

NoSQL

In this week, we will cover the following topics:

- *Definitions of the NoSQL 'approach' to databases.*
- *Alternative database forms, not just relational.*
- *ACIDity and NoSQL.*
- *A closer look into an industrial-strength NoSQL database: Neo4j.*

... and will result in the following learning outcomes

- *Appreciation of what NoSQL means.*
 - *An understanding that not all data suit a relational solution.*
 - *Appreciation that while many alternative databases exist, they can be classified into a relatively small number, based on the underlying data structure.*
 - *Understanding of how graph data can be stored and queried in a graph database, using neo4j as an example technology.*
-

Table of Contents

11.1	Definitions	3
11.1.1	What is 'SQL' in light of 'noSQL'	3
11.1.2	What is 'noSQL'	4
11.2	Classes of noSQL database	5
11.2.1	Key-value	6
11.2.2	Wide column	6
11.2.3	Document.....	6
11.2.4	Graph.....	7
11.3	Summary	7
11.4	The ACIDITY of noSQL and other features	7
11.4.1	ACID reminder	7
11.5	A noSQL example: graph databases and neo4J	8
11.5.1	What are graphs?	8
11.5.2	Why use a graph database?	9
11.5.3	Flexible data structures	10
11.5.4	Neo4J Queries: Cypher.....	12
11.6	ACID and SQL, noSQL	12
11.7	Summary	13
Figure 1:	Database choices in the post-relational landscape	5
Figure 2:	Example graphs – social network and road network.	8
Figure 3:	Relationships - Relational model v Graph model	10
Figure 4:	Dynamic accumulation of data.....	11
Figure 5:	Information from simple Cypher query.....	12

11.1 Definitions

11.1.1 What is 'SQL' in light of 'noSQL'

To get an understanding of what noSQL means, it is useful to define what we mean by SQL in this context, i.e., in the phrase **noSQL**. Of course, we have spent some time looking at SQL, both within the practical sessions and in a previous week where we had looked at several examples... querying for: Reading data, updating data etc, from a relational database.

For the current purpose, SQL should be taken to mean the structured query language, but also the object of that language, i.e., the relational databases itself. As we mentioned, the structured query language depends on the underlying language of mathematical sets, as described by Codd (TODO). When designing a database, the database designer is, consequently, required to think in terms of the tables, and the rows and columns that constitute the tables in question. Relationships are also 'coded' as tables. These constraints allow the database to be queried in a fast and effective manner, especially in comparison to the traditional storage of (manually created) and (manually updated) files.

Relational databases and SQL have now existed for decades. Relational databases and SQL will not disappear any time soon:

- The theory and practice of the relational approach is very well-established. Therefore, so is the technology of relational database management systems, and the generation of developers who have the skills to work with this technology.
- The ACID transactional features of the technology mean that these systems are very reliable, an important feature in an industrial context.
- As a consequence of being well-established, many software ecosystems integrate the use of SQL

Notwithstanding these facts, it is certainly true the relational database systems are not necessarily the only, or the best, choice of technology available. The relational model was intended for use in the context at data that is very well understood ahead of time. This means that the structure of the database can be very well-defined without the need for constantly changing its structure, the tabular schemas etc. In many applications this is still the case, and in these situations the use of the relational approach is still highly appropriate and, therefore, is frequently used. However, since the 1970s, which is the decade associated with the invention of relational database systems, the availability of data has exploded. No longer are organizations only interested in well-established and predictable sets of ordered data, they are adapting to the modern landscape and data availability, and trying to exploit the information that is inherent in all kinds of data. This includes data which is semi-structured, and generated in huge volumes.

The 'noSQL movement' is the response to this changing landscape. It is important to realise however that it does not *replace* relational database Systems, which is sometimes impression one can get from the literature. The 'noSQL' technologies very much exist alongside traditional relational database systems, to serve a different purpose, but possibly to fill a

niche within the overall software ecosystem. Database developers are therefore having to develop an 'agnostic' attitude towards database technologies, just as they are having to develop a similar attitude towards programming languages. It is becoming less common for a programmer to be referred to, for example, as a 'Java developer' and programmers must increasingly develop a repertoire of coding skills across different languages. Similarly, database development is becoming less about relational database development, even though this will remain very important. Database developers certainly should be aware of SQL approaches, hence the devotion of this week to this subject.

