Mnesia jest systemem zarządzania bazami danych (DBMS – Database Management System), który został stworzony w celu spełnienia wymagań stawianych przez aplikacje używane w systemach telekomunikacyjnych. Mają one bardzo konkretne wymagania, które nie są spełnione przez większość tradycyjnych systemów zarządzania bazami danych. Do tych wymagań zaliczają się:

* szybkie wyszukiwanie klucza/wartości w czasie rzeczywistym
* wykonywanie skomplikowanych i złożonych zapytań
* rozproszone dane z uwagi na rozproszone aplikacje
* wysoka tolerancja na błędy
* dynamiczna rekonfiguracja (dodawanie, usuwanie danych podczas działania systemu)

Mnesia została stworzona, aby zapewnić rozwiązanie dla tych wszystkich problemów.

Co ciekawe Mnesię zaprojektowali Ci sami ludzie, którzy stworzyli język Erlang, czyli Ericsson. Z tego względu Mnesia jest bardzo ściśle związana z tym językiem i czyni z niego praktycznie język programowania baz danych. Najważniejszą korzyścią z tego wynikającą jest fakt, że kompletnie znika problem dopasowania formatu danych używanego przez DBMS i formatu danych języka programowania, który jest używany do manipulacji danych.

Warto podkreślić, że Mnesia nie jest przeznaczona do zastąpienia standardowych baz danych SQL, lecz służy do innych celów.

Jeśli aplikacja jest napisana w erlangu, ze względu na ścisłe powiązanie z tym językiem, Mnesia umożliwia składowanie danych w tym samym miejscu pamięci, w którym znajduje się aplikacja. Dzięki temu opóźnienie pomiędzy otrzymaniem zapytania, a odczytem danych z bazy jest bardzo małe. W aplikacjach telekomunikacyjnych gdzie wymagany jest czas reakcji np. na poziomie 20 ms, ta cecha Mnesii jest kluczowa.

SQL posiada natomiast większą funkcjonalność. Zapytania w nim konstruowane oraz ich optymalizacja jest znacznie lepsza niż zapytania oferowane przez QLC (Query List Comprehensions) w Mnesii. Jeśli naszym głównym zadaniem jest analiza danych z bazy lepszym wyborem jest SQL.

**Ciekawostka:**

Pierwotną nazwą projektu była „Amnesia”. Jednemu z szefów nie spodobała się jednak ta nazwa. Argumentował, że nie można mieć bazy danych, która zapomina rzeczy. Ostatecznie usunięto więc początkowe ‘A’ i tak powstała Mnesia.

Projekt został opublikowany jako oprogramowanie open source i jego kod można znaleźć w sieci.

**Cechy:**

**Schemat bazy danych może być dynamicznie konfigurowany podczas pracy.**

Jest możliwe skonfigurowanie schematu bazy danych w czasie rzeczywistym bez zatrzymywania systemu.

**Tabele mogą być przenoszone i replikowane do różnych węzłów w celu lepszej ochrony danych. Reszta systemu nadal może używać tablic by dodawać, usuwać lub zmieniać rekordy.**

Tablice posiadają następujące właściwości dotyczące replikacji, gdzie każdy atrybut ma listę węzłów Erlanga jako wartość:

* ram\_copies – tylko w pamięci,
* disk\_copies – w pamięci z synchronizacją na dysk,
* disk\_only\_copies – tylko na dysku.

Zasadniczo są dwa powody dla których powinniśmy używać więcej niż jednej replikacji tabel: **lepsza ochrona przed utratą danych** oraz **prędkość**. Jeśli mamy dwie aktywne replikacje tabel, wszystkie informacje są dostępne nawet jeśli jedna zawiedzie. Co więcej, jeśli repliki istnieją na dwóch różnych węzłach, aplikacja która jest uruchomiona na jednym z takich węzłów nie musi łączyć się z siecią w celu odczytu danych. Dodatkowo replikacja danych może przynieść korzyści dla aplikacji które często odczytują dane natomiast rzadko zapisują – by otrzymać szybki czas odczytu z lokalnego węzła. Z drugiej strony jednak powoduje to dłuższy czas zapisu. Jeśli tabela ma dwa miejsca replikacje to każdy zapis musi się odwołać do obydwu a ponieważ co najmniej jeden z nich jest zdalny trwa to więcej czasu.

