MULTENANT-DATABASES

Kaushik Mukherjee

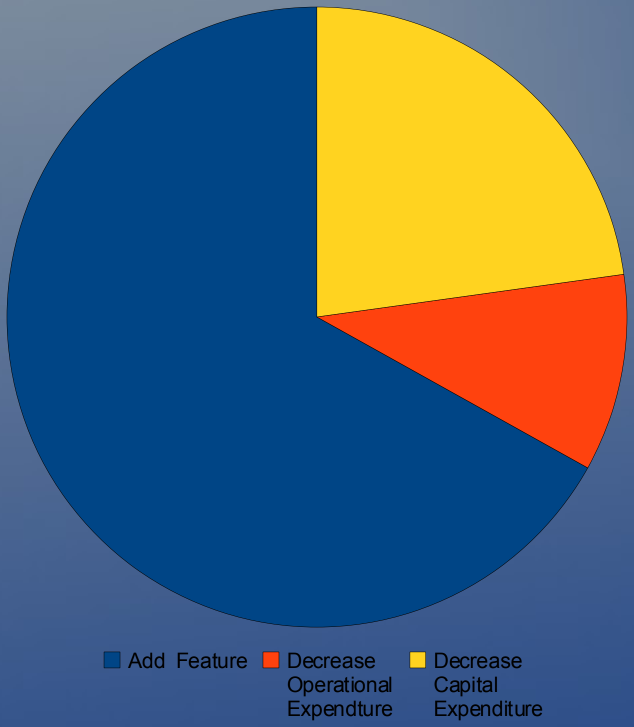
23rd Nov 2010

Outline

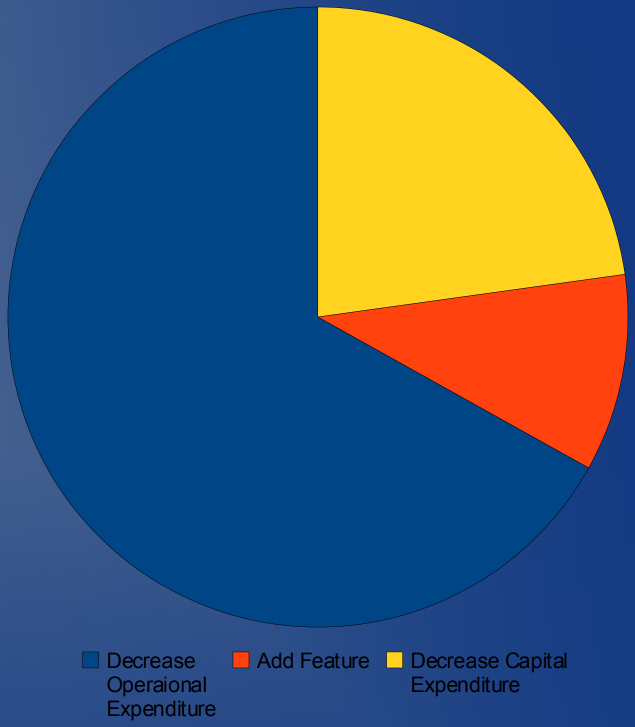
- **Motivation**
  - Software as a Service
  - Need for Multi-tenancy
  - Need for Multi-tenant Databases
- **Multi-tenant Databases**
  - Challenges
  - Design Approaches and tradeoff
  - Experiment
- **Discussion**
- **Conclusion**