11.1.2 What is 'noSQL'

In the history definitions 'noSQL' is possibly one of the most useless. This is because the phrase 'noSQL' is not really a definition; it merely refers to something that it is not – **not** SQL, **not** relational, etc. This is a bit like defining your country, and all the institutions and cultures within it, by listing a set of other countries with different customs, rules of law etc. It is a negative definition. By defining something in his negative terms, we don't get any understanding of what 'noSQL' actually is, just a list of relational features we are familiar with that should be understood as 'not that'.

The phrase noSQL should really be used to indicate that there are other database technologies available. Furthermore, a less polemic interpretation of the phrase would be 'not only SQL'. In other words, as we have suggested, although you might use relational database systems, you should be aware of different database approaches, and how they fit with different data requirements.

As Redmond and Wilson suggest:

"This is a pivotal time for the database world. For years the relational model has been the de facto option for problems big and small. We don't expect relational databases will fade away anytime soon, but people are emerging from the RDMS fog to discover alternative options... These options are collectively known as noSQL..."

Redmond and Wilson (2012)

On starting a database project, rather than immediately thinking about a relational representation of the data, some relevant example questions might be:

- Is the client data conducive to a relational database approach or a non-relational approach?
- How might we represent certain aspects of the data in relation away and other aspect of the data in a non-relational way?
- If the data speaks to a non-relational representation, then what is this representation and what are the most appropriate database technologies available?
- If the amounts of data increases rapidly, how will the database perform?

- Is the relational model an efficient way of storing this data? Where are you to going back with

Therefore, while relational databases should not be looked upon as defunct technology, on the one hand, on the other, faced with questions regarding data, one can no longer just presume that a relational approach will be the 'best fit'. Being responsible as a database developer can mean, firstly, asking the above kinds of questions.

11.2 Classes of noSQL database

Figure 1 illustrates the choice of database available in the post-relational landscape. We distinguish four general kinds of database in addition to the relational database kind that has been the main stay of the course. Notice, each type of database is of a *specific* overall kind – i.e., we are not speaking at the level of a given database product (MySQL, sequel server, Oracle), but we are referring to the core nature of the type/class of database system itself.

In the diagram we distinguish between the database and the query language. We have previously said that no SQL refers to 'not only SQL', for example, but in Figure 1 it should be read 'no SQL querying'. Whereas relational databases are perfect to act targets for an SQL language implementation, this is not the case for noSQL databases. Such databases Will have not only different underlying representations for data, but different means of accessing the database, such as a different query language.

