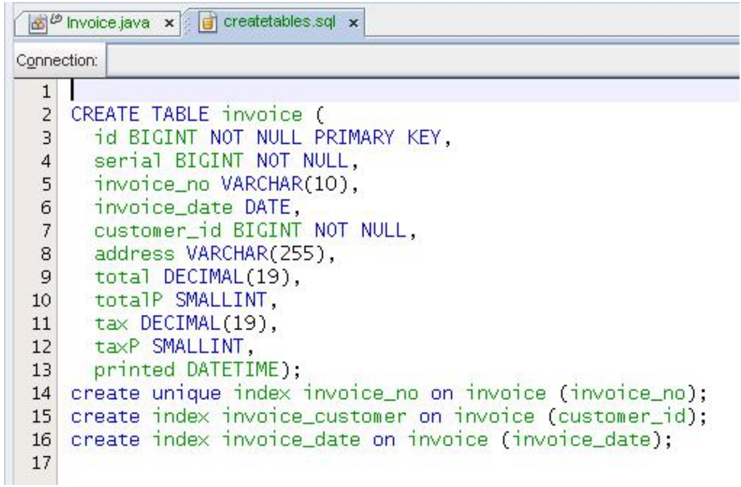
CS1320 Creating Modern Web and Mobile Applications

Lecture 19:

Databases II

Relational Database Schema

- The set of tables define the database
  - Table names
  - Set of attributes/fields for each table
  - Each attribute is typed
  - Tables can have a PRIMARY KEY
  - Tables can have permissions
  - Tables can have constraints (e.g. NOT NULL, FOREIGN KEY)
- Schema can contain other information
  - Indices on the tables
  - Virtual tables (VIEWS)



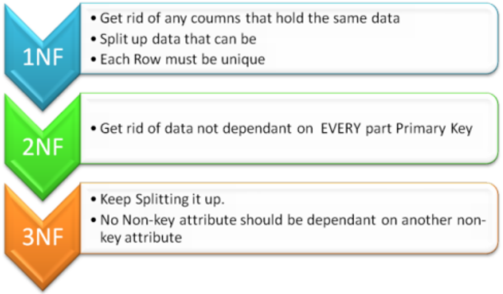
```

CREATE TABLE invoice (
  id BIGINT NOT NULL PRIMARY KEY,
  serial BIGINT NOT NULL,
  invoice_no VARCHAR(10),
  invoice_date DATE,
  customer_id BIGINT NOT NULL,
  address VARCHAR(255),
  total DECIMAL(19),
  totalP SMALLINT,
  tax DECIMAL(19),
  taxP SMALLINT,
  printed DATETIME);
create unique index invoice_no on invoice (invoice_no);
create index invoice_customer on invoice (customer_id);
create index invoice_date on invoice (invoice_date);

```

Rules for Table Organization

- How the tables are organized is important
- Based on the relationships among data items
  - If x => y (e.g. DiskId => Title, length,…) put in same table
  - If x is used multiple times, make it a separate table
    - Especially if x has multiple values or long values
  - If x can occur multiple times for y, create a separate table
    - Just for relating x and y
  - If table will be referenced by other tables, add an id field
    - As a key field



1NF
- Get rid of any coumns that hold the same data
- Split up data that can be
- Each Row must be unique

2NF
- Get rid of data not dependant on EVERY part Primary Key

3NF
- Keep Splitting it up.
- No Non-key attribute should be dependant on another non-key attribute

