



UNIVERSIDAD  
DE GRANADA



# Diseño y Desarrollo de Sistemas de Información

Grado en Ingeniería Informática

## Tema 4 – Otros modelos de datos para SI

©I. J. Blanco, F. J. Cabrerizo, C. Cruz, J.A. Díaz, M. J. Martín, M.J. Rodríguez, D. Sánchez

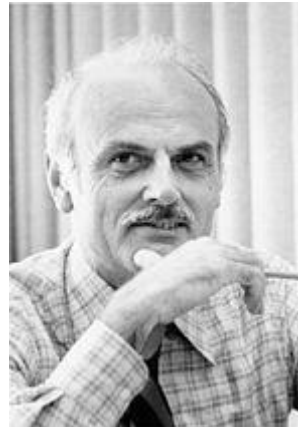
*Este documento está protegido por la Ley de Propiedad Intelectual ([Real Decreto Ley 1/1996 de 12 de abril](#)).*

*Queda expresamente prohibido su uso o distribución sin autorización del autor.*

Departamento de Ciencias de la  
Computación e Inteligencia Artificial  
<http://decsai.ugr.es>

- ☐ Concepto de Modelo de Datos
- ☐ Modelo de Datos Relacional
  - Transacciones ACID
  - Sistemas NewSQL
- ☐ Modelo Orientado a Objetos
- ☐ Modelo Objeto-Relacional
- ☐ Modelos NoSQL
  - Motivación
  - Tipos: Clave-valor, documentos, familia de columnas, grafos
  - El Teorema CAP y el modelo BASE vs ACID
- ☐ Trabajo a realizar

- ❑ Un modelo de datos es una herramienta para la **representación formal de los datos** del mundo real que son relevantes a un SI. Para ello, los modelos de datos utilizan distintos lenguajes que **describen**:
  - La **estructura** de los datos
  - Las **condiciones** que deben cumplir los datos
  - **Cómo se manejan** los datos
- ❑ Nos vamos a centrar en **Modelos Lógicos de datos** que se usan para definir y manejar datos **en un SGBD**.
- ❑ Cada modelo de datos para SGBD **ofrece determinados servicios y capacidades de representación** (tipos de datos, relaciones y restricciones) y **de uso** (consistencia, disponibilidad, tolerancia a partición, facilidad de ciertos tipos de consulta, escalabilidad, etc.)



## Edgar Frank "Ted" Codd

### Information Retrieval

P. BAXENDALE, Editor

### A Relational Model of Data for Large Shared Data Banks

E. F. Codd  
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for non-inferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the “connection trap”).

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted

### Relational Model

Activity Code	Activity Name
23	Patching
24	Overlay
25	Crack Sealing

Key = 24

Activity Code	Date	Route No.
24	01/12/01	I-95
24	02/08/01	I-66

Date	Activity Code	Route No.
01/12/01	24	I-95
01/15/01	23	I-495
02/08/01	24	I-66

- ❑ Los **SGBD relacionales** están **pensados para garantizar transacciones ACID** (Atomicity, Consistency, Isolation, Durability), que garantizan la consistencia y estabilidad de las operaciones pero requieren una gestión de bloqueo sofisticada:

**Atomicidad**: es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto, no puede quedar a medias ante un fallo del sistema. Por ejemplo, en el caso de una transacción bancaria o se ejecuta tanto el depósito como la deducción o ninguna acción es realizada (ya visto).

**Consistencia**: *Integridad*. La propiedad de consistencia sostiene que cualquier transacción llevará a la base de datos desde un estado válido a otro también válido (de acuerdo con las restricciones y reglas establecidas en la BD).

**Aislamiento**: La ejecución concurrente de dos transacciones da como resultado un estado del sistema que sería el mismo que si se hubieran ejecutado secuencialmente.

**Durabilidad**: *Persistencia*. Es la propiedad que asegura que una vez finalizada una transacción, ésta persistirá y no se perderán los cambios aunque falle el sistema.

- ❑ La “ACIDez” de las transacciones es un requisito imprescindible en aplicaciones de gestión para gobiernos, empresas y organismos financieros.
- ❑ Por ello, los SGBD relacionales clásicos (SGBDR o RDBMS en inglés, también llamados “Sistemas SQL”) siguen siendo los más empleados en estos ámbitos (y en general en el ámbito de los SI en la actualidad), con SGBD como Oracle, MySQL, Microsoft SQL Server, MariaDB, PostgreSQL, etc.
- ❑ Sin embargo, estos sistemas también tienen limitaciones que hacen que no sean los más adecuados en determinadas circunstancias:

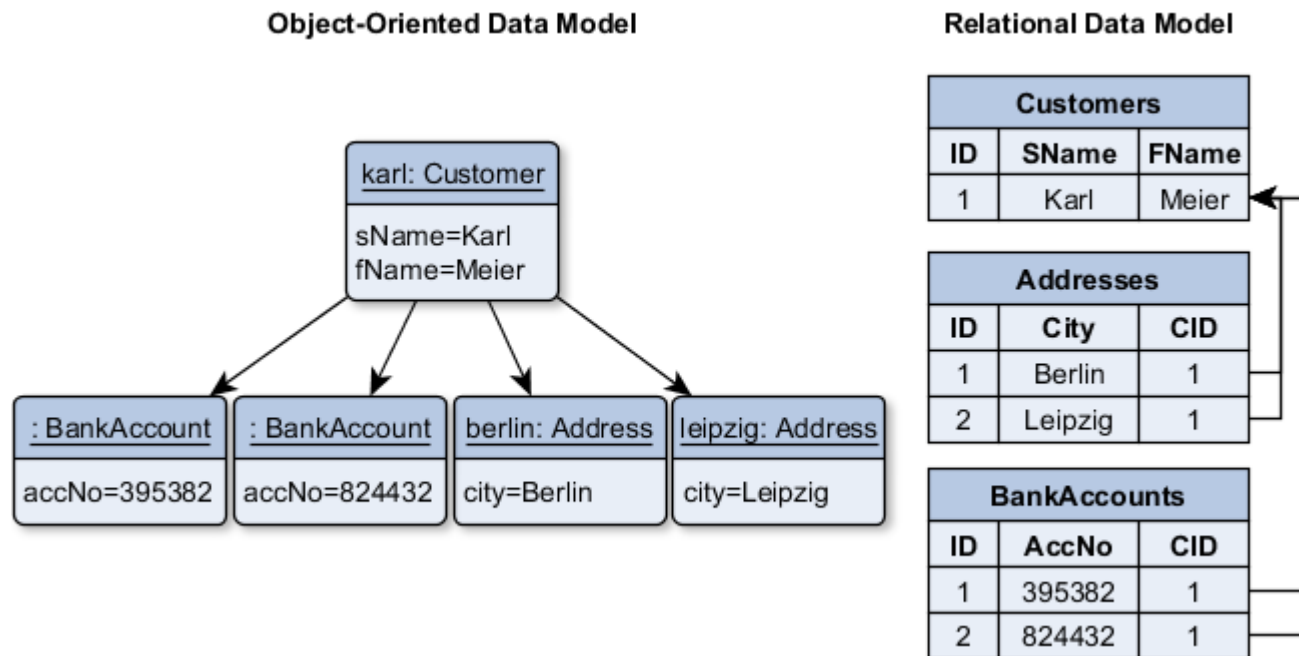
- ❑ 1. **Escalabilidad con el tráfico de transacciones.** Los SGBDR clásicos necesitan utilizar muchos recursos de tiempo y espacio para gestionar transacciones ACID, por lo que **cuando el tráfico de transacciones se hace muy intenso, se hace difícil mantener los tiempos de respuesta.** Se requiere hardware más potente y caro.
  - Los llamados “**Sistemas NewSQL**” han surgido en los últimos 10-15 años para resolver la cuestión de la Escalabilidad.
  - Son SGBD relacionales que permiten el manejo operacional de grandes volúmenes de transacciones ACID, utilizando SQL como lenguaje de consulta.