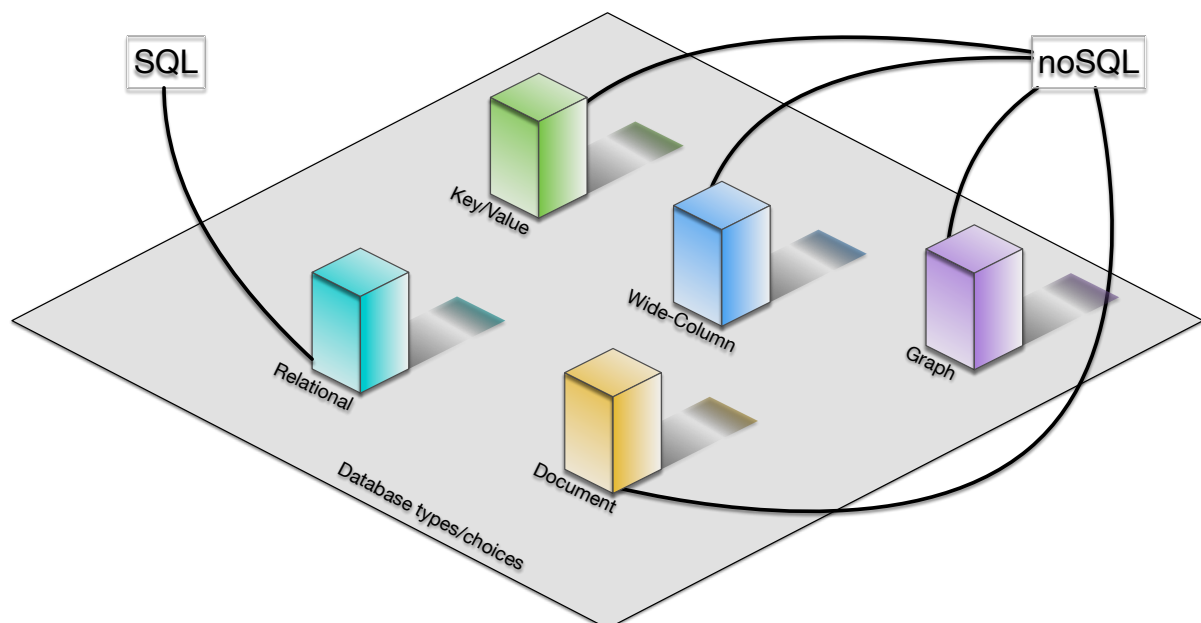


Figure 1: Database choices in the post-relational landscape

11.2.1 Key-value

Key-value storage is relatively simple. Each database entry maps a key to an associated value. The key, of course, must be unique, such that when accessed we get the value required with. The purpose of the key-value database is therefore to provide a mechanism where large quantities of separate data can be stored for rapid look-up. This model is very simple, which supports the high speed. Conversely, where the requirement is for more complicated aggregate information, possibly which depends on the ability to join sets of data, the key-value model is likely to be unhelpful.

Key-value databases can be held either in-memory or on disk. These kinds of database have arisen as a consequence of big data. Many countries, for example, have digitized healthcare records for the entire population. This use-case often requires a specific key (customer number) per-person-name; an example of a fast look-up requirement. As a data structure, the key-value database can therefore be thought of as related to the hash map (or hash table) where keys are mapped to values through a given function to 'map' one to another. The key-value database, however, may be very, very large and distributed across several servers.

11.2.2 Wide column

Wide-column storage uses tables rows and columns. Of course this is familiar to us. However, the distinguishing feature compared with relational databases is that the names and structure of the columns may vary from row to row, within the same table in a database. This feature is, therefore, very different to the strict tabular format of the relational database model we have covered early on during the course. Put differently, a wide-column database does not need to adhere to a common schema to exist in the same table.

Do not be misled by the similarities to relational databases here. As Redmond states "The difference may seem inconsequential, but the impact of this design decision runs deep. In column orientate databases, adding columns is quite inexpensive and is done on a row-by row-basis. Each row can have a different set of columns or none at all, allowing tables to remain sparse without incurring storage cost for null values". This is an advantage over the relational model, which has to store a null where no other value is required, and this takes up memory.

This no-null storage feature of wide-column databases, or in other words support for *sparse* data, is often a common advantage of noSQL databases options over the relational database framework. We will see in a later example what we mean by sparse data and why the tabular restriction of forced rectangular shapes (i.e., forced null-storage) are an unwanted constraint in the context of a sparse data example.

11.2.3 Document

In document databases the unit of storage is the document. The document itself is accessed with an identifier along with a number of types contained within the document. The appearance of the documents is often as a nested structure, which often look familiar to those who have ever studied (or even eye-balled) the structure of XML (Extensible Markup Language). Document databases are most frequently chosen when the data in question adheres to a JSON (JavaScript Object Notation) format. This format is human-readable (quite

easily for programmers due to the *name-value* pairs and *array, list, sequence, vector* data types it supports), and is also easy for machines to parse and generate.

