CS 848 Paper Presentation

A comparison of flexible schemas for software as a service

Stefan Aulbach, Dean Jacobs, Alfons Kemper, and Michael Seibold.

Presented by Bilal Sheikh

David R. Cheriton School of Computer Science University of Waterloo

15th March 2010

Software as a service (SaaS)

- A service provider owns and operates an application
- Accessed by many clients/business organizations

Google docs

salesforce.com
Success On Demand.™

Motivation for SaaS

- Leverage economy of scale to reduce ownership cost
- To avoid paying huge setup costs
- On demand licence and management of software
- Pay as you go
- Centralized feature update

Motivation for Multi-tenant DBs

- SaaS maturity Levels:
  - Ad-hoc/custom – Each client has a separate instance of application
    -
    -
  - Scalable, configurable and built with multi-tenant back-ends enabling multiple users and clients to **access shared data model**

- Multi-Tenant Databases:
  - Single database to provide shared data access to multiple clients/organizations

Requirements for Multi-tenant DBs

- Supporting multiple schemas
- Base Schema Extensibility
- On-line evolution schemas
- Performance
- Scalability

Categories

Database "owns" schema	Application "owns" schema
Good performance	Poor performance
Limited schema evolution	Schema evolution flexibility
Scale to a certain point	Better scalability

Techniques

- Database owns schema
  - Private Tables
  - Extension Tables
  - Sparse Columns } 

- Application owns schema
  - XML } 

  - Pivot Tables } 

Private Tables (base case)

- Each tenant with a private instance of base table with appropriate extensions
- Base case for comparison with other schemes

$Account_{17}$

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

(a) Private Tables

Extension Tables

- Vertically partitioned
- Separate table for each extension
- Joined to base table by row id

$Account_{Ext}$

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

Sparse Columns

- Store each value with an associated column identifier
- Explicitly created by CREATE/ALTER table statements
- Application to ensure that each tenant uses its own defined columns

Account

Tenant	Aid	Name	SPARSE			
17	1	Acme	Hospital	St. Mary	Bed	135
17	2	Gump	Hospital	State	Bed	1042
35	1	Ball				
42	1	Big	Dealer	65		

XML in IBM DB2

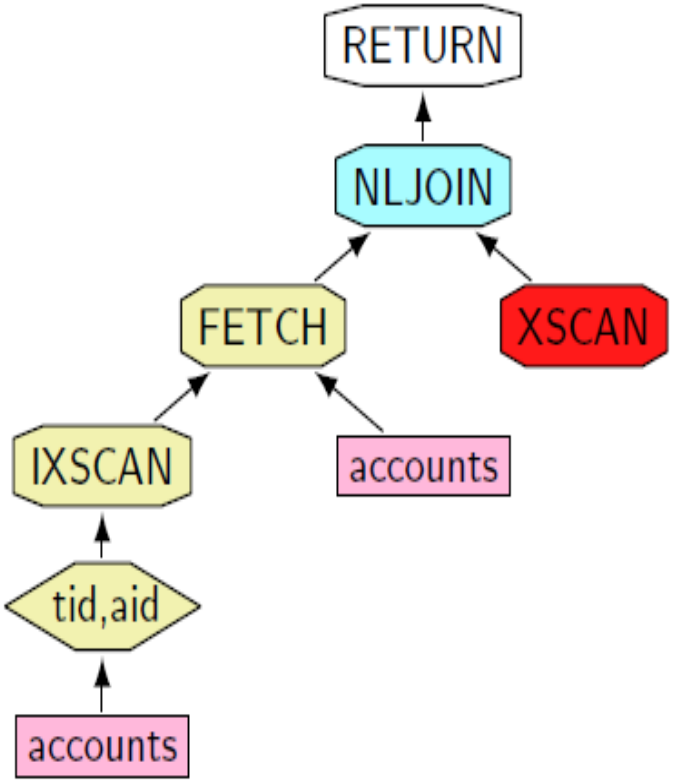
- All tenants share base tables
- A single extension column storing all extensions as XML
- Untyped documents to keep size small
- In-place updates using XQuery