- ❑ Los **Sistemas NewSQL** están optimizados para **gestionar transacciones** que (1) son **de corta duración** (short-lived), (2) **afectan a pocos datos** a los que se puede acceder rápidamente mediante índices, y (3) **son repetitivas** (mismas operaciones con distintos datos).
- ❑ Se basan en mecanismos para la distribución de los datos y las operaciones en clusters de procesadores. Ejemplos de tecnologías:
  - **Nuevas arquitecturas.** Uso a nivel interno de arquitecturas distribuidas de **nodos *shared-nothing***. Consiguen rendimiento **distribuyendo los datos en varios nodos que se ejecutan en paralelo**, cada nodo ejecutando la parte de la transacción que le corresponde. Conforme aumenta el volumen de transacciones, pueden añadirse más nodos. **Ejemplos:** CockroachDB, Google Spanner, Clustrix, H-Store, HyPer, MemSQL, NuoDB, SAP HANA, VoltDB, etc.
  - **Transparent Sharding:** Estos sistemas proporcionan una capa de **middleware** que permite dividir automáticamente las bases de datos en varios nodos, donde en cada nodo tenemos un SGBD completo a todos los niveles. **Ejemplos:** dbShards, Scalearc, Scalebase, etc..



- ❑ 2. **Impedancia entre tablas y objetos.** Hoy en día es muy habitual el uso de lenguajes de programación orientados a objetos para programar SI. Esto conlleva varios problemas:
  - Es difícil encajar el paradigma orientado a objetos con el modelo relacional, requiriéndose una división en tablas “artificial” y dificultando la consulta de “objetos” (múltiples reuniones).



<https://phauer.com/2015/relational-databases-strength-weaknesses-mongodb/>

- Existen herramientas de “mapeo” entre objetos y relaciones (**O/R mapping**), pero introducen dificultades en la programación y depuración, y afectan también al rendimiento.  
[https://en.wikipedia.org/wiki/Object%E2%80%93relational\\_mapping](https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping)
- El modelo relacional también plantea problemas si nuestro SI debe tratar datos complejos, ya que:
  - Presenta estructuras simples (ej: imposición de 1FN).
  - Poca riqueza semántica (se pierde significado en tablas).
  - No soportan tipos definidos por el usuario (sólo dominios).
  - No soportan recursividad.
  - No admite herencia.
- Una **solución** para todos estos problemas es el **uso de SGBD basados en modelos de datos orientados a objetos o modelos objeto-relacionales**.

- ❑ Los **SGBD Orientados a Objetos** pretenden solucionar problemas que se plantean por el uso de lenguajes OO en SI:
  - Resolver el **problema de la impedancia**: los modelos de datos y las estructuras de datos de los LPOO están desacoplados
  - Conseguir **persistencia de Objetos** (más allá de los programas), con almacenamiento mas eficiente y gestión de datos en memoria secundaria
  - **Independencia** de los datos respecto de los programas
- ❑ También pretenden facilitar la representación y manejo de los datos en SI en los que los datos son altamente estructurados y/o tienen asociados comportamientos complejos

## Manifiesto de los SGBD Orientados a Objetos.

- ❑ Describe características que deben cumplir los SGBD OO, dividiéndolas en tres tipos:
  - **Obligatorias:** imprescindibles para SGBD OO.
  - **Opcionales:** pueden añadirse para mejorar el sistema.
  - **Abiertas:** posibilidades adicionales aceptables, a aplicar a juicio del diseñador

<http://www.cs.cmu.edu/afs/cs/user/clamen/OODBMS/Manifiesto/htManifiesto/Manifiesto.html>

## Manifiesto de los SGBD Orientados a Objetos. Características obligatorias

### □ Al ser OO

- Objetos complejos.
- Identidad del objeto.
- Encapsulamiento.
- Tipos o clases.
- Herencia.
- Polimorfismo, sobrecarga y vinculación dinámica.
- Extensibilidad.
- Completitud de cálculos (lenguaje de propósito general).

### □ Al ser un SGBD

- Persistencia.
- Gestión del almacenamiento secundario: nivel interno.
- Concurrencia.
- Recuperación ante fallos.
- Lenguajes ad-hoc para manipulación.

<http://www.cs.cmu.edu/afs/cs/user/clamen/OODBMS/Manifiesto/htManifiesto/Manifiesto.html>

## Manifiesto de los SGBD Orientados a Objetos.

### Características opcionales

- Herencia múltiple.
- Verificación e inferencia del tipo.
- Distribución.
- Transacciones de diseño.
- Versionado

### Opciones abiertas

- Paradigma de programación.
- Sistema de representación (tipos atómicos y constructores).
- Sistema de tipos.
- Uniformidad (¿todo objetos?).

<http://www.cs.cmu.edu/afs/cs/user/clamen/OODBMS/Manifiesto/htManifiesto/Manifiesto.html>

## Productos y estándares SGBDOO puros

### ❑ Estándares:

ODMG-93 (1994), ODMG V.2.0 (1997), ODMG V.3.0 (2000).

### ❑ Productos

- ObjectStore de Object Design. Persistencia de objetos en C++, Java.
- O2 de O2, Leeluse et al. (1988). Lenguajes: C++, lenguajes de consulta (O2SQL) y programación (O2C) propios. Java.
- Gemstone de Servi Logic, Meier y Stone (1987) Persistencia de objetos en Smalltalk. Soporta también C++ y Java.
- POET de Poet Corporation. Persistencia de objetos C++, Java.
- db4o, Matisse...

[https://en.wikipedia.org/wiki/Comparison\\_of\\_object\\_database\\_management\\_systems](https://en.wikipedia.org/wiki/Comparison_of_object_database_management_systems)



## Cuándo usar SGBD Orientados a Objetos

- ❑ Su uso puede considerarse cuando nuestro SI plantea algunas de las siguientes necesidades:
  - Manejo de **objetos altamente estructurados**
  - Manejo de **objetos con comportamientos complejos**
  - Manejo de un **gran número de tipos de datos diferentes** que pueden representarse como objetos, especialmente cuando hay relaciones de herencia entre ellos
  - Manejo de un **gran número de relaciones** entre los objetos
  - Se requiere una **visión lógica de la BD como** un conjunto de **objetos**
  - Se requiere **dar persistencia a objetos** y llevar a cabo un **acceso rápido** a los mismos

- ❑ Los SGBD orientados a objetos “puros” no han tenido una gran implantación y tienen un uso minoritario.
- ❑ En su lugar, y como modelo intermedio entre los puramente OO y los relacionales, surgen los **SGBD Objeto-Relacionales**.
- ❑ Los SGBD Objeto-Relacionales (O-R) son sistemas de gestión de bases de datos **basados en el modelo relacional**, pero que **extienden dicho modelo con características propias de la orientación a objetos**.
- ❑ Su uso es apropiado en general en las mismas circunstancias que hemos indicado para SGBD Orientados a Objetos puros, aunque no ofrecen la misma eficiencia. Esto se debe a que la representación interna se hace realmente mediante un mapeo entre objetos y tablas, transparente al usuario, pero que limita la rapidez en las consultas:

- ❑ Entre las características que pueden ofrecer los **SGBD O-R** existentes encontramos que **tanto las tablas como las filas y columnas pueden tratarse como objetos**. Esto permite:
  - **Romper la 1FN** permitiendo que los dominios de atributos sean clases y que en las instancias de una relación aparezcan objetos.
  - **Permitir tablas anidadas**. Se pueden definir columnas como arrays o vectores multidimensionales y esquemas de tablas.
  - Permitir la definición de **métodos asociados a las clases**, que pueden utilizarse en el lenguaje de consulta, para lo cual se requiere el soporte de un lenguaje de programación en el SGBD.
  - Permitir **herencia** en la definición de clases (subtipos y subtablas).
  - Permitir el uso de un **lenguaje de alto nivel** (extensión de SQL) para las consultas.

## Productos y estándares SGBD Objeto-Relacionales

### ❑ Estándares

SQL: 1999, SQL: 2003.

### ❑ Productos

- Oracle, PostgreSQL, IBM DB2, ...

[https://en.wikipedia.org/wiki/Comparison\\_of\\_object%E2%80%93relational\\_database\\_management\\_systems](https://en.wikipedia.org/wiki/Comparison_of_object%E2%80%93relational_database_management_systems)

- Cada uno de estos SGBD ofrece distintas características, implementando parte de los estándares mencionados.

