DI- FCT/UNL **November 8th, 2025**

Database Systems

Test-1 2025/26

Duration: 2 hours (limited information)

Group 1 (each question is worth 2,5 values out of 20)

Consider part of a database for a supermarket, with a single store, consisting of six tables:

Employee(emp_id, name, position) Supplier(supplier_id, supplier_name, contact_phone)

Product(product_id, product_name, price, supplier_id) Customer(customer_id, name, phone)

Sale(sale_id, sale_date, customer_id, emp_id). Sale_Item(sale_id, product_id, quantity)

These tables capture staff, suppliers, products, customers, completed sales, and detailed sale line-items for each transaction in the supermarket. The primary keys are emp_id for Employee, supplier_id for Supplier, product_id for Product, customer_id for Customer, and sale_id for Sale. Sale_Item uses a composite primary key (sale_id, product_id). Regarding Foreign keys, Product.supplier_id references Supplier, Sale references Customer and Employee, and Sale_Item references both Sale and Product. This ensures that every product has a valid supplier, every sale has a valid customer and employee, and every sale item belongs to an existing sale and product.

The adopted DBMS uses blocks of 8KiB (8192 bytes). The records of Sale and Sale_Item have fixed length, while the other tables have variable length size. Estimated cardinalities, sizes, and 8-KiB block counts are: Employee with about 100 rows (90 bytes per tuple/2 blocks), Supplier with about 50 rows (72 bytes/1 block), Product with about 5,000 rows (70 bytes/43 blocks), Customer with about 20,000 rows (72 bytes/176 blocks), Sale with about 500,000 rows (20 bytes/1,221 blocks), and Sale_Item with about 2,000,000 rows (16 bytes/3,907 blocks). For each of these tables there is a secondary B+ tree (non-clustering) index on the primary key attribute(s), created with the column order indicated in the tables.

A B+ tree node can contain about 500 search keys, and it is known that seek time t_S is 9ms and the transfer time of a block t_T is 1ms, while the memory only holds 100 blocks.

Note: In this group, whenever examples are requested, these must be exclusively about this database, Additionally, all the answers must contain a brief justification.

1 a) Present two execution plans for the following SQL query, briefly justifying which of the plans has the least cost in the given database.

```
SELECT *
FROM Customer NATURAL JOIN Sale
```

1 b) Present a query for which the use of clustered file organization would be helpful.

1 c) Indicate whether in your opinion the SQL query below can benefit from the existing indices, assuming that are just a few products with price greater than 10000€?
Would you introduce extra indices for optimizing the query?

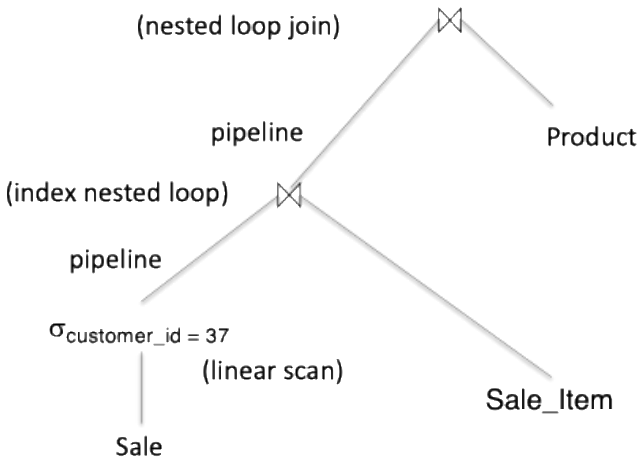
```
SELECT *
FROM Product NATURAL INNER JOIN Sale_Item NATURAL INNER JOIN Sale
WHERE price > 10000
```

1 d) Explain if a bitmap index could improve the linear scan execution of the following query:

```
SELECT COUNT(*)
FROM Product
WHERE supplier_id IN (3,28,45,2)
```

(empty page – any content will not be graded)

1 e) Consider the execution plan presented in the figure. Determine the cost of this plan knowing that in average a customer has 25 sales, each with 4 products. Assume that the height of B+-tree for the non-clustering index of Sale_Item is 3.



Group 2 (each question is worth 2,5 values out of 20)

Note: The response to each of the items in this group cannot, under any circumstances, exceed one page.

2 a) Explain why it is recommended to use surrogate keys in multidimensional models rather than relying on operational production codes or natural keys.

2 b) Describe what are the usual strategies to perform bulk loading of data of a large table with a primary key implemented as a B+-tree secondary index, and why they are usually much better than insertion in random order.

2 c) In the index nested-loop join algorithm, for each tuple $\mathbf{t_r}$ in the outer relation **r**, the index is used to look up tuples in the inner relation **s** that satisfy the join condition with tuple $\mathbf{t_r}$. The complexity of the algorithm is given by the expression: $b_r (t_T + t_S) + n_r * c$.

Derive the previous cost formula, where n_r is the number of tuples the outer relation (stored in b_r blocks of disk) and c is the cost of traversing index and fetching all matching s tuples for one tuple or **r**, assuming just 1 block of available memory. If b_b is the blocking unit for the outer relation, justify why the cost becomes $b_r (t_T + (t_S / b_b)) + n_r * c$, assuming that memory has at least $b_b + 1$ blocks. Values t_T and t_S are transfer time and seek time, respectively.

THE END

(SCRAP PAPER)