

POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering



Master's Degree Thesis

Design of a data management system for value bet detection and soccer performance analysis

Supervisors

Prof. Daniele APILETTI

Dr. Fabrizio MACHELLA

Candidate

Francesco FOSCHI

December 2021

Abstract

Investing in the stock market has become incredibly popular in recent times. More noticeably a huge number of young, risk averse and inexperienced traders are now impacting the market with over 25 millions of options contract traded daily. The instability that this trend brought to the table gave rise to increased demand for uncorrelated asset classes - i.e. assets that are not influenced by the current market situation – such as collectibles and sports betting. In a certain sense the betting and trading world have a lot in common, whoever is able to process the information at their disposal better and faster is usually going to make a profit. A classic example of how technology is impacting the stock market is HFT or High Frequency Trading, a financial trading approach that requires dedicated software to perform the acquisition and liquidation of positions on the very short-term, sometimes seconds, allowing investors to leverage small profits on a huge number of transactions. Since the first attempts to apply predictive models to horse racing in the 1970s, which failed due to the lack of data, technology has made a huge step forward. In recent years some companies have specialized in collecting and selling sports data that is now more available than ever before. With the aid of artificial intelligence we can now analyze each event in a match and evaluate its effects on the final outcome. This approach is exploited by sports organizations at all levels. Managers know which players are the most valuable and which generate the higher fan engagement. Bookmakers have a tool to correctly evaluate the performance of each team to offer profitable odds, but so do investors. The ability to identify a betting scenario where the odds don't represent correctly the possible outcome opens up the possibility for a systematic betting approach that in the long run will yield a profit.

Even if betting has always been associated to gambling, with the aid of big data and artificial intelligence, one individual can now profit from mispriced wagers in the sports market in the same way that one could profit from trading mispriced options in the stock market. The necessity of extracting the most valuable information and metrics from the data that is at our disposal represents a valuable competitive advantage as it boasts our ability to find profitable investment opportunity.

In this thesis I present a complete data management pipeline that can efficiently

extract valuable information that are later going to be used to train AI models. The theoretical approaches to identify value betting scenarios have been already openly presented in the scientific literature, on the other side the technical details are mostly kept secret by companies who don't want to disclose their design choices. I hope that this work can represent a valuable starting point for future research in the betting and sports analytic world.

Acknowledgements

Solo ora che questo percorso volge al termine mi fermo a realizzare quanto in questi anni sia cambiato. Tuttavia non sarei mai stato capace di raggiungere questo traguardo da solo e vorrei ringraziare le persone che mi sono state vicine in questi anni.

In primis vorrei ringraziare i miei relatori Daniele Apiletti e Fabrizio Machella, che mi hanno aiutato nella stesura di questo elaborato. Grazie a mia madre, mio padre e mio fratello per essere sempre stati un porto sicuro. Grazie a Francesco per esserci sempre stato, anche nei momenti più bui. Grazie ai miei amici dello stagno per tutte le risate e a Lorenzo per aver sempre creduto nei nostri progetti. Un ringraziamento speciale ai miei mentori, Massimo e Wolfgang per tutti gli insegnamenti che mi hanno reso chi sono. Grazie a Giorgia, la migliore compagna che potessi avere accanto. In fine, grazie a Camilla e Olympia per il vostro amore incondizionato.

Table of Contents

List of Tables	VIII
List of Figures	IX
1 Introduction	1
1.1 How big data is impacting sports betting	1
1.2 Value bet detection	1
2 Modern Database technology	3
2.1 The relational database	3
2.2 The NoSQL paradigm	5
2.2.1 Transactional models: ACID vs BASE	6
2.2.2 Why a NoSQL database could be the right choice	8
2.2.3 Document-oriented database	8
2.2.4 Column-family database	9
2.2.5 Key-Value database	10
2.2.6 Graph database	10
2.3 CAP Theorem	11
3 Statistical approach for performance evaluation	13
3.1 xG - Expected goals	13
3.2 VAEP - Valuing Actions by Estimating Probabilities	15
3.3 xT - Expected threat	16
4 Data structure design and architecture design	19
4.1 The WyScout data representation	19
4.2 Definition of the requirements	20
4.3 The choice of the database	22
4.3.1 A small comparison with document-oriented and column-oriented DBMS	23
4.4 Design guidelines for graph-oriented DBMS	23

4.5	Proposed data structure and feature extraction	25
4.6	Querying the database	28
4.7	Architecture design	31
5	Results	32
5.1	Data exploration possibilities	32
5.2	Read and write performance	32
6	Conclusions	35
	Bibliography	36

List of Tables

4.1	The match data structure complete reference at https://apidocs.wyscout.com/	20
4.2	The event data structure complete reference at https://apidocs.wyscout.com/	21
5.1	The indexes used in the final design	34

List of Figures

2.1	A standard table representation in a relational database	4
2.2	An example of hierarchical database structure, to access the level 3 information a user must navigate the entire hierarchy	5
2.3	An example of document	8
2.4	Representation of a row oriented storage	9
2.5	Representation of a column oriented storage	9
2.6	Graphical representation of a column oriented data store	10
2.7	An example of key-value store. Note that the content of the value can be anything as data is completely unstructured	11
2.8	CAP theorem visualization https://www.researchgate.net/figure/Visualization-of-CAP-theorem_fig2_282679529	12
3.1	The above representation shows all the shooting attempts of Hakan Chalhanoglu, where larger circles correspond to higher values of xG https://theanalyst.com/na/2021/07/what-are-expected-goals-xg/	15]
3.2	The visualization for the transition matrix https://karun.in/blog/expected-threat.html	17
3.3	2d and 3d visualization of the expected threat https://karun.in/blog/expected-threat.html	18
4.1	An example of a graph database representation	24
4.2	Examples of two bad designs	24
4.3	An example of the fan out design	25
4.4	Match data structure	27
4.5	Caption	28
4.6	A representation of the final data structure. Note that not all the attributes are reported	29
4.7	Caption	31
5.1	A portion of the data schema as represented in the <i>Neo4j Browser User Interface</i>	33

5.2	Time required to load every game from our dataset	34
-----	---	----

Chapter 1

Introduction

This work shows the results of my research conducted during the first half of 2021 during an internship at Mercurius BI srl, an Italian company that develops artificial intelligence models to trade and capitalize on the sports betting market. For this project I oversaw designing and developing a new data extraction pipeline that could serve all their current needs but would also be able to adapt to different requirements that may arise in the future. This work hinges heavily on the graph-oriented database to replace their current architecture.

1.1 How big data is impacting sports betting

Since Billy Beane's first attempt of building a competitive baseball team in 2003 using evidenced-based decision methods, it was immediately clear that approach represented a revolution for the world of sports. In the last years the amount of high-quality data available drastically increased and data acquisition processes are becoming more sophisticated. Experts developed new patterns and metrics that can be extracted from the same data enhancing the informative power of the information at our disposal.

Nowadays companies that want to maintain a competitive advantage over their competitors need to keep pushing the accuracy of their predictive models to guarantee the lowest degree of risk possible. To achieve this a good data extraction pipeline and feature engineering process is paramount.

1.2 Value bet detection

Before discussing how data is exploited to find profitable bets one must first define the concept of value bet. A value bet is a bet that offers better chances of returning a gain than a loss, in other words, a bet that you are more probable to win than

loose. In general value betting is systematic approach that when properly performed should generate returns in the long run. The term value betting is used in many odds-based games such as Poker but in this case we are going to discuss value bets in the context of soccer. Bookmakers fix the odds of a team according to the probability that the team has of winning, the lower the odds, the higher the price. For example an odd of 2.5 means that the bookmaker will pay you 2.5 times the amount you bet, that is a net return of 1.5 times your initial investment. Starting from the odds, one can compute the probability of winning of a that is implied by the given odds themselves. To do so we simply perform the division:

$$\text{implied odds probability} = \frac{100}{\text{offered odds}}$$

For example an odd of 2.5 will imply a probability of $100/2.5 = 40\%$. However, for the betting market, the probability implied by the odds takes into account the real chance of winning of a team but it doesn't reflect it purely. This is due to the fact that bookmakers take into account other factors and fix the odds based on their business model. In the end, they also have to make a profit. Now let's assume that a given team is playing with odds of 3 to win, this would imply a winning probability of 33%. However, our artificial intelligence model tells us that indeed that team has a probability to win of 50%. Now we can use the following formula to compute the expected value of our bet.

$$\text{Expected Value} = \frac{(\text{Fair probability} - \text{Market probability})}{\text{Market probability}}$$

This leaves us with the non trivial task of finding a way to extract the winning chance of a team from its performance. As shown in chapter 3, many approaches have been presented to analyze the performance of a team and compute the fair probability of winning, however the theory behind value betting is always the same, find betting scenarios that have a positive expected value.

Chapter 2

Modern Database technology

When designing a data storage architecture the choice of the underlying database system is of vital importance. A classical approach to database involved the use of relational databases, but in recent year we've seen NoSQL databases conquering the world of big data and major tech companies. In this chapter are discussed the major aspects of this two different approaches, with their relative benefits.

2.1 The relational database

Since its first adoption between the late '70s and early '80s, relational databases have always been the commercial standard for storing data. Relational database exploit the concept of relation to build an efficient representation of the data, and represent this relations with tables.

