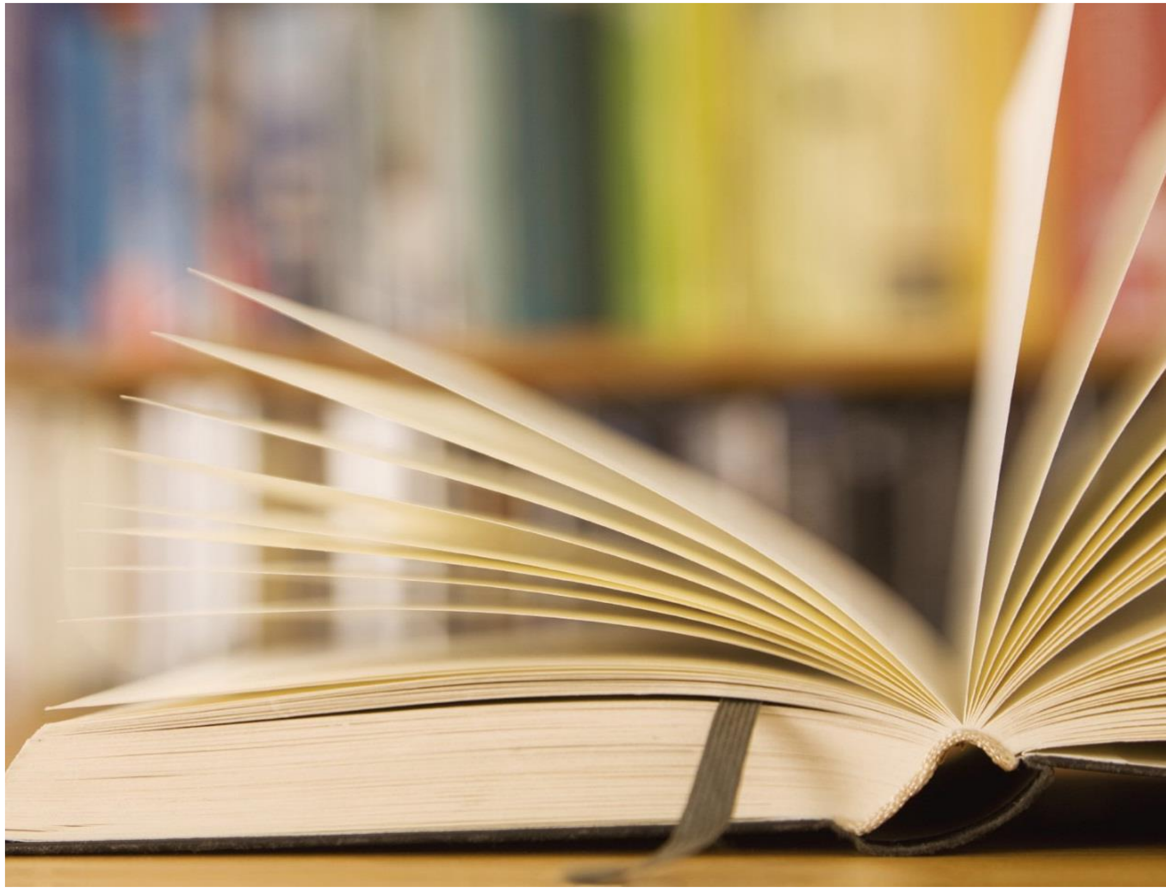
CSCI317 – Database Performance Tuning

Past Year Examination (Sample Solution)

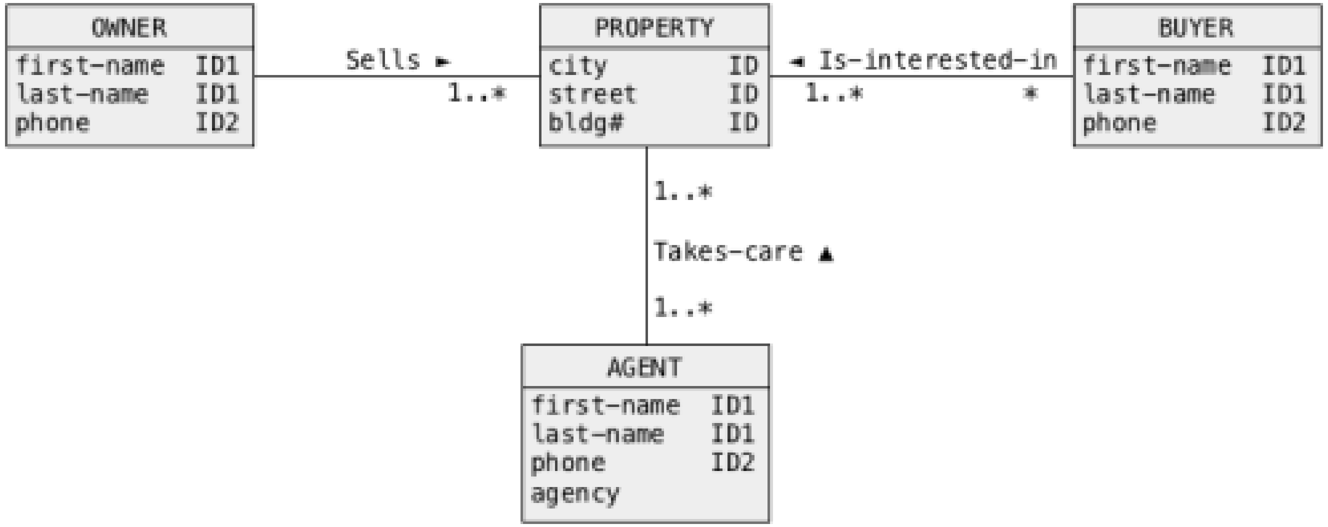
2018 Autumn Session

23 February 2019



Question 1

The following conceptual schema represents a database domain where owners sell real estate properties, buyers are interested in real estate properties and sellers take care about real estate properties.



Question 1

An objective of this task is to use denormalization, appropriate implementation of generalization, and decomposition of classes of objects to improve the performance of the following class of applications.

> Find the full names of real estate owners (attributes first-name and last-name in a class OWNER) who sell a property located in a given city (attribute city in a class PROPDERTY) and being taken care about by an agent from a given agency (attribute agency in a class AGENT)

A sample application that belongs to a class described above could be the following.

> Find the full names of real estate owners who sell a property located in Sydney and being taken care about by an agent from an agency Real Estate Demolishers.

Question 1

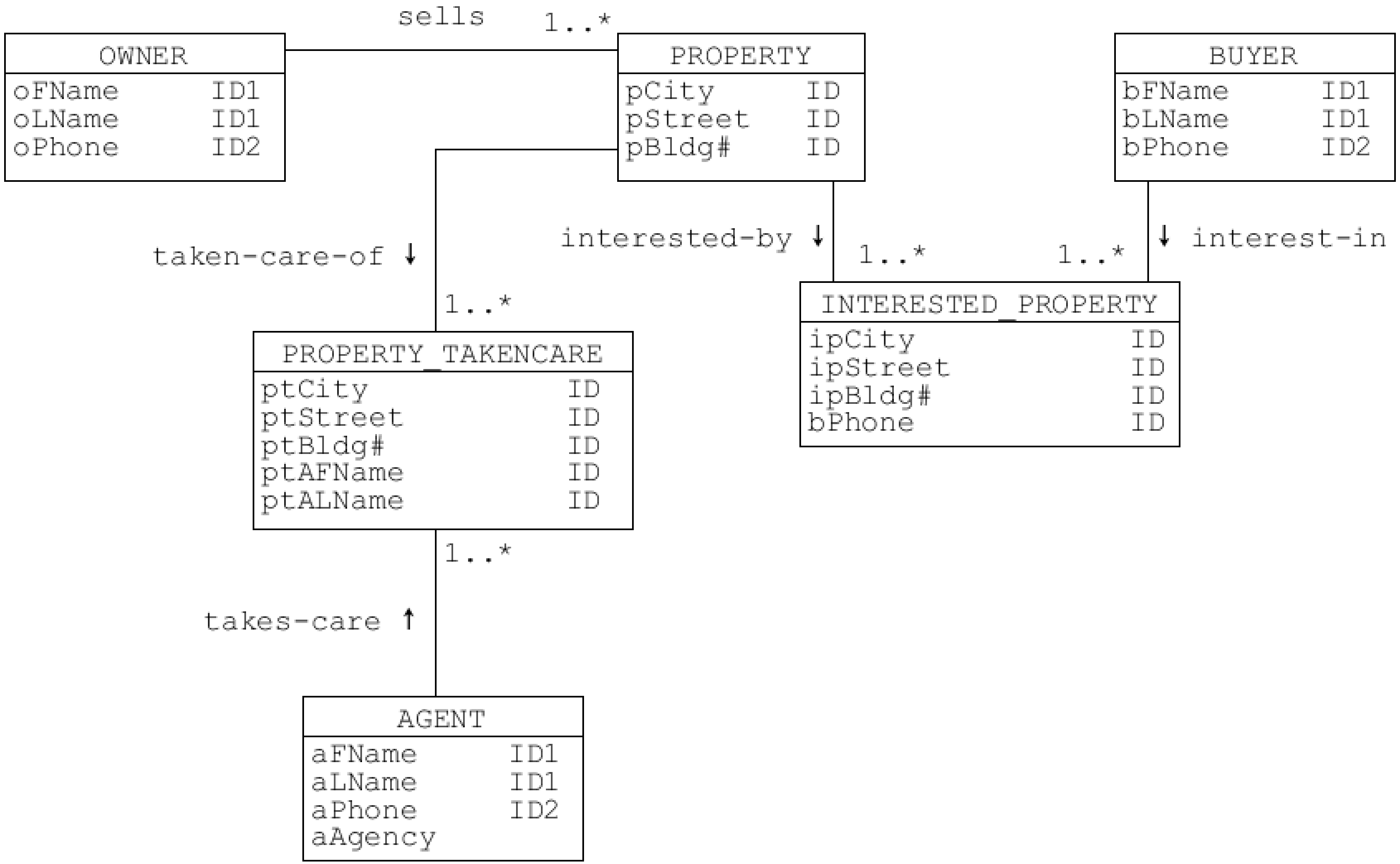
1) Perform simplification of a conceptual schema given above and redraw a simplified schema. (2 marks)
2) To improve performance of a class of database applications given above, denormalize a conceptual schema obtained in step (1) and redraw a denormalized schema. (3 marks)
3) To further improve performance, apply appropriate decompositions and redraw the final conceptual schema. (3 marks)

Question 1

To simplify a conceptual schema, we need to:

- Elimination of generalization
- Elimination of multi-value attributes
- Simplifications
  - Elimination of association classes
  - Elimination of link attributes
  - Elimination of many-to-many associations
  - Elimination of qualified associations

Question 1.1: Simplification



Question 1

To denormalize a conceptual schema, we need to:

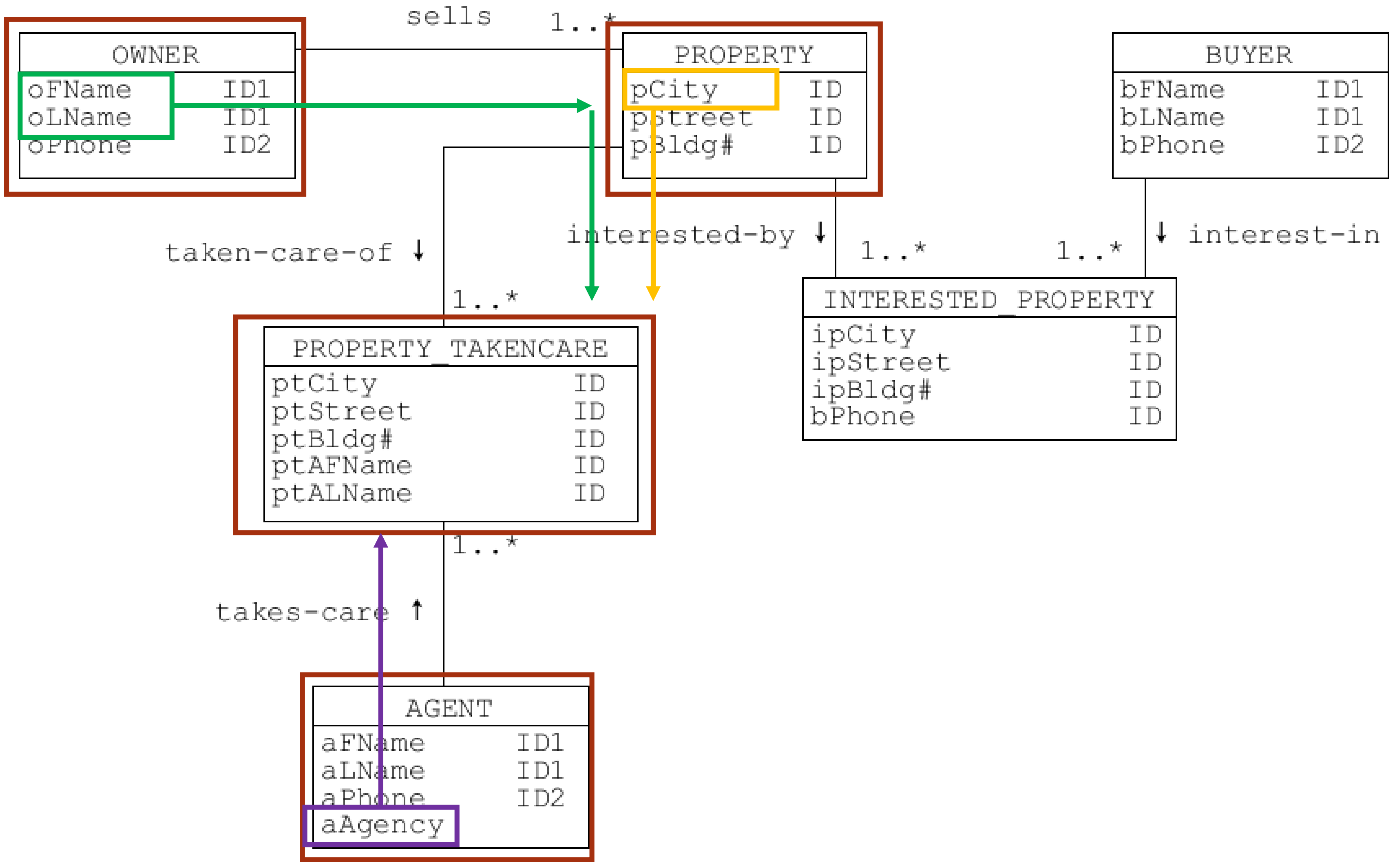
- Identify the attributes that are required to answer the application queries.
- Decompose/fragment the relational table(s) if needed.
- Copy/duplicate the required attributes to a table or closer together to avoid the need to join tables.
    - If need to copy/duplicate attributes, attributes can be copied between tables only if the two tables have one-to-one or one-to-many relationship.
        - The attributes can be copied in the direction from one to many if the association is one-to-many.
        - The attribute can be copied in either direction if the association is one-to-one.

Question 1

- The transaction query for this example is “Find the full names of real estate owners who sell a property located in Sydney and being taken care about by an agent from an agency Real Estate Demolishers.”

