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 final exam.
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! (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 may be tedious, but in the "lots of copy/paste" way. 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. The output itself you can submit as a text file if you wish, but it is probably too tedious to read.

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.

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).

There are many things I do not know, and I was curious if somebody could come up with a method that did not involve brute-force copy-paste or writing a script. Sure enough, you didn't fail me!

Method 2: Using Stored Procedure Syntax

Apparently we can use stored procedure syntax to solve this problem.

```
SET @sql = NULL;
SET SESSION group_concat_max_len = 10000;

SELECT
 GROUP_CONCAT(DISTINCT
  CONCAT(
   "IFNULL(GROUP_CONCAT((CASE WHEN value LIKE ' %", value, "% ' THEN 1 ELSE NULL END)), 0) AS ' ", value, " ' "
  )
 ) INTO @sql
FROM keyvalues;

SET @sql = CONCAT('SELECT company, ', @sql, ' FROM keyvalues GROUP BY company');
PREPARE stmt FROM @sql;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
```

Method 3: 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: `EXPLAIN` Yourself

In the solutions for Homework 2, I gave several ways of writing the same query. For part 1B:

1. No Subquery with `DISTINCT`
2. `SELECT` within a `SELECT`
3. `JOIN` as a Filter
4. Using an `IN` Subquery as a Filter
5. Formal Left Semijoin

Run a SQL `EXPLAIN` on each of these queries. Just copy and paste the query from the solutions into a text editor and make any corrections so that it will run. The `EXPLAIN` output varies by implementation, but for MySQL, you can read more about it here: `https://dev.mysql.com/doc/refman/8.0/en/explain-output.html`.

Provide the output of *each* SQL `EXPLAIN`. Then write a paragraph (approximately) detailing what you notice overall (do not write a paragraph for each one). Which query appears to be the most efficient (intentionally vague)? What is your definition of efficient? Pay attention to the number of rows, the `filtered` column, the extra column, and any information provided about joins.

Unfortunately, MySQL does not provide as much useful information as other SQL systems and much less information than big data systems like Spark and Hive.

Still, we can glean some information from the query plans produced by `EXPLAIN`.

Usually, we want the query that executes the fastest. Typically, the fastest queries optimize down to queries that filter out the most number of rows in each subquery or in the innermost joins.

Method 0: No Subquery, `DISTINCT`

```
SELECT
    highway,
    area,
    COUNT(DISTINCT DAYOFYEAR(reported)) * 100 / 365 AS percentage_of_days_closed_365,
    COUNT(DISTINCT DAYOFYEAR(reported)) * 100 / 353 AS percentage_of_days_closed_353
FROM caltrans
WHERE text LIKE '%CLOSED%DUE TO SNOW%' OR text LIKE '%CLOSED%FOR THE WINTER%'
GROUP BY highway, area
ORDER BY percentage_of_days_closed_365 DESC;

+----+-------------+----------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra                                        |
+----+-------------+----------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------+
|  1 | SIMPLE      | caltrans | NULL       | ALL  | PRIMARY       | NULL | NULL    | NULL | 102730 |    20.99 | Using where; Using temporary; Using filesort |
+----+-------------+----------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------+
```

Method 1: Select within a Select

```
EXPLAIN SELECT
    highway,
    stretch,
    COUNT(1) AS days_closed,
    100 * COUNT(1) / 365 AS pct_closed_365,
    100 * COUNT(1) / 353 AS pct_closed_353
FROM (
    SELECT
        highway AS highway,
        area AS stretch,
        DATE(reported) AS closure
    FROM caltrans
    WHERE text LIKE '%CLOSED%' AND (
        text LIKE '%FOR THE WINTER%'
            OR text LIKE '%DUE TO SNOW%')
    GROUP BY highway, stretch, closure
) result
GROUP BY highway, stretch ORDER BY pct_closed_365 DESC;

+----+-------------+------------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------+
| id | select_type | table      | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra                                        |
+----+-------------+------------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------+
|  1 | PRIMARY     | <derived2> | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   2395 |   100.00 | Using temporary; Using filesort              |
|  2 | DERIVED     | caltrans   | NULL       | ALL  | PRIMARY       | NULL | NULL    | NULL | 102730 |     2.33 | Using where; Using temporary; Using filesort |
+----+-------------+------------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------+
```