In 1970, Edgar F. Codd published an important paper called "A Relational Model of Data for Large Shared Data Banks" claiming that the adoption of the mathematical notion of "relation" could solve some of the problems related to the database models that were popular at the time, mainly the network model and the hierarchical model. Both the network and hierarchical model consisted of tree or graph data structure to keep track of the links between data using pointers. To query the data that was stored on the bottom of the tree one had to traverse it from the top, however, this implied that program had to have knowledge of the internal organization of the data in order to proficiently navigate the tree. Codd saw a major weak point in this approach: if applications bore the responsibility of navigating data to find the needed bits of information, the program would immediately break if the underlying structure of data would change. The relational model was born in a quest to achieve what Codd defines as "the independence

Id	Name	Surname
001	John	Smith
002	Olivia	Brown

CostumerId	InsurancelId	Expiration Date
001	I09BC	31/12/2025
002	I65IJ	30/06/2024

Figure 2.1: A standard table representation in a relational database

of application programs and terminal activities from growth in data types and changes in data representation”.

Codd introduced the relational model that would be later refined in the following years. In his studies he described a complete theoretical system that would guarantee solid basis for his model, including: ALPHA, a SQL-like language that he created to query the database but also important notions of relational algebra and relational calculus. The term relation that we use in this context is a purely mathematical one. Given two sets S_i, S_j a relation is a collection of ordered pairs containing one object from each set. If the object $x \in S_1$ and $y \in S_2$, then the objects are said to be related if the ordered pair (x, y) is in the relation.

This model was characterized by two main factors, the first is that data would be organized in tables, and the second is that data would be normalized. When talking about data, the process of normalization refers to a set of rules that the data should have to guarantee homogeneity and Independence of the data. During the early '70s, Codd defined the first normal form and then expanded it to the second normal form. For data to be in second normal form it should respect the following rules:

- No attribute domain has a relation as attribute, that basically implies not having another table as an attribute, which is the condition for data to be in first normal form.

- Tables don't contain non-prime key that are functionally dependent on any subset of columns that constitute the candidate key.

We recall that a candidate key is a set of columns of minimal cardinality that uniquely identify a relation. All the elements that can be part of a candidate key are called prime keys.

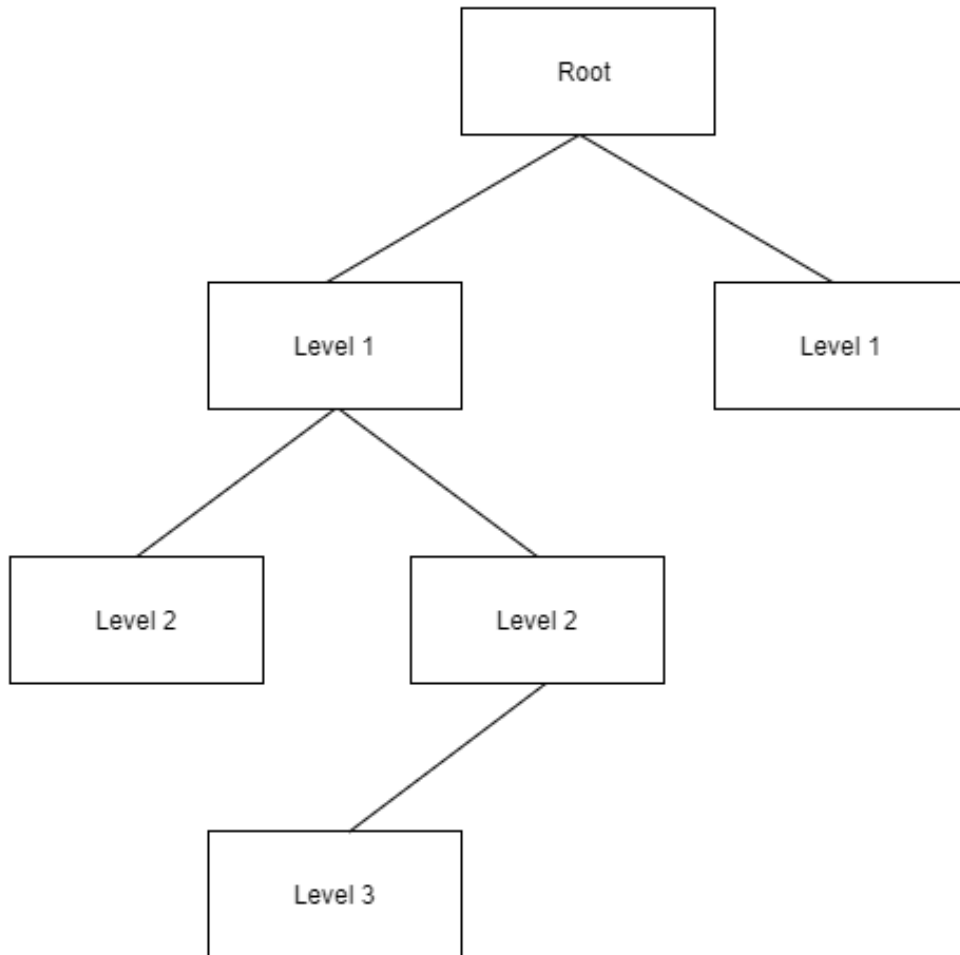


Figure 2.2: An example of hierarchical database structure, to access the level 3 information a user must navigate the entire hierarchy

2.2 The NoSQL paradigm

From its commercial success, in the '80s, relational database have been consistently used and as of today they still stand as one of the most used database systems.

However, some of the features that characterize RDBMs, proved to not work as well in certain use cases.

With the increase of the web traffic, organizations and businesses had to originally recur to vertical scaling - i.e. upgrading the machine that ran their software - to increase performance every time they incurred in a significant performance drop. However there is a technological limit to the performance that a single machine can provide, so big tech companies came to the point where vertical scaling wasn't a reliable way to increase the performance of their systems anymore. The only option remaining was to distribute the load among different machines and scale the performance horizontally - i.e. increasing the number of machines that are concurrently dedicated to a single task -, unfortunately SQL was simply not designed to work in this of environment. The table-oriented data structure requires join operation to merge data from different tables, indeed, to implement join operations efficiently a centralized system is required. Tech giants started to face the need to scale very quick and Amazon and Google presented respectively SimpleDB and BigTable in the mid 2000s to try and face the need for greater performance.

Since relational databases are often referred to as SQL systems, the word NoSQL started to appear on social network as a catchword to identify these new non-relational database systems. These new cluster based systems were developed with new objectives that were clearly different from those that guided the development of the original relational model:

- Faster performance
- Being able to operate efficiently on distributed systems
- Being able to rapidly adapt to fast paced changes of the requirements
- Having the ability to handle huge data stores of semi-structured data
- Being always available to minimize downtime

Quickly NoSQL became an answer to real world problem, and today is used by the biggest players in the tech industry to run their operations, which are the same individual that lead their development.

The NoSQL database can now be group in four different families: document-oriented based, column-family, key-value and graph database. We are going to dedicate a small section to each one of this databases

2.2.1 Transactional models: ACID vs BASE

One crucial aspect of database systems is the adopted transaction model. The transaction model establishes a set of rules in which determinates how a database

carries out basic data storing and management practices. Relational and NoSQL database use two very different approaches regarding the transactions management, they respectively use the ACID and BASE model.

The ACID paradigm is based on four characteristics that ensure the consistency of data when performing a transaction. These are:

- **Atomicity:** also known as the *all or none* principle. It implies that a transaction is either performed to its full extent or its rolled back.
- **Consistency:** before and after a transaction the state of the database remains valid. This means that inserting, removing or updating data doesn't violate the schema or any constraint or rule currently enforced on the environment.
- **Isolation:** Every transaction is executed without competing for any of the database resources with other transactions. The execution queue is moderated by the DMS that ensures the sequential execution of the queries.
- **Durability:** The results of a completed transaction will persist in the DBMS storage.

For this reason ACID compliant database a must in certain application that require the higher degree of accuracy when performing a transaction. A common example is a transfer of funds between two bank accounts. Financial application alike will use ACID databases almost exclusively. To enforce this level of security however it requires sophisticated management of resources which is a computationally demanding task for most use cases. Indeed the use of ACID transaction is pessimistic in most applications; the success of the NoSQL paradigm came partially from the relaxation of this requirements to achieve better scalability, performance and resilience.

To solve the problems that tech companies were facing - building high-available, cluster-based data stores - a more flexible approach to transaction management had to be adopted. The acronym BASE captures well this new set of driving principles:

- **Basically Available** The system is able to provide a response to almost every request.
- **Soft State** Since the system doesn't guarantee immediate consistency, the system may return data that is not updated.
- **Eventually consistent** Eventually the data on the cluster is going to be consistent on every partition

These two different approaches are a consequence of the CAP theorem, which is gonna be discussed in detail in a later section.

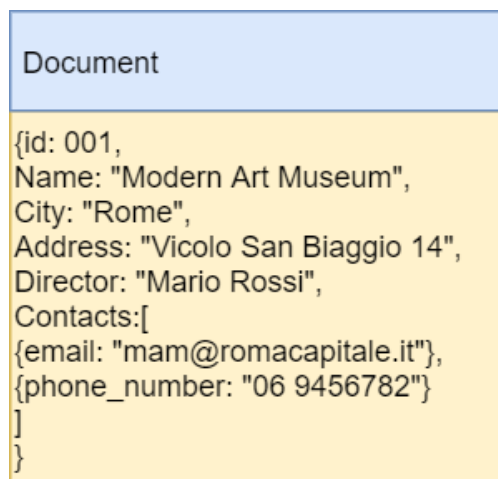
2.2.2 Why a NoSQL database could be the right choice

As discussed, NoSQL databases were born with high scalability, raw performance and availability in mind. Using one of these databases however could be justified even for smaller volumes of data if one needs to:

- Handle semi-structured or unstructured data.
- Query data with few and simple predetermined patterns.
- Your use case fits well in one of the main categories of NoSQL databases.

