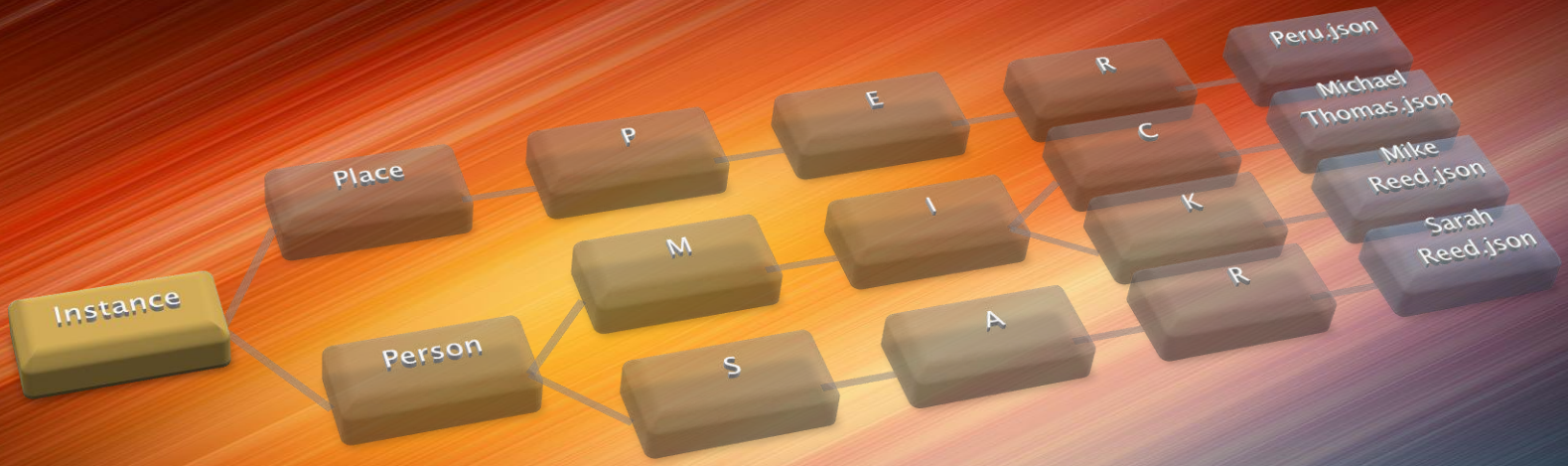
ZarahDB

- **Instance** = Root Folder for the scattered database (Zarah is Hebrew for Scattered).
- **Table** = Sub-folder. In standard DB terms the table can be said to contain any number of rows, each row is accessed by its key.
- **Key** = A single JSON file that contains all the data associated with the key, which is stored as columns, each column having a value.
- **Column** = The name for the name/value pair to be stored or retrieved.
- **Value** = A string value, which can be anything from null, a number, a set of typed characters or even a full JSON blob.

Instance



Instance
The root folder for the scattered database.

The instance contains tables, which are just file folders named the same as the table. The instance is also where the instance backup it stored. When a backup is done, a .Zip file will exist in the instance folder and will contain all the folders and files located in any folders that exist within the instance.

