University of California, Los Angeles
**Department of Computer Science**

**Computer Science 143** **Prof. Ryan Rosario**

# Homework 3

*Solutions*

**Please remember the following:**

1. Homework is mostly graded on completion. We may grade a few parts, but it will never be the majority of the grade on the assignment. So try your best, and focus on solving the problems. Consider homework (and studying the solutions) as practice for the midterm.
2. Homework must be submitted digitally, on CCLE. We will not do any paper grading. You can use a text file, but if you use Word, a PDF is preferred rather than a DOC file.
3. If there are any exercises that are difficult to do digitally (such as diagrams or math), consider scanning your drawing or math, or using a graphics program (even a readable MS Paint is fine) or Equation Editor.
4. **For the sanity of the grader** we will ask you to run the queries and submit the result. You may lose points if you only provide a query.
5. Solutions will be posted.

## Part 1: Go Long or Go Wide?

The website `keyvalues.com` allows startups and some large companies to discuss their company culture, something that some of you will find to be much more important than a huge paycheck. The purpose of this site is to provide candidates and other interested parties an in depth look at the company culture on a variety of dimensions. The content tends to discuss cultural values that are often overlooked in interviews. The table below shows some of them:

| Team Values | Personal Health | Daily Routines | Engineering | Career Growth |
|---|---|---|---|---|
| Engages with Community | Work/Life Balance | Eats Lunch Together | Open Source Contributor | Promotes from Within |
| Team is Diverse | Ideal for Parents | Light Meetings | Start-to-Finish Ownership | Good for Junior Devs |
| Risk Taking > Stability | Fosters Psychological Safety | Thoughtful Office Layout | Fast-Paced | High Retention |

| Strategy | Company Properties |
|---|---|
| Engineering-Driven | Remote Work OK |
| Data-Driven | |
| Rapidly Growing Team | |

**Exercise**

Help your professor (please)! (Though I've already done this) Take a look at the data. Write a query that creates a 0/1 matrix from this data. The rows should correspond to companies, the columns should correspond to cultural values. The value of each cell should be a 0 or 1 – a 1 if the value is associated with the company, 0 otherwise.

The query will be tedious, but in the "lots of copy/paste" way. You may want to write a script to generate the query (like in Project 1). Some of you may find a much better way to do this that does not require that. I do not know if it is possible, so if you are adventurous, you may want to research it.

**To grade this problem, please submit your query, the number of rows in the output, and the number of columns in the output. Please do not submit the matrix.**

With this 0/1 matrix, we can then create visualizations of which companies are similar to others, and which values are similar to others using singular value decomposition, principal component analysis, or multiple correspondence analysis. We won't do that for this class, but if you are interested, have some fun with the data.

**This solution is written for MySQL, but can be adapted for PostgreSQL using the queries in the slides.**

The solution to this problem is very tedious and involves a lot of copy-pasting, or does it?

### Method 1: Using a Script

One reason for this problem is that pivot tables were not covered on the midterm. The other is to teach a very important lesson: automate, automate and automate more. I've heard some say that the best software engineers automate themselves out of a job. Of course, they just move on to more rewarding projects. **I expected people would figure out that they may want to write a script that generates a SQL query.** After loading the data (we could even automate that), we get the following:

```
#!/usr/bin/python

import mysql.connector

# Usernames, passwords, hostname and database name should never be hardcoded.
# They should be put into a configuration file that is read by Python, and is
# not checked into version control. For CS143, we will ignore it.
_USER = "cs143"
_PASSWORD = "password"
_HOST = "127.0.0.1"
_DATABASE = "hw3"

if __name__ == "__main__":
        # Start a list of query tokens.
        query_components = ["SELECT company"]
        # Each column definition is the same.
        conditional_template = " IFNULL(SUM(IF(value = '{}', 1, NULL)), 0) as {}"

        # Setup MySQL connection.
        cnx = mysql.connector.connect(user=_USER, password=_PASSWORD, host=_HOST, database=_DATABASE)
        cursor = cnx.cursor()

        # Get the set of cultural values.
        values_query = "SELECT DISTINCT(value) FROM keyvalues"
        cursor.execute(values_query)
        resultset = cursor.fetchall()
        values = [val[0].replace('-', '_') for val in resultset]

        # Add each column statement to the list of query tokens.
        [query_components.append(conditional_template.format(value, value)) for value in values]

        # Join them together with comma and newline.
        pivot_query = ',\n'.join(query_components) + " FROM keyvalues GROUP BY company"

        # Get the row and column count.
        cursor.execute(pivot_query)
        resultset = cursor.fetchall()
        cnx.close()

        with open("hw3part1solution.sql", "w") as f:
        f.write(pivot_query)

        rows = len(resultset)
        firstrow, *_ = resultset
        cols = len(firstrow)
        print("There are {} rows and {} columns".format(rows, cols))
```

