CIS 602-01: Scalable Data Analysis

Databases

Dr. David Koop
Relational Database Architecture

Figure 1.1 Main components of a DBMS.

At heart, a typical RDBMS has five main components, as illustrated in Figure 1.1. As an introduction to each of these components and the way they fit together, we step through the life of a query in a database system. This also serves as an overview of the remaining sections of the paper.

Consider a simple but typical database interaction at an airport, in which a gate agent clicks on a form to request the passenger list for a flight. This button click results in a single-query transaction that works roughly as follows:

1. The personal computer at the airport gate (the “client”) calls an API that in turn communicates over a network to establish a connection with the Client Communications Manager of a DBMS (top of Figure 1.1). In some cases, this connection [Hellerstein et al., Architecture of a Database System]
Parallel DB Architecture: Shared Nothing

A shared-nothing parallel system (Figure 3.2) is made up of a cluster of independent machines that communicate over a high-speed network interconnect or, increasingly frequently, over commodity networking components. There is no way for a given system to directly access the memory or disk of another system.

Shared-nothing systems provide no hardware sharing abstractions, leaving coordination of the various machines entirely in the hands of the DBMS. The most common technique employed by DBMSs to support these clusters is to run their standard process model on each machine, or node, in the cluster. Each node is capable of accepting client SQL requests. [Hellerstein et al., Architecture of a Database System]
Stonebraker: The End of an Architectural Era

- "RDBMSs were designed for the business data processing market, which is their sweet spot"
- "They can be beaten handily in most any other market of significant enough size to warrant the investment in a specialized engine"
- Changes in markets (science), necessary features (scalability), and technology (amount of memory)
- RDBMS Overhead: Logging, Latching, and Locking
- Relational model is not necessarily the answer
- SQL is not necessarily the answer
Problems with Relational Databases

ID: 1001
Customer: Ann

Line Items:
- Order ID: 0321293533, Quantity: 2, Price: $48, Total: $96
- Order ID: 0321601912, Quantity: 1, Price: $39, Total: $39
- Order ID: 0131495054, Quantity: 1, Price: $51, Total: $51

Payment Details:
- Card: Amex
- CC Number: 12345
- Expiry: 04/2001

Orders
Customers
Order Lines
Credit Cards
### Row Stores

The following table illustrates an example of a typical row-store database:

<table>
<thead>
<tr>
<th>id</th>
<th>scientist</th>
<th>death_by</th>
<th>movie_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reinhardt</td>
<td>Crew</td>
<td>The Black Hole</td>
</tr>
<tr>
<td>2</td>
<td>Tyrell</td>
<td>Roy Batty</td>
<td>Blade Runner</td>
</tr>
<tr>
<td>3</td>
<td>Hammond</td>
<td>Dinosaur</td>
<td>Jurassic Park</td>
</tr>
<tr>
<td>4</td>
<td>Soong</td>
<td>Lore</td>
<td>Star Trek: TNG</td>
</tr>
<tr>
<td>5</td>
<td>Morbius</td>
<td>The machine</td>
<td>Forbidden Planet</td>
</tr>
<tr>
<td>6</td>
<td>Dyson</td>
<td>SWAT</td>
<td>Terminator 2: Judgment Day</td>
</tr>
</tbody>
</table>

[Primary Key]

Primary Key: `id`

---

[J. Swanhart, Introduction to Column Stores]
## Column Stores

Each column has a file or segment on disk

<table>
<thead>
<tr>
<th>id</th>
<th>Title</th>
<th>Person</th>
<th>Genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mrs. Doubtfire</td>
<td>Robin Williams</td>
<td>Comedy</td>
</tr>
<tr>
<td>2</td>
<td>Jaws</td>
<td>Roy Scheider</td>
<td>Horror</td>
</tr>
<tr>
<td>3</td>
<td>The Fly</td>
<td>Jeff Goldblum</td>
<td>Horror</td>
</tr>
<tr>
<td>4</td>
<td>Steel Magnolias</td>
<td>Dolly Parton</td>
<td>Drama</td>
</tr>
<tr>
<td>5</td>
<td>The Birdcage</td>
<td>Nathan Lane</td>
<td>Comedy</td>
</tr>
<tr>
<td>6</td>
<td>Erin Brokovich</td>
<td>Julia Roberts</td>
<td>Drama</td>
</tr>
</tbody>
</table>

[J. Swanhart, Introduction to Column Stores]
CAP Theorem

Scalability: CAP Theorem

A

Availability

Remains accessible and operational at all times.

