A Gentle Introduction to Relational and Object Oriented Databases

Frank Stajano

http://www.orl.co.uk/~fms/
fstajano@orl.co.uk

ORL Technical Report TR-98-2

THE OLIVETTI & ORACLE
RESEARCH LABORATORY

(Blank even-numbered page so that "chapters" start on an odd page when printing double-sided.)

Preface

In late 1995 I was part of a workgroup that was about to embark on a new project that would eventually use a large database. The people in the group came from different backgrounds and experiences and so, to ensure that we could all agree on basic concepts and terminology, I volunteered to prepare a talk explaining the fundamentals of relational databases, a favourite topic of mine.
The talk was very well received, so I was given the job to find out about object oriented databases and to report on that as well. I spent about a month in the library doing a literature survey, at the end of which I compiled an annotated bibliography and presented a second talk.
I made this material available on my web space and then, after a few months, forgot about it. I even ended up archiving it away to CD when I was running low on disc quota. Only recently, thanks to some flattering fan mail, did I realise that my presentations were actually being used around the world in university lectures from Austria to Australia. So, to make them visible to a wider audience, I am now collecting them in an ORL technical report, which is what I should have originally done.

This report is an exact reproduction[1] of my 1995 material. It consists of three parts: a talk on relational databases, a talk on object oriented databases and a commented bibliography on object oriented databases. The talks are intended as one-hour introductions for an audience of computer professionals, assumed to be technically competent but not familiar with the topics discussed. No prior knowledge of databases is assumed for the relational database talk, and having absorbed the first talk is a sufficient precondition for understanding the second. Knowing from experience that slides often feel bare when reprinted, I have augmented them with comments echoing what you would have heard from me if you had been present at the talk.

If you wish to use or adapt these talks as your own training material, which you are free to do as long as you credit the source and give a pointer to my page, the corresponding Powerpoint presentations are freely downloadable from `http://www.orl.co.uk/~fms/db/`.
Since I am now working on other subjects, I have no plans to keep the bibliography up to date. However I hope that you'll find this material useful as an introduction and welcome any feedback.

Cambridge, UK
May 1998

[1] Note that, since then, our domain name has changed from `cam-orl.co.uk` to simply `orl.co.uk`, and the name of our laboratory has changed from *Olivetti Research Limited* to *the Olivetti & Oracle Research Laboratory*, which we pretend still fits into the ORL acronym.

(Blank even-numbered page so that "chapters" start on an odd page when printing double-sided.)

An introduction to relational databases

Frank Stajano
Olivetti Research Limited

1

This is a short introduction to the topic of relational databases. It does not require any prior knowledge of database systems. It aims to explain what the "relational" qualifier means and why relational databases are an important milestone in database technology.

Further reading:

Relational databases are now a well-understood and mature technology and as such are covered in any good database text.

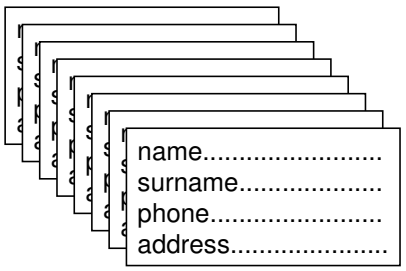
An excellent and authoritative textbook is

C. J. DATE, *An Introduction to Database Systems*, Addison-Wesley,

now in its sixth edition (1995).

Several examples in this talk come from the third edition (1981) of this book.

What is a database?



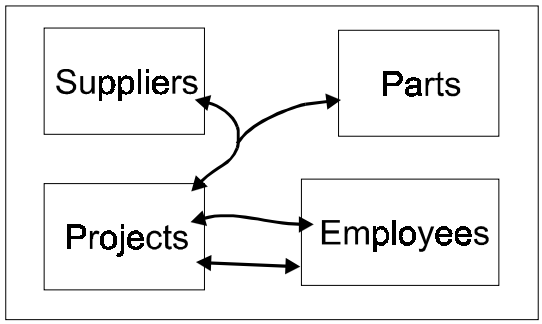
- records
- fields
- linear file of homogeneous records

What is the picture that comes to mind when we talk of a *database*?

Of course we are all familiar with concepts like records and fields. So a stack of cards like the one pictured above is perhaps your mental image of a database.