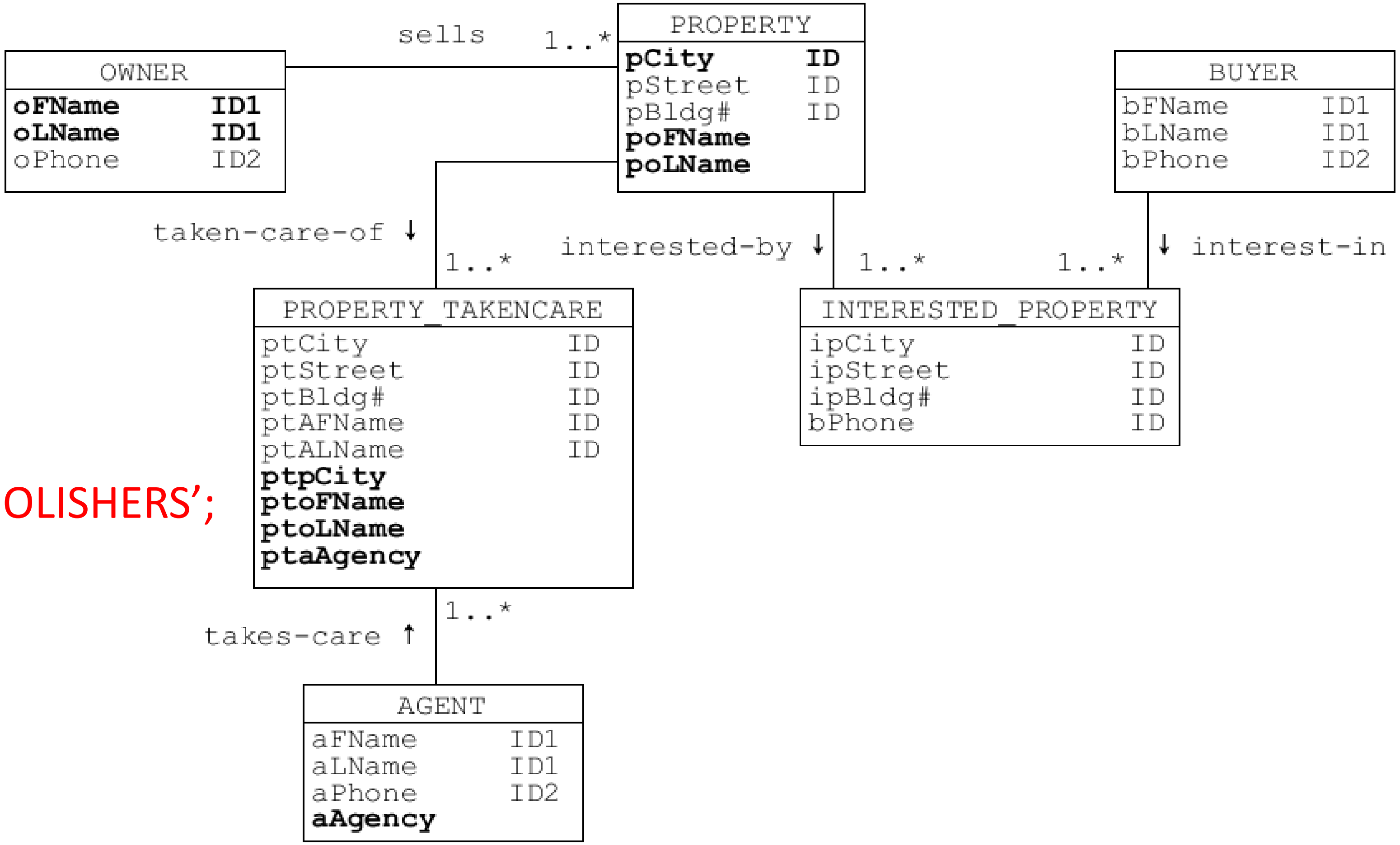
```
SELECT      O.FIRSTNAME, O.LASTNAME
FROM        OWNER O JOIN PROPERTY P
            ON O.PHONE = P.OPHONE
WHERE       P.CITY = ‘SYDNEY’
AND         (P.CITY, P.STREET, P.BLDG#) IN (
            SELECT      PTCITY, PTSTREET, PTBULDG#
            FROM        PROPERTYTAKENCARE PT JOIN AGENT A
                        ON PT.PTAPHONE = A.PHONE
            WHERE       A.AGENCY = ‘REAL ESTATE DEMOLISHERS’);
```

Question 1.2: Denormalization

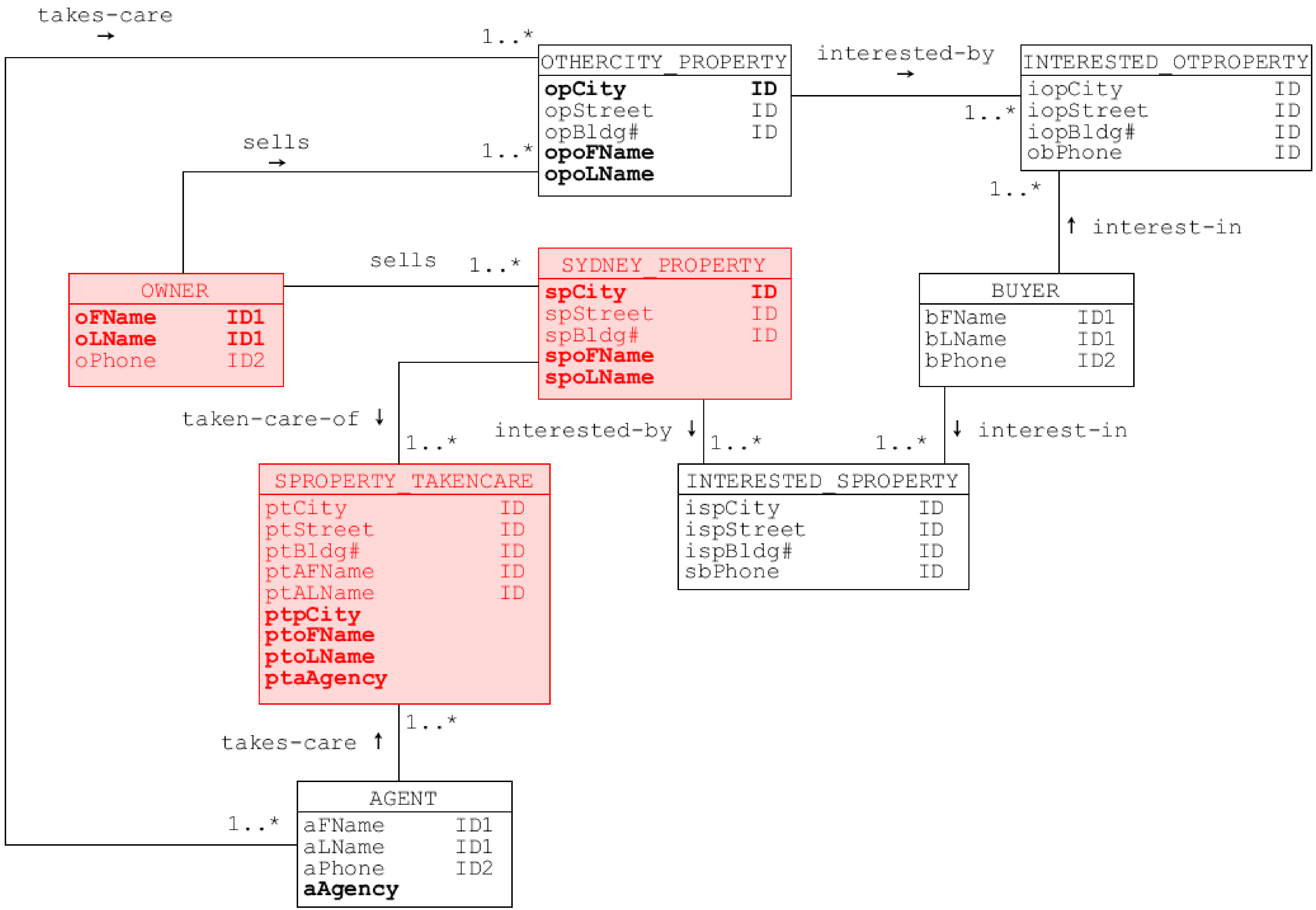


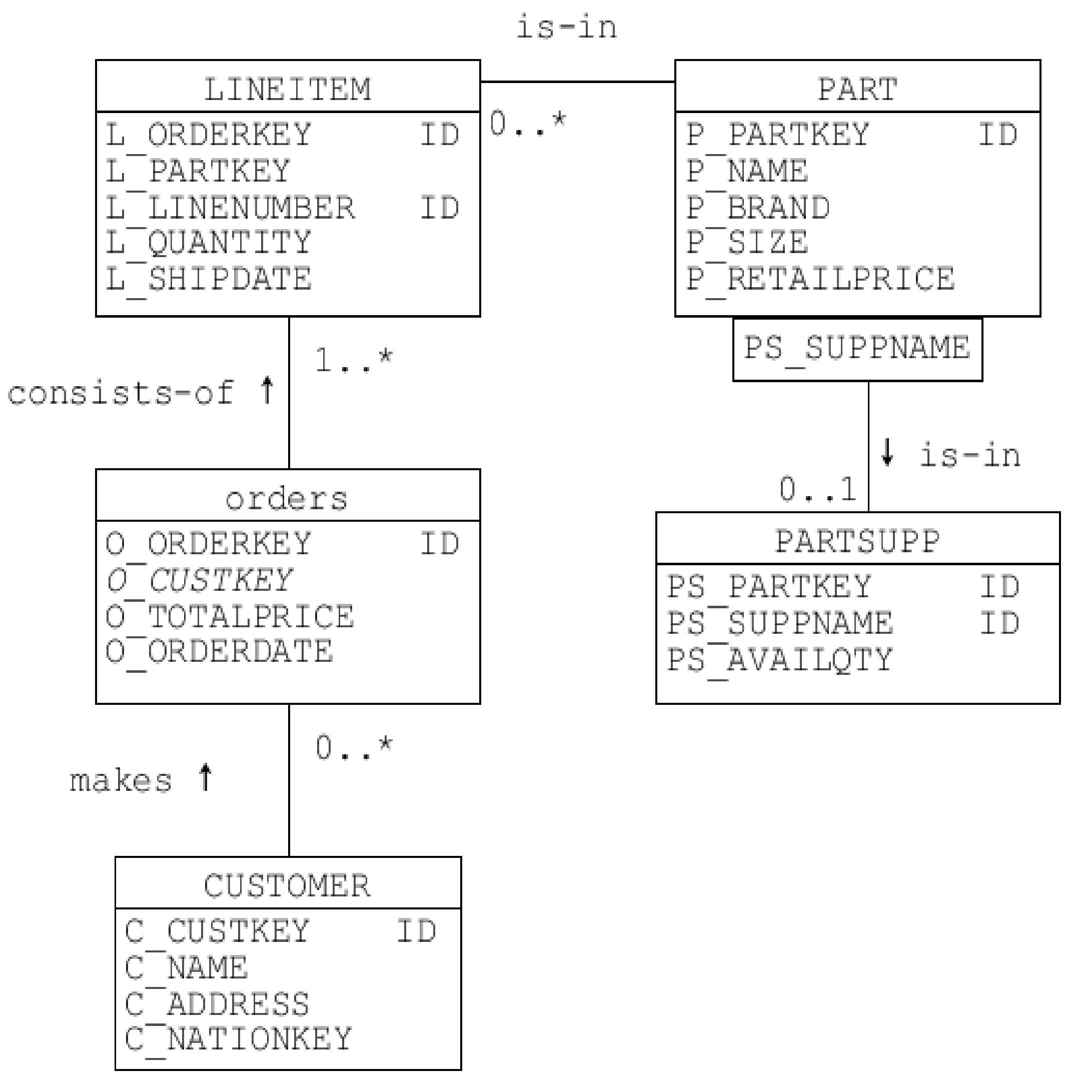
Question 1.2: Denormalization

```
SELECT  ptoFName, ptoLName
FROM    PROPERTY_TAKENCARE
WHERE   ptpCity = 'SYDNEY'
AND     ptaAgency = 'REAL ESTATE DEMOLISHERS';
```



Question 1.3: Decomposition





The questions 2, 4, 5, 6, and 7 of the examination paper are related to the simplified version of TPC-HR benchmark database used in the laboratory classes.

Assume that, the relational tables listed here occupy the following amounts of disk storage:

CUSTOMER	100 Mbytes
PART	30 Mbytes
PARTSUPP	400 Mbytes
ORDERS	500 Mbytes
LINEITEM	900 Mbytes

Question 2

For each one of SELECT statements listed below find an index that speeds up the processing of a statement in the best possible way. Note, that an index must be created separately for each one of SELECT statements. Use CREATE INDEX statement to create the indexes.

1. ```
   SELECT   P_NAME, P_BRAND
   FROM     PART ORDER BY P_SIZE;
   ```
2. ```
   SELECT (SELECT COUNT( DISTINCT O_TOTALPRICE )
    FROM ORDERS) PTOTAL,
    (SELECT COUNT( DISTINCT O_ORDERDATE)
    FROM ORDERS) DTOTAL
    FROM DUAL;
   ```

Question 2

3. SELECT O_TOTALPRICE
   FROM ORDERS
   WHERE O_ORDERDATE = '10-JAN-2017' AND
   O_CUSTKEY = 200;

4. SELECT P_NAME, AVG(P_SIZE)
   FROM PART
   WHERE P_BRAND = 'RUBBISH'
   GROUP BY P_NAME;

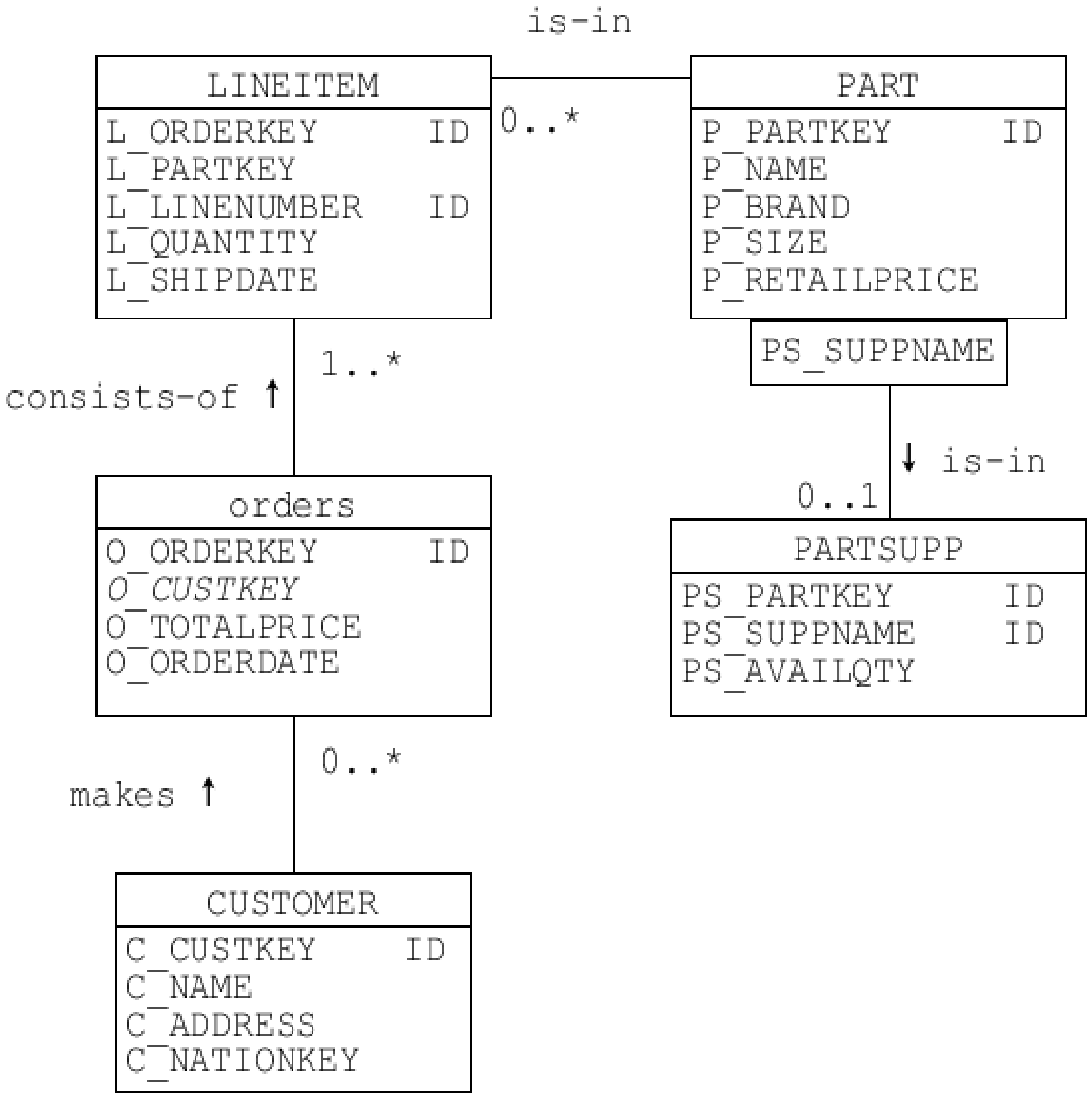
5. SELECT P_NAME, P_BRAND
   FROM PART
   WHERE P_NAME LIKE 'N%'
   INTERSECT
   SELECT P_NAME, P_BRAND
   FROM PART
   WHERE P_BRAND LIKE 'B%';