CA

Traditional relational databases: PostgreSQL, MySQL, etc.

AP

Voldemort, Riak, Cassandra, CouchDB, Dynamo-like systems

P

Pick Two!

C

Consistency
Commits are atomic across the entire distributed system.

CP

HBase
MongoDB
Redis
MemcacheDB
BigTable-like systems

Partition Tolerance
Only a total network failure can cause the system to respond incorrectly.

[E. Brewer]
NoSQL: Key-Value Databases

• Always use primary-key access

• Operations:
  - Get/put value for key
  - Delete key

• Examples
  - Memcached
  - Amazon DynamoDB
  - Project Voldemort
  - Couchbase
NoSQL: Document Databases

- Documents are the main entity
  - Self-describing
  - Hierarchical
  - Do not have to be the same
- Could be XML, JSON, etc.
- Key-value stores where values are "examinable"
- Can have query language and indices overlaid
- Examples: MongoDB, CouchDB, Terrastore
NoSQL: Column Stores

- Instead of having rows grouped/sharded, we group columns
- …or families of columns
- Put similar columns together
- Examples: Cassandra, HBase

<table>
<thead>
<tr>
<th>Column Family</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row</td>
</tr>
<tr>
<td>Row KeyX</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Row</td>
</tr>
<tr>
<td>Row KeyY</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
NoSQL: Graph Databases

- Focus on entities and relationships
- Edges may have properties
- Relational databases required a set traversal
- Traversals in Graph DBs are faster
- Examples:
  - Neo4j
  - Pregel
# Relational Databases vs. Cassandra

<table>
<thead>
<tr>
<th><strong>Relational Database</strong></th>
<th><strong>Cassandra</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Handles moderate incoming data velocity</td>
<td>Handles high incoming data velocity</td>
</tr>
<tr>
<td>Data arriving from one/few locations</td>
<td>Data arriving from many locations</td>
</tr>
<tr>
<td>Manages primarily structured data</td>
<td>Manages all types of data</td>
</tr>
<tr>
<td>Supports complex/nested transactions</td>
<td>Supports simple transactions</td>
</tr>
<tr>
<td>Single points of failure with failover</td>
<td>No single points of failure; constant uptime</td>
</tr>
<tr>
<td>Supports moderate data volumes</td>
<td>Supports very high data volumes</td>
</tr>
<tr>
<td>Centralized deployments</td>
<td>Decentralized deployments</td>
</tr>
<tr>
<td>Data written in mostly one location</td>
<td>Data written in many locations</td>
</tr>
<tr>
<td>Supports read scalability (with consistency sacrifices)</td>
<td>Supports read and write scalability</td>
</tr>
<tr>
<td>Deployed in vertical scale up fashion</td>
<td>Deployed in horizontal scale out fashion</td>
</tr>
</tbody>
</table>
Cassandra: Replication

Replication Factor 3

Row A

Row B
Cassandra: Consistency Levels

• Data is always replicated according to replication factors
• Consistency Levels: ANY (only writes), ONE, LOCAL_ONE, QUORUM, LOCAL_QUORUM
• Consistency levels defines how many replicas must fulfill the request
• LOCAL_* are local to the data center, others go across data centers
• quorum = (sum-of-replication-factors / 2) + 1
  - Each data center may have its own replication factor
• ANY provides lowest consistency but highest availability
• ALL provides the highest consistency and lowest availability (not recommended)
Multiple Data Center Replication

Write DC 1

DC 1

DC 2

Write

[R. Stupp]
Assignment 3

- Analysis of the IRS Form 990 Filings Data
- Sign up for AWS Educate
- Part 1: Run Spark Locally over Index
  - Process the index file (CSV), organize by return_type
  - Find top dates and months for filings
- Part 2: Run Spark Locally over Subset
  - Find GrossReceiptsAmt max/min/mean for IRS 990EZ filing (XML)
  - Create a histogram of change in net assets/balances
- Part 3: Run Spark on AWS over Full Dataset (Year)
  - (Same queries as above)
  - Check your code locally first
Test 2 and Project Proposals

• Test 2: Currently scheduled for Nov. 14. Interest in Nov. 16?

