Q1. Compare and contrast papers.

i. [point 15] Compare and contrast **buffer replacement policies** in the following papers:

   (1) **T1.2**, J.M. Hellerstein and M. Stonebraker, *Anatomy of a Database System*
   
   (2) **T3.2**, M. Stonebraker, *Operating System Support for Database Management*

   **Answer:**
   
   - **Compare:**
     
     - Both papers criticized LRU by identifying situations where it was not effective. For example, T3.2 listed four different database access types and explained that only one case (i.e., random access with non-zero probability of reference) works well with LRU.
   
   - **Contrast:**
     
     - T1.2 provided a broader view of DBMS architecture and discussed alternatives such as LRU-2 and CLOCK. T3.2 focused on Linux and Ingres to discuss the disadvantages of LRU.
     
     - T1.2 discussed double buffering.
     
     - T3.2 discussed pre-fetch algorithm, crash recovery, and paged virtual memory.

ii. [point 15] Compare and contrast **parallel query processing** in the following papers:

   (1) **T1.2**, J.M. Hellerstein and M. Stonebraker, *Anatomy of a Database System*
   
   (2) **T2.3**, D. DeWitt and J. Gray, *Parallel Database Systems: The Future of High Performance Database Systems*

   **Answer:**
   
   - **Compare:**
     
     - Both papers mentioned three popular parallel models (i.e., shared-nothing, shared-memory, and shared-disk) and discussed the pros and cons of each.
     
     - Both papers discussed parallel query processing mainly in the context of shared-nothing architecture.
     
     - Both papers discussed horizontal partitioning and its effects in queries.
     
     - Both papers used expert opinion style validation.
   
   - **Contrast:**
     
     - T1.2 gave an overview of structures for parallel query processing, but did not provide implementation details. The paper also talked little bit about NUMA architecture. T2.3 gives implementation details such as data partitioning techniques (e.g., range, round-robin, and hashing) and parallel relational operators.
     
     - T2.3 discussed pipeline parallelism as well as partitioned parallelism.
Q2. Consider the following paper: (T3.2) M. Stonebraker, Operating System Support for Database Management

i. [point 10] As was seen in the paper, DBMSs often must re-implement many similar services that an OS typically provides. Does it make sense to integrate the DBMS service support into the OS and what would be the advantages or disadvantages? (Limit your words 200 words)

Answer:

- Advantages:
  - DMBS and OS require similar services such as memory management, file system, and consistency control. Integration will make DBMS implementation easy by reducing many redundant functionalities.
  - Closer integration will speed up the performance of DBMS.

- Disadvantages:
  - As the functionalities of DBMS is included in OS, the services running on OS become larger and negatively impact other application programs.
  - OS functionalities tailored to DBMS may not be suitable for other types of application programs.
  - It will limit DMBS design by what OS provides.
ii. Suppose an RDBMS contains the following two tables, where primary keys are in italics:

Factories(FactoryNumber, FactoryName, FactoryAddress)
Employees(EmployeeNumber, EmployeeName, FactoryNumber)

Suppose Employees occupies a large number of disk blocks and only a small percentage of these blocks can fit into main memory, but Factories can fit into one main memory page.

Suppose a user executes the following query:

```
SELECT Factories.FactoryName, Employees.EmployeeName
FROM Factories, Employees
WHERE Employees.FactoryNumber = Factories.FactoryNumber
```

Assuming the RDBMS uses a nested-loop to execute the query, explain why invoking an LRU at each page fault would not be an optimal page-replacement algorithm.

(FYI, A nested loop is a loop within a loop, an inner loop within the body of an outer one. How this works is that the first pass of the outer loop triggers the inner loop, which executes to completion. Then the second pass of the outer loop triggers the inner loop again. This repeats until the outer loop finishes.)
Assumptions are needed to solve this problem. DBMS is an application program fetching records, not pages. OS manages buffer and disk i/o. Memory requirement for private variables f and e is not prohibitive. Factories table is accessed in outer loop (since it fits in main memory) and Employees table is accessed in inner loop in a naïve way (i.e. record at a time) as follows:

For each record \( f \) in Factories  
    For each record \( e \) in Employees  
        If \( f.FactoryNumber = e.FactoryNumber \) Then  
            Save \( f.FactoryName, e.EmployeeName \) in a resultset  
        End If  
    End For  
End For

Record-based algorithms for nested-loop with request the first “Factories” record followed by various “Employees” records. OS will fetch the first data page of “Factories” table and copy the first record to the private memory of the DBMS as shown in the first row of Table 1. Then OS will fetch various data pages of “Employees” table coping records one at a time to the private memory of DBMS as shown in next few rows of Table 1 and Figure 1(a). Since the “Employees” table is larger than the space available in the main memory buffers, this step will require buffer replacement. The first page of “Factories” table is likely to be replaced by LRU since it is never touched by OS after copying the first record in the private memory of DBMS. Thus the buffer containing the first page of “Factories” table will be used to fetch the next page of “Employees” table (see Figure 1(b)). However, we may need to bring the first page of “Factories” table back in main memory once the “Employees” table is scanned and DBMS requests second record of “Factories” table from the outer loop (see Figure 1(d)). Thus, shows that LRU leads to unnecessary page-faults.

