NoSQL و أحفاده الأربعة

NoSQL and its four descendants

بورقبة قويدر

جامعة الجلفة

sorrowdll@gmail.com

ملخص:

يُمثل NoSQL نمطاً أو نوعاً جديداً من أنظمة إدارة قواعد البيانات (Database Management Systems) حيث يتبع أسلوباً مختلفاً عن الأسلوب التقليدي لقواعد البيانات ذات الجداول المترابطة (Relational Database). من أبرز أوجه الخلاف بين هذين الأسلوبين : الجداول, حيث لا يتخذها NoSQL كوحدة الأساس لبناء قواعد البيانات على عكس الـــ Relational Database, لهذا السبب, تستخدم NoSQL لغة UnQL كبديل للغة SQL في التعامل مع البيانات.

لعل أول سؤال يتبادر إلى ذهن القارئ هو : هل أتت NoSQL لتحل محل الـــ RDBMS؟

و الجواب ببساطة : لا!

فقد اختار مؤسسو هذه التقنية الاسم NoSQL كاختصارٍ لـــ Not Only SQL للدلالة على أن هذه التقنية لم تأتِ للقضاء على الـ RDBMS و إنما تُمثل أحد البدائل المُقترحة حيث تُقدم العديد من الحلول خصوصاً في الحالات التي يكون فيها الـ RDBMS عاجزاً عن توفير حلول سهلة, فعالة و مفتوحة المصدر.

أغلب التطبيقات الموزعة (distributed applications) الموجهة للإنترنت (internet-oriented) التي تعمل على قواعد بيانات عملاقة جدا, تستخدم أحد توابع NoSQL لإدارة و تسيير قواعد بياناتها. (انظر الأمثلة الموجودة في الفقرة الرابعة)

Abstract:

NoSQL represents a new type or type of database management systems, as it follows a different approach from the traditional method for relational database databases (Relational Database). Among the most prominent differences between these two approaches: Tables, unlike Relational Database, NoSQL does not take it as the basis for building databases, for this reason, NoSQL uses UnQL as an alternative to SQL in dealing with data.

Perhaps the first question that comes to the reader's mind is: Did NoSQL replace RDBMS?

The answer is simply: No!

The founders of this technology chose the name NoSQL as an abbreviation for Not Only SQL to denote that this technique did not come to eliminate RDBMS, but rather represents one of the proposed alternatives where many solutions are presented, especially in cases where the RDBMS is unable to provide easy, effective and efficient solutions. Open source.

Most internet-oriented distributed applications that run on very giant databases use a NoSQL method to manage and manage their databases. (See examples in the fourth paragraph)

المؤلف المرسل: ..... ، الإيميل: .....

. مقدمة:

منذ أربع عقود مضت, أعطى العالم البريطاني المشهور **Edgar Frank Codd** الضوء الأخضر للأميرة المدللة **Relational Database** لتدخل قصر قواعد البيانات من أوسع أبوابه. منذ ذلك الوقت, بدأت أميرتنا الصغيرة بفرض سيطرتها في كافة أرجاء القصر, و بعد عشر سنوات من توليها إدارة شؤونه, بدأت حركة الـ **Object Database** في التمرد بقيادة **CPP** لتنضم إليها **Java** في أواخر التسعينات. لكن سرعان ما تدارك الباحثان **Christopher J. Date** و **Hugh Darwen** الأمر عندما وقَّعا اتفاق المصالحة المشهور بعنوان البيان الثالث **(The Third Manifesto)** في عام 1995 و الذي كان من شأنه إعادة روح الثقة بين كلا الطرفين.

