“CRUD” ON SQL BY PYTHON

Amir Anissian

Contents

Introduction 2

Diagram of database 2

Making Data Base 2

CRUD Operations in Python 2

Window Function 3

Query Plan 3

Introduction

Task is making a Python application for a company who want to manipulate data in their database.

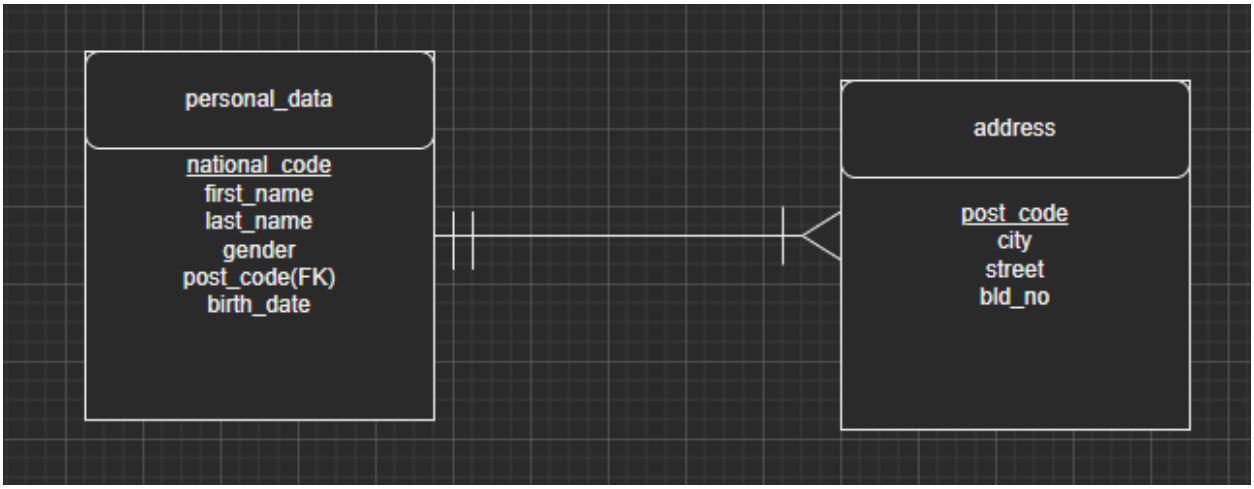
The company work is registering people and I want to help them with this app.

I will make an app by which user can *enter (Create)*, *Read*, *Update* and *Delete* (CRUD) data in app and app will update the database by these entries.

Diagram of database

I designed two tables, “address” and “personal_data”. Table of address has post_code as primary key (which is kind of ID for addresses, and it isn’t postcode as post Co use) and table of persona_data has national_code column as primary key and post_code as foreign key from another table.

P.S: I could not use street name as primary key because maybe we have same name for street in different cities and there are many buildings No. in one street. So, I use an ID for addresses which I named it postcode to make sense. I know that in real world postcode in some countries such as Sweden is related to an area but in some countries such as Iran it related to a specific apartment. Anyway, as I mentioned before, it is an ID for addresses, and it is different from general meaning of postcodes that post companies use for addressing.



Making Data Base

I have made two tables, “address” and “personal_data”, in SQL (Table Creation’s queries) and I provide the data of 20 people and 50 different addresses. People information is contained *national_code*, *first name*, *last name*, *postcode*, *gender* and *date of birth* (with limitation from 1900-01-01 to current date). Address table is contained *postcode* (address ID), *city*, *street* and *No.*

CRUD Operations in Python

I have made the apps in Python which do CRUD on the database, you can find scripts in (Python scripts). By running each script app ask user to enter data and database will be created one row by that data. App

can take national code and update or delete that row on database. On the other hand, by taking postcode (address ID), app can update or delete the data on address table.

For updating app ask user to choose which field he/she wants to update.

If user doesn't import a number for national code or postcode (address ID), app will print an error message and ask user to enter a number.

In addition, if user enter a date before 1900-01-01 or after current date for date of birth, app asks him/her to 'Check date of birth'.

Window Function

I try to make a window function to find out how many people live in an address. (You can find the query in Window function query and Python scripts too.)

