# Persistence Stack

Alexis Abbott Notes

# Table of Contents

# Databases

## SQL

```
--Comment
```

SQL is a DML (Data Manipulation Language).

- SELECT
- DROP
- UPDATE
- DELETE

### Normal Forms

1st Normal Form - Rule of 1: Atomic. Each thing describes one thing. Can't repeat a composite primary key (foreign keys).

2nd Normal Form - when you have a partial dependency, break it out into its own table.

3rd Normal Form - Transitive Dependencies. When a column depends on a primary key, but it does it through another column. Take anything that has its own dependency and make it its own entity.

### Data Integrity

Entity Integrity. Primary key is obviously the first line of defense.

### Referential Integrity

Foreign Keys (Referential Constraint) Makes sure you can't assign a foreign key that doesn't exist.

- ON UPDATE
- ON DELETE
- ON INSERT

ON UPDATE will cascade through the tables.

ON DELETE will take no action if a child is referencing it, when set to RESTRICT. Cascade will delete children, but be warned! SET NULL, anything deleted that is being referenced will be set to null.

## Domain Integrity

What is allowed vs not allowed:

- Data Types
- Constraints: NOT NULL, UNIQUE, CHECK/VALIDATION TABLE

## OLTP

Online Transaction Processing is a database design that focuses on transactional oriented tasks. Involves inserting, updating, and deleting small amounts of data.

OLTP has the following characteristics:

- Transactions that involve small amounts of data.
- Indexed access to data.
- A large number of users.
- Frequent queries and updates.
- Fast response times.

OLTP systems need to have extremely high availability, because it deals with mission critical data, and has a large number of users. In order to maintain data integrity, they must be ACID compliant.

With database transactions, they're a sequence of operations performed as a logical unit of work. A transaction can only be successful if the whole sequence passes. If any part of the transaction fails, the whole transaction fails.

## OLAP

Online Analytical Processing allows users to analyze information from multiple database systems at the same time. Enables analysts to extract data from different points of view. Analysts frequently need to group, aggregate and join data which is resource intensive in relational databases.

## OLAP Cube

Containing multidimensional data, it moves away from the traditional table design. Extracted data is cleaned and transformed and loaded onto an OLAP server. It's then precalculated for further analysis.

Four types of analytical operations:

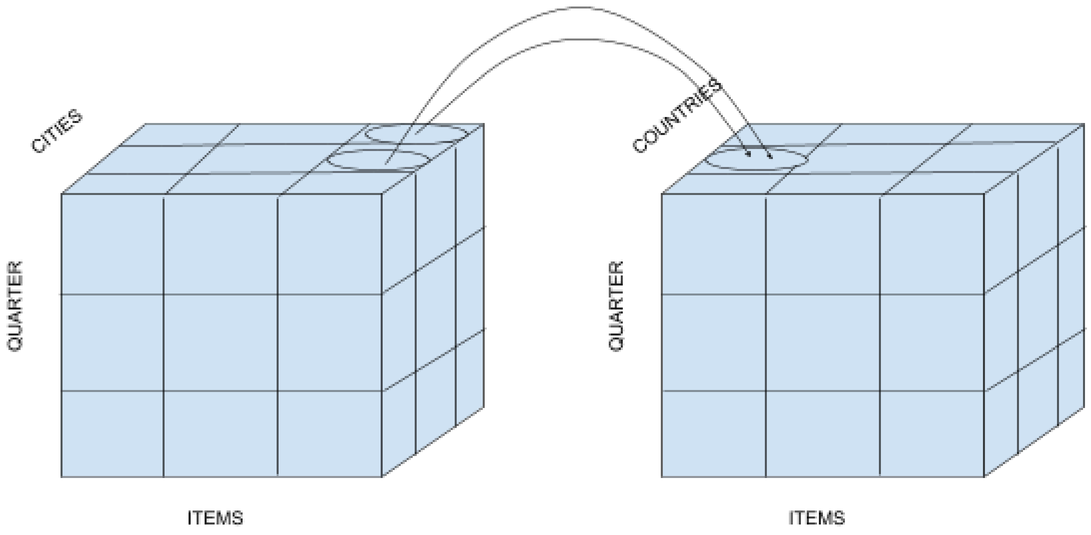
1. Roll-up
2. Drill-down
3. Slice and dice

4. Pivot (rotate)

## Roll-Up

Consolidation or aggregation.

1. Reduce dimensions.
2. Climb up concept level. (Hierarchy)

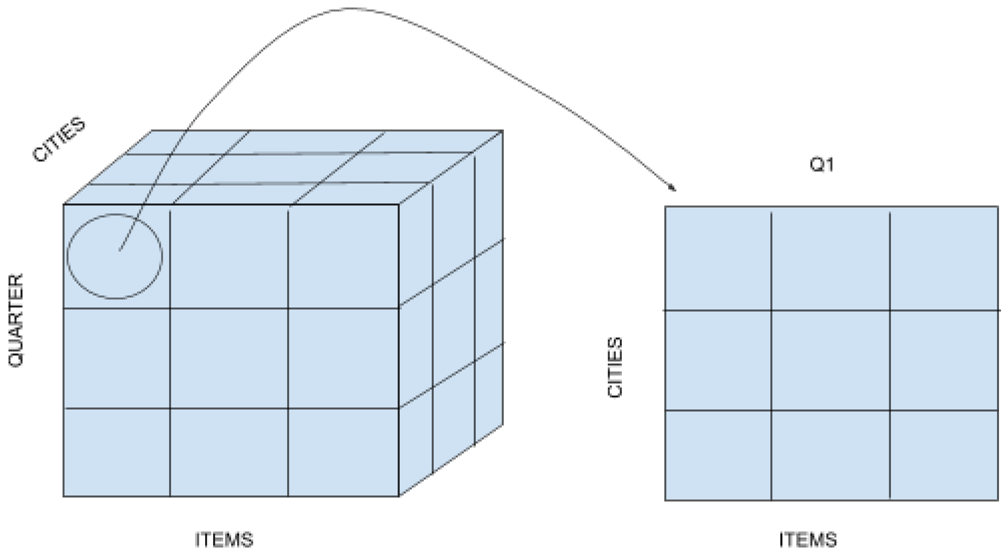


## Drill-Down

Data fragmentation into smaller parts. Opposite of roll-up.

