Problem 1
=========

English:

What are the main tasks of a query optimizer?

Bokmål:

Hva er hovedoppgavene til en spørringsoptimalisator (query optimizer)?

Nynorsk:

Kva er hovudoppgåvane til ein spørjingsoptimalisator (query optimizer)?

Answer:

The main goal is to find the best possible (typically fastest, but could also be limited by resources) execution plan for the query. The main tasks are typically focused on choice of access methods, choice of join order and choice of join algorithms.

Problem 2
=========

English:

Give an example of an SQL query that is rewritten to use semijoin before optimization (provide both the SQL query and the algebraic form after transformation). How does semijoin transformation affect the work done by the query optimizer?

Bokmål:

Gi et eksempel på en SQL-spørring som skrives om til å bruke semijoin før optimalisering (oppgi både SQL-spørringen og den algebraiske formen etter omskriving). Hvordan påvirker semijoin-omskriving arbeidet som gjøres av spørringsoptimalisatoren (the query optimizer)?

Nynorsk:

Gjev eit eksempel på ei SQL-spørjing som vert omskrive til å bruke semijoin før optimalisering (oppgjev både SQL-spørjinga og den algebraiske forma etter omskriving). Korleis påverkar semijoin-omskrivinga arbeidet som spørringsoptimalisatoren (the query optimizer) gjør?

Answer:

```
SELECT COUNT(*)
FROM department AS d
WHERE d.dnumber IN (SELECT E.dno
                    FROM employee AS e
                    WHERE e.salary > 200000);

project COUNT(*)
|
semijoin d.dnumber=e.dno
|     |
d     project e.dno
      |
      select e.salary > 200000
      |
      e
```

Semijoin transformations generally gives the query optimizer more join ordering options since subqueries are unnested, producing fewer query blocks (SELECTs) with more join/semijoin operations.

Problem 3

English:

Consider a DBMS with similar capabilities to System R and assume a schema and data set where there are indexes on all columns. This system executes the following query:

```
SELECT *
FROM Hotels h
WHERE h.city = 'Nassau' AND NOT EXISTS (
      SELECT *
      FROM Hotels h1
      WHERE h1.city = 'Nassau' AND
            h1.distance <= h.distance AND
            h1.price <= h.price AND
            (h1.distance < h.distance OR
            h1.price < h.price));
```

List all sargable predicates in this query and explain why each predicate is sargable.

Bokmål:

Anta et DBMS med egenskaper tilsvarende System R og et skjema og datasett med indekser på alle kolonner. Dette systemet utfører den følgende spørringen:

```
SELECT *
FROM Hotels h
WHERE h.city = 'Nassau' AND NOT EXISTS (
      SELECT *
      FROM Hotels h1
      WHERE h1.city = 'Nassau' AND
            h1.distance <= h.distance AND
            h1.price <= h.price AND
            (h1.distance < h.distance OR
            h1.price < h.price));
```

Oppgi alle sargbare predikater (sargable predicates) i spørringen og forklar for hvert predikat hvorfor det er sargbart (sargable).

Nynorsk:

Anta eit DBMS med eigenskapar tilsvarande System R og eit skjema og datasett med indeksar på alle kolonner. Dette systemet utførar den følgjande spørjinga:

```
SELECT *
FROM Hotels h
WHERE h.city = 'Nassau' AND NOT EXISTS (
      SELECT *
      FROM Hotels h1
      WHERE h1.city = 'Nassau' AND
```

```
            h1.distance <= h.distance AND
            h1.price <= h.price AND
            (h1.distance < h.distance OR
            h1.price < h.price));
```

Oppgjev alle sargbare predikat (sargable predicates) i spørjinga og forklar for kvart predikat kvifor det er sargbart (sargable).

Answer:

h.city = 'Nassau': Can be used to do index lookup on h.city
h1.city = 'Nassau': Can be used to do index lookup on h1.city

In System R, comparing columns to columns are not sargable. Sargable predicates are always on the form <column> <comparison operator> <value>.

h1.distance <= h.distance, h1.price <= h.price, h1.distance < h.distance, and h1.price < h.price are not sargable since the value we're searching for is not a constant that can be pushed down to an index lookup.

Problem 4

English:

In a DBMS similar to System R, we have the following selectivity factors:

```
column1 = value:  F = 1 / 10
(pred1) AND (pred2): F = F(pred1) * F(pred2)
```

Which assumptions lie behind these formulas? Give an example where these assumptions don't hold and explain why.

Bokmål:

I et DBMS som ligner på System R har vi disse selektivitetsfaktorene (selectivity factors):

```
column1 = value: F = 1 / 10
(pred1) AND (pred2): F = F(pred1) * F(pred2)
```

Hvilke antagelser ligger bak disse formlene? Gi et eksemple hvor disse antagelsene ikke holder og forklar hvorfor.

Nynorsk:

I eit DBMS som liknar System R har vi desse selektivitestfaktorane (selectivity factors):

```
column1 = value: F = 1 / 10
(pred1) AND (pred2): F = F(pred1) * F(pred2)
```

Kva for føresetnader ligg bak desse formlane? Gjev eit eksempel kor desse føresetnadene ikkje held og forklar kvifor.

Answer:

1) The formula for column1 = value assumes uniform distribution of data.

2) The formula for (pred1) AND (pred2) assumes independent values.