Account

Tenant	Aid	Name	Ext_XML
17	1	*Acme*	<ext><hospital>St. Mary</hospital> <beds>135</beds></ext>
17	2	*Gump*	<ext><hospital>State</hospital> <beds>1042</beds></ext>
35	1	*Ball*	
42	1	*Big*	<ext><dealers>65</dealers></ext>

XML in IBM DB2 (continued ...)

```
SELECT b.Tenant, b.Aid, b.Name,
       e.Dealers
FROM   Accounts b,
       XMLTABLE('i/ext' PASSING b.Ext_XML AS "i'
         COLUMNS
           Dealers INTEGER PATH 'dealers'
       ) AS e
WHERE Tenant = 42 AND Aid = 1;
```



Pivot Tables (HBase)

- Column families pre-defined as DDL
- Columns can be dynamically added to column families
- No facility of joins, sorts and groups
- Only row-at-a-time operations
- Run on single machine without Hadoop
- Compression & replication turned off

RowKey	Account	Contact
17Act1	[name:*Acme*, hospital:*St. Mary*, beds:*135*]	
17Act2	[name:*Gump*, hospital:*State*, beds:*1042*]	
17Ctc1		[· · ·]
17Ctc2		[· · ·]
35Act1	[name:*Ball*]	
35Ctc1		[· · ·]
42Act1	[name:*Big*, dealers:*65*]	

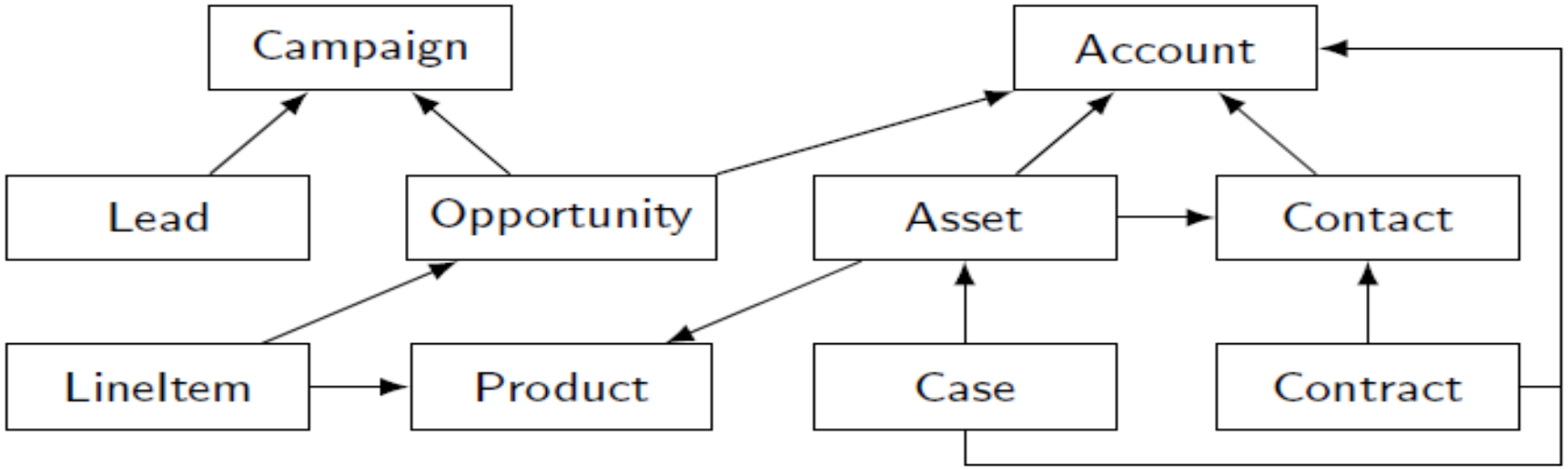
Pivot Tables (continued ...)

```
SELECT   p.Name, COUNT(c.Case_id) AS cases
FROM     Products p, Assets a, Cases c
WHERE    c.Asset = a.Asset_id
  AND    a.Product = p.Product_id
GROUP BY p.Name
ORDER BY cases DESC
```