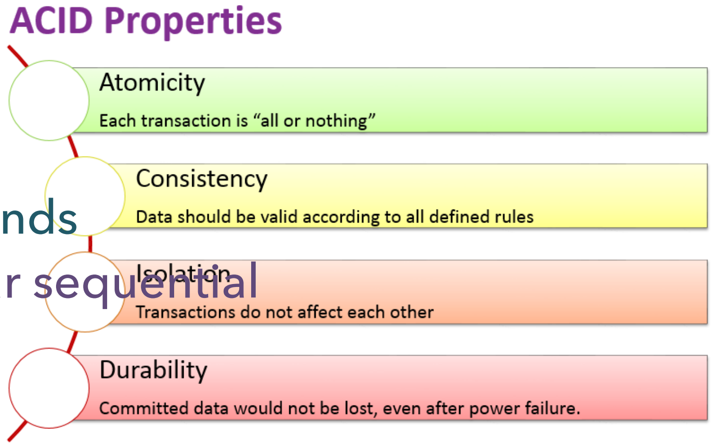
Database Manager Position

- In general this is a difficult problem
  - Need to understand semantics of the data
  - Need to understand how the data will evolve
- If you get it wrong, what happens?
  - With only queries, not much (performance)
  - If the data is updated, it might become inconsistent
    - Artist information for example
  - If the data relationships are updated, might not fit structure
    - Multiple artist CDs
- Database manager position
  - Handles setting up, changing, updating, etc. the database
  - Each project should choose a database manager for their database

"You're the database administrator, guide us in our hour of need."

Transactions

- Multiple operations might need to be done together
  - BEGIN TRANSACTION name
    - Operation
    - Operation
    - Operation
  - COMMIT TRANSACTION name
- Example: withdraw and deposit to transfer funds
- Database guarantees that transactions appear sequential
  - No intermediate changes by others
  - Operations all done or all not done (atomic)
  - ACID semantics
- Use any-db-transaction when using Node.JS



ACID Properties

Atomicity

Each transaction is "all or nothing"

Consistency

Data should be valid according to all defined rules

Isolation

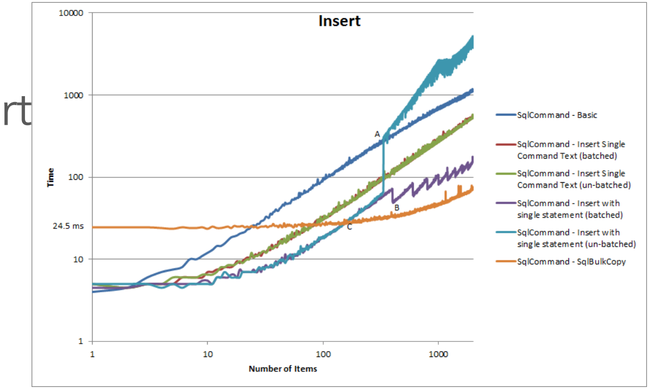
Transactions do not affect each other

Durability

Committed data would not be lost, even after power failure.

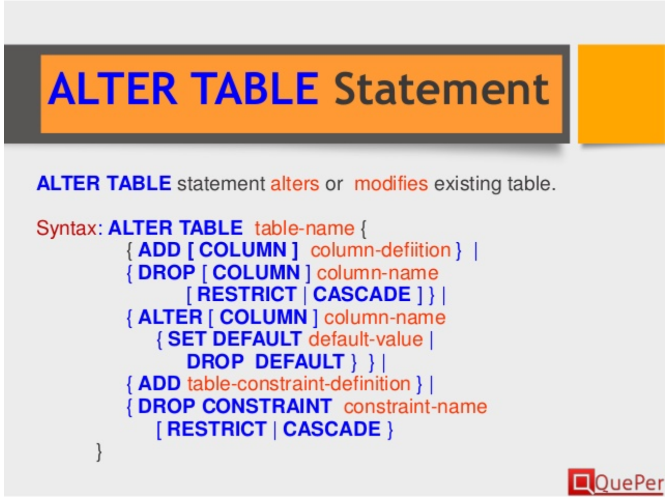
Batch Processing

- Large numbers of inserts (or updates) can be inefficient
  - But doing them all at once can be a lot faster
  - Can create BATCH updates
    - Like transactions (can use transactions for this)
    - Part of Statement interface for embedded SQL
  - Most databases provide a means for batch insert
    - Insert multiple tuples into a relation
    - Data comes from a file