Question 2

1. SELECT P_NAME, P_BRAND
   FROM PART ORDER BY P_SIZE;

Since P_NAME, P_BRAND, and P_SIZE are not part of any index, the query processor will perform a full-table scan to retrieve the required information. To speed up the retrieval process, we can perform a vertical partition of the table PART by creating the following index.

```
create index partSizeIdx on
PART(p_size, p_name, p_brand);
```

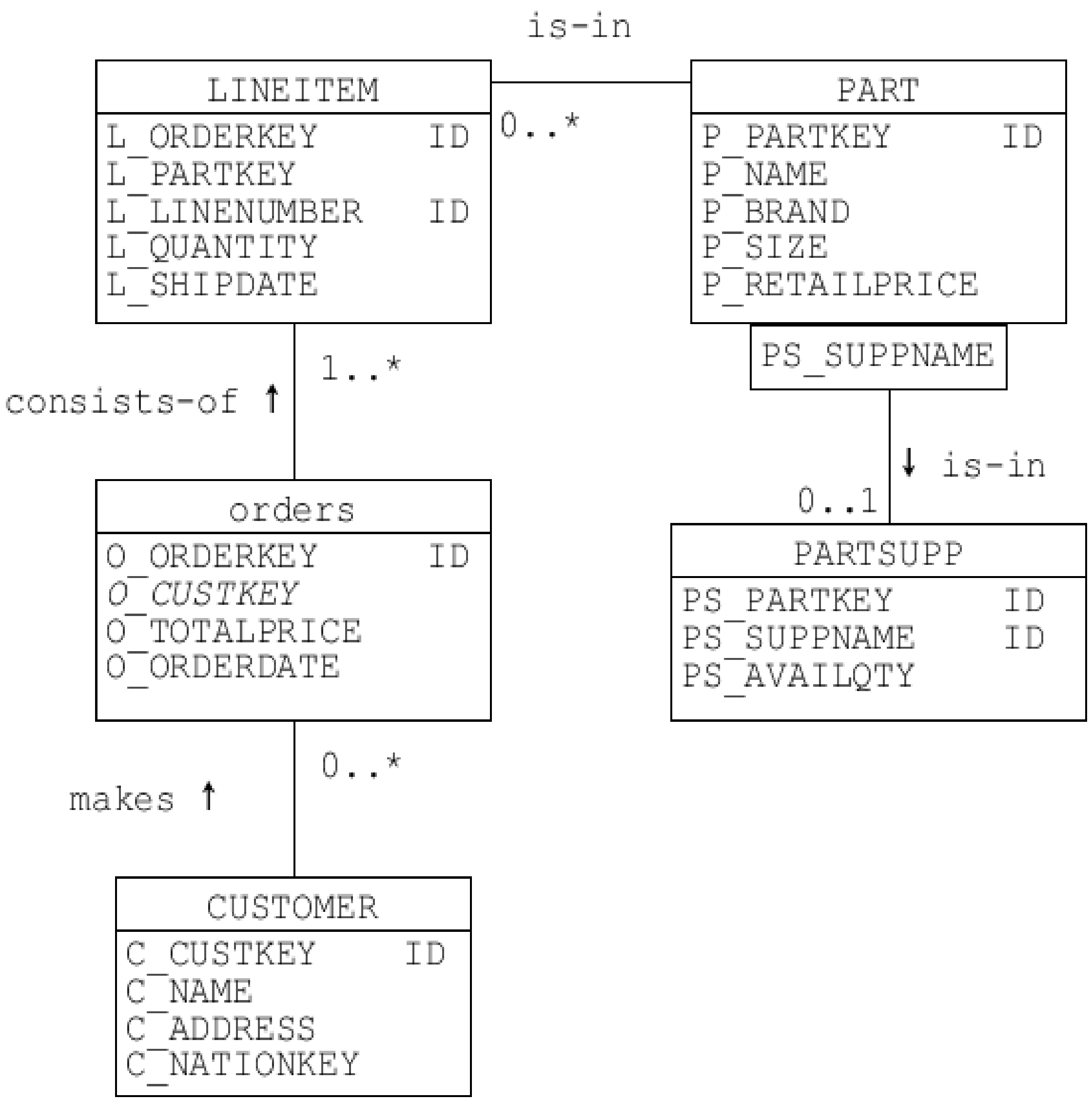


Question 2

2. SELECT (SELECT COUNT(DISTINCT O_TOTALPRICE)
FROM ORDERS) PTOTAL,
(SELECT COUNT(DISTINCT O_ORDERDATE)
FROM ORDERS) DTOTAL
FROM DUAL;

Since both the attributes o_totalprice and o_orderdate are not found in any of the indexes, the query processor will perform a full-table scan on the table ORDERS. Retrieval can be improved if the following index is created:

create index ordersIdx1 on ORDERS(o_orderdate, o_totalprice);

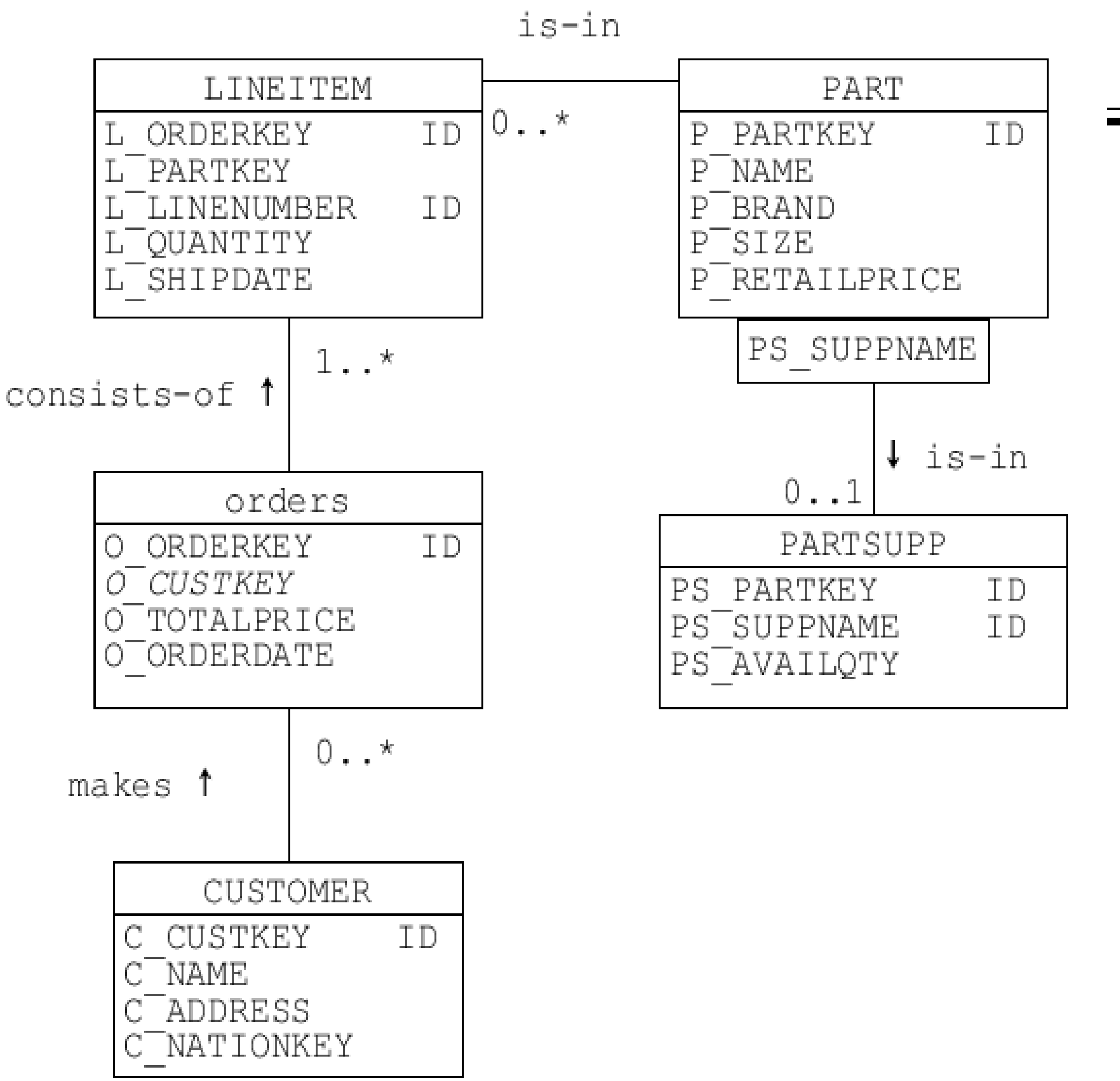


Question 2

3. SELECT O_TOTALPRICE
   FROM ORDERS
   WHERE O_ORDERDATE = '10-JAN-2017'
   AND O_CUSTKEY = 200;

Since the attributes o_totalprice, o_orderdate, and o_custkey are not available in any of the index, the query process will perform a full-table scan on ORDERS table to retrieve the required information. To speed up the retrieval process, the following index can be created:

```
create index ordersIdx3 on
ORDERS(o_orderdate, o_custkey,
o_totalprice);
```

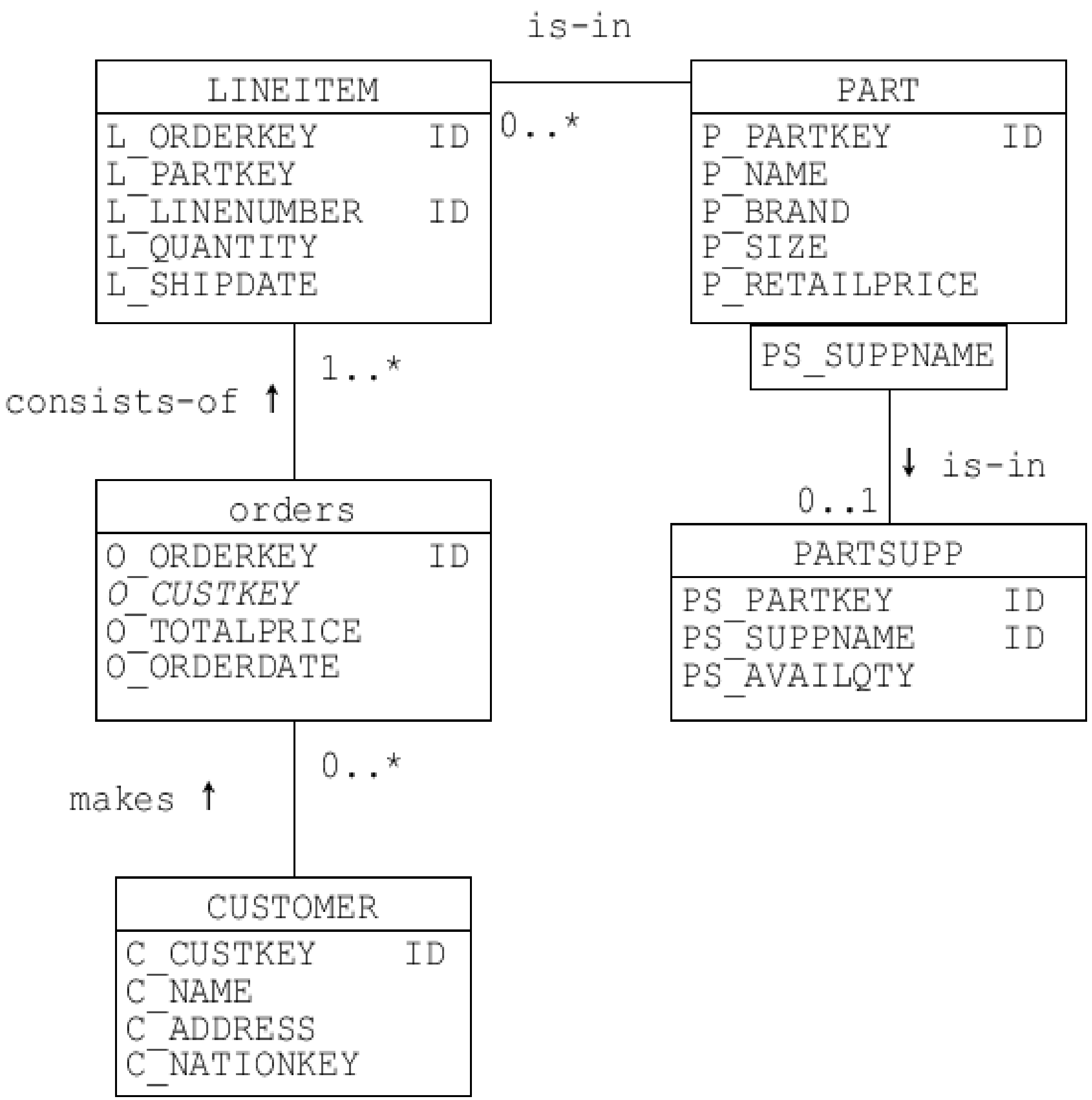


Question 2

4. SELECT P_NAME, AVG(P_SIZE)
   FROM PART
   WHERE P_BRAND = 'RUBBISH'
   GROUP BY P_NAME;

Since p_grand, p_name, and p_size cannot be found in any of the indexes, the query processor will perform a full-table scan on the relational table PART. To speed up the retrieval process, the following index can be created:

```
create index ordersIdx4 on
PART(p_brand, p_name, p_size);
```

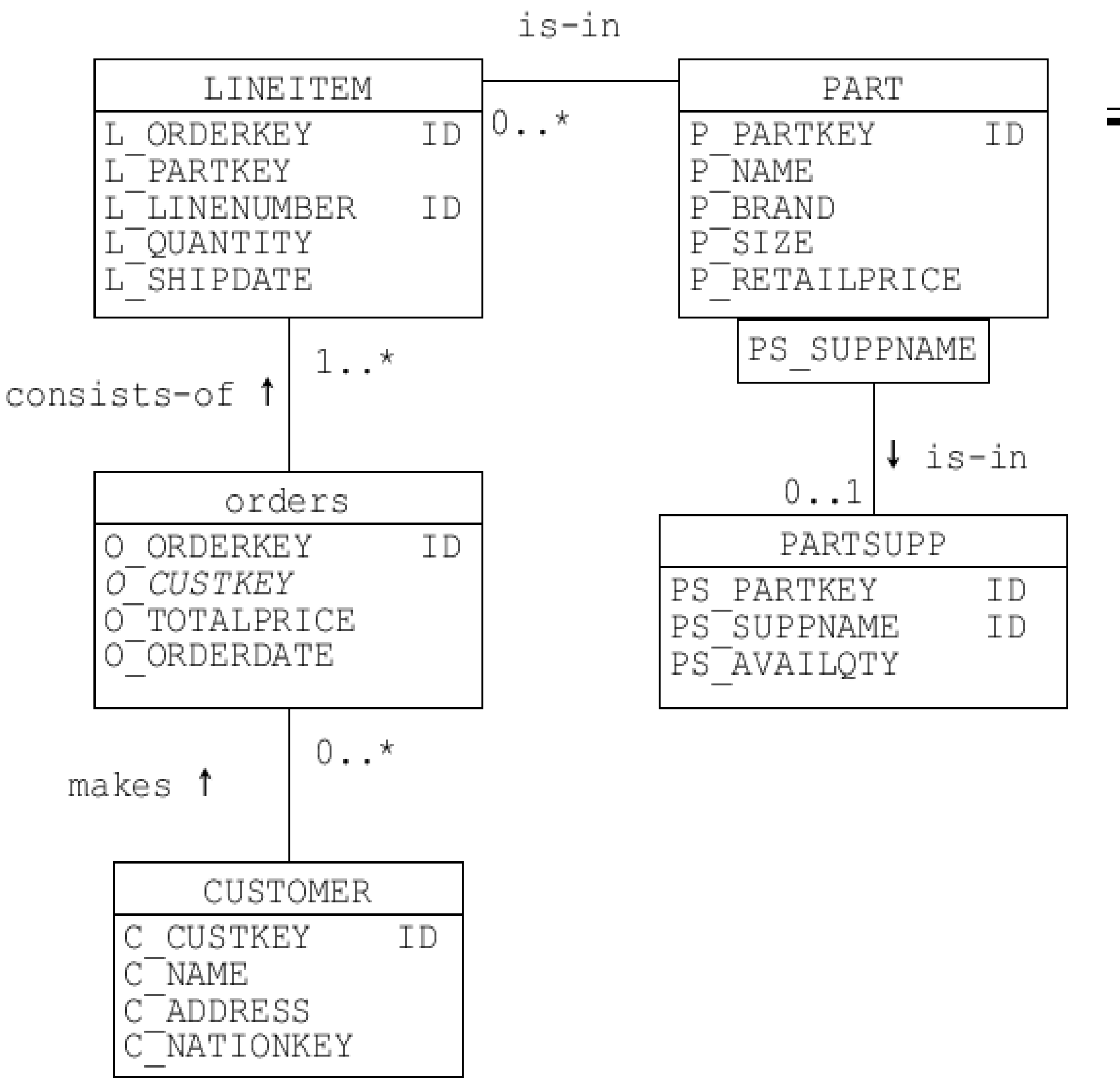


Question 2

5. SELECT P_NAME, P_BRAND
   FROM PART
   WHERE P_NAME LIKE 'N%'
   INTERSECT
   SELECT P_NAME, P_BRAND
   FROM PART
   WHERE P_BRAND LIKE 'B%';

Since both p_name and p_brand cannot be found in any of the indexes the table has, the query processor will perform a full-table scan to retrieve the required information. To speed up the retrieval process, the following index can be created.:

```
create index ordersIdx5 on
PART(p_brand, p_name);
```



Question 3

A relational table:

BASKETS(bid, product, quantity, sname, recorded)

Contains information about the contents of customer baskets.