2.2.3 Document-oriented database

Document oriented databases are a very popular category of database system. They allow storing data in a way that is very close to the way a JSON file is organized. This system allows to store and retrieve data in a semi-structured way and are very capable in keeping together data that needs to be accessed simultaneously.



```
{id: 001,  
Name: "Modern Art Museum",  
City: "Rome",  
Address: "Vicolo San Biaggio 14",  
Director: "Mario Rossi",  
Contacts:[  
  {email: "mam@romacapitale.it"},  
  {phone_number: "06 9456782"}  
]  
}
```

Figure 2.3: An example of document

Documents themselves are very flexible objects: the fact that their data structure is self defined makes changing the data structure easy. Moreover, documents can be nested: this removes the need to perform any join operation between table, however the data schema must be query oriented - i.e. to store the information we need in an efficient way, we need prior knowledge of the required information. Their flexible schema, fast performances and high scalability, increased the popularity of document oriented data stores, especially in agile development environments.

2.2.4 Column-family database

Traditional relational databases were designed to read data from a single row at the time since every row represented a relation. For this reason the design choices focused on having a row store architecture. Rows of table would simply be stored on the disk sequentially.

StudentID	Exam	Grade
001	History	28
003	Ethics	30

001	History	28	003	Ethics	30
-----	---------	----	-----	--------	----

Figure 2.4: Representation of a row oriented storage

This approach proves very effective when writing a new row to the disk, as you simply append it at the end of the current data. This however proves inefficient when we want to read data, and even worst when we want to perform a join operation. This is due to the fact that usually we bring more data then we need to perform the join. This inefficiencies are not very noticeable when we work with small data stores but drastically impact our performance when we have billions of entries to go through.

Column oriented database was designed to work in data warehouses where retrieving and analyzing data is the core task. In this format, the values of the columns are stored close together.

StudentID	Exam	Grade
001	History	28
003	Ethics	30

001	003	History	Ethics	28	30
-----	-----	---------	--------	----	----

Figure 2.5: Representation of a column oriented storage

Each value in a column is then assigned to a key, which can though as the row index in the classical RDBMs. This key has the function of keeping track of which columnar values belong to the same entity.

This approach was first presented in a paper from 2005[1], where the advantages of such representation for mostly-read environment are presented. The main benefit of this storing approach is the reading performance, mainly due to the need to

access a smaller portion of data. Moreover columns can be saved on different disks and the search can be parallelized, making a dominant design in the cloud data warehouse market.

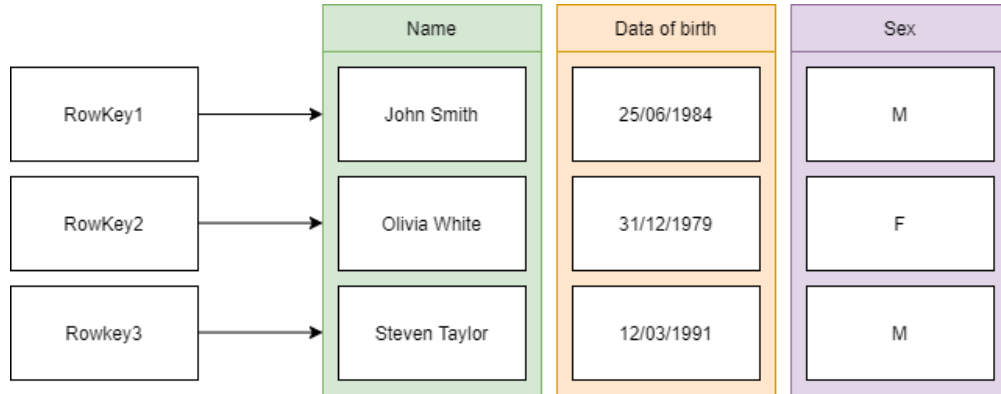


Figure 2.6: Graphical representation of a column oriented data store

2.2.5 Key-Value database

Key value databases are the simplest NoSQL stores. Their key-value entry is very close to the concept of dictionary: given a certain key the system is able to access the value stored in the database. This model allows for the most freedom in defining the data structure: the key is agnostic with respect to the content stored in the value part. This internal structure guarantees extreme speed and scalability.

2.2.6 Graph database

Both the key-value and document-oriented database are so called *aggregate oriented*, data is internally stored to face a particular criterion that is often to keep often used data together. Since the level of data aggregation is predetermined, it is easy to spread collections of data between different machines. Grouping data on a different aggregation level would require adding foreign keys to perform a join operation at the application level, which would become a very expensive task very fast. Graph database on the other hand are modeled on graph theory. They allow to easy model connections and links between data so that the user can explore data from any perspective. Graph data store are not the best at handling huge amounts of data, however they offer very deep analysis options. Relations in graph database are modeled as edges and entities are modeled as vertices. Both vertices and edges are semi structured data, so the use of this model makes for infrastructures that can quickly adapt to changes in the data representation, however deeper changes in the architecture require a start from scratch or a mass update.

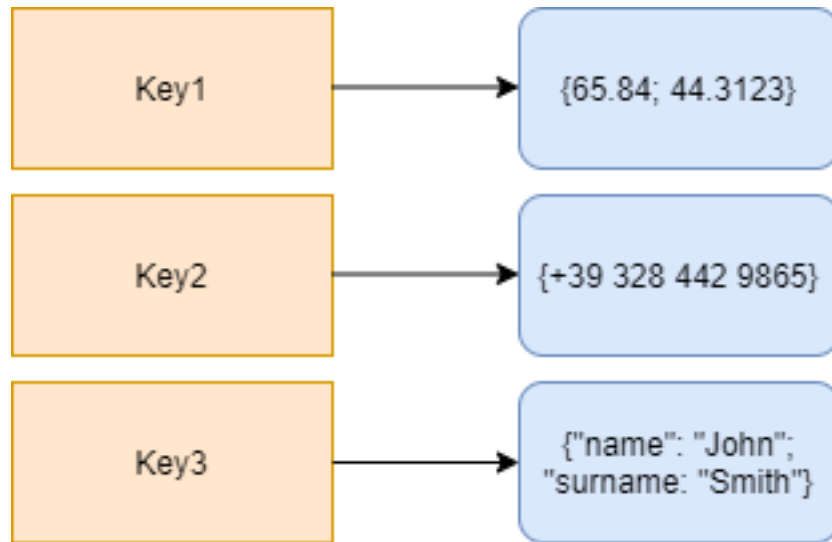


Figure 2.7: An example of key-value store. Note that the content of the value can be anything as data is completely unstructured

2.3 CAP Theorem

In computer theory, the CAP theorem states that a distributed database can only guarantee two out of the three following features:

- **Consistency:** all the machines that are part of the cluster always contain the same data
- **Availability:** the system is always able to provide a successful response, even if it's not the most updated one
- **Partition Tolerance:** the ability of the system to keep operating even after a network failure takes place.

From the theorem we can derive three main design archetype:

- **CA:** the system is consistent and available at all time. This is the approach followed by relational data stores. Since this system is not able to handle partitions we can't improve the resilience of the system to faults. In distributed environment however is generally not possible to forgo partitioning.
- **CP:** when a network error causes a disagreement between two node the system has to suspend the activity of one of the two nodes until consistency is restored.
- **AP:** we don't care about providing obsolete data if a partition occurs, when the partition is solved, consistency can be eventually restored.

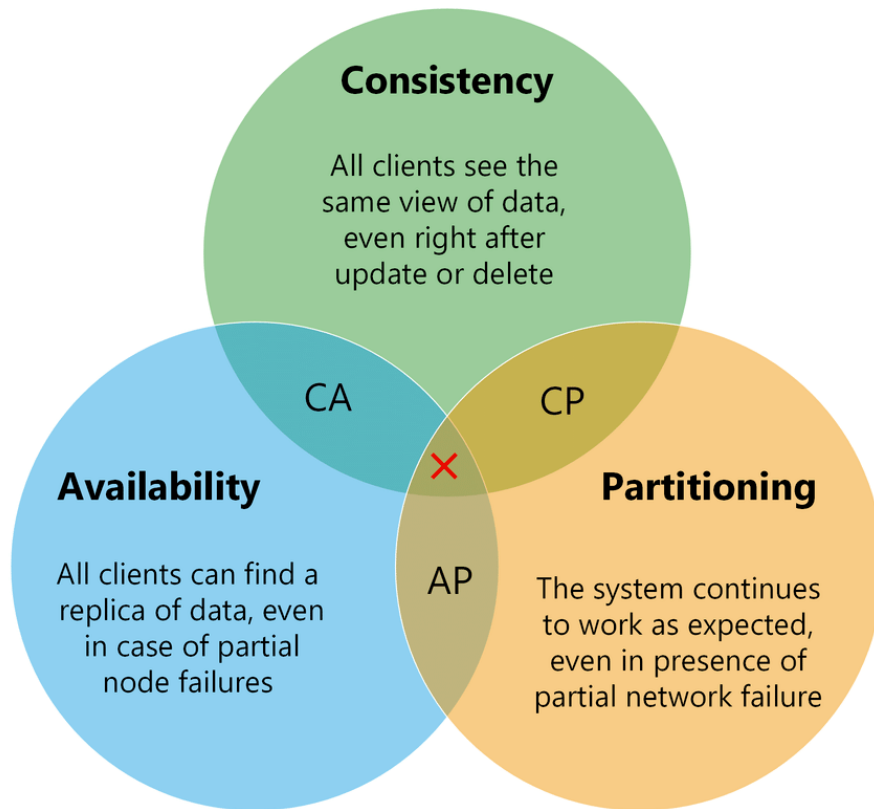


Figure 2.8: CAP theorem visualization https://www.researchgate.net/figure/Visualization-of-CAP-theorem_fig2_282679529

Considering the fact that partitioning the data is necessary to operate at scale, this theorem leaves us with the choice between delivering an available infrastructure or a consistent one. Of course this is true only if we want to guarantee all these properties at the same time, however these requirements can be handled by designers and managed to a finer level.