Table

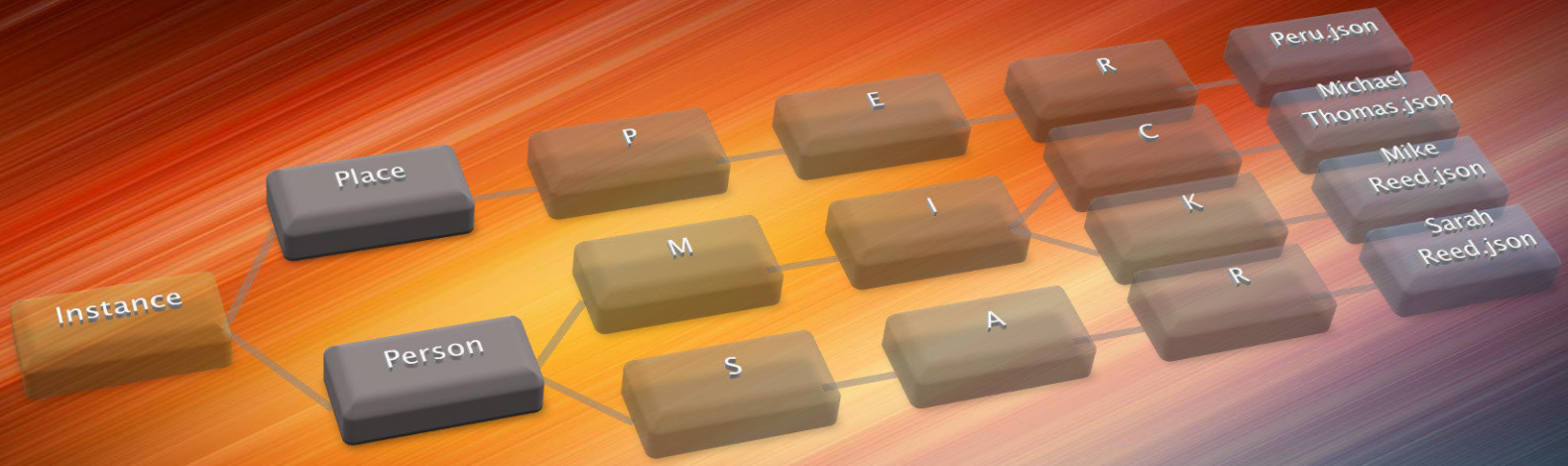
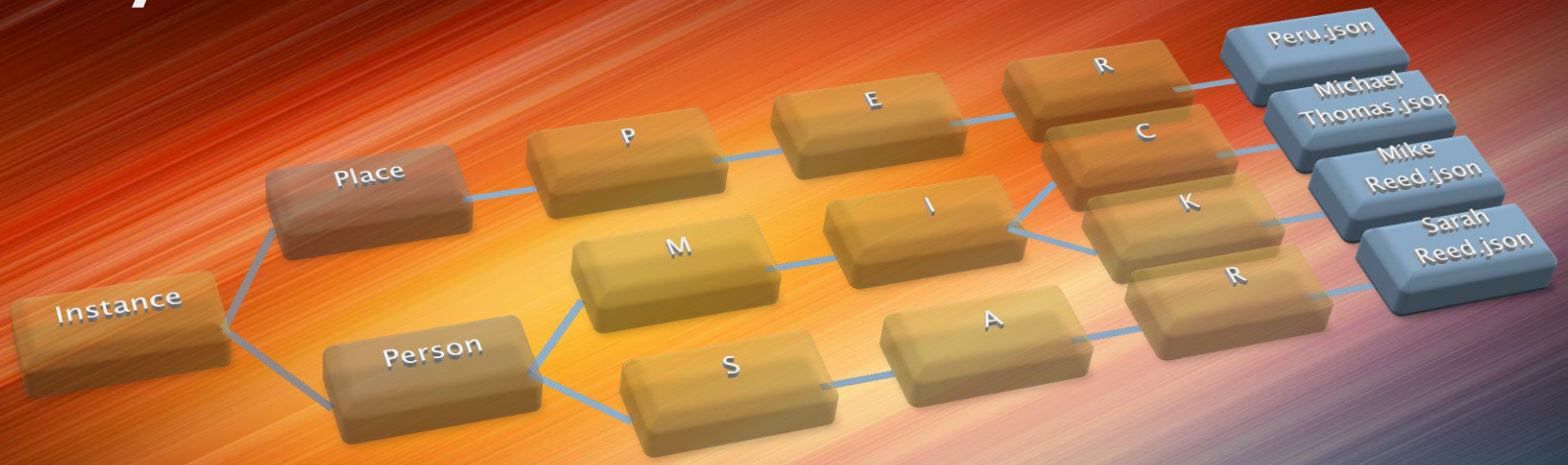


Table
Tables are just file folders with the same name as the table.