Schema Updates

- Tables are going to change
  - Applications evolve over time
  - New data is needed, old data no longer needed
- Adding tables is easy
- Modifying tables is possible
  - ALTER TABLE ADD newfield int DEFAULT 0
  - Can remove fields, add constraints, add indices, …
  - Beware of constraints between tables
- This can be an expensive operation
  - And might require other computation (computing initial values)
- Can be tricky
  - Might involve dependencies, indices, constraints
  - Maintaining a test database and a production database
  - Create update scripts; test on test database before applying to production



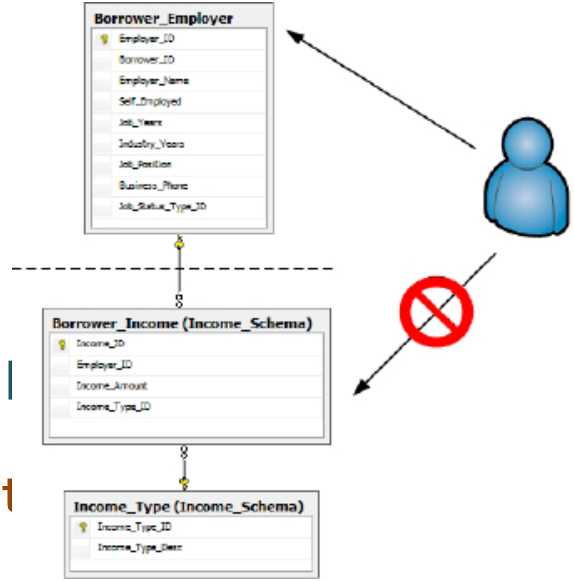
ALTER TABLE Statement

ALTER TABLE statement alters or modifies existing table.

```
Syntax: ALTER TABLE  table-name {
            { ADD [ COLUMN ]  column-defiition }  |
            { DROP [ COLUMN ] column-name
                  [ RESTRICT | CASCADE ] } |
            { ALTER [ COLUMN ] column-name
               { SET DEFAULT default-value |
                  DROP  DEFAULT }  } |
            { ADD table-constraint-definition } |
            { DROP CONSTRAINT  constraint-name
               [ RESTRICT | CASCADE }
        }
```

Permissions

- External database systems control access
  - Users (independent of system users)
  - Permissions (on a table-by-table basis)
- When you set up the database you can set u
  - With different permissions on different tables
- This can provide a safeguard in your applicat
  - Application normally runs as a limited access user
  - Administrative tasks run with more privileges
- Think about permissions when you set up your database
  - Create a one or more separate users for your application



Database Systems

- MYSQL, PostgreSQL, Oracle, SqlServer, DB2, Derby, Sqlite3…
  - All support core SQL (more or less)
- Many support extensions
  - Regular expressions, transitive closure
  - XML-based querying (using XQUERY)
  - User defined functions and operators
  - Cluster or distributed operation
  - Additional data and query types (e.g. geo data & queries)
- What do you want to rely on
  - Different environments might have different availabilities
  - Migrating can be problematic (esp. with extensions)
  - System updates change things as well
  - Some allow sharing, others do not

Which SQL Database System to Use

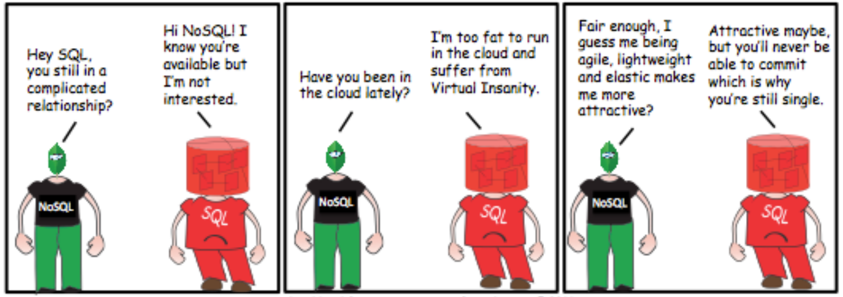
- Sqlite3, Derby
  - **Only** for experimentation, small-scale applications
  - Embedded use only (single access point)
  - Using these in your project is not a good idea
- MySQL, PostgreSQL
  - Most common for web applications
    - Extensions to both to handle scaling are available
  - Roughly equivalent
- Db2, SqlServer, Oracle
  - For larger databases, better support

MySQL or PostgreSQL?

