DBMS

10 January 2018 15:16

- What is a DBMS:
  A piece of software which creates and manages databases.
  Provides an interface for users to access data in the database, has internal query engine which knows the low-level details of the database.

- CRUD operations:
  Create: insert new data into the DB.
  Retrieve: query existing data in the DB.
  Update: Modify objects in the DB.
  Delete: no, you are not a cyberman (remove data from the database).

- ACID transactions:
  Atomicity: either all actions of a transaction are carried out or none are (even if it crashes mid-transaction).
  Consistency: every transaction applied to a consistent database leaves it in a consistent state.
  Isolation: transactions are isolated, or protected, from the effects of other concurrently executed transactions.
  Durability: if a transaction completes successfully then its effects persist.

- Why are there so many DBMS options?
  No single system solves all data problems.

There are several fundamental trade-offs faced by DBMS options.
A particular DBMS might be tailored to a specific type of query/transaction.

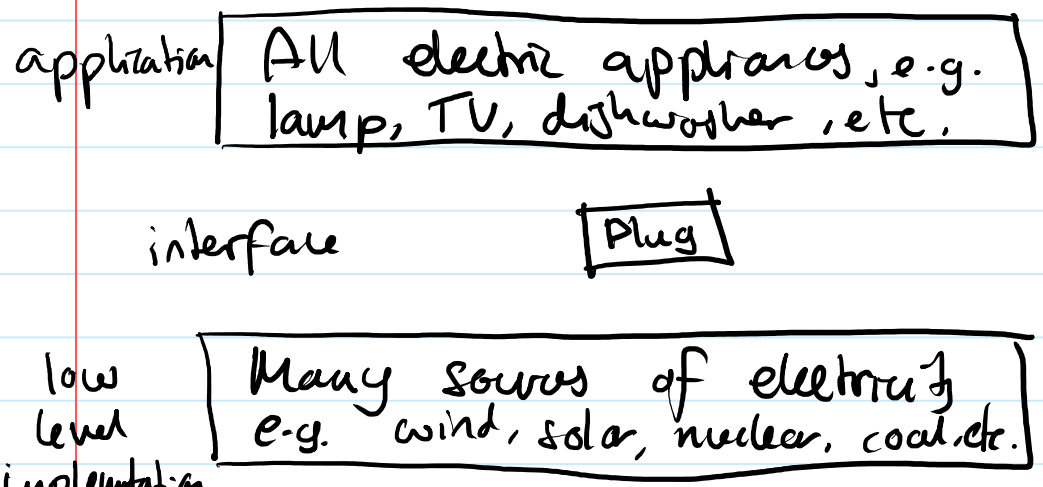
- Redundant data: data which could be deleted and then reconstructed from other data in the database.

Data redundancy is problematic when you want fast update speed as you'd have to lock all the redundant copies of data.
Data redundancy is useful when you want fast read speed, e.g. precomputing and caching results of common queries.

- A note on interfaces:

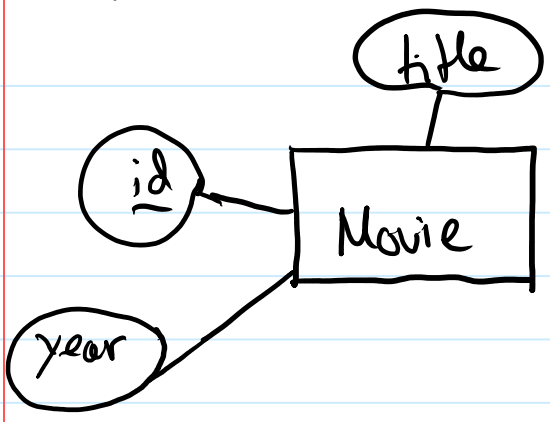
We have the model where we have an interface between the application and the low level implementation.

e.g.

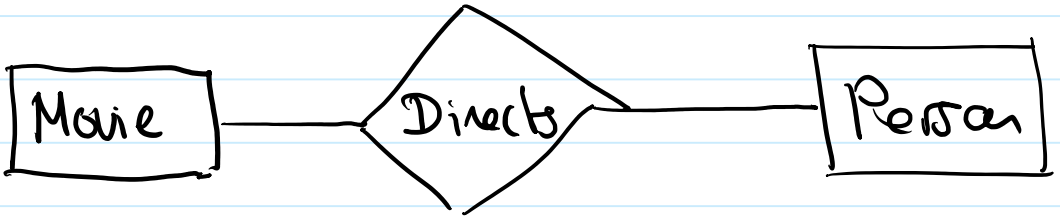


ER Modelling

10 January 2018 16:30



Entities (squares) are the nouns of our model.
Attributes (ovals) represent the properties.
A key (underlined attribute) uniquely identifies an entity instance.