The physical parameters of the relational table BASKETS are as follows:

1) the total number of rows in the table is 10^5
2) blocking factor is equal to 10 rows per block,

Question 3

3) The attributes `bid` and `product` form a composite primary key, primary keys are always indexed, the height of B*-Tree index on `(bid, product)` is equal to 3, the total number of blocks at a leaf level of an index on (bid, product) is equal to 5×10^2,

4) the total numbers of distinct values in the columns of the table `BASKETS` are as follows:

Bid	Products	Quantity	Sname	Recorded
10^5	10^3	3×10^3	10	3×10^2

5) a database designer implemented an index on an attribute `sname`, the height of B*-Tree index on `sname` is equal to 2.

6) The total number of data blocks at a leaf level of an index on `sname` is equal to 10^2.

Question 3

Find the optimal query execution plans for each one of the queries listed above and estimate how many read block operations are needed to implement each one of the plans, i.e. estimate the total number read block operations needed to compute each one of the queries. Show your calculations. Express the query execution plans as the short stories about how the system plans to compute the queries given below. A solution of each one of the cases listed below is worth 2 marks.

1)

```
SELECT COUNT(DISTINCT product)
FROM BASKETS;
```

Question 3

2)

```
SELECT MAX(quantity)
FROM BASKETS;
```

3)

```
SELECT quantity
FROM BASKETS
WHERE sname = ‘Golden Bolts Pty Ltd’ AND
        recorded = ‘12-DEC-2012’;
```

Question 3

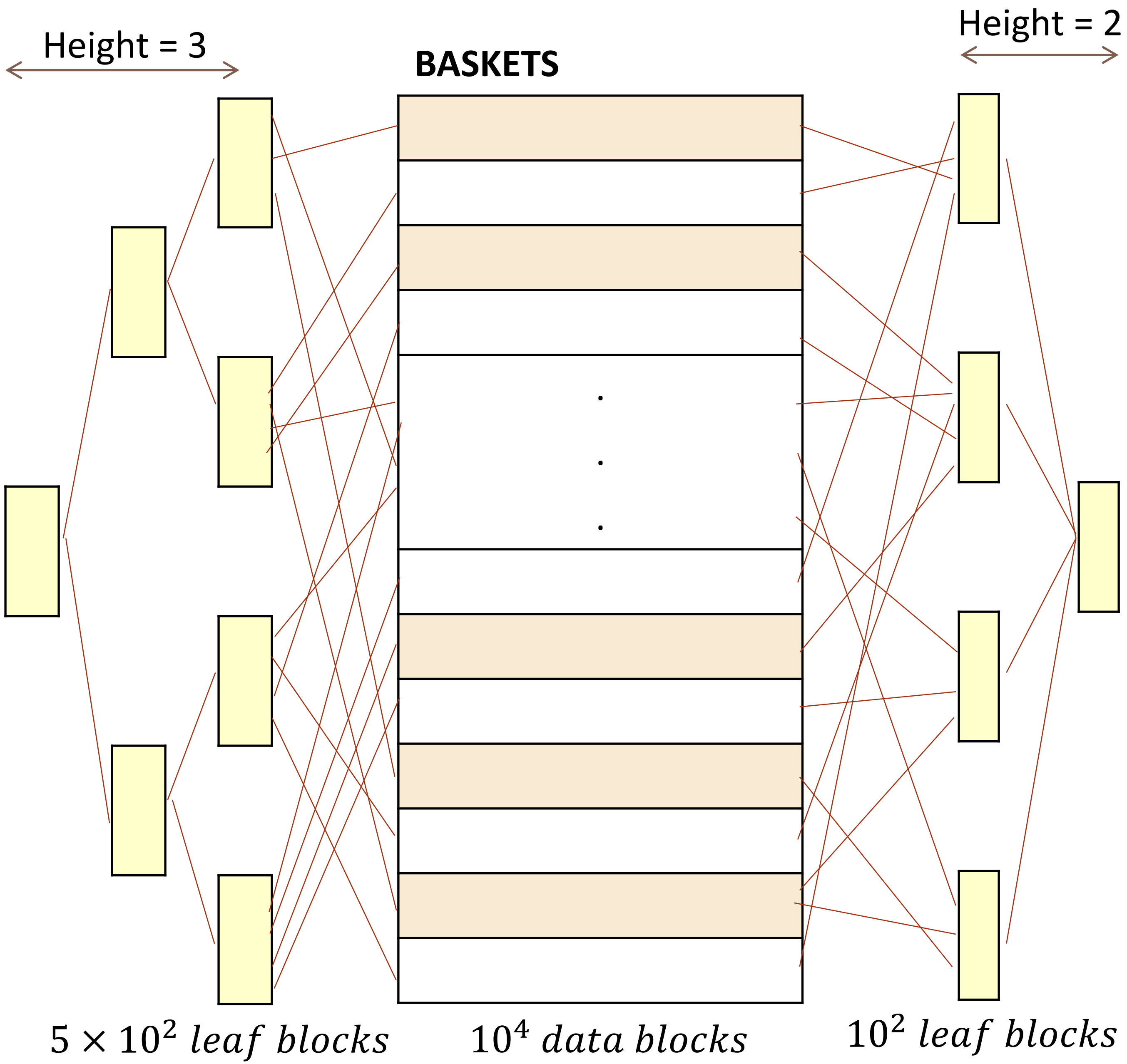
4)

```
SELECT product
FROM BASKETS
WHERE bid = 100 or product = ‘bolt’;
```

5)

```
SELECT sname, COUNT(*)
FROM BASKETS
WHERE sname IN (‘Golden Bolts Pty Ltd’, ‘Microsoft Corp.’)
GROUP BY sname;
```

A simple sketch of the scenario will help:



- $10^5\ rows$
- $blocking\ factor = 10$
- $Hence, data\ blocks = \frac{10^5 rows}{10} =$ $10^4\ data\ blocks$
- Primary key index: (bid, product)
- Additional index: (sname) Non-primary key

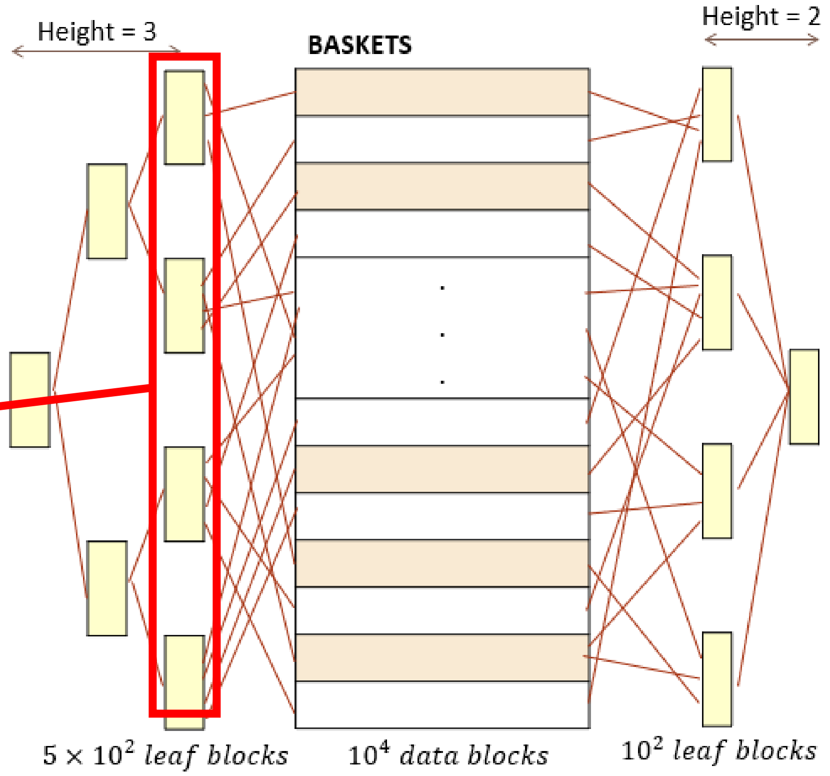
Bid	Products	Quantity	Sname	Recorded
10^5	10^3	3×10^3	10	3×10^2

1. SELECT COUNT(DISTINCT product)
   FROM BASKETS;

Since product is part of the primary key, the query processor is able to get the necessary information from the primary key index. The query processor will perform a horizontal full-scan of leaf nodes of the primary key index. As the scan is done, respective group of product is form, and finally, the aggregate count of the products in each product group is done. Total number of read blocks perform is $\mathbf{5 \times 10^2}$ leaf blocks.

- $10^5\ rows$
- $blocking\ factor = 10$
- $Hence, data\ blocks = \frac{10^5 rows}{10} = 10^4\ data\ blocks$
- Primary key index: (bid, product)
- Additional index: (sname) Non-primary key

Bid	Products	Quantity	Sname	Recorded
10^5	10^3	3×10^3	10	3×10^2

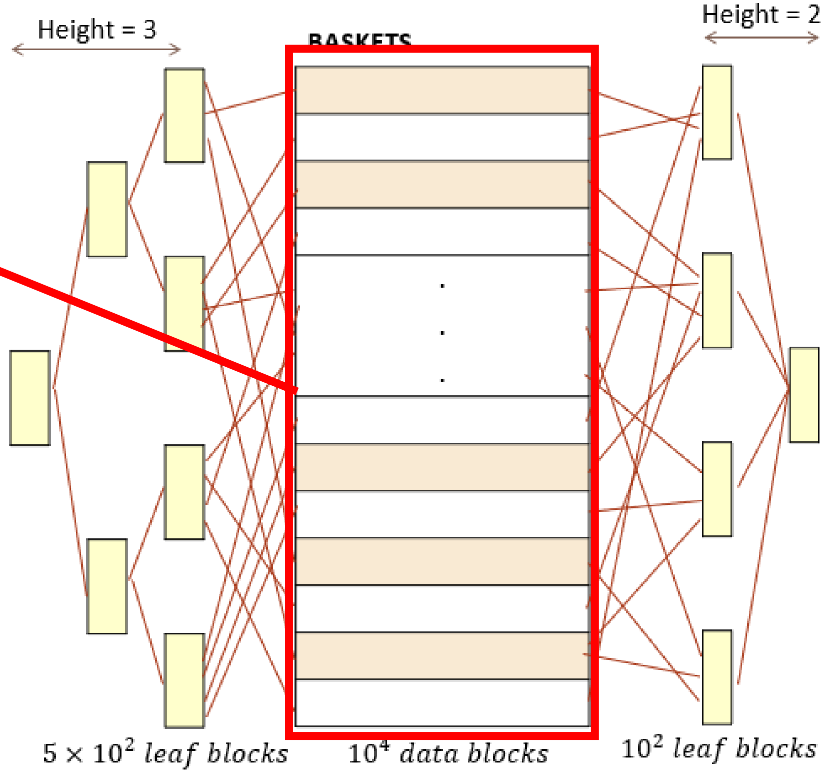


2) SELECT MAX(quantity)
FROM BASKETS;

The attribute quantity is not part of any indexes created for the table BASKETS. The query processor will perform a full-table scan of the relational table BASKETS and sort aggregate on the attribute quantity. Total number of read blocks perform equals the total number of data blocks, that is, $\mathbf{10^4}$ data blocks.

- $10^5\ rows$
- $blocking\ factor = 10$
- $Hence, data\ blocks = \frac{10^5 rows}{10} = 10^4\ data\ blocks$
- Primary key index: (bid, product)
- Additional index: (sname) Non-primary key

Bid	Products	Quantity	Sname	Recorded
10^5	10^3	3×10^3	10	3×10^2



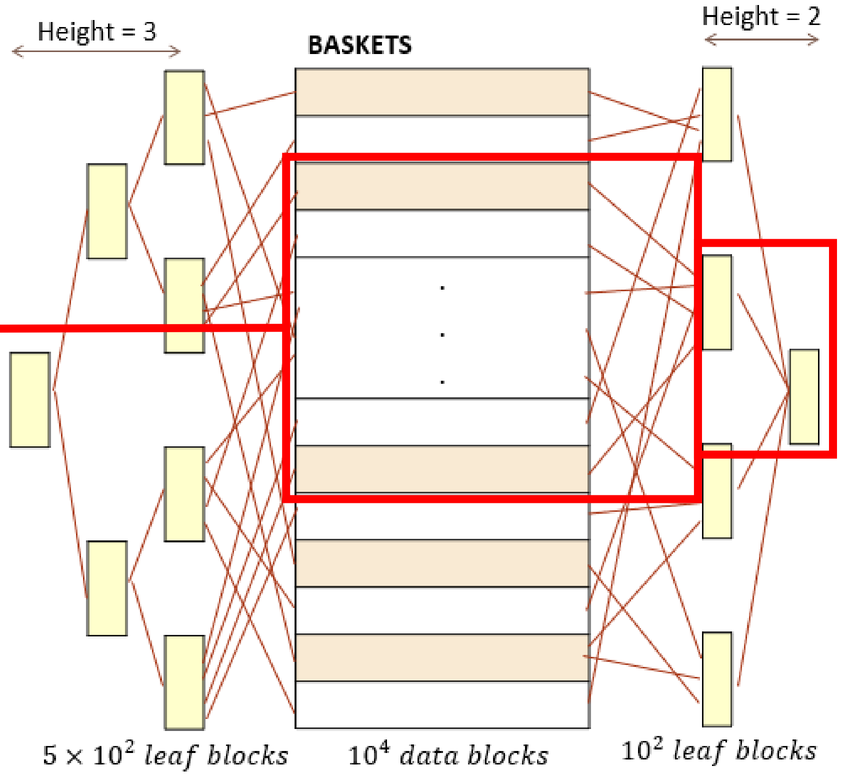
3) SELECT quantity
FROM BASKETS
WHERE sname = ‘Golden Bolts Pty Ltd’
AND recorded = ‘12-DEC-2012’;

The table has a non-key index on sname. The query processor will plan to traverse the index vertically to reach to the leaf node for the key ‘Golden Bolts Pty Ltd’. The processor will then plan to perform the fetching of rows based on average case, that is, (best-case + worst-case)/2. Total number of read blocks perform is (height of non-key index + (best-case + worst-case)/2 = $2 + \left(\frac{\left(10^5/_{10}\right)}{10}\right) + \left(\frac{\left(10^5/_{10}\right)}{1}\right) =$
$2 + (1000 + 10000) = 11{,}002$ blocks.