11.2.4 Graph

Graph databases are designed to deal with data that is inherently ‘relational’ in the sense of being linked or connected. A graph data structure, as a mathematical object, is defined as nodes (or vertices) and links (or edges). The graph representation is best used when data can therefore be represented in this way. The mathematics relating to graphs is also concerned with how one can easily (or quickly) traverse a graph between, for example, two given nodes, what the shortest path between the nodes is etc.

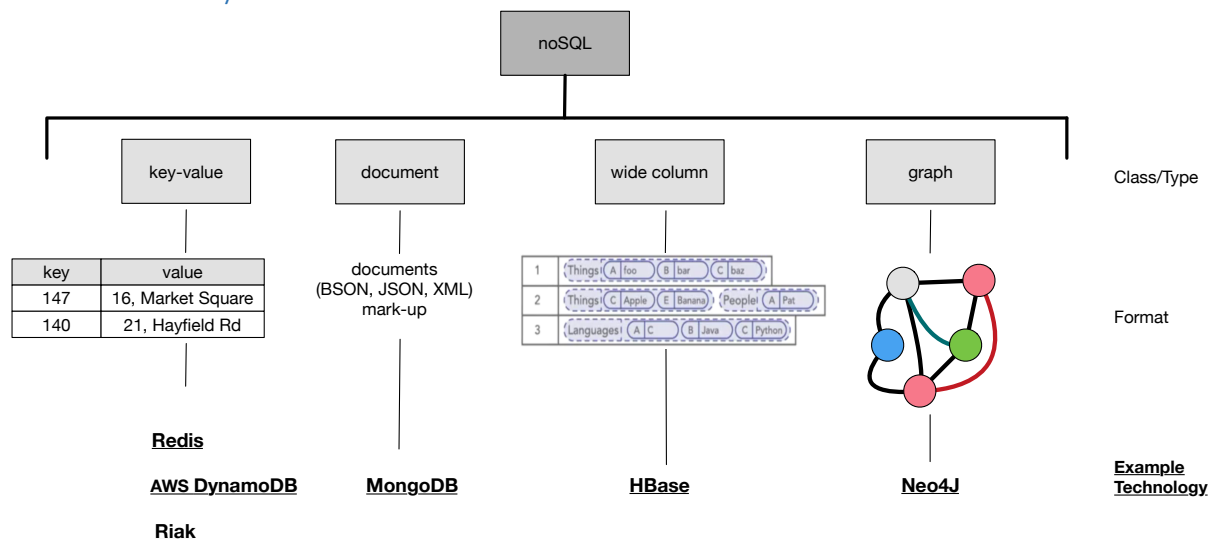
Graph databases are often a good choice where the data can be seen as a *network*. Of course, there can be many kinds of network – examples include transportation networks, social networks, networks based on customer-product interactions. The list is literally endless.

Redmond (2012) states that graph databases are “*One of the less commonly used database styles*”, although it is worth bearing in mind (this is true of any textbook) that the printed page is often rapidly out of date, especially in computational subjects:

“We’ve seen industry after industry being eaten by graphs. In each case, the adoption of graph technology has resulted in better products and more remarkable customer experiences... Four of the top 10 global retailers today use Neo4J. Behind them, their non-adapting competitors are struggling to make it...”

Ian Robinson graph databases (2015)

11.3 Summary



11.4 The ACIDITY of noSQL and other features

11.4.1 ACID reminder

11.5 A noSQL example: graph databases and neo4J

11.5.1 What are graphs?

The notion of graphs originates from the ideas of graph theory. As mentioned, nodes (or vertices) and links (or edges) are used as the basic units. A node, for example could represent an object/thing and the links represents relationships between objects/things. Graphs are highly abstract and can represent many kinds of objects and relationships. It is a good idea therefore to think about some examples.

Geographic example: transport networks can be represented as graphs. For example, road networks consist of nodes and links. The links can represent the roads themselves. The nodes can represent the places where roads meet, or in other words road *junctions*. In this example, we might be interested in knowing the distance between two given junctions. One way to proceed would be to record the distance of each road and store this as a value on the associated link in the database.