The world's most **popular** open source database

The world's most **advanced** open source database

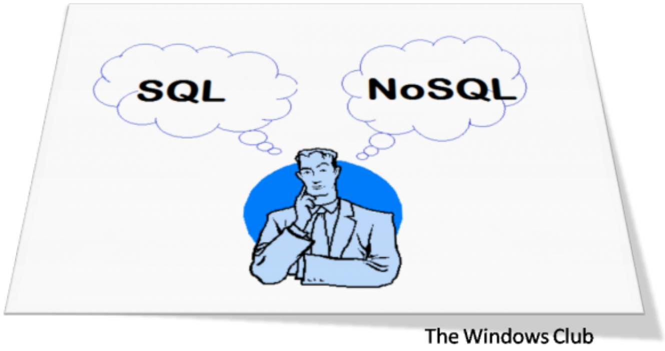
SQL Databases


Hey SQL, you still in a complicated relationship?
Hi NoSQL! I know you're available but I'm not interested.
NoSQL
SQL
Have you been in the cloud lately?
I'm too fat to run in the cloud and suffer from Virtual Insanity.
NoSQL
SQL
Fair enough, I guess me being agile, lightweight and elastic makes me more attractive?
Attractive maybe, but you'll never be able to commit which is why you're still single.
NoSQL
SQL

App Man Adventures - www.appdynamics.com © 2011

- Pros
  - ACID semantics (transactions, guaranteed consistency)
  - Queries handled by the database engine (little code)
  - Portable
  - Data consistency through the schema
  - Supports complex indices, triggers, constraints
- Cons
  - Replication of the database is difficult
  - Doesn’t scale that well to huge databases
  - Not good at storing unstructured documents (blobs)
  - Complex Database Manager issues
  - Changing schema requires thought and effort
- Can we do better for web applications?

NOSQL Databases



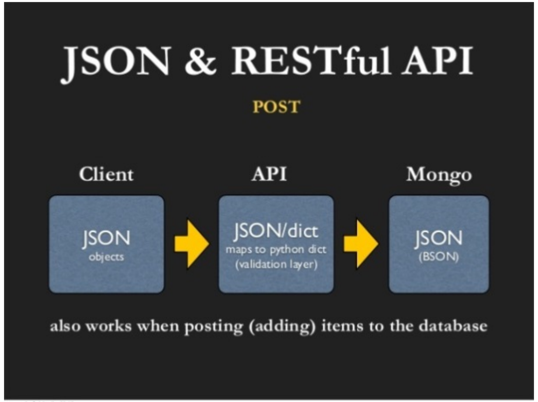
The Windows Club

- No SQL
  - Essentially key-value stores
    - Arbitrary values, single key; look up by key
    - Indexed by key
  - Eventual consistency (no transactions)
- Not-Only SQL
  - Trend to make these more like SQL databases
  - NewSQL databases combine SQL queries with No-SQL-style stores
- Index on particular fields
  - Field value -> { key } is what is in index
- Simple queries
  - Given a set of field values, find corresponding data values
  - Done my intersecting key sets
  - Take resultant key set and allow iteration over values

MongoDB : a simple (sophisticated) NoSQL

- Store JSON objects (i.e. JavaScript)

```
db.cds.insert( {
  diskid : '2a04b804',
  title  : 'Body & Soul',
  artist : 'SPEED',
  length : 1210,
  genre  : 'jpop',
  year   : 1996,
  tracks : [
        { name   : 'Body & Soul',artist : 'SPEED',length : 302,number : 0,offset : 187 },
        { name   : 'I Remember',artist : 'SPEED',length : 278,number : 1,offset : 22892 },
        { name   : 'Body & Soul (hand Bag Mix)',artist : 'SPEED',length : 323,number : 2,offset : 43777 },
        { name   : 'Body & Soul (instrumental)',artist : 'SPEED',length : 302,number : 3,offset : 68070 }
] } );
```