Today's Software Service Industry

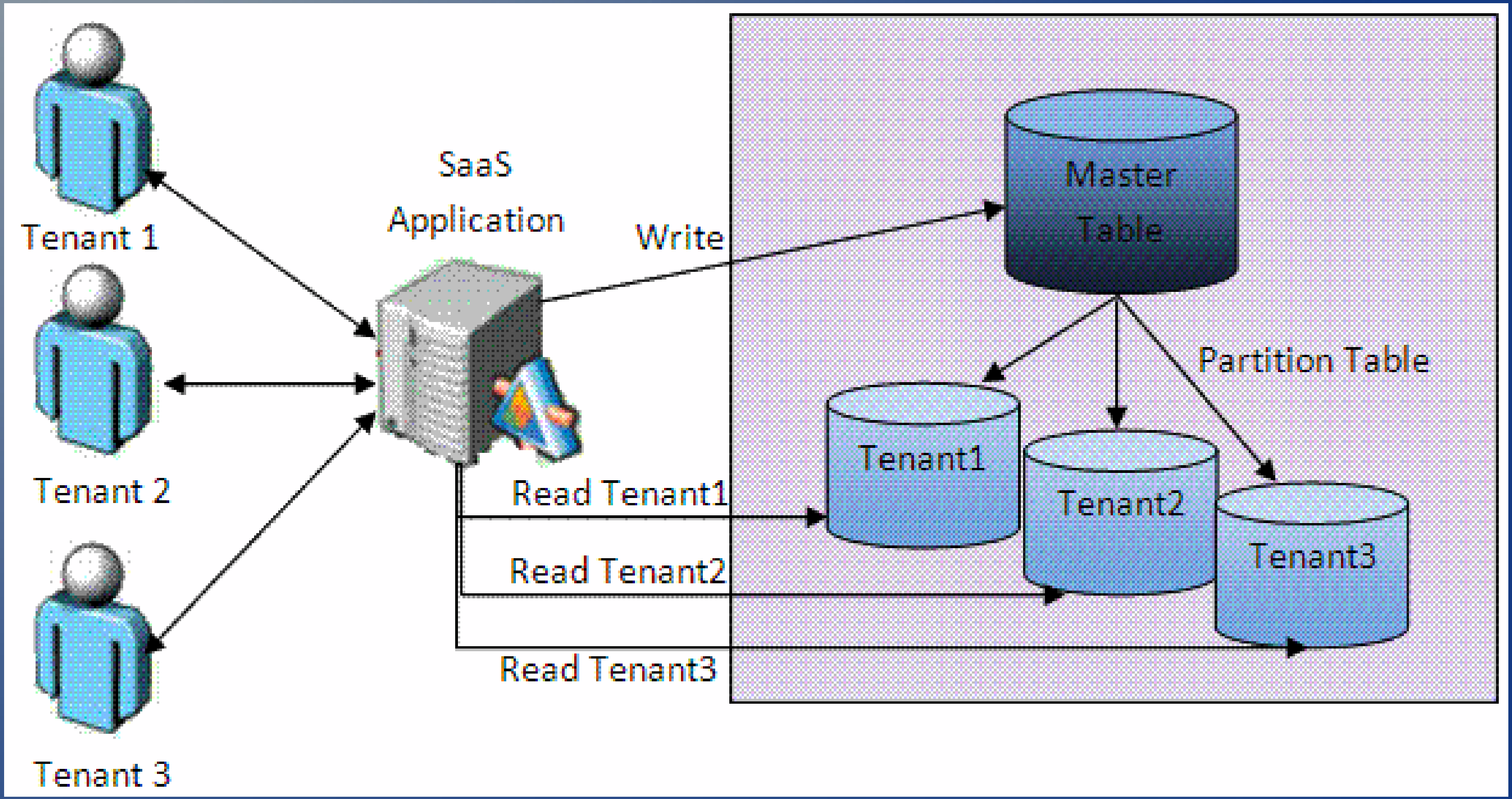
- **On Premise Software**



- **Software as a Service**



Multi-tenancy



Why Multi-tenant databases ?

- Consolidating multiple databases onto same operational system

- Reduces Total Cost of Ownership

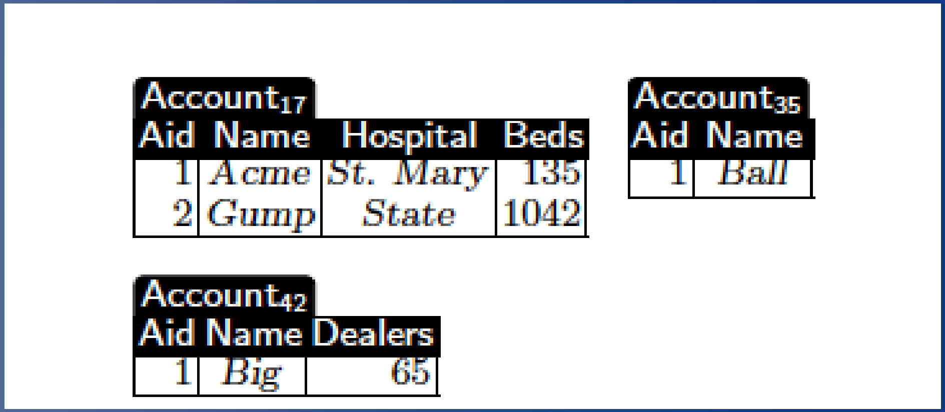
Challenges in Multi-tenant Database

- **Scalability**
  - Tradeoff between cost handling many tables and cost query rewriting