Testbed For Multi-Tenant Databases

Multi-Tenant Database Testbed

- A CRM application service
- Single & multi-row creates, reads and updates
- Schema extension for tenants
- Dynamic Schema Evolution
- Mimics multiple web-hosts



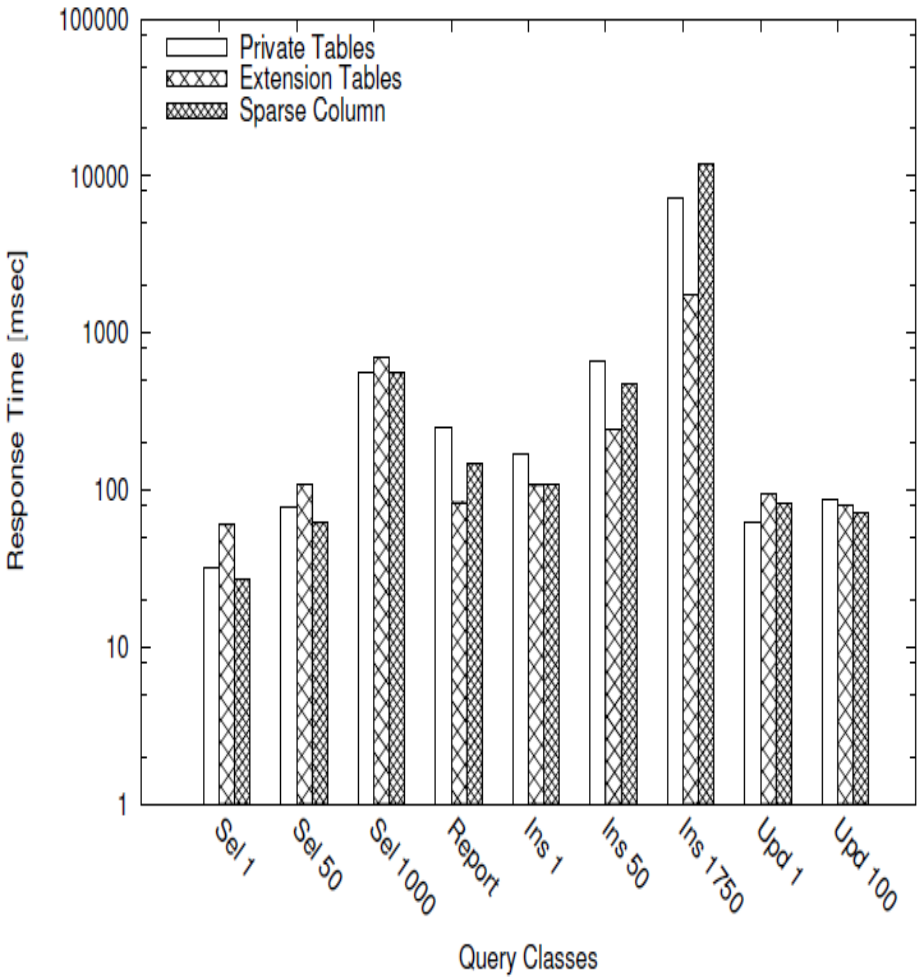
Benchmark Queries

- Select 1: Select all attributes of a single entity
- Select 50:
- Select 1000:
- Reporting: Run one of five reporting queries that perform aggregation and/or parent-child-roll-ups.
- Insert 1: Insert one new entity instance
- Insert 50:
- Insert 1750:
- Update 1: Update a single entity
- Update 100:

Experimental Results [Microsoft SQL Server]

