La Fragmentation Horizontale Revisitée: Prise en Compte de l’Interaction de Requêtes

Amira KERKAD
amira.kerkad@ensma.fr

Ladjel BEL LATRECHE
bellatreche@ensma.fr

Dominique GENIET
dominique.geniet@ensma.fr

EDA
Blois – June 13rd 2013
Data warehousing: Star Schema

- One fact table (volume)
- Many dimension tables

Star Join Queries

Selections on dimension tables

⇒ Multiple joins (between fact and dimension tables)
⇒ No direct joins between dimensions

Requirements

⇒ Lowering response time
Data warehousing: Star Schema

- One fact table (volume)
- Many dimension tables

Star Join Queries

Selections on dimension tables

⇒ Multiple joins (between fact and dimension tables)
⇒ No direct joins between dimensions

Requirements

⇒ Lowering response time

⇒ Optimization is crucial
Data warehousing: Star Schema

- One fact table (volume)
- Many dimension tables

Star Join Queries

Selections on dimension tables

- Multiple joins (between fact and dimension tables)
- No direct joins between dimensions

Requirements

- Lowering response time

⇒ Optimization is crucial

Horizontal Partitioning is well adapted for Star Join Queries
Objective

Decompose table instances into disjoint groups of instances

Two types:

Primary [Ceri’82]  Derived [Ceri’82]
**Objective**

Decompose table instances into disjoint groups of instances

**Two types:**

- **Primary [Ceri’82]**
  - \( \text{Customer}(\text{Id, Name, Gender}) \)

- **Derived [Ceri’82]**
  - \( \text{Sales}(\text{Cust_Id, Prod_Id, Quantity}) \)

- \( \text{Gender} = \text{‘Female’} \):
  - Cust_Female
  - Sales_Female
- \( \text{Gender} = \text{‘Male’} \):
  - Cust_Male
  - Sales_Male

\( \text{Sales} \bowtie \text{Cust_Female} \)

\( \text{Sales} \bowtie \text{Cust_Male} \)
Objective

Decompose table instances into disjoint groups of instances

Two types:

- Primary [Ceri’82]
  \[ \text{Customer}(\text{Id}, \text{Name}, \text{Gender}) \]
  - Gender = ‘Female’
    - Cust_Female
  - Gender = ‘Male’
    - Cust_Male

- Derived [Ceri’82]
  \[ \text{Sales}(\text{Cust_Id}, \text{Prod_Id}, \text{Quantity}) \]
  - Sales_Female
  - Sales_Male

\[ \Rightarrow \text{Optimizing selections and joins} \]
Objective

Decompose table instances into disjoint groups of instances

Two types:

Primary [Ceri’82]
Customer(\text{Id}, \text{Name}, \text{Gender})

Gender = ‘Female’

Cust_Female

Sales \Join\ Cust_Female

Gender = ‘Male’

Cust_Male

Sales \Join\ Cust_Male

Derived [Ceri’82]
Sales(\text{Cust_Id}, \text{Prod_Id}, \text{Quantity})

Optimizing selections and joins
Objective

Decompose table instances into disjoint groups of instances

Two types:

- **Primary [Ceri’82]**
  - *Customer*(Id, Name, Gender)
  - Gender = ‘Female’ → Cust_Female
  - Gender = ‘Male’ → Cust_Male

- **Derived [Ceri’82]**
  - *Sales*(Cust_Id, Prod_Id, Quantity)
  - Sales ⋉ Cust_Female → Sales_Female
  - Sales ⋉ Cust_Male → Sales_Male

Input:

⇒ Optimizing selections and joins
Objective
Decompose table instances into disjoint groups of instances

Two types:

Primary [Ceri’82]
Customer(Id, Name, Gender)

Gender = ‘Female’
Cust_Female

Gender = ‘Male’
Cust_Male

Derived [Ceri’82]
Sales(Cust_Id, Prod_Id, Quantity)

Sales ⋉ Cust_Female
Sales_Female

Sales ⋉ Cust_Male
Sales_Male

Input:
Data warehouse \{Fact, D_1, D_2, ..., D_d\}

⇒ Optimizing selections and joins
Objective
Decompose table instances into disjoint groups of instances

Two types:

Primary [Ceri’82]