## Slice

One dimension is selected and a sub-cube is created.

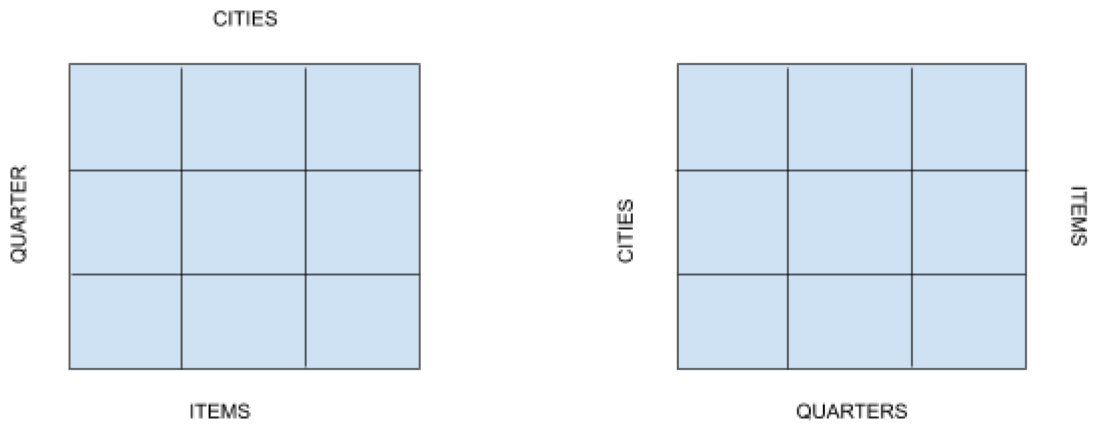


### Dice

A slice with 2 or more dimensions that create a sub-cube.

### Pivot

Rotate data access to get a different presentation.



## Types of OLAPs

| | |
|---|---|
| Relational OLAP (ROLAP): | Extended RDBMS with multidimensional data mapping to perform the standard relational operation. |
| Multidimensional OLAP (MOLAP): | MOLAP implements operation in multidimensional data. |
| Hybrid OLAP (HOLAP): | Aggregated totals are stored in a multidimensional database while the detailed data is stored in a relational database. Offers efficiency of ROLAP while having MOLAP performance. |
| Desktop OLAP (DOLAP): | User downloads partial data to a local database, but it’s very limited. |
| Web OLAP (WOLAP): | Accessible via the web. Three-tiered architecture - client, middleware, database server. |
| Mobile OLAP: | Same as desktop, but mobile. |
| Spatial OLAP: | Facilitates management of both spatial and non-spatial data in a Geographic Information System (GIS). |

## ACID

Atomicity, Consistency, Isolation, Durability is a standard set of properties that guarantee database transactions are processed reliably. It's especially concerned with recovery after failure.

### Atomicity

Means that it either all succeeds or fails. "All or nothing".

### Consistency

All data will be valid according to all defined rules, including constraints, cascades, and triggers that have been applied on the database.

### Isolation

No transactions will be impacted by any other transaction.

### Durability

Once a transaction is complete, it will remain in the system, even if there's a crash. If the system tells the user the transaction succeeded, it must, in fact, be successful.

A database is consistent if and only if the results are of successful transactions.

## Cassandra

Cassandra is a database that works across many nodes and is constantly checking through peers to update each other. It's column centric instead of row centric and each column is saved in a file on a disk. Each column works on a key/value pairing.

| ID | Name | Age |
|---|---|---|
| 1 | {Name:Jack} | {Age:22} |
| 2 | {Name:Heather} | {Age:40} |
| 3 | {Name:Sam} | {Age:17} |

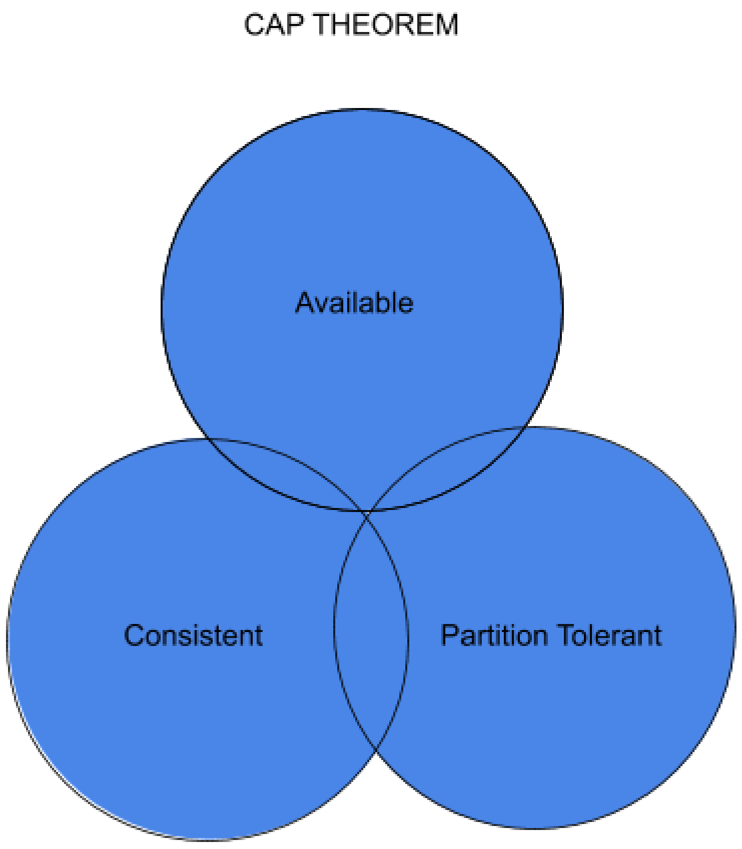
If you have something repeated often, like gender, you can duplicate rows.

| | |
|---|---|
| {Gender:M}*3 | {Name:Mark} |
| | {Name:Tony} |
| | {Name:Tim} |
| {Gender:F} | {Name:Sarah} |

Cassandra allows for truly empty fields, not needing null or void, in order to save on space.

