With the advancements in computing technologies and the increasing processing power of machines, the necessity to shoehorn data into predefined structures is being gradually replaced by the ability to build models that can accommodate the organic growth and evolving nature of data.

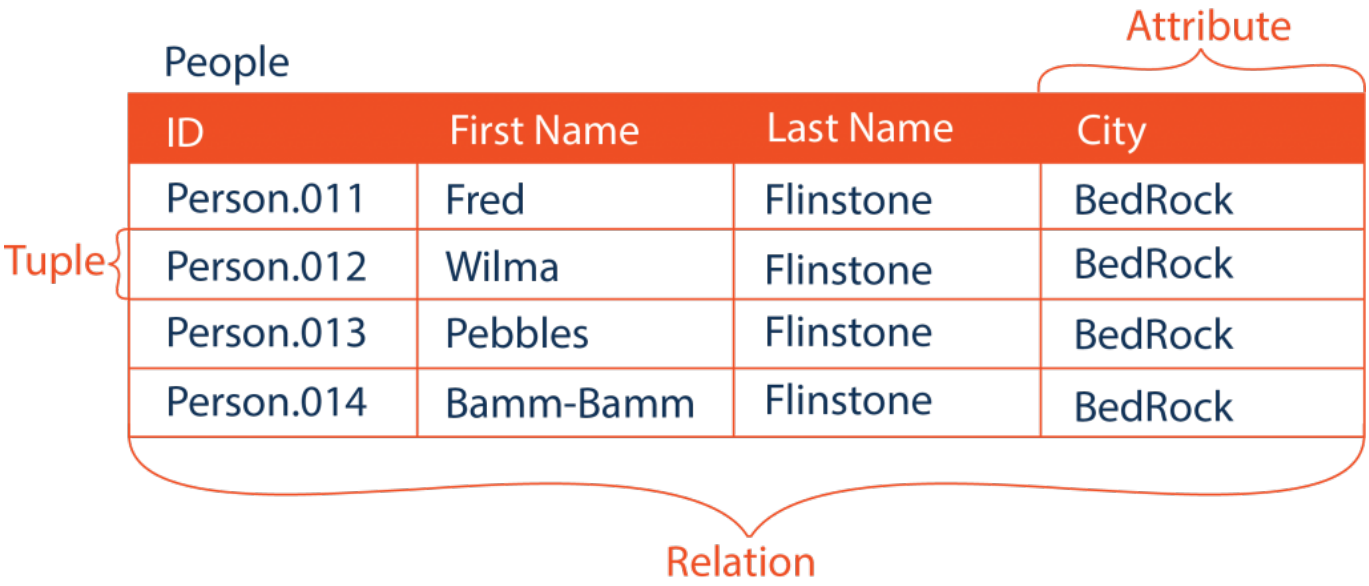
Systems become more and more sophisticated in representing our “messy” world and the interconnections we live by in it. The richer our databases get and the better (more expressive) the connections of the data items in them, the deeper our analysis and the bigger our potential to understand, manage and create processes to move our business forward.

That said, having the best home for your data is a must. It is to be a space where data items not only live together but also are understood and used accordingly for insights into the relationships they exist in.

Relationships expressed with tables, lots of tables: Relational Databases

For many years relational databases have been the dominant database choice for data storage and retrieval. The relational model stores data in tables, with rows representing instances of entities and columns representing the values attributed to each entity.

For example, if you want to express that Fred, Wilma and Pebbles Flintstone, together with the notorious Bamm-Bamm Rubble are instances of the entity Person and live in the instance of City Bedrock, you can do that with the following table, where you will describe these relationships:



As things change with time, and the number and the variety of relationships grow, you will need additional tables. For the newly occurred relationships to be expressed, you will create more and more tables.

In relational databases, references (i.e., connections) to other rows and tables are made with the help of the so-called JOINs. This means that in order to connect an entity from one table to another, you create a third table, which matches the records from both tables.