Chapter 3

Statistical approach for performance evaluation

In 1993, Vic Barnett and Sarah Hilditch conducted research on the effects that an artificial pitch could have on the home team performances. They concluded that:

“Quantitatively we find for the AP (artificial pitch) group about 0.15 more goals per home match than expected and, allowing for the lower than expected goals against in home matches, an excess goal difference (for home matches) of about 0.31 goals per home match. Over a season this yields about 3 more goals for, an improved goal difference of about 6 goals.”

For the first time a given team was associated with an expected performance. This is at the core of the techniques that are currently used to detect profitable betting scenarios. As explained in the section discussing value bets detection, evaluating the true probability of a team to win is paramount, much more so in a sport like soccer, where the goal is generally a rare event and results can be often decided by a single goal of difference. Since the inception of this approach many different frameworks were born, the most relevant being the *Expected Goal*, the *Expected Threat* and *Valuing Action by Estimating Probabilities* frameworks. These models address different aspects of the game. While the first focuses on the more relevant event in a soccer game, the shot, the other two maintain a broader view on the game, weighting the actions that in the long term will bring to a goal.

3.1 xG - Expected goals

A good definition for the expected goal metric could be the following:

“Expected Goals, or xG, are the number of goals a player or team should have scored when considering the number and type of chances they had in a match.”

Since the objective of soccer is to score as many goals as possible, when looking for a metric to evaluate the performance of a team or player, it is natural to be willing to evaluate the scoring chances that a team can generate. It is immediately clear to any soccer fan that shooting is not enough to make a goal. One may argue that if a team takes enough chances at it, eventually a shot will reach the back the net. However, it would be silly to think that taking a high number of shots, regardless of their quality, would significantly increase your scoring chances. In a certain sense the target measure of the xG model is exactly the intrinsic quality of a shot, the probability that a shot, taken in certain condition would end up turning in a goal. Figure 3.1 shows the position of all the shots that Hakan Chalhanoğlu took during the 2019-2020 Serie A season, with their relative xG value represented by the dimension of the circle. It's easy to see that shots that were taken far away from the goal represent less of a threat for the defending team.

At this point should be clear that not all scoring chances are created equal. Finding the most influential factors to compute expected goals have been object of discussion for years. The distance from the goal and the position of the shot have always been the most relevant factors for the evaluation of the quality of the shot, it's easy to see that a penalty shot and a header from the penalty mark don't have much in common other than the position of the shot and the likelihood of these two events ending up in a goal is very different. Many other casualties have been studied separately, for example the speed of the shot ([2]; [3]) and its accuracy [4]; [5];[6]). In this regard, it had a great relevance an article published in 2012 by Sam Green, a data scientist that at the time was working for Opta Sports, a sports analytics company. In the article Green analyzed the performance of the Premier League strikers and used the expected goal framework to give an objective interpretation to each players scoring record, moreover he explored the factors that made a shooting opportunity a good shooting opportunity and compared the results of the strikers that tried their luck with difficult shots against those who took fewer but higher quality opportunity.

Having said this, there is not a specific formula to calculate xG. Many models have come up since the introduction of xG and every user and organization has tried to improve the quality of its prediction by factoring in different elements for example what part of the body was used to take the shot or if the shot was taken during a counterattack and so on. Depending on the granularity of the data to which we apply the expected goal method it can be used to estimate the potential of a single event, player or an entire team, both offensive and defensive. The intrinsic value of the xG framework is clear nowadays. Some clubs, like the club FC Midtjylland won their first Danish league title using this method for the recruitment of players ([7]). Analysts and coaches gain a clear advantage in having

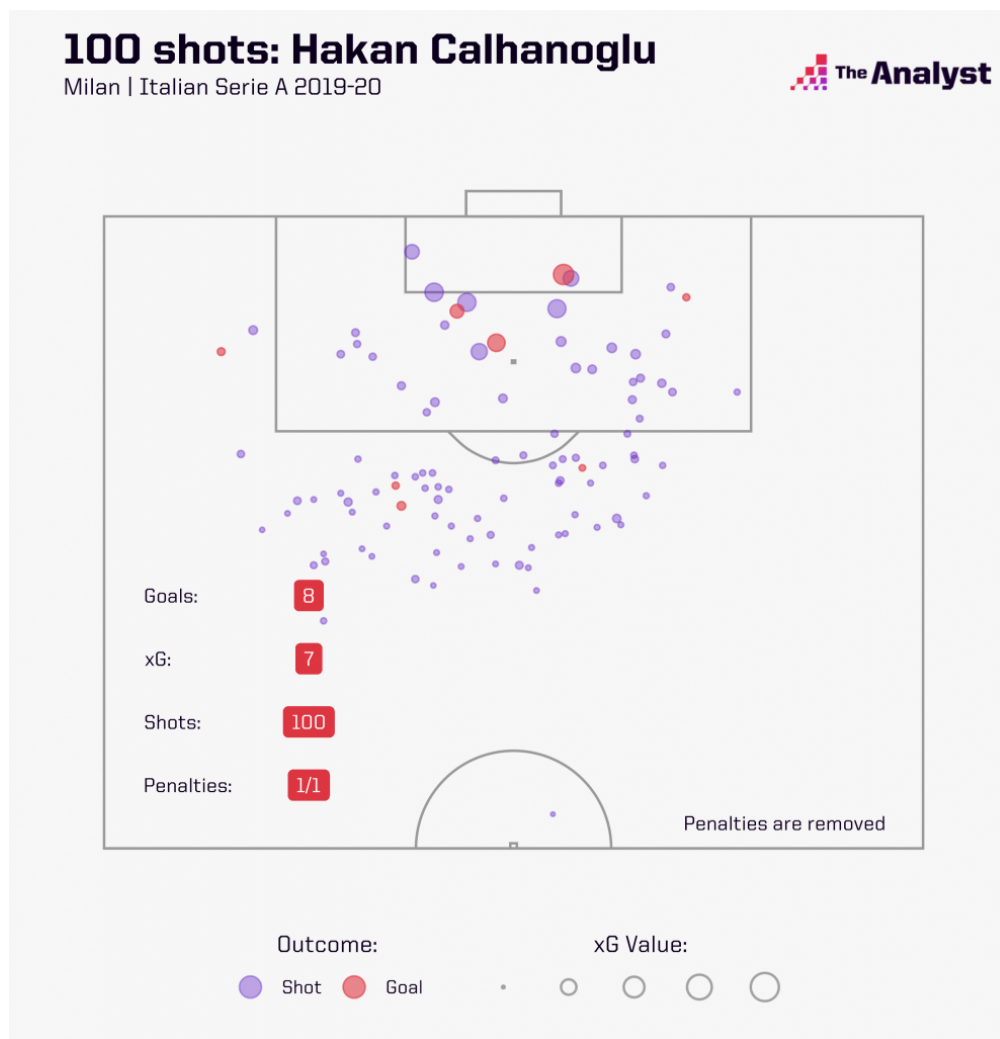


Figure 3.1: The above representation shows all the shooting attempts of Hakan Calhanoglu, where larger circles correspond to higher values of xG
<https://theanalyst.com/na/2021/07/what-are-expected-goals-xg/>

a statistical tool to evaluate players performance, and now it's getting used as a tool to exploit the betting market.

3.2 VAEP - Valuing Actions by Estimating Probabilities

In 2019 a PhD student at KU Leuven published a paper where he discusses the importance of evaluating a broader range of actions with respect to those taken into

account by the expected goal model. As highlighted before, shooting in soccer is a rare event so how does one evaluate the impact that a player is going to have on a game if he's directly accounted for scoring goals? The intuition was to create an evaluation framework that would include a broader range of actions. This objective poses different challenges. How would one evaluate a successful pass or a decisive defensive intervention? The VAEP model aims at answering these exact questions. To compute the VAEP value of an action we first need to define some notation. As an assumption, we divided the game in different game states $S = [a_1, a_2, \dots, a_n]$, and that for each of these states we can compute the chance that, given that game state, the home team will score or concede. Respectively $P_{scores}(S_i, h)$ and $P_{concedes}(S_i, h)$. Once we have stated these two simple quantities, evaluating an action simply means: how much will an action affect any of these two probabilities? The formula for the VAEP value of an action can be computed as follows:

$$V(a_i, x) = \Delta P_{scores}(a_i, x) + (-\Delta P_{concedes}(a_i, x)) \quad (3.1)$$

Please note that the VAEP value is simply the contribution of an action in increasing the scoring probability of a team plus the contribution in lowering the chances of conceding a goal. In this formula we denoted a given action as a_i and as x the team of the player that performed the actions. Using VAEP it's easy to extract an evaluation metric for player performances, we just have to sum the value of all the actions he performs and average it over the window of time that we want to compute.

$$rating(p) = \frac{1}{m} \sum_{a_i \in A_p^T} V(a_i) \quad (3.2)$$

Using this formula we can compute ratings for players with different granularity, the more natural being the VAEP for 90 minutes, but also analyzing the performance of a player over a season could be helpful.

3.3 xT - Expected threat

Expected threat is another metric that has gained popularity in recent years. Just like the VAEP it tries to evaluate a broader range of actions besides shots. To do so the expected threat metric analyzes the possessions of each team. The argument here is that every valuable scoring occasion is built up by each team with a series of passes and dribbles that will eventually move the ball in a position where shooting is easier. When using this method, the pitch is divided into $M \times N$ zones. Each of these zones is characterized by three different probabilities:

1. **Move probability:** the probability that will move the ball from a specific zone to another. The information about the destination of the pass or the dribble is stored into a transition matrix.