### Fault Tolerant

Data is replicated to multiple nodes(computers), so if one goes down, there's still access to data. It's also decentralized.



Cap Theorem says you can have 2 of these, but not all 3.

**Consistent:** Every node returns the most recent data.
**Available:** Get or give data, we should always make those changes.
**Partition Tolerant:** System will continue to function even in a network failure.

Joins on a single machine are fine, but not in a distributed database. It's impossible to perform joins in Cassandra.

Instead of a model first approach, we design Cassandra for queries. This leads to denormalization of tables/data.

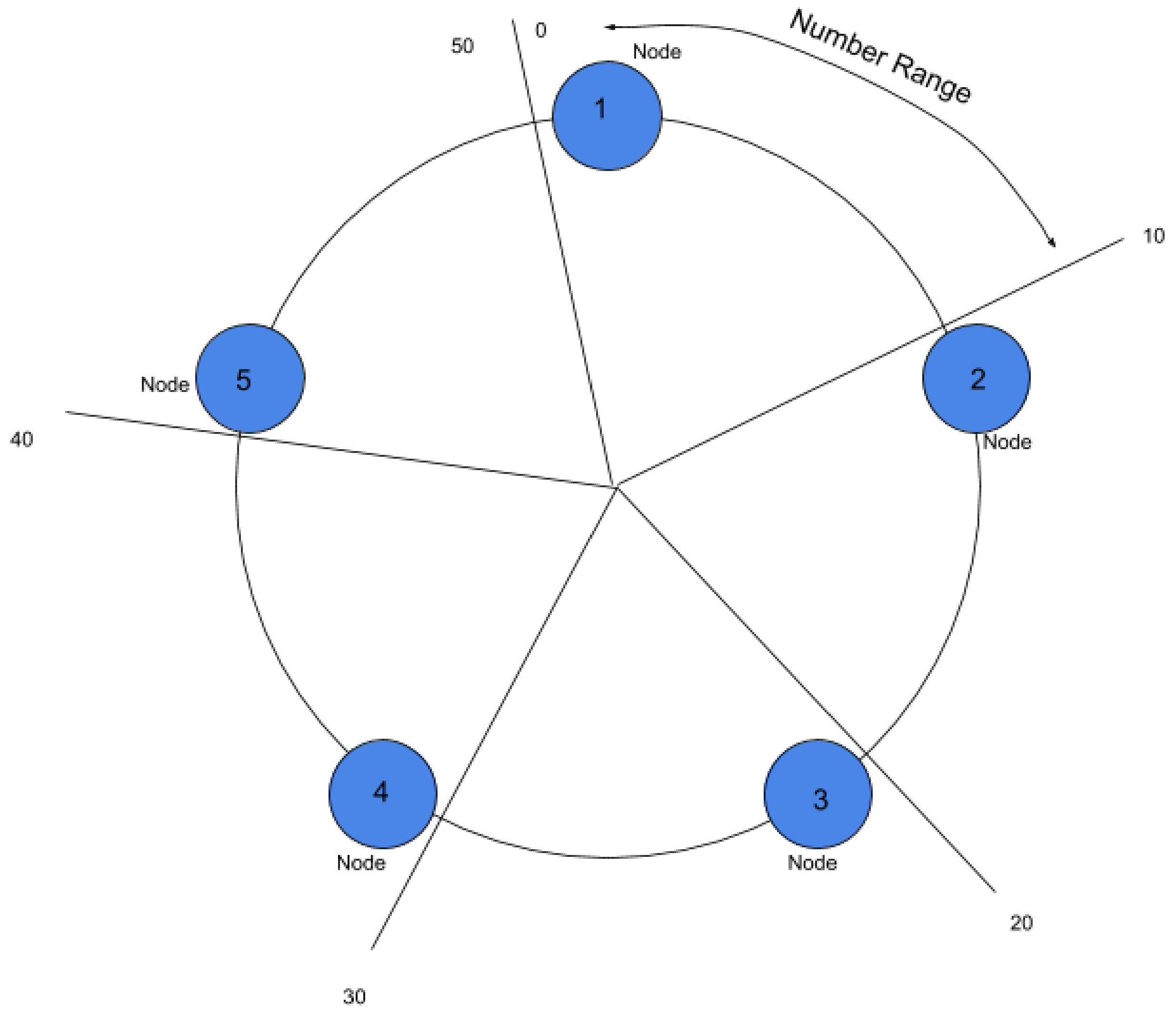
Normalization is the avoidance of repeating information.

## Partitioning

Group like data on a single node. Example: we want to query all patients at a particular clinic, we'll have less to search through because they'll all be grouped on a single computer in a separate database.

The difference between a partition key and a primary key is use and placement. Partition keys do not need to be unique, it simply decides which node the data will be stored. In Cassandra, we should only access data using the partition key. If we want to access info by the primary key, a new table, designed for that should be created.

The partition key gets passed to a hash function. It turns every partition key (string) into a unique idea. It’s how Cassandra decides how to group data. They’re known as Tokens and they’re 64 bit integers. Tokens will be assigned to nodes via a number range in a Ring.



## Virtual Nodes

Cassandra will break up a node range further into smaller ranges. This allows for more flexibility in being able to add nodes to the Ring. This also allows for adding more data to more powerful nodes (better computers). This is configurable inside of Cassandra.