For instance, in order to express that Pebbles Flintstone became Bamm-Bamm's wife, another table has to be created. This will be a junction table, representing Bamm-Bamm (Person.014) as an instance of the entity Husband and Pebbles (Person.013) as an instance of the entity Wife.

Marriage Status

Husband	Wife
Person.011	Person.012
Person.013	Person.014

Any other relationships you might want to add would need to be explicitly described in a table, which refers to the first table you've created. That is, if you want to also express that Fred (Person.011) and Wilma (Person.012) are Pebble's (Person.013) parents, you will have to do that with yet another JOIN where Fred and Wilma are instances of the entity Parent and Pebbles is an instance of the entity Child:

Parents

Parent	Child
Person.011	Person.013
Person.012	Person.013

It is only with all these additional, explicitly expressed relationships that you will be able to use the data above to find the answer of more complex queries, such as: "In which city does the child of Fred live?" or "Who is Bamm-Bamm's wife?? or "Where does the father of Bamm-Bamm's wife live?".

Needless to say, such relational representations are suitable for simple data models and connections that fit into a tabular format. They are perfectly fine for financial records, inventories, lists of students, etc. When it comes to mapping complex networks of relationships, though, the processes of joining are most often than not inefficient, time-consuming and computer power consuming.

Interconnected data (the most obvious example being the data from social networks) are everything but easy to tame with the above mechanism of creating more and more junction tables and additional elements to record the ever-increasing number of relationships between data items. The relational model turns out to be too expensive and resource-consuming to express the richness and the interconnectedness of exponentially growing in volume and variety data.

The huge amount of heterogeneous, diverse data that surround us is to be approached differently.

Relationships as paths of (machine) understanding: Graph Databases

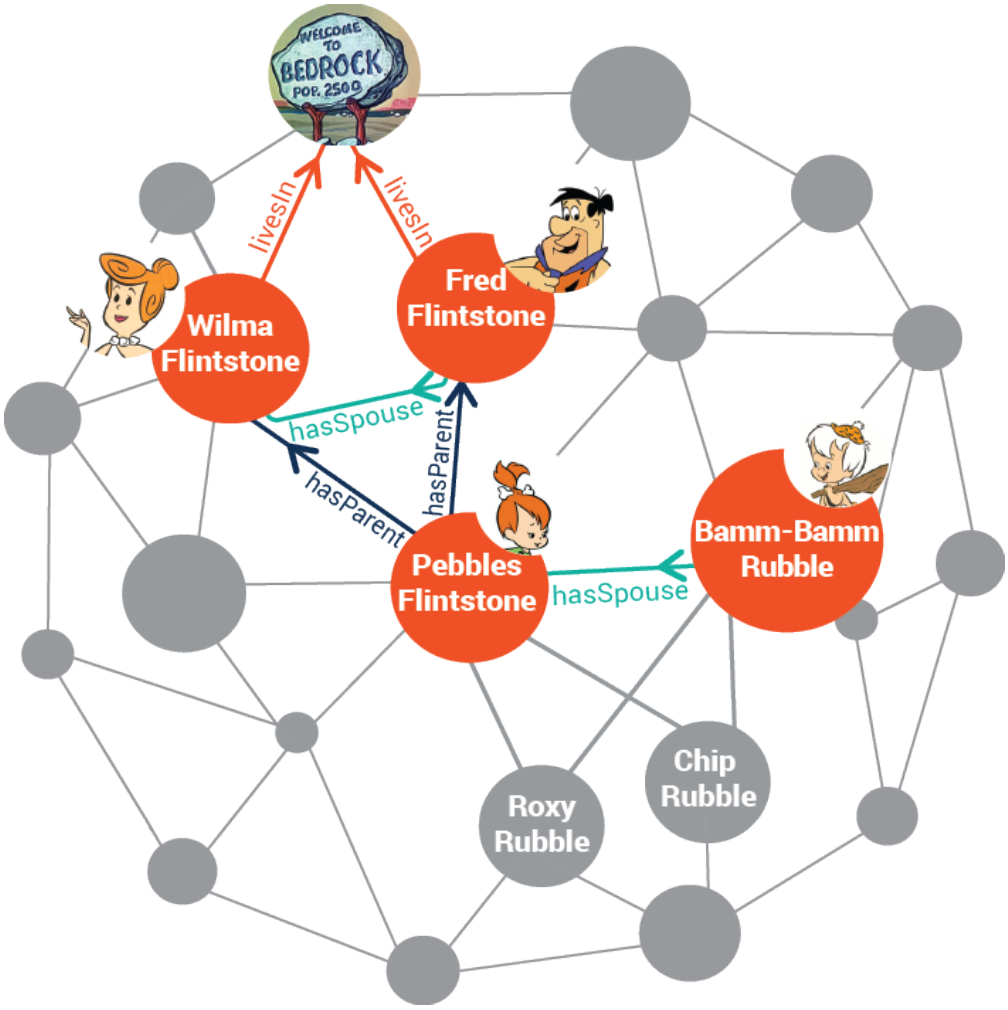
Outside the tables of the relational databases, there lie paths that enable managing highly connected data, working with complex queries and having readily available relationships, without the need to express them explicitly.

