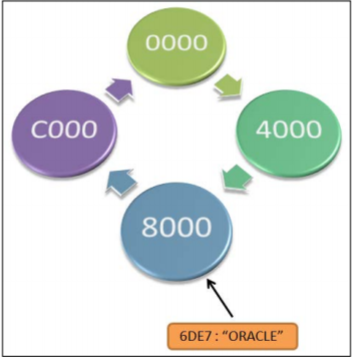
**خلاصه‌نویسی کاساندرا**

**قابلیت‌های کاساندرا**

- یک دیتابیس NoSQL [[1]](#footnote-1)و از نوع Column-basedها است.

- نظیر به نظیر[[2]](#footnote-2) و نامتمرکز[[3]](#footnote-3) است. به همین‌خاطر master-slaveی در کار نیست و دارای uptime در 100% مواقع است. دارای تعدادی node است که به هرکدام از آن‌ها یک محدوده‌ای از tokenها اختصاص یافته است. این tokenها با استفاده از partitionerها که توابع درهم‌سازی[[4]](#footnote-4) هستند، به وجود آمده‌اند. مثلا در شکل زیر، چون 6DE7 در محدوده‌ی پشتیبانی node آبی است، بنابراین اطلاعاتش در همان node قرار می‌گیرد.



- به زبان جاوا نوشته شده است؛ بنابراین از جاوای native پشتیبانی می‌کند.

- از داده‌های ساختارمند[[5]](#footnote-5) و غیرساختاریافته[[6]](#footnote-6) پشتیبانی می‌کند.

- مقیاس‌پذیری خطی[[7]](#footnote-7) دارد؛ یعنی ظرفیت[[8]](#footnote-8) و سرعت cluster متناسب با تعداد nodeهای آن است. به عبارت دیگر، هر چه تعداد nodeها بیش‌تر باشد، سرعت و ظرفیت نیز بیش‌تر خواهد بود.

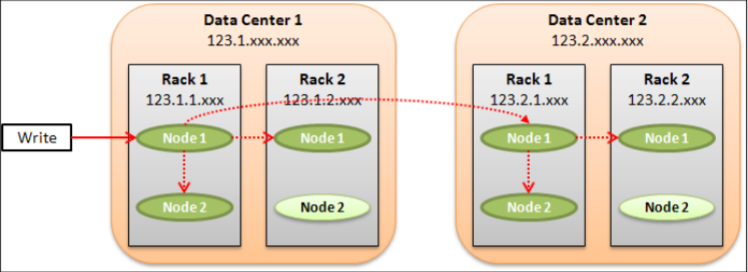
- کارایی[[9]](#footnote-9) بالایی دارد. سرعت بالای آن به دلیل ساختار آن و وجود SSTableها است. زیرا داده‌ها در آن (همان‌طور که از اسمش پیداست)، به صورت مرتب‌شده هستند و به همین دلیل پیدا کردن داده‌ها بسیار سریع اتفاق می‌افتد. سرعت نوشتن هم به دلیل log-based و memory-based بودن آن (چون هنگام نوشتن، هیچ عملیات خواندنی در آن اتفاق نمی‌افتد) بسیار بسیار سریع است. یکی دیگر از دلایل سرعت بالای نوشتن، استفاده از I/O ترتیبی[[10]](#footnote-10) و نه تصادفی است!

- توزیع داده و replicationها، روی همه‌ی nodeهای cluster به صورت خودکار صورت می‌گیرد.

- برای تشخیص خرابی این‌طور نیست که یک Boolean، با توجه به بالا/پایین بودن node با مقدار true/false برگرداند. نحوه‌ی کار آن به این صورت است که با توجه به بعضی از ویژگی‌ها یک عدد برمی‌گرداند.