Social network example: social networks are also graphs and consist of nodes and links. In this case, the nodes are usually represented as people and the interactions between people as the links. These days, Facebook is a commonly understood social network, where each person is related to others by the 'friend relationship'.

The entire graph represents the network of relationships; it consists of the entire set of objects within a system and the interactions between them. An illustration of the example graphs considered is presented in Figure 2.

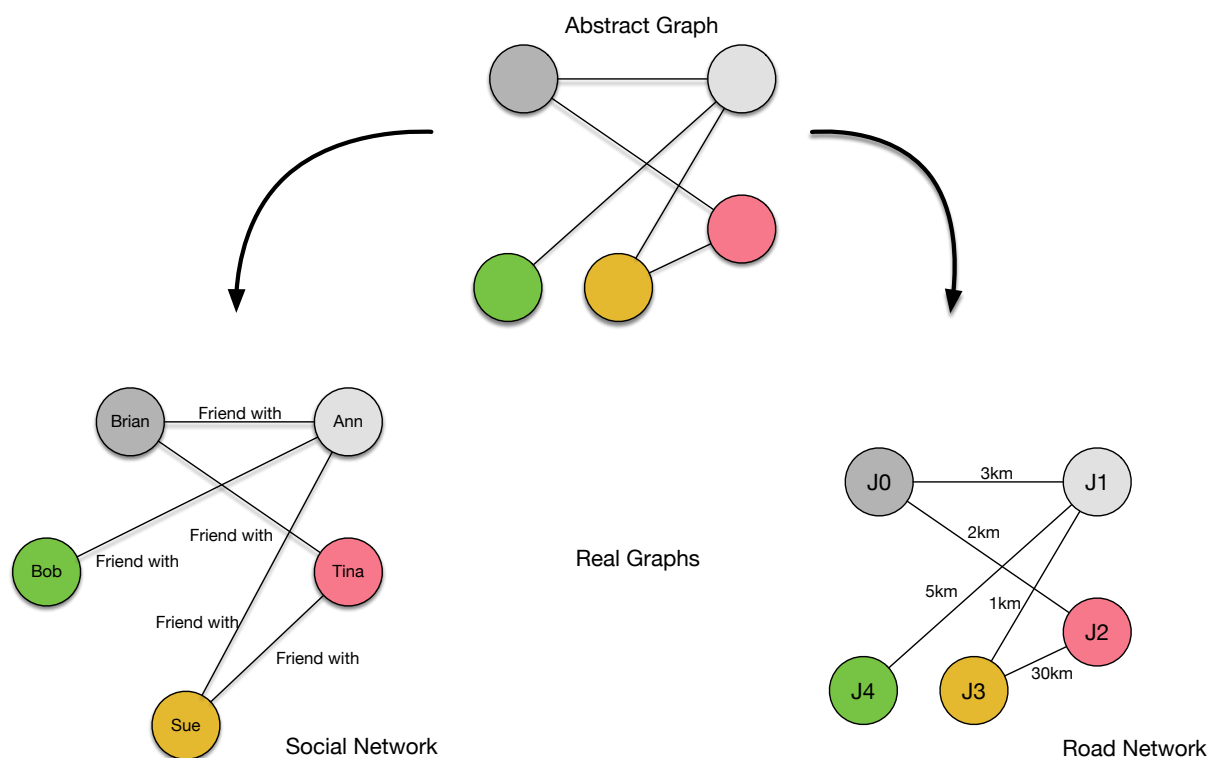


Figure 2: Example graphs – social network and road network.

11.5.2 Why use a graph database?

Graph databases are a relatively new 'kid on the block'. The reasons you might use a graph database can be appreciated by the critique often level against the relational approach when the data speaks naturally to a network representation. That is, the relational approach cannot:

- model or store data and relationships without complexity.
- perform well when the number and levels of relationship increase. In other words relational databases do not scale well when the data is naturally relational.

