Distributed databases

Acknowledgements to A. Kemper and Eickler
Distributed Databases

- Motivation:
  - Former times:
    - Bank with subsidiaries
  - Nowadays:
    - Virtual organization

  + Subsidiaries should work on data of local customers
  + Central site should be able to access all data
Other scenarios

Federation:
- Commercial database systems (DB2…)
- Networked Graphs in RDF/SPARQL

Distribution in the Cloud
- Large scale data processing
- RDF (current diploma thesis)
Terminology

Distributed Database

- Collection of information units, distributed on multiple computers connected with communication net

→ Ceri & Pelagatti (1984)

Cooperation between autonomously working stations for performing global tasks

Loosely integrated databases / Multi-database systems

- Autonomous working
- Autonomous structuring/schema
- No joint administration
Distributed Database system

Station S₁

Kommunikationsnetz

Station S₂

Station S₃
Design of distributed database system

- Global schema
- Fragmentation schema
- Allocation schema

Local schema

Local DBMS

Local DB

Station $S_1$

...
Fragmentation and Allocation of a Relation

Fragmentation: Fragments contain data with likewise access patterns

Allocation: Fragments are assigned to the stations

- with replication
- without replication
Fragmentation Allocation

Station $S_1$

Station $S_2$

Station $S_3$
Fragmentation

horizontal fragmentation:
Partitioning the relation in disjoint tuple sets

vertical fragmentation:
Summarization of attributes with likewise access patterns

Combined fragmentation: Application of horizontal and vertical fragmentation on one relation
Soundness requirements

- Reconstructable
- Complete
- Disjoint (to some extent)
### Example relation Professors

<table>
<thead>
<tr>
<th>PersNr</th>
<th>Name</th>
<th>Rang</th>
<th>Raum</th>
<th>Fakultät</th>
<th>Gehalt</th>
<th>Steuerklasse</th>
</tr>
</thead>
<tbody>
<tr>
<td>2125</td>
<td>Sokrates</td>
<td>C4</td>
<td>226</td>
<td>Philosophie</td>
<td>85000</td>
<td>1</td>
</tr>
<tr>
<td>2126</td>
<td>Russel</td>
<td>C4</td>
<td>232</td>
<td>Philosophie</td>
<td>80000</td>
<td>3</td>
</tr>
<tr>
<td>2127</td>
<td>Kopernikus</td>
<td>C3</td>
<td>310</td>
<td>Physik</td>
<td>65000</td>
<td>5</td>
</tr>
<tr>
<td>2133</td>
<td>Popper</td>
<td>C3</td>
<td>52</td>
<td>Philosophie</td>
<td>68000</td>
<td>1</td>
</tr>
<tr>
<td>2134</td>
<td>Augustinus</td>
<td>C3</td>
<td>309</td>
<td>Theologie</td>
<td>55000</td>
<td>5</td>
</tr>
<tr>
<td>2136</td>
<td>Curie</td>
<td>C4</td>
<td>36</td>
<td>Physik</td>
<td>95000</td>
<td>3</td>
</tr>
<tr>
<td>2137</td>
<td>Kant</td>
<td>C4</td>
<td>7</td>
<td>Philosophie</td>
<td>98000</td>
<td>1</td>
</tr>
</tbody>
</table>
Abstract:

For 2 predicates $p_1$ and $p_2$ there are 4 fragments:

$$R1 := \sigma_{p_1 \wedge p_2}(R)$$
$$R2 := \sigma_{p_1 \wedge \neg p_2}(R)$$
$$R3 := \sigma_{\neg p_1 \wedge p_2}(R)$$
$$R4 := \sigma_{\neg p_1 \wedge \neg p_2}(R)$$

$n$ partitioning predicates $p_1, \ldots, p_n$ yield $2^n$ fragments
Useful grouping of professor according to faculty