- **Allow Schema Extensibility**
  - Multiple tenant share tables
  - Need for tenant specific schema extensibility

Design Approaches

- **Private Table**
  - Natural Thing to do - each tenant gets a private schema
  - Low cost on query transformation
  - Less consolidation



Account$_{17}$

Aid	Name	Hospital	Beds
1	*Acme*	*St. Mary*	135
2	*Gump*	*State*	1042

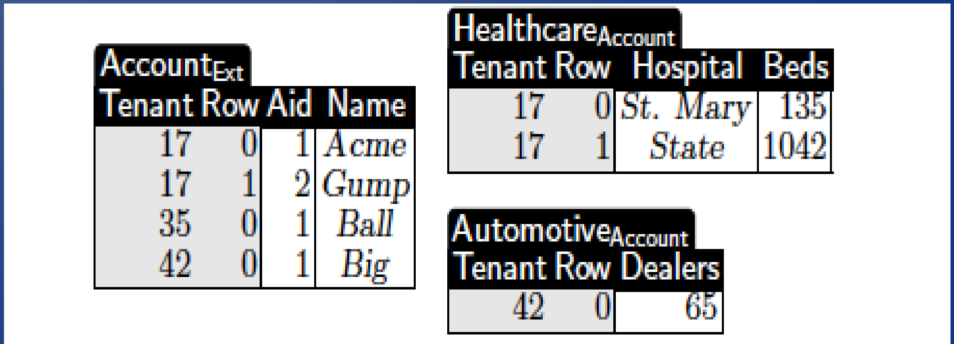
Account$_{35}$

Aid	Name
1	*Ball*

Account$_{42}$

Aid	Name	Dealers
1	*Big*	65

- **Extension Table**
  - Split off extensions into separate tables
  - Higher cost on Query transformation
  - Slightly better consolidation



Account$_{Ext}$

Tenant	Row	Aid	Name
17	0	1	*Acme*
17	1	2	*Gump*
35	0	1	*Ball*
42	0	1	*Big*

Healthcare$_{Account}$

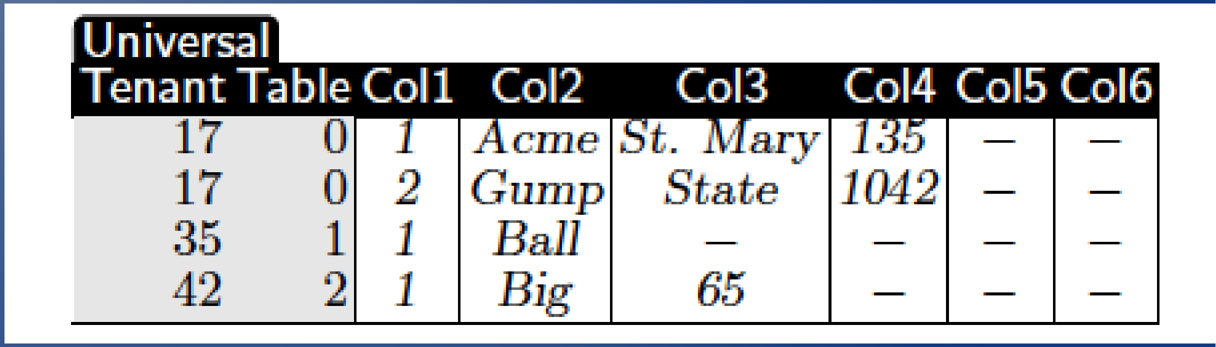
Tenant	Row	Hospital	Beds
17	0	*St. Mary*	135
17	1	*State*	1042

Automotive$_{Account}$

Tenant	Row	Dealers
42	0	65

Universal Table

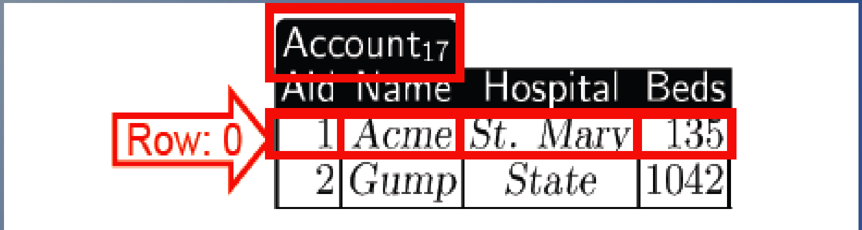
- **Generic Structure with VARCHAR value columns**
  - n-th Column of the logical table is mapped to Col-n in the universal table
  - Extensibility
- **Disadvantages**
  - Many *Null* Values
  - Not type safe
  - No Indexing