Note:
Best-case = (total number of rows / total number of different supplier) / maximum number of rows in a block.
Worst-case = (total number of rows / total number of different supplier) / minimum number of row in a block.

- 10^5 *rows*
- *blocking factor* $= 10$
- *Hence, data blocks* $= \frac{10^5 rows}{10} =$ 10^4 *data blocks*
- Primary key index: (bid, product)
- Additional index: (sname) Non-primary key

Bid	Products	Quantity	Sname	Recorded
10^5	10^3	3×10^3	10	3×10^2

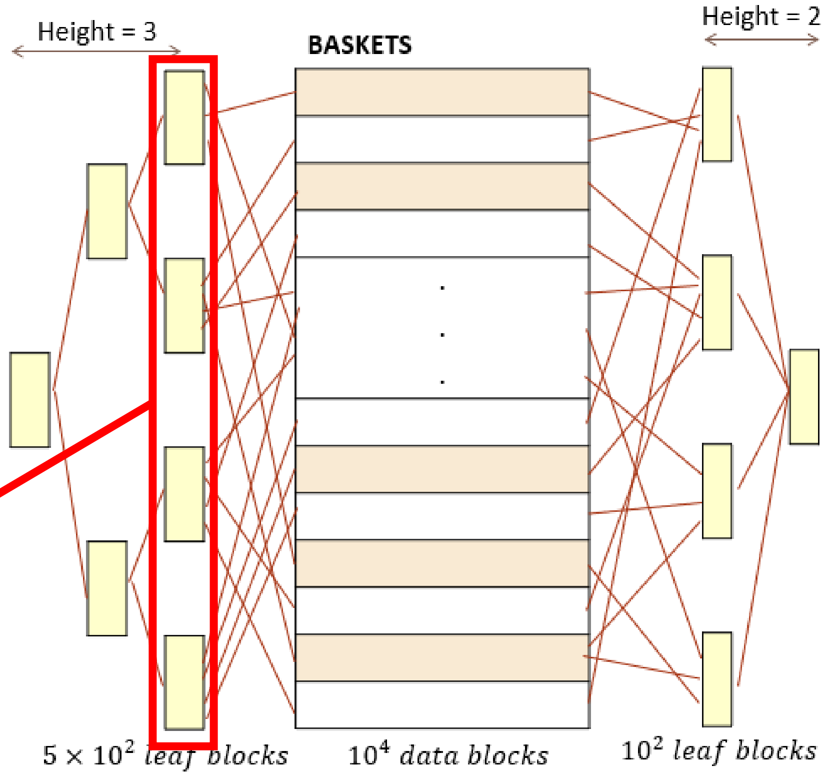


4) SELECT product
FROM BASKETS
WHERE bid = 100 or product = ‘bolt’;

The ‘WHERE’ clause contains a logical ‘OR’ operator and hence the primary key index cannot be used to retrieve the specific record that meet the requirements. However, since both the attributes bid and product can be found in the primary key index, the query processor will perform a horizontal full-scan on the leaf blocks to find the products that satisfy both the conditions. Total number of read-block performed is 5×10^2 leaf blocks.

- 10^5 *rows*
- *blocking factor* $= 10$
- *Hence, data blocks* $= \frac{10^5 rows}{10} = 10^4$ *data blocks*
- Primary key index: (bid, product)
- Additional index: (sname) Non-primary key

Bid	Products	Quantity	Sname	Recorded
10^5	10^3	3×10^3	10	3×10^2

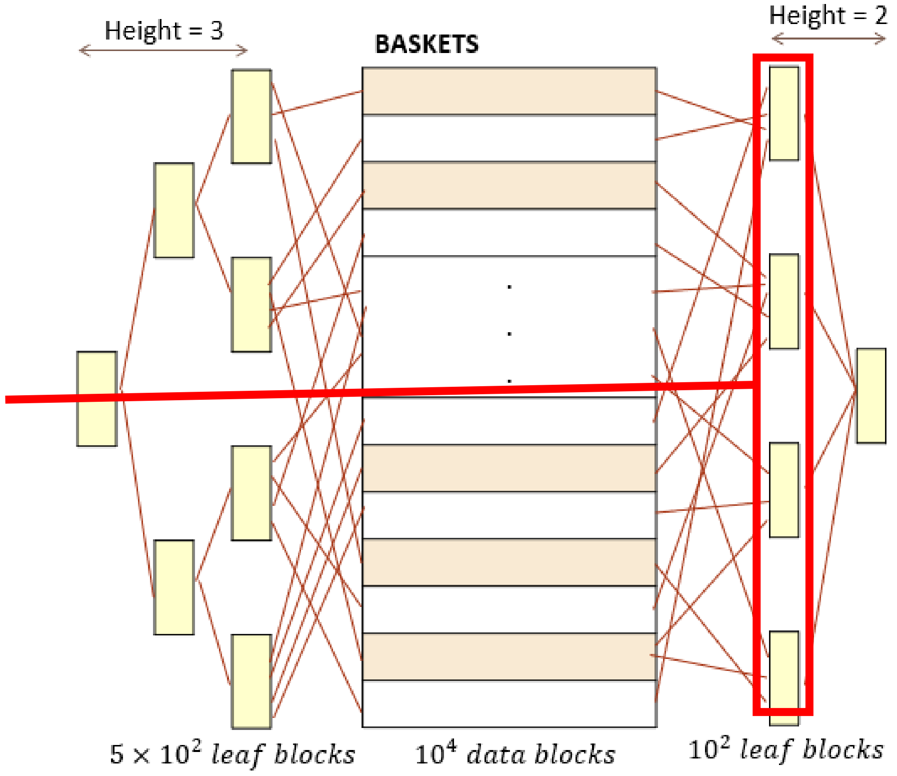


5) SELECT sname, COUNT(*)
FROM BASKETS
WHERE sname IN (‘Golden Bolts Pty Ltd’, ‘Microsoft Corp.’)
GROUP BY sname;

- 10^5 $rows$
- $blocking\ factor = 10$
- $Hence, data\ blocks = \frac{10^5 rows}{10} =$ 10^4 $data\ blocks$
- Primary key index: (bid, product)
- Additional index: (sname) Non-primary key

Bid	Products	Quantity	Sname	Recorded
10^5	10^3	3×10^3	10	3×10^2

The attribute sname is found in the non-key index, and the query processor will create an execution plan to perform a horizontal scan on the leaf-level nodes. As the scanning is done, groups are formed and aggregate count on sname will be carried out. At the end of the scanning, the aggregate count will be return. Total number of read blocks is 10^2.



Question 4

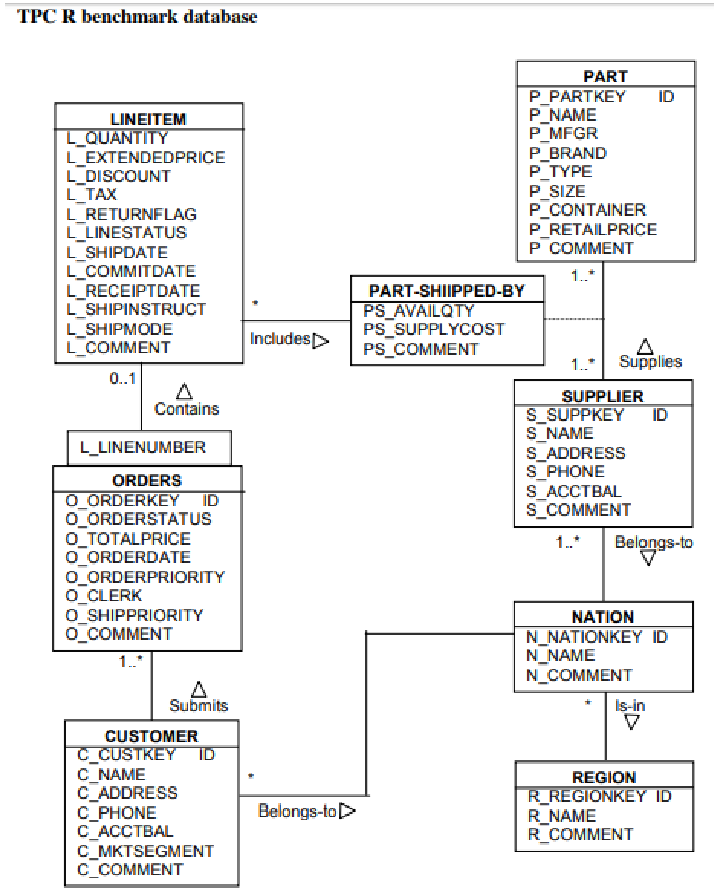
Consider a simplified TPC–HR benchmark database listed on page 2 of the final examination paper.

1) Write SELECT statement that implements the following query.

   For each name of part (P_NAME) find its total quantity (L_QUANTITY) of ordered by the customers. (2 marks)

2) Write a sequence of SQL statements that denormalize TPC-HR benchmark database and create an index such that SELECT statement implemented in the previous step can be computed faster than with the original relational tables of TPC-HR benchmark database.

Note, that denormalization must change a structure of the database and it must change the contents of the database. No other transformation of the database is required. (6 marks)



Question 4

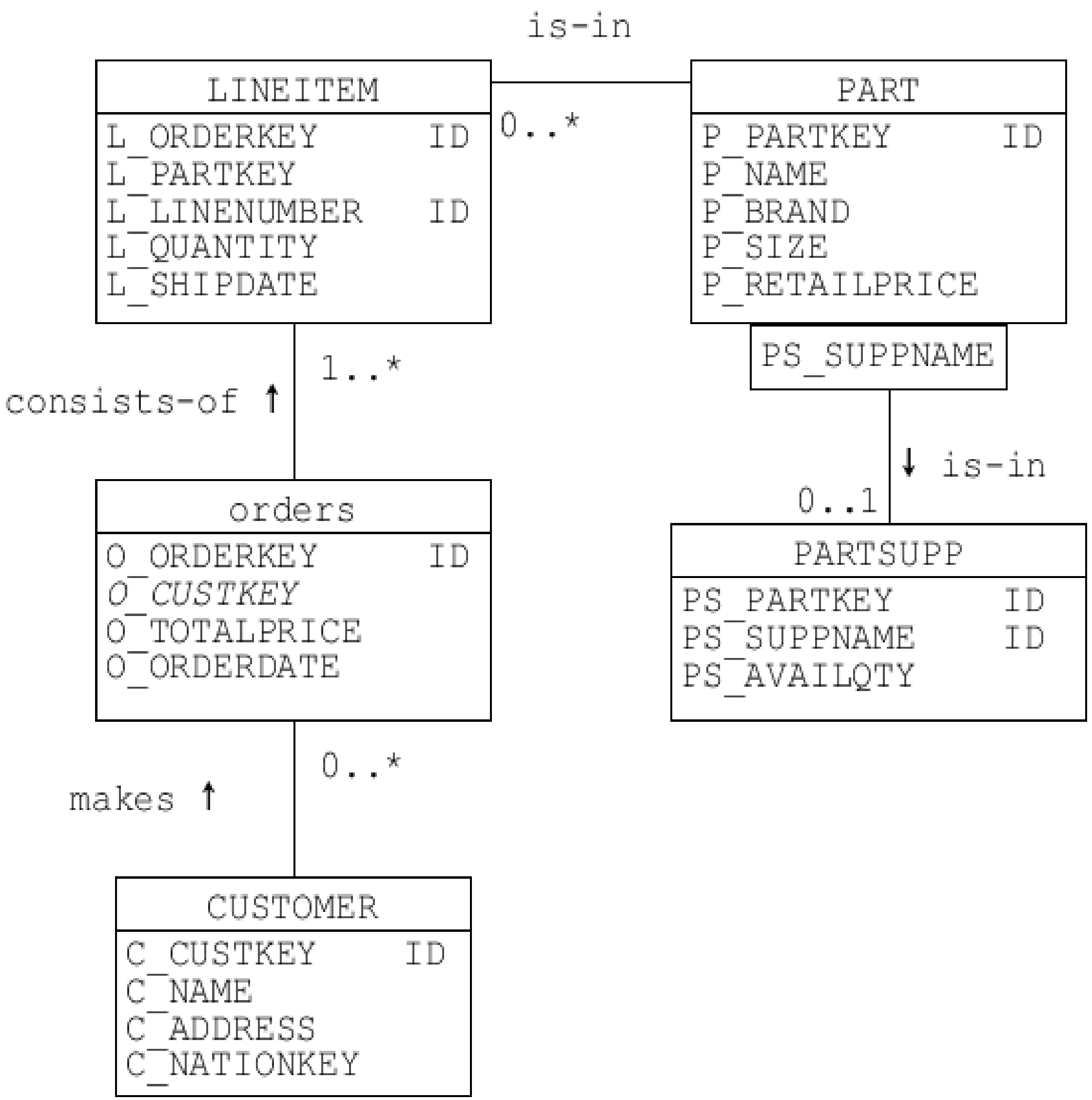
1)

SELECT P_NAME, SUM(L_QUANTITY)

FROM PART P, LINEITEM L

WHERE P_PARTKEY = L_PARTKEY

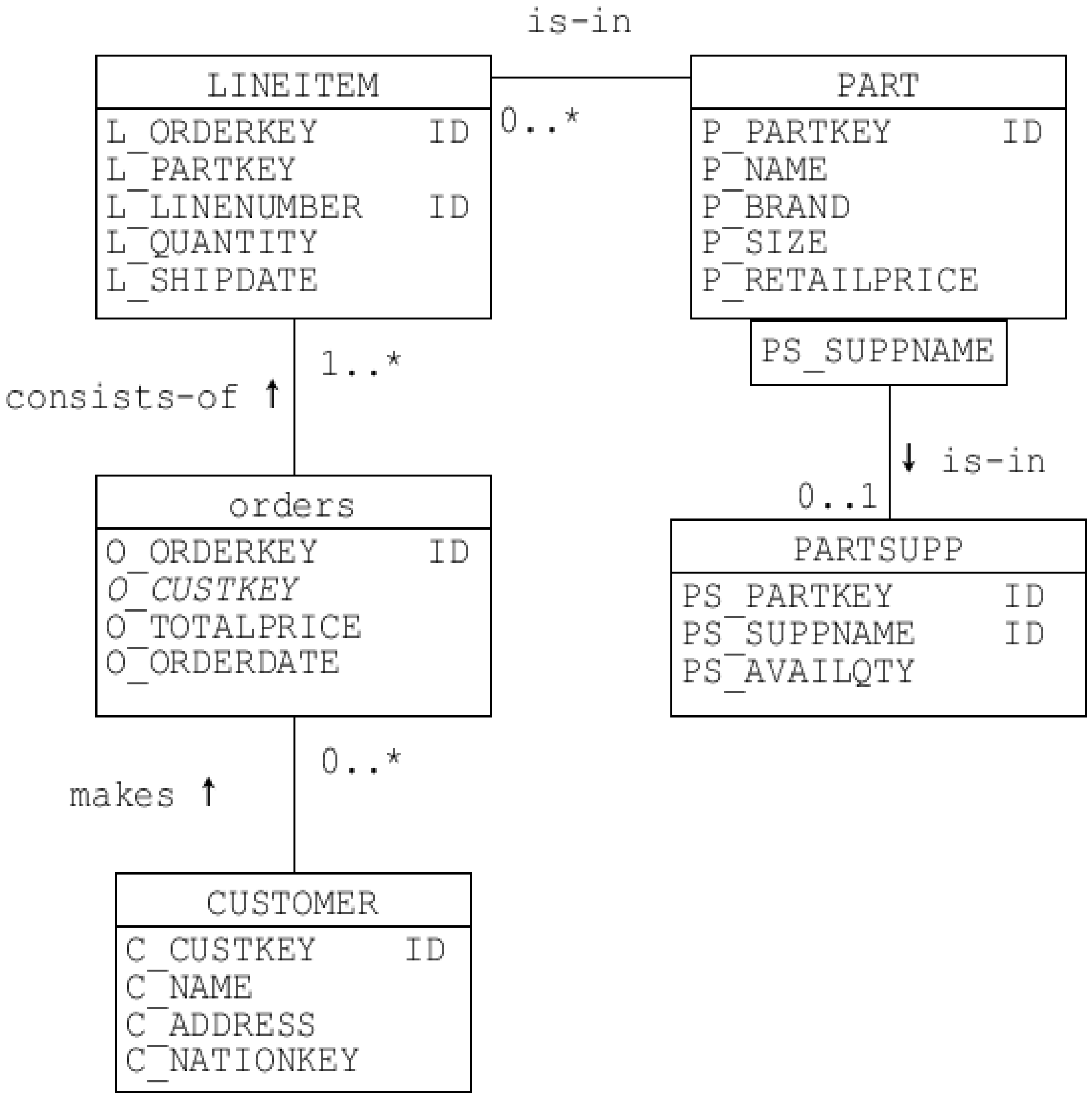
GROUP BY P_NAME;



Question 4

2)

To denormalized the TPCHR benchmark database in order to speed up the processing of the query specified in (1), we can copy the attribute P_NAME to the relational table LINEITEM. In this way, we can avoid joining the relational tables LINEITEM and PART and hence reducing the hash join and reducing the cartesian processing. The required information can then be obtained from the LINEITEM table alone.

