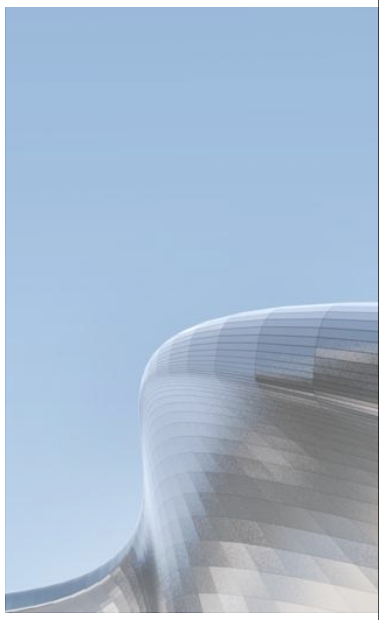
PART 06

Широко-колонкові СУБД



Ширококолонкові СУБД

Модель даних: База даних організована навколо **дворівневої** ключової структури:

- **Перший ключ:** Визначає, **на якому вузлі** фізично зберігаються дані (для шардування).
- **Другий ключ:** Визначає, **як сортуються** рядки всередині цього вузла (для швидкого пошуку).

Схема: Гнучка і розріджена (Sparse). Кожен рядок може мати **необмежену** кількість колонок, і на диску зберігаються лише ті, які містять дані. Рядки являють собою множину триплетів (назва колонки, значення, мітка часу).

Доступ до даних: Забезпечується **надзвичайно низька** затримка (Low-Latency) та **висока пропускна здатність** при доступі до даних за Першим ключем, навіть при величезних обсягах даних (**Big Data**).

Масштабування: Шардування (фрагментація) даних відбувається за **Першим ключем** для лінійної масштабованості.

Мова запитів: Використовується SQL-подібна мова (наприклад, CQL) з фокусом на **запитах по ключу**.

Варіанти використання колонкових СУБД

Типове застосування:

- **Збір подій** (Event Logging): Зберігання кліків, логів, телеметрії.
- **Оперативні профілі**: Швидкий доступ до історії активності користувача.
- **Розподілена Інвентаризація**: Відстеження запасів на великій кількості складів.
- **Геопросторові дані**: Зберігання даних про переміщення об'єктів та GPS-координат.
- **Зберігання метрик в реальному часі**: Системи моніторингу та діагностики продуктивності (APM).
- **Служби обміну повідомленнями**: Зберігання історії чатів та повідомлень з високою доступністю.
- Сценарії, де критично важливі **максимальна доступність** та **велика пропускна здатність** запису.

Rank			DBMS	Database Model	Score		
Jul 2025	Jun 2025	Jul 2024			Jul 2025	Jun 2025	Jul 2024
1.	1.	1.	Apache Cassandra	Wide column, Multi-model	108.76	+0.49	+9.63
2.	2.	2.	Apache HBase	Wide column	22.72	+0.04	-5.01
3.	3.	3.	Microsoft Azure Cosmos DB	Multi-model	21.68	-0.75	-5.44
4.	4.	↑ 5.	ScyllaDB	Wide column, Multi-model	3.98	+0.02	+0.15
5.	5.	↓ 4.	Datastax Enterprise	Wide column, Multi-model	3.66	+0.04	-1.44
6.	6.	↑ 7.	Microsoft Azure Table Storage	Wide column	2.61	+0.00	-0.80
7.	7.	↑ 8.	Google Cloud Bigtable	Multi-model	2.56	-0.03	-0.37

Apache Cassandra

Створена у 2008 році в компанії Facebook для забезпечення швидкого пошуку у власній системі вхідних повідомлень.

Основна мова: Написана переважно мовою Java.

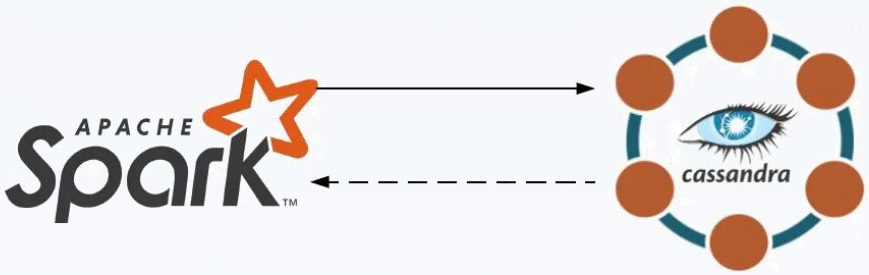
Тип СУБД: Децентралізована, розподілена Wide-Column база даних. Оптимізована для Big Data та високої доступності, застосовує модель кінцевої узгодженості (Eventual Consistency).

Мова запитів: Використовує CQL (Cassandra Query Language) — декларативну мову, подібну до SQL.

Ліцензія: Розповсюджується під вільною ліцензією Apache 2.0.

Поточна версія: Актуальна стабільна версія — 4.1.x.