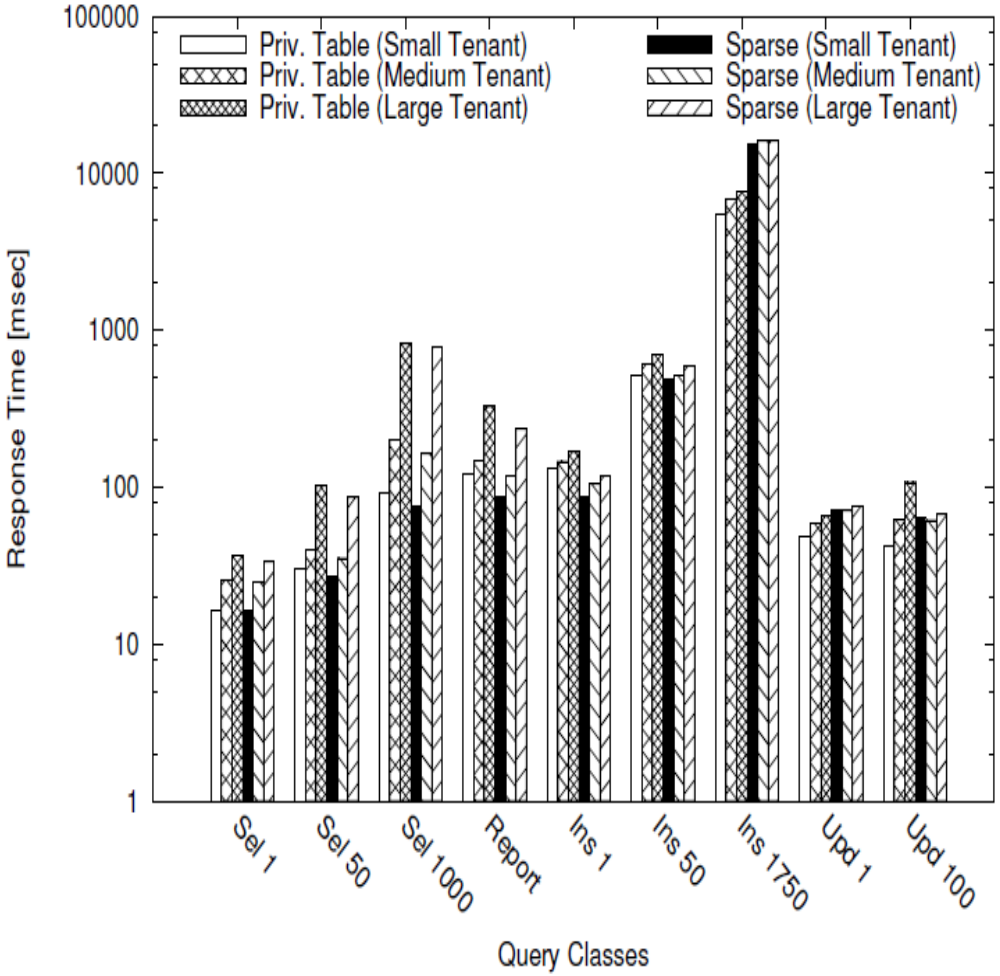
Private Tables vs Extensions vs Sparse Columns (Response Time)

- Extension table suffer on Selects because of Joins, Updates because have to update multiple tables
- Extension tables perform well for inserts because the base shared is between tenants and most likely in the buffer pool
- Sparse Columns perform same or better than private tables



