых столбцах.

Первым делом, получив файл с ошибками, можно заменить символы, которые могут мешать распознаванию строки. Следующий код позволяет также обработать все еще большие файлы, так как обработка построчная.

**import** csv

**%**reload\_ext autotime

n **=** 0

src\_file **=** open(source\_file\_path, mode**=**"r", encoding**=**"utf-8")

file\_ch **=** open(file\_path\_for\_change, mode**=**'a', encoding**=**'utf-8')

run **=** True

head **=** src\_file.readline()

file\_ch.writelines(head)

**while** run:

    row1 **=** src\_file.readline()

**if** row1 **==** '':

        run **=** False

**break**

**if** **not** row1[:5] **==** '\*\*\*\*\*':

        row2 **=** row1.replace('\n', '')

­­        row2 **=** row2.replace("'", "`")

        row2 **=** row2.split(';')

        s **=** ';'.join(row2) + '\n'

        file\_ch.writelines(s)

        n **+=** 1

src\_file.close()

file\_ch.close()

print(f'{n} строк')

print('fin')

Таким образом мы заменили одинарные кавычки на обратные кавычки.

Для замены экранированных точек с запятой, которые могут встречаться в содержимом базы и мешать распознаванию столбцов, можно открыть CSV в приложении «Блокнот» и заменить ’\;’ на ‘;’.

После данной замены еще часть данных загружается в MySQL, но файл может быть все еще достаточно большой для дальнейшей работы. Поэтому разбить его на несколько других поменьше можно используя библиотеку Pandas. В следующем коде можно задать размер чанка и каждый чанк будет записан в новый файл.

**%**reload\_ext autotime

**import** pandas as pd

file **=** source\_file\_path

**def** read\_csv(file\_name):

**for** chunk **in** pd.read\_csv(file, sep**=**';', chunksize**=**21\_240, low\_memory**=**False, encoding**=**'utf-8'):

**yield** chunk

**def** process\_dataframe(dataframe):

**return** dataframe

n **=** 0

**for** df **in** read\_csv(file):

    chunk **=** process\_dataframe(df)

**if** n **==** 0:

        chunk.to\_csv(first\_file\_path, index**=** False, sep**=**';', encoding**=**'utf-8')

        n **+=** 1

**elif** n **==** 1:

        chunk.to\_csv(second\_file\_path, index**=** False, sep**=**';', encoding**=**'utf-8')

        n **+=** 1

**elif** n **==** 2:

        chunk.to\_csv(third\_file\_path, index**=** False, sep**=**';', encoding**=**'utf-8')

        n **+=** 1

**elif** n **==** 3:

        chunk.to\_csv(fourth\_file\_path, index**=** False, sep**=**';', encoding**=**'utf-8')

        n **+=** 1

**elif** n **==** 4:

        chunk.to\_csv(fifth\_file\_path, index**=** False, sep**=**';', encoding**=**'utf-8')

print('fin')

Таким образом файл размером в 84\_948 строк будет разбит на 4 файла по 21\_237 строк. Пятый файл на всякий случай, если строки неправильно посчитаны. В данном случае он остается пустым.

Код завершает свою работу, когда обработается последняя порция данных, запущенная генератором.

После загрузки файла с исправленными символами в MySQL, часть данных опять попадает в отдельный файл. Теперь можно воспользоваться возможностями Excel по замене точек с запятыми. Для того, чтобы Excel корректно прочитал CSV файл, надо его импортировать. Во вкладке «Данные» нажать на кнопку «Из текстового/CSV-файла», далее выбрать файл, кодировку и разделитель – точку с запятой. Excel правильно распределяет данные по колонкам с заголовками и можно воспользоваться инструментом по замене всех точек с запятой на запятые. Вкладка «Главная» - «Найти и выделить» - «Заменить…»

Даты нужно сохранить в правильном формате, чтобы потом не исправлять их в MySQL. Выделить столбец с датой, зайти в «Формат ячеек» и выбрать дату в правильном формате: например, 2012-12-10.

