SNS COLLEGE OF TECHNOLOGY

(An Autonomous Institution)

Re-accredited by NAAC with A+ grade, Accredited by NBA(CSE, IT, ECE, EEE & Mechanical)

Approvedy by AICTE, New Delhi, Recognized by UGC, Affiliated to Anna University, Chennai

Department of Computer Applications

Non Relational databases

Course: NoSQL Database system

Class / Semester: II MCA / III Semester

The issues

- The concept of a data model
- "If you want massive, on demand scalability, you need non-relational database" (industry spin?)
- **key/value store** (no official name exists)
- **xml "databases"**
- Remember fundamentals?

Fundamentals

- C.J.Date
  - A model of anything is a construction – as concrete or abstract as we wish.
  - Capturing certain aspects (the ones we are interested in)
  - Features and properties of the model match in a certain way things in the world
  - A data model is fairly abstract.
  - The properties we are interested in are *intrinsic*.

Cont.

- *Intrinsic* - naturally belonging or essential.
- That is, we are interested in the logical aspects of data, not the physical aspects.
  - What's the difference?
- A good data model captures the logical aspects – ideally ALL the logical aspects
- A good data model should not include anything else

A good data model

- Includes the set of objects and the set of operators for operating on those objects.
- Objects allow us to model the structure of the data
- Operators allow us to model the behaviour of the data
- Together this constitutes an *abstract logical machine (C.J.Date)*

Physical v logical (again)

- The abstract logical machine is distinct from it's implementation (which is physical)
- The model is what the users know about – the implementation is what the users don't know about
- So, objects and operators serve as the basis for further investigations into the nature of the data.

Theory vs pragmatics (again)

- In short a model of data is really a theory of data
- It’s not static. Hopefully it becomes more faithful over time.

Pascal on Data Models

Desirable characteristics

- **Generality** – ability to represent any kind of data
- **Formality** – sound theoretical basis
- **Completeness** – supports all four components
  - Data types, structure, integrity, manipulation
- **Simplicity** – as simple as possible (but not too simple)

Benefits of RDBMS

- Simplicity
- Robustness
- Flexibility
- Performance
- Scalability
- Compatibility

Key Value Databases

Tony Bain., (2009) Is the Relational Database Doomed?

- No official name exists
- Sometimes called document oriented, internet facing, attribute oriented, sharded point arrays, distributed hash-tables, and key/value databases
- Been around for a while.
- Generally associated with web and cloud application environments

Oracle's NoSQL

- Recall ACIDS (RDBMS), NoSQL is BASE
- Basically Available
- Soft state
- Eventually consistent

- Developers are expected to design around these limitations.

Scalability

- What do you do if your DB load increases 3 fold overnight?
- RDBMS scale up on single server node ok – when capacity is reached – you need to scale out and distribute data across multiple servers.
- Result – complexity (at physical level) of RDBMS starts to rub
- What if you need thousands of nodes?

The New Breed

- Been around for a while but come into their own with Cloud and Web apps.
- In particular very very large social media applications (linkedin, twitter, facebook)
- Focus on scalability at the expense of the traditional RDBMS features.

Fundamentally different to RDBMS

Relational	Key/Value
Tables, columns and rows – strict table definition re attribute data types etc.	Domains – a little like tables, but no strictly defined schema like a table (ie, attribute column definitions). Domain is basically a bucket that you put items into
Strict data model is defined prior to implementation – strongly typed, implements integrity constraints and relationships enforce data integrity	Items identified by keys and any item has a dynamic set of attributes associated with it
Data model based on a natural representation of data. Not on the applications required functionality (*data independence*)	Some implementations attributes are all of a string type. In other implementations attributes have simple types int, string arrays etc.
Data model normalised – normalisation establishes relationships of associated tables (PK – FK linkages	No relationships defined between domains or within domains

No Entity Joins?

- Key value DBs are Item-Oriented
- All relevant data relating to an item are stored within that item
- For example. A domain may contain customer purchased items and order information.
- Data is commonly duplicated across items in a domain

Cheap disk space and scalability

- This is accepted practice because no longer is disk space at a premium (lazyness?)
- Relational model requires tables to be joined to regroup related attributes

Example

Car	
Key	**Attributes**
1	Make: Nissan Model: Pathfinder Colour: Green Year: 2003
2	Make: Nissan Model: Pathfinder Colour: Blue Colour: Green Year: 2005 Transmission: Auto

Some relationships exist

but there seems to be no rule about which ones should be defined

- Some relationships are created between "core" entities
- The example Bain gives is in an online store you wouldn't store customer and
- product attributes in the same domain. (well, I suppose that's one good point!)
- Orders in these semi-related domains contain keys to the relevant domain and items

Where is the integrity?