- ❑ Oracle permite la creación de “Object Types” (clases) especificando tanto sus propiedades como métodos.

```
CREATE TYPE person_typ AS OBJECT (
  idno      NUMBER,
  first_name VARCHAR2(20),
  last_name  VARCHAR2(25),
  email      VARCHAR2(25),
  phone      VARCHAR2(20),
  MAP MEMBER FUNCTION get_idno RETURN NUMBER,
  MEMBER PROCEDURE display_details ( SELF IN OUT NOCOPY person_typ ));
/
```

Para indicar las propiedades y métodos

```
CREATE TYPE BODY person_typ AS
  MAP MEMBER FUNCTION get_idno RETURN NUMBER IS
  BEGIN
    RETURN idno;
  END;
  MEMBER PROCEDURE display_details ( SELF IN OUT NOCOPY person_typ ) IS
  BEGIN
    -- use the PUT_LINE procedure of the DBMS_OUTPUT package to display details
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(idno) || ' ' || first_name || ' ' || last_name);
    DBMS_OUTPUT.PUT_LINE(email || ' ' || phone);
  END;
END;
/
```

Para definir el código de los métodos con PL/SQL (ya lo veremos en la Práctica 3)

- ❑ Oracle permite la creación de “Object Types” (clases) especificando tanto sus propiedades como métodos.

```
CREATE TYPE person_typ AS OBJECT (
  idno      NUMBER,
  first_name VARCHAR2(20),
  last_name  VARCHAR2(25),
  email      VARCHAR2(25),
  phone      VARCHAR2(20)
  MAP MEMBER FUNCTION get_idno RETURN NUMBER,
  MEMBER PROCEDURE display_details ( SELF IN OUT NOCOPY person_typ ));
/
```

**Member:** indica que esta función accede a propiedades de una instancia del objeto.

**Map:** indica que el valor devuelto se puede usar para identificar el objeto, comparar y ordenar.

```
CREATE TYPE BODY person_typ AS
  MAP MEMBER FUNCTION get_idno RETURN NUMBER IS
  BEGIN
    RETURN idno;
  END;
  MEMBER PROCEDURE display_details ( SELF IN OUT NOCOPY person_typ ) IS
  BEGIN
    -- use the PUT_LINE procedure of the DBMS_OUTPUT package to display details
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(idno) || ' ' || first_name || ' ' || last_name);
    DBMS_OUTPUT.PUT_LINE(email || ' ' || phone);
  END;
END;
/
```

Simplemente devuelve el valor de la propiedad **idno**

- ❑ Oracle permite la creación de “Object Types” (clases) especificando tanto sus propiedades como métodos.

```
CREATE TYPE person_typ AS OBJECT (
  idno      NUMBER,
  first_name VARCHAR2(20),
  last_name  VARCHAR2(25),
  email      VARCHAR2(25),
  phone      VARCHAR2(20),
  MAP MEMBER FUNCTION get_idno RETURN NUMBER,
  MEMBER PROCEDURE display_details ( SELF IN OUT NOCOPY person_typ ));
/
```

**Member:** indica que este procedimiento accede a propiedades de una instancia del objeto.

```
CREATE TYPE BODY person_typ AS
  MAP MEMBER FUNCTION get_idno RETURN NUMBER IS
  BEGIN
    RETURN idno;
  END;
  MEMBER PROCEDURE display_details ( SELF IN OUT NOCOPY person_typ ) IS
  BEGIN
    -- use the PUT_LINE procedure of the DBMS_OUTPUT package to display details
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(idno) || ' ' || first_name || ' ' || last_name);
    DBMS_OUTPUT.PUT_LINE(email || ' ' || phone);
  END;
END;
/
```

Simplemente imprime las propiedades del objeto en la salida estándar



- ❑ Los “Object Types” pueden usarse como tipos de atributos en la creación de tablas.

```
CREATE TABLE contacts (
  contact      person_typ,
  contact_date DATE );
```

Esta es la clase que hemos creado en las diapositivas anteriores. Las celdas de la columna **contact** contendrán instancias de objetos de la clase **person\_typ**.

```
INSERT INTO contacts VALUES (
  person_typ (65, 'Verna', 'Mills', 'vmills@example.com', '1-650-555-0125'),
  to_date('24 Jun 2003', 'dd Mon YYYY'));
```

Un ejemplo de inserción de un objeto de tipo **person\_typ** en una tupla. Aquí el nombre del tipo actúa como constructor, creando el objeto que se inserta.

```
SELECT c.contact.get_idno() FROM contacts c;
```

Ejemplo de uso del método `get_idno` definido antes para la clase **person\_typ**. Esta consulta nos devolvería el conjunto de identificadores de los objetos en la columna **contact** para toda la tabla **contacts**.

- ❑ Los objetos que aparecen en columnas de una sola tabla, o que se usan como propiedades de otros objetos, se denominan “Column Objects” (Objetos columna).
- ❑ Los objetos que pueden aparecer en columnas de más de una tabla, o que son representaciones persistentes de objetos que existen fuera del SGBD, se denominan “Row Objects” (Objetos fila), y deben ser almacenados en un tipo de tabla especial en la cual las filas son objetos: “Object Tables”.

**CREATE TABLE** person\_obj\_table **OF** person\_typ;

Se crea una tabla en la cual cada fila es un solo objeto fila, en este caso del tipo `person_typ` que hemos definido antes.

- ❑ Esta tabla puede verse como:
  - Una tabla de una única columna que contiene objetos tipo `person_typ`, o
  - Como una tabla clásica, en la cual las propiedades del tipo `person_typ` son las columnas de la tabla.
  - Puede usarse de ambas formas en consultas y operaciones SQL.

- ❑ Oracle permite también crear tablas anidadas que representan colecciones (conjuntos).

```
CREATE TYPE people_typ AS TABLE OF person_typ;
/
```

```
CREATE TYPE dept_persons_typ AS OBJECT (
  dept_no CHAR(5),
  dept_name CHAR(20),
  dept_mgr person_typ,
  dept_emps people_typ);
/
```

Se usa “AS TABLE OF” en lugar de simplemente “AS”. El tipo `people_typ` es un conjunto de `person_typ`.

El jefe del depto. (**dept\_mgr**) es un objeto tipo `person_typ`. Los empleados se almacenan en la tabla anidada **dept\_emps**, que es de tipo `people_typ` que acabamos de definir.

- ❑ Además, y entre otras cosas, Oracle permite:
  - Definir **referencias a objetos** (punteros lógicos) para navegar entre ellos.
  - Definir **jerarquías** de clases.
  - Definir **vectores ordenados**.
  - Crear vistas con objetos.
  - Polimorfismo a través de la jerarquía y las vistas.
  - Modificación de tipos a través de “ALTER TYPE”.
  - Usar extensiones de SQL (ya hemos visto cosas) para el manejo de objetos.
  - Acceso a objetos con Java usando JDBC, y mapeo de clases Java a tipos Oracle para facilitar la persistencia y recuperación de objetos.
  - También con C++
- ❑ Todo ello es una capa de abstracción sobre tablas, muy útil, al coste de que en el nivel interno el almacenamiento no es el óptimo (no se puede tener todo).

Para más información:

<https://docs.oracle.com/en/database/oracle/oracle-database/19/adobj/index.html>

- ❑ 3. **Poca flexibilidad en los esquemas**, que son rígidos y predefinidos, lo cual limita su evolución incluso cuando incluimos objetos. También suelen ser limitados en tamaño.
- ❑ 4. El modelo es **poco adecuado para determinados tipos de datos**, como:
  - **Datos no estructurados o semiestructurados**, como grandes cantidades de texto o imágenes, que se suelen almacenar en columnas como BLOBs (Binary Large Objects), sobre los que los SGBD relacionales no ofrecen facilidades de manejo en manipulación, consultas, etc.
  - **Datos con estructuración muy compleja**, como datos estructurados en forma de grafos, de difícil visualización y manejo en base al modelo relacional, objetos complejos, y datos tabulares en los que cada fila tiene un esquema diferente.
- ❑ 5. **Bajo rendimiento en consultas y operaciones cuando los datos están estructurados de forma compleja**, como en los casos que acabamos de mencionar.

- ❑ El término **NoSQL** ("not only SQL") representa una categoría general de SGBD que difieren de los SGBD relacionales en diferentes aspectos:
  - **No usan SQL** como el principal lenguaje de consultas.
  - Los datos almacenados **no requieren estructuras fijas**, alta flexibilidad de esquemas
  - **No siempre soportan operaciones JOIN**,
  - **No siempre garantizan** completamente **ACID** (atomicidad, consistencia, aislamiento y durabilidad),
  - Habitualmente **escalan bien horizontalmente**.
- ❑ Estos modelos aparecen asociados a nuevos requisitos en SI, relacionados con la necesidad de conseguir una **alta eficiencia en el almacenamiento de grandes cantidades de tipos de datos poco compatibles con el modelo relacional** por parte de un gran número de usuarios y aplicaciones. Google, Facebook, Amazon, Instagram, etc.

