Tutorial 5: Query Optimization

Implementation of Databases (DBS2)
Arik Ermshaus

Tutorial Appointments

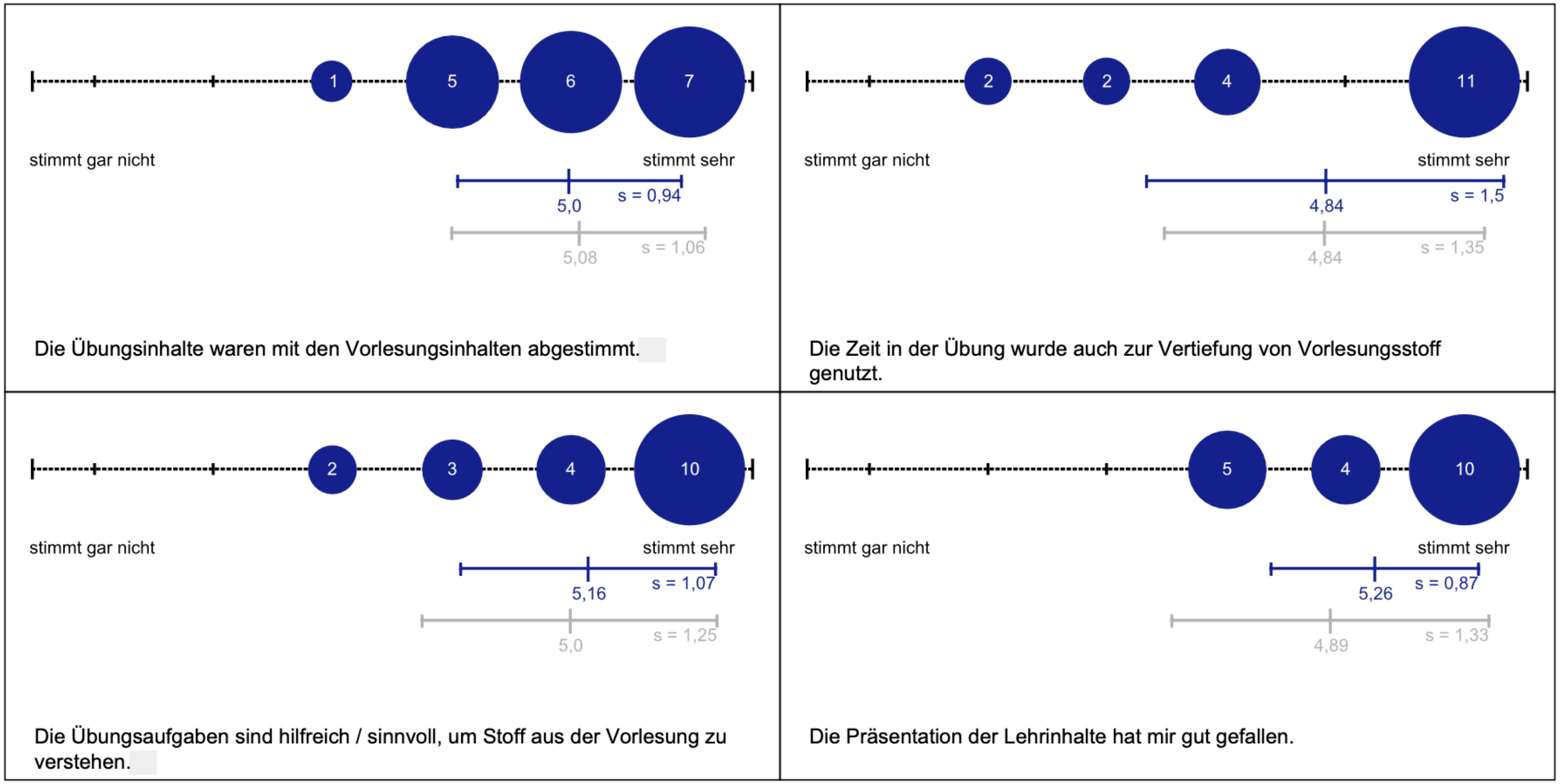
Week	Topic
16.10 - 20.10	-
23.10 - 27.10	Organisation, Exercise Sheet 1
30.10 - 03.11	Q&A
06.11 - 10.11	Q&A
13.11 - 17.11	Exercise Sheet 2
20.11 - 24.11	Q&A
27.11 - 01.12	Q&A
04.12 - 08.12	Exercise Sheet 3
11.12 - 15.12	Q&A
18.12 - 22.12	Q&A
25.12 - 29.12	-
01.01 - 05.01	-
08.01 - 12.01	Exercise Sheet 4
15.01 - 19.01	Q&A
22.01 - 26.01	Q&A
29.01 - 02.02	**Exercise Sheet 5**
05.02 - 09.02	Q&A
12.02 - 16.02	Exam preparation

Disclaimer: Timetable is provisional, and will (probably) change!

Table of Contents

- **Lehrevaluation**
- Solutions of Exercise Sheet 4
- Exercise Sheet 5
- Query Optimization
- Joins

Koordinaten Vorlesung und Übung



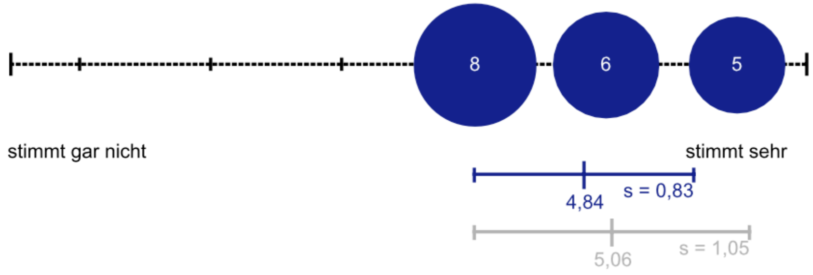
Die Übungsinhalte waren mit den Vorlesungsinhalten abgestimmt.

Die Zeit in der Übung wurde auch zur Vertiefung von Vorlesungsstoff genutzt.

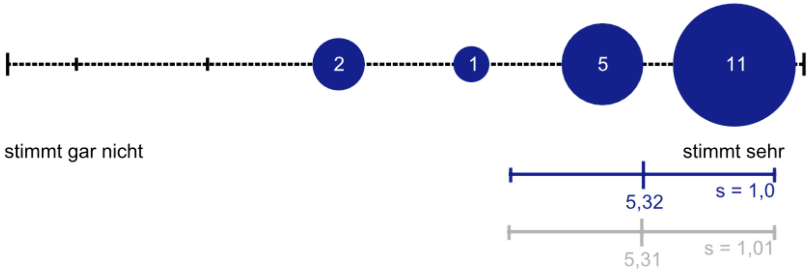
Die Übungsaufgaben sind hilfreich / sinnvoll, um Stoff aus der Vorlesung zu verstehen.

Die Präsentation der Lehrinhalte hat mir gut gefallen.

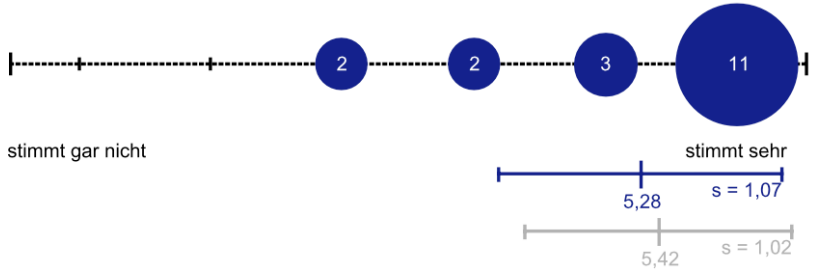
Ausgestaltung der Übung



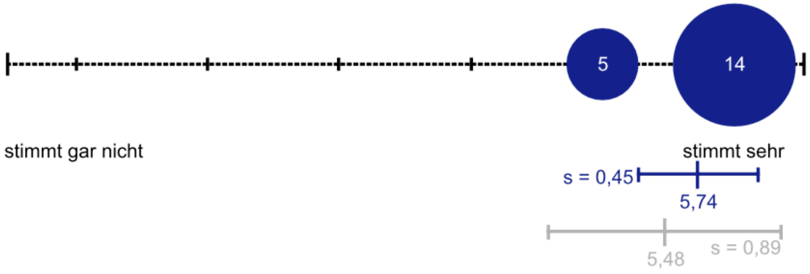
Die Studierenden wurden zur aktiven und selbstständigen Auseinandersetzung mit den Lehrinhalten angeregt.



Der / Dem Lehrenden war es wichtig, dass die Studierenden etwas lernen.

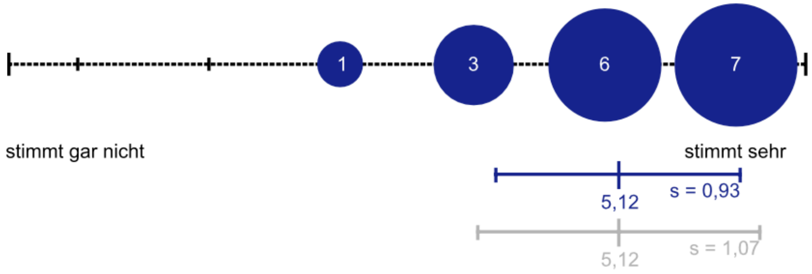


Die / Der Lehrende schuf eine Atmosphäre, in der Studierende Fragen und Kommentare zum Stoff ohne Scheu äußern konnten.

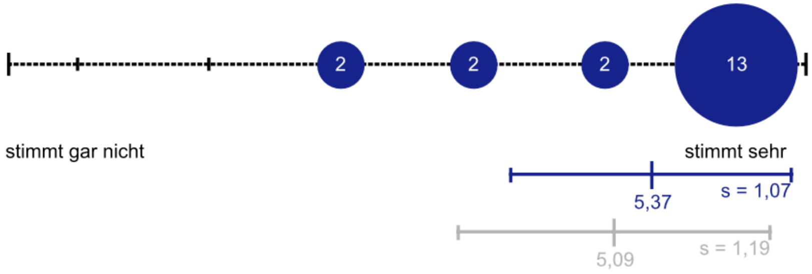


Die / Der Lehrende hat klar und deutlich gesprochen.

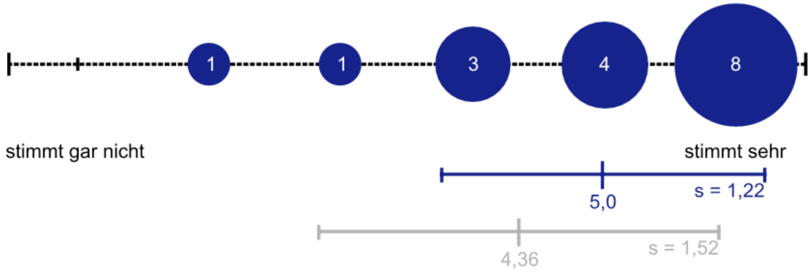
Ausgestaltung der Übung



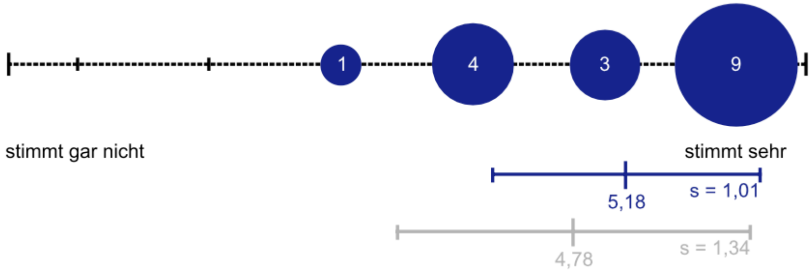
Die / Der Lehrende stellte Verbindungen zu bereits besprochenem Stoff aus der Veranstaltung dar.