Running the script yields the result of **67** rows (companies) and **47** columns (cultural values).

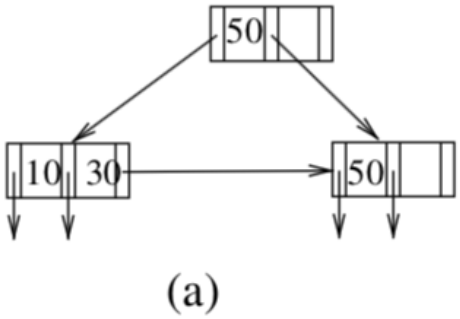
**Method 2: Brute-Force Copypasta**

```
SELECT company,
        IFNULL(SUM(IF(value = 'bonded_by_product', 1, NULL)), 0) as bonded_by_product,
        IFNULL(SUM(IF(value = 'project_ownership', 1, NULL)), 0) as project_ownership,
        IFNULL(SUM(IF(value = 'friends_outside_work', 1, NULL)), 0) as friends_outside_work,
        IFNULL(SUM(IF(value = 'feedback', 1, NULL)), 0) as feedback,
        IFNULL(SUM(IF(value = 'impressive_teammates', 1, NULL)), 0) as impressive_teammates,
        IFNULL(SUM(IF(value = 'personal_growth', 1, NULL)), 0) as personal_growth,
        IFNULL(SUM(IF(value = 'fast_paced', 1, NULL)), 0) as fast_paced,
        IFNULL(SUM(IF(value = 'lunch_together', 1, NULL)), 0) as lunch_together,
        IFNULL(SUM(IF(value = 'remote_ok', 1, NULL)), 0) as remote_ok,
        IFNULL(SUM(IF(value = 'customer_first', 1, NULL)), 0) as customer_first,
        IFNULL(SUM(IF(value = 'many_hats', 1, NULL)), 0) as many_hats,
        IFNULL(SUM(IF(value = 'quality_code', 1, NULL)), 0) as quality_code,
        IFNULL(SUM(IF(value = 'open_communication', 1, NULL)), 0) as open_communication,
        IFNULL(SUM(IF(value = 'internal_promotion', 1, NULL)), 0) as internal_promotion,
        IFNULL(SUM(IF(value = 'retention', 1, NULL)), 0) as retention,
        IFNULL(SUM(IF(value = 'product_driven', 1, NULL)), 0) as product_driven,
        IFNULL(SUM(IF(value = 'worklife_balance', 1, NULL)), 0) as worklife_balance,
        IFNULL(SUM(IF(value = 'light_meetings', 1, NULL)), 0) as light_meetings,
        IFNULL(SUM(IF(value = 'flex_hours', 1, NULL)), 0) as flex_hours,
        IFNULL(SUM(IF(value = 'inclusive', 1, NULL)), 0) as inclusive,
        IFNULL(SUM(IF(value = 'psychologically_safe', 1, NULL)), 0) as psychologically_safe,