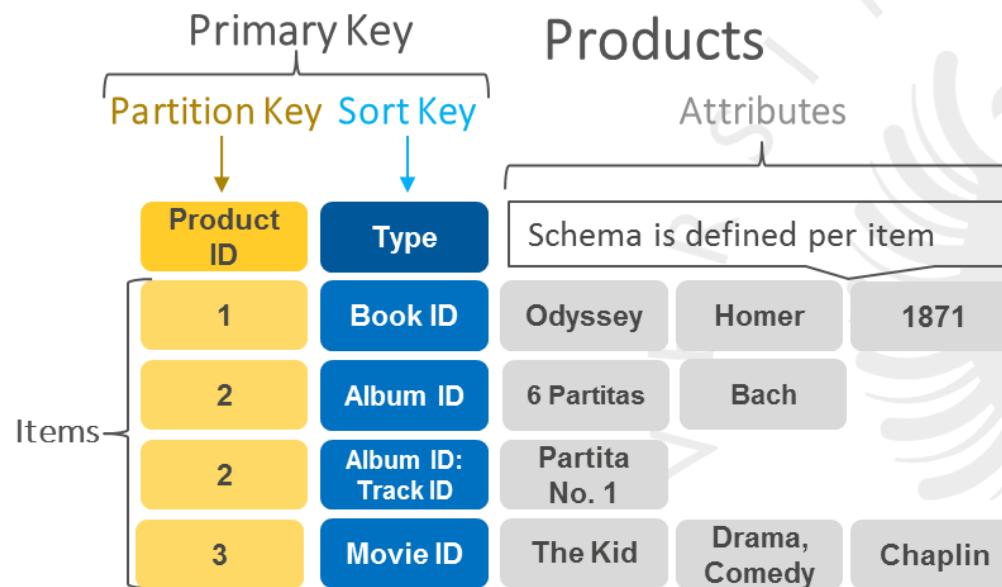
Se han diseñado para potenciar aspectos como:

- ❑ **Flexibilidad:** las bases de datos NoSQL generalmente ofrecen **esquemas flexibles** que permiten un desarrollo más rápido y más iterativo. El modelo de datos flexible hace que las bases de datos NoSQL sean ideales para datos semiestructurados y no estructurados.
- ❑ **Escalabilidad:** al igual que NewSQL, las bases de datos NoSQL generalmente están diseñadas para **escalar usando clústeres distribuidos de hardware** en lugar de escalar añadiendo servidores caros y sólidos. Algunos proveedores de la nube manejan estas operaciones fuera del alcance, como un servicio completamente administrado.
- ❑ **Alto rendimiento:** las bases de datos NoSQL están **optimizadas para modelos de datos específicos** (como documentos, clave-valor y gráficos) y patrones de acceso que permiten **un mayor rendimiento** que el intento de lograr una funcionalidad similar con bases de datos relacionales.
- ❑ **Alta funcionalidad:** las bases de datos NoSQL proporcionan **API altamente funcionales** y tipos de datos que están diseñados específicamente **para cada uno de sus respectivos modelos de datos**.



- ❑ Modelos **Clave-Valor**
- ❑ Modelos **Orientados a Columnas** (también llamados **Wide-column** o **Column-family**)
- ❑ Modelos orientados a **Documentos**
- ❑ Modelos orientados a **Grafos**

- ❑ Estos modelos almacenan **datos como un conjunto de pares clave-valor**, en los que una clave sirve como un identificador único.
- ❑ Tanto las claves como los valores pueden ser cualquier cosa en principio, desde objetos simples hasta objetos compuestos complejos.
- ❑ Son altamente divisibles y permiten el escalado horizontal a escalas que otros modelos no pueden alcanzar.



<https://aws.amazon.com/es/nosql/key-value/>

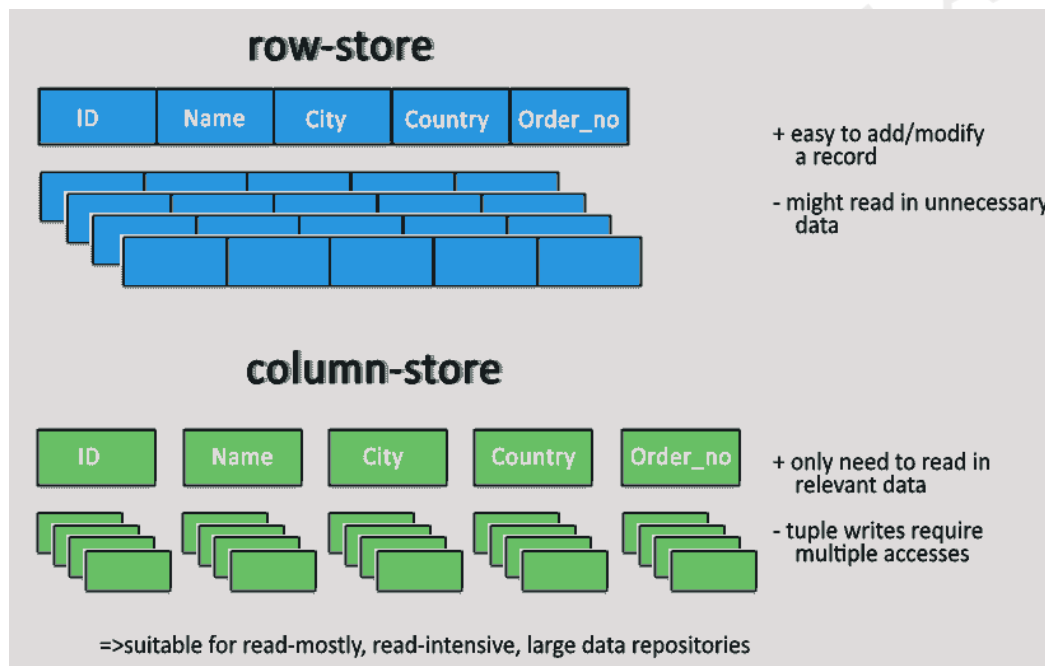
❑ Su precursor fue Amazon Dynamo

❑ Ejemplos:

- BigTable, de Google
- DynamoDB, de Amazon (reimplementación de Dynamo)
- Project Voldemort, de LinkedIn
- Riak
- Redis
- Oracle NoSQL
- Etc.

[https://en.wikipedia.org/wiki/Key%E2%80%93value\\_database](https://en.wikipedia.org/wiki/Key%E2%80%93value_database)

- ❑ Es un modelo “tabular” que en lugar de estar optimizado para almacenar físicamente registros de tipo “fila” (tupla en SGBD relacionales), está optimizado para almacenar columnas.
- ❑ Facilita operaciones de lectura rápidas por columnas (sólo se leen las columnas necesarias, en lugar de registros completos de filas). A cambio, se ralentizan las escrituras.



<https://mkhernandez.wordpress.com/2019/01/19/column-oriented-nosql-databases/>

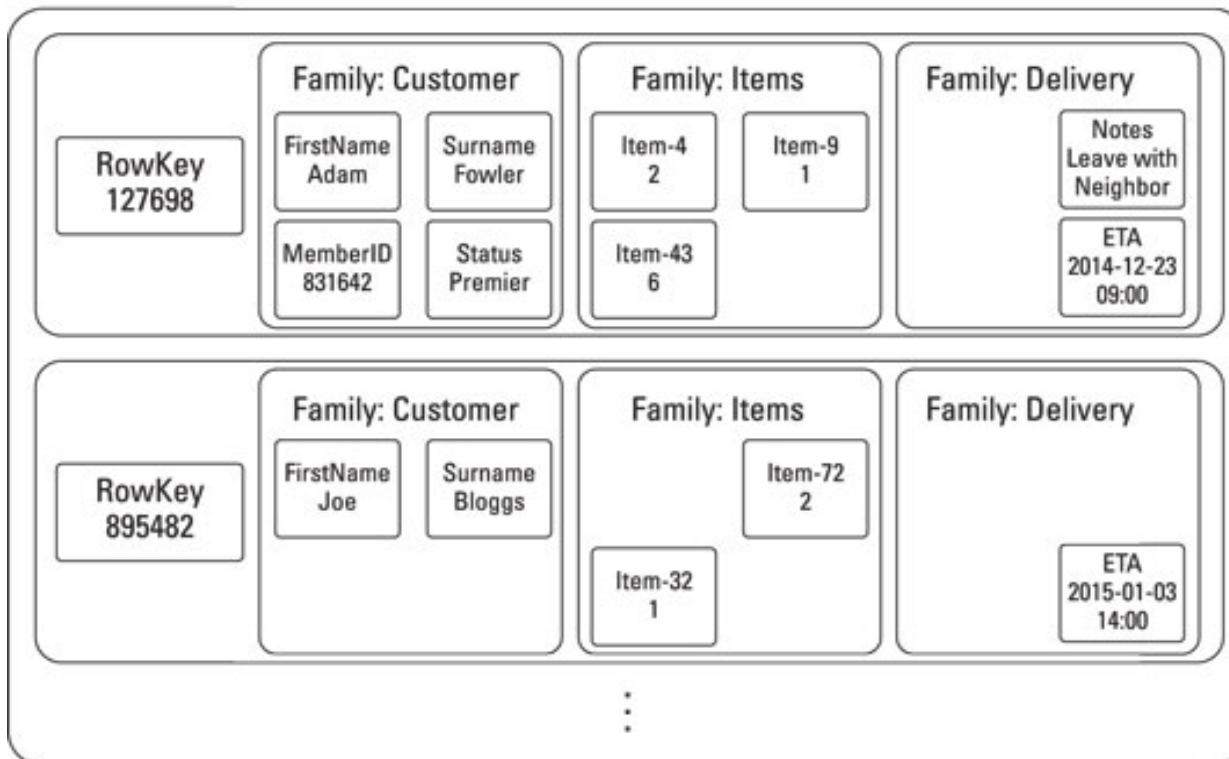
- ❑ Pero además, no todas las “filas” tienen la misma configuración de columnas, lo que da mucha más flexibilidad.