Customer(Id, Name, Gender)

Gender = ‘Female’

Cust_Female

Gender = ‘Male’

Cust_Male

Derived [Ceri’82]

Sales(Cust_Id, Prod_Id, Quantity)

Sales Female

Sales Male

Input:

Data warehouse {Fact, D_1, D_2, ..., D_d}

Workload Q

Optimizing selections and joins
Objective

Decompose table instances into disjoint groups of instances

Two types:

- **Primary** [Ceri’82] 
  \[\text{Customer}(\text{Id}, \text{Name}, \text{Gender})\]

- **Derived** [Ceri’82] 
  \[\text{Sales}(\text{Cust_Id}, \text{Prod_Id}, \text{Quantity})\]

  \[\text{Sales} \Join \text{Cust_Female}\]

  \[\text{Sales} \Join \text{Cust_Male}\]

\[\Rightarrow\text{Optimizing selections and joins}\]

Input:

- Data warehouse \{Fact, D_1, D_2, ..., D_d\}
- Workload Q
- \(W: \text{threshold (fixed by DBA)}\)
Horizontal Data Partitioning

Objective

Decompose table instances into disjoint groups of instances

Two types:

- **Primary [Ceri’82]**
  - `Customer(Id, Name, Gender)`
  - `Gender = ‘Female’` → `Cust_Female`
  - `Gender = ‘Male’` → `Cust_Male`

- **Derived [Ceri’82]**
  - `Sales(Cust_Id, Prod_Id, Quantity)`
  - `Sales ⋉ Cust_Female`
  - `Sales ⋉ Cust_Male`

⇒ Optimizing selections and joins

**Input:**

- Data warehouse \{Fact, D_1, D_2, ..., D_d\}
- Workload Q

**Output:**

W : threshold (fixed by DBA)
Horizontal Data Partitioning

Objective
Decompose table instances into disjoint groups of instances

Two types:

- **Primary** [Ceri’82]:
  - `Customer(Id, Name, Gender)`
  - `Gender = 'Female' → Cust_Female`
  - `Gender = 'Male' → Cust_Male`

- **Derived** [Ceri’82]:
  - `Sales(Cust_Id, Prod_Id, Quantity)`
  - `Sales ⋉ Cust_Female → Sales_Female`
  - `Sales ⋉ Cust_Male → Sales_Male`

⇒ Optimizing selections and joins

Input:
- Data warehouse \{Fact, D_1, D_2, ..., D_d\}
- Workload Q
- W : threshold (fixed by DBA)

Output:
- Set D' ⊆ D of partitioned dimensions
Horizontal Data Partitioning

Objective

Decompose table instances into disjoint groups of instances

Two types:

- **Primary** [Ceri’82]
  - \( \text{Customer}(\text{Id}, \text{Name}, \text{Gender}) \)
  - \( \text{Gender} = \text{‘Female’} \) → \( \text{Cust_Female} \)
  - \( \text{Gender} = \text{‘Male’} \) → \( \text{Cust_Male} \)

- **Derived** [Ceri’82]
  - \( \text{Sales}(\text{Cust_Id}, \text{Prod_Id}, \text{Quantity}) \)
  - \( \text{Sales} \bowtie \text{Cust_Female} \)
  - \( \text{Sales} \bowtie \text{Cust_Male} \)

⇒ Optimizing selections and joins

Input:
- Data warehouse \{\text{Fact, D}_1, \text{D}_2, ..., \text{D}_d\}
- Workload Q \( W : \text{threshold (fixed by DBA)} \)

Output:
- Set \( \text{D}' \subseteq \text{D} \) of partitioned dimensions
- Set of N fragments of facts \( \text{F}_1, ..., \text{F}_N \)
Objective

Decompose table instances into disjoint groups of instances

Two types:

- **Primary** [Ceri’82]
  - \( \text{Customer}(\text{Id}, \text{Name}, \text{Gender}) \)
  - \( \text{Gender} = \text{‘Female’} \) → \( \text{Cust}_\text{Female} \)
  - \( \text{Gender} = \text{‘Male’} \) → \( \text{Cust}_\text{Male} \)