2. **Shoot probability:** the probability that a player will shoot in a given position
3. **Goal probability:** how often a shot from a certain position is converted into a goal

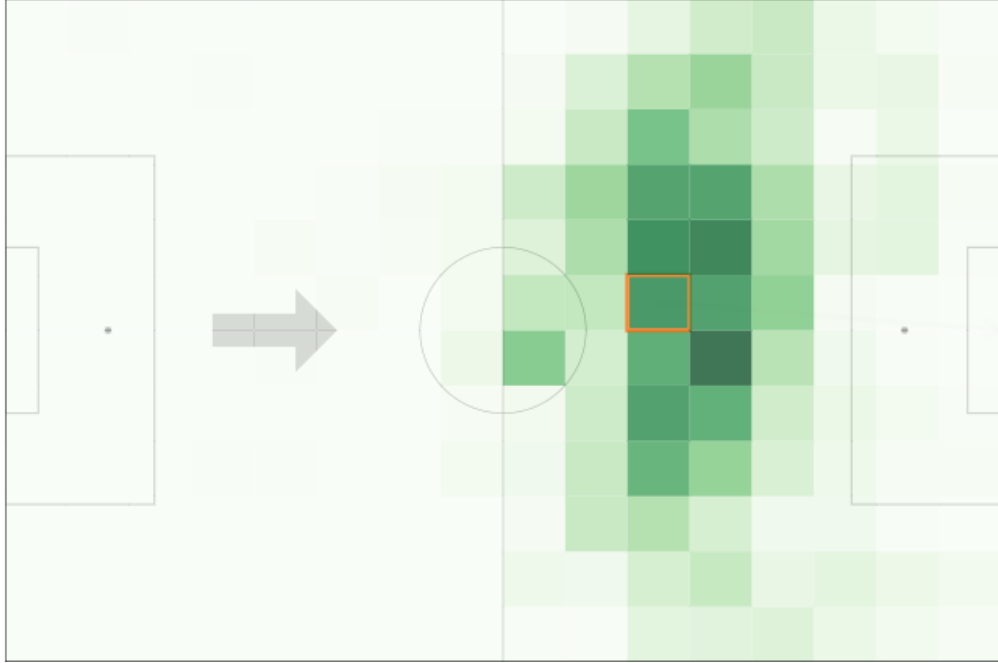


Figure 3.2: The visualization for the transition matrix
<https://karun.in/blog/expected-threat.html>

Starting from this quantities we can compute the expected threat value iteratively for each zone. The formula to compute the xT is:

$$V(x, y) = P_s(x, y) + \sum_{\forall(z, w)} T_{(x, y) \rightarrow (z, w)} \times V_{(z, w)} \quad (3.3)$$

where (x, y) is the current position and (z, w) is any other position in the pitch. From the formula you can see that expected threat is recursively defined. To compute it we initially start by setting the value of $V(x, y)$ to zero for each zone. Then we start computing $V(x, y)$ for each zone for the first time. At this moment only the contribution due to the probability of scoring has positive value, which makes it the same as an xG model. The next iteration will give us the probability of scoring after a pass in the same possession and so on. This notion makes the expected threat model very easy to interpret, as it basically gives us the chance to score a goal in the next n actions, where n is the number of iterations that were computed. With this notion we can evaluate the performance of a player by summing the positive difference in xT that his actions generate.

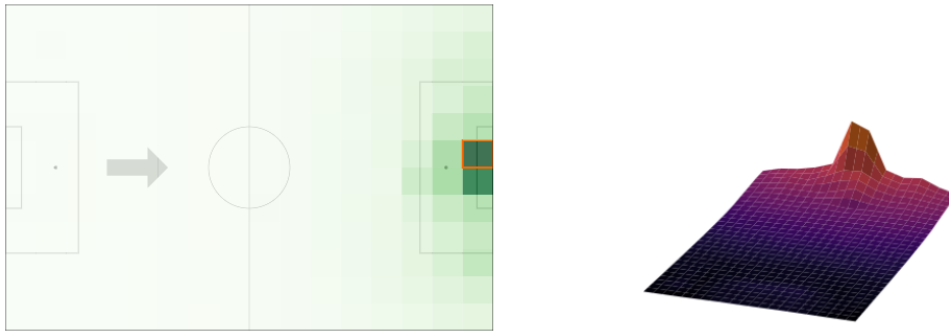


Figure 3.3: 2d and 3d visualization of the expected threat <https://karun.in/blog/expected-threat.html>

Chapter 4

Data structure design and architecture design

The design of a data structure and feature extraction pipeline that is able to manage sport related data is not an easy task. The main sources of data that can be used for scientific performance evaluation are optical tracking data and event stream data. Optical tracking data is used by sampling images at high frequencies and then extract the location of the players and the ball with the aid of a dedicated software. Event stream instead is the list of all the events that happened in a game with their position and time. For my research I have been working with data provided by WyScout which is of the second type. In this chapter I will discuss the implementation details of the design I proposed to efficiently exploit this kind of data.

The use of event stream data conditions heavily our design choice and poses different challenges. In general, not all provider adhere to the same data representation, moreover an update of the provider API may break the code as backwards compatibility is not guaranteed. For this reason companies have to tailor their software to work with a single provider, most of the time creating a lock-in effect. Indeed this project was designed to specifically work with the WyScout API version 2.

4.1 The WyScout data representation

Every event is reported in a JSON file that contains various information, divided in two main parts: the information on the match and the list of the events. Table 4.1 and 4.2 respectively report the data structure of the provided data.

This organization of the documents allow us to quickly process the data once we find the correspondence between the information we are interested in and the

Attribute	Description
WyId	Unique match Id
Label	A string that reports the teams name and final result e.g. Chelsea - Arsenal,2-1
Date	Local date and time for the match
DateUTC	UTC date and time for the match
Status	Possible status for the match e.g. cancelled, played, postponed etc.
Duration	Regular, Extra time or Penalties
Winner	Id of the winning team
CompetitionId	Id of the competition
SeasonId	Id for the season
RoundId	In competitions with rounds, is a unique Id that defines the round
Game week	In competitions spanning over various weeks, it's the week of the match, 0 otherwise
Teams Data	nested dictionaries containing data for the team: coach, formation, lineup etc.
Venue	String with the name of the venue
Referees	An array containing the unique Id of the referees for the match and their role
HasDataAvailable	A boolean value with the availability of the information for the match

Table 4.1: The match data structure
complete reference at <https://apidocs.wyscout.com/>

relative tags and IDs.

4.2 Definition of the requirements

As stated in the previous chapter the main objective of this data extraction is to predict the true chances of winning of every team. For this purpose the metric that Mercurius chose to adopt is the expected goal - xG. The list of parameters

Attribute	Description
Id	a unique event identifier
EventId	An Id that describes the type of event, i.e. shot, pass, duel etc.
PlayerId	Unique identifier of the player performing the action
TeamId	Unique identifier of the player's team
MatchId	Unique identifier of the match
Matchperiod	1H - first half 2H - second half E1 - first extra time E2 - second extra time P - penalties time
EventSec	Seconds elapsed since the start of the match
EventName	A string with the explicit type of event i.e. shot, pass, duel, etc.
SubEventId	An Id that describes a sub category of events e.g. head pass
SubEventName	A string with the explicit subtype of event i.e. head pass, hand pass etc.
Positions	X and Y coordinates of the starting and ending position of the event
Tags	An array of tags to describe specific situations e.g. the goal tag is added to shot that score a goal

Table 4.2: The event data structure complete reference at <https://apidocs.wyscout.com/>

that are considered for the computation of this metric is extremely long, so for the purpose of this thesis we are going to consider only to the following ones:

- The **angle** of the shot with respect to the center of the goal
- If the shot is a **penalty**
- If the shot is happening after an attempt of **block** by the goalie

- If the shot follows a **key pass**
- If the shot is labeled as an **opportunity**
- If the shot is performed during a **counter attack**
- If the shot is the consequence of a **dangerous ball lost**
- If the shot is preceded by a **corner**
- The **game state**
- If the player is shooting with its **natural foot**

Hence the main duty of this new database would be to easily extract the data needed to compute the xG value for each shot. The second task was to implement a new set of queries that involved dealing with sequences of events like the following ones:

1. Find all the uninterrupted sequences of actions that end with a shot.
2. Find, for each game of a given team in a season, the average number of sequences that start in a defensive sector of the pitch and end in an offensive zone.

Moreover, these queries should be easily accomplished on different levels of granularity: we need to compute the xG value both for a single player and entire teams, over the course of a single game, a set of games or an entire season.

As the choice of a NO-SQL database system is heavily query oriented, these queries highlight the need of finding a convenient way to analyze series of events. Regarding the system requirements, since the total amount of raw data is relatively small, approximately 36 giga-bytes, we don't need to partition our data store, for this reason we can achieve perfect local consistency. The system would be run on a server hosted on the Amazon Web Services platform, and should be able to get automatically updated every time WyScout releases data about recently played games.

4.3 The choice of the database

Given the requested queries, the choice of the database was a very important task. When dealing with the computation of the xG value for each shot, almost every DBMS could be used to efficiently solve the issue, since all we need to do is to extract a given set of properties of each shot. However it was also necessary to find

a database that could efficiently store sequences of events and query the patterns inside the database. For this reason a graph oriented database was chosen, in particular I settled for Neo4j. The main reasons for choosing a graph database are the following:

1. The logical model of the data can be reflected in the physical model, making very easy to design the data store.
2. The graph-oriented environment allows us to exploit the DBMS capabilities to find the patterns that we are looking for, without recurring to complex aggregation phases.
3. Neo4j is a reliable system with a large community and support, so it's a good choice for mission critical application.