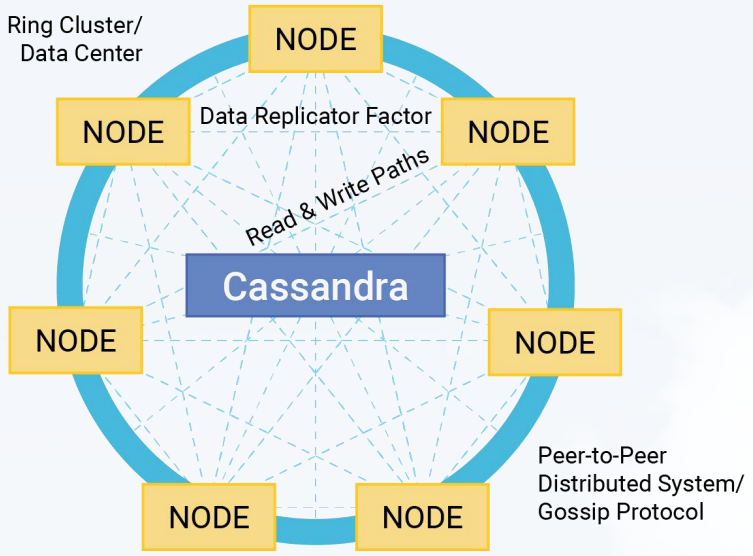
Екосистема: Часто використовується в парі з Apache Spark для виконання складної аналітики та обробки величезних масивів даних.



Apache Cassandra: Ключові характеристики

Ключові характеристики Cassandra:

- **Децентралізована архітектура** (Peer-to-Peer): Нема головного вузла (master). **Усі вузли рівноправні**. Це забезпечує стійкість до відмов - вихід з ладу одного вузла не зупиняє роботу всієї системи.
- **Лінійна масштабованість**: Продуктивність збільшується лінійно з додаванням нових вузлів у кластер.
- **Висока доступність і стійкість до відмов**: Дані реплікуються на кілька вузлів (серверів) в кластері. Ви самі настроюєте коефіцієнт реплікації.
- **Налаштована узгодженість** (Tunable Consistency): Cassandra реалізує модель **AP** (доступність та стійкість до розбиття) з теореми CAP, але дозволяє розробнику **налаштовувати рівень узгодженості** (Consistency Level) для кожної операції читання/запису, балансуючи між узгодженістю та доступністю.



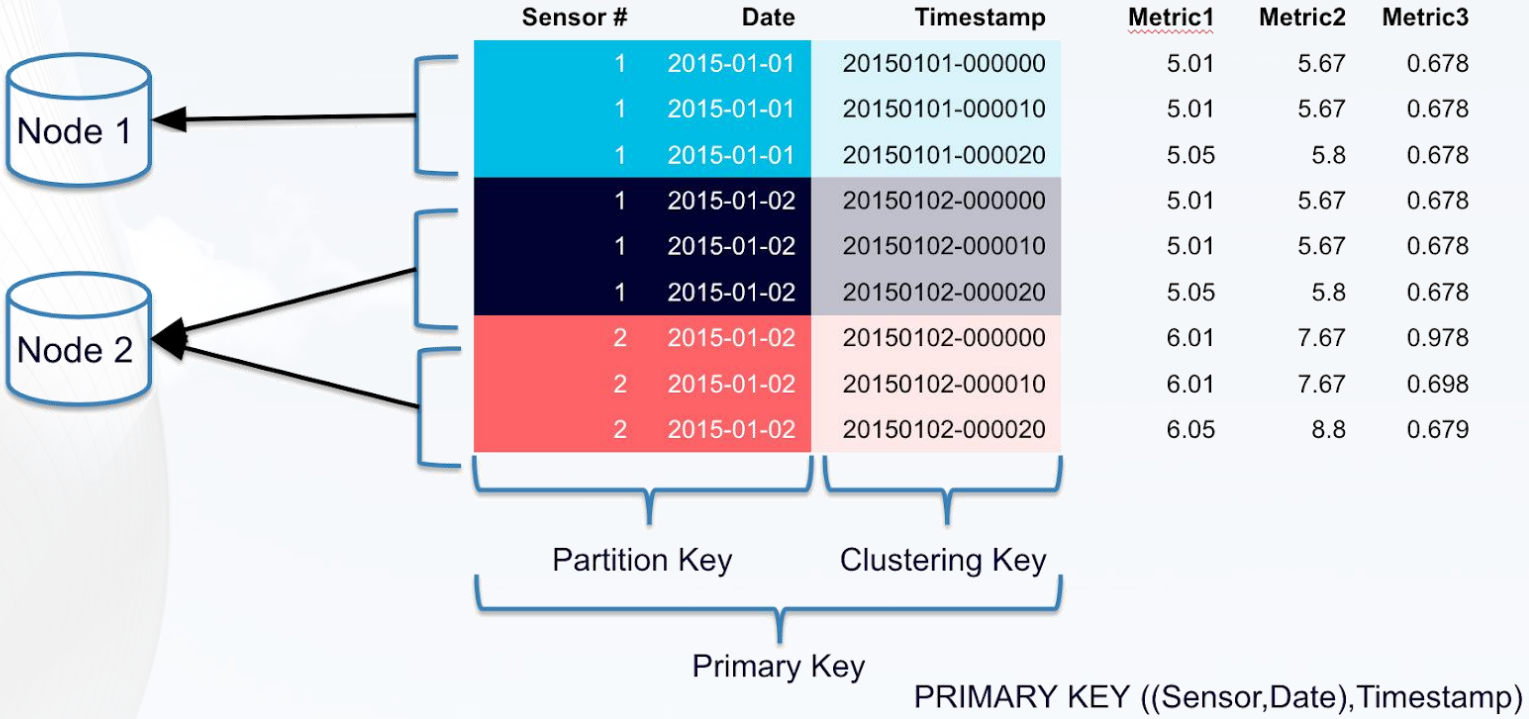
- **Мова запитів CQL** (Cassandra Query Language): Дуже схожий на SQL.
- **Модель даних**: Рядки ідентифікуються за первинним ключем, який складається з **ключа розділу** (Partition Key) і, опціонально, **кластеризуючих ключів** (Clustering Keys). Дані Cassandra проектуються **навколо шаблонів запитів** (query patterns), що є радикальним відмінністю від нормалізації в реляційних БД.