Cassandra has a replication strategy and built in racks.

## Replication Factor

Configurable in Cassandra, how ever many times you wish your data to be replicated, whatever number you pick, will be how many nodes it's put on. You cannot set the Replication Factor higher than the number of nodes on the cluster.

## CQL

Cassandra Query Language. CQL has no joins.

## CQLSH

Cassandra Shell. To start the shell, type:

```
cqlsh
```

## Network Topology Strategy

**SimpleStrategy** - when you only have one datacenter.
**NetworkTopologyStrategy** - for when you have more than one datacenter. This is the recommended deployment because it's easier to expand later if needed.

## KeySpace

Setting up a new keyspace:

```
CREATE KEYSPACE keyspace_name WITH replication = {'class':'SimpleStrategy',
'replication_factor':'3'} AND durable_writes='true';
```

You can check on your keyspaces with:

```
DESCRIBE KEYSPACES;
```

You can delete with:

```
DROP KEYSPACE keyspace_name;
```

To use a keyspace:

```
USE keyspace_name;
```

The terminal prompt will change, not unlike a directory change, so we can keep track of what keyspace we're in. A keyspace can contain many tables.

```
CREATE TABLE table_name (id int PRIMARY KEY, col1 type, col2 type);
```

You can look at the tables with:

```
DESCRIBE TABLES;
```

Get information on a single table with:

```
DESCRIBE TABLE table_name;
```

Guess what!? Magic, you can delete tables as well.

```
DROP TABLE table_name;
```

### Composite Partition Keys

Cassandra also has a composite partition key which uses multiple columns to identify where data will reside.

In the schema we use the Primary Key Function.

```
PRIMARY KEY(col1, col2)
```

We can combine even more with:

```
PRIMARY KEY((col1, col2), col3));
```

### Clustering Column

A column which helps decide what node the data will be stored.

### Consistency

You can set consistency levels on the ring. Whatever your Replication Factor is, you can set a number equal to or less than that to get a response about updates to understand how consistent your data is. Once the consistency level has been met, that many nodes have returned a response, the data will be compared and the newest data will be served in the request.

### Quorum (Operation)

A majority of the Replicas respond, the read will be the one with the most recent timestamp. The formula is:

Replication Factor + 1

2

To check what our level is, we use:

```
CONSISTENCY;
```

We can change it to a number (text number: one, two, three…) or Quorum.

```
CONSISTENCY QUORUM;
```

To add data to a table, we use a standard INSERT:

```
INSERT INTO table_name (col1, col2) VALUES ('val1', 'val2');
```

To read from a table, we use a standard SELECT:

```
SELECT * FROM table_name;
```

In Cassandra, we always want to query based off the Primary Key. The clustering columns are helpers that can narrow down our search.

Even though Cassandra allows for empty fields, it won't allow this in a clustering column. Cassandra allows for replacing data with an INSERT if the Primary Key is matched.

If you leave a field empty, in the terminal, if you query and receive null, that's just for display purposes. Cassandra isn't actually storing anything.

If you use a partition key with clustering columns, they must be part of your query.

You can use the function writetime (colName) to get the time entered. You can also use UPDATE.

### TTL

Time to Live. You can choose for how long your update lasts. Might be good for something like a link that's only available for so long, or a coupon that needs to be expired after a duration. When TTL executes, it deletes it.

ALTER TABLE is the standard for changing table information.

You can have collections in table fields. Denoted by comma delimited list in curly brackets. If you need to add to the collection, use a + sign. You can remove entries with the - sign. You can remove all in the collection by updating with an empty set.

You can create a secondary index, but it’s not recommended. This denotes poor data modeling.

## UUID

A unique identifier which should not be duplicated. Traditional relational databases increment an integer, but with Cassandra being across a network and numerous noes, incrementing can’t guarantee a unique identifier. Cassandra has a uuid() function which generates a unique identifier virtually guaranteed to never duplicate.

Another function for creating a UUID is now() which bases it on time. This has the added benefit in that it can be sorted by time.

## Counters

Incrementing or decrementing columns that can be used for things like view counts or items purchased. They can only be created in dedicated tables and can’t be the Primary Key. We also can’t delete or index a counter.

We use the counter type. We can only update these, not insert them.

## Import CSV

To import a CSV, the table needs to already exist. It can do about 2 million rows, after that it will likely need the batch command.

```
COPY table_name (cols…) FROM ‘/path/filename.csv’ WITH DELIMITER=’,’ AND
HEADER=TRUE;
```

The header is if you have a row describing the columns.

**Cassandra doesn’t like UTF-8 comma delimited as a save type.**

## Export

```
COPY table_name (cols…) TO ‘/path/filename.csv’ WITH DELIMITER=’,’;
```

Add “AND HEADER=TRUE” to add a row describing the columns. You can also specify what you want to export.

```
COPY table_name (col1, col5) TO ‘/path/filename.csv’ WITH DELIMITER=’,’;
```

### Materialized Views

You can create a new table that extends a base table to be searchable on secondary keys. Cassandra is responsible for maintaining the consistency of the data between the two. However - this can reduce performance, so use wisely.

```
CREATE MATERIALIZED VIEW keyspace.table_child AS SELECT * FROM
keyspace.table_parent WHERE field_name IS NOT NULL …etc… (filters) PRIMARY
KEY (new_PK, old_PK, cluster_key);
```

It must include the old primary key and cluster keys.

## Peer to Peer Architecture

A leader/follower approach leads to chaining nodes, meaning if the leader goes offline, it cascades down the line through subsequent nodes in the chain. This could mean the next follower might be elected a leader and could create gaps in data flow, causing confusion. At that point, you lose the benefit of replication.

In Peer to Peer, each node is equal and all nodes can read or write to each other. A Coordinator Node sends data asynchronously to all the nodes that need it. It doesn't matter where the data comes from, the point of entry node becomes the Coordinator Node. It uses checksum to verify the consistency of data.