You may also notice that LRU ends up using a large number of memory buffers for “Employees” table, even though allocating just one buffer for the table in inner loop may be adequate for Nested loop ignoring the difference between sequential and random disk accesses. If you assume a DBMS controls buffers and uses page level algorithm, the behavior of LRU will be different.

<table>
<thead>
<tr>
<th>DBMS Request</th>
<th>OS Action</th>
</tr>
</thead>
</table>
| A first record of Factories to its private memory | A’ a. fetch first page of Factories from disk to main memory buffer  
b. copies first record of Factories from main memory buffer to private memory space of DBMS application program |
| B next records of Employees, one at a time to its private memory | B’ a. fetch next page of Employees if needed  
b. copies next record of Employees to private space of DMBS application. |
| C compares first record of factory with the record in private memory. writes results if needed | C’ copies result record from private space of DBMS to a memory buffer |
| Repeats B & C till all records of Employees are exhausted | repeats B’ till buffer is full  
(assuming Employees > buffer space) |
| Now, LRU buffer is the one holding first page of Factories. Next fetching of Employees data page will put Factories data page out of memory. | |

Table 1. Trace of DBMS request and OS action to process the query in Q2 (ii).
The following small example illustrates the situation of page eviction of Factories table data in memory buffer. We assume that the Factories table has 4 records and fits in a page block. The size of Employees table is 4 blocks. The size of main memory buffer is 3 blocks. To compare the first record of Factories with records in Employees, OS fetches the page of Factories table from disk to main memory buffer ((a) in Figure 1). The remaining two empty blocks of memory buffer will be filled with first two pages of Employees. As the inner loop reaches the end of page 2, third block of Employees will be fetched from disk and replace the first page block occupied by Factories ((b) in Figure 1) because Factories is the least recently used page. The evicted Factories data will cause page fault. The similar situation will continuously occur till the outer loop is finished.

Figure 1. An example of page fault situation during the query process in Q2 (ii).
Q3. Consider the following paper: (T2.1) P.Griffiths Selinger, M.M. Astrahan, D.D. Chamberlain, R.A. Lorie and T.G. Price, Access Path Selection in Relational Database Management System

i. [point 5] In System R, a user need not know how the tuples are physically stored and what access paths are available. The optimizer chooses the one which minimizes "total access cost" for performing the entire predicates (SQL). But, there are some possibilities that the optimizer can choose a wrong one. What kind of aspects can affect the wrong choice?

Answer:
• The statistics in system catalog do not always reflect the actual usage of tables.
• The computation of join cost is based on selectivity and cardinality, assuming uniform distribution, predicate independence and join independence. These assumptions may not be true.
• The condition of execution environments (e.g., buffers) is not modeled.

ii. [point 5] Explain the difference between join and Cartesian products. According to the optimization strategy in the paper, joins are ordered to postpone Cartesian products. Explain why it is reasonable.

Answer:
• Join has predicates, but Cartesian product has no predicate. Thus, Cartesian product contains m * n rows, where m is the number of rows in the first table and n is the number of rows in the second table. This product is the set of all possible combinations.
• The purpose of postponing Cartesian product is to keep the working set of tuples as small as possible throughout query execution. As the working set is small, the data delivery inside query tree is much more efficient.
Q4. Read the following paper: Encapsulation of Parallelism in the Volcano Query Processing System (pp 155 - 164 in textbook)

i. [point 10] What research contributions does each paper make?

Answer:
• The operator described in the paper encapsulates parallelism. As a result, the implementation is much easier.
• The paper proposed uniform and flexible interface. Thus, the operators can be inserted anywhere in an execution tree and combined with any operator. It resulted in better extensibility.
• Introduction of two new parallel query processing approaches.
• First open literature regarding parallel query operators.

ii. [point 10] What validation methodologies were used to support the claims in the paper?

Answer: Computer simulation, experimental evaluation, working prototype.

iii. [point 10] Compare and contrast parallel query processing in the given paper with the following paper: (2) T2.3, D. DeWitt and J. Gray, Parallel Database Systems: The Future of High Performance Database Systems

Answer:
• Compare: Both papers discussed the parallelism in join operations and data partitioning.
• Contrast:
  o T2.3 claimed that shared-nothing was most efficient architecture. In this paper, Volcano system was implemented on a shared-memory architecture.
  o T2.3 discussed parallelism at architecture model level. Volcano did not.
  o T2.3 discussed data partitioning and allocation mechanisms, but Volcano did not.
  o The given paper discussed vertical parallelism, but T2.3 did not.

iv. [point 10] What assumptions have been made in each paper? List 2 to 3 assumptions and criticize those assumptions with respect to its realism.

Answer:
• For intermediate results, Volcano uses virtual devices. The operators are not affected by the use of virtual devices, and can be programmed as if all input comes from a disk-resident file and output is written to a disk file. ➔ The volatile property of virtual devices may affect the operators.
• In the performance test, authors assumed that the overhead can be kept in acceptable limits. ➔ They may discuss the side effects when the overhead is not acceptable.
• Procedure calls is more efficient than system calls for query processing operations. ➔ This claim is mostly true, but needs more backup information.