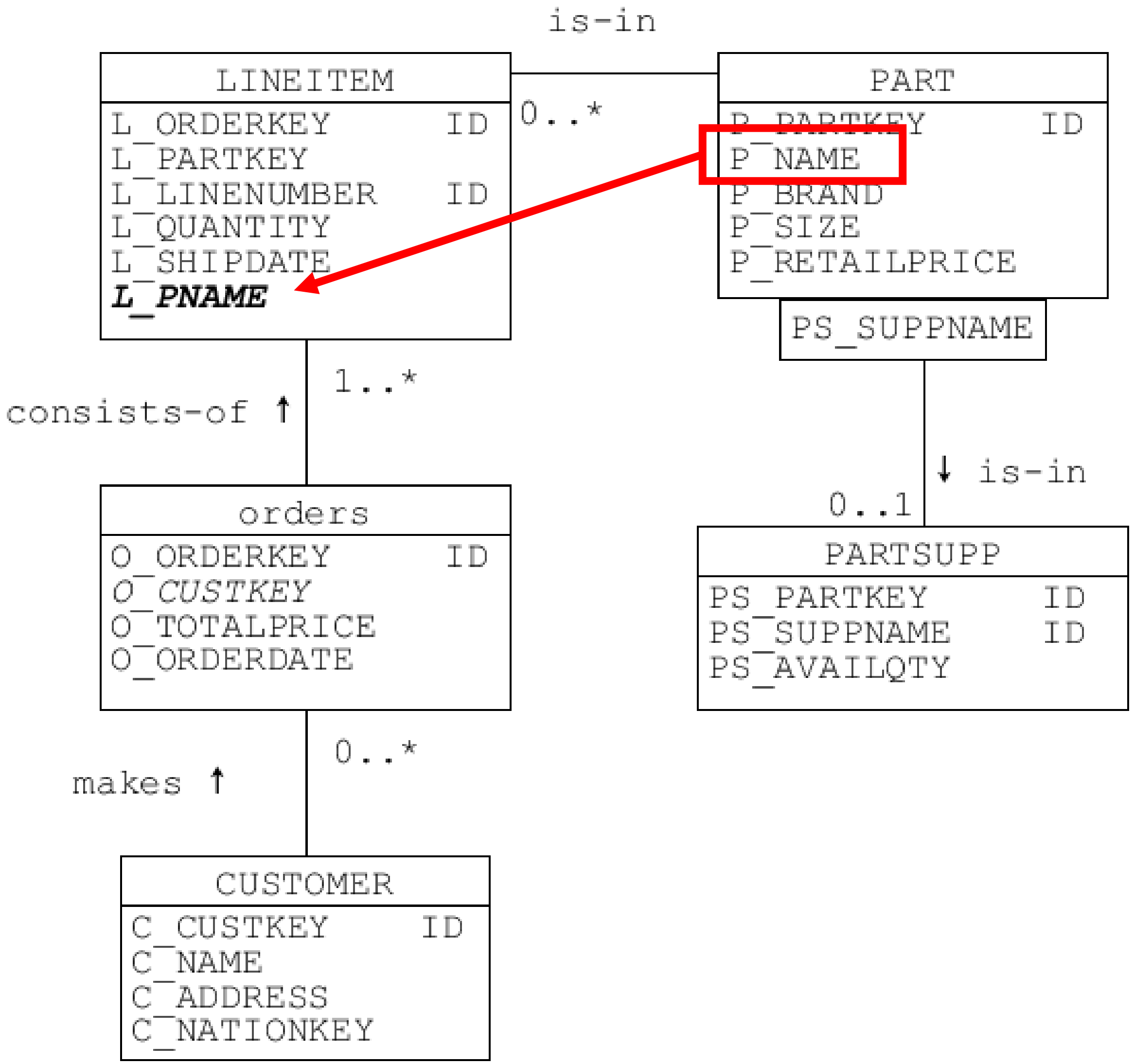


Question 4

2)

- Alter the table LINEITEM to add the attribute L_PNAME.

ALTER TABLE LINEITEM
ADD (L_PNAME VARCHAR2(55) NOT NULL);

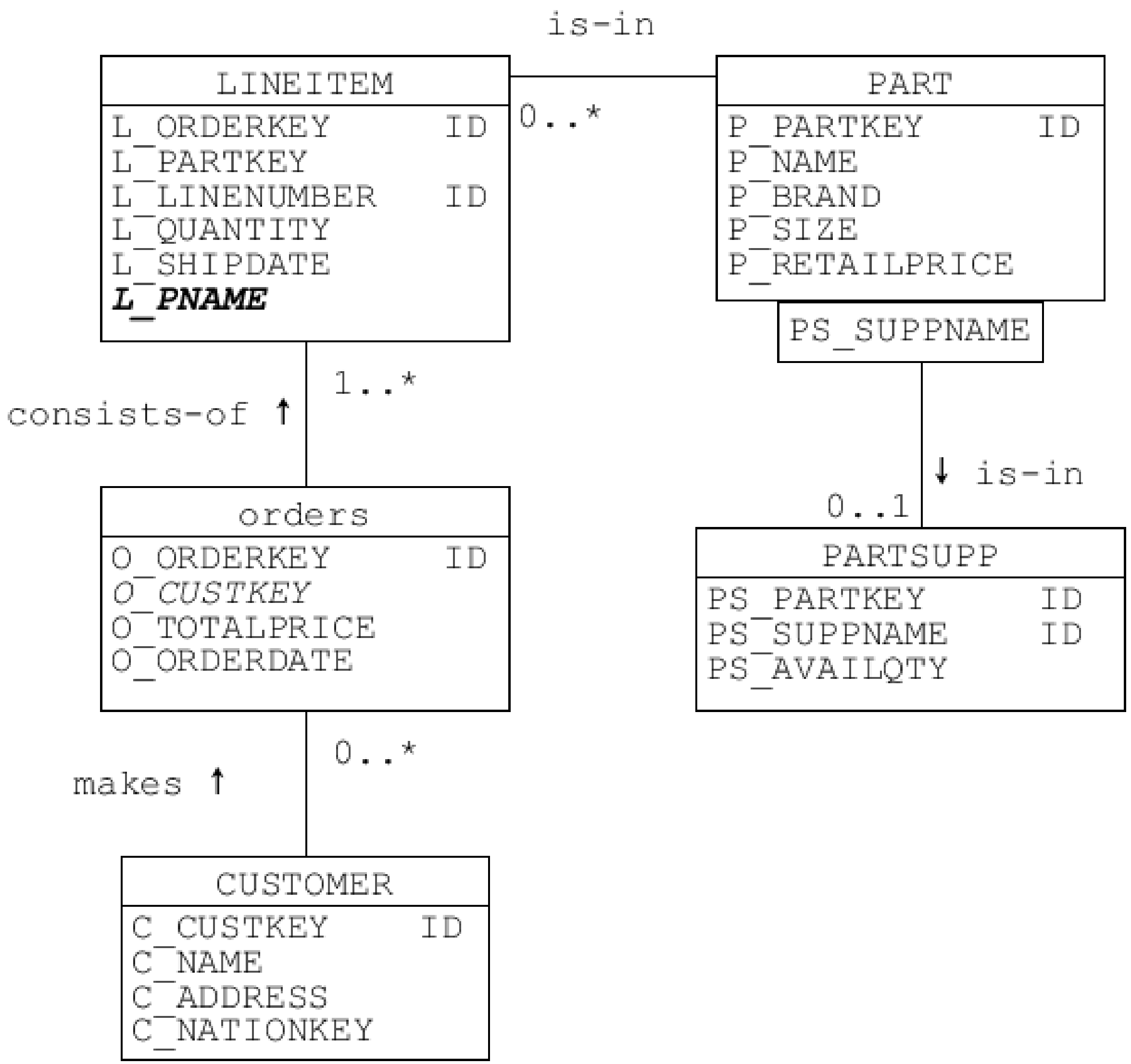


Question 4

2)

- Update the LINEITEM table to populate the newly added attribute L_PNAME with the respective name from PART table.

```
UPDATE LINEITEM
SET L_PNAME = (
        SELECT P_NAME
        FROM PART
        WHERE P_PARTKEY =
        L_PARTKEY)
WHERE L_PARTKEY = P_PARTKEY;
```

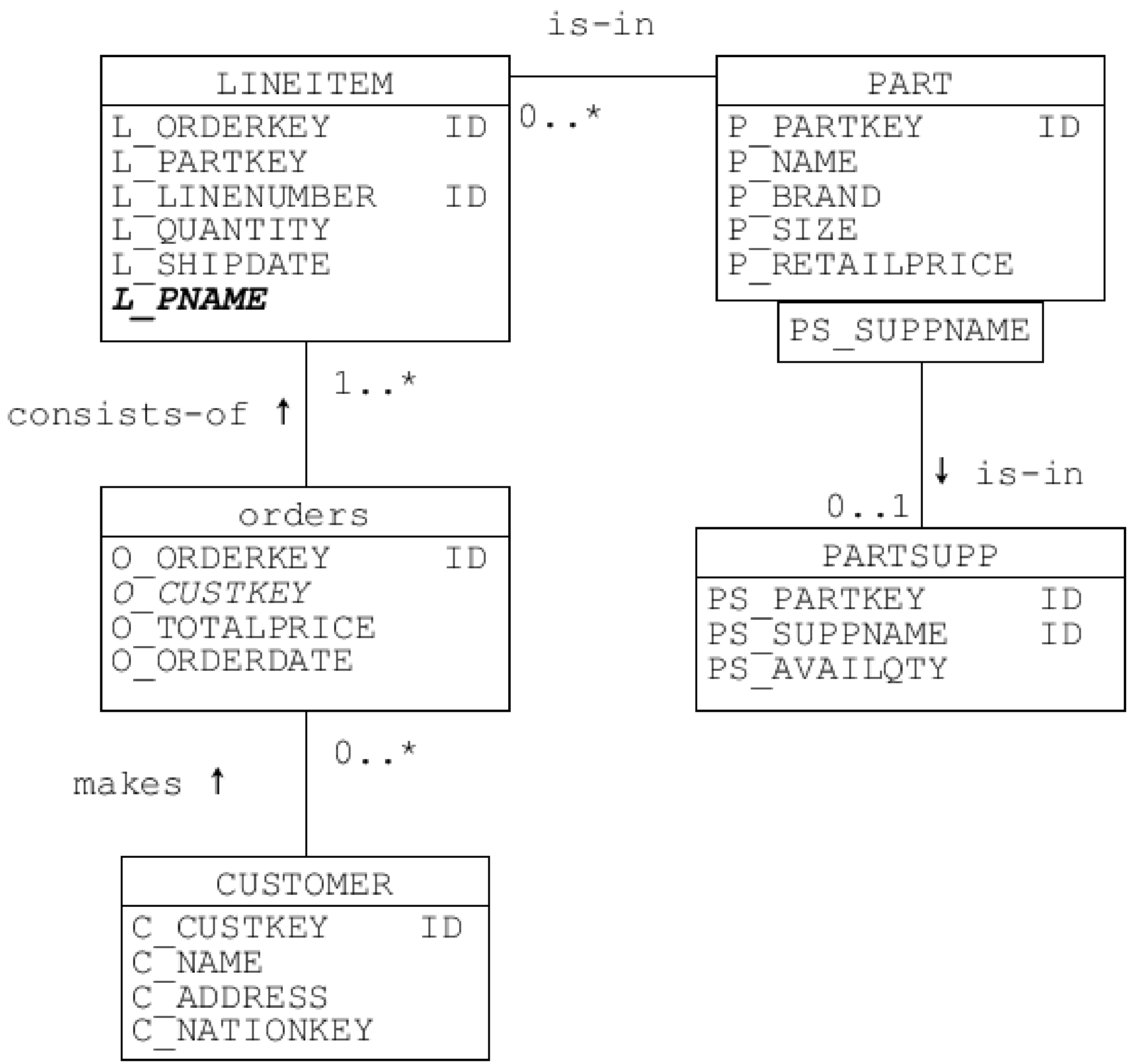


Question 4

2)

- Create index consisting of L_PNAME and L_QUANTITY to vertically partition the denormalized LINEITEM table to speed up the query.

CREATE INDEX LIPNAMEQTY ON LINEITEM (L_PNAME, L_QUANTITY);

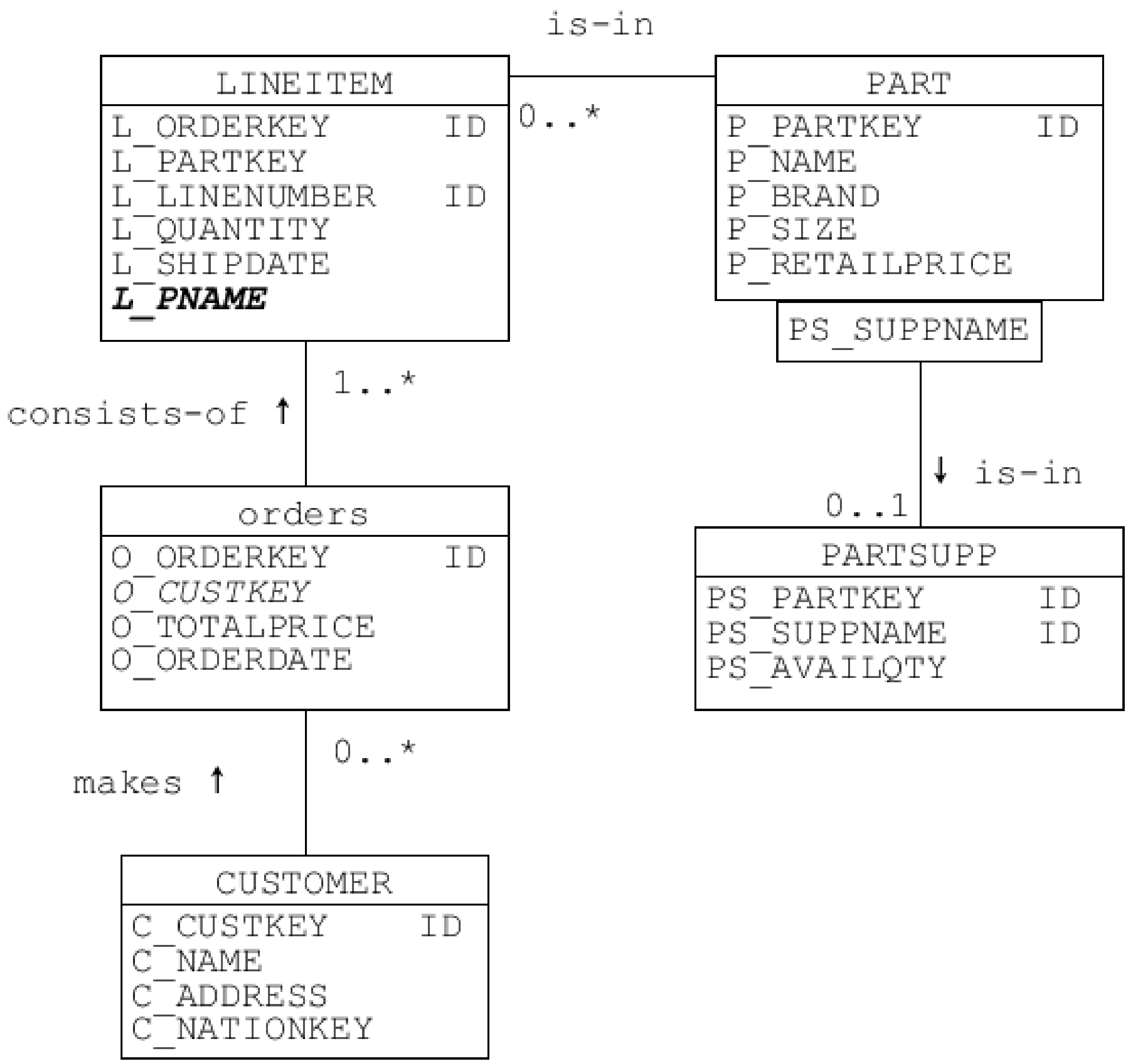


Question 4

2)

- A new query, but returning the same result as the query specified in (1), is as follow:

```
SELECT L_PNAME,
SUM(L_QUANTITY)
FROM LINEITEM
GROUP BY L_PNAME;
```



Question 5

Consider a fragment of simple JDBC application listed below. It is a typical example of a pretty poor, from performance point of view, JDBC program. Rewrite a code written below to improve the performance of the application it is included in. There is no need to write the entire JDBC application.