Далее файл надо сохранить как CSV. По умолчанию Excel предлагает сохранить CSV файл с разделителем запятой. Для того, чтобы на самом деле разделителем стала точка с запятой, нужно поменять региональные настройки Windows: пуск – панель управления – изменение форматов даты, времени и чисел – вкладка «форматы» - кнопка «дополнительные параметры» - разделитель элементов (точка с запятой). Даже, если при сохранении написано, что разделитель запятая, при правильных настройках файл сохранится с нужным нам разделителем. openOffice Calc может быть альтернативой Excel, до определенного размера файла. Он еще проще в использовании. Можно открыть CSV файл, заменить необходимые символы и сохранить изменения.

Теперь еще одна порция данных импортируется в MySQL и остается удалить emoji, которых довольно много в базе и их сложно распознать. Их наличие или отсутствие для дальнейшего анализа данных не критично.

Функция для удаления emoji:

**def** remove\_emoji(inputString):

    emoji\_pattern **=** re.compile("["

    u"\U0001F600-\U0001F64F" # emoticons

    u"\U0001F300-\U0001F5FF" # symbols & pictographs

    u"\U0001F680-\U0001F6FF" # transport & map symbols

    u"\U00002702-\U000027B0"

    u"\U000024C2-\U0001F251"

    u"\U0001f926-\U0001f937"

    u'\U00010000-\U0010ffff'

    u"\u200d"

    u"\u2640-\u2642"

    u"\u2600-\u2B55"

    u"\u23cf"

    u"\u23e9"

    u"\u231a"

    u"\u3030"

    u"\ufe0f"

    u"\u2069"

    u"\u2066"

    u"\u200c"

    u"\u2068"

    u"\u2067"

    "u\u1f929"

    "u\u1f9e9"

    "u\u23f0"

    "]+", flags**=**re.UNICODE)

**return** emoji\_pattern.sub(r'', inputString)

Основной код:

1. Начало формы

**import** csv

**%**reload\_ext autotime

**import** re

**import** sys

n **=** 0

src\_file **=** open(source\_file\_path, mode**=**"r", encoding**=**"utf-8")

file\_ch **=** open(file\_path\_for\_changing, mode**=**'a', encoding**=**'utf-8')

run **=** True

head **=** src\_file.readline()

file\_ch.writelines(head)

**while** run:

    row1 **=** src\_file.readline()

**if** row1 **==** '':

        run **=** False

**break**

    #print(row1)

**if** **not** row1[:5] **==** '\*\*\*\*\*':

        row2 **=** row1.replace('\n', '')

        row2 **=** remove\_emoji(row2)

        row2 **=** row2.split(';')

        s **=** ';'.join(row2) + '\n'

        file\_ch.writelines(s)

        n **+=** 1

src\_file.close()

file\_ch.close()

print(f'{n} строк')

print('fin')

Удаление дубликатов:

CREATE TABLE `table\_tmp` SELECT DISTINCT \* FROM `curricula\_vitae`;

DROP TABLE `curricula\_vitae`;

RENAME TABLE `table\_tmp` TO `curricula\_vitae`; -- (448 дубликатов в workexp)

2.1 Изменение типов данных.

Для того, чтобы установить тип данных year в таблице addedu для столбца graduate\_year, пустые ячейки необходимо заменить на значение Null.

**UPDATE** addedu **SET** graduate\_year = Null **WHERE** graduate\_year = '';

После этого можно поменять тип данных.

**ALTER** **TABLE** addedu

**MODIFY** **COLUMN** graduate\_year **YEAR**;

MySQL Server поддерживает однобайтный тип данных **YEAR**, предназначенный для представления значений года в формате YYYY. Диапазон возможных значений - от 1901 до 2155. В одной из ячеек указан год 1900 и пометка в столбце grad\_year\_mistake, что дата указана с ошибкой. Замена значения 1900 на **Null** уберет ошибку, мешающую поменять тип данных столбца на тип данных **YEAR**.

Следующий запрос выведет структуру таблицы, где можно посмотреть типы данных.

**DESC** addedu;

Удаляем некорректные данные.

* **UPDATE** addedu **SET** graduate\_year = NULL **WHERE** grad\_year\_mistake = 1;

Не полные даты. Решаем в зависимости от заказчика. Поскольку заказчик я, 0 заменяю на 01. Учитывая, что значение года может пригодиться. Таких данные укладываются примерно в одну тысячную процента.

Проверяем количество некорректно введенных дат, в которых вместо дня месяца или самого месяца указан 0.

**SELECT** count(\*) **FROM** curricula\_vitae

**WHERE**  DAYOFMONTH(date\_inactivation) = '0' OR MONTH(date\_inactivation) = '0';

Убрала пробелы и лишние символы для типа данных **SET**.