Apache Cassandra: Особливості зберігання даних

Кожна сутність (рядок) у таблиці Cassandra унікально ідентифікується своїм **Первинним ключем**.

Первинний ключ завжди складається **з двох частин** (навіть якщо друга частина порожня):

- **Ключ розділу** (Partition Key): Визначає, на якому вузлі (шарді) у кластері зберігатимуться дані цієї сутності.
- **Кластеризуючі ключі** (Clustering Keys): Визначають порядок сортування колонок усередині цього розділу на диску.



Sensor #	Date	Timestamp	Metric1	Metric2	Metric3
1	2015-01-01	20150101-000000	5.01	5.67	0.678
1	2015-01-01	20150101-000010	5.01	5.67	0.678
1	2015-01-01	20150101-000020	5.05	5.8	0.678
1	2015-01-02	20150102-000000	5.01	5.67	0.678
1	2015-01-02	20150102-000010	5.01	5.67	0.678
1	2015-01-02	20150102-000020	5.05	5.8	0.678
2	2015-01-02	20150102-000000	6.01	7.67	0.978
2	2015-01-02	20150102-000010	6.01	7.67	0.698
2	2015-01-02	20150102-000020	6.05	8.8	0.679

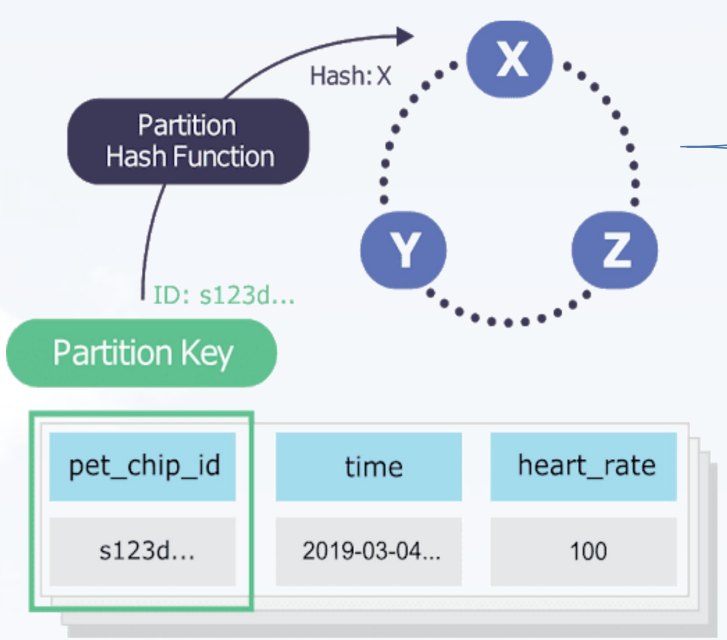
PRIMARY KEY ((Sensor,Date),Timestamp)

Apache Cassandra: Особливості зберігання даних

Коли ви вставляєте дані:

- **Partition Key** використовується для обчислення хешу (токену), що вказує на фізичне розташування даних (вузол).
- **Clustering Keys** визначають, як колонки у межах цього розділу будуть упорядковані на диску.

```
CREATE TABLE heartrate_v1 (
   pet_chip_id uuid,
   time timestamp,
   heart_rate int,
   PRIMARY KEY (pet_chip_id, time)
);
```