With graph-oriented database we can express otherwise complex relations in a relatively simple way: for example we could generate a node for a season and then link to that node all the games that took place in that season. In the next section we are going to analyze how the data has been modeled.

4.3.1 A small comparison with document-oriented and column-oriented DBMS

During the research phase three different DBMS have been tested, one for each family. MongoDB, Cassandra and Neo4j where the candidates for the document-oriented, column-oriented and graph-oriented DBMS. As mentioned before, when analyzing only the xG value for every shot, every category of DBMS accomplished the task very easily. MongoDB in particular can basically store the entire match document without needing any preprocessing, making the write operations to the data store very fast. Similarly Cassandra can do the same by uploading massive Json files collections at once. However once we moved to testing the queries concerning sequences of events it became clear that a graph-oriented DBMS was the best fit. Verifying a condition for a sequence trough a document or a tabular database required a preaggregation phase computed explicitly via software, then the data on sequences could be stored in a document or in a table. Neo4j instead allows us to query for specific Node-Link-Node patterns, and more importantly, can handle variable length path analysis with ease.

4.4 Design guidelines for graph-oriented DBMS

A graph database consists of nodes, which represents entities, and edges, which represent relationships between nodes.

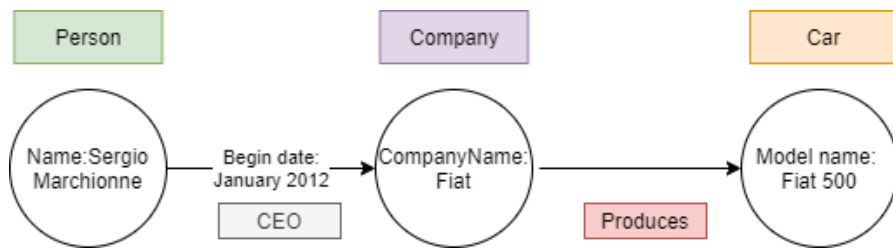


Figure 4.1: An example of a graph database representation

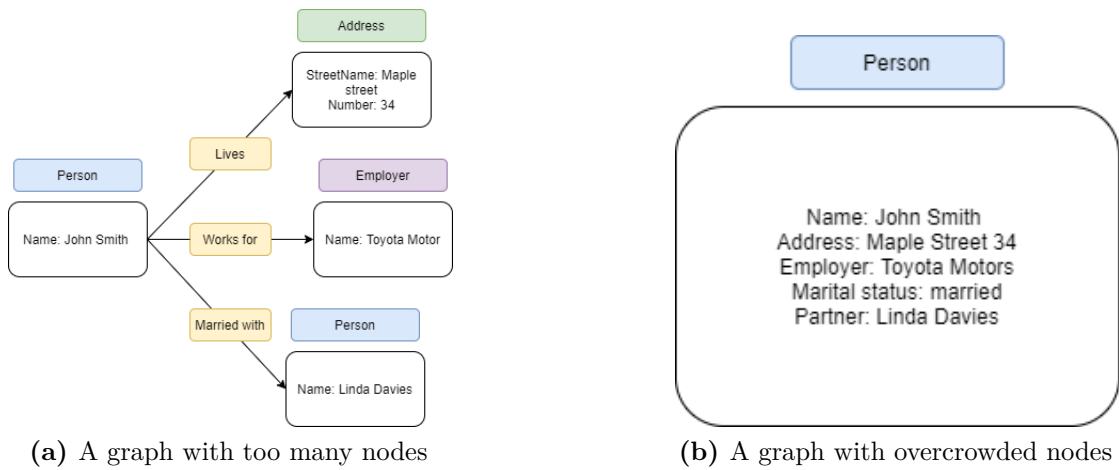


Figure 4.2: Examples of two bad designs

Both nodes and links are characterized by one or more labels and both can store additional informations. This representation is extremely powerful but also extremely easy to understand. Links can represent a wide variety of concepts and/or predicates which makes graph-oriented DBMS very flexible tools. When designing the data structure however one should be aware of some basic concepts. The first is to avoid extreme data fragmentation or concentration. Since data is unstructured one could have very few nodes with a high number of attributes or a very high number of nodes with very few attributes. Of course both of these approaches are generally wrong, graph should model significant entities as nodes and keep the marginal information as attributes.

Another pattern that should be avoided, especially when the the number of nodes in the graph increases drastically is the super node pattern. A super node is a node that has a very high number of connections to other nodes. This pattern can severely impact both read and write performance, but can be solved using fan out, a solution that was implemented even in my proposed design.

In figure 4.3 its shown a classic example of the fan out technique. It would be

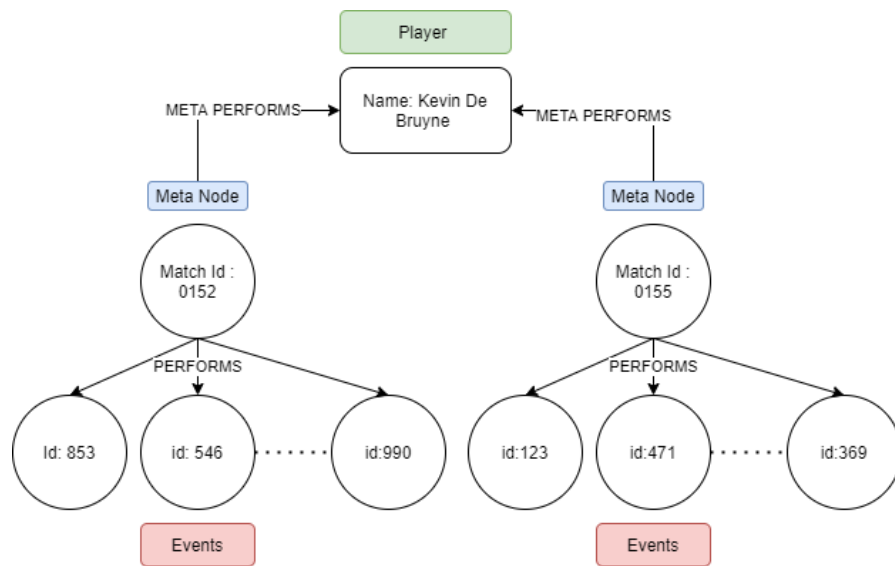


Figure 4.3: An example of the fan out design

nice to keep track of all the actions a player performed during his career, however a player performs a huge number of actions every match. Hence every node player would quickly degenerate in a super node. To solve this we create meta nodes that group the actions by the game in which they were performed. Even when there's not a natural way to group entities in a convenient way is important to keep the number of edges connected to a node relatively small.

4.5 Proposed data structure and feature extraction

Having discussed the requirements and some common design guidelines I am going to present the proposed data structure and discuss the design choices. Recalling the data structure of the WyScout API, I am going to present the various parts that make up the complete data model in the same order in which they are processed during the feature extraction phase. Note that the DBMS is first created and seeded with all the data current available, then is updated every time a new match is played. The feature extraction process is carried out for every JSON file that we currently have on hand during the database seeding procedure, then is performed again every time we want to update it with a new JSON file. It's important to consider that the data is highly connected, so the feature extraction procedure had to be designed to be run sequentially on a high number of documents.

1) Preprocessing

Before loading the data in the data base some preprocessing steps are required. First of all we need to clean the data. WyScout changed the way arrays are reported after 2015, so we need to convert the old representation to the new one - after 2015 every attribute in the JSON file is reported as an array, to have a consistent representation we need to wrap non-array attributes into one element arrays. Some features are then computed for each element:

- If the event is a shot we compute the angle with respect to the center of the goal.
- The game state and player state. To describe this two attributes is better to use an example. Let's assume that team A is winning the game against team B with a score of 2 to 1. All the events for team will have a game state of +1, and all the events for team B will have a game state of -1. So game state is simply the goal difference computed for each team. Player state is pretty much the same but considers the number of players on the field for each team - i.e. the number of red cards.
- The representation of the position is changed. WyScout represents position as the percentage distance on each axis starting from the bottom left corner - e.g. a coordinate of $(x = 50, y = 50)$ is the center of the field, while a coordinate $(x = 50, y = 100)$ is the center of the enemy team goal. For consistency with other Mercurius' software we need to shift the coordinates so that the center of the field is in $(x = 0, y = 0)$

2) Loading the match

Every time a document is loaded we create a new node with the Id of the match. Then we check the season id the game has been played in, if we haven't already created a node with such Id we also create a Season node. Then we create two Period nodes for the match, representing the first and second half of the game respectively.

