COMP306 Database Management Systems **Date: 21.05.2020**

Group Assignment - Term Project Phase 3 - Physical Design

(Subject No: 30 - UNESCO Endangered Languages Program)

Group 17, Masters: Yasin Uslu – 71028, Shukhrat Khuseynov – 70495.

Report

SQL scripts: All the required steps were implemented. Please run triggers and views before DML statements, otherwise there will be a compilation error.

Part 1

These are the estimated average lengths for three tables of our database:

	Initial #of rows	#of rows in 3 months	#of rows in a year	Avg. row length **(approx. digits/places)**
Volunteer	5	200	500	**32**
Languages	5	20	35	**40**
Research	5	40	90	**140.5**

The avg. row lengths can also be understood as bytes approximately.

Part 2

We expect 20 on average and at most 50 concurrent users to use this database, which are volunteers having computer and paper responsibilities. These are the expected numbers of transactions for three tables of our database (in a given format):

3m **/** 12m **/ avg**	INSERT	UPDATE	DELETE	SELECT	**Transactions** (total)
Volunteer	200 / 500 / **350**	80 / 100 / **90**	50 / 130 / **90**	200 / 1400 / **800**	530 / 2130 / **1330**
Languages	20 / 40/ **30**	100 / 500 / **300**	30 / 150 / **90**	680 / 1820 / **1250**	830 / 2510 / **1670**
Research	40 / 90 / **65**	90 / 200 / **145**	30 / 70 / **50**	870 / 1930 / **1400**	1030 / 2290 / **1660**

The most intensive transactions are expected to be on selects, as seen above (which is often).

Part 3

For the disk storage we choose the database block size to be 4096 bytes (4KB), since the default OS block size for Linux and Windows machines is 4096 bytes. It potentially seems to be efficient to have the same block size for the database as in the operating system.

In our database we can use three tablespaces: one for languages and cultural heritage with its subtypes, one for UNESCO offices, cities, volunteers with their subtypes and citizenships, and one for remaining tables like events, researches, etc. We chose to have more than one tablespace to have better efficiency, more balanced memory distribution, and possibility to store tablespaces on different discs which is good for recovery purposes. On this tablespaces, we can keep up to 100-200 data files, to secure ourselves from possible data losses.

For most tables which are going to have lots of rows we plan to use primary indexes, which are single-level and sparse. For smaller tables such as for president and vice president, which are rarely modified and have chronological order to begin with, we can use unordered files like log. For tables such as for government and citizenship, which have relatively low amounts of entries, we plan to use ordered sequential files. We use indexing for larger tables for the sake of improved performance while accessing the rows.

We decided to use RAID 5 level, as it is highly popular, and since we use indexing which will be stored as parity, which is suitable for this RAID level. It is less costly than RAID 10 level, which might excel in many more parameters, but considering a limited budget in this project, RAID 5 is suitable. Although it takes time for writes, its read performance is good. Moreover, it is relatively safe during data losses since parity and blocks are distributed in a round-robin fashion.

Part 4

For physical data protection, we are planning to use the Recovery Manager (RMAN) with cumulative incremental backups of L0 and L1 levels. Although it will take more time to back up the database, we prefer it for faster recovery, given that the time for recovery might be limited. So, the schedule is as follows: L0 backup on every last day of month, L1 backup on every Sunday, and incremental backup at the end of each day. For example, if the database crashes on Friday, 22nd of May: first of all, RMAN will recover everything cumulatively for 30th of April (L0), then cumulatively for last Sunday, which is 17th of May (L1), and lastly, for each day from Monday to Thursday one by one incrementally will be recovered.

For disaster recovery, we are thinking to use active data guard architecture with 1 physical standby (no need for logical one) and asynchronous transaction shipping (not to freeze the production database). If something goes wrong, i.e. any crash, then the database will be using the standby site. The short-time data (in terms of seconds or minutes) can be recovered by this technology.

We prefer a cold backup strategy, which can be done at nights. UNESCO projects do not require 24 hour access, so cold backup is easier and more efficient. For logical data protection, aiming error investigation, we might also use flashback technologies. Moreover, the data recovery advisor will be used for planning the recoveries and minimizing time for problem identification. Lastly, critical checks of the database are planned to be twice a year to ensure that the recovery system works well.

Part 5
We prefer using mandatory (role-based) security mechanisms for our database, since it is easier to manage. There is no need for too many details of granting privileges in this UNESCO program, which are available in discretionary mechanisms. We plan to have few roles (user types): a database administrator, top managers (president and vice president), 10 volunteers responsible for inserting and updating the database, and a standard user (a volunteer, which is dedicated to work with the database by only viewing it). The database administrator will be responsible for setting and regularly changing the password for all types of user, especially the ones who can do more than only viewing.

Part 6
Assuming 20 concurrent users on average, as stated above, fifty times as much of it will be 1000 concurrent users, which is a lot for our current database system. First of all, since concurrency in databases requires some degree of serialized computation, which is a must for the correct output due to ACID properties of RDBMS, the scheduling will slow down the performance. Some transactions have to wait before some others commit. This is the bottleneck of relational databases. Assuming that almost all transactions will be reads (viewing only) simplifies the problem. The CPU and memory have also their limitations, so a single server or a few might not be enough. Having more physical standby sites, adding a logical standby server, clustering the database might help. Some sources suggest that having more tablespaces helps for concurrency cases, we can increase the specified number and partitioning the database accordingly. Moreover, albeit costly, RAID10 (1+0) level can be used in this case to improve the performance. Lastly, due to the increased amount of data (together with the increased concurrent users), the deployment of multi-level indexing such as B+ tree can be useful for speed and performance in general.

Part 7
If we had two more weeks to work on this database, we could enter many more rows with inserts and add some other triggers on SQLite for better functioning of our database system. We could try to do performance optimizations of our database system considering CPU and memory parameters and optimizing concurrent usage, if it is possible. We could also try doing some transactions by choosing appropriate scheduling types. In general, other than trying new features, we would check and analyze the database for possible bugs and drawbacks to make it perfect.

Work distribution
It was more or less equal. For conceptual design, we were doing everything together while meeting. Similarly, for logical and physical design phases, although we had a remote education, we were still doing everything together by using Lucidchart, repl.it, and Google Docs having also called each other by phone simultaneously to imitate a real meeting.