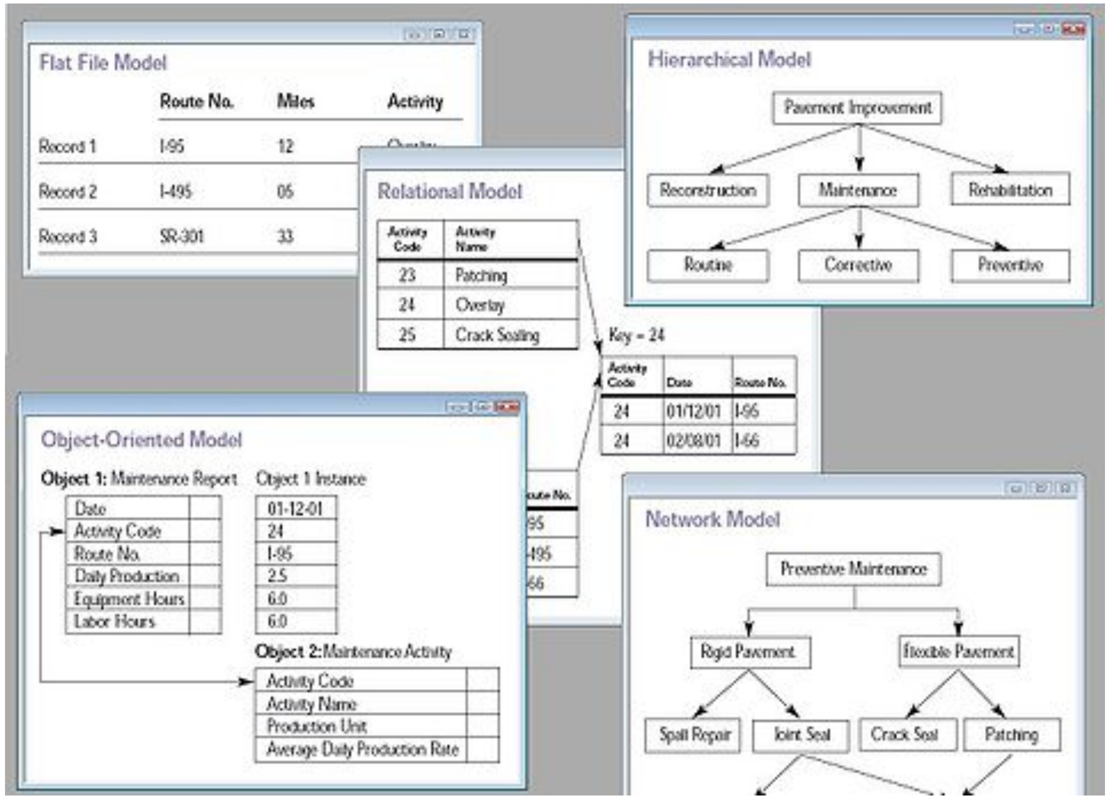
Traditional and NoSQL Database Models

This article defines explains some types of database models and describes data modeling in the context of the development of a database subsystem by focusing on its requirements, high level design, detailed design, and implementation. The first diagram depicts 5 categories of data models: flat files, early data models (network and tree), relational, and post-relational.



Collage of five types of database models

A **database model** is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized and manipulated. The most popular example of a database model is the relational model, which uses a table-based format.

Database Model Examples

Common logical data models for databases include:

- Hierarchical database model. It is the oldest form of data base model. It was developed by IBM for IMS (information Management System). It is a set of organized data in tree structure. DB record is a tree consisting of many groups called segments. It uses one to many relationships.

- Network model
- Relational model
- Entity–relationship model
- Object model
- Document model
- Entity–attribute–value model
- Star schema

Physical data models include:

- Inverted index
- Flat file

Other models include:

- Associative model
- Correlational model
- Multidimensional model
- Multivalue model
- Semantic model
- XML database
- Named graph
- Triplestore