post_code	national_code	first_name	last_name	gender	birth_date	city	street	bld_no	num_pple_per_add
11872	202111105167	Kazuhito	Cappelletti	M	2012-11-10	Laholm	Skeppsmäklargatan	17	1
12030	200503084964	Guoxiang	Nooteboom	M	2005-03-08	Halmstad	Båtbyggargatan	16	1
12031	198802204761	Berni	Genin	M	1988-02-20	Falsterbo[1]	Rorgängargatan	15	1
12032	198603214558	Eberhardt	Terkki	M	1986-03-21	Falköping	Fartygsgatan	14	1
12061	198306074355	Patricio	Bridgland	M	1983-06-07	Arboga	Båtbyggargatan	13	1
12062	199902084152	Mary	Sluis	F	1999-02-08	Nyköping	Stapelgatan	12	1
12063	199812155571	Emil	Jespersen	M	1998-12-15	Varberg	Sjökortsgatan	11	1
12063	200708243949	Duangkaew	Piveteau	F	2007-08-24	Varberg	Sjökortsgatan	11	2
12064	201912263746	Sumant	Peac	F	2019-12-26	Kungälv	Pollargatan	10	1
12065	198306063654	Amir	Anissian	M	1983-06-06	Kalmar	Kanalvägen	9	1
12065	199310073543	Saniya	Kalloufi	M	1993-10-07	Kalmar	Kanalvägen	9	2
12066	198711153340	Tzvetan	Zielinski	F	1987-11-15	Helsingborg	Fendergatan	8	1
12068	200601213137	Anneke	Preusig	F	2006-01-21	Visby	Båtklubbsgatan	7	1
12069	200802112934	Kyoichi	Maliniak	M	2008-02-11	Södertälje	Maltgatan	6	1
12070	199012022731	Chirstian	Koblick	M	1990-12-02	Västerås	Vindkraftsvägen	5	1
12071	201104252528	Parto	Bamford	M	2011-04-25	Lund	Vattenkraftsvägen	4	1
12079	198307071456	Emma	Ericsson	F	1983-07-07	Skara	Solkraftsvägen	3	1
12079	200208082325	Bezalel	Simmel	F	2002-08-08	Skara	Solkraftsvägen	3	2
12079	202208083654	Atousa	Miri	M	2022-08-08	Skara	Solkraftsvägen	3	3
13570	198306062122	Georgi	Facello	M	1983-06-06	Sigtuna	Skrubbasandsvägen	2	1

For example, in this window function we can see most of the households contain one person.

Query Plan

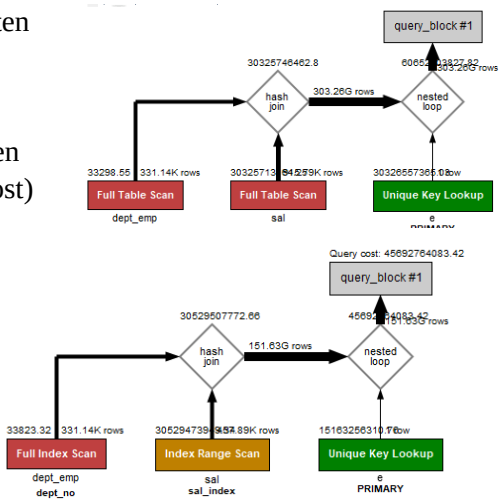
At first, I tried to check the query plan with this query:

```
SELECT
   dept_emp.dept_no, e.emp_no, sal.salary
FROM
   employees AS e
      INNER JOIN
   salaries AS sal
 USING (emp_no)
      INNER JOIN    dept_emp;
```

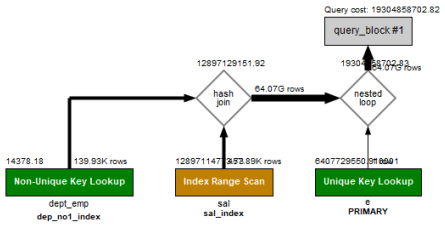
in this case all bars were red. It means our queries have been written in a high-cost way. So, I tried to find a solution to my queries become cheap.

I tried indexing and it just turn the employees table change to green because I select its primary key. Other tables are still red (High cost)

When I use indexing with a condition on salary table it become orange. (Medium- partial index)



When I put a condition on dept_emp and filter it, dept_emp's bar became green too.



I tried also to change the position of tables in the join part of query, but I couldn't make any change in color of bars anymore. I have indexing salary and emp_no columns but it doesn't become green. On another try, I tried indexing an irrelevant column, from_date, to see if it has any effect on our color of bars. It hasn't any effect.

At last, I try to change condition on salary table from "> 60000" to "= 60000" and the color of salary bar change to green. When I compare Explain result of these two queries, we can see in figure 1 query scan 457893 rows but in figure2 query scan just 32 rows.

lect_type	table	partitic	type	possible_keys	key	key_len	ref	rows	filtered	Extra
MPLE	dept_emp	NULL	ref	PRIMARY,dep_no_index,dep_no1_index	dep_no1_index	16	const	139932	100.00	Using where; Using index
MPLE	e	NULL	eq_ref	PRIMARY,emp_no_index	PRIMARY	4	employees.dept_emp.emp_no	1	100.00	Using index
MPLE	sal	NULL	range	sal_index	sal_index	4	NULL	457893	100.00	Using where; Using index; Using join buffer (ha...

Figure 1Explain with condition "salary >6000"

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	sal	NULL	ref	sal_index	sal_index	4	const	32	100.00	Using index
1	SIMPLE	dept_emp	NULL	ref	PRIMARY,dep_no_index,dep_no1_index	dep_no1_index	16	const	139932	100.00	Using where; Using index
1	SIMPLE	e	NULL	eq_ref	PRIMARY,emp_no_index	PRIMARY	4	employees.dept_emp.emp_no	1	100.00	Using index

Figure 2 Explain with condition "salary=60000"

In conclusion, I can say when we make any limitation on scanning of the tables it become cheap, effective and fast.