Die Lerninhalte wurden hinreichend mit nachvollziehbaren Beispielen veranschaulicht.

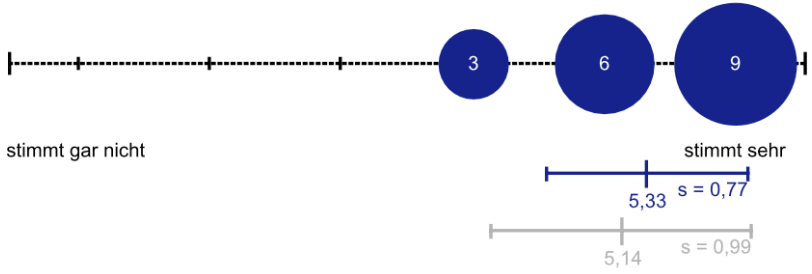


Die Korrektur der Übungsaufgaben war hilfreich.

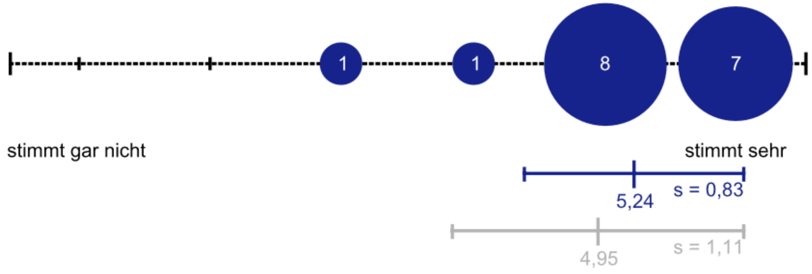


Die veranstaltungsbegleitenden Materialien erleichterten das Verständnis des Inhalts / Stoffes.

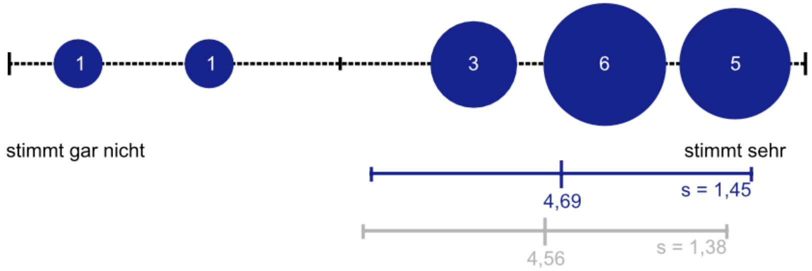
Interaktion in der Übung



Die / Der Lehrende stellt bei ihren / seinen Antworten zu fachlichen Fragen sicher, dass diese verstanden wurden.

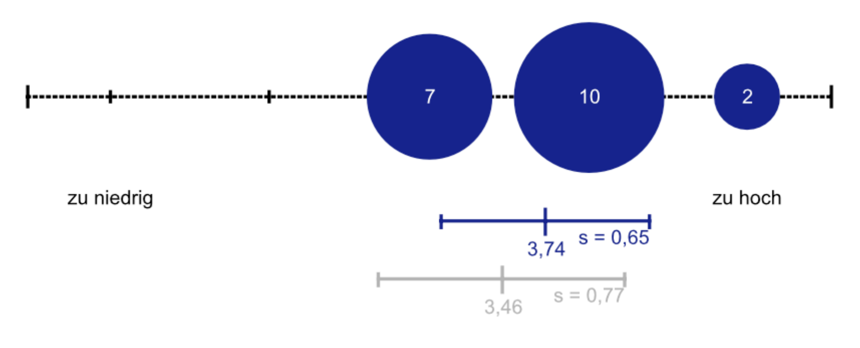


Die / Der Lehrende passte ihre / seine Erklärungen an das Leistungsniveau der Übungsteilnehmer/innen im Rahmen der gegebenen Zeit an.

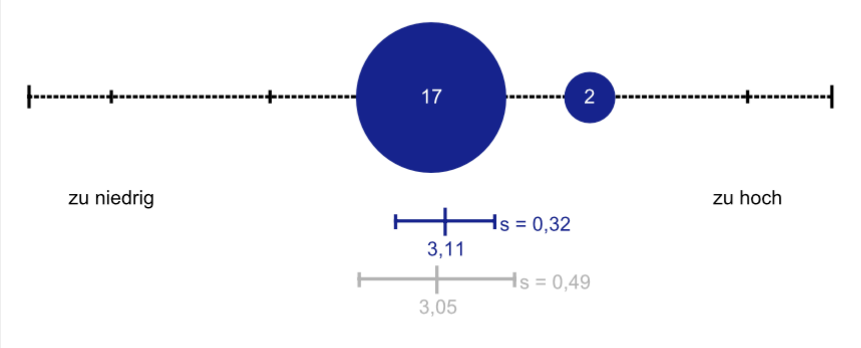


Die / Der Lehrende passte ihre / seine Übungsaufgaben an das Leistungsniveau der Übungsteilnehmer/innen im Rahmen der gegebenen Zeit an.

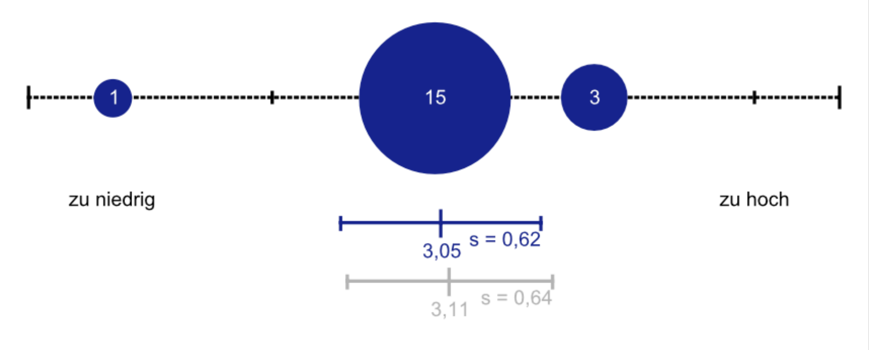
Allgemeiner Eindruck der Übung



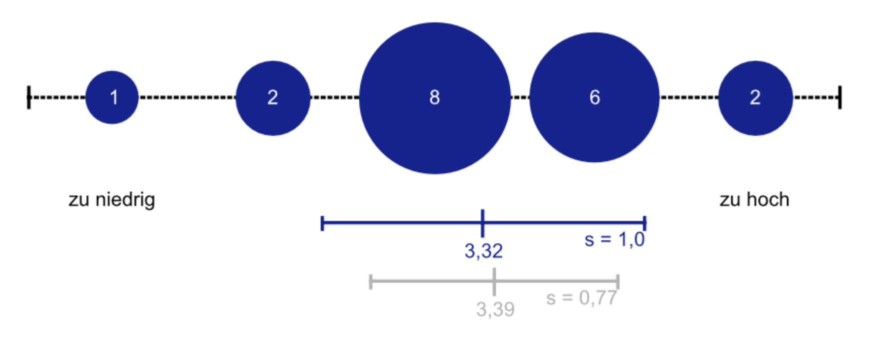
Den Arbeitsaufwand durch die Übung empfand ich als



Den Redeanteil der/des Lehrenden empfand ich als

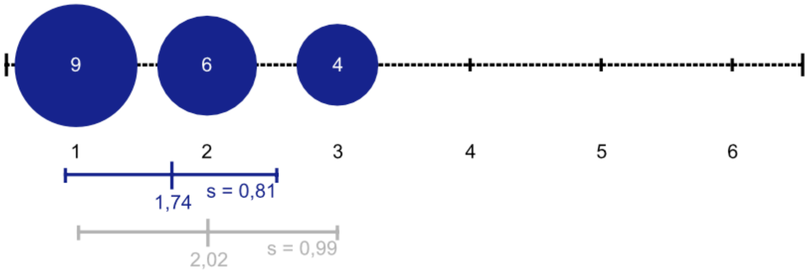


Das Tempo der Veranstaltungen empfand ich als

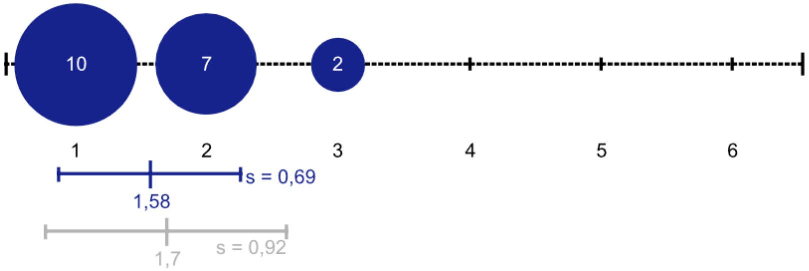


Den Schwierigkeitsgrad der Übungsaufgaben empfand ich als

Gesamteindruck Übung und Lehrender

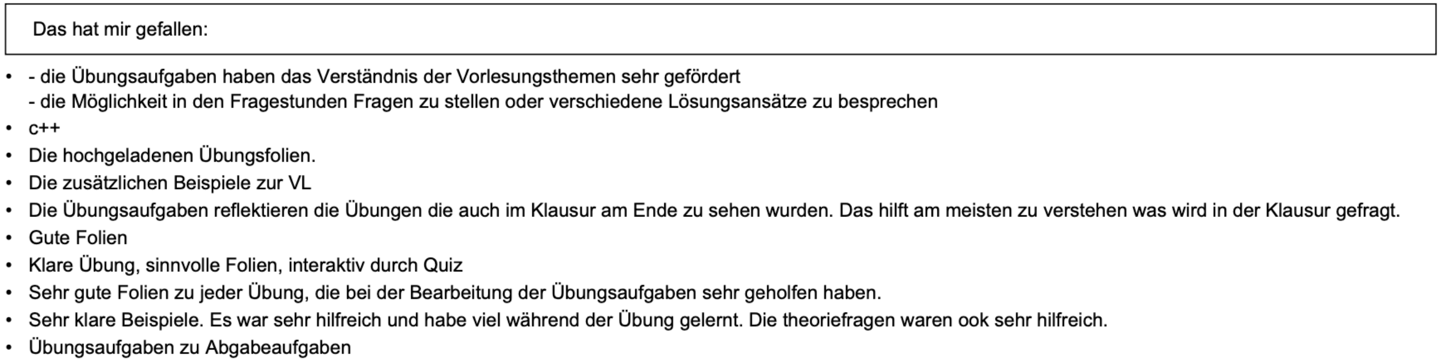


Alles in allem bewerte ich die Übung mit der Schulnote (1 sehr gut, 2 gut, 3 befriedigend, 4 ausreichend, 5 mangelhaft, 6 ungenügend)