If a network partition happens, and nodes become unable to see each other, it depends on the Replication Factor on whether or not the data request succeeds. Once the partition is gone, the data will sort itself out. A cluster aware client will make smart decisions about what Coordinate Nodes to use during a network partition.

### Snitch

A Snitch is responsible for knowing the topology of the node mapping.

**Simple Snitch**: When all nodes are in the same data center and same rack. This is default.

**Property File Snitch**: We manually tell the Snitch where everything is. It will have items like IP address, Data Center, Rack and so on. In large clusters, this could become very difficult to maintain. Gone out of "fashion" due to this manual overhead.

**Gossiping Property File Snitch**: Largely replaced the standard Property File Snitch. It sits on the node itself, informing on itself with things like Data Center and Rack. Uses the Gossip Protocol. It's a lot more automated.

**Web Services Snitch**: Works in conjunction with others. Its purpose is to monitor performance of the nodes. Depending on a variety of factors, the Coordinating Node will interact with those nodes that the Dynamic Snitch has learned work best with each other.

You can adjust the Snitch in the cassandra.yaml file. Changing the Rack and Data Center information can be found in the cassandra-rackdc.properties file.

## Gossip Protocol

Not synchronous, the Gossip Protocol is about the nodes constantly checking in with each other, reporting on what nodes they've interacted with and themselves. It's very efficient and is only passing small chunks of data to check in: status of the node and well being of the cluster.

**Heartbeat State**: Information about when the node started and this particular gossiping session. (Timestamp)

**Application State**: Current status of the node - Normal/Leaving/Joining. Data Center and Rack schema, load, severity (I/O pressure). These are all good signifiers of the performance of the cluster.

It's also checking what is the most up to date. Only updates if it's not up to date.

## Cassandra Write Path

There's a MemTable, in memory, that keeps track of partition keys. When it's written to, it will then write to the Commit Log, on disk. If there's an entry in both the MemTable and Commit Log, it reports back a successful write. If the MemTable goes down, it can be recreated from the Commit Log.

The MemTable will maintain order, but the Commit Log will just append to the end of the file. Eventually the entire MemTable will be stored on disk and then get flushed. It will create on disk an SS Table (Sorted String Table), but in an immutable form. This will delete the MemTable and the Commit Log.

These SS Tables are much easier to read from as they're ordered and represent the data model and we don't have to sort through unordered records. They're also more durable than MemTables because they're written to disk.

For spinning disks, having one to house Commit Logs and another to house SS Tables increases performance. Eventually SS Tables will be compacted

## Cassandra Read Path

The timestamp will be checked between the MemTable and SS Table and it will take the most current timestamp.

Steps Cassandra goes through for reading data:

1. **Bloom Table:** Lets Cassandra know if the data exists at all in the SS Table.
2. **Key Cache:** Stores the byte data where our data begins. It only stores data for frequently accessed data. If it doesn't, it goes to step 3.
3. **Partition Summary:** This is used if our index grows very big. Groups indexes together. Essentially an index of indexes.
4. **Partition Index:** This will house the start of the byte information for each partition, to let the search know where to begin.
5. **SS Table:** Read through the whole table. Obviously not ideal.

Caching **-** row caching is disabled by default, where counter and key caching are enabled by default. Cassandra saves caches to disk frequently, so if the node goes down, it can easily be rebuilt.

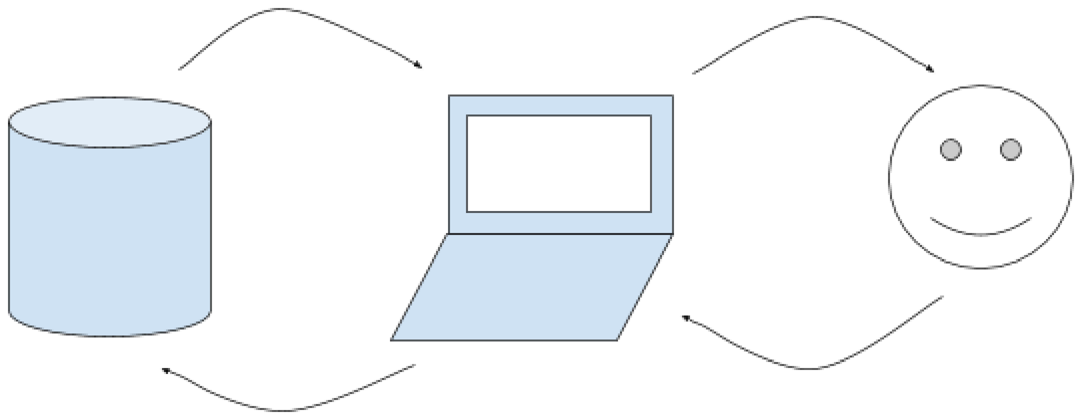
## Compaction

Consolidation using delayed I/O as a background process for performative reasons. It also cleans up data and removes old SS Tables as it goes.

Two tables will get compared and whatever record is most up to date will get written to a new SS Table. Deleting a record is called Tombstoned.