Row A	Column 1	Column 2	Column 3	...
	Value	Value	Value	
Row B	Column 2	Column 3	Column 4	...
	Value	Value	Value	

[https://www.researchgate.net/publication/264859776\\_Automated\\_Schema\\_Design\\_for\\_NoSQL\\_Databases/figures?lo=1](https://www.researchgate.net/publication/264859776_Automated_Schema_Design_for_NoSQL_Databases/figures?lo=1)

- Finalmente, es habitual que las columnas que pueden aparecer en cada fila se dividan en “familias de columnas” que representan distintos tipos de información, y que son las que están optimizadas para lectura rápida (almacenamiento por familias).

Order Table



<https://www.dummies.com/programming/big-data/columnar-data-in-nosql/>

❑ Su precursor fue Google BigTable

❑ Ejemplos:

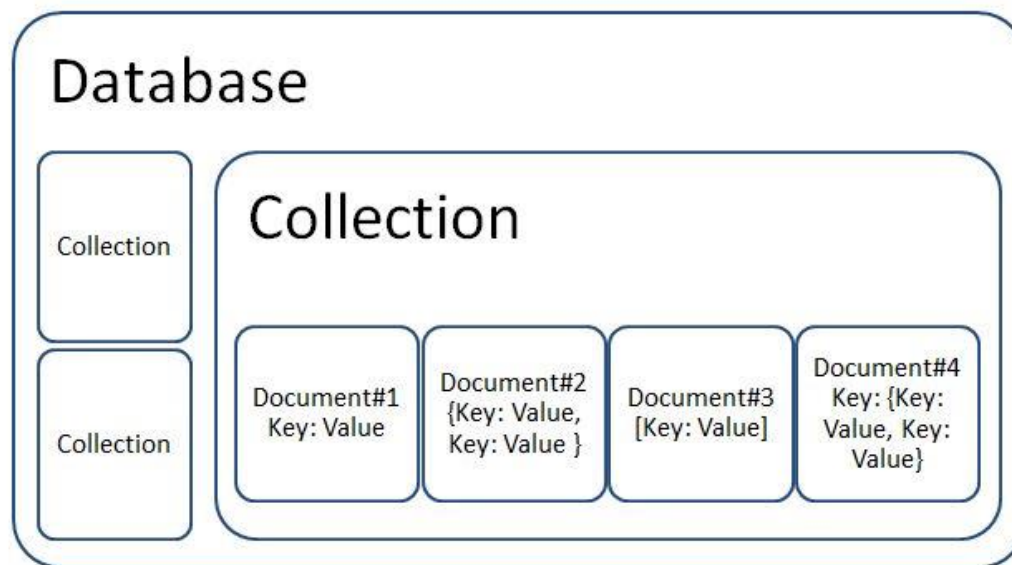
- Cassandra
- Hbase
- Accumulo
- Etc.

[https://en.wikipedia.org/wiki/Wide-column\\_store](https://en.wikipedia.org/wiki/Wide-column_store)





- ❑ Almacenan **colecciones de documentos de formato libre**, típicamente usando JSON, alternatively XML, YAML o BSON (codificación binaria de JSON), entre otras. Cada documento almacena un conjunto de pares clave-valor, pudiendo crear estructuras complejas (los valores pueden ser a su vez conjuntos de pares clave-valor). Buenas cuando hay muchos atributos poco compartidos por tuplas.
- ❑ Modelado de datos natural, amigables al programador para el desarrollo rápido de aplicaciones Web.



<https://medium.com/@mark.rethana/introduction-to-nosql-databases-c5b43f3ca1cc>

❑ La precursora fue Lotus Notes

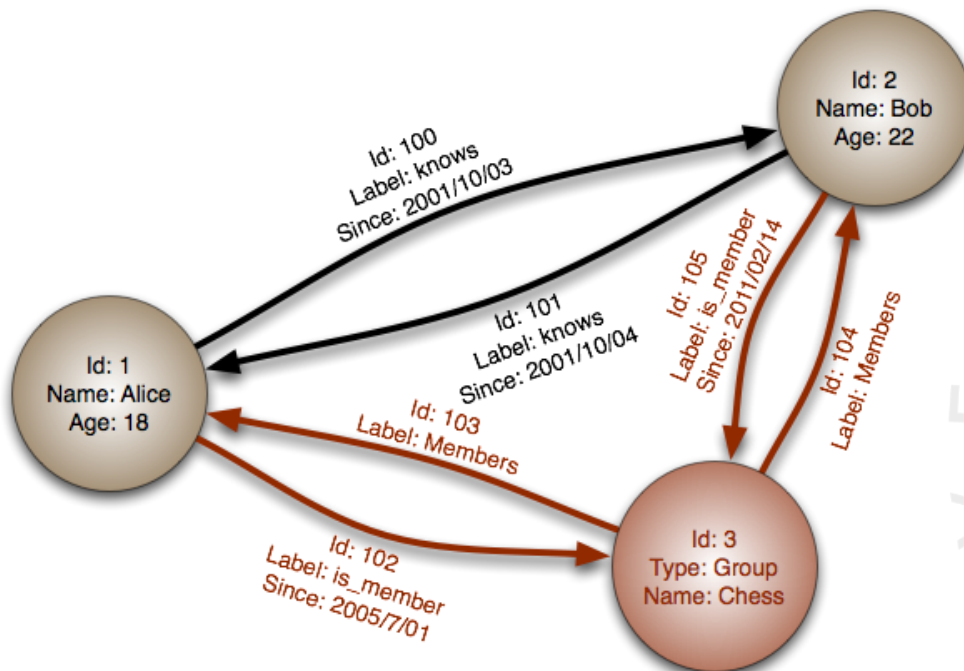
❑ Ejemplos:

- CouchDB
- MongoDB
- Etc.

[https://en.wikipedia.org/wiki/Document-oriented\\_database](https://en.wikipedia.org/wiki/Document-oriented_database)



- ❑ Modelo basado en **nodos**, **relaciones (dirigidas)** entre nodos y **conjuntos de propiedades** asociadas tanto a nodos como relaciones. Las propiedades toman la forma de pares **clave-valor**.
- ❑ Útiles cuando los datos vienen estructurados en forma de grafo de manera natural en el dominio de la aplicación, y queremos explotar de forma eficiente dichas conexiones haciendo recorridos, etc.



#### ❑ Ejemplos:

- AllegroGraph
- VertexBD
- Neo4j
- Etc.

[https://en.wikipedia.org/wiki/Graph\\_database](https://en.wikipedia.org/wiki/Graph_database)

## All in the NoSQL Family

NoSQL databases are geared toward managing large sets of varied and frequently updated data, often in distributed systems or the cloud. They avoid the rigid schemas associated with relational databases. But the architectures themselves vary and are separated into four primary classifications, although types are blending over time.



### Document databases

Store data elements in document-like structures that encode information in formats such as JSON.



Common uses include content management and monitoring Web and mobile applications.