بعد ثلاث سنوات من عقد الاتفاق, بدأت الأصوات تتعالى من جديد و السبب هذه المرة هو الغلاء الصارخ لأسعار الـ **DBMS** بالإضافة إلى القيود التي تفرضها الرخص التجارية, لذا بدأ وجهاء القصر بالبحث عن زعيم جديد يُمكنه إقناع المحتجين و تلبية طلباتهم مع الحفاظ على الأدوار التي كانت تقوم بها الأميرة المدللة. في الحادي عشر من حزيران 2009 اجتمع وجهاء القصر بقيادة الحكيم **Shashank Tiwari** لتجديد البيعة و انتهت الجلسة بتعيين الزعيم **NoSQL** ملكاً لقصر قواعد البيانات خلفاً للأميرة المدللة. بعد ساعات قليلة من توليه السلطة, عقد الزعيم اجتماعاً طارئاً حضره وكلاء الأسر التي تضررت أثناء حكم الصغيرة المدللة و تكفل بتحمل المسؤولية و إصلاح ما أفسدته الأميرة الصغيرة. كانت تلك مجرد مقدمة مختصرة أردتُ من خلالها استعراض مختلف مراحل التطور التي مرت بها أنظمة إدارة قواعد البيانات, ابتداء من نشأتها وصولاً إلى يومنا هذا. سنكتفي بهذا القدر من التنظير لننتقل إلى الجانب العملي من هذه المقالة.

- ACID و BASE أيهما الأفضل ؟

التعامل مع البيانات الحساسة الموجودة في قواعد البيانات يتطلب آلية محددة و دقيقة بحيث تضمن تكامل و تناسق البيانات أثناء إجراء عدة عمليات في آن واحد. يُمثل كل من **ACID** و **BASE** فلسفة في التصميم و آلية مختلفة للحفاظ على تكامل البيانات أو اتاحتها حيث يرى **BASE** أن الأولوية تكون دائما لإتاحة البيانات **(availability)** وجعلها قابلة للتصفح من طرف مختلف عُقَّد النظام بينما يرى نظيره **ACID** أن الأولوية يجب أن تكون من نصيب تناسق البيانات فيما بينها **(consistency)** حسب القواعد المُحددة.

في هذه الفقرة سنتطرق إلى أوجه الخلاف بين هذين الأسلوبين و أيهما الأفضل و متى يكون ذلك ؟

يُعتبر **BASE** اختصاراً لـ **(Basically Available, Soft-state, Eventually-consistent)** و يعتمد على الخصائص الثلاث التالية :

1-Basically Available : و تعني أن النظام مُتاح بشكل دائم و يمكن الوصول إليه حتى في حالة الـ **desynchronization**. في هذه الحالة نجد أن الـ **availability** مضمونة بشكل كامل على عكس الـ **consistency** لأن النظام سيرد على جميع الطلبات لكن قد يحتوي الرد على بيانات غير متناسقة **(inconsistent data)** أو يكون الرد مثلا عبارة عن **failure**.

2-Soft-state : تدل هذه الخاصية على أنه يمكن لحالة النظام **(state of the system)** أن تتغير مع مرور الوقت حتى لو لم تكن هناك بيانات جديدة حيث يُمكن أن تحدث تغييرات في قاعدة البيانات بسبب الـ **eventual consistency**.

3-Eventual consistency : و تعني أن النظام لا يهتم بالـ **consistency** بعد إجراء **transaction** معينة لكن في ظرف معين من الزمن و بعد عدد غير محدد من العمليات سيستقر النظام على مجموعة من البيانات المتناسقة و التي تلتزم بالـ **consistency rules** مع أن هذه الأخيرة هي آخر من يعلم على عكس ما يحدث في **ACID**.

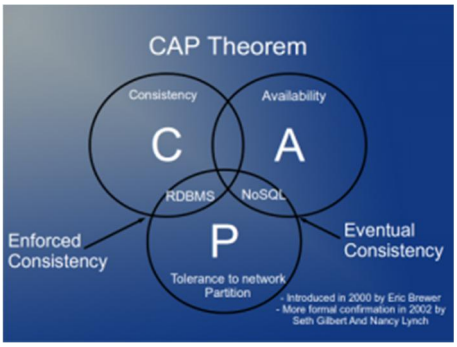
يُعد **ACID** من أكبر منافسي **BASE** حيث يعتمده حالياً معظم الـ **SGBD** مثل **Oracle, SQL Server** و **Informix.** يرتكز **ACID** على الخصائص الأربع التالية :

1-الـ **Atomicity** : تنص على هذه الخاصية على أن أي **transaction** إما أن يتم تنفيذها بشكل كامل أو إلغائها بشكل كامل و بالتالي لا توجد **half-completed transaction.**

2-الـ **Consistency** : كل **transaction** يجب أن تنقل مجموعة البيانات من حالة متناسقة **(consistent state)** إلى أخرى مماثلة مع الحفاظ على الـ **integrity constraints.**

3-الـ **Isolation** : تضمن هذه الخاصية التنفيذ المتزامن لمجموعة من الـ **requests** في آن واحد حيث يتم تنفيذ كل واحدة بمعزل عن البقية و بالتالي لا يمكن لأي **transaction** أن تصل إلى **transaction** أخرى لم يكتمل تنفيذها (أي في حالة الـ **unfinished**).