3 fragmentation predicates:

\[ \rho_1 \equiv \text{Fakultät} = \text{Theologie}' \]
\[ \rho_2 \equiv \text{Fakultät} = \text{Physik}' \]
\[ \rho_3 \equiv \text{Fakultät} = \text{Philosophie}' \]

\[ \text{TheolProfs'} := \sigma_{\rho_1 \land \neg \rho_2 \land \rho_3}(\text{Professoren}) = \sigma_{\rho_1}(\text{Professoren}) \]
\[ \text{PhysikProfs'} := \sigma_{\neg \rho_1 \land \rho_2 \land \rho_3}(\text{Professoren}) = \sigma_{\rho_2}(\text{Professoren}) \]
\[ \text{PhiloProfs'} := \sigma_{\neg \rho_1 \land \neg \rho_2 \land \rho_3}(\text{Professoren}) = \sigma_{\rho_3}(\text{Professoren}) \]
\[ \text{AndereProfs'} := \sigma_{\neg \rho_1 \land \neg \rho_2 \land \neg \rho_3}(\text{Professoren}) \]
## Extension of Professors and Lecture

### Professoren

<table>
<thead>
<tr>
<th>PersNr</th>
<th>Name</th>
<th>Rang</th>
<th>Raum</th>
</tr>
</thead>
<tbody>
<tr>
<td>2125</td>
<td>Sokrates</td>
<td>C4</td>
<td>226</td>
</tr>
<tr>
<td>2126</td>
<td>Russel</td>
<td>C4</td>
<td>232</td>
</tr>
<tr>
<td>2127</td>
<td>Kopernikus</td>
<td>C3</td>
<td>310</td>
</tr>
<tr>
<td>2133</td>
<td>Popper</td>
<td>C3</td>
<td>52</td>
</tr>
<tr>
<td>2134</td>
<td>Augustinus</td>
<td>C3</td>
<td>309</td>
</tr>
<tr>
<td>2136</td>
<td>Curie</td>
<td>C4</td>
<td>36</td>
</tr>
<tr>
<td>2137</td>
<td>Kant</td>
<td>C4</td>
<td>7</td>
</tr>
</tbody>
</table>

### Vorlesungen

<table>
<thead>
<tr>
<th>VorlNr</th>
<th>Titel</th>
<th>SWS</th>
<th>Gelesen Von</th>
</tr>
</thead>
<tbody>
<tr>
<td>5001</td>
<td>Grundzüge</td>
<td>4</td>
<td>2137</td>
</tr>
<tr>
<td>5041</td>
<td>Ethik</td>
<td>4</td>
<td>2125</td>
</tr>
<tr>
<td>5043</td>
<td>Erkenntnistheorie</td>
<td>3</td>
<td>2126</td>
</tr>
<tr>
<td>5049</td>
<td>Mäeutik</td>
<td>2</td>
<td>2125</td>
</tr>
<tr>
<td>4052</td>
<td>Logik</td>
<td>4</td>
<td>2125</td>
</tr>
<tr>
<td>5052</td>
<td>Wissenschaftstheorie</td>
<td>3</td>
<td>2126</td>
</tr>
<tr>
<td>5216</td>
<td>Bioethik</td>
<td>2</td>
<td>2126</td>
</tr>
<tr>
<td>5259</td>
<td>Der Wiener Kreis</td>
<td>2</td>
<td>2133</td>
</tr>
<tr>
<td>5022</td>
<td>Glaube und Wissen</td>
<td>2</td>
<td>2134</td>
</tr>
<tr>
<td>4630</td>
<td>Die 3 Kritiken</td>
<td>4</td>
<td>2137</td>
</tr>
</tbody>
</table>
Example Vorlesungen from the university schema:
Fragmentation into groups with likewise SWS-Zahl

\[ 2\text{SWSVorls} := \sigma_{\text{sWS}=2}(\text{Vorlesungen}) \]
\[ 3\text{SWSVorls} := \sigma_{\text{sWS}=3}(\text{Vorlesungen}) \]
\[ 4\text{SWSVorls} := \sigma_{\text{sWS}=4}(\text{Vorlesungen}) \]

Unsuitable fragmentation for querying
```
select Titel, Name
from Vorlesungen, Professoren
where gelesenVon = PersNr;
```

results in:

$$\Pi_{\text{Titel, Name}}((\text{TheolProfs'} \Join 2\text{SWSVorls}) \cup \text{(TheolProfs'} \Join 3\text{SWSVorls}) \cup ... \cup (\text{Philoprofs'} \Join 4\text{SWSVorls}) )$$