Explain all details why your version of JDBC code is more efficient than the original one

```
Statement stmt1 = conn.createStatement();
Statement stmt2 = conn.createStatement();
ResultSet rset1 = stmt1.executeQuery(
                                "SELECT DISTINCT C_CUSTKEY " +
                                "FROM CUSTOMER" );
long c_custkey = 0;
while ( rset1.next() )
{
  c_custkey = rset1.getInt(1);
  ResultSet rset2 = stmt2.executeQuery( "SELECT * " +
                                        "FROM ORDERS " +
                                        "WHERE C_CUSTKEY = " + c_custkey );
  long o_orderkey = 0;
  long total = 0;
  while ( rset2.next() )
  {
    o_orderkey = rset2.getInt(1);
    total++;
  }
  System.out.println( c_custkey + "    " + counter);
}
```

Question 5

- Analysing the code, and it is noted the segment of application processes two loop to compute the total price of orders made by each customer.
- The first statement ‘SELECT DISTINCT C_CUSTKEY FROM CUSTOMER;’ is sent to the dbms server which in turn returns the set of customer key.
- The second statement ‘SELECT * FROM ORDERS WHERE C_CUSTKEY = c_custkey;’ is constructed and send to the dbms server for execution.
- The dbms server will return a set of rows satisfying the query condition for each customer key that is matching.

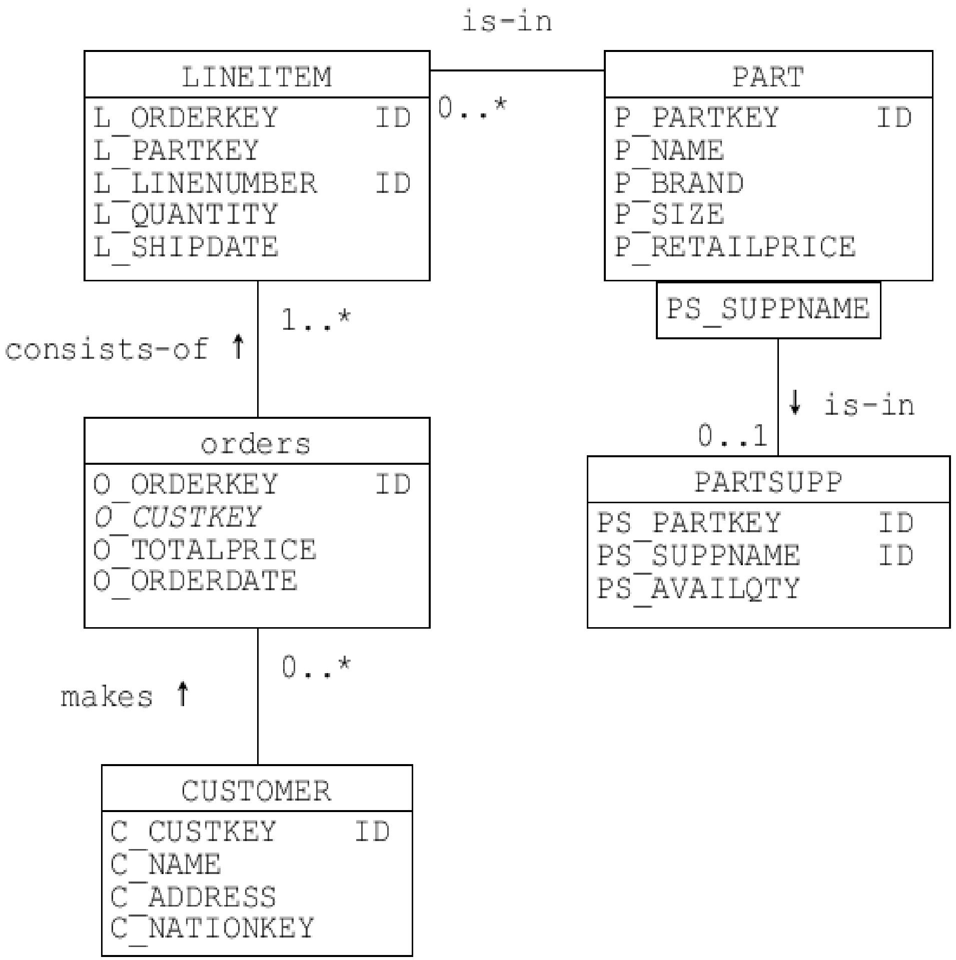
Question 5

- The application is then process through this set of rows using a while loop to compute the total order price for each customer.
- The in efficiency of this segment of code comes from the two loops. These two loops increase the processing at the client site (application).
- To improve the performance of the application (client site), we can let the server, which is more powerful, to compute the total order price for each customer before returning the aggregated value to the application (client) to display.

Question 5

- This can be done by combining the two queries as follow:

```
ResultSet rset1 = stmt1.executeQuery(
"SELECT C_CUSTKEY, O_ORDERKEY, SUM(O_TOTALPRICE) " +
"FROM ORDERS, CUSTOMER " +
"WHERE C_CUSTKEY = O_CUSTKEY " +
"GROUP BY C_CUSTKEY" );

C_CUSTKEY = rset1.getInt(1);
O_ORDERKEY = rset1.getInt(2);
Total = rset1.getInt(3);
```



Question 6

Consider SELECT statement given below.

```
SELECT P_PARTKEY, P_NAME, COUNT(*)
FROM   PART JOIN PARTSUPP
       ON PART.P_PARTKEY = PARTSUPP.PS_PARTKEY
GROUP BY P_PARTKEY, P_NAME
UNION
SELECT P_PARTKEY, P_NAME, 0
FROM   PART
WHERE  P_PARTKEY NOT IN ( SELECT PS_PARTKEY
                          FROM   PARTSUPP );
```

Question 6

A fragment of query execution plan that describe the extended relational algebra expression for SELECT statement above is the following.

1) Draw a syntax tree of a query processing plan given above.

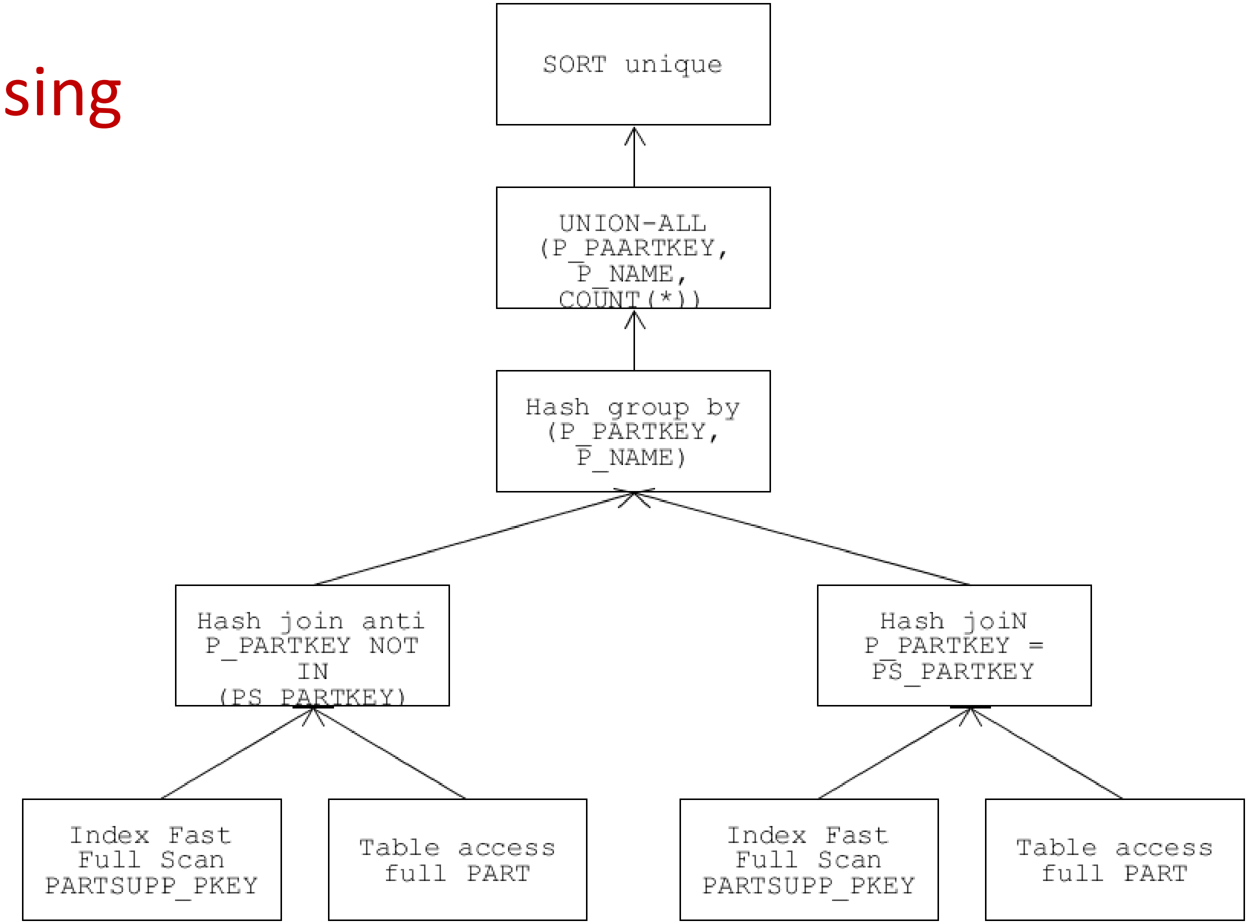
```
-----------------------------------------------------
| Id  | Operation                 | Name          | ...
-----------------------------------------------------
|   0 | SELECT STATEMENT          |               |
|   1 |  SORT UNIQUE              |               |
|   2 |   UNION-ALL               |               |
|   3 |    HASH GROUP BY          |               |
|*  4 |     HASH JOIN             |               |
|   5 |      TABLE ACCESS FULL    | PART          |
|   6 |      INDEX FAST FULL SCAN| PARTSUPP_PKEY |
|*  7 |    HASH JOIN ANTI         |               |
|   8 |     TABLE ACCESS FULL     | PART          |
|   9 |     INDEX FAST FULL SCAN  | PARTSUPP_PKEY |
-----------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("PART"."P_PARTKEY"="PARTSUPP"."PS_PARTKEY
   7 - access("P_PARTKEY"="PS_PARTKEY")
```

Question 6

2) A query processing plan given above reveals that relational table PART and an index PARTSUPP_PKEY on a primary key of PARTSUPP table are accessed twice (see lines 5, 6 and 8, 9 above). Find SELECT statement that retrieves the same information from the relational tables PART and PARTSUPP and such that it accesses a relational table PART and an index PARTSUPP_PKEY only once.

Question 6

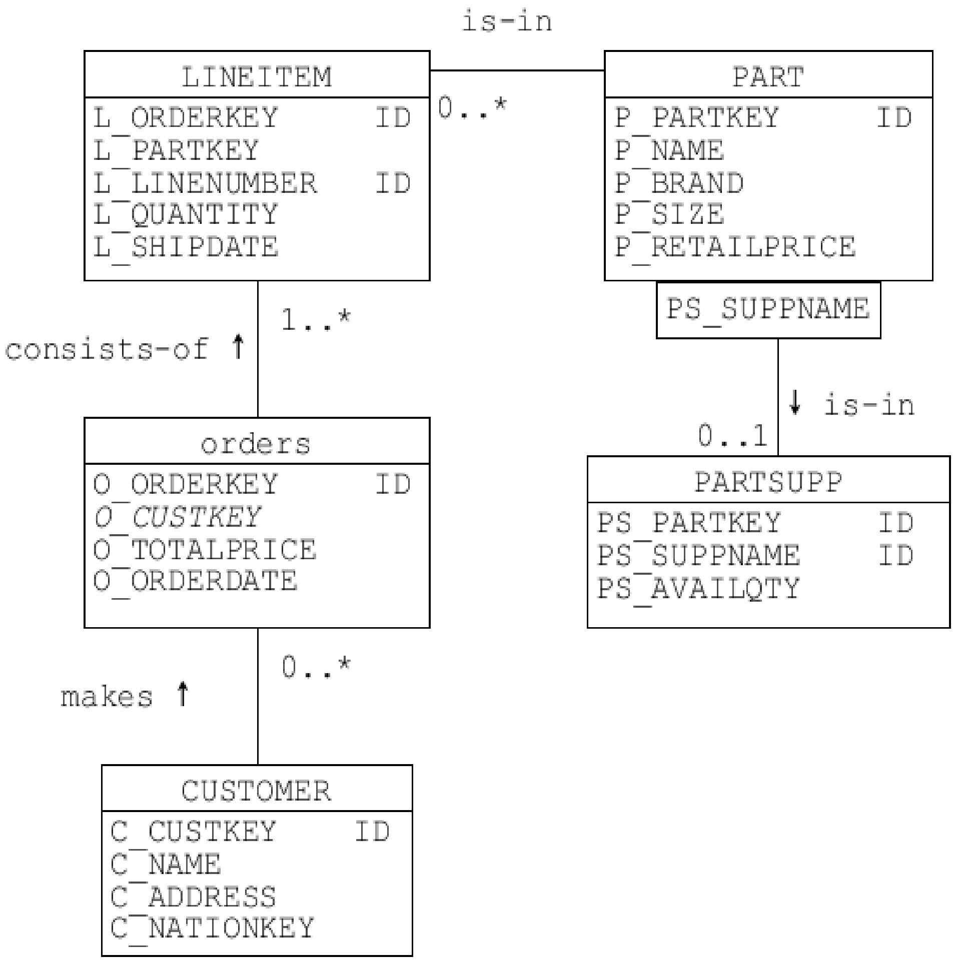
The query processing syntax tree:



Question 6

- A new select statement, but producing the same result as the original statement can be constructed using outer join construct as follow:

```
SELECT P_PARTKEY, P_NAME, COUNT(*)
FROM PART LEFT OUTER JOIN PARTSUPP
      ON P_PARTKEY = PS_PARTKEY
GROUP BY P_PARTKEY, P_NAME;
```



Question 7

Consider the SELECT statements given below. Each one of the given SELECT statements joins two or more relational tables. For each SELECT statement propose the best method for the implementation of the join algorithm. Justify your choice ! Note, that answers without the exhaustive and correct justifications score no marks!

Consider the following implementations of join operation:

i. Cartesian product join
ii. Nested loop join
iii. Nested loop join with one or both arguments kept in transient memory
iv. Index-based join
v. Sort-merge join
vi. Hash join
vii. Hash antijoin

Question 7

Assume that no more than 50 Mbytes of transient memory can be invested into the computations of join operation and that size of a bucket in hash implementation of join operation is always less than 5 Mbytes.

The sizes of relevant relational tables are listed at the bottom of the Introduction page of the final examination paper.