Method 2: Join as a Filter

```
SELECT
    highway,
    stretch,
    COUNT(1) AS days_closed,
    100 * COUNT(1) / 365 AS pct_closed_365,
    100 * COUNT(1) / 353 AS pct_closed_353
FROM (
    SELECT
        c.highway AS highway,
        c.area AS stretch,
        DATE(c.reported) AS closure
    FROM caltrans c
    JOIN (
        SELECT
        DISTINCT
            highway,
            area
        FROM caltrans
        WHERE text like '%CLOSED%' AND (
            text LIKE '%FOR THE WINTER%' OR text LIKE '%DUE TO NOW%')
    ) snow_highways ON c.highway = snow_highways.highway
    WHERE text like '%CLOSED%' AND (
        text LIKE '%FOR THE WINTER%' OR text LIKE '%DUE TO SNOW%')
    GROUP BY highway, stretch, closure
) closures
GROUP BY highway, stretch
ORDER BY pct_closed_365 DESC;

+----+-------------+------------+------------+------+---------------+-------------+---------+---------------+--------+----------+----------------------------------------------+
| id | select_type | table      | partitions | type | possible_keys | key         | key_len | ref           | rows   | filtered | Extra                                        |
+----+-------------+------------+------------+------+---------------+-------------+---------+---------------+--------+----------+----------------------------------------------+
|  1 | PRIMARY     | <derived2> | NULL       | ALL  | NULL          | NULL        | NULL    | NULL          |  24001 |   100.00 | Using temporary; Using filesort              |
|  2 | DERIVED     | c          | NULL       | ALL  | PRIMARY       | NULL        | NULL    | NULL          | 102730 |     2.33 | Using where; Using temporary; Using filesort |
|  2 | DERIVED     | <derived3> | NULL       | ref  | <auto_key0>   | <auto_key0> | 8       | hw2.c.highway |     10 |   100.00 | NULL                                         |
|  3 | DERIVED     | caltrans   | NULL       | ALL  | PRIMARY       | NULL        | NULL    | NULL          | 102730 |     2.33 | Using where; Using temporary                 |
+----+-------------+------------+------------+------+---------------+-------------+---------+---------------+--------+----------+----------------------------------------------+
```

Method 3: Using an `IN` Subquery as a Filter

```
SELECT
    highway,
    stretch,
    COUNT(1) AS days_closed,
    100 * COUNT(1) / 365 AS pct_closed_365,
    100 * COUNT(1) / 353 AS pct_closed_353
FROM (
    SELECT
        c.highway AS highway,
        c.area AS stretch,
        DATE(c.reported) AS closure
    FROM caltrans c
    WHERE (highway, area) IN (
        SELECT
        DISTINCT
            highway,
            area
        FROM caltrans
        WHERE text like '%CLOSED%' AND (
            text LIKE '%FOR THE WINTER%' OR text LIKE '%DUE TO SNOW%')
    ) AND text LIKE '%CLOSED%' AND (
        text LIKE '%FOR THE WINTER%' OR text LIKE '%DUE TO SNOW%')
    GROUP BY highway, stretch, closure
) closures
GROUP BY highway, stretch
ORDER BY pct_closed_365 DESC;

+----+--------------+-------------+------------+--------+---------------+------------+---------+--------------------------+--------+----------+------------------------------------------
| id | select_type  | table       | partitions | type   | possible_keys | key        | key_len | ref                      | rows   | filtered | Extra
+----+--------------+-------------+------------+--------+---------------+------------+---------+--------------------------+--------+----------+------------------------------------------
|  1 | PRIMARY      | <derived2>  | NULL       | ALL    | NULL          | NULL       | NULL    | NULL                     |   2395 |   100.00 | Using temporary; Using filesort
|  2 | DERIVED      | c           | NULL       | ALL    | PRIMARY       | NULL       | NULL    | NULL                     | 102730 |     2.33 | Using where; Using temporary; Using 
|  2 | DERIVED      | <subquery3> | NULL       | eq_ref | <auto_key>    | <auto_key> | 265     | hw2.c.highway,hw2.c.area |      1 |   100.00 | NULL
|  3 | MATERIALIZED | caltrans    | NULL       | ALL    | NULL          | NULL       | NULL    | NULL                     | 102730 |     2.33 | Using where
+----+--------------+-------------+------------+--------+---------------+------------+---------+--------------------------+--------+----------+------------------------------------------
```

Method 4: Formal Semi Left Join

```
SELECT
    highway,
    stretch,
    COUNT(1) AS days_closed,
    100 * COUNT(1) / 365 AS pct_closed_365,
    100 * COUNT(1) / 353 AS pct_closed_353
FROM (
    SELECT
        c.highway AS highway,
        c.area AS stretch,
        DATE(c.reported) AS closure
    FROM caltrans c
    WHERE EXISTS (
        SELECT
            1
        FROM caltrans
        WHERE text like '%CLOSED%' AND (
            text LIKE '%FOR THE WINTER%' OR text LIKE '%DUE TO SNOW%')
        ) AND text LIKE '%CLOSED%' AND (text LIKE '%FOR THE WINTER%' OR text LIKE '%DUE TO SNOW%')
        GROUP BY highway, stretch, closure
    ) closures
GROUP BY highway, stretch
ORDER BY pct_closed_365 DESC;

+----+-------------+------------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------+
| id | select_type | table      | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra                                        |
+----+-------------+------------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------+
|  1 | PRIMARY     | <derived2> | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   2395 |   100.00 | Using temporary; Using filesort              |
|  2 | DERIVED     | c          | NULL       | ALL  | PRIMARY       | NULL | NULL    | NULL | 102730 |     2.33 | Using where; Using temporary; Using filesort |
|  3 | SUBQUERY    | caltrans   | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 102730 |     2.33 | Using where                                  |
+----+-------------+------------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------+
```

A few things to note from the MySQL `EXPLAIN` documentation.