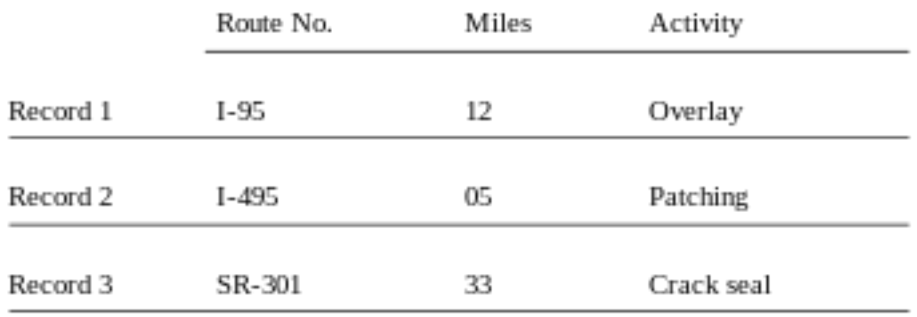
Relationships and functions

A given database management system may provide one or more models. The optimal structure depends on the natural organization of the application's data, and on the application's requirements, which include transaction rate (speed), reliability, maintainability, scalability, and cost. Most database management systems are built around one particular data model, although it is possible for products to offer support for more than one model.

Various physical data models can implement any given logical model. Most database software will offer the user some level of control in tuning the physical implementation, since the choices that are made have a significant effect on performance.

A model is not just a way of structuring data: it also defines a set of operations that can be performed on the data. The relational model, for example, defines operations such as select (project) and join. Although these operations may not be explicit in a particular query language, they provide the foundation on which a query language is built.

Flat model



	Route No.	Miles	Activity
Record 1	I-95	12	Overlay
Record 2	I-495	05	Patching
Record 3	SR-301	33	Crack seal

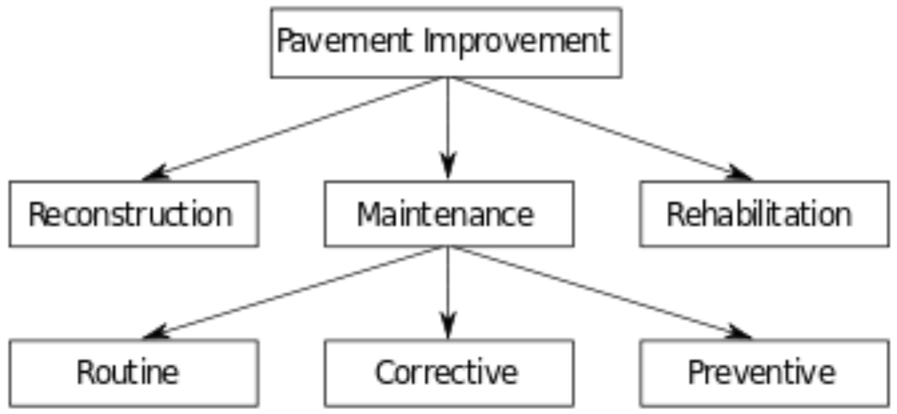
Flat File Model

The flat (or table) model consists of a single, two-dimensional array of data elements, where all members of a given column are assumed to be similar values, and all members of a row are assumed to be related to one another. For instance, columns for name and password that might be used as a part of a system security database. Each row would have the specific password associated with an individual user. Columns of the table often have a type associated with them, defining them as character data, date or time information, integers, or floating point numbers. This tabular format is a precursor to the relational model.

Early data models

These models were popular in the 1960s, 1970s, but nowadays can be found primarily in old legacy systems. They are characterized primarily by being navigational with strong connections between their logical and physical representations, and deficiencies in data independence.