3) Loading the events

Once the match has been loaded we start loading all the events. Events are the single more important entities for our model and are by far the largest in cardinality, averaging at around 1600 per match. We look for the Match node that we just created and link all the events for the game to the respective Period Node. The use of the Period node was implemented to speed up the retrieval of events: this is because a graph-oriented database when searching for a node connected to another

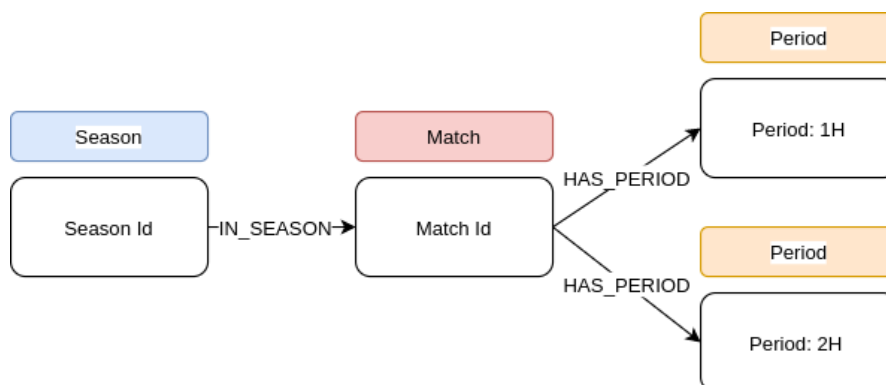


Figure 4.4: Match data structure

needs to traverse all the edges until it finds a correspondence. By dividing the events in two subgroups, if we know in which periods an event took place we can consistently halve the retrieval time for this nodes. This is also the phase in which we collect all the information necessary to compute the xG for every shot. To do so we need to extract the following parameters for every event:

- The event category - shot, pass, duel, etc.
- Starting and ending position
- Time elapsed since the beginning of the match
- The match Id - this is a redundant information, but is extracted for index optimization purposes
- The current game state - i.e. the difference in score and players on the field for the team performing the action
- The angle with respect to the center of the goal, valid only for shots
- Various tags to describe the context
- The player Id - again redundant but used for index optimization

4) Loading Players and "Participation" nodes

This portion of feature extraction sees a major adoption of the fan out pattern. For every match we want to load all the players that took part in it and create a node for each one of them, assuming they are not already in the data base. We iterate along the list of events and extract a list of distinct player Ids, their team Ids, the season Id and the match Id. Then we create a new Participation that

couples each player to all the actions he performed during a match. The result is shown in figure 4.5. The Participation node conveniently stores the information about the player, the match, the team the player played for at the time and is linked to the season node. In this way we can compute the expected goal with different granularity, depending on the need.

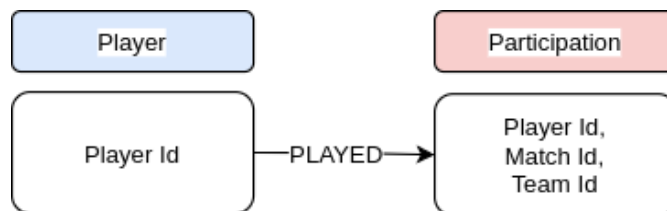


Figure 4.5: Caption

5) Link the events and the participation nodes

Once we created all the participation nodes, we iterate along every event of the match and we link it with the corresponding participation node.

6) Creating the "Next" relation

The next relation is the key aspect that allows the analysis of sequences of actions. Simply, every event is linked to the event that will happen next. This kind of relation has an attribute team that can assume two different values: "same" and "different". Basically if one event and the next one are done by players of the same team we label the link with the "same" label and we do otherwise in the opposite case. This simple trick allows us to express the concept of possession for a team: an arbitrary number of actions performed by the same team represents a possession, and the pattern matching capabilities of Neo4j allow us to look for sequences of events where all the "next" relation in the chain have the attribute "same".

4.6 Querying the database

Querying a graph database can be done in different ways, depending on how the data structure has been defined. Nodes can be retrieved directly with the corresponding Id, but also through a logical pattern. Let's say that we have a database with 100 games divided in 5 seasons, if we want to look for the game with $Id = 42$ we can use two different approaches:

- Query the database for the Match node with $Id = 42$. This query will require to check at most 100 nodes.

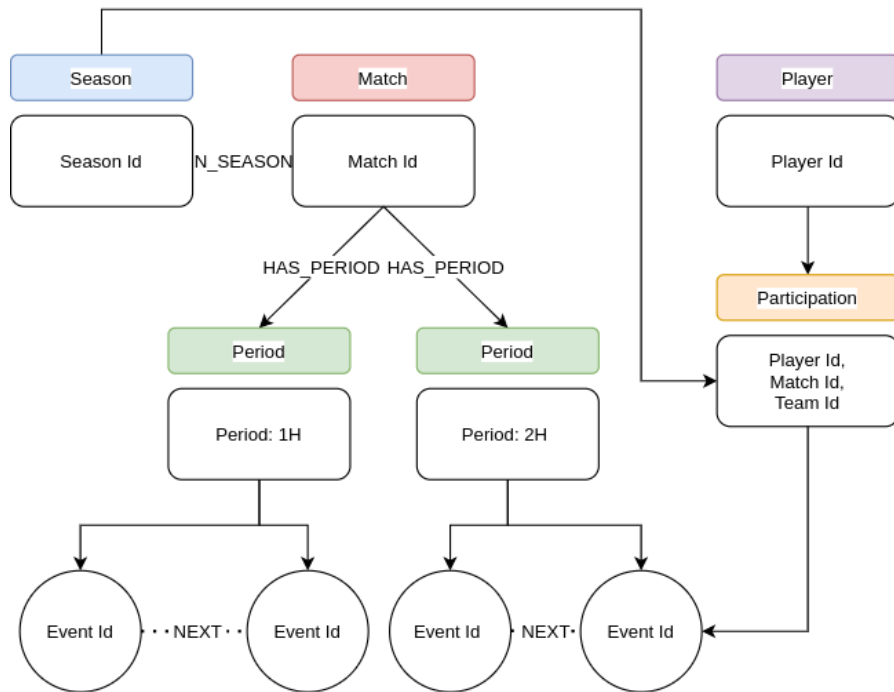


Figure 4.6: A representation of the final data structure. Note that not all the attributes are reported

- If we know in which season the game was played - e.g. $SeasonId = 3$ - we can query the database to query only the match nodes linked to the season node with $Id = 3$. This will take at most 5 checks to find the correct season and at most 20 checks (assuming the 100 matches are equally divided among the 5 seasons) to find the correct match.

Please consider that with the use of indexes the retrieval performance of relevant information is quicker, however this querying approach should be kept in mind when working with graph database, as it was found experimentally that following a known path is faster than looking for a specific node, even when indexed. The code reported in 1 shows how this approach is implemented. To load all the events in the corresponding match we need to iterate over all the events contained in the JSON file, create a node for each one of them and finally link this new node to the right period node of its corresponding event. At this time, the node of the match and periods are already in the database, but to operate a faster search we look for them starting from the Season id. Since the number of season is much smaller than the number of matches, this restricts the search space substantially.

Algorithm 1 The query used to load the events for every match, an example of how using paths optimizes the retrieval time.

```
1: WITH CALL apoc.load.json($path)
2:   YIELD valuew
3: UNWIND value['events'] as event
4: MATCH(s:Seasonid:value['match']['seasonId'])<-[IN_SEASON]-
   (m:Matchid:value['match']['wyId'])-[:HAS_PERIOD]->(p:Periodperiod:
   event['matchPeriod'])
5: CREATE (e: Eventid: event['id'])-[:IN_PERIOD]->(p)
6: SET e.name = event['eventName'],
7:   e.position = [event['positions'][0]['x'], event['positions'][0]['y']],
8:   e.time = event['eventSec'],
9:   e.matchId = value['match']['wyId'],
10:  e.teamId = event['teamId'],
11:  e.state = event['state'],
12:  e.angle = event['angle'],
13:  e.tags = event['tags'],
14:  e.subEventId = event['subEventId']
```

4.7 Architecture design

The designed database system of course requires to be interfaced with the pre-existing software that is currently in use by the company. All Merurius' services are hosted on Amazon Web Services platform, so also the database had to be hosted on an AWS service. To ensure complete operation, the architecture must include various components:

- A server hosting the DBMS
- A **seeding script** that loads all the matches currently available on the database storage.
- A way to load new matches every time WyScout provides an **update**

Fortunately the AWS environment provides powerful tools to build a web infrastructure that is able to satisfy all our requirements. In particular, an EC2 machine was used to host the DBMS, an S3 disk was used for the mass storage, while a stateless Lambda function was used to load new games in the DBMS. In figure 4.7 a scheme of the final architecture is presented.

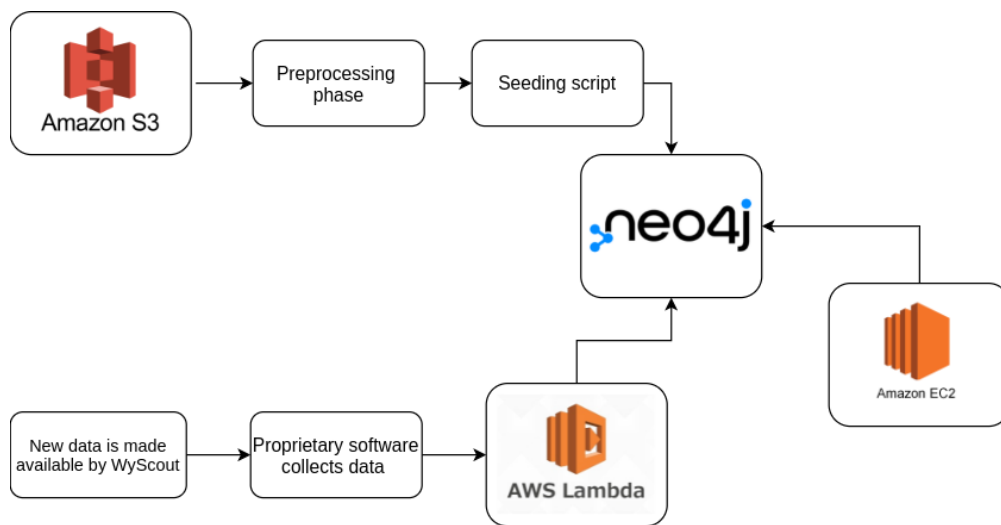


Figure 4.7: Caption

It's important to note that also the lambda functions performs the preprocessing phase and loads the match in the same way the seeding script does.

Chapter 5

Results

In this chapter are presented the performance and the possibilities that the designed DBMS provides. The dataset that was made available for testing is restricted to a total of 646 matches from the 2019-2020 Premier League season and the 2020-2021 Serie A season, with a total size of 282,9 MB. The resulting graph will have in order 1,1 millions of nodes and 3,2 millions of edges. The small dimension of the dataset is mainly related to the low availability of free data and the high cost associated with buying stream event data from a provider such as WyScout or StatsBomb.