Universal

Tenant	Table	Col1	Col2	Col3	Col4	Col5	Col6
17	0	*1*	*Acme*	*St. Mary*	*135*	–	–
17	0	2	*Gump*	*State*	*1042*	–	–
35	1	*1*	*Ball*	–	–	–	–
42	2	*1*	*Big*	*65*	–	–	–

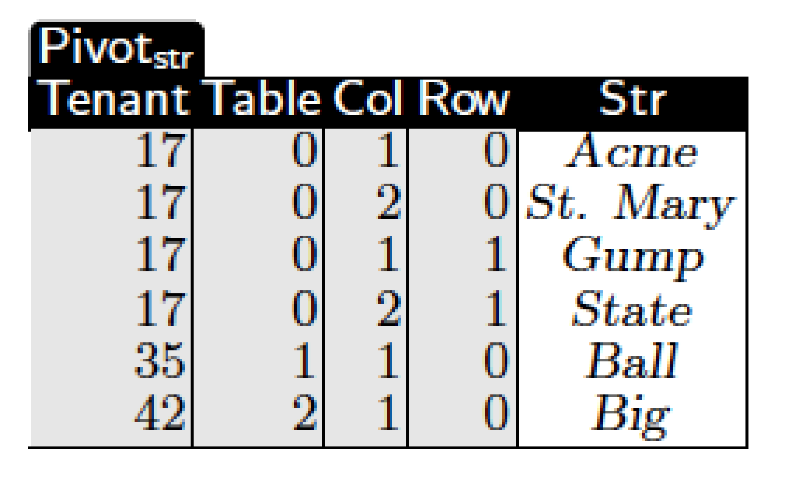
Pivot Table



Row: 0

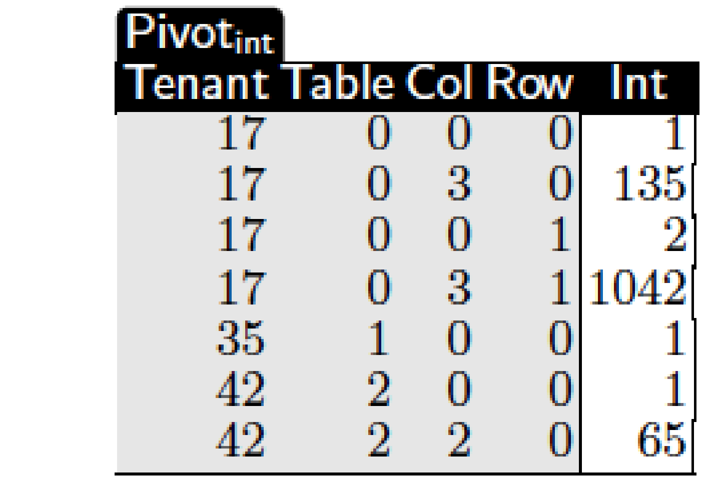
Account$_{17}$

Aid	Name	Hospital	Beds
1	*Acme*	*St. Mary*	135
2	*Gump*	*State*	1042



Pivot$_{str}$

Tenant	Table	Col	Row	Str
17	0	1	0	*Acme*
17	0	2	0	*St. Mary*
17	0	1	1	*Gump*
17	0	2	1	*State*
35	1	1	0	*Ball*
42	2	1	0	*Big*

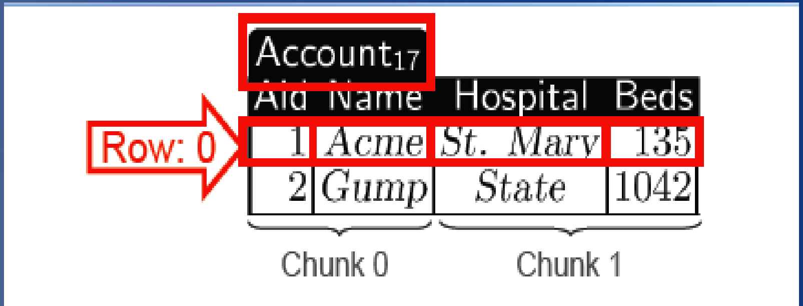


Pivot$_{int}$

Tenant	Table	Col	Row	Int
17	0	0	0	1
17	0	3	0	135
17	0	0	1	2
17	0	3	1	1042
35	1	0	0	1
42	2	0	0	1
42	2	2	0	65