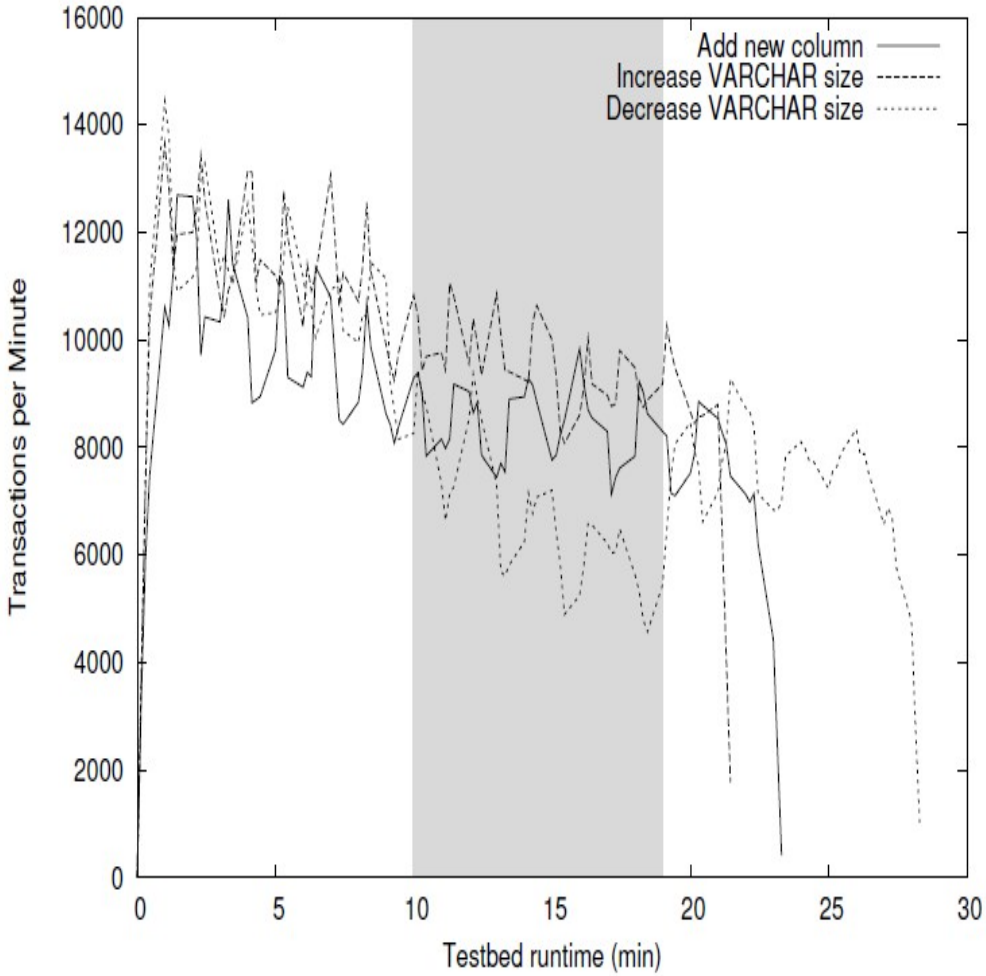
Private Tables vs Sparse Columns (Tenant Size)

- Response time increased with the number of tenants
- Sparse columns consistently outperformed Private Columns except for large inserts case



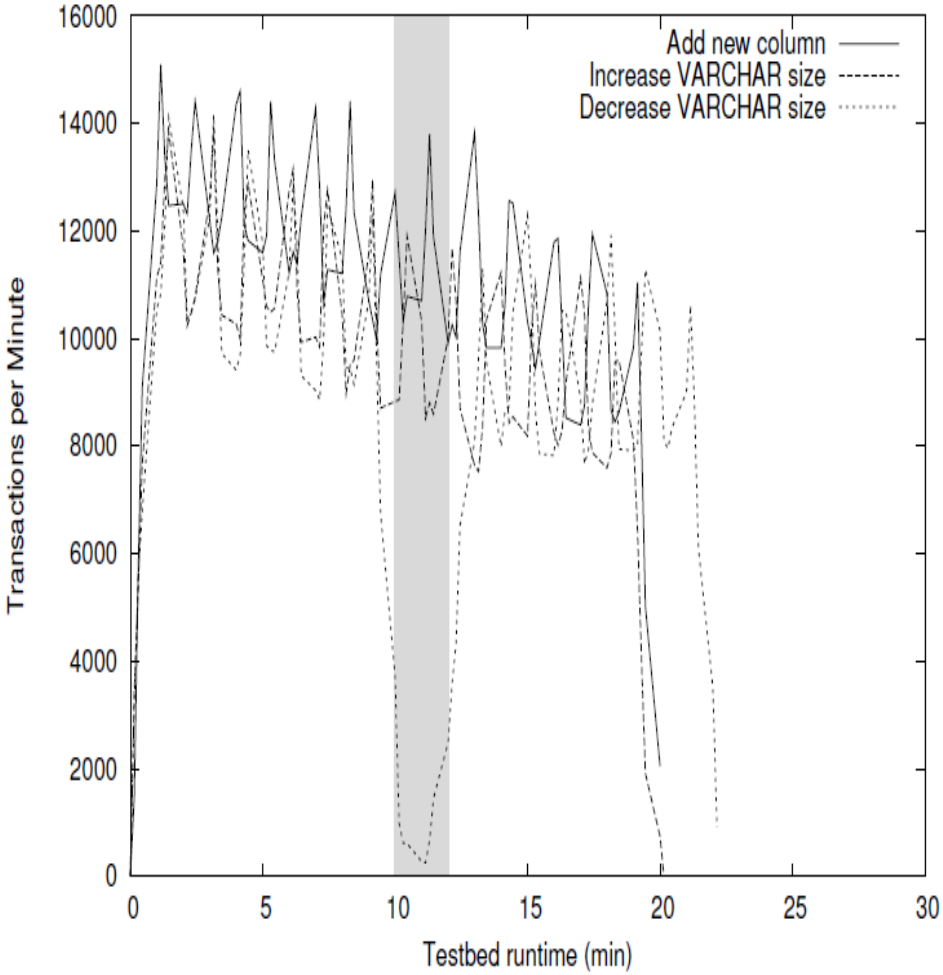
Private Tables (Schema Evolution)