Well, if it is, please *strike it out with a big red cross*! Thinking of a linear file of homogeneous records as the archetype for a database is as reductive as thinking of a skateboard as the archetype for a roadworthy vehicle. A flat file is only a very, very restricted form of database.

What is a database?



"What is the average salary of employees who work on projects using parts that cost >$2000 and that are supplied by Rolls-Royce?"

A real database is typically a repository for heterogeneous but interrelated pieces of information.

The example above, inspired by [Date 81], describes an enterprise in which there are *employees* who work on *projects* (see the arrow) and where projects use *parts* that are supplied by *suppliers* (see the triple arrow). There is also an extra arrow between employees and projects to represent another relationship, namely that some employees are managers in charge of some projects. At this stage we are not going into details on how this information and these interconnections are actually stored in the database: we are just remarking that a database is a rather more complex object than the flat file we saw in the previous slide.

One of the most typical properties of the database is its ability to respond to complex, nested queries like the one pictured above.

What is a relational database?

- Supports relational data structure
- Has Data Manipulation Language at least as powerful as the relational algebra

(We'll have to come back to that...)

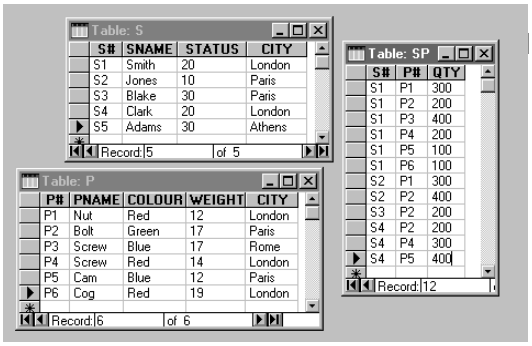
We've agreed, at least on a very general level, on what a database is. Now, what is the meaning of the "relational" qualifier?

This slide presents a formal definition, but the terminology doesn't make much sense yet, so we'll have to make a digression and then come back to this definition later.

For the moment, note that there are two requirements: one on the data structure and another on the DML.

The DML, by the way, is the programming language used to express operations that interrogate or update the database. The natural language query of the previous slide, for example, would have to be translated into the database's DML before being executed.

Example of relational db



Terms and concepts:

- tuple
- domain
- attribute
- key
- integrity rules

5

Basic terms and concepts of relational databases may be explained more easily by referring to an example (this one is borrowed from [Date 81]).

Suppliers are stored in a table called S (top left); each row of the table represents one supplier. The table is in fact equivalent to the deprecated flat file of homogeneous records of our opening slide, with each row being a record and each column being a field. Note however that the whole database is composed of several such tables, not just one. There is a table P for parts and a table SP that tells us which parts, and in what quantity, are supplied by which supplier. (The SP table thus represents one of the "arrows" in the abstract diagram we saw earlier, while the S and P tables represent "plain rectangles" — if this rather loose description makes sense to you.)

Each row is essentially a list of *n* values (*n* being the number of columns of the table), or an n-tuple in mathematical terms. We call it a *tuple* for short.

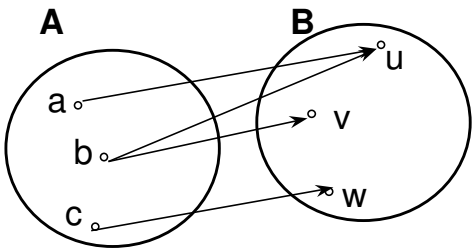
Each column represents a field of the record and is called an *attribute*.

The contents of each attribute (e.g. the "colour"attribute for a part) can only take values from a given set; this set of permissible values for a column is called a *domain*.

A *key* is an attribute or combination of attributes that uniquely identifies a row. For example the P# (part number) attribute uniquely identifies the part, in the sense that given a part number there is at most one part (one tuple) matching the part number. The part colour does not uniquely identify the part, as there could well be two parts with the same colour. Note that qualifying a set of attributes as a key is a *semantic decision* that can only be taken with knowledge of the meaning of the data in the database; it *cannot be inferred* by simply looking at the current instance of the database. If, for example, at a given point in time there were no two parts with the same colour, the colour attribute would still not be a valid key for the P table — as long as there is the possibility that parts with the same colour as other parts are inserted later in the database. Note also the case of table SP where no single attribute can be key: the S# - P# pair has to be taken as key.