Hierarchical model

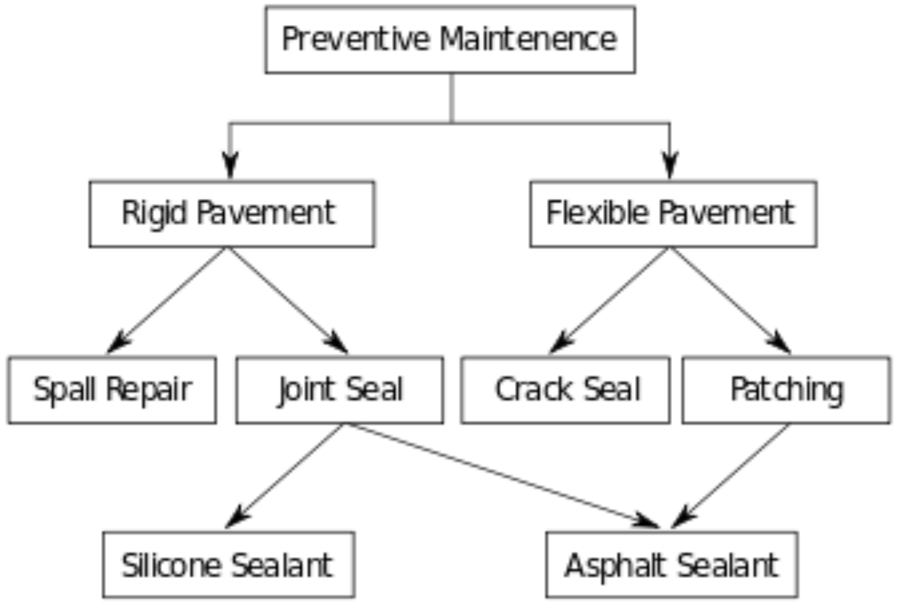


Hierarchical Model

In a hierarchical model, data is organized into a tree-like structure, implying a single parent for each record. A sort field keeps sibling records in a particular order. Hierarchical structures were widely used in the early mainframe database management systems, such as the Information Management System (IMS) by IBM, and now describe the structure of XML documents. This structure allows one one-to-many relationship between two types of data. This structure is very efficient to describe many relationships in the real world; recipes, table of contents, ordering of paragraphs/verses, any nested and sorted information.

This hierarchy is used as the physical order of records in storage. Record access is done by navigating downward through the data structure using pointerscombined with sequential accessing. Because of this, the hierarchical structure is inefficient for certain database operations when a full path (as opposed to upward link and sort field) is not also included for each record. Such limitations have been compensated for in later IMS versions by additional logical hierarchies imposed on the base physical hierarchy.

Network model



Network Model

The network model expands upon the hierarchical structure, allowing many-to-many relationships in a tree-like structure that allows multiple parents. It was most popular before being replaced by the relational model, and is defined by the CODASYL specification.

The network model organizes data using two fundamental concepts, called *records* and *sets*. Records contain fields (which may be organized hierarchically, as in the programming language COBOL). Sets (not to be confused with mathematical sets) define one-to-many relationships between records: one owner, many members. A record may be an owner in any number of sets, and a member in any number of sets.

A set consists of circular linked lists where one record type, the set owner or parent, appears once in each circle, and a second record type, the subordinate or child, may appear multiple times in each circle. In this way a hierarchy may be established between any two record types, e.g., type A is the owner of B. At the same time another set may be defined where B is the owner of A. Thus all the sets comprise a general directed graph (ownership defines a direction), or *network* construct. Access to records is either sequential (usually in each record type) or by navigation in the circular linked lists.

The network model is able to represent redundancy in data more efficiently than in the hierarchical model, and there can be more than one path from an ancestor node to a descendant. The operations of the network model are navigational in style: a program maintains a current position, and navigates from one record to another by following the relationships in which the record participates. Records can also be located by supplying key values.

Although it is not an essential feature of the model, network databases generally implement the set relationships by means of pointers that directly address the location of a record on disk. This gives excellent retrieval performance, at the expense of operations such as database loading and reorganization.

Popular DBMS products that utilized it were Cincom Systems' Total and Cullinet's IDMS. IDMS gained a considerable customer base; in the 1980s, it adopted the relational model and SQL in addition to its original tools and languages.

Most object databases (invented in the 1990s) use the navigational concept to provide fast navigation across networks of objects, generally using object identifiers as "smart" pointers to related objects. Objectivity/DB, for instance, implements named one-to-one, one-to-many, many-to-one, and many-to-many named relationships that can cross databases. Many object databases also support SQL, combining the strengths of both models.

