Database Systems Architecture
Q&A session - 10 december 2013

Stijn Vansummeren

General Course Information

Objective:

To obtain insight into the internal operation and implementation of database systems.

- Storage management
- Query processing
- Transaction management

General Course Information

What you may expect on the exam (1/2)

Upon successful completion of this course, the student should master the following competences:

1. Translating a given SQL expression into the Relational Algebra
2. Improving a relational algebra expression by, where possible, removing redundant joins in select-project-join subexpressions
3. Improving a relational algebra expression by, where possible, (a) replacing cartesian products by joins; and (b) pushing selections and projections
4. Describing and being able to implement traditional secondary-memory index structures (BTrees, Hashing)
5. Being able to describe and demonstrate the shortcomings of traditional index structures with respect to multi-dimensional search keys. In addition, explaining the studied multi-dimensional indexes by means of an example
6. Describing the most important implementation algorithms (one-pass, sorting, hashing, index) for each of the relational algebra operators, as well as judging the cost of each operator, and knowing their limitations of applicability

General Course Information

What you may expect on the exam (2/2)

Upon successful completion of this course, the student should master the following competences:

7. Given a logical query plan and given base statistics about the size and distributions of the database relations, constructing a heuristically optimal physical query plan, by estimating the sizes of the intermediate results and correspondingly comparing the possible implementations. When joins can be reordered, choosing the order with the least cost.
8. Solving exercises on logging
9. Solving exercises on concurrency control
10. Solving exercises on recoverability
11. Being able to reconstruct the studied proofs

General Course Information

Exam date

- 13 or 14 january?
- VUB students?

Translation of SQL into relational algebra

Question

Can you give an example of a SQL query involving a GROUP BY and an Aggregation of some sort and how it is translated into relational algebra?

Answer. Consider

SQL:
```
SELECT E.Title
FROM Employees E, Department D
WHERE E.did = D.did and D.name = 'IT'
GROUP BY E.Title
HAVING AVG(E.Salary) > 50000
```

Algebra: ???

Translation of SQL into relational algebra

Question

Can you give an example of a SQL query involving a GROUP BY and an Aggregation of some sort and how it is translated into relational algebra?

Answer. Consider

SQL:

```
SELECT E.Title
FROM Emp E, Dept D
WHERE E.did = D.did and D.name = 'IT'
GROUP BY E.Title
HAVING AVG(E.Salary) > 50000
```

Algebra:

$$\pi_{\text{E.Title}} \sigma_{\text{AVG(E.Salary)} > 50000} \gamma_{\text{E.Title,AVG(E.Salary)}} \sigma_{\text{E.did} = \text{D.did} \wedge \text{D.name} = \text{'IT'}} (\rho_E(\text{Emp}) \times \rho_D(\text{Dept}))$$

Optimization of select-project-join expressions

Question

How do you prove that if there is a homomorphism $h\colon Q_2 \to Q_1$ then Q_1 is contained in Q_2?

Optimization of select-project-join expressions

Definition

A conjunctive query is an expression of the form

$$Q\underbrace{(x_1, \ldots, x_n)}_{\text{head}} \leftarrow \underbrace{R(t_1, \ldots, t_m), \ldots, S(t'_1, \ldots, t'_k)}_{\text{body}}$$

Here $t_1, \ldots, t'_k$ denote variables and constants, and $x_1, \ldots, x_n$ must be variables that occur in $t_1, \ldots, t'_k$. We call an expression like $R(t_1, \ldots, t_m)$ an atom. If an atom does not contain any variables, and hence consists solely of contstants, then it is called a fact.

Optimization of select-project-join expressions

Semantics of conjunctive queries

Consider the following toy database D:

R	
1	2
2	3
2	5
6	7
7	5
5	5

S
2
7

as well as the following conjunctive query over the relations $R(A, B)$ and $S(C)$:

$$Q(x, y) \leftarrow R(x, y),\ R(y, 5),\ S(y).$$

Intuitively, Q wants to retrieve all pairs of values (x, y) such that (1) this pair occurs in relation R; (2) y occurs together with the constant 5 in a tuple in R; and (3) y occurs as a value in S. The formal definition is as follows.