1. In the column labeled “rows”, MySQL estimates the number of rows it must scan.
2. ”Filtered” gives us an estimate of the percentage of rows a particular operation will filter and provides the number of rows that will be handed to an outer subquery or a join higher up in the plan.
3. The column “extra” mentions if any special sorting, indexing or temporary tables were constructed.
4. When extra includes a filesort, MySQL tries to determine how to retrieve rows in sorted order. (Chapter 10-12)
5. When extra includes a “using temporary”, MySQL creates a temporary table to store intermediate results (remember, we want these to be as small as possible).
6. When extra includes a “using WHERE” MySQL will use the WHERE predicate to restrict which rows are sent to the next query up in the hierarchy.

Putting this together, we want to find the plan that examines the fewest number of rows in the innermost subqueries or joins, and filters out rows accordingly as we move up the hierarchy. We want to choose a plan that minimizes the use of filesort and temporary tables as they require additional scans over the data. We also want to see more uses of WHERE (filter filter filter).

This yields method 4 (semi left join) as the most efficient.

Something I Did Not Know: Semi- and anti- joins are very commonly cited ways to speed up subqueries and provide “hints” to the optimizer to construct a more efficient plan.

Oracle:
`https://blog.rackspace.com/speeding-queries-semi-joins-anti-joins-oracle-evaluates-exists-not`

`https://docs.oracle.com/cd/E17952_01/mysql-5.6-en/semi-joins.html`

PostgreSQL:
`https://www.credativ.com/credativ-blog/2010/02/postgresql-optimizer-bits-semi-and-anti-joins`

MariaDB (open source fork of MySQL, now owned by Oracle):
`https://mariadb.com/kb/en/library/semi-join-subquery-optimizations/`

Part 3: Join and Optimization Algorithms

Chapter 12 is rather tedious, so I have asked everyone to read this chapter. It is so tedious that it would take multiple lectures for me to address the class and also answer questions. While we have not focused on nitty-gritty math involved in disk block reads (aside from on the exam), it is important to be aware of it as you may be asked in which situation to use which join algorithm, and you will probably want to use this math to justify your response. Remember, data wins arguments.

Exercises

Special thanks to TA Jin for most of these sample responses.

1. Suppose we have two relations L and R. The nested-loop join algorithm is presented below:

```
for each $t_l \in L$:
  for each $t_r \in R$:
    if $t_l.K = t_r.K$ then output tuple $(t_l, t_r)$
```

   If L has 100,000 tuples, how many times is relation R scanned?

   R is scanned once for each tuple in L, so R is scanned 100,000 times. If m is the number of tuples in L and n is the number of tuples in R, then the time complexity of the join is $O(mn) \approx O(n^2)$ if $m \approx n$.

2. The indexed nested-join loop is similar, but instead of doing a linear scan over R, we build an index on it.

```
for each $t_l$ in $L$:
  $X$ = index-lookup($t_l.K$)
  for each $r \in X$:
    output tuple $(t_l, r)$
```

   (a) How many times is each tuple in both L and R scanned?

   This one ended up being tricky for students. The answer depends on whether or not we consider the construction of the index, or if we assume the index is already built.

   If the index has not been constructed, R is scanned exactly once. If the index has already been constructed, R is scanned 0 times. Instead, for each tuple in L, we *probe* the index (probing is $O(1)$ whereas I have been using scan to represent a liner $O(n)$ scan).

   Each tuple in L is scanned exactly once.

(b) In the worst case, what is the number of block transfers required for this join if we assume R is indexed with a B+-tree?

Suppose the cardinality of R and L is $|R|$ and $|L|$ respectively, and the number of blocks they occupied is b_R and b_L. The memory has M blocks. The index occupied N blocks. On average, there are J matching L tuples per an R tuple.

(a) As we need to find all matched tuples using index for each tuple in L, for each tuple in L, we need to visit it once. For tuples in R, as we can reduce the number of visits blocks with the help of index, if there are J matching L tuples per an R tuple on average, for each tuple in R, we need to visit it $\frac{1}{J}$ times on average.

(b) In the worse case, the index is not built on the join key. In this case, the index join is equivalent to nested-loop join. Then the number of block transfers will be $b_L + |L| * b_R$.