## DB2

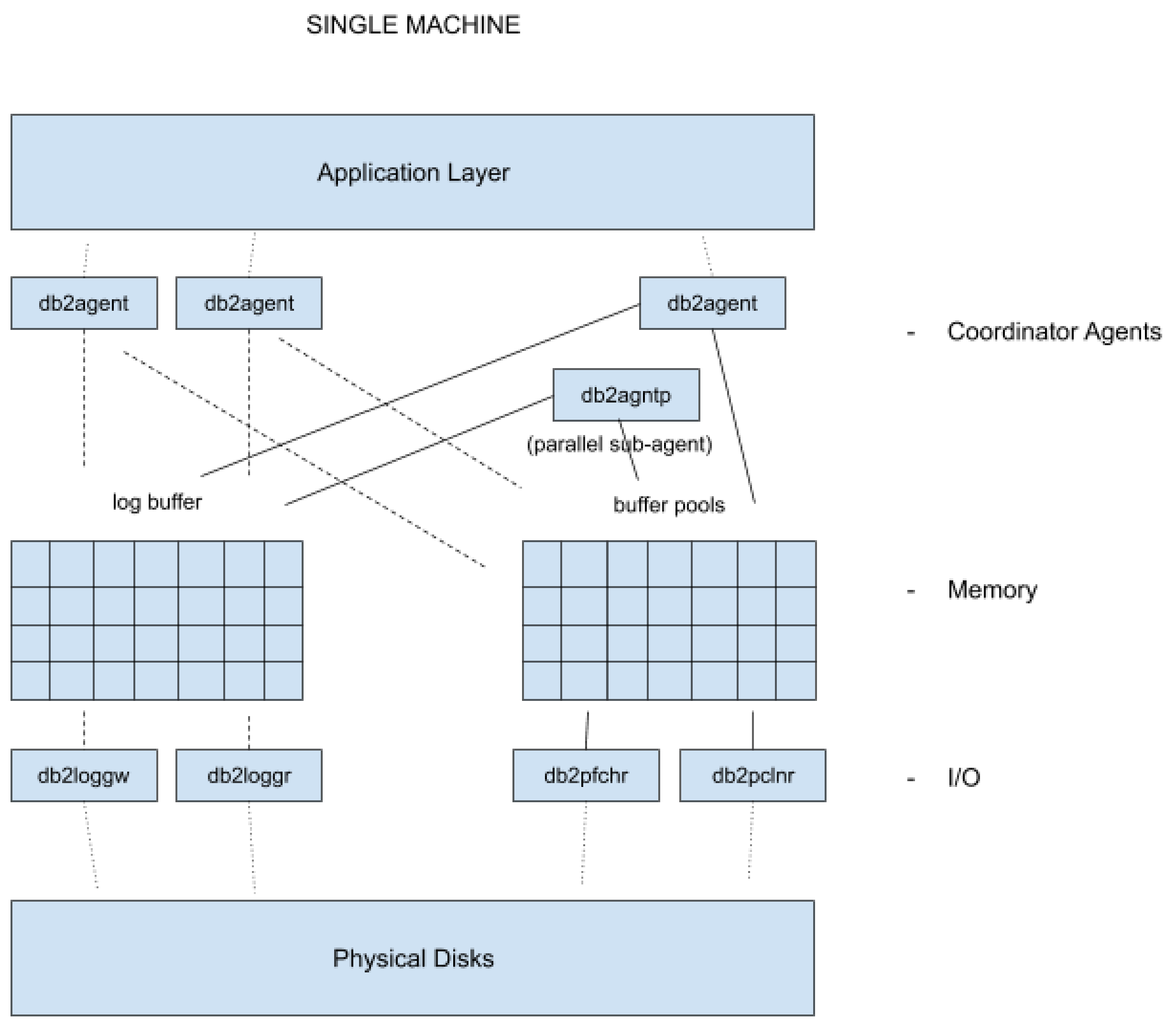
Transactional Process:



User asks for data from application, application gets data and passes it to the user via the application. A database is also good for analytics to help shape business decisions. DB2 supports both structured and unstructured data.

BLU Acceleration
In memory processor

You can have different installations on the same machine.



Log stores a blueprint. In case of failure or data issues, we can restore our data using the logs.

db2pclnr - page cleaner
db2pfchr - prefetcher

db2loggw - log writer (to disk)
db2loggr - log reader (used to recover blueprint

## Processing a Select Statement

Application layer will request data. An agent will collect the request and then follow these steps:

1. Pass statement to agent.
2. Look for execution plan in packet cache and create a plan if needed. Somewhat like Google Maps, in rerouting based on road conditions.
3. Look for pages in buffer pool. If it's not there, it will be collected from disk.

4. Fetch data from buffer pool.
5. Return result to application.

## Processing an Update Statement

1. Pass statement to agent.
2. Look for execution plan in packet cache, create plan if needed.
3. Look for page in buffer pool.
4. Read page(s) in to buffer pool if needed. (By agent.)
5. Update data in buffer pool and write change to log buffer (simultaneously).
6. Write log record to log file.
7. Return confirmation to application.
8. Write changed page(s) to container (asynchronously).

## Storage Locations

| | | |
|---|---|---|
| /db2/db2prd | | Home |
| | /db2_software | Software |
| | /sqllib | Instance |
| /db2/PRD | | |
| | /db2prd | Database Directory |
| | /log_dir | Active Logs |
| | /db2dump | Diagnostic Path |
| | /sapdata1 | DB2 Data Container Files |
| | (sapdataN) | |

## EDU

Engine Dispatchable Unit - Processes/Threads

You can have several databases under the same instance.

**EDUs Per Instance**

| | |
|---|---|
| db2sysc[s.exe] | System Controller Process |
| db2sysc | Main System Controller |
| db2tcpcm | TCP/IP Listener |
| db2ipccm | IPC Listener |

| | |
|---|---|
| db2acd* | Autonomic - Computer Process incl. Health Monitor |
| db2wdog* | Watchdog Process (Unix/Linux only) |

**EDUs Per Connection**

| | |
|---|---|
| db2agent | Coordinator agent |
| db2agntp | Subagents (optional) |

**EDUs Per Database**

| | |
|---|---|
| db2loggw | Log Writer |
| db2loggr | Log Reader |
| db2logmgr | Log File Manager |
| db2pfchr | Prefetcher |
| db2pclnr | Page Cleaner |
| db2lock | Deadlock Detector |
| db2stmm | Self-Tuning Memory Manager |

**Others**

| | |
|---|---|
| db2fmp* | Fenced Mode Process |
| db2bp* | CLP Backend Process |
| db2ckpwd* | PWD Check Processes (Unix/Linux only) |

* These EDUs are processes, even in a thread-based concept.

```
db2prd:> db2pd -edus
db2prd:> db2_local_ps
anyone:> ps -ef | grep db2
```

## CLP

Command Line Processor. Executes database utilities, SQL statements and online help.