The tables help to separate different types of data. Each table has a single data element that functions as the key to access the data contained in the table. The name of the table typically defines type of data contained in the table. Every key stored in the table should be something that directly relates to the name of the table.

Key

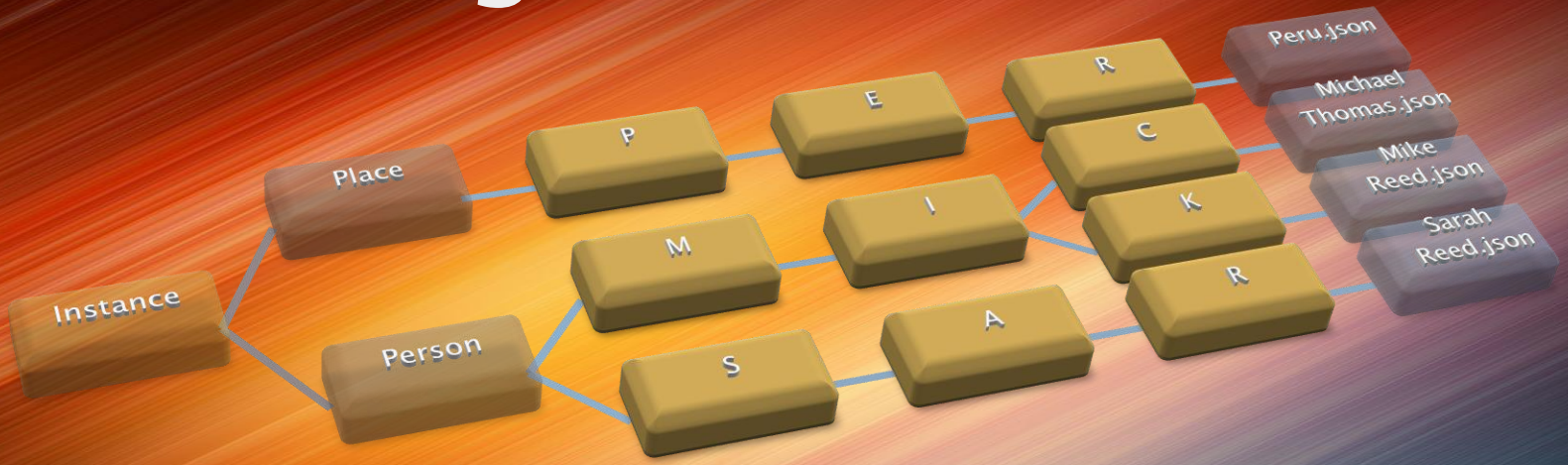


Key

- Keys are stored in JSON files. The name of each file matches the key, or is very close to the key.

In the above example there are two tables, “Place” and “Person”. Place has one key for “Peru” and Person has three keys, “Mike Reed”, “Michael Thomas” and “Sarah Reed”.