**Lokacje tabel są tzw. transparentne dla programisty. Program sam adresuje nazwy tablic i to system śledzi lokacje tabel.**

* Program będzie działał bez względu na to gdzie znajdują się dane. Nie robi to różnicy czy dane znajdują się na lokalnym węźle czy na zdalnym. Oczywiście operacje na zdalnym węźle będą wolniejsze.
* Baza może być rekonfigurowana i tablice mogą być przenoszone między węzłami – te operacje nie wpływają na program użytkownika.

**Transakcje mogą być rozpowszechniane i duża liczba funkcji może być wywoływana w ramach jednej transakcji .**

Transakcja jest to zbiór operacji na [bazie danych](http://pl.wikipedia.org/wiki/Baza_danych). Warunki jakie powinny spełniać transakcje bardziej szczegółowo opisują zasady [**ACID**](http://pl.wikipedia.org/wiki/ACID) (Atomicity, Consistency, Isolation, Durability - Atomowość, Spójność, Izolacja, [Trwałość](http://pl.wikipedia.org/wiki/Trwa%C5%82o%C5%9B%C4%87_%28bazy_danych%29)).

**Kilka transakcji może wykonywać się równocześnie, ich wykonanie jest w pełni synchronizowane przez system zarządzania bazą danych. Mnisia upewnia się, że żadne dwa procesy nie będą manipulować jedną daną jednocześnie.**

**Transakcji może zostać przypisana właściwość, by została uruchomiona na wszystkich węzłach w systemie lub na żadnym. Transakcje mogą również zostać pominięte na rzecz tak zwanych „dirty operations” które redukują ogólne koszty i są bardzo szybkie.**

Dirty operations - szybsze operacje beztransakcyjne, ale mogą popsuć spójność danych - sami musimy się martwić o właściwości ACID. Poważną zaletą stosowania operacji beztransakcyjnych jest szybkość (o dziesiętny rząd większa niż dla transakcyjnych).

Są różne typy transakcji oraz sposoby by uruchomić zapytanie:

**Transaction** – transakcje w Mnesi pozwalają uruchomić serie operacji na bazie w jednym bloku. Cały blok zostanie uruchomiony na jednym węźle lub na żadnym – albo całkowicie zakończy się z sukcesem lub całkowicie z porażką. Ten typ aktywności jest częściowo asynchroniczny. Lokalnie jest synchroniczny natomiast od innych węzłów oczekuje jedynie deklaracji że operacja zostanie wykonana – nie że została wykonana. Jeśli operacja się nie powiedzie to transakcja zostanie odwrócona lecz w późniejszym czasie.

#### sync\_transaction – ten typ aktywności jest podobny do transakcji ale jest on synchroniczny. Transakcja synchroniczna zaczeka na ostateczne potwierdzenie, że operacja powiodła się w 100%

#### async\_dirty – podobnie do transakcji lecz omija wszystkie protokoły transakcji i blokady aktywności.

#### sync\_dirty – analogicznie do transaction i sync\_transaction

#### ets – ostatni możliwy kontekst aktywności. Omija wszystko co robi Mnesia i wykonuje „raw operations” . Brak replikacji. Nie jest to coś czego używa się często.

**Kiedy użyć Mnesi.**

Gdy potrzebujemy replikacji danych.

Gdy chcemy wykonywać skomplikowane i złożone zapytania.

Gdy potrzebujemy wykonania atomowych transakcji do aktualizacji kilku rekordów jednocześnie.

Gdy chcemy mieć system o wysokiej tolerancji na różnego rodzaju awarie.

**Do czego nie używać Mnesi.**

Do programów przetwarzających teksty oraz pliki z danymi binarnymi.

Do bardzo dużych baz danych (dziesiątki i setki gigabajtów)

Jako archiwum danych np. przechowywanie kopii zapasowej