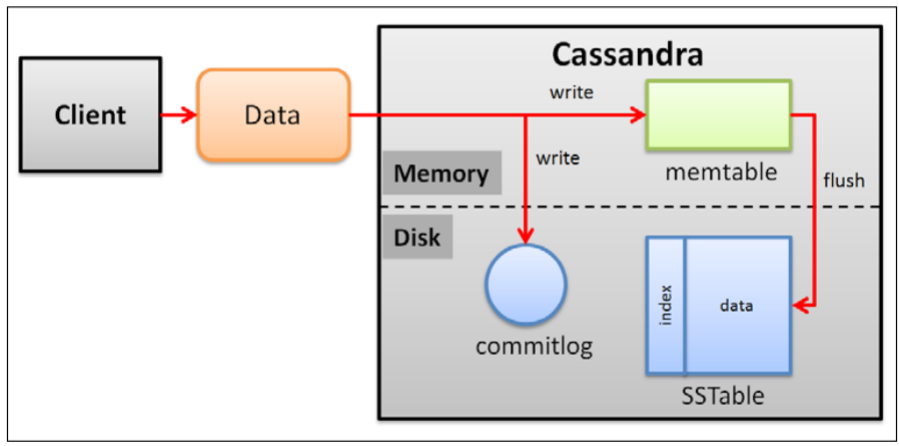
Coordinator: به nodeی می‌گویند که کاربر جواب درخواست خود را از آن دریافت می‌کند. هر nodeی روی cluster می‌تواند درخواست‌های خواندن/نوشتن را قبول کند و coordinator شود. Coordinator تشخیص می‌دهد که کدام nodeها مسئولیت نگهداری از داده‌ی درخواست را دارند و به صورت یک proxy بین nodeها و کاربر[[11]](#footnote-11) عمل می‌کند.

Snitch: Snitch روی چگونگی توزیع replicaها با در نظر گرفتن تنظیمات دیتاسنترها و rackها تاثیرگذار است. در واقع snitch تشخیص می‌دهد که کدام دیتاسنترها و rackها برای مطلع کردن کاساندرا از توپولوژی routing درخواست‌ها به صورت کارآمد استفاده می‌شوند.



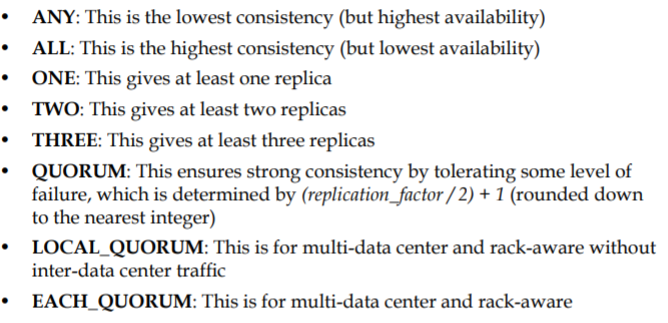
Seed node: این nodeها، اولین nodeهایی هستند که در cluster برای بهبود و تسهیل راه‌اندازی[[12]](#footnote-12) پردازه‌ها برای nodeهای جدیدی که تازه به cluster اضافه شده‌اند، از آن‌ها استفاده می‌شود.

- با ایده‌برداری و ادغام دو دیتابیس Google BigTable و Amazon Dynamo توسط شرکت Facebook ساخته شده است. کاساندرا با استفاده از مکانیزم commitlog از Google BigTable ماندگاری[[13]](#footnote-13) داده‌ها را تضمین می‌کند. هرگاه یک درخواست نوشتن داده توسط یک node دریافت می‌شود، درون commitlog هم نوشته می‌شود. داده‌ای که به‌روز می‌شود، درون یک ساختار حافظه‌ای به نام memtable نوشته می‌شود. هرگاه memtable پر شود، داده‌های داخل آن روی ساختار حافظه‌ی disk ذخیره می‌شوند.



Gossip: پروتکلی است که nodeها با یک‌دیگر صحبت می‌کنند؛ در واقع پروتکلی است برای تبادل اطلاعات nodeها درباره خودشان و همسایه‌هاشان (حداکثر تا 3 همسایه). اطلاعات مربوط به Gossip روی سیستم محلی برای اجازه دادن سریع برای restart شدن node روی آن هم ذخیره می‌شود.

