

Pisa University  
  
  
TASK 1  
LARGE-SCALE AND MULTI-STRUCTURED DATABASES

**feasibility study for a Key-Value traslation of the database model**  
academic year 2019-2020  
  
  
  
  
stefano petrocchi, Andrea Tubak, Francesco Ronchieri, Alsessandro Madonna

Summary

[Study 3](#_Toc23606189)

[Application 3](#_Toc23606190)

[Description 3](#_Toc23606191)

[Load 3](#_Toc23606192)

[Key-Value Model 3](#_Toc23606193)

[Caracteristics 3](#_Toc23606194)

[Disadvantages 3](#_Toc23606195)

[Suitable Data Types 4](#_Toc23606196)

[LevelDB 4](#_Toc23606197)

[Entities Analisys 4](#_Toc23606198)

[User 5](#_Toc23606199)

[Film 5](#_Toc23606200)

[Cinema 5](#_Toc23606201)

[Comment 5](#_Toc23606202)

[Projection 5](#_Toc23606203)

[Conclusion 6](#_Toc23606204)

[Implementation 6](#_Toc23606205)

# Study

## Application

### Description

### Load

An estimate of the factor that contribute to the load of the application:

* Pisa is a medium sized Italian city and has a population of nearly 100’000 inhabitants considering foreign students.  
  Is estimated that about 50% of peoples in Italy goes to cinema once a year.
* Are estimated 1’000 daily active users.
* In Pisa are located seven cinemas, one of them multi-room.
* About 50 new film are estimated to be projected every month.
* Cinemas are open every day from 3:00 pm to 1:00 am, are estimated 5 projection for cinema room every day.

These are the numbers to consider for the future scalability of the application.

## Key-Value Model

### Caracteristics

A ***key****-****value*** store is a database which uses an array of *keys* where each key is associated with only one *value* in a collection. The *key-value* stores usually do not have query languages as in ***relational databases*** to retrieve data. They only provide some simple operations such as get, put and delete.

*Key-value* databases work in a very different way from *[relational databases](https://en.wikipedia.org/wiki/Relational_database" \o "Relational database)*. *RDBs* pre-define the data structure in the database as a series of tables containing fields with well-defined [data types](https://en.wikipedia.org/wiki/Data_type). Exposing the data types to the database program allows it to apply a number of optimizations. In contrast, *key-value* systems treat the data as a single opaque collection, which may have different fields for every record. This offers considerable flexibility and more closely follows modern concepts like [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming). Because optional values are not represented by placeholders or input parameters, as in most *RDBs*, *key-value* databases often use far less [memory](https://en.wikipedia.org/wiki/Computer_memory) to store the same database, which can lead to large performance gains in certain workloads.

### Disadvantages

Disadvantages of *key-value* models:

* The only queries that are efficient are simple, one-row-at-a-time queries.
* Is not really a data model, indeed there is no association between attributes that form an entity.
* Is hard to use most ordinary SQL operations such as JOIN or GROUP BY.
* There isn’t a possibility to choose an appropriate SQL data type for the value.
* There isn’t a possibility to use many SQL constraints such as FOREIGN KEY or NOT NULL.
* Pivoting the result to resemble a conventional result (one column per attribute) is a complex and expensive query.
* Is needed to write a lot more application code to reassemble collections of key/value pairs into objects.

### Suitable Data Types

Type of data suitable for storing in a key-value pair data-store:

* **Data of indeterminate form.** For example, each HTML page  is different. Defining a schema for such page is complex. Since relational database expects schema, it is not possible  to store the HTML page. *Key-value* data-store does not require a schema and it would be a best fit for such data.
* **Data of huge size and quantity.**  *RDBs* are optimized for small rows that  supports table fitting within a single server. In contrary Key-value data-stores supports storing large objects with huge quantity spread across multiple servers.
* **Unrelated Data.** Application might require storing unrelated data which is not suitable to be stored in  *RDBs*. Since Key-value pair data stores are not based on relations, storing such unrelated data is supported.

### LevelDB

***LevelDB*** is a *key-value* store built by Google. It can support an ordered mapping from string keys to string values. The core storage architecture of LevelDB is a log-structured merge tree (LSM), which is a write-optimized B-tree variant. It is optimized for large sequential writes as opposed to small random writes.

*Key-value* store supports the mapping from the key to the corresponding value. In SSTable the layout of key and value is managed as adjacent string sequence.

## Entities Analisys

Considering the database entities schema of the application:



### User

***User***entity represent a client or a not-technical administrator of the application. The information contained in this entity regards app login information, registry information and privilege level information. That information is small size data that can be naturally organized in a tabular form.

*User* is related with *Comment*, *Film* and *Cinema* entities.

An estimate of the typical transactions load for this entity in a day is:

* *Very small* number of ***write*** transactions (profile creation, deletion or modification).
* A number of ***read*** transactions less then active users ( *500* logs in).
* A number of new ***favourite*** cinema and film similar to that of log in.

### Film

***Film*** entity represent a film and its description. The information contained in this entity can be of indeterminate form and dimension (i.e. for the description).

*Film* is related with *User*, *Comment* and *Projection* entities.

An estimate of the typical transactions load for this entity in a day is:

* *Very small* number of ***write*** transactions (small number of new films every month).
* A number of ***read*** transactions slightly greater than that of active users ( *> 1’000* ).

### Cinema

***Cinema*** entity represent a cinema and few general information. That data is small sized and can be naturally organized in a tabular form.

*Cinema* is related with *User*, *Comment* and *Projection* entities.

An estimate of the typical transactions load for this entity in a day is:

* *Insignificant* number of ***write*** transactions.
* A number of ***read*** transactions slightly greater than that of active users ( *> 1’000* ).

### Comment

***Comment*** entity represent a user comment. The information contained in this entity can be of indeterminate dimension.

Comment can be related to cinemas and films but forms of comment that are nested or related to other entities could also be implemented.

An estimate of the typical transactions load for this entity in a day is:

* A number of ***write*** transactions similar to that of log in (  *500* ).
* A number of ***read*** transactions largely greater than that of active users (*>>> 1’000* ).

### Projection

**Projection** entity represent the information of a single projection of a specific film in a specific cinema. That data is small sized and can be naturally organized in a tabular form.

*Projection* is related to *Film* and *Cinema* entities.

An estimate of the typical transactions load for this entity in a day is:

* *100* ***write*** transactions (five projection for a cinema room daily).
* A number of ***read*** transactions greater than that of active users ( *>> 1’000* ).

## Conclusion

### RDBs recommended Entities

From the entities analysis emerges that the write transactions for *User*, *Film* and *Cinema* entities are executed in a very small number, both in absolute value and comparing them with the number of readings, even foreseeing a strong expansion of the application. The number of read transactions scale in proportion to the number of users.

Furthermore, those entities are strongly related by the possibility of having favourite cinemas and films, this relation has to be updated when one of this entity changes.

*User* and *Cinema* contains small sized data and can be naturally organized in a tabular form, the description of a *Film* can theoretically be of indeterminate form and dimension.

According to this study *User*, *Film* and *Cinema* entities are preferably mouldable through a ***RDBs*** model because of their static structure and the reduced load that they entail in terms of transactions compared to the overall system. Only the description of a film could be manage using a *key-value* model because of the indeterminate form and size of that field.

### Key-Value recommended Entities

# Implementation