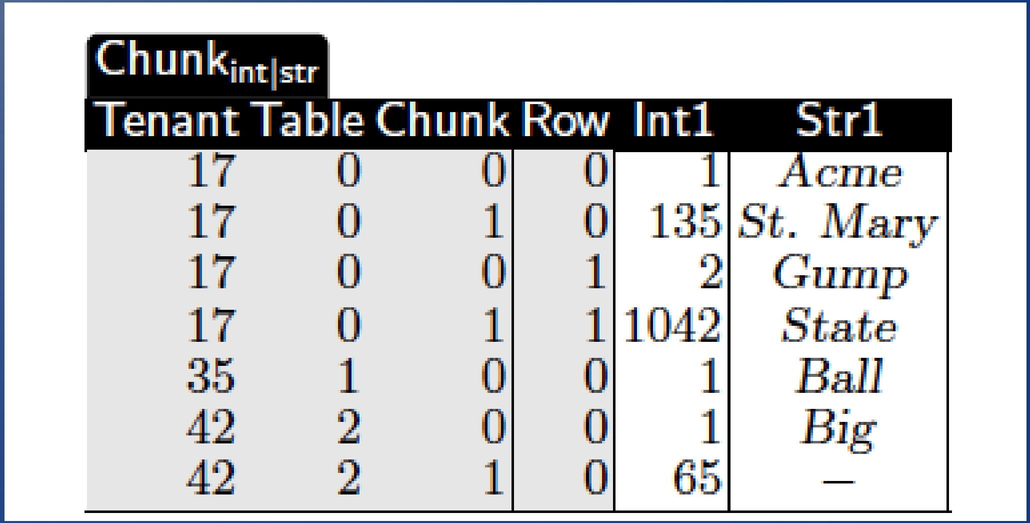
Chunk Table

- **Generic Structure**
  - Suitable if data-set can be partitioned into dense subsets
  - Derived from Pivot table
- **Performance**
  - Fewer joins for reconstruction if densely populated subsets can be extracted
  - Reduced meta-data/data ratio dependent on the chunk size
  - Indexable



Account$_{17}$

Aid	Name	Hospital	Beds
1	*Acme*	*St. Mary*	135
2	*Gump*	*State*	1042

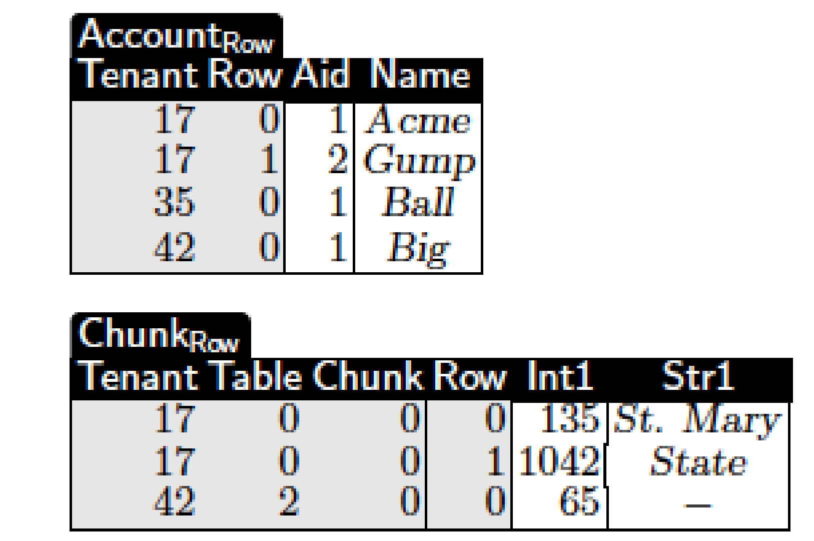


Chunk$_{int|str}$

Tenant	Table	Chunk	Row	Int1	Str1
17	0	0	0	1	*Acme*
17	0	1	0	135	*St. Mary*
17	0	0	1	2	*Gump*
17	0	1	1	1042	*State*
35	1	0	0	1	*Ball*
42	2	0	0	1	*Big*
42	2	1	0	65	—

Row Fragmentation

- **Combine different schema mappings for getting best fit**
  - Mixes Extension and Chunk Tables
  - Each fragment can be stored in an optimal schema layout
- **Optimal row fragmentation depends on**
  - Workload
  - Data distribution
  - Data popularity