These disadvantages can have other consequences. When the data is best represented as a graph, the relational approach will often:

- result in query complexity grows with need for complex joins.
- Result in schema re-design when new data types are introduced.

This brings us to the reasons why we might want to use a graph database. Graph databases:

- contain structures to directly model and store relationships. The modelling of relationships do not require the creation of tables.
- Exist with query languages that support data relationships naturally, so queries on the database are often simpler.
- Carries forward the advantage of ACID from SQL. This is not necessarily true of other noSQL databases.
- are whiteboard friendly. This means that they model data as it occurs naturally.
- scalability optimized for graphs.
- also often have drivers, as MySQL does, to support application programming in popular high-level languages.

It seems a strange thing to say that '*relational* database management systems do not naturally support *relational* data'. However, let us think about what we must do when adopting the relational approach to represent relationships, following the social network example, above. We must create a table to represent the relationships, whereas using a graph model we can accomplish the data storage in a way that is more natural, storing it as a 'relationship'. This is illustrated in Figure 3.

Therefore, when we come to choosing which database model to use, it is important to think about the modelling domain I'm what it is we want to achieve by storing the data. A further example of an appropriate use-case for graph databases, again as illustrated above, is the storage of transport networks. Transport networks are sparse networks – i.e., junctions are connected by a relatively small number of links, compare to the number of links possible and, given the number of junctions in the transport network. The point is not that it is impossible to store transport network data within relational database system, but that these days, it is not necessary to do this and 'better' technologies are available, i.e., technologies is more appropriate to the task.

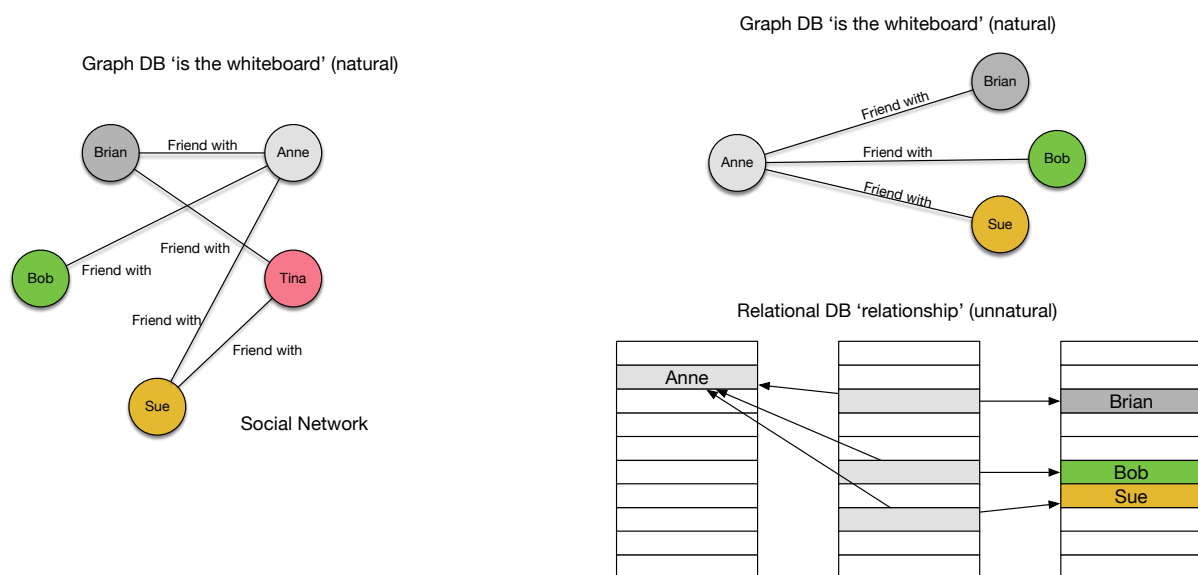


Figure 3: Relationships - Relational model v Graph model

11.5.3 Flexible data structures

In graph models the data structures are 'nodes' and 'relationships':

- **Nodes:** are the objects in the graph. Nodes can be labelled and can also have name-value properties.
- **Relationships:** the Late notify type direction. Relationships can also have a main value properties.