Scattering



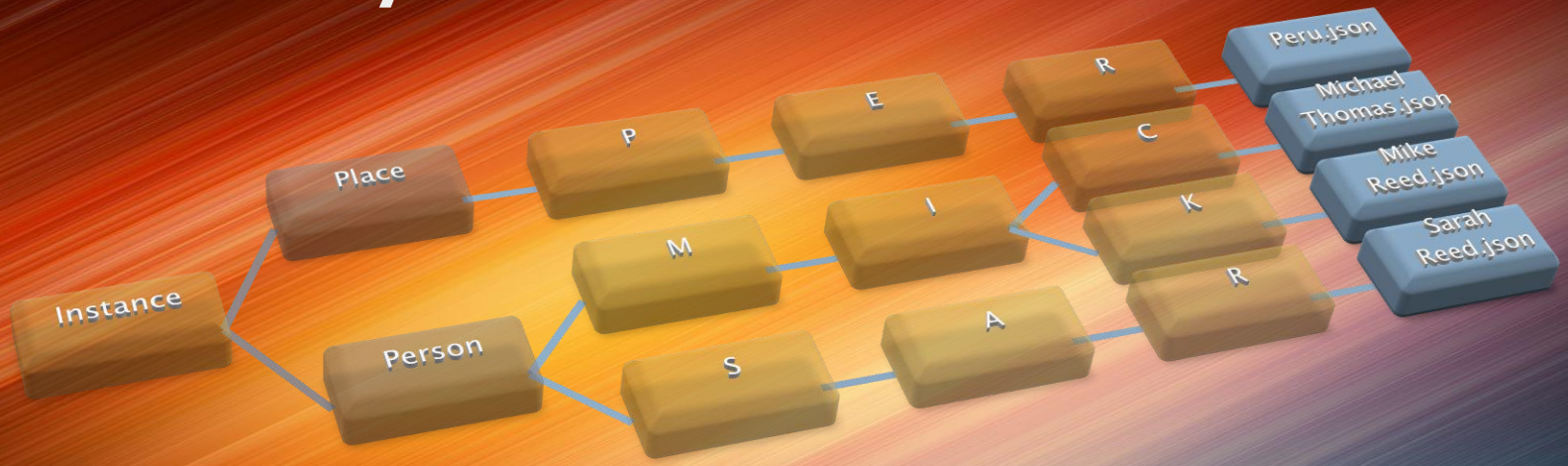
Scattering

- Keys are scattered so that only a few reside in any one folder.

Depending on how much data is expected to be stored, and how unique the keys will be, different levels of scattering can be set. The example above, we use a "MaxDepth" of three. The default depth for a ZarahDB is five levels of scattering.

Note that each file resides under folders that spell out the key. "Peru" for instance is stored under the "P", "E", "R" folders. Both "Michael Thomas" and "Mike Reed" are stored under the "M" and then the "I" folders, but because they are spelled differently, "Mike Reed" is next under the "K" folder, while "Michael Thomas" is stored in "C" as it's final folder.

The Key Files



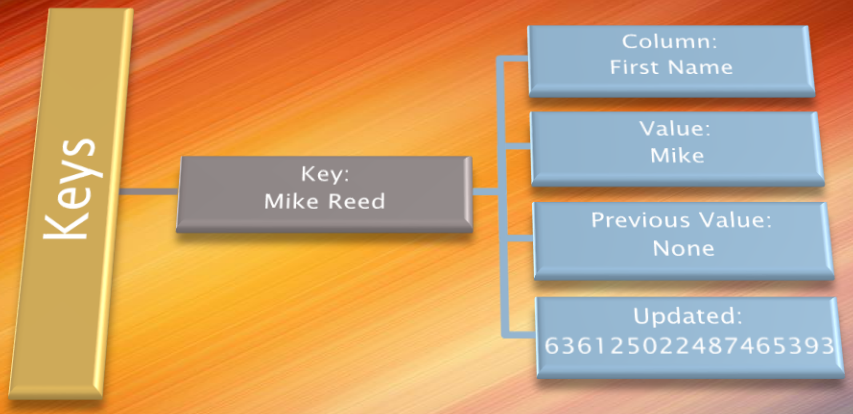
Key Files

- Each key file contains JSON. Typically a single key is stored in each key file.

Anything we store about Peru will be stored in the “Peru.json” file. Anything about Mike Reed will be stored in the “Mike Reed.json” file.

Let’s look inside a key file and see how the data is stored.