Tunable Consistency: درجه‌ی سازگاری یا Consistency level عمل نوشتن، تعداد ackهایی است که از replicaها دریافت می‌شود.

****

**مکانیزم‌های Repair**

- **Read Repair:** اگر replica ناسازگار[[14]](#footnote-14) باشد، coordinator یک آپدیت ارسال می‌کند تا سازگار شود.

- **Hinted Handoff:** هدف آن کاهش مدت زمان بازیابی یک failed node هنگام پیوستن دوباره به cluster است. امکان نوشتن قطعی ر ا با قربانی کردن سازگاری عمل خواندن، می‌دهد. اگر هنگامی که یک عمل write انجام می‌شود، یکی از replicaها پایین باشد، replicaی سالم یک hint ذخیره می‌کند. در بدترین شرایط وقتی همه‌ی replicaها، پایین هستند، coordinator یک hint ذخیره می‌کند. بعد از این که دوباره nodeها بالا آمدند، با استفاده از hintهای ذخیره شده، بعد از handoff کامل، عمل خواندن سازگار است.

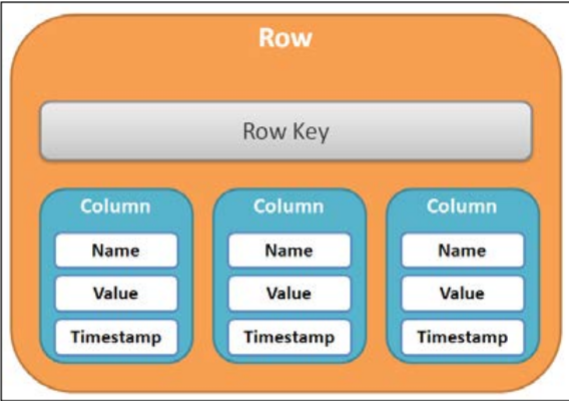
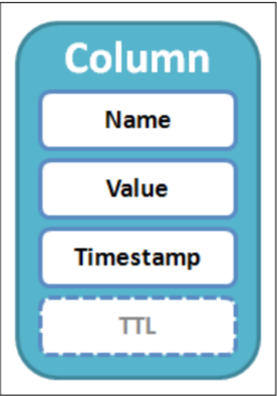
**نکته:** hint دارای اطلاعاتی در مورد مکان failed node، row key موثر و داده‌ی نوشته شده است.

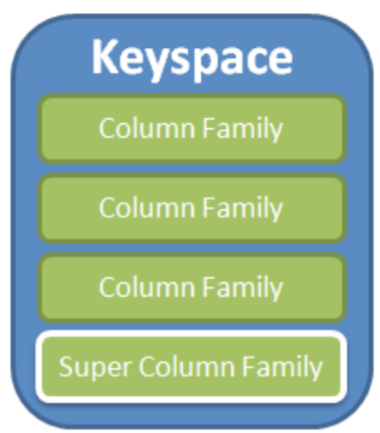
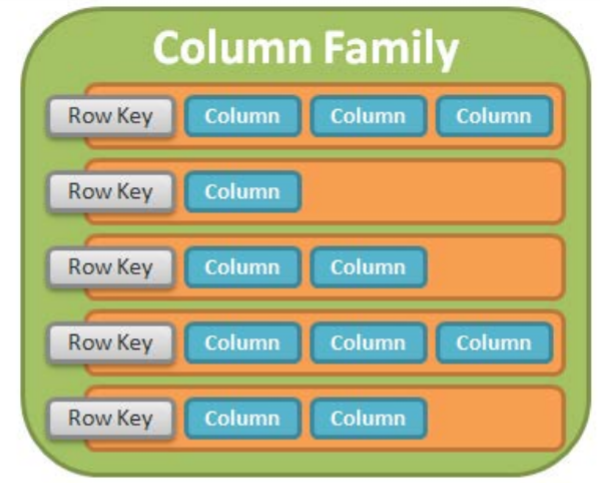
- **Anti Entropy:** این اطمینان را می‌دهد تا همه‌ی اطلاعات موجود در همه‌ی nodeها، به‌روز هستند.