Optimization of select-project-join expressions

Semantics of conjunctive queries

Consider the following toy database D:

$$R \quad \boxed{\begin{array}{cc} 1 & 2 \\ 2 & 3 \\ 2 & 5 \\ 6 & 7 \\ 7 & 5 \\ 5 & 5 \end{array}} \qquad S \quad \boxed{\begin{array}{c} 2 \\ 7 \end{array}}$$

as well as the following conjunctive query over the relations $R(A, B)$ and $S(C)$:

$$Q(x, y) \leftarrow R(x, y),\ R(y, 5),\ S(y).$$

A substitution f of Q into D is a function that maps variables in Q to constants in D. For example:

$$f\colon \begin{array}{ccc} x & \mapsto & 1 \\ y & & 2 \end{array}$$

Optimization of select-project-join expressions

Semantics of conjunctive queries

Consider the following toy database D:

R	
1	2
2	3
2	5
6	7
7	5
5	5

S
2
7

as well as the following conjunctive query over the relations $R(A, B)$ and $S(C)$:

$$Q(x, y) \leftarrow R(x, y),\ R(y, 5),\ S(y).$$

A matching is a substitution that maps the body of Q into facts in D. For example:

$$f\colon \begin{array}{ccc} x & \mapsto & 1 \\ y & & 2 \end{array}$$

Optimization of select-project-join expressions

Semantics of conjunctive queries

Consider the following toy database D:

R

1	2
2	3
2	5
6	7
7	5
5	5

S

2
7

as well as the following conjunctive query over the relations $R(A,B)$ and $S(C)$:

$$Q(x,y) \leftarrow R(x,y),\ R(y,5),\ S(y).$$

The result of a conjunctive query is obtained by applying all possible matchings to the head of the query. In our example:

$$Q(D) = \{(1,2),(6,7)\}.$$

Optimization of select-project-join expressions

Question

How do you prove that if there is a homomorphism $h\colon Q_2 \to Q_1$ then Q_1 is contained in Q_2?

Answer (1/2)

- A homomorphism from Q_2 to Q_1 is a function that maps variables in Q_2 to variables in Q_1 such that $h(\textit{body}_2) \subseteq \textit{body}_1$ and $h(\textit{head}_2) = \textit{head}_1$.
- To prove that $Q_1 \subseteq Q_2$ we need to show that $Q_1(D) \subseteq Q_2(D)$ for every database D.
- So, let D be an arbitrary database. Fix an arbitrary tuple $t \in Q_1(D)$. We have to prove that $t \in Q_2(D)$.
- Since $t \in Q_1(D)$ we know by definition of the semantics of conjunctive queries that $t = f(head_1)$, with f a matching of Q_1 into D.
- Now consider the composition $f \circ h$ of f with h. Clearly, this is a substitution of Q_2 into D.

Translation of SQL into relational algebra

Question

How do you prove that if there is a homomorphism $h\colon Q_2 \to Q_1$ then Q_1 is contained in Q_2?

Answer (2/2)

- Because h is a homomorphism we know that $h(body_2) \subseteq body_1$.
- Consequently $f(h(body_2)) \subseteq f(body_1) \subseteq D$.
- In other words, $f \circ h$ is a matching of Q_2 into D, and hence $f(h(head_2)) \in Q_2(D)$.
- As such, $t = f(head_1) = f(h(head_2)) \in Q_2(D)$, as desired.

Optimization of select-project-join expressions

Containment of conjunctive queries is decidable

$$A(x,y) \leftarrow R(x,w),\ G(w,z),\ R(z,y)$$
$$B(x,y) \leftarrow R(x,w),\ G(w,w),\ R(w,y)$$

Golden method to check whether $B \subseteq A$:

1. First calculate the canonical database D for B:

R

x	w
w	y

G

w	w

2. Then check whether $(x,y) \in A(D)$. If so, $B \subseteq A$, otherwise $B \not\subseteq A$.

Optimization of select-project-join expressions

Containment of conjunctive queries is decidable

$$A(x, y) \leftarrow R(x, w),\ G(w, z),\ R(z, y)$$
$$B(x, y) \leftarrow R(x, w),\ G(w, w),\ R(w, y)$$

Fact: $B \subseteq A \Leftrightarrow (x, y) \in A(D)$ with D the canonical database for B.