Join-Graph for this problem:
Solution: derived fragmentation

TheolProfs' \text{TheolVorls}

PhysikProfs' \text{PhysikVorls}

PhiloProfs' \text{PhiloVorls}

\text{TheolVorls} := \text{Vorlesungen } \Pi_{\text{Titel, Name}} ((\text{TheolProfs'} \Join_{p} \text{TheolVorls}) \cup (\text{PhysikProfs'} \Join_{p} \text{PhysikVorls}) \cup (\text{PhiloProfs'} \Join_{p} \text{PhiloVorls}))

\text{with } p \equiv (\text{PersNr} = \text{gelesenVon})
Abstract representation:
Arbitrary vertical fragmentation does not guarantee reconstructability.

2 possible approaches to ensure reconstructability:

1. Each fragment contains the primary key of the original relation – destroy disjointness.
2. Each tuple of the original relation is assigned a unique surrogate (= artificially generated object identifier), which is part of each vertical fragment of a tuple.
Vertical Fragmentation (Example)

The university administration is interested in:
PersNr, Name, Gehalt (salary) and Steuerklasse (taxation category)

ProfVerw := \( \Pi_{\text{PersNr, Name, Gehalt, Steuerklasse}} \) (Professoren)

Teaching and research relates only to
PersNr, Name, Rang (status), Raum (room) and Fakultät (faculty):

Profs := \( \Pi_{\text{PersNr, Name, Rang, Raum, Fakultät}} \) (Professoren)

Reconsting the original relation Professoren:

Professoren = ProfVerw \( \bowtie \) \( \text{ProfVerw.PersNr} = \text{Profs.PersNr} \) Profs
Combined fragmentation

a) horizontal fragmentation following vertical fragmentation

\[
\begin{array}{c}
R \\
R_{21} \\
R_{22} \\
R_{23}
\end{array}
\begin{array}{c}
R_1 \\
R_2
\end{array}
\]

b) vertical fragmentation following horizontal fragmentation

\[
\begin{array}{c}
R \\
R_1 \\
R_2 \\
R_3
\end{array}
\begin{array}{c}
R_{31} \\
R_{32}
\end{array}
\]
Reconstruction after combined fragmentation

Case a)

\[ R = R_1 \cup (R_{21} \cup R_{22} \cup R_{23}) \]

Case b)

\[ R = R_1 \cup R_2 \cup (R_{31} \cap \kappa = R_{32} \cap R_{32}) \]
Tree representations of fragmentations (example)
Allocation

Individual fragments can be allocated to multiple nodes
Allocation for our example without replication
⇒ **redundancy-free** allocation

<table>
<thead>
<tr>
<th>Node</th>
<th>Remark</th>
<th>Allocated fragments</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_{Verw}</td>
<td>Administration</td>
<td>{ProfVerw}</td>
</tr>
<tr>
<td>S_{Physik}</td>
<td>Dean’s office Physik</td>
<td>{PhysikVorls, PhysikProfs}</td>
</tr>
<tr>
<td>S_{Philo}</td>
<td>Dean’s office Philosophy</td>
<td>{PhiloVorls, PhiloProfs}</td>
</tr>
<tr>
<td>S_{Theol}</td>
<td>Dean’s office Theology</td>
<td>{TheolVorls, TheolProfs}</td>
</tr>
</tbody>
</table>
Transparency in distributed databases

Degree of transparency that a distributed database management system gives to the user for accessing distributed data

Different degrees of transparency:
- Fragmentation transparency
- Allocation transparency
- Local schema transparency
Fragmentation transparency

Example query requiring fragmentation transparency:

```sql
select Titel, Name
from Vorlesungen, Professoren
where gelesenVon = PersNr;
```

Example for change operation requiring fragmentation transparency:

```sql
update Professoren
set Fakultät = 'Theologie'
where Name = 'Sokrates';
```
Example (continued)