Example:

```
SELECT *
FROM computer_science_students
WHERE gender = 'M' AND name = 'Adam';
```

The underlying assumptions break because

1) This data set has a gender imbalance, so there are not 1/10 males, and there are not 1/10 people named 'Adam'

2) There is a correlation between gender and having the name 'Adam'

Problem 5

English:

Explain how System R decides between explicit sorting and using an index to read data in sorted order.

Bokmål:

Forklar hvordan System R velger mellom eksplisitt sortering og bruk av en indeks til å lese data i sortert rekkefølge.

Nynorsk:

Forklar kordan System R vel mellom eksplisitt sortering og bruk av ein indeks til å lese data i sortert rekkefølgje.

Answer:

System R tracks interesting orders, i.e., orderings useful for merge join, ORDER BY and GROUP BY. As it builds the query plan bottom-up, it keeps track of the best plan overall and the best plan for any interesting order. When an operation occurs that can benefit from a sorted order, it compares the cost of explicitly sorting the overall best plan and the best plan that doesn't require explicit sorting (i.e., already in sorted order).

Problem 6

English:

Explain how score guided hash rank join differs from the basic hash rank join algorithm and how it affects performance.

Bokmål:

Forklar hvordan score guided hash rank join avviker fra den basale hash rank join-algoritmen og hvordan det påvirker ytelsen.

Nynorsk:

Forklar korleis score guided hash rank join avvik frå den basale hash rank join-algoritma og korleis det påverkar ytinga.

Answer:

Score guided HRJN differs from the basic algorithm by reading from its operands at different speeds, i.e., it doesn't switch between the operands for each step, but instead uses the terms in the threshold computation to decide which operand to read from in the next step.

This lowers the threshold faster than alternating, meaning that the score guided algorithm in general produces the final result faster.

Problem 7

English:

Given an LSM-tree C0, C1, C2, explain why C2 > C1 > C0 and why it's better to have more components than to let existing components always grow larger and larger as data is added.

Bokmål:

Gitt et LSM-tre C0, C1, C2, forklar hvorfor C2 > C1 > C0 og hvorfor det er bedre å ha flere komponenter enn å la eksisterende komponenter alltid vokse større og større når data legges til.

Nynorsk:

Gjeve eit LSM-tre C0, C1, C2, forklar kvifor C2 > C1 > C0 og kvifor det er betre å ha fleire komponentar enn å la eksisterande komponentar alltid vokse større og større når data leggjes til.

Answer:

In general, the purpose of growing sizes is the ability to store more data, and since data migrates from C0 to C1 to C2, C2 will be the larger component.

C0 has to fit in memory. It can't grow larger than the available memory. As the data set grows, C1 and C2 will grow larger than C0.

As more data is added, if the difference between C1 and C0 grows too large, the merge step will have to read large amounts of data from C1 in order to merge a single row from C0. It's more efficient if the relative size difference between Ck and Ck+1 is not too large. E.g., therefore, it's beneficial to add C2 instead of growing C1 much larger than C0. Similarly, as C2 grows, it will be beneficial to add C3 so that the difference between C1 and C2 doesn't grow too large.

Problem 8

English:

Give an example where a lazy update anywhere system experiences an inconsistency. Explain how/why the inconsistency occurs.

Bokmål:

Gi et eksempel hvor et lazy update anywhere-system får inkonsistens (inconsistency). Forklar hvordan/hvorfor inkonsistensen oppstår.

Nynorsk:

Gjev eit eksempel kor eit lazy update anywhere-system får inkonsistens (inconsistency). Forklar kordan/kvifor inkonsistensen oppstår.

Answer:

Many possible solutions, but the student is expected to provide a history where an inconsistency occurs, e.g., two conflicting updates.

Problem 9
=========

English:

What does the DistributedUnion operator in Spanner do, and why does the optimizer try to push it upwards in the query plan?

Bokmål:

Hva gjør operatoren DistributedUnion i Spanner, og hvorfor forsøker optimalisatoren (the optimizer) å dytte den oppover i spørringsplanen (the query plan)?

Nynorsk:

Kva gjer operatoren DistributedUnion i Spanner, og kvifor freistar optimalisatoren (the optimizer) å dytte den oppover i spørjingsplanen (the query plan)?

Answer:

The DistributedUnion operator gathers and concatenates the results from several parallel execution threads on individual shards. (Note: The paper says "used to ship a subquery to each shard [...] and to concatenate the results").

By pushing DistributedUnion upwards in the query plan, more execution is done locally on each shard, in parallel.

Problem 10
==========

English:

Consider two R-trees over the same data set, one with a high amount of node overlap, and one with little node overlap. How does this difference affect execution of point queries?

Bokmål:

Anta to R-trær over det samme datasettet, ett med mye overlapp (node overlap) og ett med lite overlapp. Hvordan påvirker denne forskjellen eksekveringen av punktspørringer (point queries)?

Nynorsk:

Anta to R-trær over det same datasettet, eitt med mykje overlapp (node overlap) og eitt med lite overlapp. Korleis påverkar denne skilnaden eksekveringa av punktspørjingar (point queries)?

Answer:

In the R-tree with little overlap, point selects will follow few paths down to leaf nodes. In the R-tree with a high amount of node overlap, point selects will have to follow more paths to leaves, in general reading more data to produce the same result.