• Project Proposals:
  - Identify dataset or project and provide detailed writeup
  - Must have scalability concerns and those must be described
  - AWS Public Datasets
  - Awesome Public Datasets
  - Specific Datasets:
    • AIS Data
    • NBA Data
    • IRS 990 Data
  - Due in similar timeframe (Nov. 14 or 16)
Project Proposal: Dataset Analysis

• Description of dataset(s). Include the URL(s). If a dataset is not available online, please describe how you have access to it.

• Existing work. List papers or Web articles with existing analyses of the dataset(s) or similar datasets. (at least 3)

• Questions. List questions you would like to answer about the data. (at least 3) Be specific. You are not obligated to answer every question you list here for the project.

• Techniques. List the techniques you plan to use to answer the questions. Be specific. "Visualize the data" is not specific. "Create a choropleth map showing each county colored by incidence of disease" is specific.

• Scalability. Be specific about what scalability challenges you face with the dataset
Project Proposal: Research Project

• Description of the problem. Describe the problem at a high-level first, then add any relevant details.

• Existing work. List papers that address the problem or related problems. (at least 3)

• Scalability Challenges: What are the scalability challenges you face with this particular problem?

• Ideas. What do you plan to investigate to solve the problem?

• Evaluation. How do you plan to compare your work to the existing solutions to show improvement? What tests do you plan to run?
Spanner: Google's Globally-Distributed Database

J. C. Corbett et al.
Spanner Server Organization

![Spanner Server Organization Diagram]

[Corbett et al., 2012]
Interleaved Schema

CREATE TABLE Users {
    uid INT64 NOT NULL, email STRING
} PRIMARY KEY (uid), DIRECTORY;

CREATE TABLE Albums {
    uid INT64 NOT NULL, aid INT64 NOT NULL,
    name STRING
} PRIMARY KEY (uid, aid),
INTERLEAVE IN PARENT Users ON DELETE CASCADE;

[Corbett et al., 2012]
TrueTime

- API to try to keep computers on a globally-consistent clock
- Uses GPS and Atomic Clocks!
- Time masters per datacenter (usually with GPS)
- Each machine runs a timeslave daemon
- Armageddon masters have atomic clocks
- API:

<table>
<thead>
<tr>
<th>Method</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT.now()</td>
<td>TT.interval: [earliest, latest]</td>
</tr>
<tr>
<td>TT.after(t)</td>
<td>true if t has definitely passed</td>
</tr>
<tr>
<td>TT.before(t)</td>
<td>true if t has definitely not arrived</td>
</tr>
</tbody>
</table>

[Corbett et al., 2012]
Concurrency Control

- Use TrueTime to implement concurrency control
- Types of reads and writes:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Timestamp Discussion</th>
<th>Concurrency Control</th>
<th>Replica Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-Write Transaction</td>
<td>§ 4.1.2</td>
<td>pessimistic</td>
<td>leader</td>
</tr>
<tr>
<td>Read-Only Transaction</td>
<td>§ 4.1.4</td>
<td>lock-free</td>
<td>leader for timestamp; any for read, subject to § 4.1.3</td>
</tr>
<tr>
<td>Snapshot Read, client-provided timestamp</td>
<td>—</td>
<td>lock-free</td>
<td>any, subject to § 4.1.3</td>
</tr>
<tr>
<td>Snapshot Read, client-provided bound</td>
<td>§ 4.1.3</td>
<td>lock-free</td>
<td>any, subject to § 4.1.3</td>
</tr>
</tbody>
</table>

- Use Two-Phase Commits (2PC)