**UPDATE** curricula\_vitae

**SET** drive\_licences  = REPLACE(drive\_licences, ' ', '');

Убрала ошибочные данные.

**UPDATE** edu **SET** graduate\_year = NULL **WHERE** grad\_year\_mistake = 1;

2.2 Primary Key

В таблице organizations всего 914287 строк.

**SELECT** id\_organization, COUNT(id\_organization)

**FROM** organizations

**GROUP** **BY** id\_organization

**HAVING** COUNT(id\_organization) > 1;

Данный запрос выведет 3 значения столбца id\_organizations, которые не являются уникальными и мешают установить параметр PRIMARY KEY для данной колонки.

**SELECT** **DISTINCT** count(\*) **FROM** organizations;

Данный запрос выводит количество уникальных строк. В данном случае это количество совпадает с количеством строк в таблице. Таким образом не уникальны только сами ключи. Их можно заменить вручную, так как их всего 3.

Выводим дубликаты и смотрим какие два столбца можно использовать в качестве идентификаторов.

**SELECT** \* **FROM** organizations **WHERE** id\_organization = '1020202219091';

Меняем один из дубликатов id\_organization, чтобы сделать его уникальным и проверяем через предыдущий запрос получилось или нет.

**UPDATE** organizations **SET** id\_organization = '1020202219092'

**WHERE** id\_organization = '1020202219091' AND address\_code = '0200000000000';

И можено поменять тип данных.

**ALTER** **TABLE** organizations

**MODIFY** id\_organization **VARCHAR**(40) **PRIMARY** **KEY**;

Когда нет уверенности, что запрос отработает как положено, можно создать временную таблицу и, в случае успешного изменения данных, вернуть данные из временной таблицы в основную.

**CREATE** **TEMPORARY** **TABLE** tmp\_tab **AS** **SELECT** **DISTINCT** \* **FROM** your\_table;

**DELETE** **FROM** your\_table;

**INSERT** **INTO** your\_table **SELECT** \* **FROM** tmp\_tab;

**DROP** **TABLE** tmp\_tab;

В столбце birthday таблицы curricula\_vitae много данных, которые нам не могут понадобиться из-за того, что они либо ошибочные, что мы можем увидеть по столбцу birthday\_mistake, либо не могут быть верными, т.к. не относятся к нашему столетию и не находятся в промежутке от 1900 до 2155. Именно данные из этого промежутка может содержать тип данных YEAR.

Такие данные легко повредить. Восстанавливать таблицу из CSV очень долго. Восстанавливать столбец по идентификатору представляет проблемы. Простой способ это создать как раз временную таблицу TEMPORARY TABLE, проделать в ней все необходимые изменения и отправить данные обратно в curricula\_vitae. Это достаточно быстрый процесс. Создание временной таблицы занимает примерно 10 минут. Удаление данных из curricula\_vitae через DELETE – примерно 25 минут. Из CSV таблица размером с curricula\_vitae грузится порядка полутора суток.

Первое, что мы делаем во временной таблице со столбцом birthday – заменяем пустые ячейки на NULL и заменяем на NULL ячейки в строках которых столбец birthday\_data = ‘1’. Таким образом заменяем 2 779 250 ячеек из всего 10 576 544 строк таблицы curricula\_vitae.

Затем меняем тип данных на INT, чтобы убрать лишние символы и работать потом с длиной строки. Далее делаем следующий запрос:

**UPDATE** tmp\_tab

**SET** birthday = NULL

**WHERE** birthday < 1901

   OR birthday > 2155

   OR LENGTH(TRIM(birthday)) < 4

   OR LENGTH(TRIM(birthday)) > 4;

Убираем несуществующие года: года, которые не входят в требуемый промежуток и длина которых меньше или больше, чем 4-хзначное значение года. Функция TRIM() убирает лишние пробелы, если они есть. Смотрим получившиеся данные, которые не NULL – их 7 782 668 (со всеми ошибками изначально таких данных было 7 797 294, удалено 14 626 ячеек) и меняем тип данных столбца на тип данных year. Примерно 74% данных от общего количество ячеек с NULL с которыми можно работать.

Далее можно удалить данные из curricula\_vitae, вставить из TEMPORARY TABLE и TEMPORARY TABLE удалить.