4-Durability : بمجرد الانتهاء من تنفيذ الـ **request** بنجاح, لن يتم التراجع عن التعديلات الناجمة عن تلك العملية حتى لو تعطل النظام لاحقاً.



بالنسبة للتطبيقات الموزعة **(distributed computer systems)**فتنص نظرية **Brewer** المشهورة باسم **CAP Theorem**بأنه يمكن فقط ضمان اثنتين من الخصائص الثلاث التالية في نفس الوقت:

Consistency: في نفس الوقت, تُشاهد كافة عُقَد النظام نفس البيانات.

Availability: تحصل جميع الطلبات على رد يُوضح نجاح تنفيذ العملية أو عدمه.

Partition Tolerance: لا يتوقف النظام عن العمل إلا في حالة تعرضه لتعطل كامل, وفي الحالة المعاكسة تظل الشبكات الفرعية تعمل بشكل مستقل.

بشكل عام, عادة ما يبقى الخيار ما بين الـ **Availability** و الـ **Consistency** بافتراض وجود الـ **Partition Tolerance** كخاصية ثابتة في أغلب الأنظمة الموزعة. تُستخدم **NoSQL** في الغالب في التطبيقات الموزعة التي تلتزم بقواعد **ACID** و تعتمد أحد خيارات **Brewer.**

بقي أن نُشير إلى أن **NoSQL** عادة ما تُستخدم في قواعد البيانات المُوجهة للانترنت.

-أحفاد الزعيم الأربعة :

للزعيم أربعة أحفاد:

1-Key-values Stores

1-**1-تعريف** : توجد علاقة بين المفتاح و القيمة, تماما كما هو الحال مع جدول هاش **(HashMap),** القيمة قد تكون سلسلة محارف أو **serialized object** مثلاً, غياب الـ **datatyping** في هذا الـ **model** له تأثيرٌ مهم جداً على الـ **Querying** لأن التواصل مع قاعدة البيانات سيقتصر على ثلاث عمليات فقط : **PUT, GET** و**DELETE.**

2-**1- أمثلة:**

-Memcached: يقوم بتخزين البيانات فقط في الـ **Random-access memory,** عندما يتوقف النظام عن العمل يتم فقدان كافة البيانات لذلك يُستخدم عادة لإدارة و تسيير الـ **Cache.** يعمل على **Windows, Linux** و **Mac OS**وهو متوفر برخصة **Permissive.** ظهر الـ **Memcached**لأول مرة قبل عشر سنوات من الآن حيث كانت تستخدمه **LiveJournal** وفي عام 2010 وصل انتشاره إلى عدة مواقع عالمية نذكر منها على سبيل المثال : **Orange, Wikipedia, YouTube, Facebook, Twitter.**

-Voldemort: تم إصدار أول نسخة منه في 2009 و قد أُخذ اسمه من الشخصية الخيالية للساحر **Lord Voldemort** في سلسلة **Harry Potter** المشهورة. تُعتبر النسخة 1.3.0 آخر إصداراته المستقرة حيث لا يزال في تطوير مستمر. اختار **Voldemort**الثنائي **AP** من نظرية **.Brewer** قامت بإنشائه شركة **LinkedIn** حيث كُتب بلغة **Java** لذا فهو متعدد المنصات.

-Amazon DynamoDB: توفره شركة **Amazon** حيث يتبع للفرع الخاص بخدمات الويب, يُعتبر أحد توابع **DynamoDB,** والفرق بينهما يكمن في اختلاف الـ **implementation**حيث نجد الأول يعمل حسب أسلوب الـ **multi-master**بينما يعمل الآخر وفقاً لأسلوب الـ **single master.** استُخدمت عدة لغات برمجة لكتابة هذا النظام, منها **Java, Node.js, .NET, Perl, PHP, Python:** و **Ruby.**

:Column-Oriented-2

2-1- **تعريف** : يُشبه إلى حد ما الجداول الموجودة في الـ **RDBMS**إلا أن عدد أعمدته ديناميكي على عكس الجداول التقليدية حيث يُمكن وجود عدة **records** بأعمدة مختلفة مما يسمح لك بتفادى وضع **NULL** في الخانات الزائدة.