Relationships (diamonds) represent the verbs in our model, the relationships are between entities.

NB: relationships can have attributes.

- Weak entity: an entity whose existence depends entirely on the existence of another entity.

- Cardinality:
  1-1: both 1-n and n-1.
  1-n: every member of T is related to at most 1 member of S.

at most 1 member of S.
n-1: every member of S is related to at most 1 member of T.
n-n: no constraint.

Relational Databases

10 January 2018 17:14

Suppose S & T are sets, a (binary) relation over S and T is any set R with $R \subseteq S \times T$. (S_i, T_i) could be in R.

- Relational Algebra:

$\sigma_p(R)$ is selection where p is a predicate on attribute values.

$\pi_x(R)$ is projection, i.e. selecting only specific columns, where x is a list of columns to select.

$\rho_{\{x \to r, z \to \omega\}}(R)$ is renaming. Self-explanatory.

$R \cup S$ is union. e.g.

R

A	B
20	10
10	5

S

A	B
20	10
5	7

$R \cup S$

A	B
20	10
10	5
5	7

$R \cap S$ is intersection, e.g.

R

¬	B
20	10
11	10
4	99

S

A	B
20	10
77	1000

$R \cap S$

A	B
20	10

R-S is difference, e.g.

R

A	B
20	10
5	6
9	9

S

A	B
20	10
2	9

R-S

A	B
5	6
9	9

R x S is product, but not cartesian product, see below:

R

A	B
20	10
11	10
4	99

S

C	D
14	99
77	100

R x S

A	B	C	D
20	10	14	99
20	10	77	100
11	10	14	99
11	10	77	100
4	99	14	99
4	99	77	100

- Natural Join:
  This is R ⋈ S, it is the set of tuples in R and S that are equal on their common attribute names.

- Anomalies:

Anomalies can be caused by data redundancy, there are 3 types of anomaly: Consider this table.

movie_id	title	year	pid	name
1	Star Warz	2001	1	Bob
2	Henry Patter	2020	2	Steve

Insertion anomalies: how can we tell if a newly inserted record is consistent with all other records. What if we want to insert a film without a director.

Deletion anomalies: we may unintentionally delete information, for example, deleting a film here will also delete its director, if we delete the last film by a director, we delete the director.

Update anomalies: what if a director's name is misspelled, we may update it correctly for one movie but not for another.

Thus a transaction performing a simple update but containing consistency checks may end up locking the entire table.

Solution: normalisation.

"The key, the whole key and nothing but the key."

SO HELP ME CODD

One problem: we have to work hard to combine information in tables (joins) to produce useful results. So, some queries may be expensive.

- Foreign Key: an attribute in one table which is the primary key in another table (or the same one).

Referential integrity: a database has referential integrity if all foreign key constraints are satisfied.

- Indices: A database index is a data structure, created and maintained within a database system, these can greatly reduce the time needed to locate records. Slows down updates though.

- Multisets:
SQL uses multisets not sets. This is because duplicates are important for aggregate functions like min, max, avg, sum, etc.

- The NULL Conundrum:
NULL is a placeholder, not a value

+	T	F	N
T	T	T	T
F	T	F	N
N	T	N	N

·	T	F	N
T	T	F	N
F	F	F	F
N	N	F	N

x	¬x
T	F
F	T
N	N

use IS NULL to check for null as x = NULL

will return NULL.
For some reasonable group by treats NULL as equal.

- Cardinality Schemas:

m:n	R(<u>X, Z</u>, A)
m:1	R(<u>X</u>, Z, A)
1:m	R(X, <u>Z</u>, A)
1:1	R(<u>X</u>, Z, A) and/or R(X, <u>Z</u>, A)

X and Z are the primary keys of the other two tables representing the two related entities.

Graph Based Databases

11 January 2018 15:27

- Transitive closure:
  Suppose R is a binary relation over S, $R \subseteq S \times S$. The transitive closure of R, R^+, is the smallest binary relation on S such that $R \subseteq R^+$ and R^+ is transitive.
  $(x,y) \in R^+ \wedge (y,z) \in R^+ \Rightarrow (x,z) \in R^+$.

- Cypher:

```
match   (node:Label) - [:RELATION] -> (node2:Label2)
where   properties
return  node.property, count(*) ;
```

NB: the group by is implicit here.