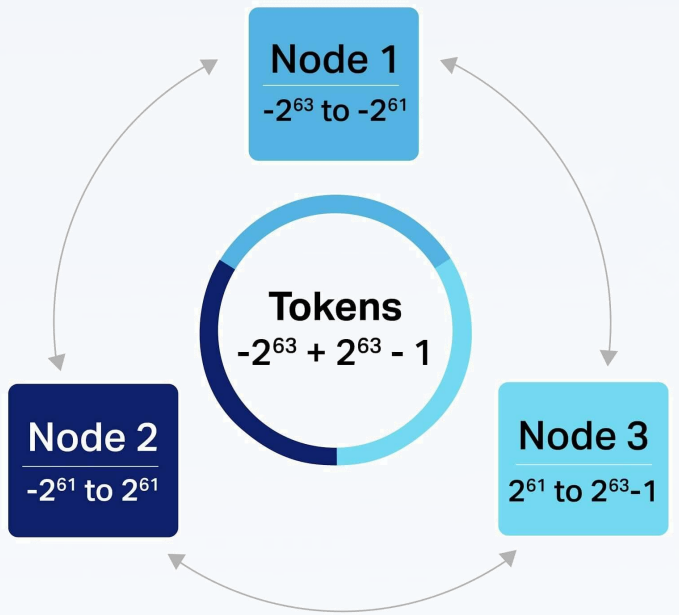
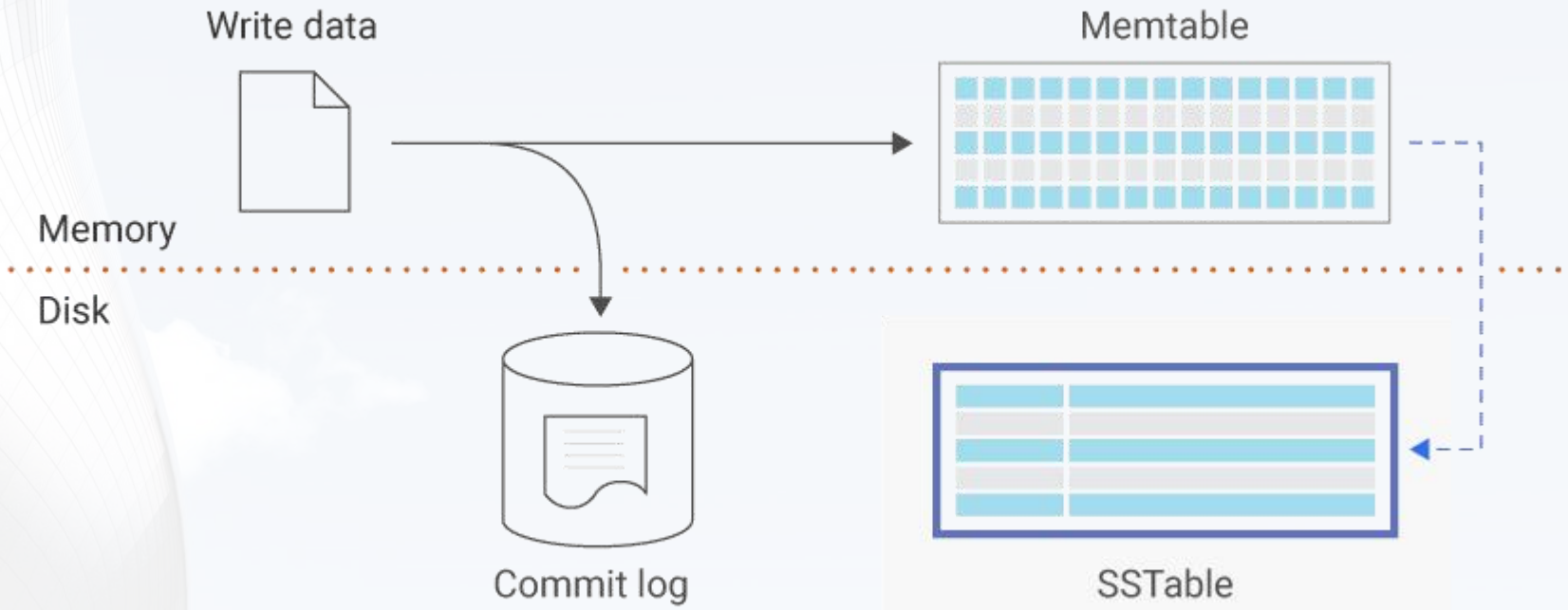


Diagram of Cassandra cluster
with 3 nodes and token-based ownership

Apache Cassandra: Запис даних

Cassandra використовує модель зберігання, яка **максимізує швидкість запису** та забезпечує стійкість даних завдяки незмінним файлам на диску.