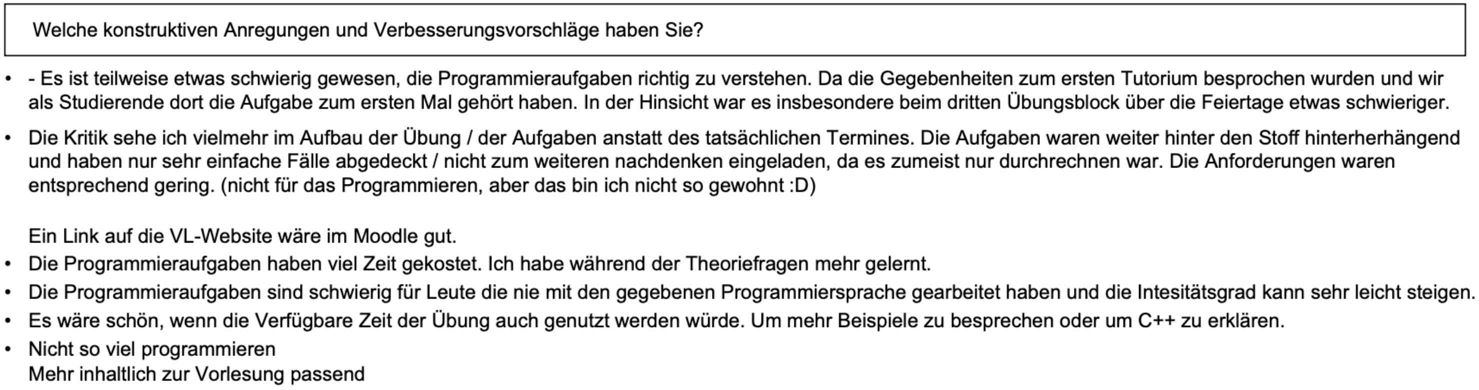
Alles in allem bewerte ich die Leistung der Dozentin / des Dozenten mit der Schulnote (1 sehr gut, 2 gut, 3 befriedigend, 4 ausreichend, 5 mangelhaft, 6 ungenügend)

Freitextkommentare



Das hat mir gefallen:

- - die Übungsaufgaben haben das Verständnis der Vorlesungsthemen sehr gefördert
  - die Möglichkeit in den Fragestunden Fragen zu stellen oder verschiedene Lösungsansätze zu besprechen
- c++
- Die hochgeladenen Übungsfolien.
- Die zusätzlichen Beispiele zur VL
- Die Übungsaufgaben reflektieren die Übungen die auch im Klausur am Ende zu sehen wurden. Das hilft am meisten zu verstehen was wird in der Klausur gefragt.
- Gute Folien
- Klare Übung, sinnvolle Folien, interaktiv durch Quiz
- Sehr gute Folien zu jeder Übung, die bei der Bearbeitung der Übungsaufgaben sehr geholfen haben.
- Sehr klare Beispiele. Es war sehr hilfreich und habe viel während der Übung gelernt. Die theoriefragen waren ook sehr hilfreich.
- Übungsaufgaben zu Abgabeaufgaben



Welche konstruktiven Anregungen und Verbesserungsvorschläge haben Sie?

- - Es ist teilweise etwas schwierig gewesen, die Programmieraufgaben richtig zu verstehen. Da die Gegebenheiten zum ersten Tutorium besprochen wurden und wir als Studierende dort die Aufgabe zum ersten Mal gehört haben. In der Hinsicht war es insbesondere beim dritten Übungsblock über die Feiertage etwas schwieriger.
- Die Kritik sehe ich vielmehr im Aufbau der Übung / der Aufgaben anstatt des tatsächlichen Termines. Die Aufgaben waren weiter hinter den Stoff hinterherhängend und haben nur sehr einfache Fälle abgedeckt / nicht zum weiteren nachdenken eingeladen, da es zumeist nur durchrechnen war. Die Anforderungen waren entsprechend gering. (nicht für das Programmieren, aber das bin ich nicht so gewohnt :D)

  Ein Link auf die VL-Website wäre im Moodle gut.
- Die Programmieraufgaben haben viel Zeit gekostet. Ich habe während der Theoriefragen mehr gelernt.
- Die Programmieraufgaben sind schwierig für Leute die nie mit den gegebenen Programmiersprache gearbeitet haben und die Intesitätsgrad kann sehr leicht steigen.
- Es wäre schön, wenn die Verfügbare Zeit der Übung auch genutzt werden würde. Um mehr Beispiele zu besprechen oder um C++ zu erklären.
- Nicht so viel programmieren
  Mehr inhaltlich zur Vorlesung passend

Table of Contents

- Lehrevaluation
- **Solutions of Exercise Sheet 4**
- Exercise Sheet 5
- Query Optimization
- Joins

Task 1: Multi-dimensional Index Structures

- Scenario: Traffic monitoring system collects master data from cameras
  - Table: external ID, latitude, longitude, tech. specs, installation data
- Query: For camera, retrieve neighbouring ones in partially open-ended area

(a) Which algorithm (Composite Index, Grid File, scanning) is most suitable to answer query?

- Grid File with latitude and longitude
- Created to answer spatial range and nearest neighbour queries
- Can efficiently handle partially-open ranges

Task 1: Multi-dimensional Index Structures

- Scenario: Bookstore has large recent book DB (from 2000 onwards)
  - Table: title, author, publication year, topic, ISBN number
- Query: Books by of topic and publication year, or simply topics

(b) Which algorithm (Composite Index, Grid File, scanning) is most suitable to answer query?

- Composite Index with topic followed by publication year
- Users frequently search indexed combination
- Composite Index can also efficiently retrieve books by topic

Task 1: Multi-dimensional Index Structures

- Scenario: Movie streaming service has large DB with views
    - Table: movie ID, user ID, viewing data, duration, age group
- Query: Sample of 40-60% data points with all attributes

(c) Which algorithm (Composite Index, Grid File, scanning) is most suitable to answer query?

- Scanning over all data points and attributes
- Large portion of data needed
- Index does not offer performance benefits

Task 2: Query Execution Plans

- Relation R(A, B, C, D), 1M tuples, A primary key between 0 … 1M-1
  - Data block: 10 unspanned tuples, completely filled, not sorted
- Search data using one of three execution plans
  - i. Scan table of relation R
  - ii. Search R.A in B+ tree (height = 3, 100 leaf entries), load retrieved tuples
  - iii. Search R.A in hash blocks (2 per bucket, max. 100 entries, 50% fill degree)
- Data structures exist, no blocks buffered, access costs 1 IO

(a) Which execution plan has fewest number of worst-case IOs to answer query:

“Return all tuples in R”

i. Read all data blocks: $\frac{10^6}{10} = 10^5$ I/O

ii. Traverse to leaf with smallest value, traverse all leaf nodes, read all tuples: $(3-1) + \frac{10^6}{100} + 10^6 = 1.010.002$

iii. Search buckets, traverse hash blocks, read all tuples: $2 \cdot 10.000 + 10^6 = 1.020.000$

- Optimal execution plan is table scan (i)

Task 2: Query Execution Plans

- Relation R(A, B, C, D), 1M tuples, A primary key between 0 … 1M-1
  - Data block: 10 unspanned tuples, completely filled, not sorted
- Search data using one of three execution plans
  1. Scan table of relation R
  2. Search R.A in B+ tree (height = 3, 100 leaf entries), load retrieved tuples
  3. Search R.A in hash blocks (2 per bucket, max. 100 entries, 50% fill degree)
- Data structures exist, no blocks buffered, access costs 1 IO

(b) Which execution plan has fewest number of worst-case IOs to answer query:

“Return all tuples in R, where A < 100”

i. Read all data blocks: $\frac{10^6}{10} = 10^5$ I/O

ii. Traverse to leaf with smallest value, read tuples IDs in block, read all tuples: $(3-1) + \frac{100}{100} + 100 = 103$

iii. Search buckets, traverse 200 hash blocks, read all tuples: $2 \cdot 100 + 100 = 300$

- Optimal execution plan is B+ tree (ii)

Task 2: Query Execution Plans

- Relation R(A, B, C, D), 1M tuples, A primary key between 0 … 1M-1
  - Data block: 10 unspanned tuples, completely filled, not sorted
- Search data using one of three execution plans
  i. Scan table of relation R
  ii. Search R.A in B+ tree (height = 3, 100 leaf entries), load retrieved tuples
  iii. Search R.A in hash blocks (2 per bucket, max. 100 entries, 50% fill degree)
- Data structures exist, no blocks buffered, access costs 1 IO

(c) Which execution plan has fewest number of worst-case IOs to answer query:

"Return all tuples in R, where A = 100"

i. Read all data blocks: $\frac{10^6}{10} = 10^5$ I/O

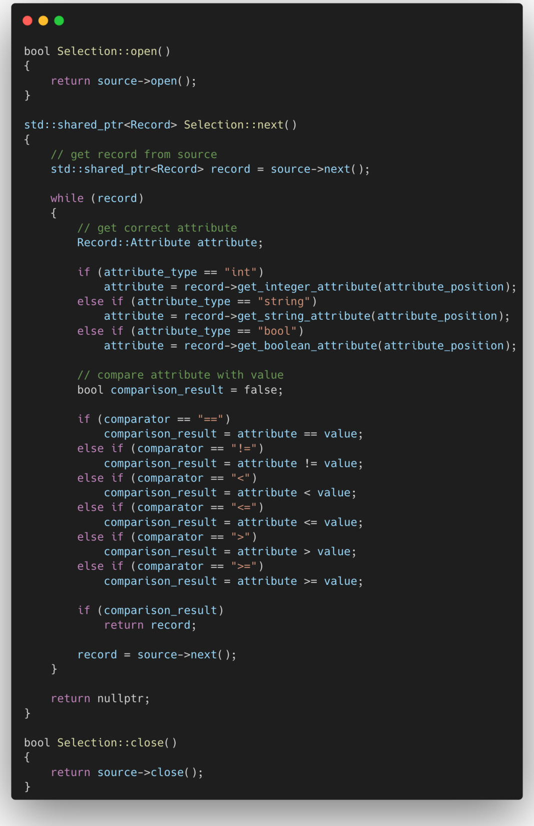
ii. Traverse to leaf with value 100, read tuple ID in block, read tuple: $3 + 1 = 4$

iii. Search bucket, traverse 2 hash blocks, read tuple: $2 + 1 = 3$

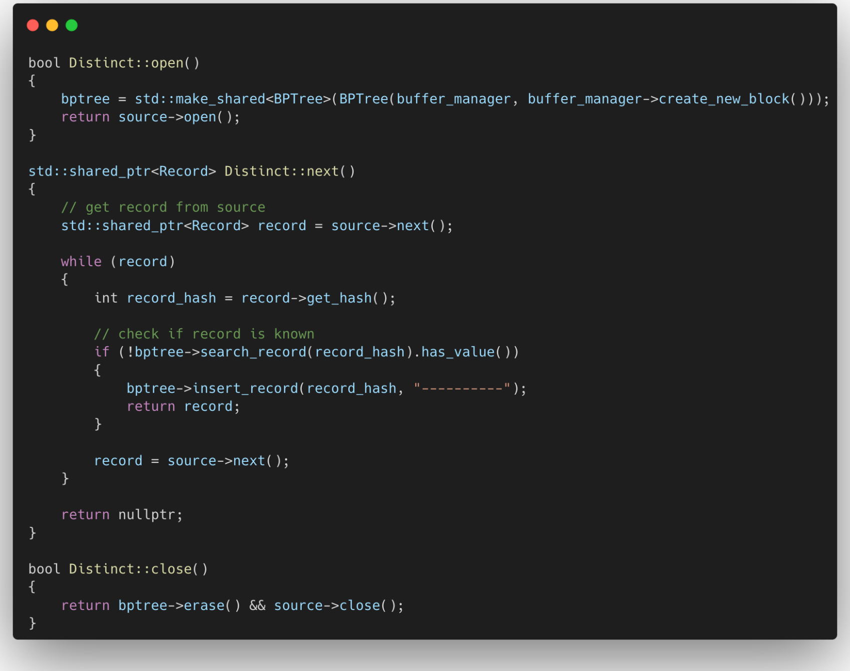
- Optimal execution plan is block hashing (iii)