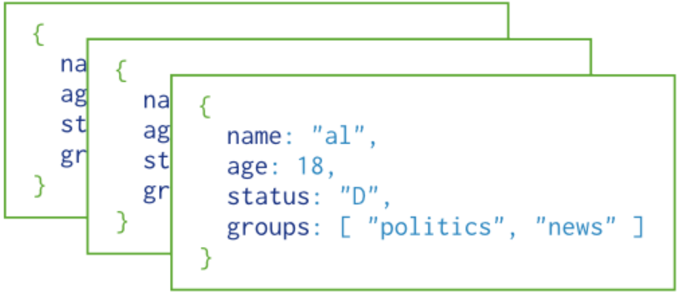
JSON & RESTful API

POST

also works when posting (adding) items to the database

mercoledì 4 luglio 2012

MongoDB Collections



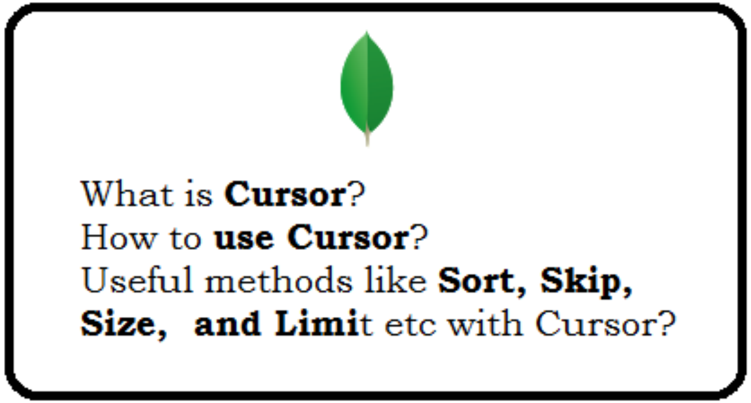
```
{
  name: "al",
  age: 18,
  status: "D",
  groups: [ "politics", "news" ]
}
```

Collection

- Basic Storage is a collection
  - Set of json objects
  - Each has a unique identifier (generated by the system)
  - Fast lookup based on identifier
- Access methods on a collection
  - db.<collection>.find(query, projection)
  - Query: { field : { $eq : <value> }, … } ($eq,$gt, …)
    - Field can be "field" or "field.subfield"
    - Can do most queries over a single collection
  - Projection: { field : 0|1, …. }
    - 1 => include field, 0 => exclude field

MongoDB Find

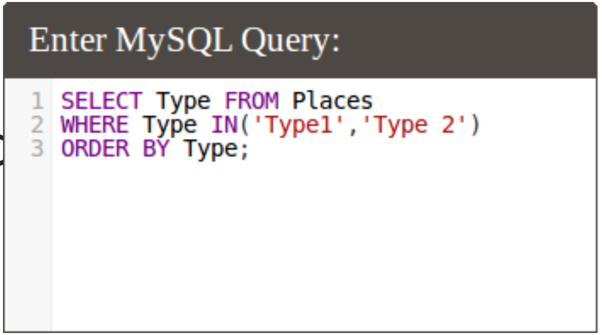
- Yields an cursor (iterator) over th
- Can Apply other methods to thi
  - Sort to sort the results
  - Limit to limit the number of items returned
  - Skip to skip k results
  - Filtering to further restrict the results
  - toArray to get the results as an array



What is **Cursor**?
How to **use Cursor**?
Useful methods like **Sort, Skip, Size, and Limi**t etc with Cursor?