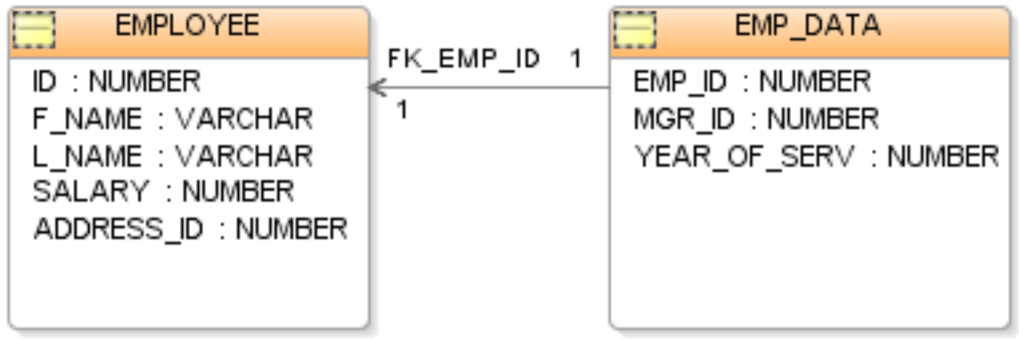
Inverted file model

In an *inverted file* or *inverted index*, the contents of the data are used as keys in a lookup table, and the values in the table are pointers to the location of each instance of a given content item. This is also the logical structure of contemporary database indexes, which might only use the contents from a particular columns in the lookup table. The *inverted file data model* can put indexes in a set of files next to existing flat database files, in order to efficiently directly access needed records in these files.

Notable for using this data model is the ADABAS DBMS of Software AG, introduced in 1970. ADABAS has gained considerable customer base and exists and supported until today. In the 1980s it has adopted the relational model and SQL in addition to its original tools and languages.

Document-oriented database Clusterpoint uses inverted indexing model to provide fast full-text search for XML or JSON data objects for example.

Relational model



Two tables with a relationship

The relational model was introduced by E.F. Codd in 1970[1] as a way to make database management systems more independent of any particular application. It is a mathematical model defined in terms of predicate logic and set theory, and its implementing it have been used by mainframe, midrange and microcomputer systems.

The products that are generally referred to as relational databases in fact implement a model that is only an approximation to the mathematical model defined by Codd. Three key terms are used extensively in relational database models: *relations*, *attributes*, and *domains*. A relation is a table with columns and rows. The named columns of the relation are called attributes, and the domain is the set of values the attributes are allowed to take.

The basic data structure of the relational model is the table, where information about a particular entity (say, an employee) is represented in rows (also called tuples) and columns. Thus, the "relation" in "relational database" refers to the various tables in the database; a relation is a set of tuples. The columns enumerate the various attributes of the entity (the employee's name, address or phone number, for example), and a row is an actual instance of the entity (a specific employee) that is represented by the relation. As a result, each tuple of the employee table represents various attributes of a single employee.

All relations (and, thus, tables) in a relational database have to adhere to some basic rules to qualify as relations. First, the ordering of columns is immaterial in a table. Second, there can't be identical tuples or rows in a table. And third, each tuple will contain a single value for each of its attributes.

A relational database contains multiple tables, each similar to the one in the "flat" database model. One of the strengths of the relational model is that, in principle, any value occurring in two different records (belonging to the same table or to different tables), implies a relationship among those two records. Yet, in order to enforce explicit integrity constraints, relationships between records in tables can also be defined explicitly, by identifying or non-identifying parent-child relationships characterized by assigning cardinality (1:1, (0)1:M, M:M). Tables can also have a designated single attribute or a set of attributes that can act as a "key", which can be used to uniquely identify each tuple in the table.

A key that can be used to uniquely identify a row in a table is called a primary key. Keys are commonly used to join or combine data from two or more tables. For example, an *Employee* table may contain a column named *Location* which contains a value that matches the key of a *Location* table. Keys are also critical in the creation of indexes, which facilitate fast retrieval of data from large tables. Any column can be a key, or multiple columns can be grouped together into a compound key. It is not necessary to define all the keys in advance; a column can be used as a key even if it was not originally intended to be one.

A key that has an external, real-world meaning (such as a person's name, a book's ISBN, or a car's serial number) is sometimes called a "natural" key. If no natural key is suitable (think of the many people named *Brown*), an arbitrary or surrogate key can be assigned (such as by giving employees ID numbers). In practice, most databases have both generated and natural keys, because generated keys can be used internally to create links between rows that cannot break, while natural keys can be used, less reliably, for searches and for integration with other databases. (For example, records in two independently developed databases could be matched up by social security number, except when the social security numbers are incorrect, missing, or have changed.)

The most common query language used with the relational model is the Structured Query Language (SQL).

Properties of Relational Databases

1. Values are atomic

Columns in a relational table are not repeating groups, this property ultimately simplifies data manipulation logic.

2. Each column and row has a unique name

The columns and rows are not distinctive by position and therefore they must have a name which is distinct from the rest of the table in order to identify them.

3. Each row is unique

No two rows are identical which ensures that every row has significance without duplication.