- But, relationships are not defined in the data model, just loosely modeled at the application level.
- ***No enforcement of integrity at all in the data model***
- Means you can delete customers and the products they have ordered
- Data integrity enforcement the sole responsibility of the application.

Data Access

Relational	Key/Value
Insert, Update, Delete, Select via SQL	Insert, update, delete and select via some API method call.
SQL queries access tables or multiple tables using join operations	Some examples provide basic SQL like syntax for defining filtering criteria
SQL includes functions for aggregation, grouping, filtering etc	Some basic filter predicates such as =, !=, <,>, and so on
Ability to embed business logic (rules) in the database using various categories of constraints	All application and data integrity code is in the application

Application Interface

Relational	Key/Value
Have specific APIs or make use of generic API such as OLE-DB, JDBC, ODBC etc.	Usually offers SOAP, and or REST APIs over which data access calls can be made SOAP (XML based data exchange message protocol, for web based services) REST (Representational State Transfer, used for distributed media, a bit different to SOAP but essentially doing the same thing)
Data represented at logical level in a format that represents it's natural structure, so needs to be mapped between application code (particularly with OO application Object Classes) and the relational structure	Data effectively entwined in the application based on it's structure. Concept of data independence is out the window.

The advantages

- Benefit of key/value is meant to be their scalability and suitability for use in the cloud
- Usually implemented in massive distributed DB environments
- Easy to dynamically scale
- Also choice of vendors who require cheap or free data store platforms with massive potential to scale

More natural fit with code

- Relational models and Application coded Object models are clearly different and require Object-to-relational maps to bring them together.
- Key/value database can be built closely matching the object class model
- Reduction in development time

Disadvantages

- Constraints in the relational model ensure data do not violate data integrity
- This simply does not exist in key/value databases
- Thus, any bugs in app code could well lead to data integrity issues. This is not the case in a relational model (because the data is independent to the application remember)

Cont.

- No real query language (have to write code)
- Relational model forces you to go through the modelling process
- This results in robust, formal logical model that reflects data requirements (as opposed to application requirements)
- Data is then *application independent* and we all agreed from Ted Codd on, that this was a good thing.
- This means lots of applications can front-end the database – not possible in key/value dbs

Cont.

- Standards – there are none!
- The relational model is rich with standardisation. SQL, the data's logical structure and so on, portability is relatively straight forward.
- What is the standard the Cloud oriented DBs are based upon? There is none. This is emerging technology.
- Each have their own APIs, query interfaces, and so on. If you don't like the one you are using it is difficult to switch – if you don't like Oracle you can move to SQLServer in a day.

Cont.

- Many key/value databases are still beta
- Limited analytics (complex query facility)
- Relational model offers extremely rich OLTP options, data warehouse, data mining etc, these options don't yet meaningfully exists for key value
- So, you have to build a analytical DB from scratch. (or import to relational)

Examples (Cloud)

- Amazon – SimpleDB
  - Provided by amazon services, still in beta, there is a free version
  - Many limitations – eg. No datatypes except for strings (wtf?) everything is stored, retrieved compared in string.
- Google AppEngine Data Store
  - Richer data type support
  - Must build applications based on Googles webservice platform

Cont.

- Microsoft: SQL Data Services
  - SDS - Part of Azure webservices platform
  - Sits on top of SQLServer
  - While underlying structures may be relational you don't have access to this
  - SDS is key/value
  - Microsoft seem to be offering on customer demand but worry about data management issues compared to RDBMS

Non-Cloud examples

- CouchDB
  - Free, open source
  - Apparently poster-child for the no-SQL community
  - Part of Apache
  - Version 1.0.0 had a major data loss bug.
    oops.

Cont.

- Project Voldemort
  - Came out of the LinkedIn project
  - Very new, current release 0.90.1
  - We await with interest a first stable release.
- Mongo
  - Document oriented
  - Designed to be a true Object database
  - Version 2.0.3 released Feb 2012

Cont.

- **Oracle NoSQL** DB
- Offers XML/Xquery, Java Object, SQL or Key/Value options
- All admin provided by an API
- Support for mobile devices
- Offers recoverable ACID transactions
- Scalable to 256Tb as single data file – multiple data files supported.

Conclusions

- Popular when data is heavily document oriented
- Appears to be a very ad-hoc approach to DB implementation
- No real data model associated with this approach
- No or little data integrity
- All constraints and business rules need to be enforced in the application code

Conclusion cont.

- These "databases" are cheap, (or free) and very new, some not even out of beta release
- Extreme limitations and dubiously suited to organisations that rely upon data integrity – ie, and professional commercial enterprise
- Used by many of the social networking sites, twitter, LinkedIn, facebook etc.

Is the relational model doomed?

- We should still store our Enterprise in Relational DBs

RDBMS doomed? Not yet.