- **Derived** [Ceri’82]
  - \( \text{Sales}(\text{Cust}_\text{Id}, \text{Prod}_\text{Id}, \text{Quantity}) \)
  - \( \text{Sales} \bowtie \text{Cust}_\text{Female} \) → \( \text{Sales}_\text{Female} \)
  - \( \text{Sales} \bowtie \text{Cust}_\text{Male} \) → \( \text{Sales}_\text{Male} \)

⇒ Optimizing selections and joins

**Input:**
- Data warehouse \( \{\text{Fact}, \text{D}_1, \text{D}_2, \ldots, \text{D}_d\} \)
- Workload \( \text{Q} \)
- \( W \) : threshold (fixed by DBA)

**Output:**
- Set \( D' \subseteq D \) of partitioned dimensions
- Set of \( N \) fragments of facts \( F_1, \ldots, F_N \)

**Objectives:**
Horizontal Data Partitioning

- **Objective**
  Decompose table instances into disjoint groups of instances

- **Two types:**
  - **Primary [Ceri’82]**
    - *Customer*(Id, Name, Gender)
  - **Derived [Ceri’82]**
    - *Sales*(Cust_Id, Prod_Id, Quantity)

  **Gender= ‘Female’**
  - Cust_Female

  **Gender= ‘Male’**
  - Cust_Male

  - **Sales** ⋉ Cust_Female
  - Sales_Female

  - **Sales** ⋉ Cust_Male
  - Sales_Male

  ⇒ Optimizing selections and joins

- **Input:**
  Data warehouse \{Fact, D_1, D_2, ..., D_d\}
  Workload Q
  \(W : \text{threshold (fixed by DBA)}\)

- **Output:**
  Set \(D' \subseteq D\) of partitioned dimensions
  Set of \(N\) fragments of facts \(F_1, ..., F_N\)

- **Objectives:**
  Lowering execution cost of Q
Horizontal Data Partitioning

Objective
Decompose table instances into disjoint groups of instances

Two types:
- Primary [Ceri’82]
  \( \text{Customer}(\text{Id}, \text{Name}, \text{Gender}) \)
  \( \text{Gender} = '\text{Female}' \)
  \( \text{Cust}_\text{Female} \)
  \( \text{Gender} = '\text{Male}' \)
  \( \text{Cust}_\text{Male} \)
- Derived [Ceri’82]
  \( \text{Sales}(\text{Cust}_\text{Id}, \text{Prod}_\text{Id}, \text{Quantity}) \)

\( \text{Sales} \bowtie \text{Cust}_\text{Female} \) \( \text{Sales}_\text{Female} \)
\( \text{Sales} \bowtie \text{Cust}_\text{Male} \) \( \text{Sales}_\text{Male} \)

Optimizing selections and joins

Input:
- Data warehouse \( \{ \text{Fact}, D_1, D_2, ..., D_d \} \)
- Workload \( Q \)
- \( W : \text{threshold (fixed by DBA)} \)

Output:
- Set \( D' \subseteq D \) of partitioned dimensions
- Set of \( N \) fragments of facts \( F_1, ..., F_N \)

Objectives:
- Lowering execution cost of \( Q \)
- \( N \leq W \)
Classification of Optimization Techniques

First Classification
[DEXA’07]

Structures

- Materialized Views (MV)
- Index (IX)
- Horizontal Partitioning (HP)
- Vertical Partitioning (VP)
- Parallel Computing (PC)

Non redundant
- Horizontal Partitioning (HP)
- Parallel Computing (PC)

Redundant
- Materialized Views (MV)
- Index (IX)
- Vertical Partitioning (VP)
Classification of Optimization Techniques

First Classification
[DEXA’07]

Second Classification
[DAWAK’08]
Classification of Optimization Techniques

Multi View Processing Plan

Query Interaction [Sellis’88]
Classification of Optimization Techniques

Multi View Processing Plan

Query Interaction [Sellis’88]
Classification of Optimization Techniques

Multi View Processing Plan

Query Interaction [Sellis’88]
Horizontal Data Partitioning Evolution

Classification of HDP approaches

- **Unconstrained**
  - Affinity [Karlapalem, 1996]
  - Minterms [Ozsuz and Valduriez, 1999]

- **Constrained**
  - Cost Model [Bellatreche et al., 2000]
  - Data Mining [Mahboubi and Darmont, 2009]
Horizontal Data Partitioning Evolution