#### EXAMPLES:

Couchbase Server, CouchDB, MarkLogic, MongoDB



### Graph databases

Emphasize connections between data elements, storing related "nodes" in graphs to accelerate querying.



Common uses include recommendation engines and geospatial applications.



#### EXAMPLES:

Allegrograph, IBM Graph, Neo4j



### Key-value databases

Use a simple data model that pairs a unique key and its associated value in storing data elements.



Common uses include storing clickstream data and application logs.



#### EXAMPLES:

Aerospike, DynamoDB, Redis, Riak



### Wide column stores

Also called table-style databases—store data across tables that can have very large numbers of columns.



Common uses include Internet search and other large-scale Web applications.



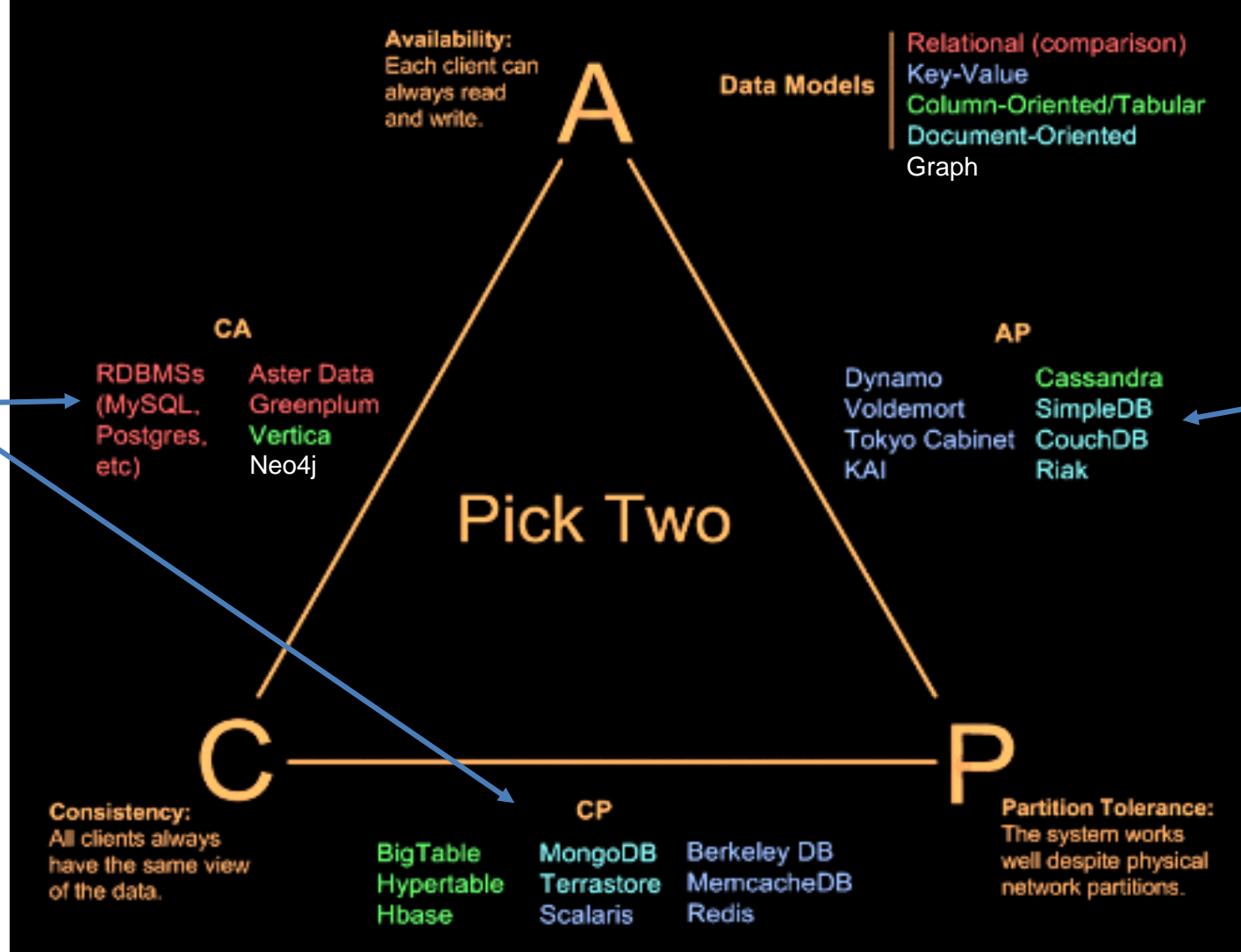
#### EXAMPLES:

Accumulo, Cassandra, HBase, Hypertable, SimpleDB

- ❑ “Es imposible para un sistema computacional distribuido ofrecer simultáneamente las siguientes tres garantías”:
  - **Consistencia (Consistency)**: todos los nodos ven los mismos datos al mismo tiempo.
  - **Disponibilidad (Availability)**: garantiza que cada petición recibe una respuesta acerca de si tuvo éxito o no. Cada cliente siempre puede leer y escribir.
  - **Tolerancia a Partición (Partition tolerance)**: el sistema tiene que seguir funcionando aunque existan fallos o caídas parciales en alguna de las partes en que se divida el sistema.

- ❑ Los sistemas NoSQL que garantizan **consistencia** cumplen ACID. Los que no, una alternativa flexible: **BASE**
  - **BA**sic availability: Siempre se obtiene una **respuesta** del sistema a una petición de datos aunque esta sea un **fallo** o que sean **inconsistentes** o estén en fase de **cambio**.
  - **S**oft-state: el **estado** del sistema **cambia** constantemente a lo largo del tiempo, incluso **aunque no haya entradas de datos** en ese periodo, debido a la **consistencia eventual**.
  - **E**ventual consistency: **eventualmente**, el **sistema se volverá consistente** a partir de que deje de recibir datos. Los datos se propagarán pero **el sistema seguirá recibiendo datos sin evaluar la consistencia** de los datos **para cada transacción** antes de avanzar a la siguiente.
- ❑ Algunos sistemas NoSQL son configurables, permitiendo indicarles si preferimos que actúen para garantizar ACID o BASE.

# Visual Guide to NoSQL Systems



ACID

BASE

Imagen modificada del original en <https://blog.nahurst.com/visual-guide-to-nosql-systems>



- ❑ Los principales **problemas de NoSQL** en general son:
  - **Falta de un lenguaje único y estándar.** Es difícil migrar de una a otra (al contrario de lo que ocurre con los SGBD relacionales, basados todos ellos en SQL), incluso dentro de los SGBD basados en el mismo modelo.
  - La falta de un lenguaje único obliga también a aprender el lenguaje específico de cada SGBD cuando se va a usar.
  - Los lenguajes están diseñados para consultas muy simples y rápidas. Las **consultas complejas requieren programación** en el SI.
  - **El mantenimiento de la consistencia** tiende a separarse de los datos, al contrario de lo que ocurre en SGBD relacionales, y **debe mantenerse en las aplicaciones**.
  - Los SGBD NoSQL están **menos orientados a la compartición de datos por parte de muchas aplicaciones**, y más pensados para una única aplicación. Esto alivia algunos de los problemas anteriores, pero limita su aplicabilidad.
  - **Poca madurez** en comparación con los SGBD relacionales, al ser más recientes.