First possibility: $(x, y) \notin A(D)$

In this case we have just constructed a counter-example because $(x, y) \in B(D)$.

R

x	w
w	y

G

w	w

Optimization of select-project-join expressions

Containment of conjunctive queries is decidable

$$A(x, y) \leftarrow R(x, w),\ G(w, z),\ R(z, y)$$
$$B(x, y) \leftarrow R(x, w),\ G(w, w),\ R(w, y)$$

Fact: $B \subseteq A \Leftrightarrow (x, y) \in A(D)$ with D the canonical database for B.

Second possibility: $(x, y) \in A(D)$

- But this establishes a homomorphism from A to B. And thus $B \subseteq A$.

Indexing

Questions

- Kindly re-explain when we have to recreate the root if inserting in a B-tree. Similarly for deletion.

Answer (1/2)

We will have to create a new root block when:

1. The root block is full and
2. We insert a new record to the BTtree, and due to the recursive way that we insert records to the BTree, it is the case that we need to add a new record to the root. Hence, we split the root block in two and distribute its keys over these two blocks. A new root is created that contains pointers to these two blocks.

(See also indexing slides)

Indexing

Questions

- Kindly re-explain when we have to recreate the root if inserting in a B-tree. Similarly for deletion.

Answer (2/2)

We will have to delete a root block when:

1. The root is itself not a leaf.
2. The root block contains only two pointers. And
3. We delete a record from BTtree, and due to the recursive way that we delete records from the BTree, it is the case that we need to remove one pointer from the root — causing it to have only one pointer left. Then block that this pointer points to becomes the new root.

(See also indexing slides)

Indexing

Questions

- Could you explain linear hashing again?

Answer

See indexing slides.

Physical Operators

Questions

- Could you explain hash-based set union again?

Answer

See following slides + blackboard.

Physical Operators

Hash-based set union

We can also alternatively compute the set union $R \cup_S S$ as follows (R and S are assumed to be sets, and we assume that $B(R) \leq B(S)$):

1. Partition, by means of hash function(s), R in buckets of at most $M - 1$ blocks each. Let k be the resulting number of buckets, and let R_i be the relation formed by the records in bucket i.
2. Partition, by means of the same hash function(s) as above, S in k buckets. Let S_i be the relation formed by the records in bucket i.
   Observe: the records in R_i and S_i have the same hash value! A record t hence occurs in both R and S if, and only if, there is a bucket i such that t occurs in both R_i and S_i.
3. We can hence compute the set union by calculating the set union of R_i and S_i, for every $i \in 1, \ldots, k$. Since every R_i contains at most $M - 1$ blocks, we can do so using the one-pass algorithm.

Note: in contrast to the sort-based set union, the output of a hash-based set union is unsorted!

Physical Operators

Hash-based set union

How do we partition R in buckets of at most $M-1$ blocks?

1. Using $M-1$ buffers, we first hash R into $M-1$ buckets.
2. Subsequently we partition each bucket separately in $M-1$ new buckets, by using a new hash function distinct from the one used in the previous step (why?)
3. We continue doing so until the obtained buckets consists of at most $M-1$ blocks.

Cost-Based Plan Selection

Question

- Could you give summarize when the presence of an index makes a difference w.r.t. cost-based plan selection.

Answer (1/6)

An index may be beneficial for selections, where it can be used instead of a table scan.

Example Table $R(A \text{ int}, B \text{ int})$ with 10^6 records, clustered index on A. We can fit 1000 records in a block. $V(R, A) = 1000$.

What is the best way to evaluate $\sigma_{A=10}(R)$?

Cost-Based Plan Selection

Question

- Could you give summarize when the presence of an index makes a difference w.r.t. cost-based plan selection.

Answer (2/6)

An index may be beneficial for selections, where it can be used instead of a table scan.

Example Table $R(A \text{ int}, B \text{ int})$ with 10^6 records, unclustered index on A. We can fit 1000 records in a block. $V(R, A) = 1000$.

What is the best way to evaluate $\sigma_{A=10}(R)$?

Cost-Based Plan Selection

Question

- Could you give summarize when the presence of an index makes a difference w.r.t. cost-based plan selection.