Classification of HDP approaches

Horizontal Data Partitioning

- Unconstrained
  - Affinity [Karlapalem, 1996]
  - Minterms [Ozsu and Valduriez, 1999]

- Constrained
  - Cost Model [Bellatreche et al. 2000]
  - Data Mining [Mahboubi and Darmont, 2009]

Query Interaction

Diagram showing query interaction.
Horizontal Data Partitioning Evolution

Classification of HDP approaches

- Unconstrained
  - Affinity [Karlapalem, 1996]
  - Minterms [Ozsu and Valduriez, 1999]
- Constrained
  - Cost Model [Bellatreche et al. 2000]
  - Data Mining [Mahboubi and Darmont, 2009]

Query Interaction

Diagram showing relationships and data structures.
Outlines

- Motivating example
- Algebra
- EQHDP
- Experiments
- Conclusion & perspectives
Motivating example

First join very **expensive** and needs optimization

- Selection stage (HDP)
- **Spread** benefit through the workload...
Motivating example

First join very **expensive** and needs optimization

- Selection stage (HDP)
- **Spread** benefit through the workload...
Motivating example

First join very **expensive** and needs optimization

⇒ **Selection stage** (HDP)

⇒ **Spread** benefit through the workload...

1) **Group** queries
2) **Elect** one query in each group
3) **Steer** HDP process
Motivating example

First join very expensive and needs optimization
⇒ Selection stage (HDP)
⇒ Spread benefit through the workload...

1) Group queries
2) Elect one query in each group
3) Steer HDP process
Motivating example

First join very expensive and needs optimization
- Selection stage (HDP)
- Spread benefit through the workload...

- How to elect query (criterion)?
- Algebra to handle generate HDP schema?
- Prune predicates and steer HDP by query interaction?

1) Group queries
2) Elect one query in each group
3) Steer HDP process
Algebra allows to generate an encoding and a HDP schema.

### Generating incremental encoding

#### Horizontal Split

<table>
<thead>
<tr>
<th>Att1</th>
<th>val1</th>
<th>val2</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td>Att1</td>
<td>val1</td>
<td>val2</td>
<td>Else</td>
</tr>
</tbody>
</table>

#### Vertical Split

<table>
<thead>
<tr>
<th>Att1</th>
<th>val1</th>
<th>val2</th>
<th>Else</th>
</tr>
</thead>
<tbody>
<tr>
<td>Att2</td>
<td>val1</td>
<td>else</td>
<td></td>
</tr>
<tr>
<td></td>
<td>val1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Generating HDP schema

<table>
<thead>
<tr>
<th>Quantity</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Season</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Color</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Gender</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantity</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Season</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Type</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Color</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Gender</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Algebra allows to generate an encoding and a HDP schema

**Generating incremental encoding**

**Horizontal Split**

Att1: val1 | val2 | Else → Att1: val1 | val2 | Else

**Att1**

val1 | val2 | Else

**Vertical Split**

Att1: val1 | val2 | Else → Att1: val1 | val2 | Else

**Att2**

val1 | else

**Generating HDP schema**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Season</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Color</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Gender</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantity</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Season</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Type</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Color</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Gender</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Algebra allows to generate an encoding and a HDP schema.

**Generating incremental encoding**

**Horizontal Split**

<table>
<thead>
<tr>
<th>Attribute 1</th>
<th>val1</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Vertical Split**

<table>
<thead>
<tr>
<th>Attribute 1</th>
<th>val1</th>
<th>val2</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute 2</th>
<th>val1</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Generating HDP schema**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Season</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Color</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Gender</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Merge**

**Split**

**Multiple View Processing Plan**
Algebra allows to generate an encoding and a HDP schema

**Generating incremental encoding**

- **Horizontal Split**
  - Att1: val1 else
  - Att1: val1 val2 Else
  - Att2: val1 else