Task 3: Query Execution

Implement “Selection” and “Distinct” query operators as pipelines.



```cpp
bool Selection::open()
{
    return source->open();
}

std::shared_ptr<Record> Selection::next()
{
    // get record from source
    std::shared_ptr<Record> record = source->next();

    while (record)
    {
        // get correct attribute
        Record::Attribute attribute;

        if (attribute_type == "int")
            attribute = record->get_integer_attribute(attribute_position);
        else if (attribute_type == "string")
            attribute = record->get_string_attribute(attribute_position);
        else if (attribute_type == "bool")
            attribute = record->get_boolean_attribute(attribute_position);

        // compare attribute with value
        bool comparison_result = false;

        if (comparator == "==")
            comparison_result = attribute == value;
        else if (comparator == "!=")
            comparison_result = attribute != value;
        else if (comparator == "<")
            comparison_result = attribute < value;
        else if (comparator == "<=")
            comparison_result = attribute <= value;
        else if (comparator == ">")
            comparison_result = attribute > value;
        else if (comparator == ">=")
            comparison_result = attribute >= value;

        if (comparison_result)
            return record;

        record = source->next();
    }

    return nullptr;
}

bool Selection::close()
{
    return source->close();
}
```



```cpp
bool Distinct::open()
{
    bptree = std::make_shared<BPTree>(BPTree(buffer_manager, buffer_manager->create_new_block()));
    return source->open();
}

std::shared_ptr<Record> Distinct::next()
{
    // get record from source
    std::shared_ptr<Record> record = source->next();

    while (record)
    {
        int record_hash = record->get_hash();

        // check if record is known
        if (!bptree->search_record(record_hash).has_value())
        {
            bptree->insert_record(record_hash, "----------");
            return record;
        }

        record = source->next();
    }

    return nullptr;
}

bool Distinct::close()
{
    return bptree->erase() && source->close();
}
```

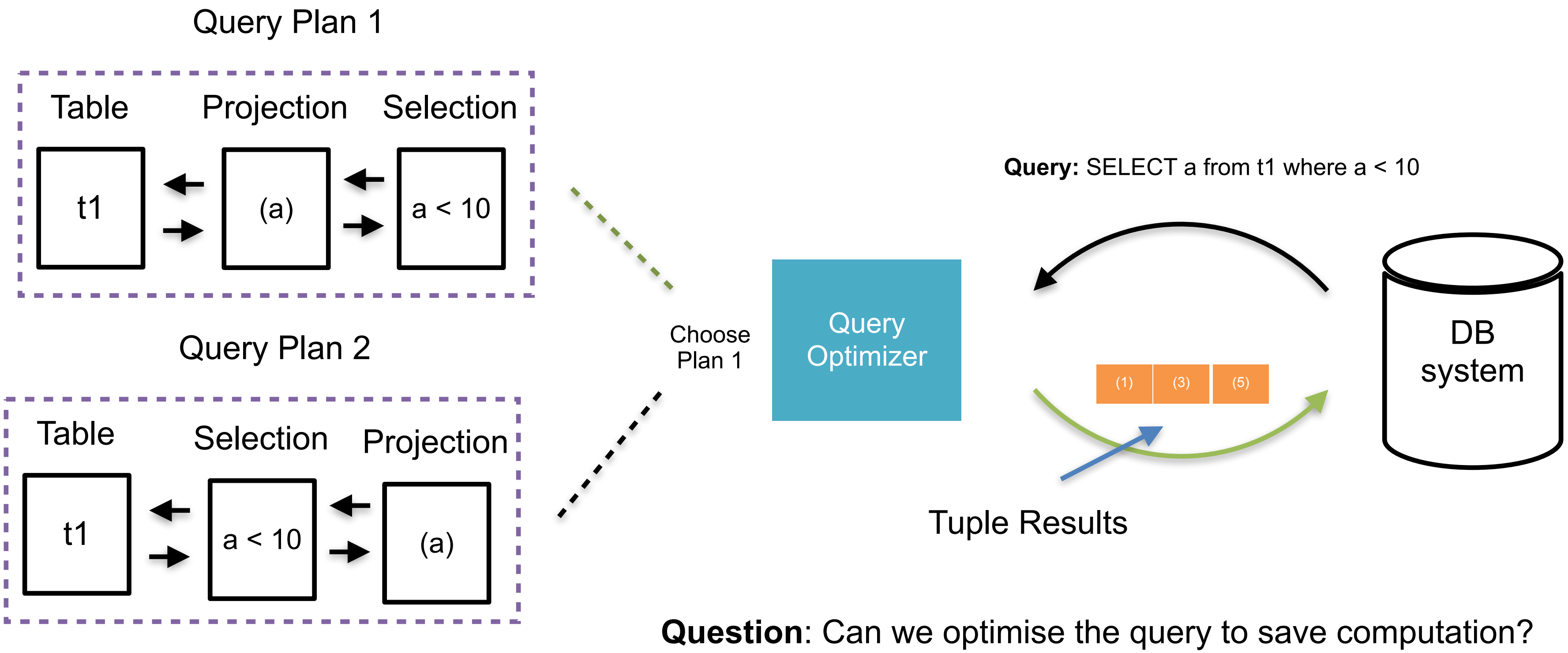
Table of Contents

- Lehrevaluation
- Solutions of Exercise Sheet 4
- **Exercise Sheet 5**
- Query Optimization
- Joins

Table of Contents

- Lehrevaluation
- Solutions of Exercise Sheet 4
- Exercise Sheet 5
- **Query Optimization**
- Joins

Recap: Query Optimization



Question: Can we optimise the query to save computation?

- Task: Find cheapest execution plan to answer query
- Enumerate different semantically equal plans, choose cheapest one
- Trade-Off: Optimization itself costs time

Recap: Query Optimization Techniques

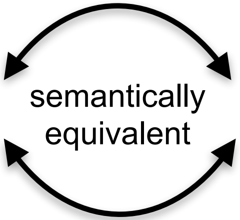
	Rule-based	**Cost-based**
Techniques	Fixed algebraic term rewriting	Query plan cost estimation
Examples	Break and push, merge operations	Uniform cost model, histograms
Advantages ✓	Simple, fast	(unbiased) estimation of optimal plan
Disadvantages ✗	Non-optimal, ignores data	Complex, enumerating all plans impracticable

- Query optimization can be grouped into two approaches
- Different paradigms, both can be combined
- Cost-based typically results in cheaper execution plans

Example: Query Rewriting

SQL Query

```
SELECT R.b, S.a
FROM R, S
WHERE R.a = S.a
AND R.b < S.b
AND R.a >= 10
```

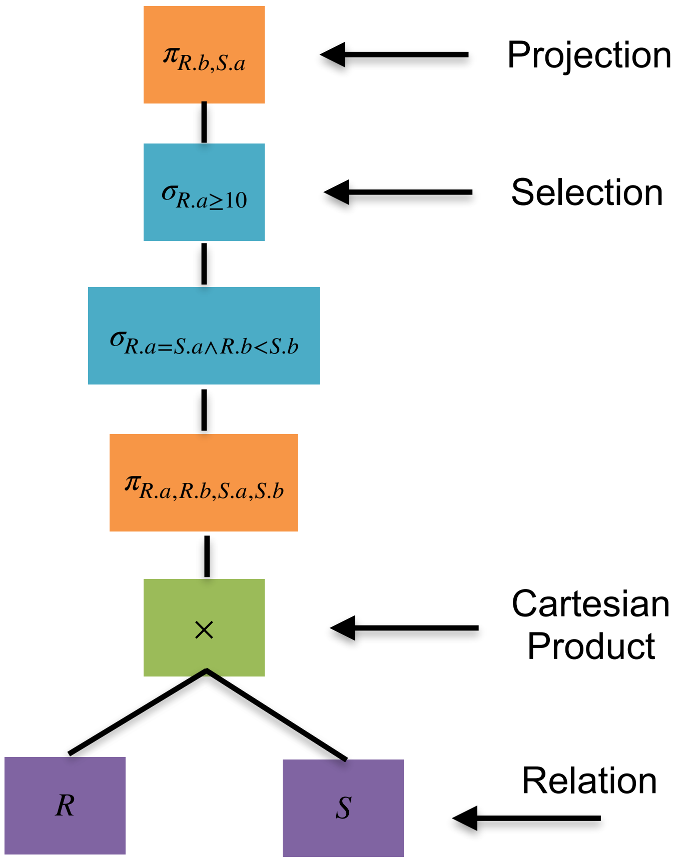

semantically equivalent

Relational Algebra

$$\pi_{R.b,S.a}(\sigma_{R.a \geq 10}(\sigma_{R.a=S.a \wedge R.b<S.b}(\pi_{R.a,R.b,S.a,S.b}(R \times S))))$$

- Queries are rewritten to relational algebra for optimization
- Expressions can be visualised with operator tree

Operator Tree


$\pi_{R.b,S.a}$

Projection

$\sigma_{R.a \geq 10}$

Selection

$\sigma_{R.a=S.a \wedge R.b<S.b}$

$\pi_{R.a,R.b,S.a,S.b}$

$\times$

Cartesian Product

R

S

Relation

Example: Query Minimization

SELECT R.b, S.a **FROM** R, S **WHERE** R.a = S.a
AND R.b < S.b AND R.a >= 10

Relational Algebra

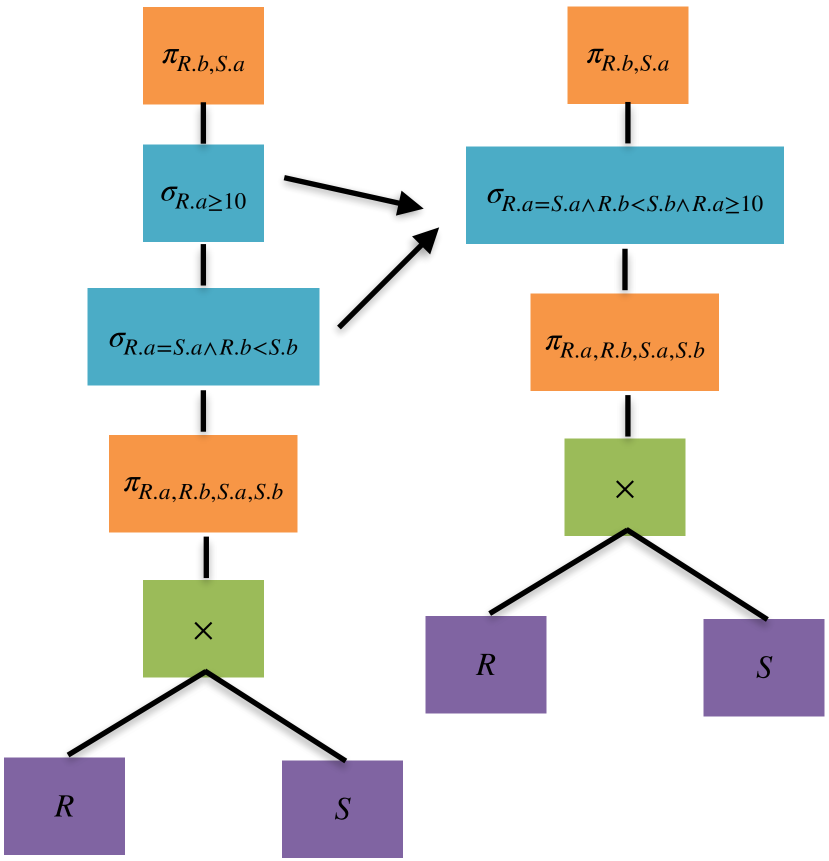
$$\pi_{R.b,S.a}(\sigma_{R.a\geq 10}(\sigma_{R.a=S.a\wedge R.b<S.b}(\pi_{R.a,R.b,S.a,S.b}(R\times S))))$$