- ALTER TABLE statements
- 5 for each of the 195 tenants
- Add new column, Increase VARCHAR size & Decrease VARCHAR size



Sparse Tables (Schema Evolution)

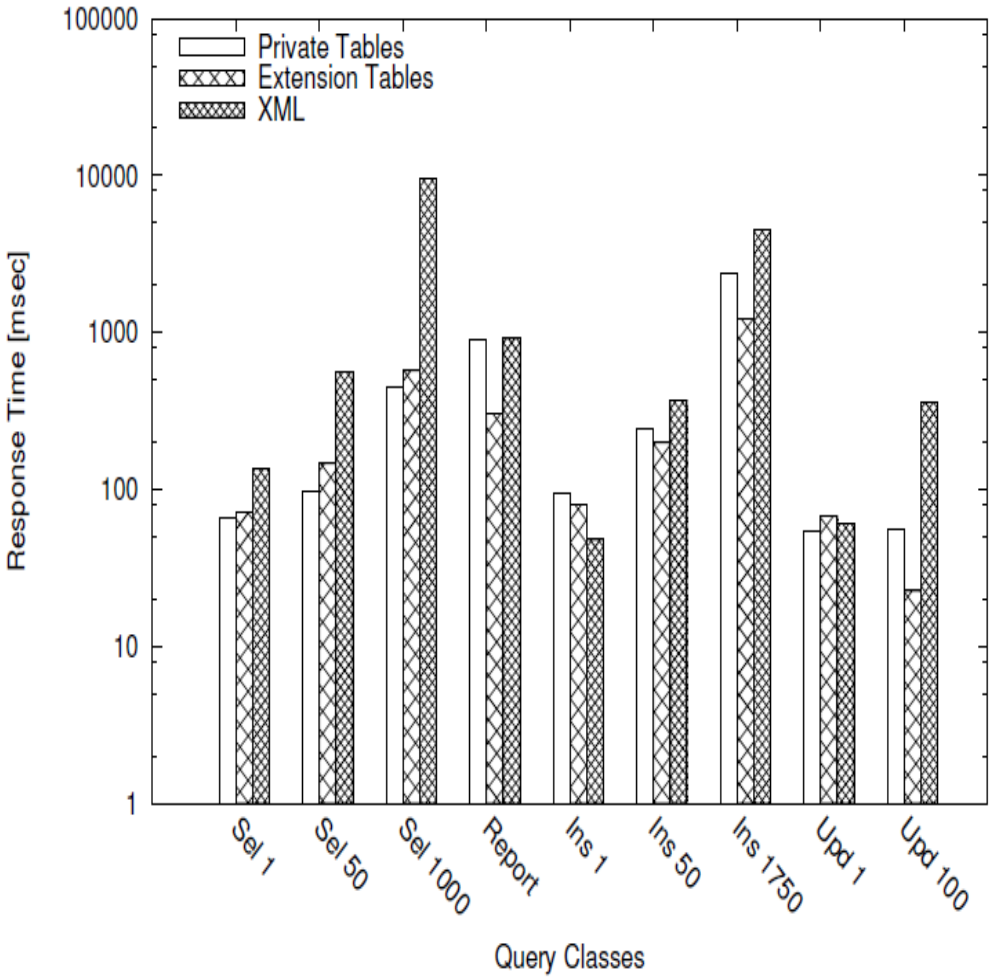
- Tables shared
- 5 ALTER TABLE statements
- Schema-Only alterations had no effect
- Decreasing the size of VARCHAR caused throughput to almost drop to 0
- Less drop in overall transactions/minute than Private Tables