Note how each tuple in the SP table refers to a tuple in S and a tuple in P by means of their respective keys. Surely a reference to, say, supplier S1, wouldn't make any sense if there were no S1 tuple in table S. *Integrity rules* express constraints that the database must satisfy in order to be internally consistent; one such rule, called *referential integrity*, is that tuples in SP can only refer to tuples in S and P *that actually exist*. As a consequence of this, when a supplier is deleted it must be ensured that, as well as deleting its S tuple, all the SP tuples referring to its shipments are removed as

Why “relational”?



r: **A** → **B**

r = { (a,u), (b,u), (b,v), (c,w)}

r ⊆ A × B

relation ⇔ table

r

a	u
b	u
b	v
c	w

6

Ok, so we’ve seen the basic concepts and the corresponding terminology. But this still doesn’t tell us why this family of databases has this strange name of “relational”.

Well, it all comes from the mathematical concept of *relation*. The illustration above depicts a relation *r* between two sets *A* and *B*. A relation, as you will recall, is a subset of the Cartesian product of the sets on which it is defined. Here r is a subset of A×B — which by the way is only another way of saying that r is a set of couples (2-tuples) with the first element taken from A and the second taken from B.

Sounds familiar? If it does, it’s because you’ve recognised the isomorphism between a mathematical relation and a database table like the ones we were dealing with on the previous slide.

So this is the origin of the name “relational”. These databases are called relational because they store their data in tables that are isomorphic to mathematical relations. And, as we’ll see, this isomorphism brings many benefits: thanks to it, the relational model of data rests on a solid mathematical foundation that allows it to exploit many useful techniques and theorems from set theory.

Relational operators

- Relational
  - select
    rel **WHERE** *boolean-xpr*
  - project
    rel **[** *attr-specs* **]**
  - join
    rel **JOIN** *rel*
  - divide by
    rel **DIVIDEBY** *rel*
- Set-based

  ∪
  rel **UNION** *rel*

  ∩
  rel **INTERSECT** *rel*

  \
  rel **MINUS** *rel*

  ×
  rel **TIMES** *rel*

7

The relational algebra is a language for manipulating relations, yielding other relations. The operators of the relational algebra are shown above. Note that, while those in the first column have been invented for database purposes, those in the second column are well-known from set theory. Because a relation is a set of tuples, we can apply the set operators to yield new relations (but note that union, intersect and minus can only be applied to pairs of relations that share the same attributes!)

The select operator takes a relation and a predicate (boolean expression) and returns the subset of all the tuples in the original relation for which the predicate evaluates to true.

The project operator takes a relation and a set of attributes (column names) and returns another relation with just the specified columns.

The join operator (this is a simplified description) takes two relations with a common attribute and makes a new relation whose attributes are the union of the attributes of the two incoming relations. Every result tuple is a combination of two source tuples that match on the common attribute.

Note that some of these operators are just there for convenience, as they can be expressed in terms of the others. Intersect, join and the esoteric divideby (not described here) are non-essential.

Example query 1

Get supplier names for suppliers who supply part P2.

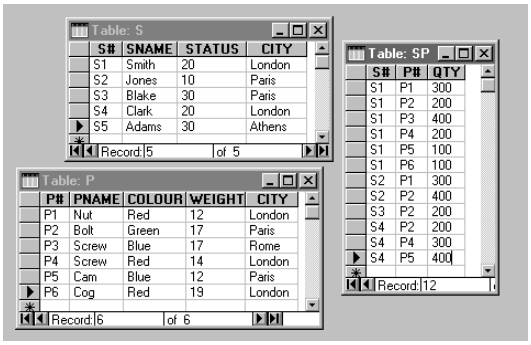


Table: S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Record: 5 of 5

Table: SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Record: 12

Table: P

P#	PNAME	COLOUR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Record: 6 of 6

```
((S JOIN SP) WHERE P# = 'P2')[SNAME]
```


Reading someone else's solved exercises does you no good.

Cover the result and try to express the query yourself using the operators previously described.

Example query 2

Get supplier numbers for suppliers who supply at least one red part.