2-**2-أمثلة:**

- **Cassandra:** تمت كتابتها بلغة جافا من طرف **Facebook**حيث أُعلن عن اولى اصداراتها في 2008 لكن سرعان ما تخلى عنها لصالح شركة **Apache,** تُعتبر الحسناء **Cassandra**أحد أفراد عائلة الـ **BigTable** بشهادة العميد **. Jeff Hammerbacher**تم تصميمها أساسا لإدارة و تسيير قواعد بيانات عملاقة, موزعة على عدة **servers** مع ضمان اتاحة البيانات بشكل دائم. تحتل الحسناء **Cassandra** المرتبة 11 من بين أنظمة قواعد البيانات الاكثر استخداماً في العالم, حسب احصائيات موقع **.DB-Engines** يستخدمها الآن كلٌّ من **Twitter, Digg** و **Reddit** بالإضافة إلى الـ **Facebook**طبعاً.

- **HBase:** يُمكن من تخزين جداول ضخمة جداً بشكل منظم, كُتب بجافا, متعدد المنصات و متوفر برخصة **Apache2.0,** ينتمي هو الآخر إلى عائلة الـ **.BigTable** يُعتبر **HBase** أحد توابع الـ **Hadoop** حيث يُستخدم عادة مع **HDFS** لتسهيل عملية التوزيع لكن مع ذلك كما يُمكن ايضا استخدامه في مجالات أخرى متعددة.

- **Apache Accumulo:** تم إنشائه عام 2008 من طرف وكالة الامن القومي **(NSA)** و استولت عليه شركة البرمجيات **Apache**بعد سنتين من اطلاقه. في 21 آذار 2012 خرج من حضانة **Apache** لينتقل إلى مستوى أعلى و بعد سنة من تمرده اعلن عن نسخته الجديدة 1.4.3 في الثامن عشر من آذار 2013. يعتمد هذا النظام على تقنية الـ **BigTable** و يحتل المرتبة الثالثة ضمن أفراد عائلته حسب احصائيات موقع**DB-Engines.**

Document-Oriented-3

3-1-**تعريف** : يعتمد هذا الـ **model** على نموذج الـ**Key-values,** القيمة في هذه الحالة هي ملف **JSON** أو **XML,** الهدف من تمثيل كهذا هو إمكانية الحصول على مجموعة بيانات مرتبة بشكل شجري من خلال مفتاح واحد فقط, نفس العملية تُقابلها مجموعة من الـ **joins** في الـ **RDBMS** التقليدية.

3-**2-أمثلة:**

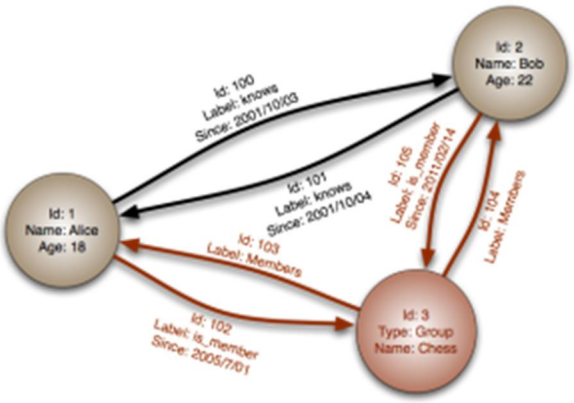
- **CouchDB:** نظام لإدارة قواعد البيانات, مُوجه بشكل خاص إلى تطبيقات الويب, تم إطلاقه في 2005 و بعد ثلاث سنوات, اشترته **Apache** ليصبح أحد مشاريعها المنافسة على الساحة. كُتب بلغة **Erlang,** متوفر برخصة **.Apache** يستخدم ملفات **JSON** لتخزين البيانات و **JavaScript** كلغة استعلام تعتمد على خوارزمية**.MapReduce**

- **Couchbase Server:** كان يُعرف سابقاً باسم **Membase,** يسمح بتنفيذ آلاف العمليات في آن واحد كما يضمن اتاحة البيانات لكافة المستخدمين في أي وقت. كُتب بـ

C, CPPو **Erlang,** متعدد المنصات. تم إطلاق آخر نسخة منه في الثالث عشر من أيلول 2013.