Cypher has a built in allshortestpaths() which when applied to a path pattern returns the shortest instance of that path.

e.g.

```
match path = allshortestpaths((p:Person) - [:ACTS_IN*]
- (p2:Person))
... ;
```

Note the *, this means any number of acts_in can be in the path. You could do [ACTS_IN*2] for 2 acts_in arcs.

Remember you can do (p: Person {name : "Bob"}).

NB: single = for equality check.

Document Oriented Databases

11 January 2018 16:47

Optimised for reading, so used when data is seldom updated but often read.

A document-oriented database stores data in the form of semi-structured objects.

Examples are JSON and XML, these are key-value stores.

Example JSON & XML:

JSON:

```
{"menu": {
    "id" : "file",
    "value" : "File",
    "popup": {
        "menuitem" : [
            {"value": "New", "onclick": "NewDoc()"},
            {"value": "Open", "onclick" : "OpenDoc()"}
        ]
    }
}
```

XML:

```
<menu id="file" value="File">
    <popup>
        <menuitem value="New" onclick="NewDoc()"/>
        <menuitem value="Open" onclick="OpenDoc()"/>
```

</popup>
</menu>

Distributed Databases

28 May 2018 17:29

Social data is not relational or document oriented.

OLTP: Online Transaction Processing
OLAP: Online Analytical Processing

	OLAP	OLTP
Supports	analysis	day-to-day operations
Data is	historical	current
Transactions are	reads	updates
Optimized for	reads	updates
Data redundancy	high	low
Database size	humongous	large

Use OLAP for warehoused data, OLTP for operational, dynamic data.

Data (hyper cube):
- dimensions associated with a hierarchy
- point queries facts

Human...

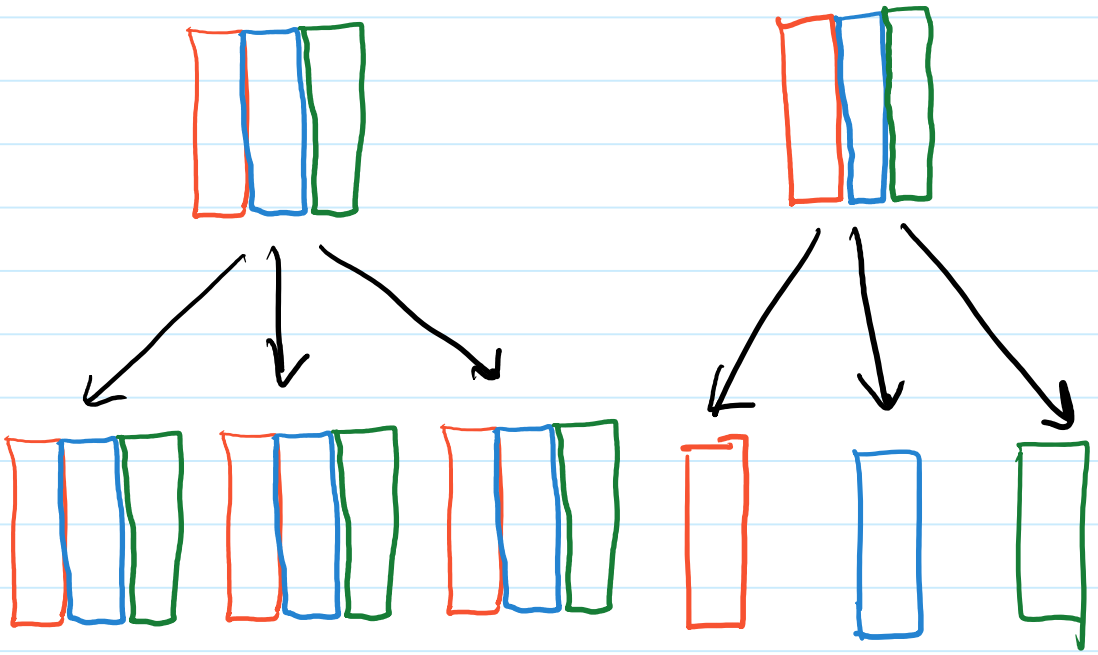
Examinable?

- Distributed Databases:

Why?
- Scalability: dataset/workload can be too much for one machine.
- Fault tolerance: service can survive failure of some machines.
- Lower latency: data can be located closer to widely distributed users.

How?

Replication or Partitioning



(NB: partitions are often replicated.)

Problems:

Hard to achieve all 3 of the CAP concepts:

- Consistency: all reads return up-to-date data.
- Availability: all clients can find a copy of the data.
- Partition tolerance: system continues to function if part of system fails.

So, in a highly distributed system we are left engineering a trade-off between availability and consistency.

This gives rise to the notion of eventual consistency, i.e. if writes stop the system would eventually be consistent.