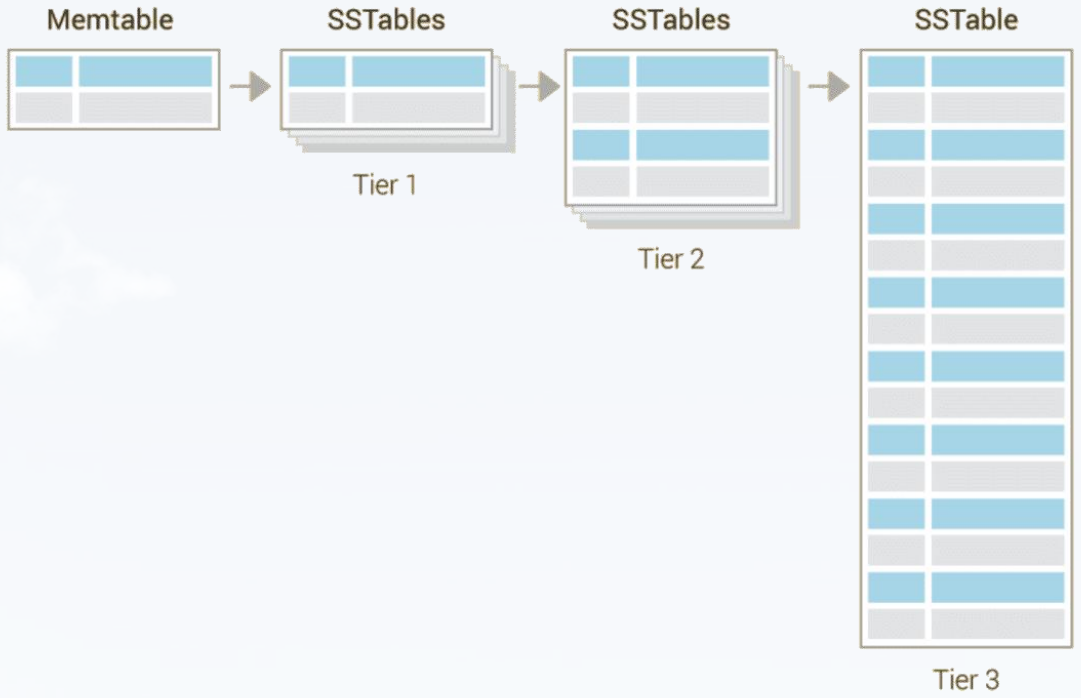
Apache Cassandra: Запис даних

Етап	Компонент	Місце зберігання	Призначення
Запис	**CommitLog**	**Диск** (HDD/SSD)	**Журнал Фіксації.** Дані негайно записуються сюди послідовно. Це забезпечує **стійкість (durability)**: якщо вузол вийде з ладу до того, як дані потраплять у пам'ять, їх можна відновити з цього логу при перезапуску.
	Memtable	**Оперативна пам'ять** (RAM)	**Активна структура даних.** Дані, записані в CommitLog, одночасно поміщаються у Memtable у відсортованому вигляді. Уся подальша обробка та читання нових даних відбувається тут.
Скидання на диск	**SSTable**	**Диск** (HDD/SSD)	**Sorted String Table.** Це кінцевий формат зберігання даних на диску. Ключові особливості: • **Незмінність** (Immutable): Файли SSTable ніколи не змінюються після створення. • **Сортування**: Дані всередині SSTable відсортовані за Partition Key (ключем розділу). • **Версіонування**: Кожне оновлення або видалення створює новий запис в новій Memtable, який потім перетворюється на новий SSTable. Видалені дані позначаються спеціальним маркером — Tombstone (Надгробок).

Apache Cassandra: Запис даних: компактування

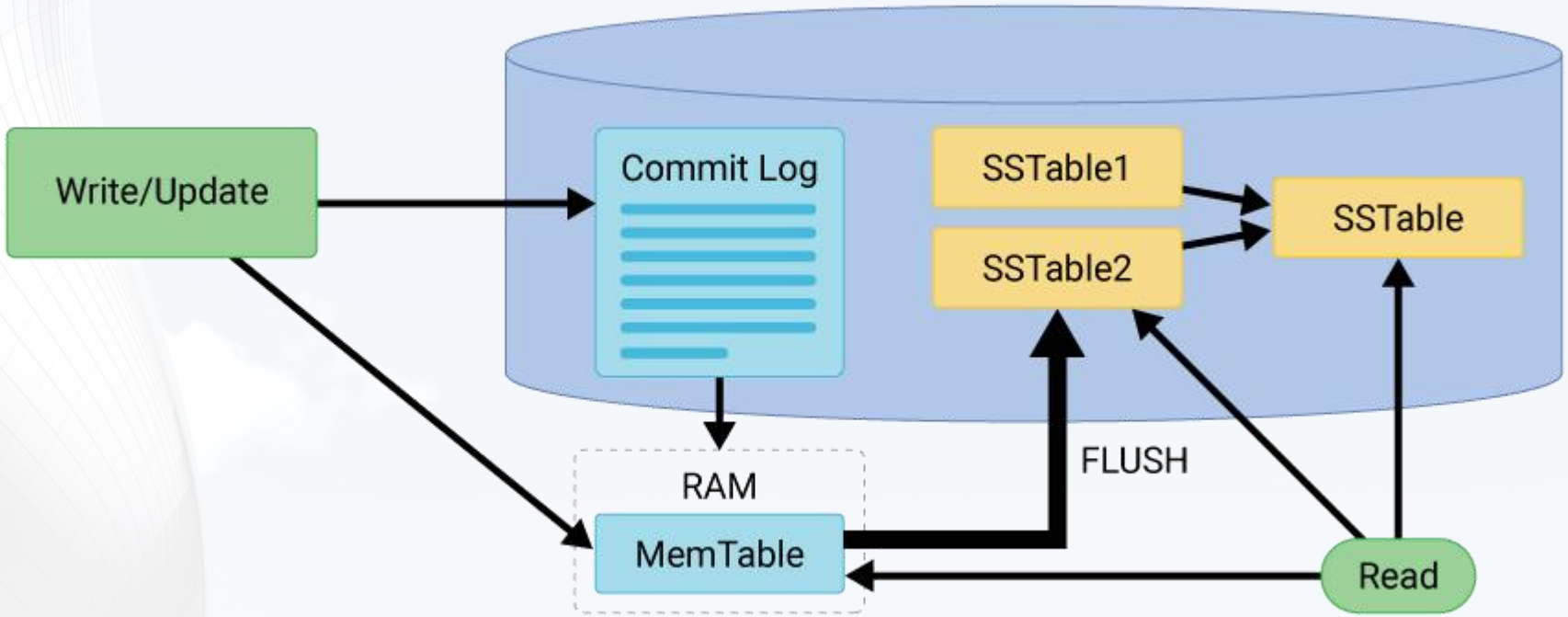
Через те, що оновлення та видалення створюють нові незмінні SSTable, на диску накопичуються їхні старі версії, "сміття" та "надгробки" (Tombstones). Проблему вирішує компактування.