4. The sequence of columns and rows is insignificant

This allows the information in the table to be retrieved with ease and without and disruption to the data within the table. Also, it allows many more users of the table because the data is not confined to one format and is not changed when information is added or removed.

5. Column values are of the same kind

Each value within a specific column holds the same type of information and is of the same units as all the other data in the chart. i.e. monthly salaries and only monthly salaries are within the column.

Keys

- A **key** is the tool to unlock access to tables. Knowledge of the key enables us to locate specific records, and cross the relationships between tables.
- Certain fields may be designated as keys, which means that searches for specific values of that field will use indexing.
- A **candidate key** is any field, or combination of fields, that uniquely identifies a record. The field/s of the candidate key must contain unique values (if the values were duplicated, they would be no longer identify unique records), and cannot contain a null value.
- A **primary key** is the candidate key that has been chosen to identify unique records in a particular table.
- A **foreign key** is a reference to a key in another table. A relationship between two tables is created by creating a common field to the two tables.
- Foreign keys allow us to ensure what is called "referential integrity". This means a foreign key that contains a value must refer to an existing record in the related table. For example, take a look at these two tables:

Dimensional model

The dimensional model is a specialized adaptation of the relational model used to represent data in data warehouses in a way that data can be easily summarized using online analytical processing, or OLAP queries. In the dimensional model, a database schema consists of a single large table of facts that are described using dimensions and measures. A dimension provides the context of a fact (such as who participated, when and where it happened, and its type) and is used in queries to group related facts together. Dimensions tend to be discrete and are often hierarchical; for example, the location might include the building, state, and country. A measure is a quantity describing the fact, such as revenue. It is important that measures can be meaningfully aggregated—for example, the revenue from different locations can be added together.

In an OLAP query, dimensions are chosen and the facts are grouped and aggregated together to create a summary.

The dimensional model is often implemented on top of the relational model using a star schema, consisting of one highly normalized table containing the facts, and surrounding denormalized tables containing each dimension. An alternative physical implementation, called a snowflake schema, normalizes multi-level hierarchies within a dimension into multiple tables.

A data warehouse can contain multiple dimensional schemas that share dimension tables, allowing them to be used together. Coming up with a standard set of dimensions is an important part of dimensional modeling.

Its high performance has made the dimensional model the most popular database structure for OLAP.

Post-relational database models

Products offering a more general data model than the relational model are sometimes classified as *post-relational*.[2] Alternate terms include "hybrid database", "Object-enhanced RDBMS" and others. The data model in such products incorporates relations but is not constrained by E.F. Codd's Information Principle, which requires that all information in the database must be cast explicitly in terms of values in relations and in no other way

Some of these extensions to the relational model integrate concepts from technologies that pre-date the relational model. For example, they allow representation of a directed graph with trees on the nodes. The German company *sones* implements this concept in its GraphDB.

Some post-relational products extend relational systems with non-relational features. Others arrived in much the same place by adding relational features to pre-relational systems. Paradoxically, this allows products that are historically pre-relational, such as PICK and MUMPS, to make a plausible claim to be post-relational.

The resource space model (RSM) is a non-relational data model based on multi-dimensional classification.[4]

Graph model

Graph databases allow even more general structure than a network database; any node may be connected to any other node.