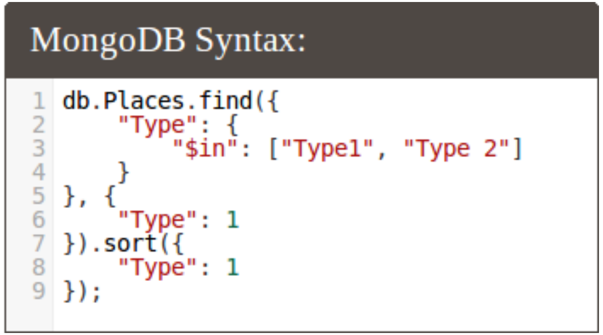
MongoDB Example Queries

- db.cds.count()
- db.cds.find({ artist : 'Taylor Swift' }).sort({year: 1}).limit(10)
- db.cds.find({ "tracks.artist" : 'Jaques Brel', "tracks.name" : "Ma
- db.co



Enter MySQL Query:

```
SELECT Type FROM Places
WHERE Type IN('Type1','Type 2')
ORDER BY Type;
```

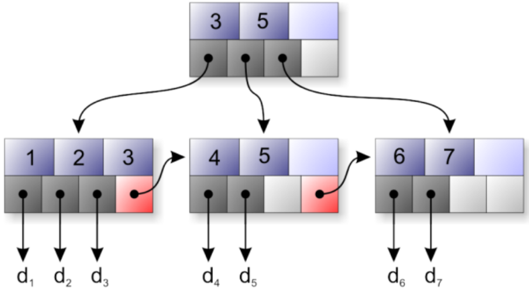


MongoDB Syntax:

```
db.Places.find({
    "Type": {
        "$in": ["Type1", "Type 2"]
    }
}, {
    "Type": 1
}).sort({
    "Type": 1
});
```

MongoDB Indices

- db.<collection>.createIndex(keys,options)
  - Keys: { field : 1 } (for ascending)
- Mongo supports text indexing
  - Keys: { field : ‘text’ }
  - Keys: { “$**” : “text” }
  - Limit of one text index per collection
- Text query
  - db.cds.find($text : { $search: ‘Paris’ })
  - Words: if multiple, then OR of the words (ala Google, w/ ranking)
  - Can use ‘ ”word1” “word2” “word phrase” ’ as well (and)

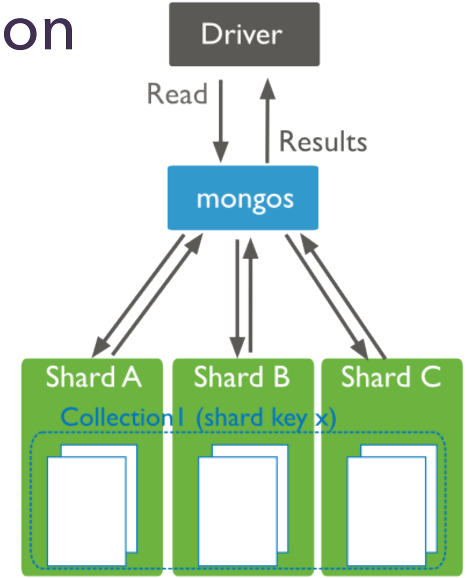


MongoDB CRUD

- db.<collection>.insert ({ … })
  - Newer versions support insertMany
- db.<collection>.update({ query}, { update })
  - Query: similar to a find query
  - Update: { $set : { field : 'value' } }
- db.<collection>.remove({ query })
- Bulk operations via db.<collection>.bulkWrite([…])

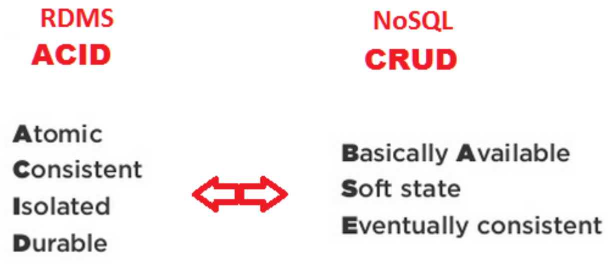
Distributed Mongo

- Mongo makes it easy to split up a collection
  - Into distinct portions called shards
  - Based on mongo-generated key or on user key
    - Can shard database to match different servers
  - Makes very large databases possible
  - Makes very large databases faster
- Also does distributed processing
  - Handles scaling to very large databases