Rule 4: Cascading selections

$$\pi_{R.b,S.a}(\sigma_{R.a=S.a\wedge R.b<S.b\wedge R.a\geq 10}(\pi_{R.a,R.b,S.a,S.b}(R\times S)))$$

- Relational algebra expressions can be minimised
- Equivalent expressions: compute same result
- See VL slides on “query optimization” for set of rules

Operator Tree



Example: Query Minimization

SELECT R.b, S.a **FROM** R, S **WHERE** R.a = S.a
AND R.b < S.b AND R.a >= 10

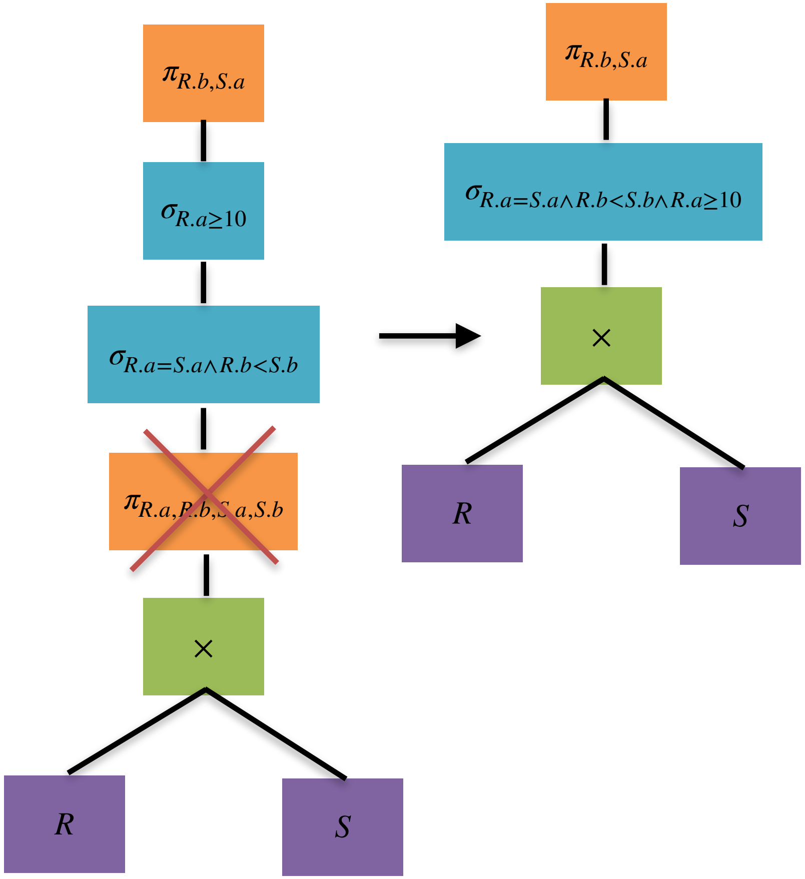
Relational Algebra

$$\pi_{R.b,S.a}(\sigma_{R.a=S.a \wedge R.b<S.b \wedge R.a \geq 10}(\cancel{\pi_{R.a,R.b,S.a,S.b}}(R \times S)))$$

Delete unnecessary projection

$$\pi_{R.b,S.a}(\sigma_{R.a=S.a \wedge R.b<S.b \wedge R.a \geq 10}(R \times S))$$

Operator Tree



$\pi_{R.b,S.a}$
$\sigma_{R.a \geq 10}$
$\sigma_{R.a=S.a \wedge R.b<S.b}$
$\cancel{\pi_{R.a,R.b,S.a,S.b}}$
×
R
S
$\pi_{R.b,S.a}$
$\sigma_{R.a=S.a \wedge R.b<S.b \wedge R.a \geq 10}$
×
R
S

- Relational algebra expressions can be minimised
- Equivalent expressions: compute same result
- See VL slides on “query optimization” for set of rules

Task 1: Query Rewriting

SELECT R.b, S.a **FROM** R, S **WHERE** R.a = S.a
AND R.b < S.b **AND** R.a >= 10

↓

$$\pi_{R.b,S.a}(\sigma_{R.a=S.a \wedge R.b<S.b \wedge R.a \geq 10}(R \times S))$$

- **Question**: Why do we use algebraic expressions for query optimization?

Task 1: Query Rewriting

SELECT R.b, S.a **FROM** R, S **WHERE** R.a = S.a
AND R.b < S.b **AND** R.a >= 10

$$\downarrow$$

$$\pi_{R.b,S.a}(\sigma_{R.a=S.a \wedge R.b<S.b \wedge R.a \geq 10}(R \times S))$$

- **Question**: Why do we use algebraic expressions for query optimization?

Rewriting Rules

Cost Estimation

Assignment to Physical Operators

Task 2: Query Rewriting

$$\pi_{R.b,S.a}(\sigma_{R.a\geq 10}(\sigma_{R.a=S.a \wedge R.b<S.b}(\pi_{R.a,R.b,S.a,S.b}(R \times S))))$$

Rule 4: Cascading selections

$$\pi_{R.b,S.a}(\sigma_{R.a=S.a \wedge R.b<S.b \wedge R.a\geq 10}(\pi_{R.a,R.b,S.a,S.b}(R \times S)))$$

- **Question**: Which of the following conditions must apply for two semantically equivalent algebraic expressions?

(A) Same result

(B) Same operators

(C) Same relations

(D) Same operator order

Task 2: Query Rewriting

$$\pi_{R.b,S.a}(\sigma_{R.a \geq 10}(\sigma_{R.a=S.a \wedge R.b<S.b}(\pi_{R.a,R.b,S.a,S.b}(R \times S))))$$

$$\Big\downarrow \quad \Big\downarrow$$

Rule 4: Cascading selections

$$\pi_{R.b,S.a}(\sigma_{R.a=S.a \wedge R.b<S.b \wedge R.a \geq 10}(\pi_{R.a,R.b,S.a,S.b}(R \times S)))$$

- **Question**: Which of the following conditions must apply for two semantically equivalent algebraic expressions?

[✓ (A) Same result]

(B) Same operators

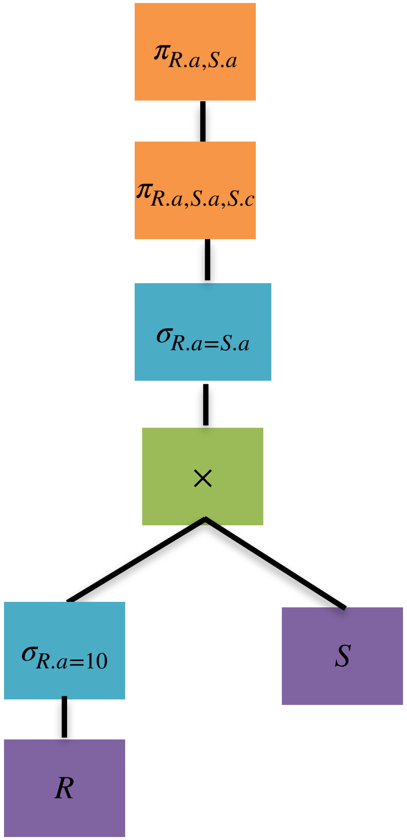
[✓ (C) Same relations]

(D) Same operator order

Task 3: Query Minimization

SELECT R.a, S.a **FROM** R, S **WHERE** R.a = S.a **AND** R.a = 10

$$\pi_{R.a,S.a}(\pi_{R.a,S.a,S.c}(\sigma_{R.a=S.a}(\sigma_{R.a=10}(R) \times S)))$$



- **Question**: What is the minimal number of operators for the above query?

(A) 2

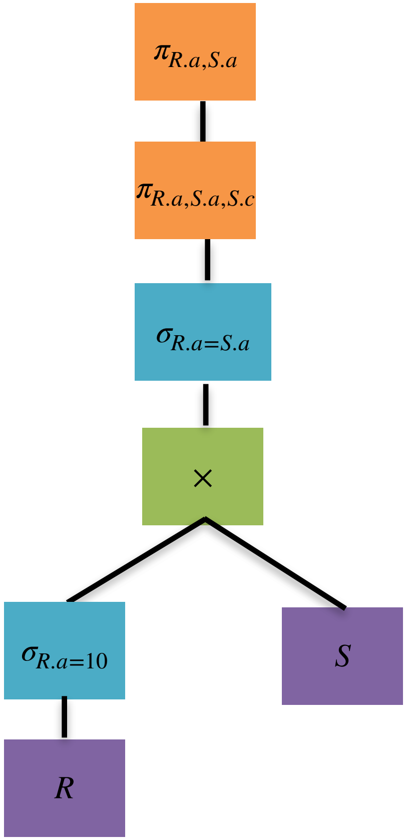
(B) 3

(C) 4

Task 3: Query Minimization

SELECT R.a, S.a **FROM** R, S **WHERE** R.a = S.a **AND** R.a = 10

$$\pi_{R.a,S.a}(\pi_{R.a,S.a,S.c}(\sigma_{R.a=S.a}(\sigma_{R.a=10}(R) \times S)))$$



- **Question**: What is the minimal number of operators for the above query?

(A) 2 [✓ (B) 3] (C) 4

$$\pi_{R.a,S.a}(\sigma_{R.a=10}(R) \bowtie_{R.a=S.a} S))$$

Recap: Query Rewriting

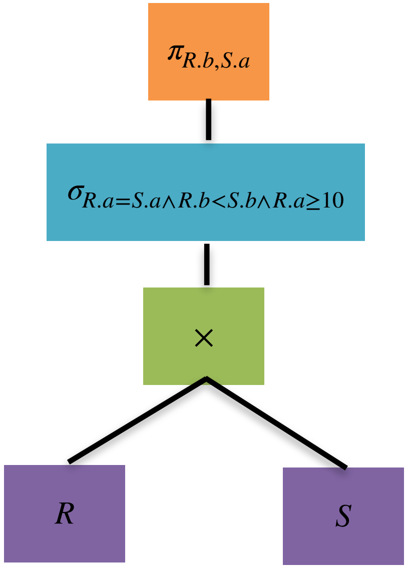
SELECT R.b, S.a **FROM** R, S **WHERE** R.a = S.a
AND R.b < S.b AND R.a >= 10

Relational Algebra

$\pi_{R.b,S.a}(\sigma_{R.a=S.a \wedge R.b<S.b \wedge R.a \geq 10}(R \times S))$

Question: Is this minimal query optimal (cheapest to compute)?