Answer (3/6)

An index may be beneficial for selections, where it can be used instead of a table scan.

Example Table $R(A\ \text{int}, B\ \text{int})$ with 10^6 records, clustered index on A. We can fit 1000 records in a block. $V(R, A) = 1000$.

What is the best way to evaluate $\sigma_{A>10}(R)$?

Cost-Based Plan Selection

Question

- Could you give summarize when the presence of an index makes a difference w.r.t. cost-based plan selection.

Answer (4/6)

An index may be beneficial for selections, where it can be used instead of a table scan.

Example Table $R(A \text{ int}, B \text{ int})$ with 10^6 records, clustered BTree index on A. We can fit 1000 records in a block. $V(R, A) = 1000$.

What is the best way to evaluate $\sigma_{A>500}(R)$?

Cost-Based Plan Selection

Question

- Could you give summarize when the presence of an index makes a difference w.r.t. cost-based plan selection.

Answer (5/6)

An index may be beneficial for joins, where it can be used as a more efficient alternative for the one-pass join, provided that the relation by which we join is very small.

Example Table $R(A \text{ int}, B \text{ int})$ with 10^6 records, clustered BTree index on A. Table $S(A \text{ int}, C \text{ int})$ with 10 records. We can fit 1000 records (R or S) in a block. $V(R, A) = 1000$. We have 5 buffers available.

What is the best way to evaluate $R \bowtie S$?

Cost-Based Plan Selection

Question

- Could you give summarize when the presence of an index makes a difference w.r.t. cost-based plan selection.

Answer (6/6)

An index may be beneficial for joins, where it can be used as a more efficient alternative for the one-pass join, provided that the relation by which we join is very small.

Example Table $R(A \text{ int}, B \text{ int})$ with 10^6 records, unclustered index on A. Table $S(A \text{ int}, C \text{ int})$ with 10 records. We can fit 1000 records (R or S) in a block. $V(R, A) = 1000$. We have 5 buffers available.

What is the best way to evaluate $R \bowtie S$?

Logging

Question

- Could you summarize the different logging methods and contrast their strengths and weaknesses?

Undo Logging Rules

Undo 1:

If transaction T modifies the database element X that held value $\textsc{old}$

- Write $\langle T, X, \textsc{old} \rangle$ to the log
- We are only allowed to write the new value for X to disk if the corresponding log record has already been written to disk.

Undo 2:

If transaction T commits, then

- Write all pages with modified database elements to disk
- Then, write $\langle \text{COMMIT}\, T \rangle$ to the log and disk, as soon as possible.

Inconvenience of undo logging:

Latency: all modified elements must be flushed to disk before a user is notified that the transaction has committed. (If the commit record is not yet flushed, there is the possibility that we undo it in case of a crash.)

Redo Logging Rules

Redo 1:

If transaction T modifies the database element X setting its value to $\textsc{new}$

- Write $\langle T, X, \textsc{new} \rangle$ to the log

Redo 2:

If transaction T commits, then

- Write $\langle \text{COMMIT}\, T \rangle$ to the log, and flush the log to the disk.
- Only then, write the new value for X to disk.

Hence, all log entries must be written to disk, before modifying any database element on disk.

Inconvenience

No need to wait to tell the user that the transaction has committed. However, if a transaction is very large, we may not have enough memory to keep all modified elements in memory.

Undo/Redo Logging Rules

Undo/Redo 1:

- Write $\langle T, X, \text{OLD}, \text{NEW} \rangle$ to the log if transaction T modifies database element X that held the value OLD to the value NEW
- Log records must be flushed to disk before corresponding modified pages are written to disk.
- When the transaction commits, write $\langle \text{COMMIT}\, T \rangle$.
- Modified database pages can be written to disk before or after the corresponding commit $\langle \text{COMMIT}\, T \rangle$ record, (but after their corresponding log record!)

Inconvenience

No latency; no increased memory usage; more I/O during recovery.

Logging

Question

- What is the purpose of checkpointing?

Answer

- Checkpointing gives a means to ensure that we can throw away a part of the log (i.e. some part before the last successfully complete checkpoint). If we did not have this, we would have ever-growing logs, which is not practical.