Mike Reed.json



Each JSON File

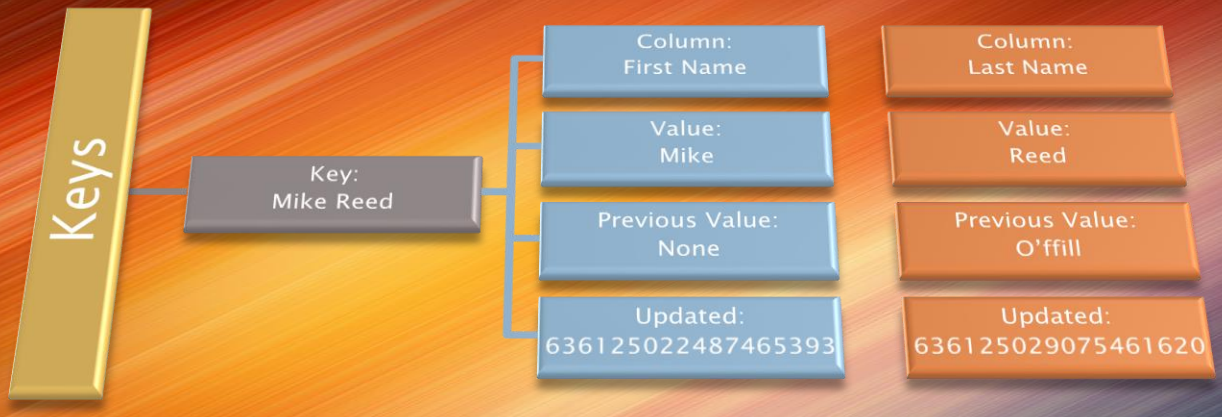
- Each key file can contain any number of keys, but typically only a single key.

Under each key we find columns. In the above example we are only storing one column. The column is named “First Name” and the value is “Mike”.

If we ever update this value, the previous value gets stored before the value is updated, so we always know if a value has changed and we know the value prior to the last change.

We also know when the value was last updated. This big number equals the number of ticks that have elapsed since 12:00:00 midnight, January 1, 0001. There are 10,000 ticks in a millisecond and 1000 milliseconds per second. So a tick is a really small unit of time! 10,000,000 ticks per second!!!

Mike Reed.json



Additional Columns

- Any reasonable number of columns can be stored.

As columns are added, they simply get added to the structure. There are no reasonable limits to the number of columns that can be added to a single key.

Values are just blob fields. They can contain simple data or full documents, images or any type of data. Typically the limit is 8K of data for each value, but this can be extended to any reasonable size.

Note that in the example above, Mike has changed his name, his prior last name was "O'ffill".

- Transactions
- Indexes
- Map Reduce
- Scaling
- Impendence Mismatch
- Object Database
- Disk Subsystems (Physical Scattering – Sharding)
- NoSQL
- Microsoft .NET based – C#
- Non-relational
- Open-source
- Cluster Friendly
- Schema-less
- Data Models
  - Key/Value = Here is a key, get me the value.
  - Document = Stores a full document per key. Here is a Key, get me a JSON blob.
    - Return/Write/Update portions of a document
    - No Schema – just return all the values for a key
    - Implicit Schema
- Aggregate-Oriented Database
- Column-Family Database
- Best used when there is a unique key
- Worst used when there are two or more keys, or aggregates are needed.
- Consistency
- ACID – BASE – Transactions – Atomic Updates
- Aggregates are transaction boundaries, so you get atomic updates automatically.
- Write/Write conflict – use the updated ticks as part of the update. Offline-lock. Version stamp (time).
- AWS EFS – Availability, resilience, cross zone updates.
- Consistency vs. Availability
- Safety vs. Liveness
- Two drivers
  - Large amounts of data – more than one server can handle… use scale out vs. clusters.
  - Easier Development, read and write at the domain level, serialize a full complex object and read and write the json directly.
- Not a graph database
- Agile Analytics – Ken Collier
- Polygot Persistence – Storing to multiple DBs for different data, or for slicing the same data multiple ways.
- Eventual Consistency
- Nothing to install. The OS is the DB! – But there is a NuGet package making it easy to add to a .Net project, a Zip file to install a web service, and a .MSI package to install a stand alone ZarahDB service which exposes a Swagger UI and RestFul API.