- **Компактування** (Compaction): Це фоновий процес, який періодично об'єднує декілька малих SSTable у один більший. Його мета: зменшити кількість файлів, які потрібно перевіряти під час читання, та фізично видалити старі версії даних і Tombstones.



Apache Cassandra: Читання даних

Читання даних: Щоб відповісти на запит, Cassandra повинна перевірити: Memtable + CommitLog + усі відповідні SSTable на диску, щоб знайти найновішу версію даних.



Apache Cassandra: CQL: Keyscaces

Аналогом бази даних в Cassandra є Keyspace (простір ключів).

1. Управління Keyspace (простором ключів)

Keyspace — це контейнер для ваших таблиць, який визначає **фактор реплікації** (RF) та **стратегію реплікації** (як дані розподіляються та дублюються по вузлах).

Дія	Команда CQL	Призначення
Створення	CREATE KEYSPACE	Створює новий простір ключів із заданою стратегією та RF.
Використання	USE	Перемикає поточний контекст на вказаний Keyspace.
Видалення	DROP KEYSPACE	Видаляє Keyspace та всі дані в ньому.

Приклад створення простору ключів:

```
CREATE KEYSPACE my_lab_keyspace
WITH replication = {
    'class': 'SimpleStrategy',        -- SimpleStrategy: використовується для одного дата-центру.
    'replication_factor': 2           -- Кожна копія даних зберігається на двох вузлах.

};
```

Apache Cassandra: CQL: Таблиці

2. Створення Таблиці

Створення таблиці у Cassandra є критичним, оскільки саме тут визначається, як дані будуть **розділені** та **відсортовані**. Це досягається за допомогою PRIMARY KEY.

Ключ	Призначення
Partition Key	Обов'язковий. Визначає, на якому вузлі фізично буде зберігатися рядок.
Clustering Key	Необов'язковий. Визначає порядок сортування рядків всередині однієї партиції.

Приклад створення таблиці:

```
CREATE TABLE my_lab_keyspace.sensor_data (
    device_id text,
    reading_time timestamp,
    temperature float,
    humidity float,
    PRIMARY KEY (device_id, reading_time)
);
```

- Partition Key: device_id (дані одного пристрою будуть зберігатися разом).
- Clustering Key: reading_time (дані пристрою будуть відсортовані за часом, що забезпечує швидкий пошук часових діапазонів).

Apache Cassandra: CQL: DML

Дія	Команда CQL	Призначення
Запис	INSERT	Вставляє нові дані. Якщо рядок з таким PRIMARY KEY вже існує, він буде **оновлений** (фактично, додається нова комірка з вищою міткою часу).
Читання	SELECT	Отримує дані. **Обов'язково** має включати умову на **Partition Key** в умові WHERE.
Оновлення	UPDATE	Використовується для зміни існуючих колонок. У Cassandra UPDATE функціонально **ідентичний** INSERT і є операцією дозапису.

Приклади

1. **Запис** (з узгодженістю):

```
CONSISTENCY QUORUM;        -- Встановлюємо високу узгодженість
INSERT INTO my_lab_keyspace.sensor_data
    (device_id, reading_time, temperature)
VALUES
    ('sensor_a', '2025-11-23 10:00:00+0200', 25.5);
```

2. **Читання** (з узгодженістю):

```
CONSISTENCY QUORUM;
SELECT * FROM my_lab_keyspace.sensor_data
WHERE device_id = 'sensor_a'
  AND reading_time > '2025-11-23 09:00:00+0200';
```

Apache Cassandra: CQL vs SQL

Ключові відмінності між CQL і SQL обумовлені розподіленою архітектурою Apache Cassandra.