        IFNULL(SUM(IF(value = 'diverse_team', 1, NULL)), 0) as diverse_team,
        IFNULL(SUM(IF(value = 'eq_iq', 1, NULL)), 0) as eq_iq,
        IFNULL(SUM(IF(value = 'parents', 1, NULL)), 0) as parents,
        IFNULL(SUM(IF(value = 'open_source', 1, NULL)), 0) as open_source,
        IFNULL(SUM(IF(value = 'continuous_delivery', 1, NULL)), 0) as continuous_delivery,
        IFNULL(SUM(IF(value = 'engages_community', 1, NULL)), 0) as engages_community,
        IFNULL(SUM(IF(value = 'cross_dep', 1, NULL)), 0) as cross_dep,
        IFNULL(SUM(IF(value = 'safe_env', 1, NULL)), 0) as safe_env,
        IFNULL(SUM(IF(value = 'rapid_growth', 1, NULL)), 0) as rapid_growth,
        IFNULL(SUM(IF(value = 'new_tech', 1, NULL)), 0) as new_tech,
        IFNULL(SUM(IF(value = 'creative_innovative', 1, NULL)), 0) as creative_innovative,
        IFNULL(SUM(IF(value = 'data_driven', 1, NULL)), 0) as data_driven,
        IFNULL(SUM(IF(value = 'flat_organization', 1, NULL)), 0) as flat_organization,
        IFNULL(SUM(IF(value = 'engineering_driven', 1, NULL)), 0) as engineering_driven,
        IFNULL(SUM(IF(value = 'agile_dev', 1, NULL)), 0) as agile_dev,
        IFNULL(SUM(IF(value = 'pair_programs', 1, NULL)), 0) as pair_programs,
        IFNULL(SUM(IF(value = 'team_oriented', 1, NULL)), 0) as team_oriented,
        IFNULL(SUM(IF(value = 'internal_mobility', 1, NULL)), 0) as internal_mobility,
        IFNULL(SUM(IF(value = 'physical_wellness', 1, NULL)), 0) as physical_wellness,
        IFNULL(SUM(IF(value = 'benefit_company', 1, NULL)), 0) as benefit_company,
        IFNULL(SUM(IF(value = 'interns', 1, NULL)), 0) as interns,
        IFNULL(SUM(IF(value = 'junior_devs', 1, NULL)), 0) as junior_devs,
        IFNULL(SUM(IF(value = 'risk_taking', 1, NULL)), 0) as risk_taking,
        IFNULL(SUM(IF(value = 'good_beer', 1, NULL)), 0) as good_beer,
        IFNULL(SUM(IF(value = 'office_layout', 1, NULL)), 0) as office_layout
FROM keyvalues GROUP BY company
```