[Corbett et al., 2012]
### Two-Phase Commit Scalability

<table>
<thead>
<tr>
<th>participants</th>
<th>latency (ms)</th>
<th>mean</th>
<th>99th percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17.0 ±1.4</td>
<td>75.0 ±34.9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>24.5 ±2.5</td>
<td>87.6 ±35.9</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>31.5 ±6.2</td>
<td>104.5 ±52.2</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>30.0 ±3.7</td>
<td>95.6 ±25.4</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>35.5 ±5.6</td>
<td>100.4 ±42.7</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>42.7 ±4.1</td>
<td>93.7 ±22.9</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>71.4 ±7.6</td>
<td>131.2 ±17.6</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>150.5 ±11.0</td>
<td>320.3 ±35.1</td>
<td></td>
</tr>
</tbody>
</table>

[Corbett et al., 2012]
Distribution of TrueTime Epsilons

The data shows that these two factors in determining the distribution of TrueTime Epsilons are generally not a problem. However, there can be significant tail-latency issues that cause higher values of the base value plus communication delay to the time masters. The reduction in tail latencies began on March 30 due to networking improvements. The data shows that the contributors of transient network-link congestion were due to local-clock uncertainty, and therefore, there can be significant tail-latency issues that cause higher values of the base value of the time-master uncertainty (which is generally 0). How-ever, there can be significant tail-latency issues that cause higher values of the base value plus communication delay to the time masters.

Figure 6: Distribution of TrueTime Epsilons

[Corbett et al., 2012]
Google Cloud Spanner

- https://cloud.google.com/spanner/

- Features:
  - Global Scale: thousands of nodes across regions / data centers
  - Fully Managed: replication and maintenance are automatic
  - Transactional Consistency: global transaction consistency
  - Relational Support: Schemas, ACID Transactions, SQL Queries
  - Security
  - Highly Available
Google Cloud Spanner: NewSQL

Cloud Spanner is a powerful solution that combines the best of both worlds: the structured, predictable world of relational databases and the scalable, flexible world of NoSQL databases.

### Comparison Table

<table>
<thead>
<tr>
<th>Feature</th>
<th>Cloud Spanner</th>
<th>Traditional Relational</th>
<th>Traditional Non-Relational</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Schema</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Consistency</strong></td>
<td>Strong</td>
<td>Strong</td>
<td>Eventual</td>
</tr>
<tr>
<td><strong>Availability</strong></td>
<td>High</td>
<td>Failover</td>
<td>High</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>Horizontal</td>
<td>Vertical</td>
<td>Horizontal</td>
</tr>
<tr>
<td><strong>Replication</strong></td>
<td>Automatic</td>
<td>Configurable</td>
<td>Configurable</td>
</tr>
</tbody>
</table>

Source: [https://cloud.google.com/spanner/](https://cloud.google.com/spanner/)

Building consistent systems for transactions and inventory management in the financial services and retail industries.

Supporting high-volume systems that require low latency and high throughput in the advertising and media industries.

Rely on Strong Consistency, Scale, and Performance.
Causes of Spanner Incidents

- **52.5% User**
- **7.6% Network**
- **10.9% Other**
- **12.1% Cluster**
- **13.3% Bug**
- **3.7% Operator**

Before we get to Spanner, it is worth taking a look at the evolution of Chubby, another wide-area system that provides both consistency and availability. The original Chubby paper [Bur06] mentioned nine outages of 30 seconds or more in 700 days, and six of those were network related (as discussed in [BK14]). This corresponds to an availability worse than 5 9s (at best), to a more realistic 4 9s if we assume an average of 10 minutes per outage, and potentially even 3 9s at hours per outage.

For locking and consistent read/write operations, modern geographically distributed Chubby cells provide an average availability of 99.99958% (for 30s+ outages) due to various network, architectural and operational improvements. Starting in 2009, due to “excess” availability, Chubby’s Site Reliability Engineers (SREs) started forcing periodic outages to ensure we continue to understand dependencies and the impact of Chubby failures.