Operator Tree



- Query minimization does not specifically aim at reducing intermediate results
- Many heuristics exist for rewrite strategies
  1. Break selections / projections into atomics, push down the tree
  2. Replace selection + cartesian product with join
  3. Merge neighbouring selections / projections

Example: Rule-based Query Optimization

SELECT R.b, S.a **FROM** R, S **WHERE** R.a = S.a
AND R.b < S.b AND R.a >= 10

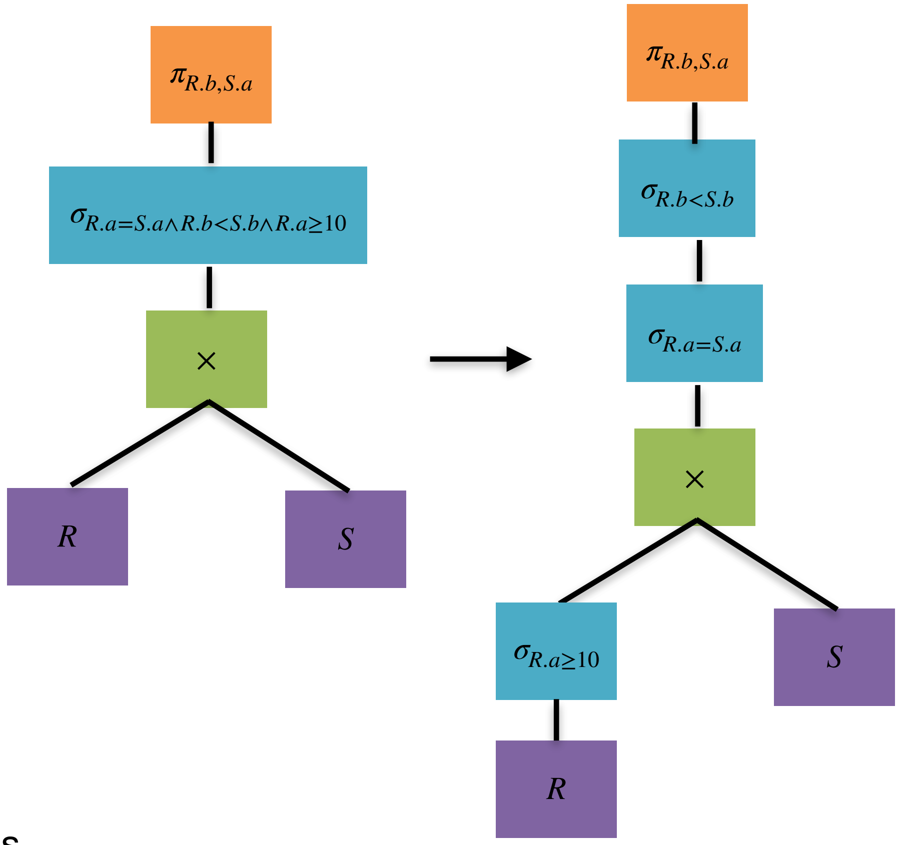
Relational Algebra

$$\pi_{R.b,S.a}(\sigma_{R.a=S.a \wedge R.b<S.b \wedge R.a\geq 10}(R \times S))$$

Break selections / conjunctions into atomics, push down tree

$$\pi_{R.b,S.a}(\sigma_{R.b<S.b}(\sigma_{R.a=S.a}(\sigma_{R.a\geq 10}(R) \times S)))$$

Operator Tree



- Relational algebra expressions can be optimised
- Equivalent expressions: compute same result
- See VL slides on “query optimization” for heuristics

Example: Rule-based Query Optimization

SELECT R.b, S.a **FROM** R, S **WHERE** R.a = S.a
AND R.b < S.b AND R.a >= 10

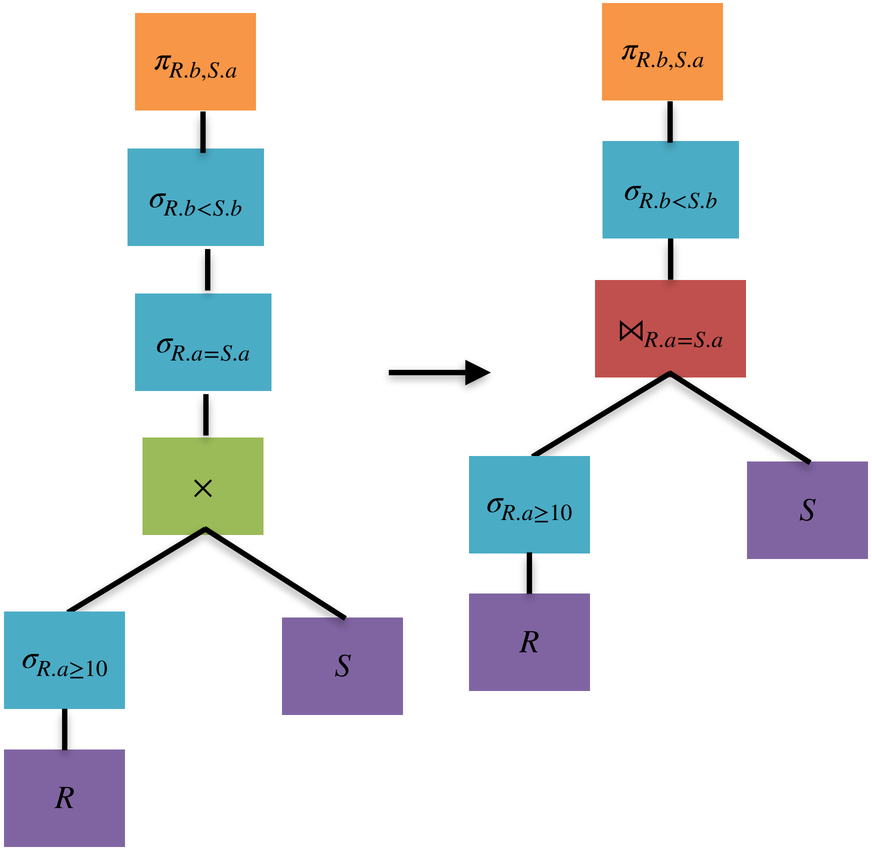
Relational Algebra

$$\pi_{R.b,S.a}(\sigma_{R.b<S.b}(\sigma_{R.a=S.a}(\sigma_{R.a\geq 10}(R) \times S)))$$

Replace selection + cartesian product with join

$$\pi_{R.b,S.a}(\sigma_{R.b<S.b}(\sigma_{R.a\geq 10}(R) \bowtie_{R.a=S.a} S))$$

Operator Tree



- Relational algebra expressions can be optimised
- Equivalent expressions: compute same result
- See VL slides on “query optimization” for heuristics

Example: Cost estimation

Selections:

$$|\sigma_{R.a=10}(R)| = \frac{|R|}{v(R,a)} = \frac{100}{10} = 10$$

$$|\sigma_{R.a>5}(R)| = \frac{|R|}{const} = \frac{100}{3} \approx 33$$

(Assume const = 3)

Conjunctions:

$$|\sigma_{R.a=10 \wedge R.b>20}(R)| = \frac{|R|}{v(R,a) \cdot 3} = \frac{100}{10 \cdot 3} \approx 3$$

$$|\sigma_{R.a=10 \wedge R.a=5}(R)| = 0$$

(contradictory)

Assume: uniform distribution
|R| = 100, v(R,a) = 10, v(R,b) = 5
|S| = 200, v(S,a) = 20, v(S,b) = 10

- Task: estimate size of intermediate results (cardinality)
- Assigns cost to query; makes its efficiency comparable
- Enumerate possible query rewritings, choose cheapest one

Example: Cost estimation

Cartesian Product: $|R \times S| = |R| \cdot |S| = 100 \cdot 200 = 20000$

Join: $$|R \bowtie_{R.b=S.b}| = \frac{|R| \cdot |S|}{max(v(R,b), v(S,b))} = \frac{100 \cdot 200}{max(5,10)} = 2000$$

… and more (see VL slides on cost estimation)

Assume: uniform distribution
|R| = 100, v(R,a) = 10, v(R,b) = 5
|S| = 200, v(S,a) = 20, v(S,b) = 10

- Task: estimate size of intermediate results (cardinality)
- Assigns cost to query; makes its efficiency comparable
- Enumerate possible query rewritings, choose cheapest one

Task 4: Cost estimation

$$\sigma_{S.b=5}(S)$$

Assume: uniform distribution
|R| = 100, v(R,a) = 10, v(R,b) = 5
|S| = 200, v(S,a) = 20, v(S,b) = 10

- **Question**: What is the estimated cost of the above term?

(A) 5

(B) 10

(C) 20

(D) 25

Task 4: Cost estimation

$$\sigma_{S.b=5}(S)$$

Assume: uniform distribution
|R| = 100, v(R,a) = 10, v(R,b) = 5
|S| = 200, v(S,a) = 20, v(S,b) = 10

- **Question**: What is the estimated cost of the above term?

(A) 5 (B) 10 [✓ (C) 20] (D) 25

$$|\sigma_{S.b=5}(S)| = \frac{|S|}{v(S,b)} = \frac{200}{10} = 20$$

Task 5: Cost estimation

$$\sigma_{R.a=5 \wedge R.b=10}(R)$$

Assume: uniform distribution
|R| = 100, v(R,a) = 10, v(R,b) = 5
|S| = 200, v(S,a) = 20, v(S,b) = 10

- **Question**: What is the estimated cost of the above term?

(A) 1

(B) 2

(C) 5

(D) 10

Task 5: Cost estimation

$$\sigma_{R.a=5 \wedge R.b=10}(R)$$

Assume: uniform distribution
|R| = 100, v(R,a) = 10, v(R,b) = 5
|S| = 200, v(S,a) = 20, v(S,b) = 10

- **Question**: What is the estimated cost of the above term?

(A) 1 [✓ (B) 2] (C) 5 (D) 10

$$|\sigma_{R.a=5 \wedge R.b=10}(R)| = \frac{|R|}{v(R,a) \cdot v(R,b)} = \frac{100}{10 \cdot 5} = 2$$

Task 6: Cost estimation

Assume: uniform distribution
|R| = 100, v(R,a) = 10, v(R,b) = 5
|S| = 200, v(S,a) = 20, v(S,b) = 10

- **Question**: Is a uniform distribution realistic to assume for real data? If not, how can cost estimation be improved?

Task 6: Cost estimation

Assume: uniform distribution
|R| = 100, v(R,a) = 10, v(R,b) = 5
|S| = 200, v(S,a) = 20, v(S,b) = 10

- **Question**: Is a uniform distribution realistic to assume for real data? If not, how can cost estimation be improved?

Test for correct distribution

Histograms

Table of Contents

- Lehrevaluation
- Solutions of Exercise Sheet 4
- Exercise Sheet 5
- Query Optimization
- **Joins**

Recap: Joins

Query: SELECT * from t1, t2 where t1.a = t2.a

Table t1

a	b
1	A
3	A
2	B
2	C

$\bowtie_{t1.a=t2.a}$

Table t2

a	c
1	D
4	F
3	A
1	D

→

Join Result

a	b	c
1	A	D
1	A	D
3	A	A

Question: How can we implement joins efficiently?