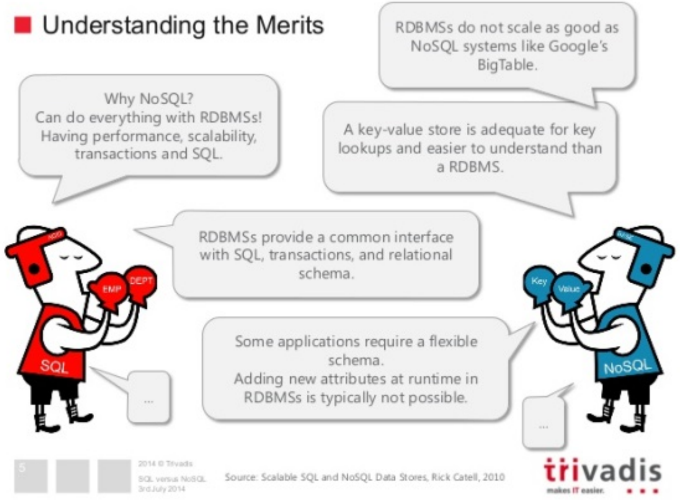
Mongo Transactions

- No ACID guarantees
  - Clients may see writes before they are committed
  - If a query updates multiple documents, might read some updates and not others
- BASE instead
  - High availability
  - Eventual consistency
- Newer versions provide limited tran



NOSQL Databases

- Pros
  - Can store arbitrary objects
  - Easy to set up and use (little management)
  - Easy to change
  - Scale nicely
- Cons
  - No transactional guarantees
  - Data consistency is up to the user
  - Complex (multiple collection) queries need to be coded explicitly
  - More memory intensive



Understanding the Merits

RDBMSs do not scale as good as NoSQL systems like Google's BigTable.

Why NoSQL? Can do everything with RDBMSs! Having performance, scalability, transactions and SQL.

A key-value store is adequate for key lookups and easier to understand than a RDBMS.

RDBMSs provide a common interface with SQL, transactions, and relational schema.

Some applications require a flexible schema. Adding new attributes at runtime in RDBMSs is typically not possible.

...

...

5

2014 © Trivadis

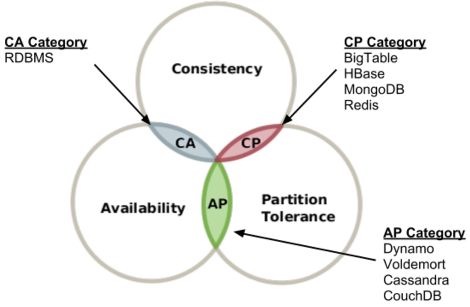
SQL versus NoSQL 3rd July 2014

Source: Scalable SQL and NoSQL Data Stores, Rick Catell, 2010

trivadis makes IT easier.

What Type of Database to Use

- How important are ACID guarantees
  - Newer versions of Mongo provide some of this
- How important is data consistency
  - SQL provides constraints and triggers
- How important is scalability
  - Newer versions of SQL database provide sharding, etc.
- How varied and complex are the queries
  - Complex, unanticipated NoSQL queries require code
- How complex is the data and how often does the schema change
  - Complex data is much easier in NoSQL
- Might want a combination of databases
  - Some for complex data
  - Some for users, authentication, etc. where ACID is useful



Next Time

- Database LAB
  - Do the pre labs
  - Look up Six Degrees of Kevin Bacon

Question

Suppose we want to go beyond SQL databases. NoSQL databases are a response to the needs of modern web applications. Which is not a characteristic of such databases?

A. They can be accessed directly from HTML5 using database extensions

B. They support eventual consistency rather than immediate consistency

C. They typically use a simple or specialized query language.

D. They provide the ability to easily shard or replicate data

E. They provide specialized implementations suitable for particular types of applications.

Elevator Talks

- Three Minute Sales Pitch (includes 30 seconds for questions)
- Convince a someone to invest in your project
- Important points
  - What is the problem
  - What is your solution
  - Why is this important
  - Why should the audience be interested

The Art of the Elevator Pitch

Making an Impression in 30 Seconds or Less

Donato Diorio | Founder & CEO
Broadlook Technologies | ddiorio@broadlook.com