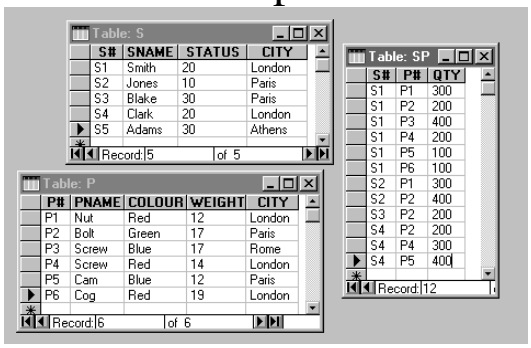


Table: S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Record: 5 of 5

Table: SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Record: 12

Table: P

P#	PNAME	COLOUR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Record: 6 of 6

```
((P WHERE COLOUR = 'RED')[P#] JOIN SP)[S#]
```

Example query 3

Get supplier names for suppliers who do not supply part P2.

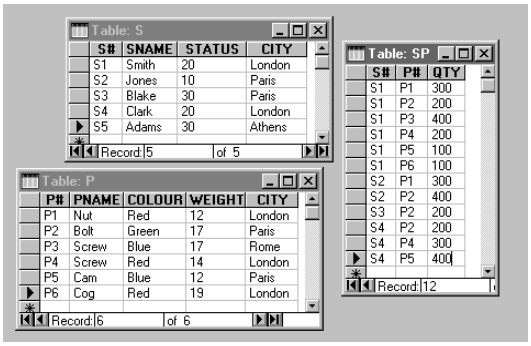


Table: S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Record: 5 of 5

Table: SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Record: 12

Table: P

P#	PNAME	COLOUR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Record: 6 of 6

```
((S[S#] MINUS (SP WHERE P#='P2')[S#])
    JOIN S)[SNAME]
```

The power of relational algebra

- Solid mathematical background
- High level: simple, powerful, expressive
- Non-procedural
- Data independent: great for optimisation

The relational algebra has many advantages.

Its mathematical background is the base of many interesting developments and theorems. Once two expressions are proved to be equivalent, a query optimiser can automatically substitute the more efficient form.

The algebra is a high level language which talks in terms of properties of sets of tuples and not in terms of for-loops. It specifies what to do without having to give details on how to do it. This is an asset.

Many relational languages

- relational algebra
- tuple-oriented relational calculus
- domain-oriented relational calculus

They have all been proved equivalent

- Implementations

  SQL, QBE, QUEL

The relational algebra is not the only available mathematical formal system for the manipulation and interrogation of a relational database. Other systems, like the ones mentioned above, have been devised which use different approaches. Fortunately, though, all these formal systems have been proved equivalent: every query that can be expressed in one of them can also be expressed in the others.

Machine implementations of these formal systems are available (sometimes with certain limitations). SQL (Structured Query Language) is the most widely used. QBE is an interesting development which uses a graphical interface to express the queries: the user fills in an “example relation” to describe the desired result.

back to our question...

What is a relational database system?

- Relational data structures
- A DML at least as powerful as the relational algebra, *even with procedural constructs taken out.*

Otherwise, just “semi-relational”

We can now go back to our definition and make sense of it.

The first requirement is that a relational DBMS must support the relational data structures; tables, of course, but also the related concepts of keys and integrity rules.

The second requirement is that the DML of the system, whatever it is, must have at least the expressive power of the relational algebra, and that it must offer this power in a declarative, non procedural form.

Systems that satisfy the first but not the second requirement are sometimes called “semi-relational”.

Are there other types of database systems?

- Hierarchical
- Network
- Relational
- Object Oriented

The above listing is in chronological order.

The first database systems (early ‘60s and before) used a hierarchical arrangement where, for example, parts were stored as sub-elements of the supplier that supplied them. This approach had several disadvantages, including the introduction of an unnecessary degree of asymmetry.

To overcome the asymmetry problem, network databases (mid ‘60s) came into being. These were mainly pointer-based structures. Querying and traversal was a low-level procedural affair.

Relational systems were born in 1969 and were soon recognised as a drastic simplification over the previous models. Everyone agreed that relational was a good thing. However it took a good decade before the commercial systems could catch up with the theory.

The late ‘80s saw the emergence of object oriented database systems as a response to the requirements of applications like CAD which dealt with many complex, nested objects. The field is still evolving very rapidly and, although everyone agrees that some degree of objectness is useful, there is no unanimous consensus on what exactly an OODBMS should be.