A key advantage no SQL databases is the ability to add new types of data overtime without redesigning the schema. As an example of this, consider to people who begin as friends, who end up falling in love, living together, and sharing the things they own such as their cars. The resulting database is then a collection of objects whose type and relationships may not be known ahead of time. Using a relational approach, the data schema would have to be updated, but using the noSQL graph DB approach, the data can simply be added dynamically without the need for such re-design.

An example situation is presented in Figure 4. Anne and Bob starts out knowing each other as friends. After some time, they fall in love, then move in together. At this stage, they have the relationships 'Friends with', 'Loves', and 'Lives with'. However, Anne then buys a car on the 07-04-2013. She therefore 'Owns' that car, but Bob has a different relationship to it; he 'Drives' it.

Name : Anne
DOB: 07-04-1988

Name : Bob
E-mail: bob@...
DOB: 02-09-1986

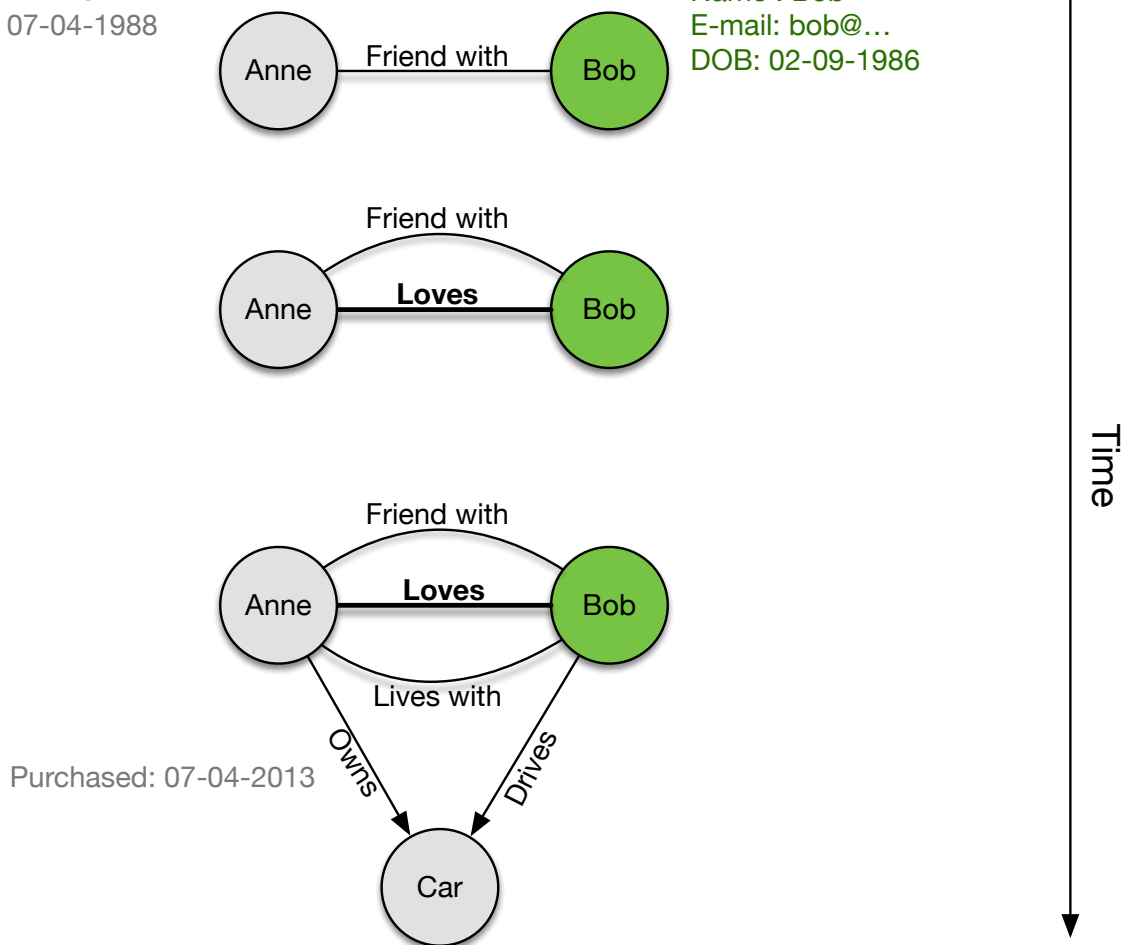


Figure 4: Dynamic accumulation of data

There are two aspects to this example worth noting:

1. The data, added over, time is not known ahead of time. However, it can still be added to the database.
2. The information associated with the objects/nodes (people and car in this case) does not share the same structure. Notice, even the people have a different structure in this respect (Anne has no e-mail).

In this way, graph databases are said to be good at representing semi-structured data. We mentioned at the very beginning of the course the difference between structured and unstructured data. No SQL databases often target Data that is not as well structured as the data found within a relational database system. In recent times, the explosion data has meant that semi-structured solutions have emerged on the market to cater for this kind of data. Such data is not completely unstructured, although it is not fully structured, either, at least in a relational sense.

11.5.4 Neo4J Queries: Cypher

The purpose here is not to cover in any depth of the Cypher query language, but rather just to note, generally, that with no SQL databases, SQL is *not* used. In the example we have given, Cypher is to Neo4J what SQL is to a relational database.

We mentioned previously that nodes could be labelled and nodes and relationships have known value properties. So, continuing the above example if we wanted to extract from the database people who lived with each other we could write the following Cypher query:

```
MATCH (:Person { name:"Bob" } ) -[:LIVES_WITH]-> (:Person { name:"Anne" } )
```

Which would return the following information from the database:



Figure 5: Information from simple Cypher query.

11.6 ACID and SQL, noSQL

At transaction, in database terms, are typically a set of instructions that are executed on a database. The instructions executed, may follow some sequential logic and therefore should be ordered in a way, according to that logic. To maintain the integrity of the database, it is said that the transactions should be:

- **Atomic:** instructions/queries are part of a unit and therefore all the instructions/queries within that unit of work should be completed successfully; or, the transaction is exited due to a failure. In the case that the transaction fails the state of the database is returned to its former state before the beginning of the transaction in question. There can be no partial transaction, it is all or nothing.
- **Consistent:** changes to the data cannot break the integrity of the database rules set.
- **Isolated:** one transaction must complete before any others are executed on the same data. This is to prevent two transaction altering data, together, potentially creating problems with the data in question.
- **Durable:** allows the results of a transaction to persist. Once a transaction has occurred it has been guaranteed.

See Transactions and the ACID test (Lynda.com 'Foundations of Programming', Simon Allardice)

Without passing the ACID test, you could imagine all kinds undesirable scenarios. For example, customers paying for holidays but without receiving a guarantee booking the room; wo customers being able to book the last available seat on outgoing flights from Heathrow; the last available sofa being bought by more than one customer at the same time etc.

SQL databases are ACID compliant. On the other hand, no SQL databases often do not fulfil that acid test. This is something that is worth bearing in mind, therefore, when you come to choosing which kind database to use.

However, although many no SQL databases do not provide ACID support some do. One of the industrial strength features of neo4J is support for fully ACID transactions.

11.7 Summary

- The diverse field known as the 'noSQL movement' a growth area in light of changes to the nature of data, which exploded in recent decades.
- The relational database approach, is out of date for some modern purposes, but still very useful.
- The noSQL movement consists of the hundreds of different kinds of databases such that you will can do database development, *not only* using SQL, but also using other kinds of databases.
- Other classes of database system were given: ('key-value', 'wide column', document-based, Graph)'.
 - Neo4J, a leading industrial strength graph database introduced the concept of a graph and its components, and (very) briefly we looked at Cypher, neo4J's query language.

~~~~~