## Comparison SQL to NoSQL Database Models

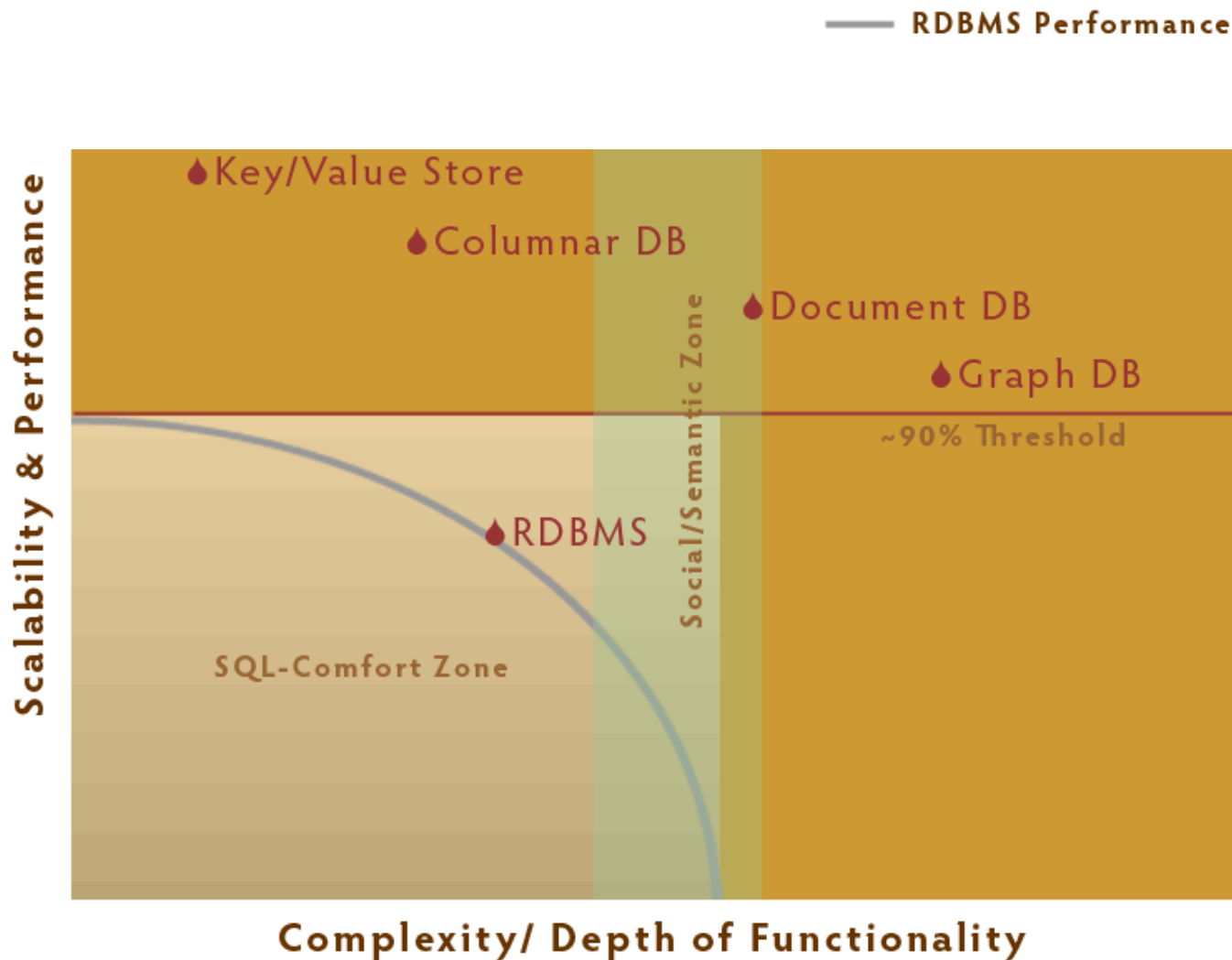
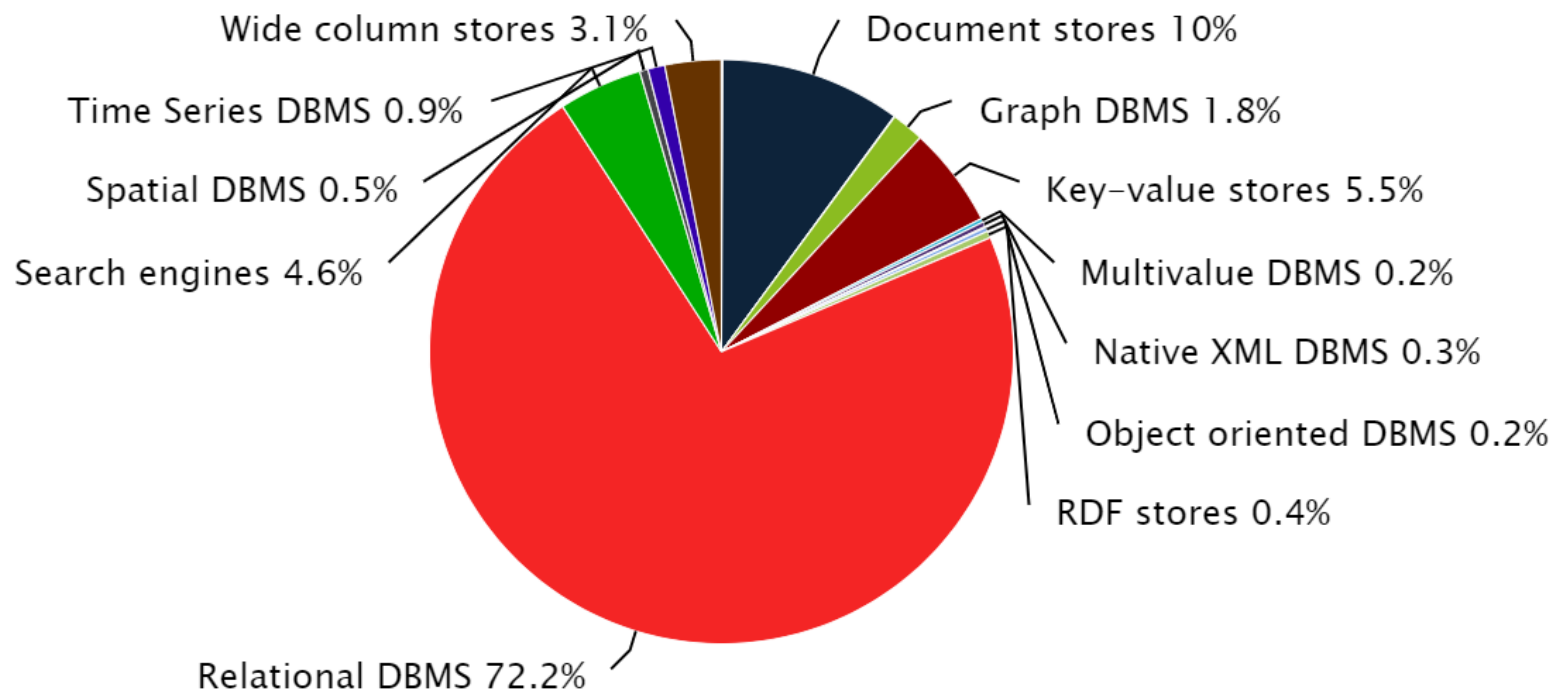













Imagen modificada del original en <http://jeffsayre.com/2010/09/17/web-3-0-smartups-moving-beyond-the-relational-database/>



© 2021, DB-Engines.com

[https://db-engines.com/en/ranking\\_categories](https://db-engines.com/en/ranking_categories)

381 systems in ranking, November 2021

Rank			DBMS	Database Model	Score		
Nov 2021	Oct 2021	Nov 2020			Nov 2021	Oct 2021	Nov 2020
1.	1.	1.	Oracle +	Relational, Multi-model 	1272.73	+2.38	-72.27
2.	2.	2.	MySQL +	Relational, Multi-model 	1211.52	-8.25	-30.12
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model 	954.29	-16.32	-83.35
4.	4.	4.	PostgreSQL + 	Relational, Multi-model 	597.27	+10.30	+42.22
5.	5.	5.	MongoDB +	Document, Multi-model 	487.35	-6.21	+33.52
6.	6.	 7.	Redis +	Key-value, Multi-model 	171.50	+0.15	+16.08
7.	7.	 6.	IBM Db2	Relational, Multi-model 	167.52	+1.56	+5.90
8.	8.	8.	Elasticsearch	Search engine, Multi-model 	159.09	+0.84	+7.54
9.	9.	9.	SQLite +	Relational	129.80	+0.43	+6.48
10.	10.	10.	Cassandra +	Wide column	120.88	+1.61	+2.13