Experimental Results (IBM DB2)

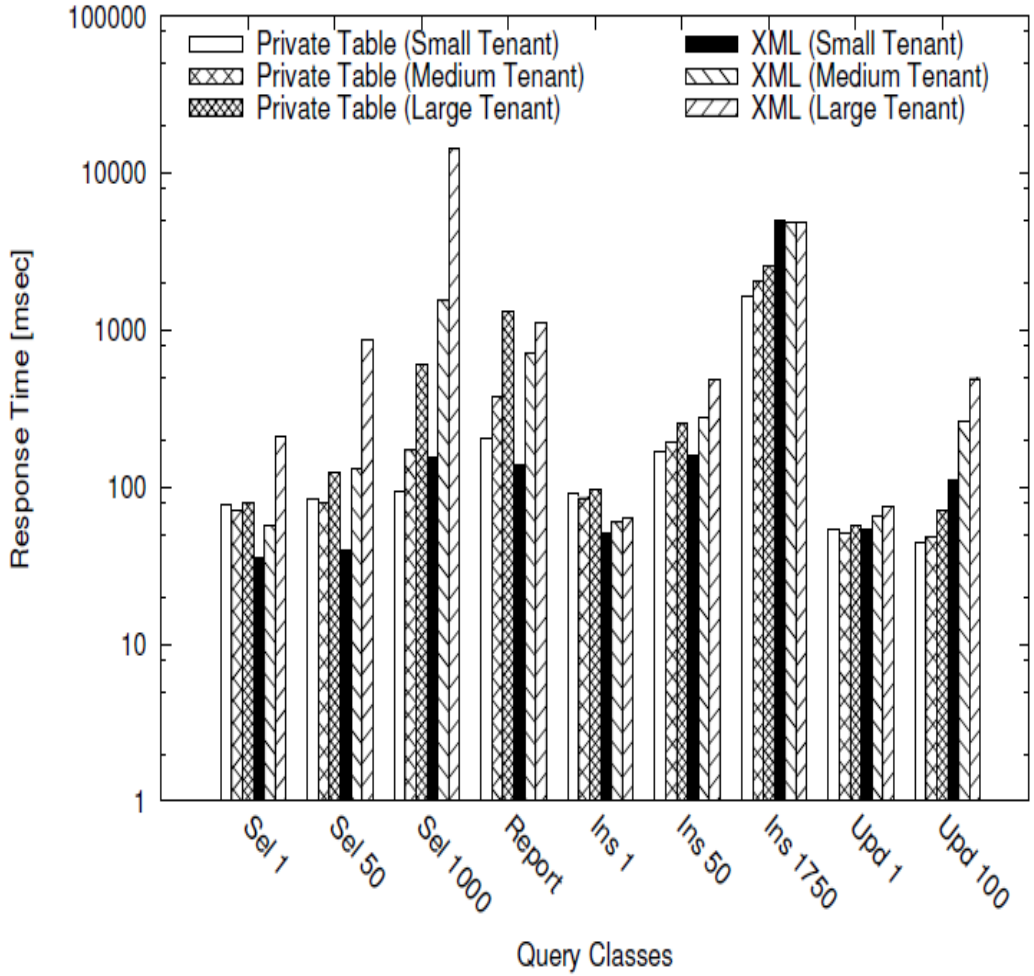
Private Tables vs Extension Tables vs pureXML (Response Time)

- PureXML performed slower overall
- Significantly slower for reads because of nested sub-query with XQuery.
- Performance for Private Tables and Extension Tables were similar to SQL Server



Private Tables vs Extensions vs pureXML (Tenant Size)