- Task: Choose cheapest implementation to execute join
- Needed in many use cases, often appears in groups (multiple joins)
- Problems: Runtime intensive to compute, can lead to large intermediate results

Example: Block-Nested Loop Join

Query: SELECT * from t1, t2 where t1.a = t2.a

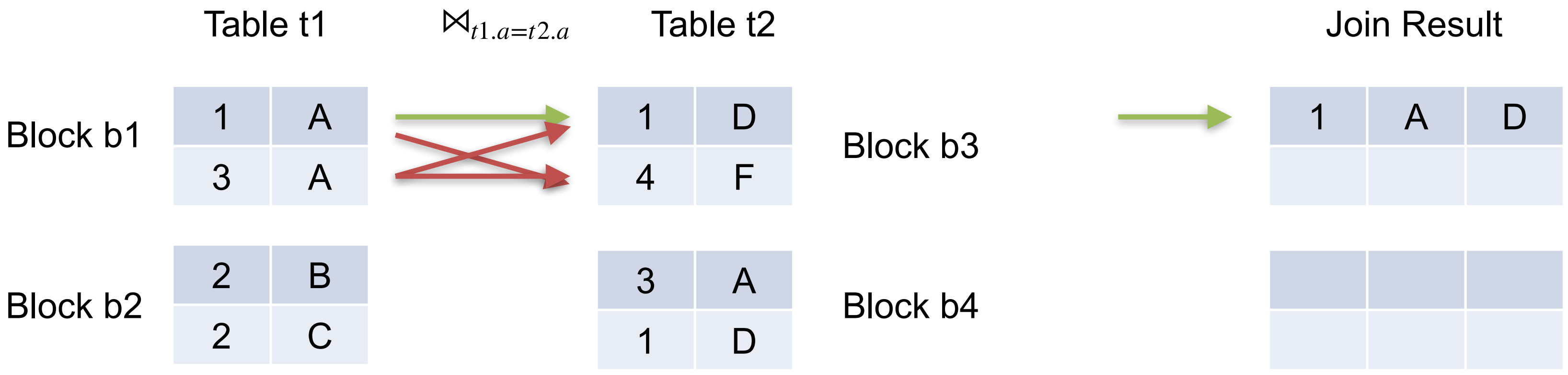


Table t1

Block b1	1	A
	3	A
Block b2	2	B
	2	C

Table t2

1	D	Block b3
4	F	
3	A	Block b4
1	D	

Join Result

1	A	D

Load: Block b1 and b3

1. Load pairs of blocks from each table
2. Evaluate join comparator for each tuple pair in blocks
3. Issue pairs that evaluate true

Example: Block-Nested Loop Join

Query: SELECT * from t1, t2 where t1.a = t2.a

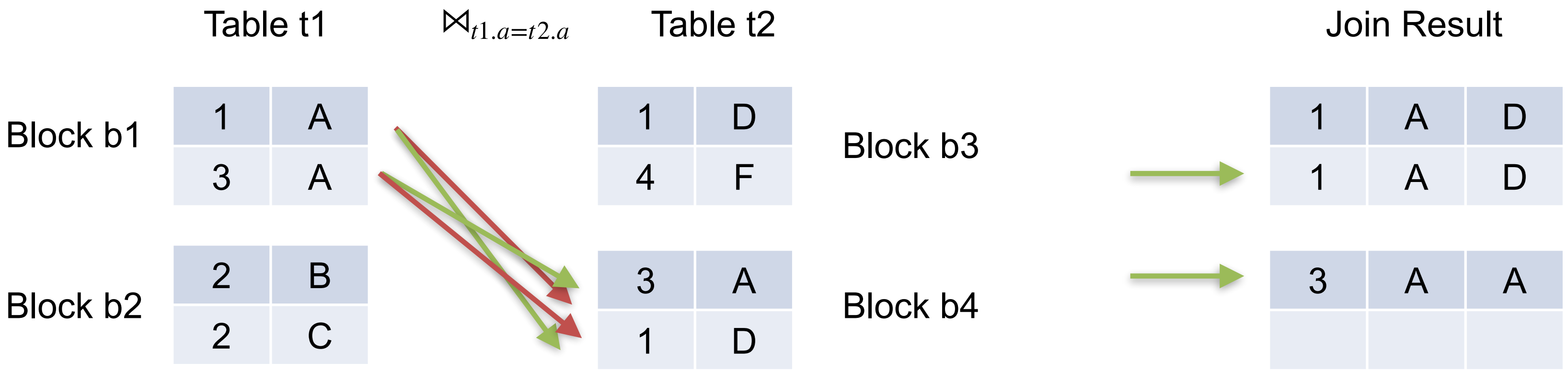


Table t1 $\bowtie_{t1.a=t2.a}$ Table t2

Block b1

1	A
3	A

Block b2

2	B
2	C

Block b3

1	D
4	F

Block b4

3	A
1	D

Join Result

1	A	D
1	A	D
3	A	A

Load: Block b4

1. Load pairs of blocks from each table
2. Evaluate join comparator for each tuple pair in blocks
3. Issue pairs that evaluate true

Example: Block-Nested Loop Join

Query: SELECT * from t1, t2 where t1.a = t2.a

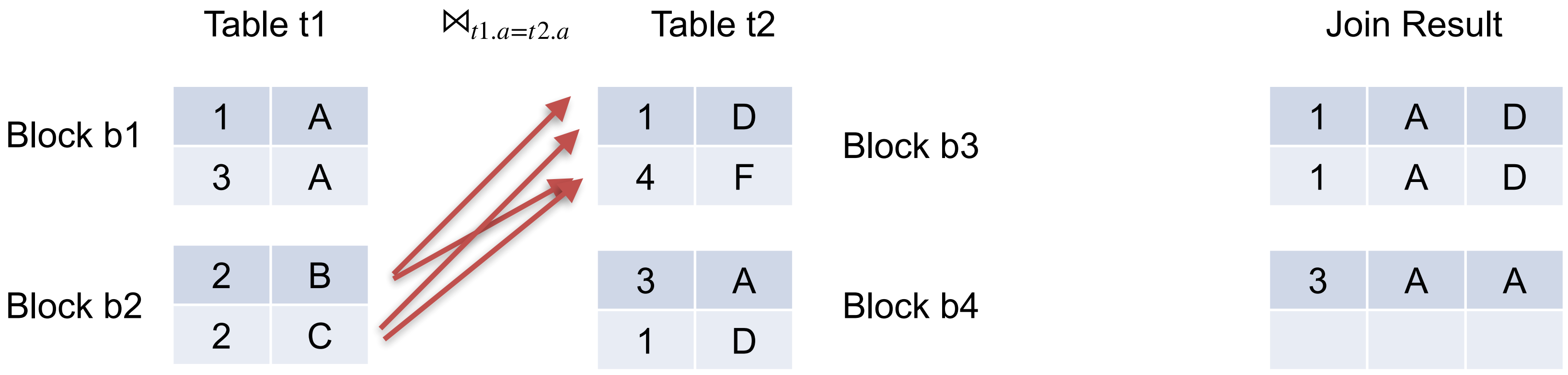


Table t1 $\bowtie_{t1.a=t2.a}$ Table t2

Join Result

Table t1

Block b1	1	A
	3	A
Block b2	2	B
	2	C

Table t2

1	D	Block b3
4	F	
3	A	Block b4
1	D	

Join Result

1	A	D
1	A	D
3	A	A

Load: Block b2 and b3

1. Load pairs of blocks from each table
2. Evaluate join comparator for each tuple pair in blocks
3. Issue pairs that evaluate true

Example: Block-Nested Loop Join

Query: SELECT * from t1, t2 where t1.a = t2.a

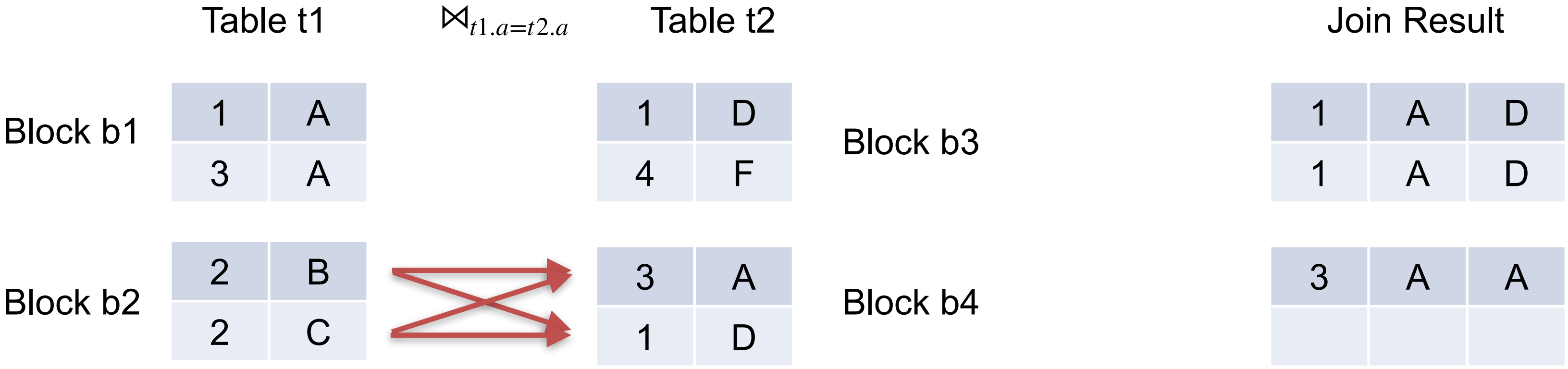


Table t1 $\bowtie_{t1.a=t2.a}$ Table t2

Block b1

1	A
3	A

Block b2

2	B
2	C

Block b3

1	D
4	F

Block b4

3	A
1	D

Join Result

1	A	D
1	A	D
3	A	A

Load: Block b4

1. Load pairs of blocks from each table
2. Evaluate join comparator for each tuple pair in blocks
3. Issue pairs that evaluate true

Example: Sort-Merge Join

Query: SELECT * from t1, t2 where t1.a = t2.a

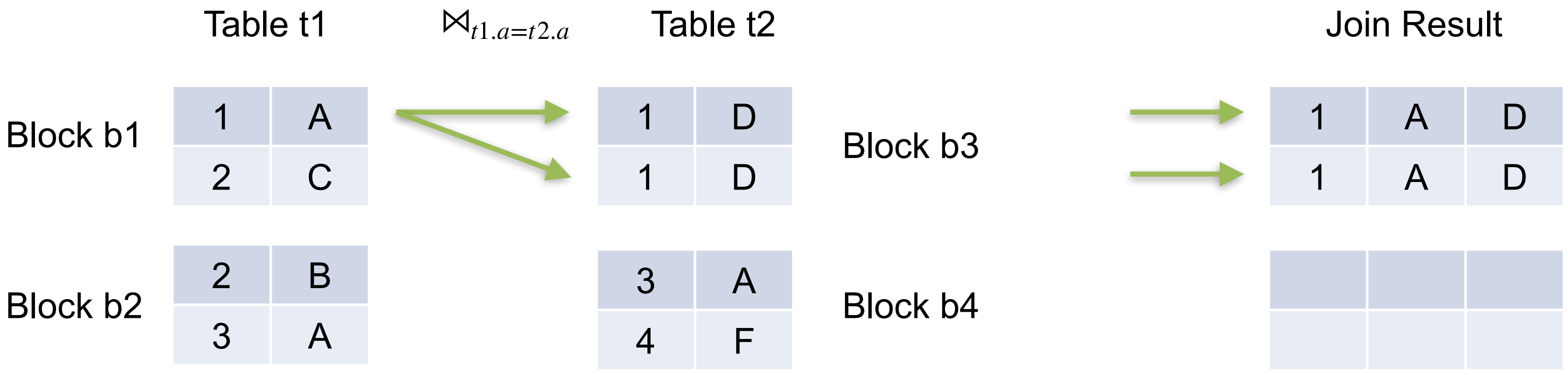


Table t1 $\bowtie_{t1.a=t2.a}$ Table t2

Join Result

Block b1

1	A
2	C

Block b2

2	B
3	A

Block b3

1	D
1	D

Block b4

3	A
4	F

Join Result

1	A	D
1	A	D

Load: Block b1 and b3

1. Sort both tables on join attribute
2. Merge sorted tables from beginning to end
3. Issue pairs that match the comparison