Some people used overly complicated queries that used several subqueries and/or several outer joins. **The code for this query comes directly from the lecture slides.**

**Part 2: Seeing the Forest for the Trees**

Consider the following B+tree for part (a). You may need to review chapter 14, or the lecture slides.




(a)

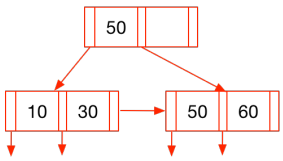
(a) Show the final B+tree structure after we insert 60, 20, and 80, in this order.

**This problem is problematic because there are different algorithms for inserting into and deleting from a B+ tree. The solution presented here matches the visual example given in lecture, but it will not match the result given by the online simulation, the method given in the book, or my rules of thumb in the lecture slides. There are differences even among online simulations.**

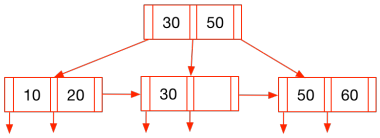
These inconsistencies come from two areas:

- If the left keys are strictly $<$ and the right keys are $\geq$ the parent key **OR** if the left keys are $\leq$ and the right keys are $>$ than the parent key.
- When there is an overflow, which key gets copied or moved up to the parent.

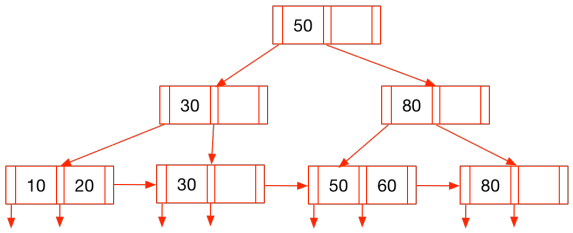
**Inserting 60** is easy, we just insert it in the proper spot. No overflow occurs.




**Inserting 20** causes an overflow in the bottom left leaf, but we know that 20 must go into this leaf, so we evict 30 and move it up a level. Unfortunately, we have lost 30 as a key which is illegal, so we must put it somewhere, but where? We cannot move it to the right subtree because 30 is less than 50, so we must create a new node under the root, only containing 30.



It gets worse. Now we need to **Insert 80**. It should go to the extreme far right of the bottom right leaf. But we have an overflow if we insert it there. We create a new node under the root with a router 80, which points to 50 and 60, and we create a new far right leaf containing only 80.




***For (b) through (d), suppose there is a relation*** $r(A, B, C)$***, with a B+-tree index with search key*** $(A, B)$***.***

Professor had such high hopes for these problems from the book. They were shattered in a million tears.

(b) What is the worst-case cost of finding records satisfying $10 < A < 50$ using this index, in terms of the number of records retrieved $n_1$ and the height $h$ of the tree?

This query does not correspond to a range query on the search key as the condition on the first attribute if the search key is a comparison condition. It looks up records which have the value of $A$ between 10 and 50. However, each record is likely to be in a different block, because of the ordering of records in the file, leading to many I/O operations. In the worst case, for each record, it needs to traverse the whole tree (cost is $h$), so the total cost is $n_1 h$.

(c) What is the worst-case cost of finding records satisfying $10 < A < 50 \wedge 5 < B < 10$ using this index, in terms of the number of records $n_2$ that satisfy this selection, as well as $n_1$ and $h$ defined above?

This query can be answered by using an ordered index on the search key $(A, B)$. For each value of $A$ this is between 10 and 50, and the system located records with $B$ value between 5 and 10. However, each record could is likely to be in a different disk block. This amounts to executing the query based on the condition on $A$, and this costs $n_1 h$. Then these records are checked to see if the condition on $B$ is satisfied. So, the total cost in the worst case is $n_1 h$.

(d) Under what conditions on $n_1$ and $n_2$ would the index be an efficient way of finding records satisfying $10 < A < 50 \wedge 5 < B < 10$?

$n_1$ records satisfy the first condition and $n_2$ records satisfy the second condition. When both the conditions are queried, $n_1$ records are output in the first stage. So, in the case where $n_1 = n_2$, no extra records are output in the first stage. Otherwise, the records which dont satisfy the second condition are also output with an additional cost of $h$ each (worst case).

**Part 3: If you Like it Then you Shoulda Put an Index on It**

(a) Why is a hash structure not the best choice for a search key on which range queries are likely?

A range query cannot be answered efficiently using a hash index, we will have to read all the buckets. This is because key values in the range do not occupy consecutive locations in the buckets, they are distributed uniformly and randomly throughout all the buckets.

(b) Our description of static hashing assumes that a large contiguous stretch of disk blocks can be allocated to a static hash table. Suppose you can allocate only $C$ contiguous blocks. Suggest how to implement the hash table, if it can be much larger than $C$ blocks. Access to a block should still be efficient.

A separate list/table as shown below can be created. Starting address of first set of $C$ blocks is $C$, and the starting address of the next set of $C$ blocks is $2C$.

Desired block address = Starting address (from the table depending on the block number) + blocksize * (blocknumber % $C$).

For each set of $C$ blocks, a single entry is added to the table. In this case, locating a block requires 2 steps: First we use the block number to find the actual block address, and then we can access the desired block.