- **Vertical Split**
  - Att1: val1 val2 Else
  - Att1: val1 val2 Else

**Generating HDP schema**

- Quantity: 1 1 1
- Season: 1 2 1
- Type: 1 2 3
- Color: 1 2
- Gender: 1 1

---

**WORKLOAD PROCESSING**

<table>
<thead>
<tr>
<th>t0</th>
<th>t1</th>
<th>t2</th>
<th>...</th>
<th>ti..</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>φ</td>
<td>→Q1</td>
<td>→Q2</td>
<td>→Qi</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **S1**: Quantity <100 else
  - Season Autumn

- **S2**: Quantity <100 else
  - Season Autumn

**Multiple View Processing Plan**

- Encoding Predicates

- **Merge**

- **Split**
Algebra allows to generate an encoding and a HDP schema

### Generating incremental encoding

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Else</th>
</tr>
</thead>
<tbody>
<tr>
<td>Att1</td>
<td>val1</td>
<td>val2</td>
<td>Else</td>
</tr>
<tr>
<td>Att1</td>
<td>val1</td>
<td>val2</td>
<td>Else</td>
</tr>
</tbody>
</table>

**Horizontal Split**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Else</th>
</tr>
</thead>
<tbody>
<tr>
<td>Att1</td>
<td>val1</td>
<td>val2</td>
<td>Else</td>
</tr>
<tr>
<td>Att2</td>
<td>val1</td>
<td>else</td>
<td></td>
</tr>
</tbody>
</table>

**Vertical Split**

### Generating HDP schema

<table>
<thead>
<tr>
<th>Quantity</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Season</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Color</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Gender</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Merge**

### WORKLOAD PROCESSING

**Encoding Predicates**

**Multiple View Processing Plan**
Algebra allows to generate an encoding and a HDP schema.
Electing Queries for HDP

**Algorithm EQHDP**

1. generate_encoding()
2. EQA()
3. e := 1;
4. split_all();
5. E := elected(e)
6. while E not empty do
   7. prune_encoding(E);
   8. sort(E);
   9. S := required_attributes(E);
10. usage(S);
11. sort_attributes(S)
12. for all a ∈ S do
   13. for all sd ∈ SubDomains(a) do
      14. if ((U(sd) = 0) and (N < W)) then
      15. merge(sd, P0);
      16. end if
      17. if ((U(sd) = k) and (N < W)) then
      18. merge(sd, Pk);
      19. end if
   20. end for
12. end for
22. while k > 0 do
23. for all a ∈ S do
   24. for all sd ∈ SubDomains(a) do
      25. if (N < W) then
      26. merge(sd, P0);
      27. else
      28. merge(sd, Pk);
      29. end if
   30. end for
31. k := k - 1;
32. end while
33. split_disjoint();
34. e := e + 1;
35. E := elected(e);
36. end while
Electing Queries for HDP

Algorithm EQHDP
1: generate_encoding();
2: EQA();
3: e := 1;
4: split_all();
5: E := elected(e)
6: while E not empty do
7: prune_encoding(E);
8: sort(E);
9: S := required_attributes(E);
10: usage(S);
11: sort_attributes(S)
12: for all a ∈ S do
13: for all sd ∈ SubDomains(a) do
14: if ((U(sd) = 0) and (N < W)) then
15: merge(sd, P_0);
16: end if
17: if ((U(sd) = k) and (N < W)) then
18: merge(sd, P_k);
19: end if
20: end for
21: end for
22: while k > 0 do
23: for all a ∈ S do
24: for all sd ∈ SubDomains(a) do
25: if (N < W) then
26: merge(sd, P_0);
27: else
28: merge(sd, P_k);
29: end if
30: end for
31: k := k - 1;
32: end while
33: split_disjoint();
34: e := e + 1;
35: E := elected(e);
36: end while

Grouping Queries & Generating Encoding

MVPP

Generate encoding

Group Queries

Electing Queries

Merge (by usage)

Split (by usage)

If (N<W)

Elect successors

Cost Model

HDP schema
Electing Queries for HDP

Motivation
Algebra
Electing Queries for HDP
Experimental Study
Conclusion & Future work