دیتابیس Google BigTable: براساس تئوری CAP دارای ویژگی‌های Consistency و Partition-Tolerance است.

دیتابیس Amazon Dynamo: یک دیتابیس Key-Value based هست که ساختار نظیر به نظیر دارد. تمرکز Dynamo بیش‌تر روی High Availability آن است. مهم‌ترین ویژگی آن، ایده‌ی Eventual Consistency است. براساس تئوری CAP، دارای ویژگی‌های Consistency و Availability است.

**مدل داده‌ای کاساندرا:**





نکته: Keyspace همانند Schema در پایگاه‌داده‌های رابطه‌ای[[15]](#footnote-15) است.

نکته: Column-Family همانند Table در پایگاه‌داده‌های رابطه‌ای است.

نکته: هر ردیف می‌تواند تعداد متفاوتی ستون داشته باشد.

**Indexes**

Primary Key

- به لیستی از کلیدهای ردیف یک جدول، کلید اصلی[[16]](#footnote-16) می‌گویند.

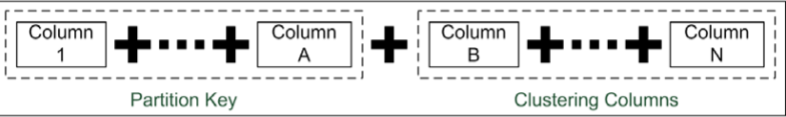
- به جزء اول کلید اصلی، کلید partition می‌گویند؛ بنابراین همه‌ی ردیف‌های مربوط به یک جدول در یک node ذخیره می‌شوند؛ زیرا partition key یکسانی دارند.

Compound Key:

- از بیش‌تر از یک ستون ساخته شده است.

- در آن ترتیب ستون‌ها مهم است.

- ساختار آن به شکل زیر است. (بخش اول آن، partition key است و بخش دوم ترتیب داده‌ها را برروی cluster مشخص می‌کند، به همین خاطر هم هست که ترتیب در آن مهم است؛ زیرا در صورت جابه‌جایی آن‌ها سناریو عوض می‌شود.



- برای تعریف compound key، به همان روش قبل از کلیدواژه primary key استفاده می‌کنیم و آن‌ها را طبق ترتیب مد نظر از چپ به راست نوشته و با ویرگول "," جدا می‌کنیم.

؟ تفاوت اصلی بین primary key ساده و compound primary key در مرتب بودن[[17]](#footnote-17) داده‌ها در compound key است.

Composite Partition Key

در مواقعی که حجم اطلاعات بالا است، از این کلید استفاده می‌شود، زیرا باعث می‌شود که داده‌ها را بین چندین گره[[18]](#footnote-18) پخش کند. (بنابراین هر وقت خواستیم داده‌ها را پخش کنیم بین گره‌های مختلف باید از آن کمک بگیریم.)

نحوه‌ی تعریف آن به استفاده از یک پرانتز اضافه درون کلیدواژه Primary Key است و ستون‌های مختلف با ویرگول از یکدیگر جدا می‌شوند.

Secondary Index

در کاساندرا برای هر جدول با این محدودیت مواجهیم که برای هر جدول فقط یک کلید اصلی داریم اما به می‌توانیم چندین secondary index روی ستون‌ها (به جز ستون کلید اصلی!) داشته باشیم.

فایده‌ی استفاده از secondary index در این است که سرعت و کارایی پیدا کردن داده‌های را در عبارات WHERE بالاتر می‌برد.

با استفاده از دستور CREATE INDEX می‌توان secondary index ساخت.

در کاساندرا می‌توان چندین[[19]](#footnote-19) secondary index داشت.

عبارت WHERE تنها در صورت که حداقل یک ستون secondary index باشد، اجرا می‌شود.