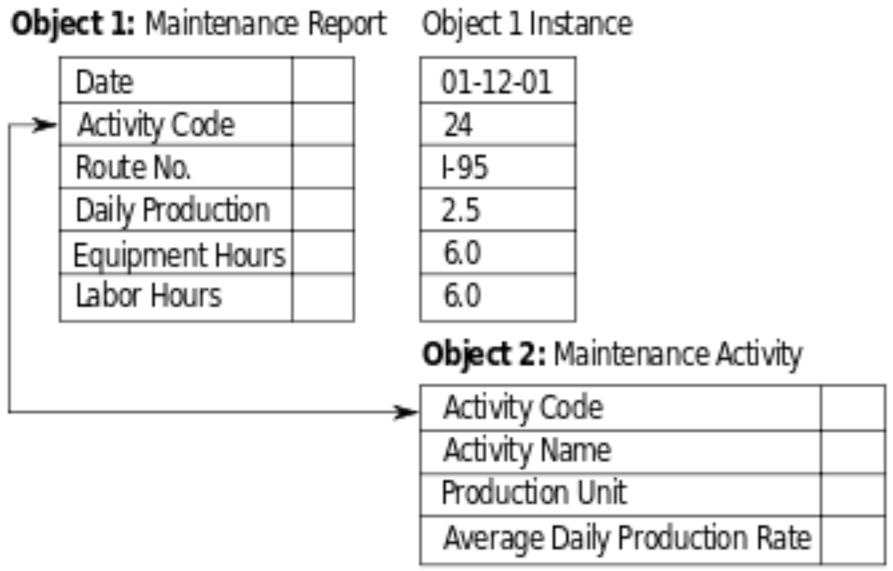
Multivalue model

Multivalue databases are "lumpy" data, in that they can store exactly the same way as relational databases, but they also permit a level of depth which the relational model can only approximate using sub-tables. This is nearly identical to the way XML expresses data, where a given field/attribute can have multiple right answers at the same time. Multivalue can be thought of as a compressed form of XML.

An example is an invoice, which in either multivalue or relational data could be seen as (A) Invoice Header Table - one entry per invoice, and (B) Invoice Detail Table - one entry per line item. In the multivalue model, we have the option of storing the data as on table, with an embedded table to represent the detail: (A) Invoice Table - one entry per invoice, no other tables needed.

The advantage is that the atomicity of the Invoice (conceptual) and the Invoice (data representation) are one-to-one. This also results in fewer reads, less referential integrity issues, and a dramatic decrease in the hardware needed to support a given transaction volume.

Object-oriented database models



Object-Oriented Model

In the 1990s, the object-oriented programming paradigm was applied to database technology, creating a new database model known as object databases. This aims to avoid the object-relational impedance mismatch - the overhead of converting information between its representation in the database (for example as rows in tables) and its representation in the application program (typically as objects). Even further, the type system used in a particular application can be defined directly in the database, allowing the database to enforce the same data integrity invariants. Object databases also introduce the key ideas of object programming, such as encapsulation and polymorphism, into the world of databases.

A variety of these ways have been triedfor storing objects in a database. Some products have approached the problem from the application programming end, by making the objects manipulated by the program persistent. This typically requires the addition of some kind of query language, since conventional programming languages do not have the ability to find objects based on their information content. Others have attacked the problem from the database end, by defining an object-oriented data model for the database, and defining a database programming language that allows full programming capabilities as well as traditional query facilities.

Object databases suffered because of a lack of standardization: although standards were defined by ODMG, they were never implemented well enough to ensure interoperability between products. Nevertheless, object databases have been used successfully in many applications: usually specialized applications such as engineering databases or molecular biology databases rather than mainstream commercial data processing. However, object database ideas were picked up by the relational vendors and influenced extensions made to these products and indeed to the SQL language.

An alternative to translating between objects and relational databases is to use an object-relational mapping (ORM) library.

References

1. E.F. Codd (1970). "A relational model of data for large shared data banks". In: *Communications of the ACM archive*. Vol 13. Issue 6(June 1970). pp.377-387.
2. *Introducing databases* by Stephen Chu, in Conrick, M. (2006) *Health informatics: transforming healthcare with technology*, Thomson, ISBN 0-17-012731-1, p. 69.
3. *Date, C. J. (June 1, 1999). "When's an extension not an extension?". Intelligent Enterprise.* ***2*** *(8).*
4. *Zhuge, H. (2008). The Web Resource Space Model. Web Information Systems Engineering and Internet Technologies Book Series.*
5. *Springer. ISBN 978-0-387-72771-4.*

Source: https://en.wikipedia.org/wiki/Database_model

Comparison Of NoSQL Database Management Systems And Models"

This course primarily covers traditional databases, but this article gives an overview of some non-traditional databases and classifies non-SQL databases according to their operational model. Non-SQL databases are schema-less, and not based on a single data model.

Introduction

NoSQL databases try to offer certain functionality that more traditional relational database management systems do not. Whether it is for holding simple key-value pairs for shorter lengths of time for caching purposes, or keeping unstructured collections (e.g. collections) of data that could not be easily dealt with using relational databases and the *structured query language* (SQL) – they are here to help.

In this DigitalOcean article, we are going to try to introduce you to various popular NoSQL database management systems and explain their purposes and functionality, so that you can decide which one to use, or if they even suit your application's needs -- *at all*.