Changing the attribute value of *Fakultät*

Transferring the Sokrates-Tuple from fragment *PhiloProf* into fragment *TheolProf* (= deleting from *PhiloProf*, inserting into *TheolProf*)

Updating derived fragmentations of *Vorlesungen* (= Inserting lectures given by Sokrates into *TheolVorls*, deleting his lectures from *PhiloVorls*)
Allocation transparency

User must know fragmentation, but not their "location"

Example query:

```sql
select Gehalt
from ProfVerw
where Name = 'Sokrates';
```
Allocation transparency (continued)

Original relation must remain reconstructable

Example:
Administration wants to know how much C4 professors earn in theology

Due to lack of fragmentation transparency the query must be reformulated:

```
select sum (Gehalt)
from ProfVerw, TheolProfs
where ProfVerw.PersNr = TheolProfs.PersNr and
   Rang = 'C4';
```
Local schema transparency

The user must know the computing node, which is the location of the fragment.

Example query:

```sql
select Name
from TheolProfs at S_{Theol}
where Rang = 'C3';
```
Local schema transparency requires that all nodes use the same data model and query language.

⇒ previous query may also be executed on the analogous computing node $S_{\text{Philo}}$

This is not possible if different DBMS are linked together

Use of different data models at local DBMS are called „Multi-Database-Systems“
Premise: Fragmentation transparency

Task of the query translator: generation of a query execution plan on fragments

Task of the query planner: generation of an efficient query execution plan → depending on the allocation of fragments to different computing nodes
Translation of SQL query to global schema into an equivalent query on fragments requires two steps:

1. Reconstruction of all global relations needed in the query from their fragments. Result is an source algebraic expression

2. Applying the query algebraic expression on the source algebraic expression.
Example

```sql
select Titel
from Vorlesungen, Profs
where gelesenVon = PersNr and
    Rang = 'C4';
```

The resulting algebraic expression is called **canonical query from**:

The resulting algebraic expression is called **canonical query from**:

```
Π_Titel
σ_Rang='C4'
D
gelesenVon=PersNr
```

```
TheolVorls PhiloVorls PhysikVorls TheolProfs PhiloProfs PhysikProfs
```
Algebraic equivalences

For efficient query execution the optimizer uses the following property:

\[(R_1 \cup R_2) \circ (S_1 \cup S_2) =
(R_1 \circ S_1) \cup (R_1 \circ S_2) \cup (R_2 \circ S_1) \cup (R_2 \circ S_2)\]

The generalization to \(n\) horizontal fragments \(R_1, \ldots, R_n\) of \(R\) and \(m\) fragments \(S_1, \ldots, S_m\) of \(S\) results in:

\[(R_1 \cup \ldots \cup R_n) \circ (S_1 \cup \ldots \cup S_m) = \bigcup_{1 \leq i \leq n} \bigcup_{1 \leq j \leq m} (R_i \circ S_j)\]

If: \(S_i = S \bowtie_p R_i\) mit \(S = S_i \cup \ldots \cup S_n\)
Then:
\[R_i \circ S_j = \emptyset \text{ für } i \neq j\]
Algebraic equivalences (continued)

For a derived horizontal fragmentation of $S$:

$$(R_1 \cup ... \cup R_n) \bowtie_p (S_1 \cup ... \cup S_m) = (R_1 \bowtie_p S_1) \cup (R_2 \bowtie_p S_2) \cup ... \cup (R_n \bowtie_p S_n)$$

For our example:

$$(\text{TheolVorls} \cup \text{PhysikVorls} \cup \text{PhiloVorls}) \bowtie ... (\text{TheolProfs} \cup \text{PhysikProfs} \cup \text{PhiloProfs})$$

To push down selections and projections in the query execution tree the following rules apply:

$$\sigma_p(R_1 \cup R_2) = \sigma_p(R_1) \cup \sigma_p(R_2)$$
$$\Pi_L(R_1 \cup R_2) = \Pi_L(R_1) \cup \Pi_L(R_2)$$
Applying these algebraic rules generates the following query plan:

- \( \Pi_{\text{Titel}} \)
- \( \sigma_{\text{Rang}='C4'} \)
- \( \text{gelesenVon}=\text{PersNr} \)
- \( \bigcup \)

Query executions can be performed locally on nodes \( S_{\text{Theol}}, S_{\text{Physik}} \), and \( S_{\text{Philo}} \). Nodes may work in parallel and transmit local results independently of each other to the node that computes the union.
Example:

```sql
select Name, Gehalt
from Professoren
where Gehalt > 80000;
```

Canonical query plan:
```
\[ \Pi \text{Name, Gehalt} \sigma \text{Gehalt > 80000} \]
```

```
\text{TheolProfs} \cup \text{PhysikProfs} \cup \text{PhiloProfs}
```

```
\text{ProfVerw}
```

```
\text{D}
```

Optimization for vertical fragmentation

For our example:

All required data are contained in ProfVerw ⇒ discard join and union.

Resulting query execution plan:

\[
\begin{align*}
\Pi_{\text{Name, Gehalt}} \\
\sigma_{\text{Gehalt}>80000} \\
\text{ProfVerw}
\end{align*}
\]

Example for query that is hard to optimize:
(Attribute Rang is lacking in ProfVerw)

\[
\begin{align*}
\text{select } \text{Name, Gehalt, Rang} \\
\text{from } \text{Professoren} \\
\text{where } \text{Gehalt > 80000;}
\end{align*}
\]
The natural join of two relations R and S

<table>
<thead>
<tr>
<th>R</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
</tr>
<tr>
<td>a₂</td>
<td>b₂</td>
<td>c₂</td>
</tr>
<tr>
<td>a₃</td>
<td>b₃</td>
<td>c₁</td>
</tr>
<tr>
<td>a₄</td>
<td>b₄</td>
<td>c₂</td>
</tr>
<tr>
<td>a₅</td>
<td>b₅</td>
<td>c₃</td>
</tr>
<tr>
<td>a₆</td>
<td>b₆</td>
<td>c₂</td>
</tr>
<tr>
<td>a₇</td>
<td>b₇</td>
<td>c₆</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>c₁</td>
<td>d₁</td>
<td>e₁</td>
</tr>
<tr>
<td>c₃</td>
<td>d₂</td>
<td>e₂</td>
</tr>
<tr>
<td>c₄</td>
<td>d₃</td>
<td>e₃</td>
</tr>
<tr>
<td>c₅</td>
<td>d₄</td>
<td>e₄</td>
</tr>
<tr>
<td>c₇</td>
<td>d₅</td>
<td>e₅</td>
</tr>
<tr>
<td>c₈</td>
<td>d₆</td>
<td>e₆</td>
</tr>
<tr>
<td>c₅</td>
<td>d₇</td>
<td>e₇</td>
</tr>
</tbody>
</table>

\[ R \Join S = \]

<table>
<thead>
<tr>
<th>R</th>
<th>D</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
</tr>
<tr>
<td>a₃</td>
<td>b₃</td>
<td>c₁</td>
</tr>
<tr>
<td>a₅</td>
<td>b₅</td>
<td>c₃</td>
</tr>
</tbody>
</table>
Join-Execution in Distributed DBMS

Even more critical than in centralized data bases

Problem: Join arguments may be distributed to two different nodes of the distributed DBMS

2 possibilities: Join-evaluation with and without filter
Most general case:

Outer argument relation $R$ is allocated to computing node $St_R$

Inner argument relation $S$ is allocated to computing node $St_S$

Result of join evaluation is required on third node $St_{Result}$
Join evaluation without filter

Nested loops

Transfer of one argument relation

Transfer of both argument relations
Nested loops

Iteration over outer relation \( R \) using iteration variable \( r \) and requesting compatible tuples \( s \in S \) with \( r.C = s.C \) (using communication net at \( St_{S} \))