<https://db-engines.com/en/ranking>

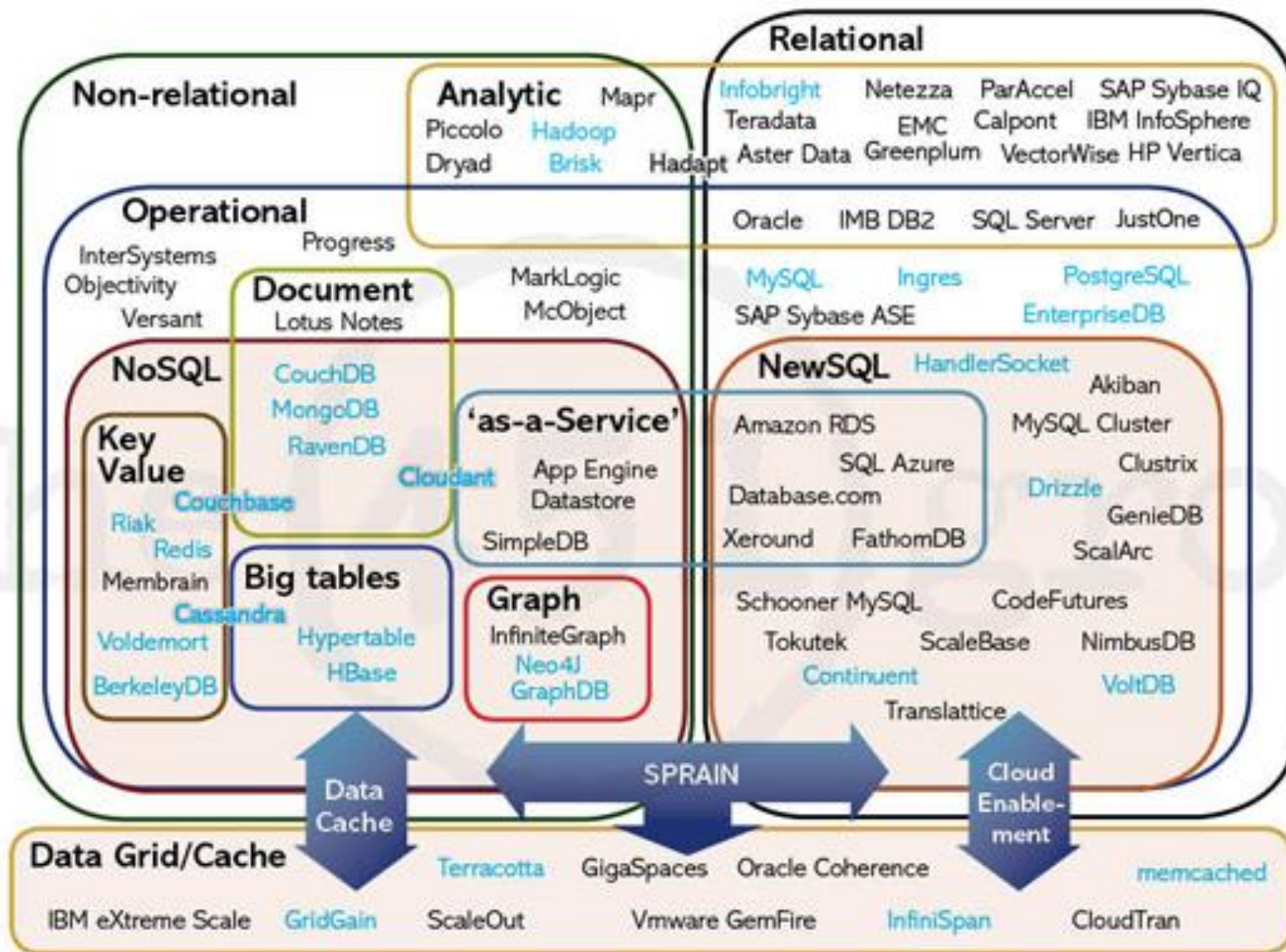
381 systems in ranking, November 2021

Rank			DBMS	Database Model	Score		
Nov 2021	Oct 2021	Nov 2020			Nov 2021	Oct 2021	Nov 2020
1.	1.	1.	Oracle +	Relational, Multi-model i	1272.73	+2.38	-72.27
2.	2.	2.	MySQL +	Relational, Multi-model i	1211.52	-8.25	-30.12
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model i	954.29	-16.32	-83.35
4.	4.	4.	PostgreSQL +	Relational, Multi-model i	597.27	+10.30	+42.22
5.	5.	5.	MongoDB +	Document, Multi-model i	487.35	-6.21	+33.52
6.	6.	↑ 7.	Redis +	Key-value, Multi-model i	171.50	+0.15	+16.08
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model i	167.52	+1.56	+5.90
8.	8.	8.	Elasticsearch	Search engine, Multi-model i	159.09	+0.84	+7.54
9.	9.	9.	SQLite +	Relational	129.80	+0.43	+6.48
10.	10.	10.	Cassandra +	Wide column	120.88	+1.61	+2.13

<https://db-engines.com/en/ranking>

Nótese que algunos de los sistemas más utilizados hoy en día permiten usar varios modelos de datos.

- ❑ Como hemos indicado, la inmensa mayoría de aplicaciones actuales no requiere del manejo de un alto volumen de datos y otras características que requieren enfoques NoSQL o NewSQL.
- ❑ Cuando se requieren, la tendencia actual es lo que se denomina **“Persistencia Políglota”**: un mismo SI hace uso de distintos modelos de datos, cada uno de ellos para tratar una parte de los datos, en función de las características de los mismos y los requisitos de uso. Es muy típico mezclar el uso de sistemas SQL y NoSQL, especialmente en proveedores de servicios a través de Web, redes sociales y plataformas de venta online (Google, Amazon, Facebook, Instagram, Twitter, etc.)
- ❑ En esa misma línea, como hemos visto, algunos **SGBD ofrecen motores de gestión de datos usando más de un modelo**, para facilitar el uso integrado en un SI, en muchos casos modelo relacional + algún modelo NoSQL.



[https://blogs.451research.com/information\\_management/2011/04/15/nosql-newsq-and-beyond/](https://blogs.451research.com/information_management/2011/04/15/nosql-newsq-and-beyond/)

<http://www.odbms.org/blog/2018/03/on-rdbms-nosql-and-newsqli-databases-interview-with-john-ryan/>

<https://www.statista.com/statistics/793854/worldwide-developer-survey-most-wanted-database/>





- ❑ El trabajo se realizará **por subgrupos** (los mismos de prácticas).
- ❑ Cada subgrupo tendrá que elegir un SGBD NoSQL de software libre e instalarlo.
- ❑ Se deberán estudiar sentencias básicas del DDL y DML del SGBD NoSQL seleccionado (el equivalente a las sentencias Create Table, Insert, Delete, Update y Select).
- ❑ Utilizando el interfaz propio del SGBD (el equivalente a SQLDeveloper de Oracle) se crearán algunas estructuras muy simples (el equivalente a dos o tres tablas en SQL) relacionadas con la temática de la práctica del grupo, aunque puedan estar fuera de los requerimientos del sistema. Ponedle imaginación.
- ❑ Asimismo, usando el mismo interfaz, se realizarán algunas operaciones de inserción, modificación, etc. y algunas consultas.



- ❑ También se buscará información sobre un mecanismo para conectarse al SGBD NoSQL desde una aplicación, aunque no debe usarse ni hay que programar nada.
- ❑ Se entregará **un único fichero .pdf por grupo** que incluirá:
  - Breve descripción de la descarga e instalación del SGBD (1-2 páginas).
  - Breve descripción del DDL y DML utilizado (no se necesita la especificación completa de la sintaxis, sólo nombrar los comandos utilizados y para qué sirven) (1-2 páginas).
  - Sentencias empleadas para la creación de estructuras, inserción/modificación/borrado de datos, y consultas (1-2 páginas).
  - Breve descripción del mecanismo de conexión al SGBD desde una aplicación (1-2 páginas).
  - Breve discusión sobre si sería adecuado para implementar el SI de la práctica (media o hasta 1 página).

❑ La evaluación se basará en:

- La **evaluación por el profesor** de la entrega, y
- La **defensa grupal** del trabajo, que consistirá en mostrar el funcionamiento de las sentencias entregadas.
- Todas las personas del grupo deben conocer todos los aspectos del trabajo realizado.

❑ El trabajo podrá entregarse en cualquier momento, y la defensa podrá realizarse también en cualquier momento antes o el mismo día del examen de la convocatoria ordinaria, acordándolo previamente con el profesor.

- ❑ Aunque sigue siendo el más utilizado, el modelo Relacional empleado por los SGBD "SQL" plantea dificultades relacionadas con la escalabilidad, impedancia con lenguajes orientados a objetos, falta de flexibilidad en la gestión de esquemas, y dificultades para el tratamiento de ciertos tipos de datos, incluyendo rendimiento.
- ❑ Para tratar de resolver estos problemas, aparecen SGBD relacionales "NewSQL" y SGBD basados en otros modelos (Orientados a Objetos, Objeto-relacionales, y NoSQL de Clave-valor, Orientados a columnas, Orientados a Documentos y Orientados a Grafos).
- ❑ Ningún modelo de datos y ningún SGBD son la panacea para todo, cada uno es adecuado para distintas cosas, por eso puede ser necesario en un SI utilizar distintos SGBD basados en distintos modelos (Persistencia Políglota)