$\text{Account}_{\text{Row}}$

Tenant	Row	Aid	Name
17	0	1	*Acme*
17	1	2	*Gump*
35	0	1	*Ball*
42	0	1	*Big*

$\text{Chunk}_{\text{Row}}$

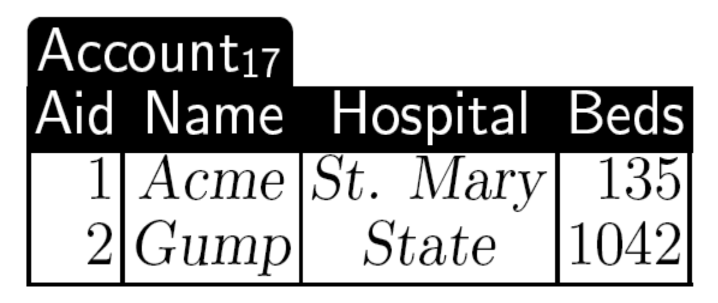
Tenant	Table	Chunk	Row	Int1	Str1
17	0	0	0	135	*St. Mary*
17	0	0	1	1042	*State*
42	2	0	0	65	–

Query Transformation

- **Reconstructing original tav requires many equi-joins**
- **Source Query**

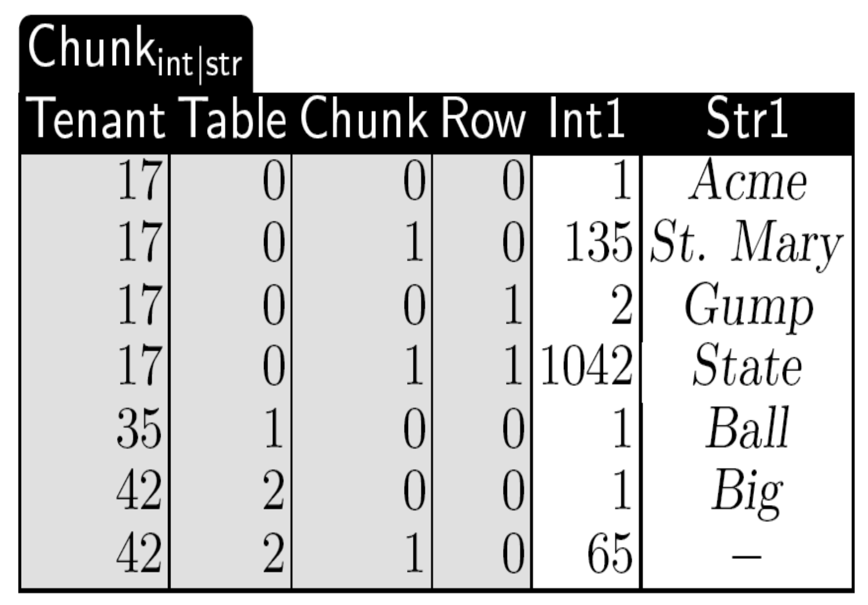
```
SELECT Beds
FROM Account17
WHERE Hospital = 'State'
```

- **Collect table and column names**
  - Account17 : Beds , Hospital
- **Obtain chunk tables and meta-data**
  - Chunk (int|str)
  - Account17 :
    - Table = 0, Tenant =17
  - Beds , Hospital :
    - Chunk =1



Account$_{17}$

Aid	Name	Hospital	Beds
1	*Acme*	*St. Mary*	135
2	*Gump*	*State*	1042



Chunk$_{int|str}$

Tenant	Table	Chunk	Row	Int1	Str1
17	0	0	0	1	*Acme*
17	0	1	0	135	*St. Mary*
17	0	0	1	2	*Gump*
17	0	1	1	1042	*State*
35	1	0	0	1	*Ball*
42	2	0	0	1	*Big*
42	2	1	0	65	–

Query Transformation

- **Generate filter query**

```
SELECT Str1 as Hospital , Int1 as Beds
FROM Chunk (int|str)
WHERE Tenant = 17 AND Table = 0 AND Chunk = 1
```

- **Replace reference in source query**

```
SELECT Beds FROM
(SELECT Str1 as Hospital Int1 as Beds
FROM Chunk (int|str) WHERE Tenant = 17
AND Table =0 AND Chunk =1) As Account17
WHERE Hospital = 'State'
```

Query Transformation

- **Structural Changes**
  - Additional Nesting
  - Joins
  - Base Table Access
- **Impact on Performance**
  - Nesting can be flattened by query optimizer
  - Joins are cheaper only if the cost of loading the chunks and applying index supported join are cheaper that loading wider conventional relation
  - Meta data columns in base tables have indexing support