Glossary

1. Database Management Systems

2. NoSQL Database Management Systems

    1. Key / Value Based
    2. Column Based
    3. Document Based
    4. Graph Based

3. Key / Value Based NoSQL Database Management Systems

    1. Popular Key / Value Based Databases
    2. When To Use

Database Management Systems

Databases are logically modeled storage spaces for all kinds of different information (data). Each database, other than schema-less ones, have a model which provides structure for the data being dealt with. Database management systems are applications (or libraries) which manage databases of various shapes, sizes, and sorts.

Note: To learn more about Database Management Systems, check out our article: Understanding Databases.

NoSQL Database Management Systems

In the past decade or so, relational database management systems have been the choice of many developers and system administrators for a variety of applications, for a variety of reasons. Despite not being exactly flexible, the powerful nature of many RDBMS allowed complex database set-ups to be created, queried and used. This was more than enough for many requirements, since it was not until long ago that different needs started to rise.

The term "NoSQL" was coined over a decade ago, funnily enough as a name to yet-another relational database. However, this database had a different idea behind it: eliminating the use of the standardised SQL. In the next years to come, others picked up and continued to grow this thought, by referring to various other non-relational databases as **NoSQL databases**.

By design, NoSQL databases and management systems are relation-less (or schema-less). They are not based on a single model (e.g. *relational model* of RDBMSs) and each database, depending on their target-functionality, adopt a different one.

There are almost a handful of different operational models and functioning systems for NoSQL databases.:

- **Key / Value:**

e.g. Redis, MemcacheDB, etc.

- **Column:**

e.g. Cassandra, HBase, etc.

- **Document:**

e.g. MongoDB, Couchbase, etc

- **Graph:**

e.g. OrientDB, Neo4J, etc.

In order to better understand the roles and underlying technology of each database management system, let's quickly go over these four operational models.

Key / Value Based

We will begin our NoSQL modeling journey with key / value based database management simply because they can be considered the most basic and backbone implementation of NoSQL.

These type of databases work by matching keys with values, similar to a dictionary. There is no structure nor relation. After connecting to the database server (e.g. Redis), an application can state a key (e.g. the_answer_to_life) and provide a matching value (e.g. 42) which can later be retrieved the same way by supplying the key.

Key / value DBMSs are usually used for quickly storing basic information, and sometimes not-so-basic ones after performing, for example, a CPU and memory intensive computation. They are extremely performant, efficient and usually easily scalable.

Note: When it comes to computers, a *dictionary* usually refers to a special sort of data object. They constitutes of arrays of collections with individual keys matching values.

Column Based

Column based NoSQL database management systems work by advancing the simple nature of key / value based ones.

Despite their complicated-to-understand image on the internet, these databases work very simply by creating collections of one or more key / value pairs that match a record.

Unlike the traditional defines schemas of relational databases, column-based NoSQL solutions do not require a pre-structured table to work with the data. Each record comes with one or more columns containing the information and each column of each record can be different.

Basically, column-based NoSQL databases are two dimensional arrays whereby each key (i.e. row / record) has one or more key / value pairs attached to it and these management systems allow very large and un-structured data to be kept and used (e.g. a record with tons of information).

These databases are commonly used when simple key / value pairs are not enough, and storing very large numbers of records with very large numbers of information is a must. DBMS implementing column-based, schema-less models can scale extremely well.

Document Based

Document based NoSQL database management systems can be considered the latest craze that managed to take *a lot* of people by storm. These DBMS work in a similar fashion to column-based ones; however, they allow much deeper nesting and complex structures to be achieved (e.g. a document, within a document, within a document).

Documents overcome the constraints of one or two level of key / value nesting of columnar databases. Basically, any complex and arbitrary structure can form a document, which can be stored using these management systems.

Despite their powerful nature, and the ability to query records by individual keys, document based management systems have their own issues and downfalls compared to others. For example, retrieving a value of a record means getting the whole lot of it and same goes for updates, all of which affect the performance.

Graph Based

Finally, the very interesting flavour of NoSQL database management systems is the graph based ones.

The graph based DBMS models represent the data in a completely different way than the previous three models. They use tree-like structures (i.e. graphs) with nodes and edges connecting each other through relations.

Similarly to mathematics, certain operations are much simpler to perform using these type of models thanks to their nature of linking and grouping related pieces of information (e.g. connected people).