A solution of each one of the cases listed below is worth 2 marks.

(1)

```
SELECT *
FROM ORDERS JOIN LINEITEM
     ON ORDERS.O_ORDERDATE +1 = LINEITEM.L_SHIPDATE;
```

(2)

```
SELECT *
FROM ORDERS JOIN LINEITEM
     ON ORDERS.O_ORDERDATE > LINEITEM.L_SHIPDATE;
```

(3)

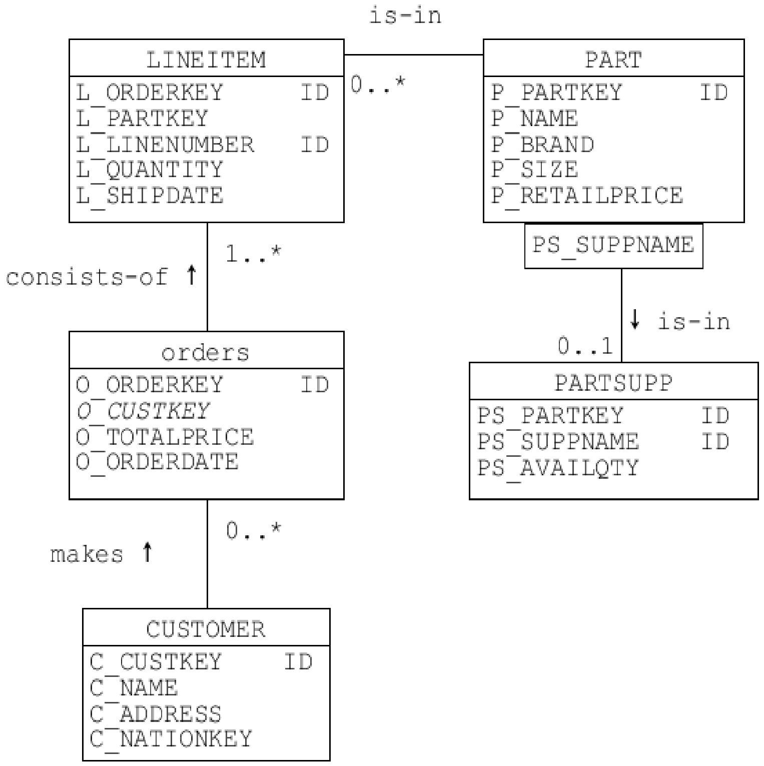
```
SELECT PART.P_NAME
FROM PART JOIN PARTSUPP
     ON PART.P_PARTKEY = PARTSUPP.PS_PARTKEY;
```

(4)

```
SELECT *
FROM ORDERS
WHERE O_ORDERKEY NOT IN ( SELECT L_ORDERKEY
                          FROM LINEITEM );
```

(5)

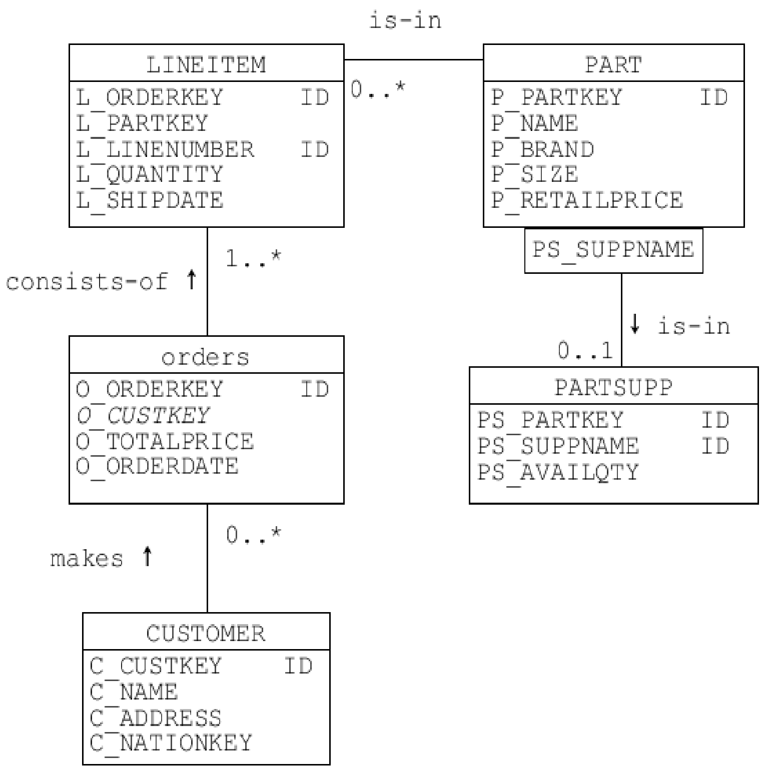
```
SELECT *
FROM PART, CUSTOMER
WHERE PART.P_NAME = 'bolt' AND CUSTOMER.C_NAME = 'bolt';
```



CUSTOMER	100 Mbytes
PART	30 Mbytes
PARTSUPP	400 Mbytes
ORDERS	500 Mbytes
LINEITEM	900 Mbytes

1)
```
SELECT *
FROM ORDERS JOIN LINEITEM
    ON ORDERS.O_ORDERDAATE +1 = LINEITEM.L_SHIPDATE;
```

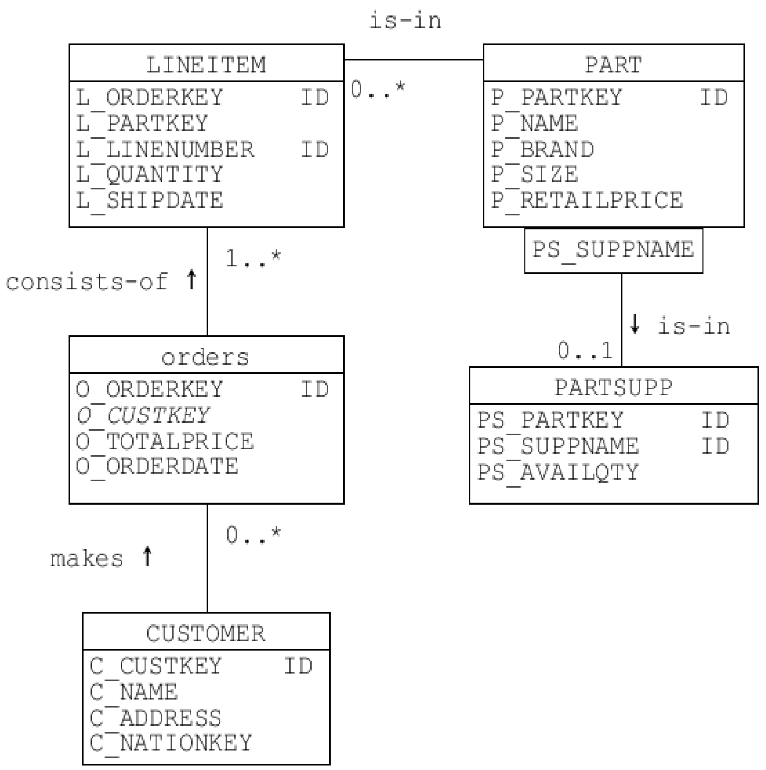
The relational tables ORDERS and LINEITEM are associated with O_ORDERKEY. However, the predicate of the query join the two tables using O_ORDERDATE and L_SHIPDATE. Since these two attributes are not part of the indexes in the two table, the two tables do not have a join condition, and hence the best method to implement the join would be a **cartesian product join**.



CUSTOMER	100 Mbytes
PART	30 Mbytes
PARTSUPP	400 Mbytes
ORDERS	500 Mbytes
LINEITEM	900 Mbytes

2) SELECT *
FROM ORDERS JOIN LINEITEM
ON ORDERS.O_ORDERDATE > LINEITEM.L_SHIPDATE;

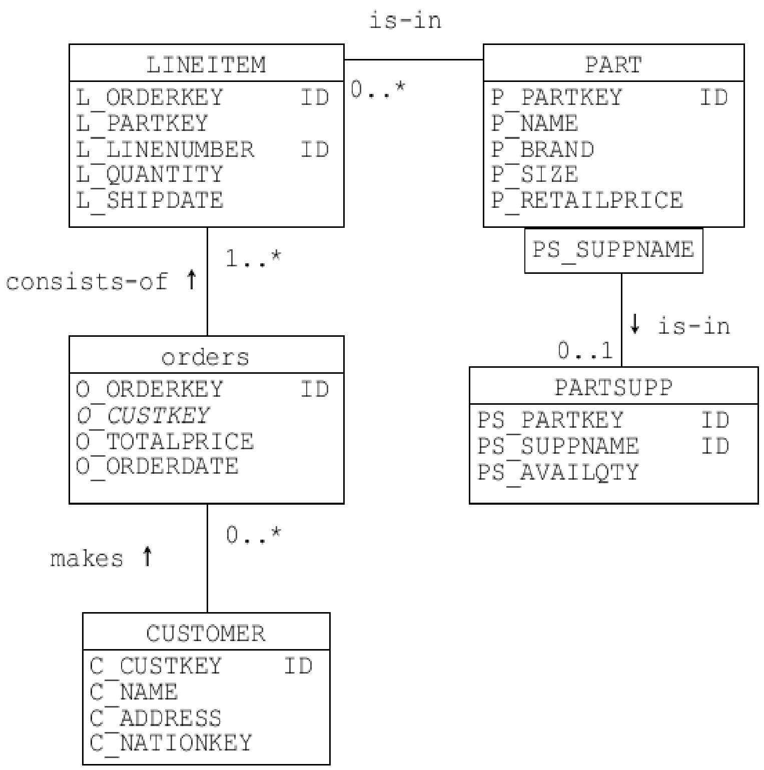
The predicate of the query involve an in-equality condition ‘O_ORDERDATE > L_SHIPDATE’ and the two tables are relatively large. The best method to implement the join would be **sort-merge join**. The sort-merge join would first sort the two tables based on the join condition, in this case the O_ORDERDATE on ORDERS and L_SHIPDATE on LINEITEM. Next the sort-merge join would merge the two sorted list together.



CUSTOMER	100 Mbytes
PART	30 Mbytes
PARTSUPP	400 Mbytes
ORDERS	500 Mbytes
LINEITEM	900 Mbytes

3) SELECT PART.P_NAME
FROM PART JOIN PARTSUPP
ON PART.P_PARTKEY = PARTSUPP.PS_PARTKEY;

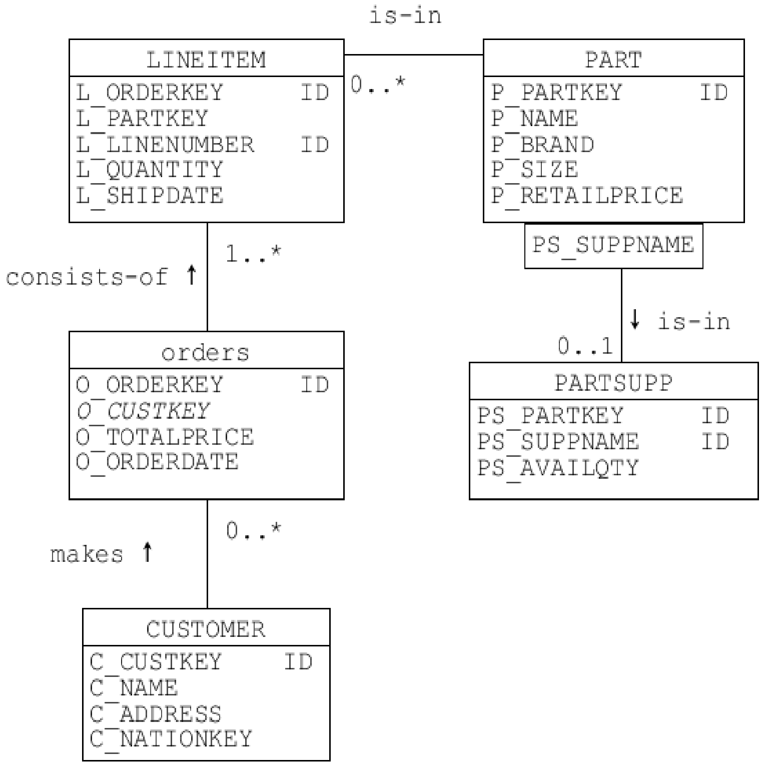
The predicate of the query involve an equijoin operator, and the attributes of the join condition are both an attribute of respective primary key of the tables. The best method to implement the join would be **Hash join**. The smaller table, in this case, PART will be used to build a hash table, based on the join key. Next the larger table PARTSUPP will be scanned and the same hashing algorithm will be done on the attributes involved in the join column(s). The result is then probed to the previously built hash table, and if they match, a row is returned.



CUSTOMER	100 Mbytes
PART	30 Mbytes
PARTSUPP	400 Mbytes
ORDERS	500 Mbytes
LINEITEM	900 Mbytes

4) SELECT *
FROM ORDERS
WHERE O_ORDERKEY NOT IN (SELECT L_ORDERKEY
FROM LINEITEM);

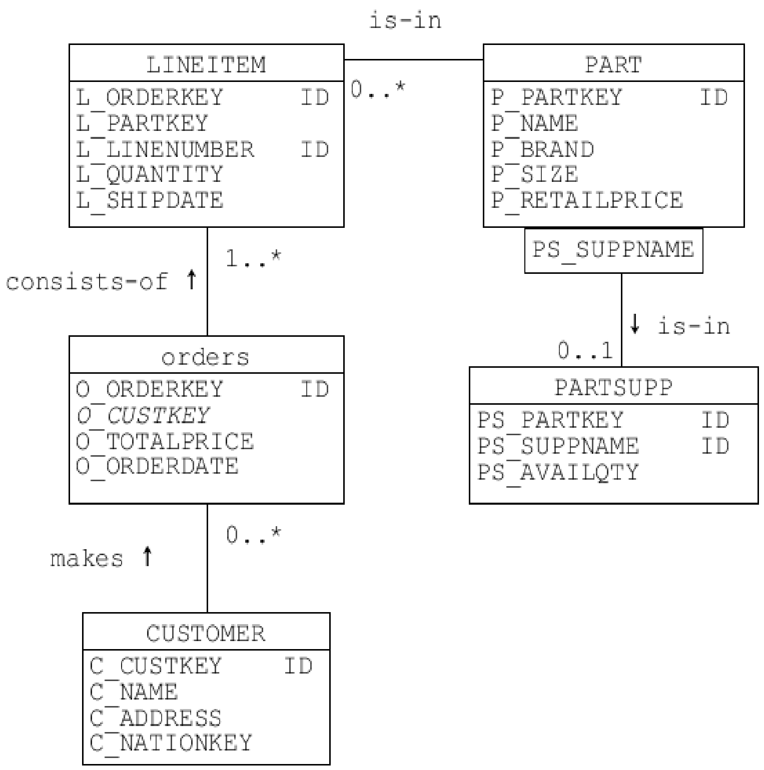
The query is constructed using subqueries with 'NOT in' clause. Both the attributes O_ORDERKEY and L_ORDERKEY are part of the primary key of the respective tables. The best method to implement the join is **Hash anti join** (or hash join anti, that is, using **anti join type** on **hashed join** operation). Hash tables are built for the two tables based on the join key. Next for each hashed key in the smaller table (ORDERS), it is then compared to the hash the larger table LINEITEM. If no match is found in the second table a row in the first table is return.



CUSTOMER	100 Mbytes
PART	30 Mbytes
PARTSUPP	400 Mbytes
ORDERS	500 Mbytes
LINEITEM	900 Mbytes

5) SELECT *
FROM PART, CUSTOMER
WHERE PART.P_NAME = ‘bolt’ AND CUSTOMER.C_NAME = ‘bolt’;

The predicate of the query consists of two conditions comparing with literal ‘bolt’ and logically AND together. Since the attributes from the tables are not used to join the table (compare to each other), the best method to implement the join would be **sort-merge join**. The processor would sort both tables individually based on the join predicates. Next, join every row from one table with every row from the other table, creating a cartesian product of the two tables and compare for matching rows. If match is found, the rows are return.



CUSTOMER	100 Mbytes
PART	30 Mbytes
PARTSUPP	400 Mbytes
ORDERS	500 Mbytes
LINEITEM	900 Mbytes