Query Evaluation Experiment

- **Goal**
  - Show if the query transformation can handle issues of scalibility
  - Evaluate impact of Join overhead
  - Evaluate impact of meta-data overhead
- **Test Query**

```
SELECT p.id, ...
     FROM parent p , child c
     WHERE p.id = c.parent
          AND p.id = ?.
```

Query Evaluation Experiments

- **Conventional Schema**

  Parent

  id col1 col2 ... col90

  Child

  id Parent col1 col2...col90

- **Chunk Schema**

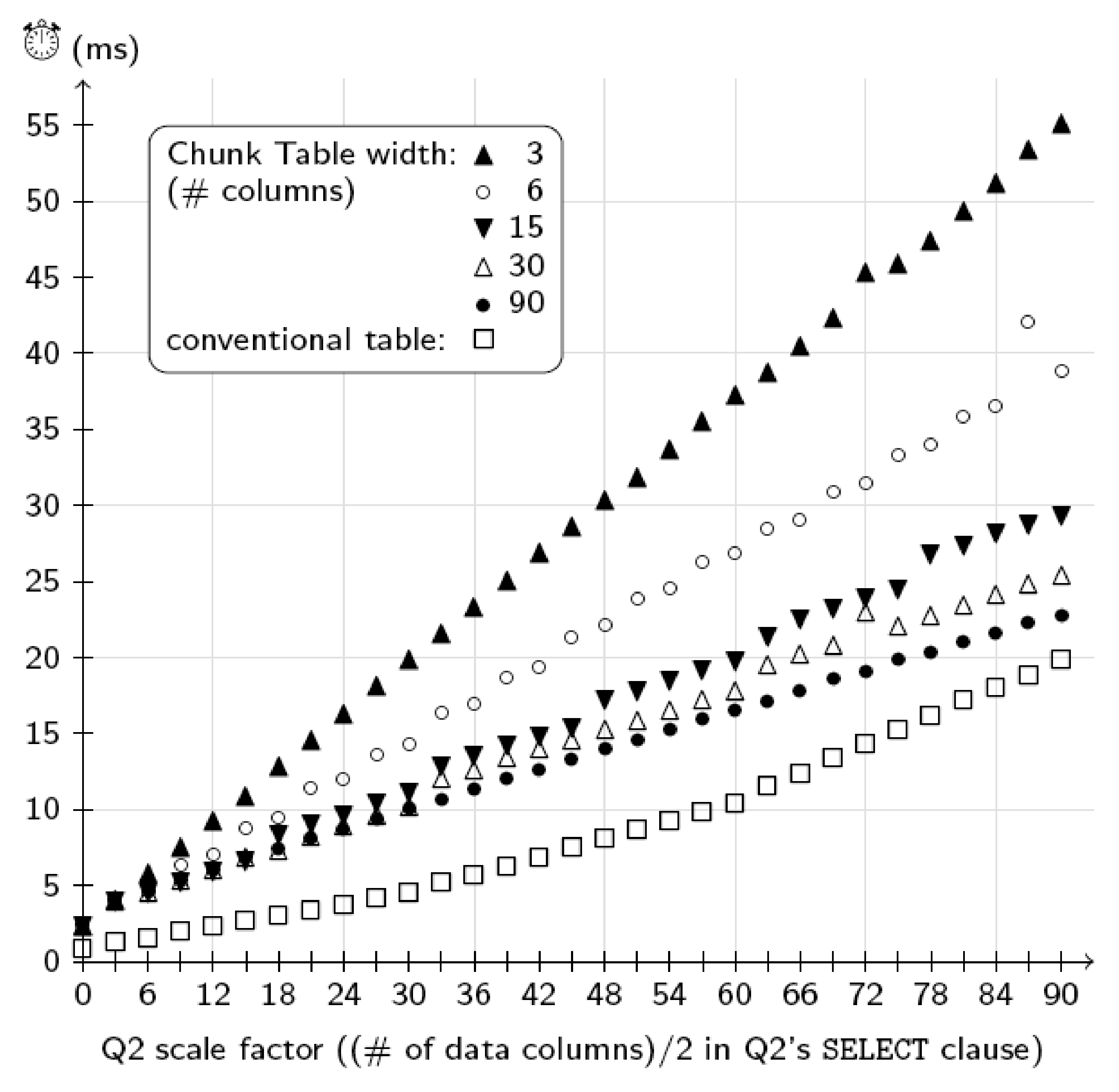
  Chunk$_{Data}$

  table chunk row int1 int2 int2 date date2 str1 str2

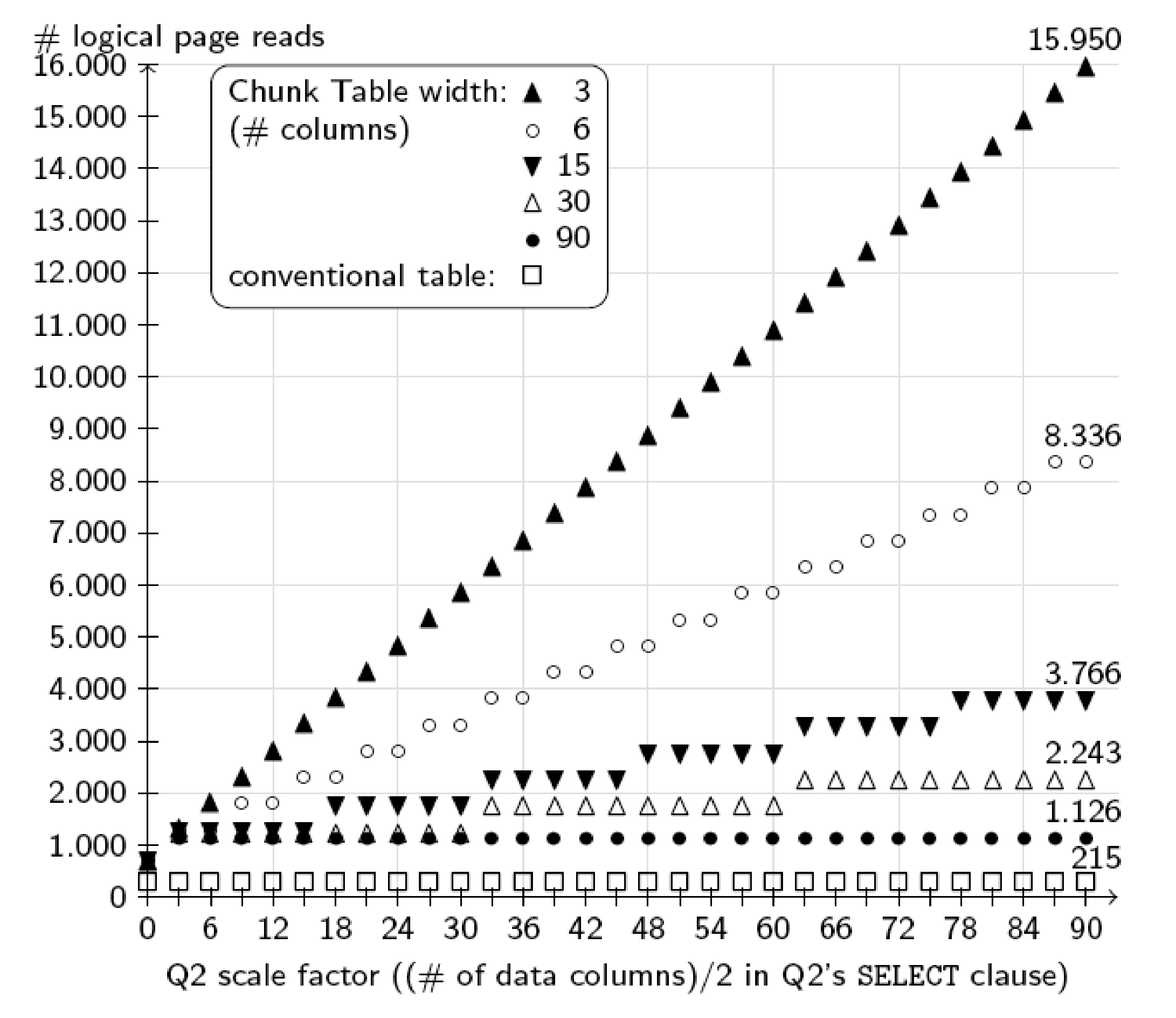
  Chunk$_{Index}$

  table chunk row int1

Join Overhead Costs



Meta-Data Costs



Discussion

- **Strengths**
  - Chunk tables is a good design for trade-of extensibility and meta data usage.
  - Chunk tables gives response time improvement over vertical partitioning

- **ShortComings/Future work**
  - No Algorithms to design chunk tables
  - Identifying the chunks is heuristic
  - No comparative experiment done with the other schema mapping techniques proposed in the paper

Conclusion

- Is chunk tables a good approach for designing multi-tenant databases?
- How practical it is for real life systems ?
- How do companies like Salesforce.com handle it ?

THANK YOU

QUESTIONS ?