- **MongoDB** : تم اطلاقه في عام 2007 من طرف شركة **gen**10 حيث اشتُق اسمها من كلمة **humongous** و تعني عملاق, فعال جداً و لايتطلب مخطط (**schema**) مُعرف مسبقاً, يستخدم ملفات من نوع **BSON** لتمثيل البيانات. كُتبت بـ **CPP** و متوفرة برخصة **AGPL**. صدرت آخر نسخها في اغسطس 2013.

4-Graph Databases



4-1- **تعريف** : يعتمد هذا الـ **model** على الـ **GraphTheory**حيث تحتوي كل عقدة (على الأقل) على مؤشر يحمل عنوان العقدة الموالية و بالتالي لا حاجة في استخدام الـ **indexes.** يُستخدم هذا النموذج عادة في الشبكات الاجتماعية حيث يسمح بالحصول على معلومات الأعضاء بسرعة, كما يُمكن أيضا من إيجاد العلاقات التي تربط الأعضاء فيما بينهم بشكل سريع جداً (مثلَ إيجادِ أصدقاءِ أصدقاءِ الأصدقاء لعضو معين).

4-2-أمثلة:

- **Neo4j:** يُعد من أشهر أنظمة قواعد البيانات التي تستخدم الـ **graphs,** تم إطلاقه في عام 2000 من طرف شركة **Neo Technology,**كُتب بلغة جافا, متعدد المنصات و متوفر برخصتين هما **GPLv3** و **AGPLv3.** تم الإعلان عن اصدار النسخة 1.0 منه في العاشر من شباط 2010.

- **OrientDB:** أعلنت شركة **Luca Garulli** عن أولى نسخه قبل ثلاث سنوات من الآن, يستخدم الـ **documents** لتخزين البيانات و يعتمد على الـ **graphs** لربط الملفات بعضها ببعض, يدعم **SQL** كلغة استعلام كما يدعم على التوالي كلا من **schema-less, schema-full** بالإضافة إلى**-schema mixed.**كُتب بالكامل بلغة جافا و قد صدرت آخر نسخة منه في نهاية تموز من العام الحالي.

- **InfiniteGraph:** كُتبت نواته بـ **CPP** و البقية بجافا وهو أحد منتجات شركة **Objectivity** حيث أعلنت عنه في عام 2010, متعدد المنصات و متوفر بنسختين إحداهما تجارية و الأخرى حرة.

بطبيعة الحال, توجد أنواع أخرى من **NoSQL** لكنها أقل شهرة مثل **ElasticSearch** وهو عبارة عن محرك بحث متعدد المنصات يعمل على قاعدة بيانات **NoSQL** كما توجد أنواع أخرى ربما لم تسمع عنها سابقاً مثل **MarkLogic, TokuMX, FleetDB** بالإضافة إلى **SciDB.**

- **جولة سريعة مع العملاق الرائع** MongoDB

-تعريف : **MongoDB** عبارة عن نظام لإدارة قواعد البيانات يستخدم ملفات **BSON** و هي اختصار لـ **Binary JSON.** يتميز **MongoDB** بخاصية الـ **ShemaLess** إذ لا يلتزم بـ **schema**معين لذا قد يختلف محتوى الملفات من حين لآخر.

يُعتبر **Mongo** أيضا نظام **scalable** حيث يُمكنه التأقلم مع آلاف الطلبات في آن واحد دون أن يؤثر ذلك على سرعة الأداء.

-المميزات

يُوفر لغة استعلام غنية و مليئة بالدوال الجاهزة.

يتميز بسرعة الأداء و الكفاءة العالية خصوصاً عند إدخال بيانات ذات حجم كبير جداً باستخدام الـ **batch mode** حيث يُمكن إدراج مائة ألف عنصر في الثانية الواحدة.

يسمح بعمل **indexing** لخصائص الملفات من أجل تسريع عمليات البحث.