Internally, Spanner provides a similar level of reliability to Chubby; that is, better than 5 9s. The Cloud version has the same foundation, but adds some new pieces, so it may be a little lower in practice for a while.
Causes of Spanner Incidents

- User: overload or misconfiguration (specific to one user)
- Cluster: non-network problems, e.g. servers and power
- Operator: misconfiguration by people
- Bug: software error that caused some problem
- Other: most are one-offs
- Network: individual data centers/regions cut off and under-provisioned bandwidth, uni-directional traffic
More Recent Tests: Spanner vs. MySQL

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.30% INSERT INTO <code>terms</code> (<code>term</code>, <code>rank</code>, <code>set_id</code>, <code>last_modified</code>) VALUES (?,?,?),(?,?,?)</td>
</tr>
<tr>
<td>2</td>
<td>0.25% INSERT INTO <code>terms</code> (<code>term</code>, <code>rank</code>, <code>set_id</code>, <code>last_modified</code>, <code>definition</code>) VALUES (?,?,?),(?,?,?),...,?</td>
</tr>
<tr>
<td>3</td>
<td>4.22% INSERT INTO <code>terms</code> (<code>term</code>, <code>rank</code>, <code>set_id</code>, <code>last_modified</code>) VALUES (?,?,?)</td>
</tr>
<tr>
<td>4</td>
<td>1.88% INSERT INTO <code>terms</code> (<code>term</code>, <code>rank</code>, <code>set_id</code>, <code>last_modified</code>, <code>definition</code>) VALUES (?,?,?),...,?</td>
</tr>
<tr>
<td>5</td>
<td>3.28% SELECT * FROM <code>terms</code> WHERE (<code>is_deleted</code> = 0) AND (<code>set_id</code> IN (??)) AND (<code>rank</code> IN (0,1,2,3)) AND (<code>term</code> != &quot;)</td>
</tr>
<tr>
<td>6</td>
<td>14.13% SELECT <code>set_id</code>, COUNT(*) FROM <code>terms</code> WHERE (<code>is_deleted</code> = 0) AND (<code>set_id</code> = ?) GROUP BY <code>set_id</code></td>
</tr>
<tr>
<td>7</td>
<td>12.56% SELECT * FROM <code>terms</code> WHERE (<code>id</code> = ?)</td>
</tr>
<tr>
<td>8</td>
<td>0.49% SELECT * FROM <code>terms</code> WHERE (<code>id</code> IN (??) AND <code>set_id</code> IN (??))</td>
</tr>
<tr>
<td>9</td>
<td>4.11% SELECT <code>id</code>, <code>set_id</code> FROM <code>terms</code> WHERE (<code>set_id</code> = ?) LIMIT 20000</td>
</tr>
</tbody>
</table>

[P. Bakkum and D. Cepeda, 2017]
More Recent Tests: Spanner vs. MySQL

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.43%</td>
</tr>
<tr>
<td></td>
<td>SELECT <code>id</code>, <code>set_id</code> FROM <code>terms</code> WHERE (<code>set_id</code> IN (??)) LIMIT 20000</td>
</tr>
<tr>
<td>11</td>
<td>0.59%</td>
</tr>
<tr>
<td></td>
<td>SELECT * FROM <code>terms</code> WHERE (<code>id</code> IN (??))</td>
</tr>
<tr>
<td>12</td>
<td>36.76%</td>
</tr>
<tr>
<td></td>
<td>SELECT * FROM <code>terms</code> WHERE (<code>set_id</code> = ?)</td>
</tr>
<tr>
<td>13</td>
<td>0.61%</td>
</tr>
<tr>
<td></td>
<td>SELECT * FROM <code>terms</code> WHERE (<code>set_id</code> IN (??))</td>
</tr>
<tr>
<td>14</td>
<td>6.10%</td>
</tr>
<tr>
<td></td>
<td>UPDATE <code>terms</code> SET <code>definition</code>=?, <code>last_modified</code>=? WHERE <code>id</code>=? AND <code>set_id</code>=?</td>
</tr>
<tr>
<td>15</td>
<td>0.33%</td>
</tr>
<tr>
<td></td>
<td>UPDATE <code>terms</code> SET <code>is_deleted</code>=?, <code>last_modified</code>=? WHERE <code>id</code> IN (??) AND <code>set_id</code>=?</td>
</tr>
<tr>
<td>16</td>
<td>12.56%</td>
</tr>
<tr>
<td></td>
<td>UPDATE <code>terms</code> SET <code>rank</code>=?, <code>last_modified</code>=? WHERE <code>id</code>=? AND <code>set_id</code>=?</td>
</tr>
<tr>
<td>17</td>
<td>1.06%</td>
</tr>
<tr>
<td></td>
<td>UPDATE <code>terms</code> SET <code>word</code>=?, <code>last_modified</code>=? WHERE <code>id</code>=? AND <code>set_id</code>=?</td>
</tr>
<tr>
<td>18</td>
<td>0.32%</td>
</tr>
<tr>
<td></td>
<td>UPDATE <code>terms</code> SET <code>definition</code>=?, <code>word</code>=?, <code>last_modified</code>=? WHERE <code>id</code>=? AND <code>set_id</code>=?</td>
</tr>
</tbody>
</table>