## Configuration Overview

1. DB2 Profile Registry
2. DB2 Database Manager Configuration
3. DB2 Database Configuration*
4. DB2 Buffer Pool Configuration*
5. DB2 Tablespace Configuration*
6. DB2 Storage Path Configuration*

*Saved in a full database backup image.

1 & 2 impact the instance. 1,3,4,5,6 impact the database.

A database instance (environment) can have many servers and databases. A server can also have more than one instance.

## DBM CFG

Database Manager Configuration. Displays current and delayed values. Requires an instance attachment. To see changes update for online parameters, you have to restart the instance.

```
$ db2 get dbm cfg for <dbsid> [show detail]
```

A less verbose way to check is to just drop the show detail flag.

```
$ db2 get db cfg for databasename
```

Check IBM for which parameters are online and offline:

```
$ db2 get dbm cfg | more
```

Lists all parameters:

```
$ db2 list db directory
```

Shows all the databases in your instance.

```
$ db2 update dbm using (parameter) (value)
```

We have to attach to the instance because you can have multiple running on the server.

```
$ db2 attach to dbinstancename
```

You can run your show detail command again to verify. Current Value and Delayed Value should be the same. If it's not, likely you need an instance update. Check IBM for parameter options.

It's possible some of the configuration parameters are populated from a read of the system. Must confirm this though.

## Federated Database System Support

A configuration parameter, that when set to true, allows for communication with other databases. If your Diagpath for some reason might not be available, you can use the Alt Diagpath for a backup.

You can cap the size of your log file in MB. The default is 0 which maintains one log file of indeterminate size. if you set a cap, it will divide by 10, and once it gets to 90% of that size it will create a new log file. The other 10% is reserved for the instance notification log. It rotates 10 logs, overwriting the oldest.

You can't create users at the database level. Everything is done at the OS level. There are "permission" levels, where you can assign groups to different roles. SYSADM (all), SYSCTRL (architecture, but no data), SYSMAINT (backups, but no data), SYSMON (snapshots, but no real system access).

## Memory

Instance memory contains all other memory consumers. Database memory claims up to 80 to 90%. Agents also use their own allocated memory. Your database memory is restricted by your instance memory.

You can check memory consumption of an instance with:

```
$ db2pd -dbptnmem
```

Cached memory is used for performance reasons.

When Controller Automatic is set to N, it's NOT automatic and instead manual. Usually high water mark will be same as current usage.

Rebooting a database is a big deal as it has to create all of the caches and package management again, you'll see performance issues.

Setting a heap limit is a soft limit. You don't want your application to fail with a hard limit.

Intra-partition parallelism takes advantage of a multiple core processor, dividing a query over the cores.

## STMM

Self Tuning Memory is a management system that impacts portions of your allocated memory if they are set to automatic. The system will share memory with memory consumers by taking memory from one consumer and giving it to another as load shifts.

Memory consumers that can be tuned automatically:

- Database Shared (database_memory)
- Buffer Pool(s) BP_16k BP_32
- Sort Heap (sortheap)
- Package Cache (pckcachesz)
- Lock List (locklist)

Those that can't:

- Utility Heap (util_heap_sz)
    - Backup Buffer (backbufsz)
    - Restore Buffer (restbufsz)
- Catalog Cache (catalogcache_sz)
- Database Heap (dbheap)
    - Log Buffer (logbufsz)

The Buffer Pool is what will use most of your database memory. It's a tendency of the database architecture to allocate about 80% of your memory to the Buffer Pool.

Parameters that can be self tuned need at least two memory consumers set to automatic. Otherwise it won't work because since it's about sharing memory, only one memory consumer can't share memory with itself. HADR systems only run self tuning on their primary partition.

On Linux/Unix we have swap or paging memory and there's real. Both together is known as virtual memory. The OS will shuffle things from real to swap and back and forth to deal with physical constraints. Ideally however, the hardware should match the needs for memory. Too much swapping is a sign the hardware needs an upgrade.

You can change how many unused memory pages are sent back to the OS.

0 -> immediately
10 -> good general setting
100 -> will never be released

## Package Cache

Memory for static and dynamic queries, access plans, and information about execution of those queries.

Available information:

- Query statements in full.
- Number of executions of a query.
- Execution time - total time on queries.
- Lock wait time - time a particular query spent waiting.
- Sort time - total time a particular query spent organizing and ordering data.
- Rows read - total number of rows a query examined.
- Rows returned - total number of rows a query has returned in result sets.
- CPU time - total CPU time a query consumed while in operation.

It doesn't have all information and so won't replace a SQL statement event monitor.

DB2 moves individual statements in and out of package cache to keep frequently executed statements in memory. A statement might be there a while before it's overwritten.

You can purge it with:

```
FLUSH PACKAGE CACHE DYNAMIC
```

This will necessitate access plans to be regenerated. Deactivating a database will also flush the cache, but resetting monitor switches does not.

## Problem SQL

Any query that causes a slow down or degradation in performance. Common problems:

- Rows read - high number of rows read even though few are selected.
- Execution time - queries that take a long time.
- CPU time - reducing CPU cycles may need to be configured to fix.
- Sort time - takes a long time to sort.

There are a variety of methods you can use to retrieve information about queries, but they can also be performative inhibiting and that needs to be taken into consideration. Problem queries can sometimes be helped application side with tactics such as caching common queries.

## Collating sequence

Database code page determines what characters you can store in the database. A code page is a numeric value given to a named code set. Default is Unicode. If you specify the code page, you must also specify the territory. This cannot be changed after database creation.

```
CREATE DATABASE dbname USING CODESET codeset TERRITORY territory COLLATE
USING collation
```

There is a resource on the IBM site that lets you know what the code page numbers and territory codes are. The collating sequence therefore is an ordering for the character set table you selected for your database. Determines how characters are sorted.