يُوفر العديد من الدوال التي نجدها في الـ **RDBMS** التقليدية مثل **(count, group by, ...)** و يُضيف مميزات أخرى متقدمة مثل استخدام سكربتات **MapReduce** بلغة **JavaScript** لزيادة سرعة الأداء بالإضافة إلى إمكانية البحث عن بيانات معينة اعتماداً على الـ**geolocation.**

القدرة على عمل **Replication** و **Partitioning** في عدة **instances.**

تتكون قواعد بيانات الـ **mongo** من مجموعة من الـ **collections,** تحتوي كل **collection** على عدد غير مُحدد من الـ **documents** حيث يعتمد حجم الـ **collection** على عدد الملفات الموجودة داخلها فعند إضافة ملف جديد تتم زيادة الحجم بشكل تلقائي ويتم تقليصه عند حذف أحد الملفات. أحد أهم الفروق بين الـ **mongo** و قواعد البيانات التقليدية هو أن بنية الملف لا تلتزم بقواعد معينة على عكس الـ **MySQL** مثلا, حيث نجد أن مكونات الـ **rows** هي نفسها في الجدول الواحد إذ يجب أن تخضع لقواعد الـ **schema** الذي تم تعريفه أما الـ **mongo** فلا يعتمد أي مخطط و بالتالي يُمكن لبنية الملف أن تتغير من وقت لأخر داخل نفس الـ**collection.**

ملاحظة : الملفات في الـ **mongo** تُقابل الصفوف في الـ **MySQL.**

-توزيع البيانات**(Data Distribution)**

يدعم الـ **mongo** نوعين من التوزيع:

Replica set : حيث يتم استخدام أحد السيرفرات كـ **master** و الآخرين كـ **slaves.** في مثل هذا التوزيع, يُنصح دائما بالبدء مع 3 سيرفرات, يتم اختيار أحدهم كسيرفر رئيسي **(master)** و الاثنان الباقيان كسيرفرات ثانوية **(slaves).** تُنسخ البيانات المُخزنة في الـ **master** إلى الـ **slaves** بشكل تلقائي و في حالة تعطل السيرفر الرئيسي يتم انتخاب السيرفر الثانوي الذي يحتوي على البيانات الأحدث. السيرفر الرئيسي فقط هو من يمكنه القيام بعمليات القراءة و الكتابة أما الاثنان الباقيان فيُسمح لهم بالقراءة فقط حيث ينحصر دورهم في تخزين البيانات الموجودة في الـ **master** خوفاً من ضياعها.

Sharding : يتم تقسيم قاعدة البيانات لضمان الـ **availability** في كل وقت. مفهوم الـ **shards** يعني مجموعات من الـ **replicaset,** كل مجموعة تحتوي على جزء من البيانات و الهدف من هذا هو توزيع أو تقسيم المهام بين مختلف الـ **replicasets.**

.الخاتمة :

في نهاية المطاف, نُذكر مجدداً أن **NoSQL** لا يُمثل حلا سحرياً لجميع مشاكل قواعد البيانات إذ يظل استخدامه مقتصراً على ميادين معينة و بالتالي يجب على المبرمج إحسان الاختيار مع الأخذ بعين الاعتبار طبيعة الـ **software architecture** و تعقيد البرمجة و التطوير في حالات معينة.

في الحقيقة, ما زال الزعيم **NoSQL** يحتاج إلى اعتراف الـ **standards** به بشكل أكبر, تماما كما هو الحال مع **JDBC** و **SQL** في الـ **RDBMS** لكن أعتقد أن الوقت ما زال مبكرا فالزعيم لا يزال في ربيع شبابه و أعتقد أن أمامه مستقبل واعد إن حافظ على سرعة الانتشار التي يتقدم بها حالياً.

-المراجع الأساسية :

1- Stephens ,J; Russell ,C Database Design and Optimization From Novice to Professional. First ed, published by Apress, Publishers, Inc New York, 2004, 332.
2- Thomas C; Carolyn B, Database Systems, third Edition , Addison Wesley, Sep 2002.
3- Risch, T; Tuning the Reactivity of Database Monitors, Hewelett--Packard lab, Palo Alto, CA, USA, RPT. No. HPL-90-17, 22 pp., Mar. 2002.

-المراجع الإضافية:

1- Jacobs, D, Kastelic F; Database Manager Database Event Monitor, IBM Technical Disclosure Bulletin, Sep. 2002.
2- 2- Jonathan L, physical database design and the strategic, Publisher

3- s ,Inc New York, 2004.