These are the paths of a graph database.

To represent and store data, graph databases use graph structures (knowledge graphs). A graph is comprised of interconnected nodes (i.e., things) and edges (i.e., relationships between things). Edges are how you can assign properties to things.

Also, instead of creating tables for each relationship separately, in a graph database you will just add edges (relationships) to corresponding nodes (things). Thus a node's connection, in turn, is connected to all the other connections of this same node.

To get back to our Flintstone's example, all the tables that you've created for every relationship between Fred, Wilma, Pebbles and Bamm-Bamm, would be expressed the following way in a graph database:



Thus, when you connect Bamm-Bamm to Pebbles, you also connect Bamm-Bamm to all of Pebble's connections – Fred, Wilma and with their other connections – city, for example. The system will implicitly hold the information about where the father of Bamm-Bamm's wife lives, without you having to keep a record of multiple joins and tables to retrieve it.

Storing data in such a manner provides the flexibility to represent complex interconnected structures and to use the information they carry in the much simpler and effective way.

When should you use a graph database?

There's simply nothing you can do with graph databases that you can't with relational ones.

That said, let's take a look at how a graph database can help you do more with data.

In a sense, with graph databases, data are allowed to organically grow and easily connect with more and more items. Share on X

It's only natural to consider a graph database for complex data, with many connections, the pattern of which you want to track and know about. A graph database would smoothly incorporate new heterogenous data and serve as a framework for storing, managing and querying highly connected data.

Graph databases are well-suited in any case when complex relationships between people, places, events, etc. are to be expressed. Typical use cases range from managing customers and personnel data, through storing and using intelligent content, to finance and investment management.

More specific application of the graph database model include:

- resource planning
- performance analysis
- fraud detection
- compliance management
- content and asset management
- recommendations
- production management
- business processes optimization
- identification of patterns and insights
- integration of heterogeneous scientific data
- enterprise search and navigation optimization

The future of data storage and management

The ability to pull data and connect them gives enterprises a significant edge when it comes to the granular understanding of the environment they operate in and the optimization of their key business processes. Share on X

This ability depends on the quality of the models chosen for data representation, storage and retrieval. The more accurately and efficiently the structure of a particular domain is mapped, represented and interconnected, the bigger the value and the potential of the digital data it creates each and every day.

Still, the decision to build a home for all your data, neatly classified and labeled, related, interconnected and easily searchable is a matter of cost and benefit analysis.

What is important is to acknowledge the opportunity for data to be turned into a resource, easily accessed and effectively used across the organization. As a database can serve not only as a storage cupboard for siloed archives but rather as a springboard for knowledge discovery.

Want to learn more about graph databases like Ontotext's GraphDB?

White Paper: The Truth About Triplestores

Download Now