This approach requires for each tuple from \( R \) one request and a compatible tuple set from \( S \) (which may be empty)

\[ 2 \ast |R| \] messages must be send
Transfer of argument relation

1. Complete transfer of argument relation (e.g. R) to node of other argument relation

2. Exploitation of Index on $S.C$ if available
1. Transfer of both argument relations to node $St_{Result}$

2. Computation of result on node $St_{Result}$ using
   a) Merge join (if sorted)
   or
   b) Hash join (if unsorted)

→ Loss of existing index for join evaluation
→ No loss of sorting of argument relation(s)
Join evaluation with filter

Filtering using semi joins

Key idea:
transfer only tuples with compatible join partners

Exploiting algebraic equivalences:
\[ R \bowtie S = R \bowtie (R \bowtie S) \]
\[ R \bowtie S = \Pi_c(R) \bowtie S \]
Join evaluation with filter
(example, filtering relation $S$)

1. Transfer of different $C$-values from $R (= \Pi_C(R))$ to $St_S$

2. Evaluation of semi join $R \bowtie S = \Pi_C(R) \bowtie S$ on $St_S$ and transfer to $St_R$

3. Evaluation of join on $St_R$, which only needs the transferred result tuples of semi joins

Transfer costs are reduced iff:

$$\| \Pi_C(R) \| + \| R \bowtie S \| < \| S \|$$

with $\| P \| =$ Size (in Byte) of a relation
Evaluation of the join $R_D S$ with semi join filtering on $S$.
Alternative evaluation plans

1. Alternative:

\[ R \leftarrow I \rightarrow \overline{I} \rightarrow St_{R} \rightarrow S \rightarrow St_{S} \rightarrow \ldots \rightarrow St_{Result} \]

2. Alternative:

\[ (R \mathbin{\Join} \Pi_{c}(S)) \mathbin{\Delta} (\Pi_{c}(R) \mathbin{\Join} S) \]
Synchronization of replicated data

Problem:

For data item A there exist several copies A1, A2, ..., An, which may reside on different nodes.

Read actions require only one copy, updates must change all existing copies.

⇒ Problems with high efficiency and availability
Quorum-Consensus approach

Trade-off between efficiency of read and update transactions
+ shifting some overhead from update to read transaction.
  Approach: assign copies $A_i$ of a replicated data item $A$ individual weights

Read quorum $Q_r(A)$
Write quorum $Q_w(A)$

The following conditions must be met:

1. $Q_w(A) + Q_w(A) > W(A)$
2. $Q_r(A) + Q_w(A) > W(A)$
Example

<table>
<thead>
<tr>
<th>Station ($S_i$)</th>
<th>Kopie ($A_i$)</th>
<th>Gewicht ($w_i$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$A_1$</td>
<td>3</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$A_2$</td>
<td>1</td>
</tr>
<tr>
<td>$S_3$</td>
<td>$A_3$</td>
<td>2</td>
</tr>
<tr>
<td>$S_4$</td>
<td>$A_4$</td>
<td>2</td>
</tr>
</tbody>
</table>

\[ W(A) = \sum_{i=1}^{4} w_i(A) = 8 \]

\[ Q_r(A) = 4 \]

\[ Q_w(A) = 5 \]
### States

#### a) Before writing

<table>
<thead>
<tr>
<th>Station</th>
<th>Kopie</th>
<th>Gewicht</th>
<th>Wert</th>
<th>Versions#</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>A₁</td>
<td>3</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>S₂</td>
<td>A₂</td>
<td>1</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>S₃</td>
<td>A₃</td>
<td>2</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>S₄</td>
<td>A₄</td>
<td>2</td>
<td>1000</td>
<td>1</td>
</tr>
</tbody>
</table>

#### b) After writing using write quorum 5

<table>
<thead>
<tr>
<th>Station</th>
<th>Kopie</th>
<th>Gewicht</th>
<th>Wert</th>
<th>Versions#</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>A₁</td>
<td>3</td>
<td>1100</td>
<td>2</td>
</tr>
<tr>
<td>S₂</td>
<td>A₂</td>
<td>1</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>S₃</td>
<td>A₃</td>
<td>2</td>
<td>1100</td>
<td>2</td>
</tr>
<tr>
<td>S₄</td>
<td>A₄</td>
<td>2</td>
<td>1000</td>
<td>1</td>
</tr>
</tbody>
</table>