Grouping Queries & Generating Encoding
Elect queries & Prune encoding

Algorithm

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>EQHDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>generate_encoding();</td>
</tr>
<tr>
<td>2.</td>
<td>EQA();</td>
</tr>
<tr>
<td>3.</td>
<td>e := 1;</td>
</tr>
<tr>
<td>4.</td>
<td>split_all();</td>
</tr>
<tr>
<td>5.</td>
<td>E := elected(e);</td>
</tr>
<tr>
<td>6.</td>
<td>while E not empty do</td>
</tr>
<tr>
<td>7.</td>
<td>prune_encoding(E);</td>
</tr>
<tr>
<td>8.</td>
<td>sort(E);</td>
</tr>
<tr>
<td>9.</td>
<td>S := required_attributes(E);</td>
</tr>
<tr>
<td>10.</td>
<td>usage(S);</td>
</tr>
<tr>
<td>11.</td>
<td>sort_attributes(S);</td>
</tr>
<tr>
<td>12.</td>
<td>for all a ∈ S do</td>
</tr>
<tr>
<td>13.</td>
<td>for all sd ∈ SubDomains(a) do</td>
</tr>
<tr>
<td>14.</td>
<td>if (U(sd) = 0) and (N &lt; W) then</td>
</tr>
<tr>
<td>15.</td>
<td>merge(sd, P_0);</td>
</tr>
<tr>
<td>16.</td>
<td>end if</td>
</tr>
<tr>
<td>17.</td>
<td>if (U(sd) = k) and (N &lt; W) then</td>
</tr>
<tr>
<td>18.</td>
<td>merge(sd, P_k);</td>
</tr>
<tr>
<td>19.</td>
<td>end if</td>
</tr>
<tr>
<td>20.</td>
<td>end for</td>
</tr>
<tr>
<td>21.</td>
<td>end for</td>
</tr>
<tr>
<td>22.</td>
<td>while k &gt; 0 do</td>
</tr>
<tr>
<td>23.</td>
<td>for all a ∈ S do</td>
</tr>
<tr>
<td>24.</td>
<td>for all sd ∈ SubDomains(a) do</td>
</tr>
<tr>
<td>25.</td>
<td>if (N &lt; W) then</td>
</tr>
<tr>
<td>26.</td>
<td>merge(sd, P_0);</td>
</tr>
<tr>
<td>27.</td>
<td>else</td>
</tr>
<tr>
<td>28.</td>
<td>merge(sd, P_k);</td>
</tr>
<tr>
<td>29.</td>
<td>end if</td>
</tr>
<tr>
<td>30.</td>
<td>end for</td>
</tr>
<tr>
<td>31.</td>
<td>k := k - 1;</td>
</tr>
<tr>
<td>32.</td>
<td>end while</td>
</tr>
<tr>
<td>33.</td>
<td>split_disjoint();</td>
</tr>
<tr>
<td>34.</td>
<td>e := e + 1;</td>
</tr>
<tr>
<td>35.</td>
<td>E := elected(e);</td>
</tr>
<tr>
<td>36.</td>
<td>end while</td>
</tr>
</tbody>
</table>

MVPP

Generate encoding

Group Queries

Electing Queries

Merge (by usage)

Split (by usage)

Elect successors

Cost Model

HDP schema
Grouping Queries & Generating Encoding

Elect queries & Prune encoding

Sort elected queries by Cost & attributes by usage

MVPP

Generate encoding

Group Queries

Electing Queries

Merge (by usage)

Split (by usage)

If (N<W)

Cost Model

Elect successors

HDP schema

Motivation
Algebra
Electing Queries for HDP
Experimental Study
Conclusion & Future work

Algorithm EQHDP

1: generate_encoding();
2: EQA();
3: e := 1;
4: split_all();
5: E := elected(e);
6: while E not empty do
7:   prune_encoding(E);
8:   sort(E);
9:   S := required_attributes(E);
10:  usage(S);
11:  sort_attributes(S);
12:  for all a ∈ S do
13:      for all sd ∈ SubDomains(a) do
14:         if ((U(sd) = 0) and (N < W)) then
15:            merge(sd, P₀);
16:         end if
17:         if ((U(sd) = k) and (N < W)) then
18:            merge(sd, P_k);
19:         end if
20:      end for
21:   end for
22:  while k > 0 do
23:     for all a ∈ S do
24:        for all sd ∈ SubDomains(a) do
25:           if (N < W) then
26:              merge(sd, P₀);
27:           else
28:              merge(sd, P_k);
29:           end if
30:        end for
31:     end for
32:     k := k - 1;
33:  end while
34:  split_disjoint();
35:  e := e + 1;
36:  E := elected(e);
37: end while
Electing Queries for HDP