Аспект	Обмеження/Розширення	Демонстрація
WHERE	У запиті SELECT у частині WHERE обов'язково має бути Ключ розділу (Partition Key)	Якщо зробити запит SELECT * FROM table WHERE some_non_key_column = 'value', Cassandra видасть помилку або вимагатиме ALLOW FILTERING (що є антипаттерном). Це показує, що Cassandra **не призначена для повного сканування**.
JOIN, GROUP BY	**Немає**	Складні аналітичні запити потрібно вирішувати **де-нормалізацією** даних або на стороні програми.
Індекси	**Обмежені** (вторинні індекси). Їх використання не рекомендується для колонок із високою унікальністю, оскільки це знижує продуктивність кластера.	Спроба створити **вторинний індекс** на унікальному полі (email) та запит по ньому покаже **повільне виконання**, ілюструючи, що Cassandra покладається на Первинний ключ, а не на індекси.
TTL	Ви можете встановити Time-To-Live для всього рядка або окремої колонки.	Вставка даних з USING TTL 3600 чудово демонструє її придатність для подій, логів чи тимчасових кешів.

Apache Cassandra: Query-Driven Modeling

Моделювання, орієнтоване на запити (Query-Driven Modeling)

У РСУБД (PostgreSQL, MySQL, ...):

- Ви **нормалізуєте** дані (уникаєте дублювання, створюєте зв'язки).
- Ви пишете **будь-які запити**, використовуючи **JOIN** для об'єднання таблиць.
- Продуктивність залежить від того, наскільки ефективно РСУБД виконає JOIN.

У Cassandra:

- Ви повинні **заздалегідь знати**, які запити виконуватимете (наприклад, "знайти користувача по email", "знайти всі кліки користувача за датою").
- Для **кожного шаблону запиту** ви створюєте **окрему таблицю** (Column Family).
- **Первинний ключ** кожної таблиці налаштовується так, щоб він ідеально відповідав умовам **WHERE** вашого запиту.

Де-нормалізація (Дублювання даних)

Оскільки Cassandra не підтримує JOIN, якщо потрібно знайти дані за двома різними критеріями, і ці критерії не можуть бути ефективно оброблені одним Первинним ключем, вам необхідно зберегти (дублювати) дані в **двох різних таблицях**.

Таким чином, ми обмінюємо дисковий просторім на швидкість читання та масштабованість.

Apache Cassandra: Роль мітки часу

Мітка часу (Timestamp) є обов'язковим та фундаментальним компонентом для **кожної комірки** даних в Apache Cassandra.

Це не просто дата створення запису; це ключовий архітектурний елемент, що забезпечує **відмовостійкість** та **версіонування** у розподіленому середовищі.

Cassandra сама додає мітку часу при виконанні CRUD-операцій.

Мітка часу виконує дві критично важливі функції:

1. Вирішення конфліктів (Last Write Wins)

Оскільки Cassandra – це децентралізована система з високою доступністю (AP-модель), **одночасний запис** однієї й тієї ж комірки може статися на **кількох вузлах-репліках**.

- Коли вузли синхронізуються, вони не порівнюють значення, а порівнюють мітки часу. Перемагає той запис, у якого мітка часу новіша. Цей механізм називається LWW (Last Write Wins - Перемагає останній запис).
- Мітка часу гарантує, що кластер прийде до узгодженого стану, вирішуючи конфлікти без потреби повного блокування.

2. Версіонування даних

Технічно Cassandra може зберігати кілька версій однієї й тієї ж комірки в SSTable, що відрізняються лише влучним часом, хоча за умовчанням вона завжди повертає останню версію.

Apache Cassandra: Налаштовувана узгодженість

На відміну від багатьох РСУБД, Cassandra дозволяє вам **самим вирішувати**, наскільки "сильною" має бути **узгодженість** під час читання та запису (тобто, скільки реплік мають підтвердити операцію):

- **Запис** (Write Consistency): Скільки реплік має успішно записати дані, перш ніж операція вважається успішною.
- **Читання** (Read Consistency): Скільки реплік має відповісти з **однаковими даними**, перш ніж вони будуть повернуті клієнту.
- Рівні варіюються від **ONE** (швидко, висока доступність, низька узгодженість) до **QUORUM** (вища узгодженість) і **ALL** (найвища узгодженість, низька доступність).
- Ви можете **для різних запитів** в одному додатку вибирати **різні рівні узгодженості,** наприклад,

```
CONSISTENCY QUORUM;
INSERT INTO my_keyspace.readings (id, timestamp, value) VALUES (1, now(), 10.5);
```

Налаштовувана узгодженість дозволяє балансувати між **високою доступністю** (AP) та **узгодженістю** даних (C) у рамках теореми CAP:

- Наприклад, ми можемо дозволити системі продовжувати працювати та приймати дані навіть при збоях у мережі або відмові вузлів (пріоритет доступності).
- Якщо ви вибрали **низький рівень узгодженості**, система **завжди буде доступна**, але іноді може повертати застарілі дані.
- Встановлення сильної узгодженості гарантуватиме, що ви завжди читаєте свіжі дані.