-- value\_counts

**SELECT** nark\_inspection\_status **AS** `Значения`, COUNT(nark\_inspection\_status) **AS** `Количество`

**FROM** curricula\_vitae **WHERE** nark\_inspection\_status **IS** NOT NULL

**GROUP** **BY** nark\_inspection\_status;

2.3 Изменение формата даты

1. -- создаем столбец после столбца
2. **ALTER** **TABLE** vacancies **ADD** **COLUMN** date\_change\_inner\_info\_2 **DATE** **AFTER** date\_change\_inner\_info;
3. -- удаляем столбец
4. **ALTER** **TABLE** vacancies **DROP** **COLUMN** date\_creation;
5. -- меняем название столбца
6. **ALTER** **TABLE** vacancies CHANGE date\_creation\_2 date\_creation **DATE**;
8. -- меняем все даты
9. **UPDATE** vacancies
10. **SET** date\_change\_inner\_info\_2 = (**SELECT** CONVERT(date\_change\_inner\_info, **DATE**));
12. -- исправляем ошибочные
13. **UPDATE** vacancies
14. **SET** date\_change\_inner\_info\_2 = STR\_TO\_DATE(date\_change\_inner\_info, '%d.%m.%Y')
15. **WHERE** date\_change\_inner\_info LIKE '\_\_.\_\_.\_\_\_\_';
17. -- value\_counts
18. **SELECT** date\_change\_inner\_info **AS** `Значения`,
19. COUNT(date\_change\_inner\_info) **AS** `Количество`
20. **FROM** vacancies **WHERE** date\_change\_inner\_info **IS** NOT NULL
21. **GROUP** **BY** date\_change\_inner\_info;
22. -- HAVING length(active) < 10;
24. -- заменяем значения ячеек
25. **UPDATE** vacancies
26. **SET** date\_change\_inner\_info = NULL
27. **WHERE** date\_change\_inner\_info = '';

2.4 Дубликаты строк в таблицах.

1. -- убираем строки-дубликаты
2. **CREATE** **TABLE** `table\_tmp` **SELECT** **DISTINCT** \* **FROM** `vacancies`;
3. -- DROP TABLE `curricula\_vitae`;
4. RENAME **TABLE** `table\_tmp` **TO** `curricula\_vitae`;
5. **SELECT** **DISTINCT** COUNT(\*) **FROM** workexp;
6. -- проверить значение в обоих столбцах
7. **SELECT** date\_change\_inner\_info, date\_change\_inner\_info\_2
8. **FROM** vacancies
9. **GROUP** **BY** date\_change\_inner\_info, date\_change\_inner\_info\_2;

2.5 Создание индекса

Когда дело доходит до оптимизации производительности баз данных MySQL, создание индексов — один из лучших способов. Индексы SQL помогают повысить производительность базы данных, обеспечивая быстрый доступ к данным, хранящимся в базе данных.

2.6 Работа с датами

-- создаем столбец после столбца

ALTER TABLE vacancies ADD COLUMN date\_inactivation\_2 DATE AFTER date\_inactivation;

-- удаляем столбец

ALTER TABLE vacancies DROP COLUMN date\_change\_inner\_info;

-- меняем название столбца

ALTER TABLE vacancies CHANGE date\_change\_inner\_info\_2 date\_change\_inner\_info DATE;

-- конвертируем все даты в созданный столбец

UPDATE vacancies

SET date\_change\_inner\_info\_2 = (SELECT CONVERT(date\_change\_inner\_info, DATE));

-- исправляем ошибочные

UPDATE vacancies

SET date\_change\_inner\_info\_2 = STR\_TO\_DATE(date\_change\_inner\_info, '%d.%m.%Y')

WHERE date\_change\_inner\_info LIKE '\_\_.\_\_.\_\_\_\_';

-- исправления в нескольо этапов

UPDATE vac\_2020\_21 SET date\_time\_change\_inner\_info\_2 =

CASE

WHEN date\_time\_change\_inner\_info LIKE '\_\_.\_\_.\_\_\_\_ \_\_:\_\_'

THEN STR\_TO\_DATE(date\_time\_change\_inner\_info, '%d.%m.%Y %H:%i')

WHEN date\_time\_change\_inner\_info LIKE '\_\_.\_\_.\_\_\_\_ \_:\_\_'