5.1 Data exploration possibilities

The usability of a technology is an aspect that is hard to evaluate numerically. Graph-oriented database offer a visual representation that is easy to understand and navigate. In particular, Neo4j offers a complete solution for quick prototyping and experimentation the *Neo4j Browser User Interface*. This is a complete GUI that allows developers to quickly develop a new data schema, but can also be used to present implementation details to others. This possibilities proved to be especially useful while debugging queries and during the design phase of the data schema and new queries. Another notable feature is the possibility to start from a single node and explore the adjacent nodes to query the database in a more interactive way.

5.2 Read and write performance

Read and write performance are usually two of the most valued aspects when designing a DBMS. In this case however the write performance of the system is not as relevant. The seeding of the DBMS can be performed nighttime, while consistency after the release of new matches data can be achieved in relatively long

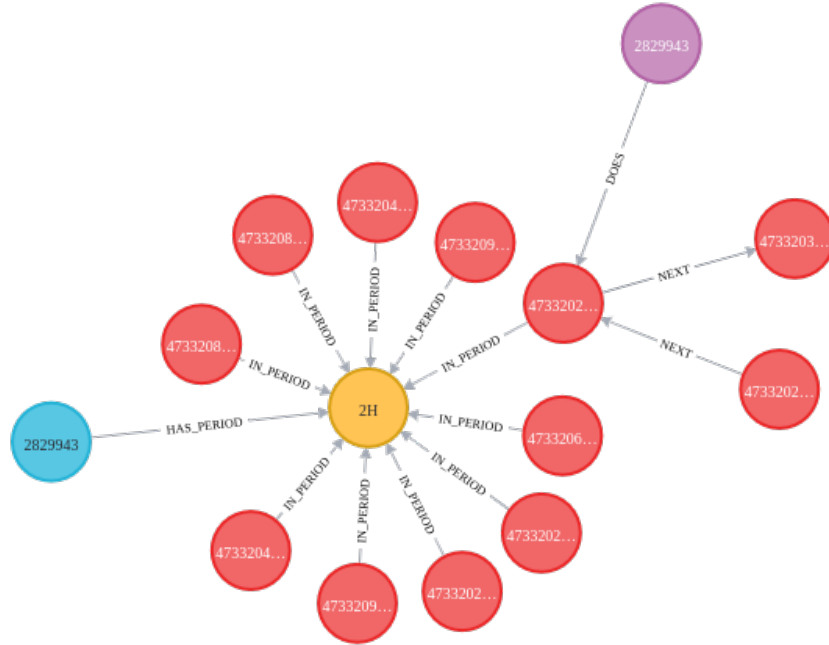


Figure 5.1: A portion of the data schema as represented in the *Neo4j Browser User Interface*

time frames - e.g. a couple hours. This is due to the fact that the AI models are not trained often. Figure 5.2 reports the loading time required to load each game from our dataset into the DBMS, the average time required to preprocess and load a game is 0.338 seconds. This kind of performance allows us to load a full 380 games season in under two minutes, which is a decent result, however to obtain this result it is very important to optimize the indexes used. The list of indexes is reported in table 5.1. Note that in Neo4j a single attribute index is used to speed up the retrieval of a node with a given Id -e.g. the match with *Id 19248* - while multiple attributes indexes are used to find a node that is linked to a starting node - e.g. the event with *Id 12313* linked to the match with *Id 19248*.

Reading performance on the other hand is a more important aspect since more rapidly available data can shorten the training cycle if we do not plan to keep all the dataset in the main memory. In case we have enough memory to do so instead we can extract the data that we need to carry out the training before hand, to reduce the impact that a real time extraction would bring. For the purpose of this work we are going to consider the expected goal as the only parameter used for the performance evaluation of every team, to accomplish this task one sample query could be: "Find all the shot that team with Id 3205 performed during season with Id 185727". When performing this kinds of query the DBMS starts streaming responses after 1ms, which is basically an immediate response. More complex

Label	Attributes	Type
Event	Id	Uniqueness
Event	Id-MatchId	Uniqueness
Match	Id	Uniqueness
Match	Id-SeasonId	Uniqueness
Participation	MatchId-PlayerId	Uniqueness
Season	SeasonId	Uniqueness

Table 5.1: The indexes used in the final design

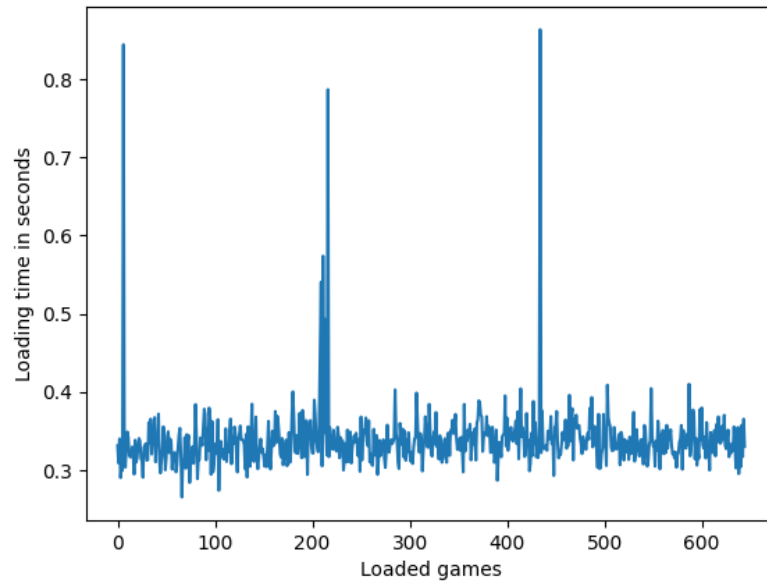


Figure 5.2: Time required to load every game from our dataset

queries require more time to be computed, for example a query like : "Find all the sequences of 5 events that start with a pass and end with a shot completed by the team with Id 1610" requires 65ms to return. This latter type of queries however represents a very advanced degree of aggregation that would otherwise be very to achieve by manipulating the data via software.

Chapter 6

Conclusions

In this work it is presented a complete DBMS architecture specifically designed for performance evaluation and value bet detection. This work wants to prove the efficacy of graph oriented DMBS in modelling soccer related data among other use cases.

The architecture is based on Neo4j, a NoSQL DBMS, which allows an agile development with rapidly changing requirements and design choices. Due to the nature of this kind of application, the system's design is query-oriented and tailored to work specifically with the WyScout event-stream data format, nonetheless the data schema can be used with other data provider that is able to provide the same contextual information. The ability of loading collections of JSON documents directly into the graph requires that the input files are strictly formatted to prevent errors, a problem which has arisen also with WyScout older data.

The system is also able to deliver great performances and functionalities. The main task that it would be able to complete was to find all the shots that a given player or team performed over a variable time window to compute the expected goal value of each of those shots. This kind of query is completed within one millisecond which is an almost latency free response. In addition, thanks to the graphs representation we are able to query the database for complex patterns with simple queries, which simplifies enormously the work load of designing new metrics. Also this kind of queries are computed extremely fast by the system.

This design should not be intended as a perfect design, but considering the lack of academic literature on the subject, it may prove as a useful starting point for further design improvements, or as a valuable solution for data scientist that want to implement new metrics, like movement chains [8]

Bibliography

- [1] Michael Stonebraker et al. «C-Store: A Column-oriented DBMS». In: *VLDB*. 2005 (cit. on p. 9).
- [2] H Dörge, Thomas Andersen, Henrik Sørensen, and Erik Simonsen. «Biomechanical differences in soccer kicking with the preferred and the non-preferred leg». In: *Journal of sports sciences* 20 (May 2002), pp. 293–9. DOI: 10.1080/026404102753576062 (cit. on p. 14).
- [3] Wolfgang Potthast, Kai Heinrich, Johannes Schneider, and Gert-Peter Brüggemann. «The success of a soccer kick depends on run up deceleration». In: July 2010 (cit. on p. 14).
- [4] Michael Bar-Eli and Ofer Azar. «Penalty kicks in soccer: An empirical analysis of shooting strategies and goalkeepers’ preferences». In: *Soccer and Society* 10 (Mar. 2009). DOI: 10.1080/14660970802601654 (cit. on p. 14).
- [5] Julen Castellano, David Casamichana, and Carlos Peñas. «The Use of Match Statistics that Discriminate Between Successful and Unsuccessful Soccer Teams». In: *Journal of human kinetics* 31 (Mar. 2012), pp. 139–47. DOI: 10.2478/v10078-012-0015-7 (cit. on p. 14).
- [6] J Finnoff, K Newcomer, and Edward Laskowski. «A valid and reliable method for measuring the kicking accuracy of soccer players». In: *Journal of science and medicine in sport / Sports Medicine Australia* 5 (Jan. 2003), pp. 348–53. DOI: 10.1016/S1440-2440(02)80023-8 (cit. on p. 14).
- [7] *How data, not people, call the shots in Denmark*. <https://thecorrespondent.com/2607/How-data-not-humans-run-this-Danish-football-club/517995289284-77644562> (cit. on p. 14).
- [8] *Movement chains*. <https://theanalyst.com/eu/2021/03/identifying-patterns-with-movement-chains/> (cit. on p. 35).
- [9] *Neo4j worst practices*. <https://neo4j.com/blog/dark-side-neo4j-worst-practices/>.

BIBLIOGRAPHY

- [10] Alex Augusto Timm Rathke. «An examination of expected goals and shot efficiency in soccer». In: *Journal of Human Sport and Exercise* 12 (2017), pp. 514–529.
- [11] Edgar F Codd. «A relational model of data for large shared data banks». In: *Communications of the ACM* 13.6 (1970), pp. 377–387.