[P. Bakkum and D. Cepeda, 2017]
Latency: Spanner vs. MySQL

Latency at 3,000 Queries per Second

Median Latency (ms)

Query

[P. Bakkum and D. Cepeda, 2017]
Latency: Spanner vs. MySQL

Latency at 9,000 Queries per Second

[Median Latency (ms)]

[Query x-axis]

[Spanner vs. MySQL]

[P. Bakkum and D. Cepeda, 2017]
Throughput: Spanner vs. MySQL

Median Latency as Throughput Increases

[Graph showing latency vs. throughput for MySQL and three different Spanner configurations (9 nodes, 15 nodes, and 30 nodes)]

[P. Bakkum and D. Cepeda, 2017]
Max Throughput vs. Nodes

Max Throughput (Queries per Second)

Nodes

[Note: Figure showing Max Throughput vs Nodes with data points and a linear trend line.]

[P. Bakkum and D. Cepeda, 2017]
Spanner: Latency vs. Nodes

Latency at 3000 QPS vs Nodes

Latencies @ 3000 QPS

Nodes

[P. Bakkum and D. Cepeda, 2017]
Discussion
F1: A Distributed SQL Database That Scales

J. Shute, R. Vingralek, B. Samwel, B. Handy, C. Whipkey, E. Rollins, M. Oancea, K. Littlefield, D. Menestrina, S. Ellner,
F1: OLTP and OLAP Together

• Distributed data storage: data is not stored at one central location
• Need to keep data and schemas in sync
• Hierarchical schemas keep data that is likely to be accessed at the same time together
• Optimistic Transactions: Long reads that keep track of timestamps and don't lock the database until the write happens
• Change History: Keep track of history as part of the database, also helps with caching
• DIY Object-Relational Mapping: don't automatically join or implicitly traverse relationships
• Protocol buffers as a way to store application data without translation + support for queries
Hierarchical Schema

Explicit table hierarchies. Example:
- **Customer** (root table): PK (CustomerId)
- **Campaign** (child): PK (CustomerId, CampaignId)
- **AdGroup** (child): PK (CustomerId, CampaignId, AdGroupId)

Rows and PKs

```
1
1,3
1,3,5

1,4
1,3,6
1,4,7

2
2,5
2,5,8
```

Storage Layout

```
Customer (1)
Campaign (1,3)
AdGroup (1,3,5)
AdGroup (1,3,6)
Campaign (1,4)
AdGroup (1,4,7)
Customer (2)
Campaign (2,5)
AdGroup (2,5,8)
```

[Shute et al., 2012]
Clustered Storage

- Child rows under one root row form a **cluster**
- Cluster stored on one machine (unless huge)
- Transactions within one cluster are most efficient
- Very efficient joins inside clusters (can merge with no sorting)

**Rows and PKs**

```
  1
  /|
 1,3 1,4
  |
1,3,5 1,3,6 1,4,7
```

**Storage Layout**

- Customer (1)
- Campaign (1,3)
- AdGroup (1,3,5)
- AdGroup (1,3,6)
- Campaign (1,4)
- AdGroup (1,4,7)
- Customer (2)
- Campaign (2,5)
- AdGroup (2,5,8)

[Shute et al., 2012]
F1 Notes

- Schema changes: allow two different schemas
- Transaction types: Snapshot, Pessimistic, Optimistic
- Change History and application to caching
- Disk latency or network latency?