Example: Sort-Merge Join

Query: SELECT * from t1, t2 where t1.a = t2.a

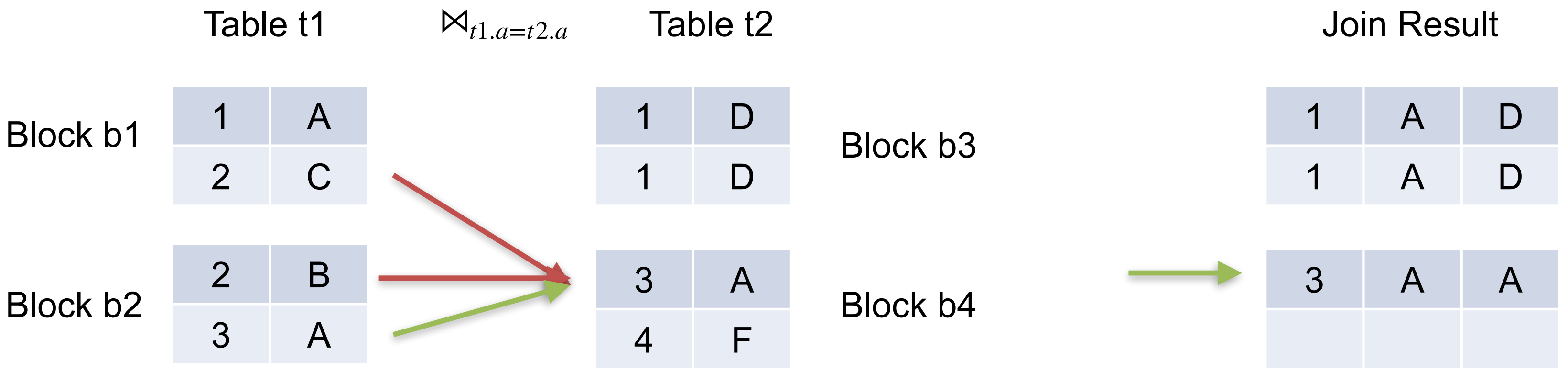


Table t1 $\bowtie_{t1.a=t2.a}$ Table t2

Join Result

Block b1

1	A
2	C

Block b2

2	B
3	A

Block b3

1	D
1	D

Block b4

3	A
4	F

Join Result

1	A	D
1	A	D
3	A	A

Load: Block b2 and b4

1. Sort both tables on join attribute
2. Traverse sorted tables from beginning to end
3. Issue pairs that evaluate true

Example: Sort-Merge Join

Query: SELECT * from t1, t2 where t1.a = t2.a

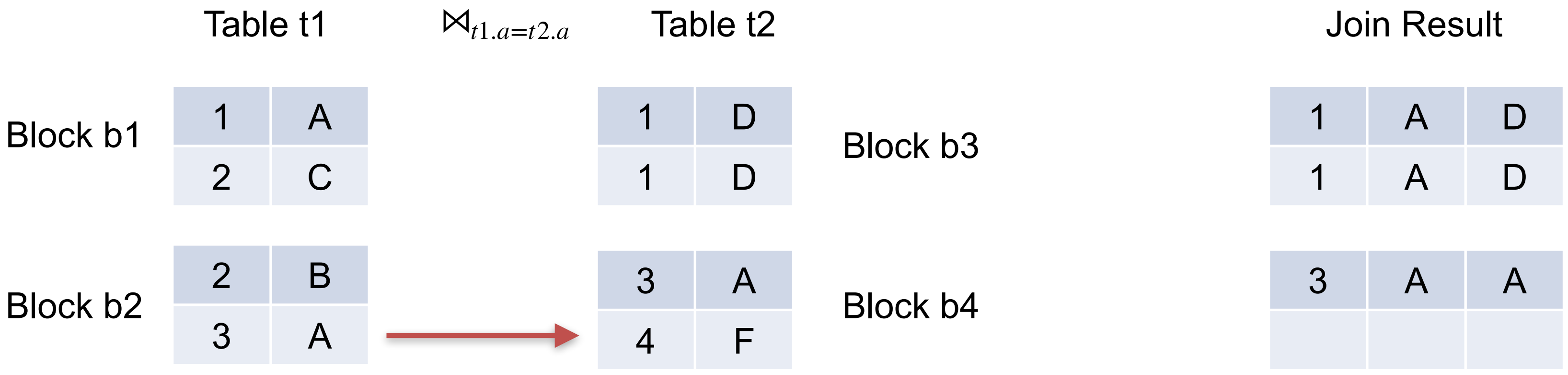


Table t1 $\bowtie_{t1.a=t2.a}$ Table t2

Table t1

Block b1	1	A
	2	C
Block b2	2	B
	3	A

Table t2

1	D	Block b3
1	D	
3	A	Block b4
4	F	

Join Result

1	A	D
1	A	D
3	A	A

1. Sort both tables on join attribute
2. Traverse sorted tables from beginning to end
3. Issue pairs that evaluate true

Example: Hash Join

Query: SELECT * from t1, t2 where t1.a = t2.a

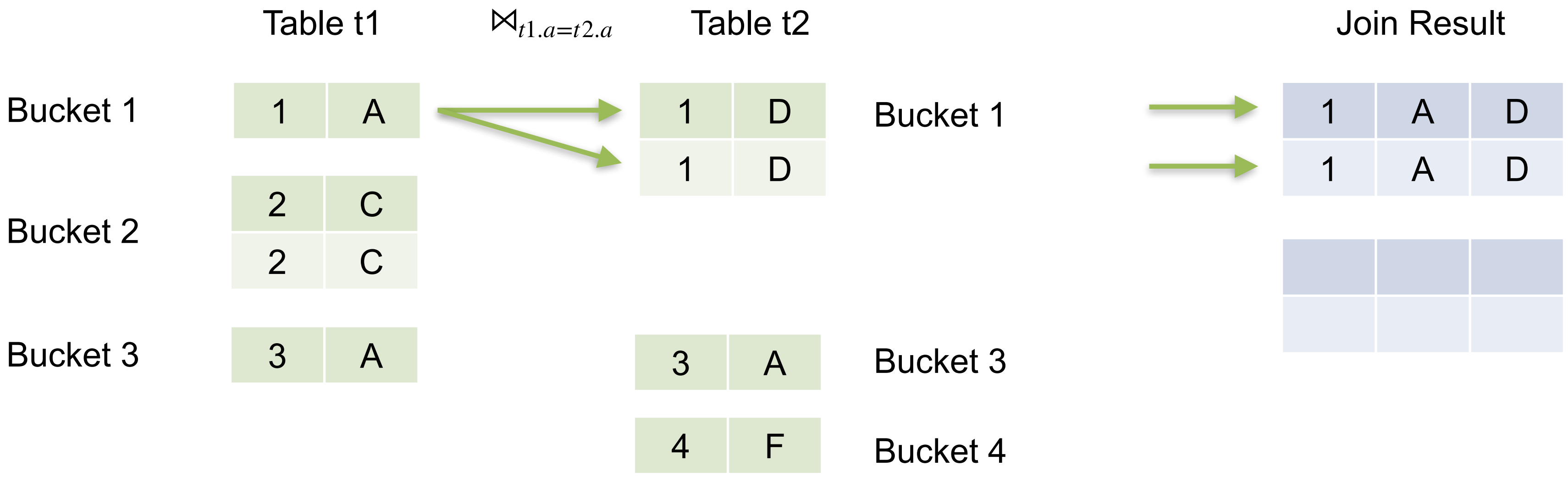


Table t1 $\bowtie_{t1.a=t2.a}$ Table t2 — Join Result

Table t1

Bucket 1	1	A
Bucket 2	2	C
	2	C
Bucket 3	3	A

Table t2

1	D	Bucket 1
1	D	
3	A	Bucket 3
4	F	Bucket 4

Join Result

1	A	D
1	A	D

Load: Buckets 1

1. Hash both tables separately on join attribute
2. Load same buckets for both tables
3. Issue joined pairs from buckets

Example: Hash Join

Query: SELECT * from t1, t2 where t1.a = t2.a

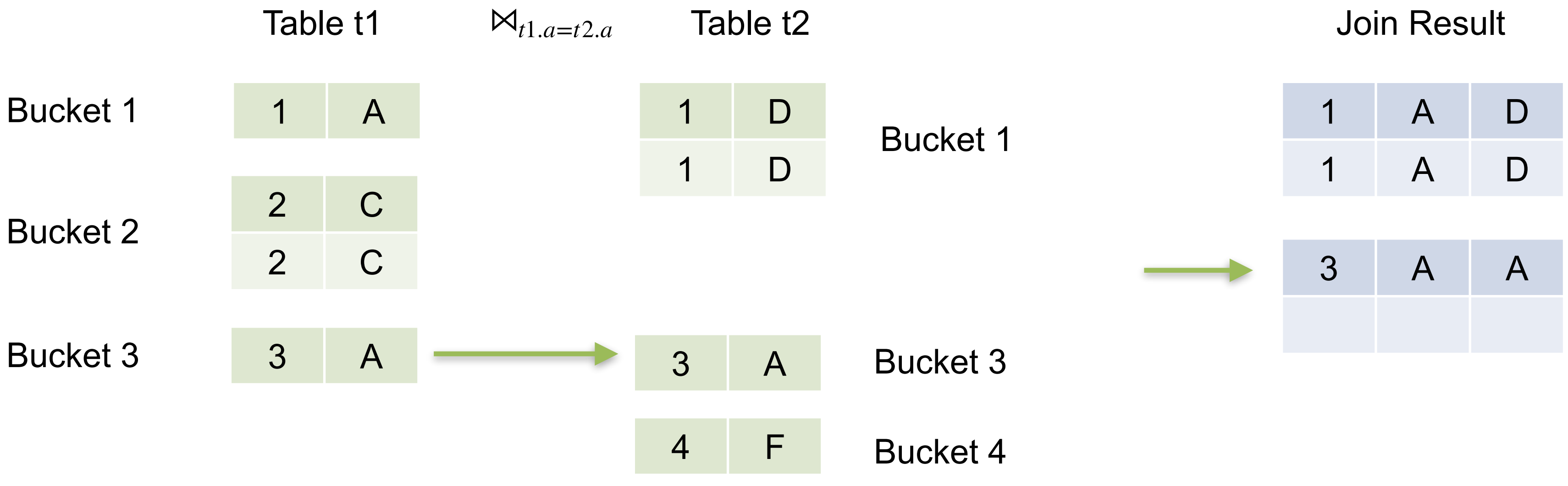


Table t1 $\bowtie_{t1.a=t2.a}$ Table t2

Table t1

Bucket 1	1	A
Bucket 2	2	C
	2	C
Bucket 3	3	A

Table t2

1	D	Bucket 1
1	D	
3	A	Bucket 3
4	F	Bucket 4

Join Result

1	A	D
1	A	D
3	A	A

Load: Buckets 3

1. Hash both tables separately on join attribute
2. Load same buckets for both tables
3. Issue joined pairs from buckets