Motivation
Algebra
Electing Queries for HDP
Experimental Study
Conclusion & Future work

Grouping Queries & Generating Encoding
Elect queries & Prune encoding
Sort elected queries by Cost & attributes by usage
From highest usage rate (by elected queries):
Merge subdomains
Having the same rate

Algorithm EQHDP
1: generate_encoding();
2: EQA();
3: e := 1;
4: split_all();
5: E := elected(e);
6: while E not empty do
7:    prune_encoding(E);
8:    sort(E);
9:    S := required_attributes(E);
10:   usage(S);
11:   sort_attributes(S);
12:   for all a ∈ S do
13:     for all sd ∈ SubDomains(a) do
14:       if ((U(sd) = 0) && (N < W)) then
15:         merge(sd, P0);
16:       end if
17:       if ((U(sd) = k) && (N < W)) then
18:         merge(sd, Pk);
19:       end if
20:     end for
21:   end for
22:   while k > 0 do
23:     for all a ∈ S do
24:       for all sd ∈ SubDomains(a) do
25:         if (N < W) then
26:           merge(sd, P0);
27:         else
28:           merge(sd, Pk);
29:         end if
30:       end if
31:     end for
32:     k := k - 1;
33:   end while
34:   split_disjoint();
35:   e := e + 1;
36:   E := elected(e);
37: end while

Cost Model

HDP schema
Electing Queries for HDP

**Motivation**

**Algebra**

**Experimental Study**

**Conclusion & Future work**

**Grouping Queries** & Generating Encoding

**Electing Queries**

**Generating Encoding**

**Group Queries**

**Electing Queries**

**Generate encoding**

**Group Queries**

**Electing Queries**

**Merge (by usage)**

**Split (by usage)**

**Elect successors**

**Cost Model**

**HDP schema**

```
Algorithm EQHDP
1: generate_encoding();
2: EQA();
3: e := 1;
4: split_all();
5: E := elected(e);
6: while E not empty do
7:    prune_encoding(E);
8:    sort(E);
9:    S := required_attributes(E);
10:   usage(S);
11:  sort_attributes(S);
12:  for all a ∈ S do
13:      for all sd ∈ SubDomains(a) do
14:         if ((U(sd) = 0) and (N < W)) then
15:            merge(sd, P0);
16:         end if
17:         if ((U(sd) = k) and (N < W)) then
18:            merge(sd, Pk);
19:         end if
20:      end for
21:   end for
22:   while k > 0 do
23:      for all a ∈ S do
24:         for all sd ∈ SubDomains(a) do
25:            if (N < W) then
26:               merge(sd, P0);
27:            else
28:               merge(sd, Pk);
29:            end if
30:         end for
31:      end for
32:      k := k - 1;
33:   end while
34:   split_disjoint();
35:   e := e + 1;
36:   E := elected(e);
37: end while
```

From highest usage rate (by elected queries): Merge subdomains Having the same rate

Sort elected queries by Cost & attributes by usage

Improvement: move to successors of elected queries (e ← e+1)
SSB of 100 GB

Workload1: 12 queries (no interaction)
Workload2: 22 queries (with interaction)

Oracle11g DBMS
Server of 32 GB of RAM
Intel Xeon CPU: 2x2.45 GHz
Experimental Study

Motivation
Algebra
E lecting Queries for HDP
Experimental Study
Conclusion & Future work

Impact of query interaction on performance

EQHDP Vs. SA

Impact of incremental encoding on performance

Impact of query interaction on performance

Number of Split/Merge to reach the solution
Experimental Study

Motivation
Algebra
Electing Queries for HDP
Experimental Study
Conclusion & Future work

Best Selectivity Factor intervals
Improving EQHDP by SF interval

Validating algorithms’ performance on Oracle11g
Scaling-Up and impact of data volume
Conclusion & Future Work

- Optimization in RDW by HDP
- Considering query interaction
- Incremental encoding for representing schemas
- Pruning predicates and steering HDP by elected queries
  - Considering query interaction in other optimization techniques
  - Include MVPP optimization in Physical Design
  - InterPhase project
Thank you