**تفاوت اصلی** کلید اصلی و secondary index در این است که primary key یک index توزیع‌شده است که از آن برای قرارگیری گره‌ای که کلید ردیف را نگه می‌دارد استفاده می‌شود؛ در حالی که secondary index، یک index محلی است که داده‌ را تنها روی گره‌ی محلی index می‌کند.

از عبارت ALLOW FILTERING زمانی استفاده می‌شود که بخواهیم به کاساندرا به صورت مستقیم اعلام کنیم که باید کوئری سنگین و پرهزینه‌ای را روی هر عبارت WHERE بدون ساختن secondary index با وجود عملکرد غیرقابل پیش‌بینی، انجام دهد.

The ALLOW FILTERING clause is also required. ALLOW FILTERING provides the capability to query the clustering columns using any condition.

**بایدها و نبایدهای secondary index:**

- روی ستون‌های با cardinality بالا (یعنی تعداد ستون‌های با مقادیر یکتا[[20]](#footnote-20) زیاد است)، secondary index تعریف نکنیم؛ زیرا سربار آن زیاد است.

- روی ستون‌هایی که اخیرا عمل آپدیت یا حذف داشته‌اند، نباید secondary index تعریف کنیم.

- روی جدول‌هایی که ستون شمارنده دارند، نباید secondary index تعریف کنیم.

- زمانی که می‌خواهیم به دنبال یک ردیف در یک partition بزرگ بگردیم.

**نکات:**

نظریه CAP:

* Consistency: یعنی داده‌ای که از هر node روی cluster می‌گیریم، به‌روزترین یا اصطلاحا most-recent باشد؛ به عبارت دیگر هنگام انجام عمل خواندن/نوشتن اطلاعات همه‌ی node مشابه باشد.
* Availability: به معنای توانایی دسترسی به cluster است حتی اگر nodeی درون cluster پایین بیاید.
* Partition-Tolerance: به معنای این است که حتی اگر ارتباط بین دو node قطع شود و هر دو بالا باشند، cluster به کار و عمل‌کرد خود ادامه دهد.

انواع Consistency:

- Strong Consistency: هر بار آپدیت رخ دهد، دیتابیس باید در تمام nodeها، آن آپدیت را انجام داده و بعد ACK برگرداند.

- Weak Consistency: به محض دریافت آپدیت و نوشتن در node اول ack را برمی‌گرداند و سپس در background روی سایر nodeها تلاش می‌کند، آپدیت‌ها را اعمال کند. ممکن است در حین این کار، این آپدیت‌ها به دلایل مختلف drop شوند و به بقیه نرسند.

- Eventual Consistency: زیرگروهی از weak-consistencyها هستند؛ با این تفاوت که اگر شبکه شلوغ نباشد و آپدیت‌ها زیاد نباشند، تلاش می‌کند که آپدیت‌ها را سریعا به تمامی nodeها برساند و روی آن‌ها اعمال کند، ولی با این حال تضمینی برای آن وجود ندارد!

1. Not only Structured Query Language [↑](#footnote-ref-1)
2. Peer-to-Peer [↑](#footnote-ref-2)
3. Decentralized [↑](#footnote-ref-3)
4. Hash Function [↑](#footnote-ref-4)
5. Structured [↑](#footnote-ref-5)
6. Unstructured [↑](#footnote-ref-6)
7. Linear Scalability [↑](#footnote-ref-7)
8. Capacity [↑](#footnote-ref-8)
9. Performance [↑](#footnote-ref-9)
10. Sequential [↑](#footnote-ref-10)
11. Client [↑](#footnote-ref-11)
12. Bootstrap [↑](#footnote-ref-12)
13. Durability [↑](#footnote-ref-13)
14. Inconsistent [↑](#footnote-ref-14)
15. Relational [↑](#footnote-ref-15)
16. Primary Key [↑](#footnote-ref-16)
17. Sorted [↑](#footnote-ref-17)
18. Node [↑](#footnote-ref-18)
19. Multiple [↑](#footnote-ref-19)
20. Unique [↑](#footnote-ref-20)