These databases are commonly used by applications whereby clear boundaries for connections are necessary to establish. For example, when you register to a social network of any sort, your friends' connection to you and their friends' friends' relation to you are much easier to work with using graph-based database management systems.

Key / Value Based NoSQL Database Management Systems

Key / Value data stores are highly performant, easy to work with and they usually scale well.

Popular Key / Value Based Databases

Some popular key / value based data stores are:

- **Redis:**

In-memory K/V store with optional persistence.

- **Riak:**

Highly distributed, replicated K/V store.

- **Memcached / MemcacheDB:**

Distributed memory based K/V store.

When To Use

Some popular use cases for key / value based data stores are:

- **Caching:**

Quickly storing data for - sometimes frequent - future use.

- **Queue-ing:**

Some K/V stores (e.g. Redis) supports lists, sets, queues and more.

- **Distributing information / tasks:**

They can be used to implement *Pub/Sub*.

- **Keeping live information:**

Applications which need to keep a *state* cane use K/V stores easily.

Column Based NoSQL Database Management Systems

Column based data stores are extremely powerful and they can be reliably used to keep important data of very large sizes. Despite not being "flexible" in terms of what constitutes as data, they are highly functional and performant.

Popular Column Based Databases

Some popular column based data stores are:

- **Cassandra:**

Column based data store based on BigTable and DynamoDB.

- **HBase:**

Data store for Apache Hadoop based on ideas from BigTable.

When To Use

Some popular use cases for column based data stores are:

- **Keeping unstructured, non-volatile information:**

If a large collection of attributes and values needs to be kept for long periods of time, column-based data stores come in extremely handy.

- **Scaling:**

Column based data stores are highly scalable by nature. They can handle an awful amount of information.

Document Based NoSQL Database Management Systems

Document based data stores are excellent for keeping a lot of unrelated complex information that is highly variable in terms of structure from one another.

Popular Document Based Databases

Some popular document based data stores are:

- **Couchbase:**

JSON-based, Memcached-compatible document-based data store.

- **CouchDB:**

A ground-breaking document-based data store.

- **MongoDB:**

An extremely popular and highly-functional database.

When To Use

Some popular use cases for document based data stores are:

- **Nested information:**

Document-based data stores allow you to work with deeply nested, complex data structures.

- **JavaScript friendly:**

One of the most critical functionalities of document-based data stores are the way they interface with applications: Using JS friendly JSON.

Graph Based NoSQL Database Management Systems

Graph based data stores offer a very unique functionality that is unmatched with any other DBMSs.

Popular Graph Based Databases

Some popular graph based data stores are:

- **OrientDB:**

A very fast graph and document based hybrid NoSQL data store written in Java that comes with different operational modes.

- **Neo4J:**

A schema-free, extremely popular and powerful Java graph based data store.

When To Use

Some popular use cases for graph based data stores are:

- **Handling complex relational information:**

As explained in the introduction, graph databases make it extremely efficient and easy to use to deal with complex but relational information, such as the connections between two entities and various degrees of other entities indirectly related to them.

- **Modelling and handling classifications:**

Graph databases excel in any situation where relationships are involved. Modelling data and classifying various information in a relational way can be handled very well using these type of data stores.

NoSQL DBMSs In Comparison To Relational DBMSs

In order to draw a clear picture of how NoSQL solutions differ from relational database management systems, let's create a quick comparison list:

When To Use NoSQL Databases

- **Size matters:**

If will be working with very large sets of data, consistently scaling is easier to achieve with many of the DBMS from NoSQL family.

- **Speed:**

NoSQL databases are usually faster - and sometimes extremely speedier - when it comes to *write*s. *Read*s can also be very fast depending on the type of NoSQL database and data being queried.

- **Schema-free design:**

Relational DBMSs require structure from the beginning. NoSQL solutions offer a large amount of flexibility.

- **Automated (or easy) replications / scaling:**

NoSQL databases are growing rapidly and they are being actively built *today* - vendors are trying to tackle common issues and one of them clearly is replication and scaling. Unlike RDBMSs, NoSQL solutions can easily scale and work with(in) clusters.

- **Multiple choices:**

When it comes to choosing a NoSQL data store, there are a variety of models, as we have discussed, that you can choose from to get the most out of the database management system - depending on your data type.

Source: https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models