Apache Cassandra: Ключові рівні узгодженості

Рівень узгодженості (CL - Consistency Level) визначає, скільки реплік (вузлів) повинні підтвердити успішність операції для її завершення.

Consistency Level	Визначення	Опис
ONE	Операція вважається успішною, як тільки її підтвердить один вузол.	Найшвидший, найменш надійний. Може призвести до читання застарілих даних.
QUORUM	Операцію має підтвердити більшість вузлів, що зберігають дані. Під більшістю розуміється RF/2 + 1	Найбільш збалансований. Забезпечує високий ступінь узгодженості.
ALL	Операцію повинні підтвердити всі вузли-репліки.	Найвища узгодженість, але найбільша затримка. Вузол-репліка може стати єдиною точкою відмови.

Використовуючи правильну комбінацію рівнів узгодженості для читання та запису, можна досягти **сильної узгодженості**.

Це досягається, коли сума рівнів узгодженості для читання та запису більша за Фактор реплікації (RF):

$$CL_{\text{читання}} + CL_{\text{запису}} > RF$$

Зазвичай для балансу між продуктивністю та надійністю в одному дата-центрі **рекомендується** використовувати:

$$CL_{\text{запису}} = \text{QUORUM} \text{ та } CL_{\text{читання}} = \text{QUORUM}.$$

Зазвичай рівень узгодженості **за замовчуванням** встановлюється на **ONE**.

Apache Cassandra: Індексування

В Apache Cassandra концепція індексування **дуже відрізняється** від реляційних СУБД (РСУБД). Вона зводиться до філософії: ваш **Первинний ключ** це ваш найкращий і **єдиний по-справжньому ефективний індекс**.

🔑 Вбудоване Індексування - Первинний ключ

1. Індекс по Ключу розділу (Partition Key Index)

- Це перший і **найважливіший рівень індексації**.
- **Як працює**: Cassandra використовує індексний файл (зберігається разом з SSTable), щоб зіставити Токен (обчислюється з Ключу розділу) з фізичним зсувом (початком) розділу на диску.
- **Результат**: Коли ви запитуєте дані по Partition Key, система миттєво визначає потрібний вузол, а потім миттєво знаходить точне розташування даних на цьому вузлі.

2. Індекс по Кластеризуючим ключам (Clustering Key Index)

- Цей індекс **працює тільки всередині знайденого Розділу**.
- **Як працює**: Оскільки дані всередині розділу вже фізично відсортовані за Clustering Keys, Cassandra може використовувати швидкий бінарний пошук (або його аналоги), щоб знайти конкретний рядок або діапазон рядків усередині розділу.
- **Результат**: Дозволяє виконувати швидкі діапазонні запити (наприклад, "всі події за останню годину") без сканування всього розділу.

Apache Cassandra: Індексування

Вторинні Індекси - Secondary Indexes

Cassandra підтримує створення вторинних індексів, але їхня концепція та обмеження роблять їх придатними лише для дуже вузького кола завдань.

Основні обмеження:

На відміну від РСУБД, вторинні індекси в Cassandra **не рекомендуються для великих кластерів** і мають два критичні недоліки:

- **Локальність зберігання індексних структур**: Вторинний індекс (SI) є локальним і містить ті записи, які відносяться до набору даних, фізично розташованих на цьому конкретному вузлі. При запиті за вторинним індексом (наприклад, city = 'London'), запит має бути розісланий **КОЖНОМУ ВУЗЛУ** в кластері, щоб зібрати всі рядки, що відповідають умові.
- **Висока кардинальність**: SI неефективні для індексування колонок з великою кількістю унікальних значень (наприклад, email, username), тому що це призводить до величезних, але неефективних індексів, які потребують сканування.

Вторинні індекси в Cassandra має сенс використовувати тільки для колонок з низькою кардинальністю (невелика кількість унікальних значень), наприклад булеві прапори (is_active = true) або коди статусу (status = 4).

В цьому випадку накладні витрати на пошук по розподіленим даним також присутні, але вони частково компенсуються обсягом даних, які будуть повернуті такими запитами.

Apache Cassandra: Індексування

Справжня стратегія індексування

У Cassandra:

Індексування = Моделювання Даних

Якщо вам потрібно виконати запит по атрибуту, який не входить до первинного ключа (наприклад, знайти користувача по email), правильний шлях - це **Де-нормалізація**:

1. Створіть нову таблицю (users_by_email).
2. Встановіть Partition Key = email.
3. При записі даних дублюйте інформацію в обидві таблиці.

Це гарантує, що кожен запит залишається миттєвим (запит по ключу), оминаючи обмеження вторинних індексів.