Unicode already has its own weight table using the Unicode Collation Algorithm (UCA). Unicode databases take more bytes.

You can also create custom collating sequences. Each single byte character in a database is represented as a unique number between 0 and 255. In hexadecimal notation, between X’00’ and X’FF’. It’s referred to as the code point of the character. The assignment of numbers to characters in a set is collectively called a code page. The numeric value of the position is called the weight. The weights are equal to the code points. The is called the *identity sequence*.

The weights in a collating sequence need not be unique. You could give an uppercase and the same lowercase letter the same weight. Having a custom collating sequence can have problems if you try and merge two databases that don’t have the same sequence. This is what SQL uses for comparisons and ORDER BY clauses.

If weights are not unique, comparisons will first be done by weight then by code point values. If they are unique, the sequence will not use code point values. Alternate collating sequence (configuration) cannot be used with unicode.

To run a SQL file in DB2:

```
$ db2 -tvf file.sql | tee file.out
```

## HADR

High Availability Disaster Recovery provides a solution for partial and complete site failures. Protects against data loss by replicating data changes from a source database - the primary - to target databases called standby. Supports up to three remote standby servers.

## LOCK LIST

Configuration is for the memory set aside for locks. When multiple applications connect to a database, DB2 can lock rows or whole tables to limit problems with concurrent access to data.

128 or 256 bytes are required to lock an object. 256 initially, 128 subsequently.

When the percentage of a lock list used by one application reaches maxlocks, the database manager will perform lock escalation. Escalation doesn't take much time, but locking entire tables decreases concurrency. Overall database performance might decrease with subsequent accesses against affected tables.

Controlling lock list size:

- Perform frequent COMMITs to release locks.
- On a lot of updates, lock the table. Uses only one lock, avoiding escalation. (Reduces concurrency of data.)
- ALTER TABLE statement has LOCKSIZE option to control how locking is done for specific table.
- Use Cursor Stability Isolation Level when possible to decrease the amount of locking.
- If application integrity requirements aren't compromised, use Uncommitted Read instead of Cursor Stability to further decrease the amount of locking.
- Set lock list to automatic caps. It will increase synchronously to avoid lock escalation or lock list full situation.

A full lock list will degrade performance since it will trigger more table locks vs row locks. It can result in deadlocks between applications which can result in transactions being rolled back. Use lock_escals to monitor escalations to analyze performance.

Maxlocks defines a percentage of the lock list held by an application before an escalation is performed. Lock escalation replaces locked rows with a locked table, thus, reducing the number of locks. It will also escalate if lock list runs out of space.

DBM decides what to escalate by looking through the lock list for the application and finding the table with the most row locks. If after this process locks drop below max locks, escalation will stop. It will keep doing this until it drops below maxlocks.

You can perform a formula to decipher how much memory you need for locking so your hardware matches your performance needs. Use the database system monitor to help you tune this configuration.

## DBHEAP

There's only one dbheap per database and supports database-wide activities.

## Table Spaces

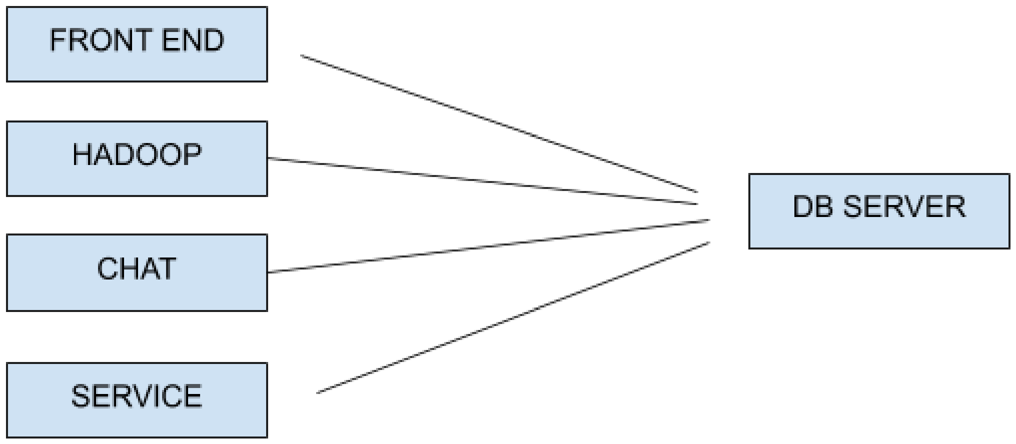
A set of volumes on a disk that holds the data sets in which tables are actually stored. Every table is stored in a table space. Table spaces are separated into equal-sized units called pages.

Types of table spaces:

- Universal
- EA-enabled
- Large Object
- XML

# KAFKA

Apache Kafka is part of a data pipeline. This is a standard distributed computing design:



When you add complexity with multiple data stores, managing communication between systems becomes more difficult. Kafka, a messaging system, does this management between nodes. It also creates a paradigm, independent of platform or language, to make interoperability easier.

Kafka is also distributed and runs in parallel.

## Terms

- Producer - App that publishes messages to a topic.
- Consumer - A subscriber to a topic.
- Partition - Topics are broken up into ordered commit logs
- Broker - A server, part of the Kafka cluster.
- Topic - Category or feed name to which records are published.
- Zookeeper - Managing and coordinating brokers. (Another Apache product.)

Each consumer is set to a particular partition and can read records by their sequential ID (called an offset) and can pick from which offset to start from. Kafka will only keep these records for as long as configured. Seven days is common.

To have parallelism, the partition will be replicated on different brokers. The replication ID matches the broker ID.

Most production environments are multiple-node, multiple cluster setups.

An Event Hub (more information earlier in notes) uses a single stable IP address and consumers and producers don’t need to know about Kafka’s configuration.