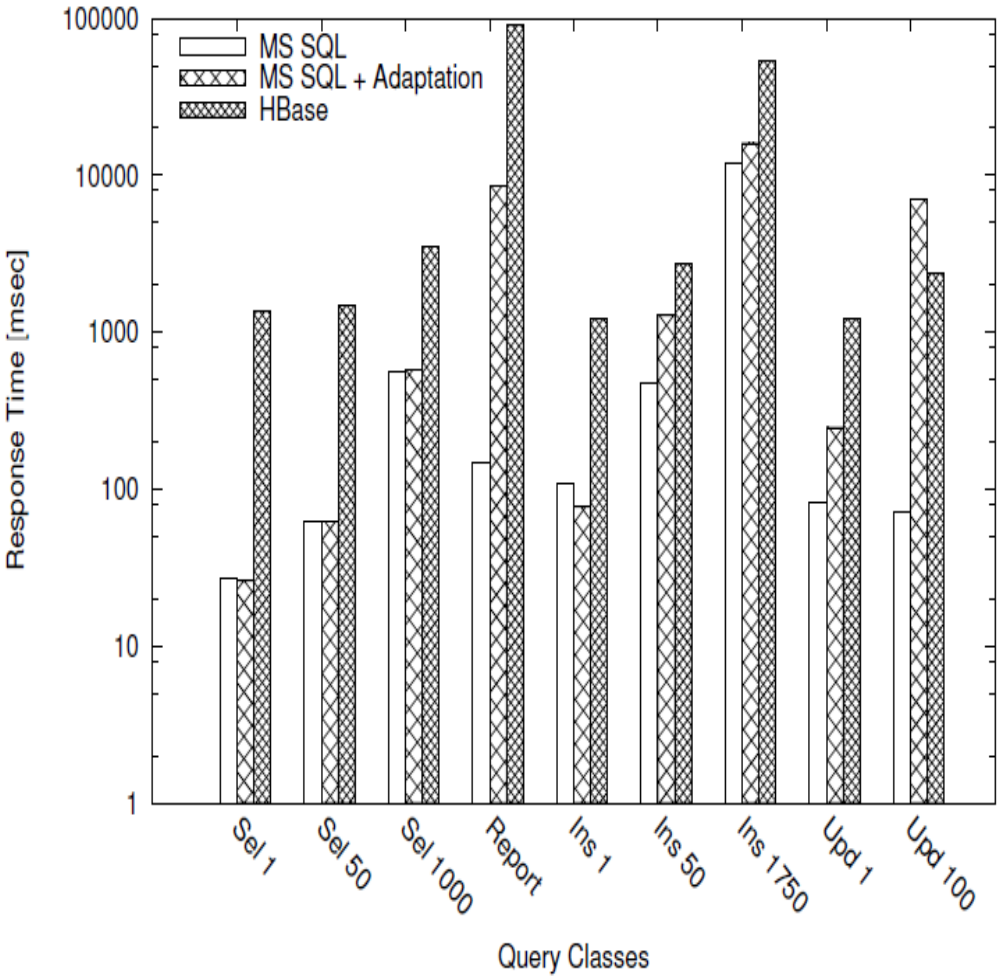
- For small tenant size XML performed better than both private tables and Extension tables but not for medium an large tenant sizes
- Significant impact on PureXM on increase in tenant size
- Schema evolution experimen run on PureXML as with Sparse Columns producing very similar results



Experimental Results (Hadoop HBase)

HBase vs SQL Server (Sparse Columns)

- HBase compared with Sparse columns with and without adaptation layer
- Adaptation layer has significant impact on the performance of both schemas
- Generally Sparse Columr perform much better tha HBase



Ideal Database system for SaaS

- Ideal database system for SaaS has not yet been developed
- Poor performance where application owns schema
- Limited scalability and schema evolution where database owns schema
- Ideal SaaS database should use Private Table mapping and the DDL should be aware of the different extensions as templates and shared schema
- The interleaving of data in other schemes cause breaks natural partitioning and therefore difficult to manipulate
- Challenge: Online schema evolution over existing data. Application should be given more control over such resource intensive processes
- Should be able to distribute data on number of servers and query it

Thank You!