THEN STR\_TO\_DATE(date\_time\_change\_inner\_info, '%d.%m.%Y %k:%i')

ELSE CONVERT(date\_time\_change\_inner\_info, DATETIME)

END;

-- value\_counts

SELECT date\_change\_inner\_info AS `Значения`,

COUNT(date\_change\_inner\_info) AS `Количество`

FROM vacancies WHERE date\_change\_inner\_info IS NOT NULL

GROUP BY date\_change\_inner\_info;

-- HAVING length(active) < 10;

-- заменяем значения ячеек

UPDATE vacancies

SET date\_change\_inner\_info = NULL

WHERE date\_change\_inner\_info = '';

-- проверить значение в обоих столбцах

SELECT date\_change\_inner\_info, date\_change\_inner\_info\_2

FROM vacancies

GROUP BY date\_change\_inner\_info, date\_change\_inner\_info\_2;

# 3. Схема базы данных.

# 4. Визуализация данных.

1. Power BI.

5.1 Сохранение уникальных значений.

CREATE TABLE diploma.vacancies\_UQ LIKE visual.vacancies\_UQ;

ALTER TABLE visual.vacancies\_UQ

MODIFY COLUMN identifier varchar(40) PRIMARY KEY;

INSERT IGNORE INTO visual.vacancies\_UQ

SELECT \* FROM diploma.vacancies ORDER BY date\_last\_updated;

5.2 Выгрузка по году для визуализации.

Сортируем по дате и году.

SELECT date\_creation AS `Значения`,

COUNT(date\_creation) AS `Количество`

FROM curricula\_vitae

GROUP BY date\_creation

ORDER BY year(date\_creation);

-- создаем пустую копию таблицы

CREATE TABLE cur\_vit\_2020\_21 LIKE curricula\_vitae;

-- меняем столбец на primary key

ALTER TABLE cur\_vit\_2020\_21

MODIFY id\_cv VARCHAR(40) PRIMARY KEY;

-- добавляем значения в таблицу, заменяя уникальный ключ, если надо

INSERT IGNORE cur\_vit\_2020\_21

SELECT \* FROM curricula\_vitae

WHERE YEAR(date\_creation) = 2020 OR YEAR(date\_creation) = 2021

ORDER BY date\_last\_updated;

-- добавляем значения в таблицу c primary key, заменяя уникальный ключ, если надо

REPLACE INTO curricula\_vitae\_2021\_tmp SELECT \* FROM curricula\_vitae\_2021

ORDER BY date\_last\_updated;

Сохранить из одной базы в другую часть таблицы:

-- из одной таблицы в другую по условию (responses\_21 заполнила)

INSERT INTO visual.responses\_21

SELECT \* FROM diploma.responses

WHERE id\_vacancy IN

(SELECT identifier FROM visual.vacancies\_21);

Выгрузка айтишников за 20\_21 год: 40\_198 записей.

CREATE TABLE visual.it\_20\_21 LIKE diploma.vac\_2020\_21;

INSERT INTO visual.it\_20\_21

SELECT \* FROM diploma.vac\_2020\_21

WHERE industry = 'InformationTechnology';

* + Пустые ячейки отрасли.

SELECT count(1) FROM vacancies\_20\_21 WHERE industry IS NULL; -- 8709

SELECT count(1) FROM vacancies\_21 WHERE industry IS NULL; -- 7911

SELECT count(1) FROM diploma.vac\_2020\_21

WHERE industry IS NULL AND YEAR(date\_creation) = 2020; -- 798

* + Количлелство ИТ вакансий за 20 и 21 год.

SELECT count(1) FROM it\_20\_21 WHERE year(date\_creation) = 2020; -- 21509

SELECT count(1) FROM it\_20\_21 WHERE year(date\_creation) = 2021; -- 18689

* + Разных наименований Information Technologies в айтишниках за 20, 21 годы.

SELECT title FROM it\_20\_21 GROUP BY title; -- 3\_591

* + Количество резюме ИТ за 20, 21

SELECT count(1) FROM it\_cv\_20\_21 WHERE year(date\_creation) = 2020; -- 72197

SELECT count(1) FROM it\_cv\_20\_21 WHERE year(date\_creation) = 2021; -- 26386

Почему выбираем direct query. Больше 20 Мб файла, при оперативке 8 Гб, файл pbix уже тормозит.

Содержание.