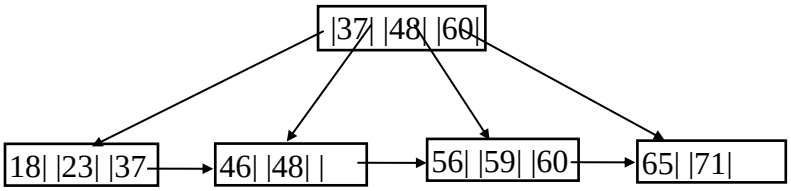
Homework 9

(Maximum 60 points)
Due end of day (11:59 p.m.) Friday November 15, 2019

1 (30 pts) **[Index structure: B+ tree]**

a) (20 pts) Given a B+ tree made up of nodes with index key entries as shown below, show the B+ tree structure after inserting an index key 21 and then after inserting an index key 10. Your answer should show the two resulting B+ trees separately in sequence. Describing the steps of inserting the keys is not required in your answer, although it is recommended for the sake of allowing for partial credits in case of a wrong answer.



b) (10 pts. *Research exercise*) B+ tree index structure is said to be an improvement of B tree index structure. The most important distinction between them is that data record pointers exist in both internal and leaf nodes (i.e., blocks) for a B tree whereas only in the leaf nodes for a B+ tree. Explain why this distinction would make B+ tree a more efficient structure (in terms of index search speed) overall than a B tree. Reference: Section 17.3.1 for B-trees.

2. (30 pts) **[Physical database design: clustering and denormalization] (Lab exercise)**

For frequently executed queries involving the join of two or more tables, we may want to merge them into one table either at the file level or at the table level, in order to make the query execute faster. Merging at the file level is called "clustering," and merging at the table level is called "denormalization." The objective of this exercise is get your hands wet a bit with merging three tables at the two levels. For each of the two exercises (a and b) below, write all statements in one script file and upload both the input script file and the runscript output to Blackboard. Clearly label each step and add comments in your script for better readability; points will be deducted for a script with no comment. Make sure to drop all database objects at the end of your script, so that you can rerun the script as many times as you want. You can modify and use the script file create_tables_dept_locs_projs.sql and insert_tuples_dept_locs_projs.sql.

a) (10 pts) [Clustering] Write a script that performs the following steps.

i. Create a cluster containing tables Department, Dept_locations, and Project, where dnumber is the clustering key. Use the table schema shown in the textbook Figure 5.7 (page 164). Omit the foreign key constraint for mgrssn in the Department table, since the table Employee is omitted in this exercise. Commit after creating the cluster and the tables in the cluster.
ii. Create an index on the clustering key (i.e., dnumber). (There are only a handful of tuples, so it's an overkill to create an index here, but let's do it for the experience.) Commit after creating the index.
iii. Insert the tuples shown in the textbook Figure 5.6 (page 162) into the clustered tables. Insert null for the column mgrssn of the Department table, since the table Employee is omitted in this exercise. You can use insert_tuples_dept_locs_projs.sql as is. (Specifically,

include the command “start insert_tuples_dept_locs_projs.sql” in your input script instead of copying all insert statements from the provided script file.)

iv. Execute the following queries as SQL SELECT statements against the clustered tables: (1) retrieve tuples from each clustered table (i.e., Department, Dept_locations, Project), (2) retrieve the location and the project (name) of a department named ‘Administration’. Note that, although the files of the three tables have been merged into one, we still see them as separate tables at the logical level.

v. Drop the index, clustered tables, and the cluster. Commit after dropping them all.

b) (20 pts) [Denormalization] Write a script that performs the following steps.

i. Merge the schemas of three tables Department, Dept_locations, and Project into one table schema. Show the resulting schema (with no duplicate columns) and show all functional and multi-valued dependencies in the way shown in Figure 14.12 (page 485) of the textbook. (Use a double arrowhead for a multi-valued dependency.)

ii. Create the merged table. Specify the primary key of the merged table. Commit after creating the table. In addition, discuss how you would enforce the functional dependency constraints identified in step bi. (Note that specifying the primary key of the merged relation does not automatically enforce the dependency constraints identified in the exercise bi, because the merged table is not in the normal form.)

iii. Insert into the merged table tuples merged from the tuples in the original tables. Commit after the insertion is done. This insertion of merged tuples can be done by executing one SQL statement which inserts the result of a SELECT statement into the merged table. The SELECT statement combines the tuples from the three original tables (i.e., Department, Dept_locations, Project) through joins. (Before executing the SELECT statement, create the original three tables using the provide script create_tables_dept_locs_projs.sql and insert tuples into them using the provided script insert_tuples_dept_locs_projs.sql. Make sure to drop the original tables after inserting tuples into the merged table.)

iv. Create an index on the column dnumber. (As in the exercise a, there are only a handful of tuples, so it’s an overkill to create an index here, but let’s do it for the experience.) Commit after creating the index.

v. Execute the following queries as SQL SELECT statements against the merged table: (1) retrieve tuples belonging to each of the original tables (i.e., Department, Dept_locations, Project), (2) retrieve the location and the project (name) of a department named 'Administration'. Make sure there are no duplicate tuples in the query results.

vi. Drop the index and the merged table. Commit after dropping them all.

Last modified: November 7, 2019