If we have the assumption that the index is built on the join key, then the worse case happened when the index cannot fit the main memory i.e. $M < N$. As we need to keep at least 3 blocks to perform join (one for reading blocks from L, one for reading blocks from R and another for output the join results). So there will be $M - 3$ blocks for loading the index. Then the average index look up cost C can be calculated as $C = \frac{M-3}{N} * 0 + \frac{N-M+3}{N} * 1$. And the total cost will be $M - 3 + b_R + |R| * (C + J)$.

3. In terms of disk I/O, which of the three join algorithms is the most efficient: naive nested-join loop, block nested-join loop, indexed nested-join loop and why? The answer probably depends on several factors. Describe these factors and come up with a heuristic describing when you may want to use each.

Suppose we have two tables of tuples R and L, the cardinality of them is $|R|$ and $|L|$ respectively, and the number of blocks they occupied is b_R and b_L. Without of generality, we assume that $|R| < |L|$. The memory has M blocks. The index is built on the join key of table L and occupies N blocks. On average, there are J matching L tuples per an R tuple.

First we compute the cost of each join algorithm: We use R as the outer relation. The total cost of naive nested-loop join will be $b_R + |R| * b_L$. The total cost of block nested-loop join will be $b_R + b_R * b_L$. And that of the index join will be the cost to load index blocks into main memory plus that of the join operation i.e. $b_R + |R| * (C + J)$.

Then we look at different factors: if one relation is small enough to fit the memory, we should use nested-loop join and use this relation as the inner one. If neither relation fits the memory, block nested-loop join will perform better than naive nested-loop join as it will process the relation in block instead of tuple. For index join, if the index is built on the join key and the selectivity of index search condition is high, then it is better to use index join to avoid linear scan on the inner relation.

4. Prove that the following equivalencies hold. Explain how you can apply them to improve the efficiency of certain queries. Feel free to define relations if you would like:

$$E_1 \bowtie_\theta (E_2 - E_3) = (E_1 \bowtie_\theta E_2 - E_1 \bowtie_\theta E_3)$$

One possible example to prove it is as following. Suppose the schema of E_1 is (A, B) and that of E_2 and E_3 is (B, C). And they will be join on the key B where θ is $B > 10$. The tuples in E_1 are (a, 8), (a, 12), (b, 11) and (c, 13). The tuples in E_2 are (5, W), (11, X), (12, Y), (13, Z). The tuples in E_3 are (12, Y), (13, Z), (16, O).

Then for the left hand side, the result of $E_2 - E_3$ is $\{(5, W), (11, X)\}$; and the result of join will be $\{(b, 11, X)\}$. For the right hand side, the results of $E_1 \bowtie_\theta E_2$ and $E_1 \bowtie_\theta E_3$ are $\{(b, 11, X), (a, 12, Y), (c, 13, Z)$ and $\{(a, 12, Y), (c, 13, Z)\}$, respectively. And the set difference between them, i.e. the final result will be $\{(b, 11, X)\}$. Then the result is the same with that of the left hand side.

The basic idea of optimization is to first perform set difference operation and then use the result of it as the right side relation of join. Then in this case, the number of tuples participating in join will be fewer than that of the original order of execution.

Another Possible Solution

Let us rename $(E_1 \bowtie_\theta (E_2 - E_3))$ as R_1, $(E_1 \bowtie_\theta E_2)$ as R_2 and $(E_1 \bowtie_\theta E_3)$ as R_3. It is clear that if a tuple t belongs to R_1, it will also belong to R_2. If a tuple t belongs to R_3, $t[E_3\text{s attributes}]$ will belong to E_3, hence t cannot belong to R_1. From these two we can say that $\forall t, t \in R_1 \Rightarrow t \in (R_2 R_3)$. It is clear that if a tuple t belongs to $R_2 R_3$, then $t[R_2\text{s attributes}] \in E_2$ and $t[R_2\text{s attributes}] \notin E_3$. Therefore: $\forall t, t \in (R_2 R_3) \Rightarrow t \in R_1$. The above two equations imply the given equivalence. This equivalence is helpful because evaluation of the right hand side join will produce many tuples which will finally be removed from the result.

5. For the pair of expressions, give instances of relations that show the expressions are not equivalent.

$$\Pi_A (R - S) \text{ and } \Pi_A(R) - \Pi_A(S)$$

Suppose the schema of R and S is (A, B). And R has tuple (1, a) and S has tuple (1,b). Then the result of $\prod_A(R - S)$ will be “1” and that of $\prod_A(R) - \prod_A(S)$ will be $\emptyset$.

Another Possible Solution

$$R = \{(1, 2)\}, S = \{(1, 3)\}$$

The result of the left hand side expression is $\{(1)\}$, whereas the result of the right hand side expression is empty.