Executive summary

- Everything (entities and relationships) is a relation ⇒ *simplicity*
- Very high level set-oriented DML
- Backed by theory (math + its own)
- Closure ⇒ nested expressions allowed

- Limits:

  some things don't fit in relations

15

Relational database systems are an important milestone in database theory and technology.

The basic idea is simple and elegant: everything (both entities, like suppliers, and relationships, like shipments of parts by suppliers) is represented in a table. So there is only one set of operations to deal with (there are no separate operations for inserting an entity instance or a relationship instance — both are performed by inserting a tuple into a table).

The data manipulation language is high-level, declarative and set-oriented: it does not involve pointer chasing or procedural descriptions of actions to be performed in a specific sequence.

The table is isomorphic to the matematical relation, which puts the relational model of data onto firm theoretical foundations that allow the development of theorems and proofs.

The relational algebra (and any equivalent language) is closed: algebraic operators take relations as operands and return relations as result, allowing the nesting of expressions to arbitrary depths.

Still, relational databases don't solve every problem. There are some data structures that don't fit well into relations; or that, when shoehorned into relations, don't lend themselves well to querying.

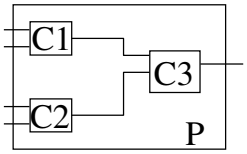
(Blank even-numbered page so that "chapters" start on an odd page when printing double-sided.)

Object Oriented Databases: An Overview

Frank Stajano
Olivetti Research Limted

The purpose of this talk is to report on the current (November 1995) status of the field of object oriented databases. It assumes an understanding of the basic concepts of relational databases.

What's wrong with relational?/1



Gate Instance	GT	GI	Parent
	2AND	C1	4AND
	2AND	C2	4AND
	2AND	C3	4AND
	4AND		

Gate Type	GT	Description
	2AND	C=A&B
	4AND	E=A&B&C&D

Pin Type	GT	PT	I/O
	2AND	A	I
	2AND	B	I
	2AND	C	O
	4AND	A	I
	4AND	B	I
	4AND	C	I
	4AND	D	I
	4AND	E	O

Wire Instance	WI	GT1	GI1	...
	W1	4AND	P	A
	W2	4AND	P	B
	W3	4AND	P	C
	W4	4AND	P	D
	W5	2AND	C1	C
	W6	2AND	C2	C
	W7	2AND	C3	C

- flattening and scattering: unnatural
- traversal: expensive
- ops on composite parts: awkward

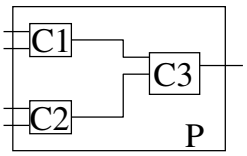
Object oriented databases emerged in the mid-80's in response to the feeling that relational databases were inadequate for certain classes of applications.

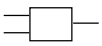
We shall expose this inadequacy by means of an example from the world of electronic CAD, borrowed from [manola94].

We have a system that represents circuits made of interconnected logic gates. The portion of diagram pictured above has three 2-input AND gates C1, C2 and C3 connected to form a bigger, 4-input AND gate P. How would we represent this in a relational database? The tables above are an attempt. We make a table describing gate types and another table with gate instances. Then a table with pin types and one with wire instances (not all columns shown because of space limitations).

There are several problems with this representation. Scattering the details of a single gate among several tables is unnatural and counterintuitive from the application's point of view, leading to complexity and inefficiency in manipulation; it also makes it not obvious which gates are part of which others. Traversal of the circuit following the logic paths is an expensive operation which requires many joins; also, these are joins of the worse type, where you join very large relations only to extract a few tuples.

What's wrong with relational?/2



Gate Type	GT	Description	Layout
	2AND	C=A&B	
	4AND	E=A&B&C&D	

- internal structure inaccessible
- no type checking on BLOB
- app must generate relocatable byte string

This is another attempt at a relational solution which uses BLOBs, or Binary Large OBjects. A BLOB is basically a byte string that is stored as is in the database; the application understands it, the database doesn't. In the pictorial representation above, the circuit diagrams in the "Layout" column stand for BLOBs.

The trouble with BLOBs is of course that you can't run queries on them: Does a 4-AND contain any 2-ANDs? The BLOB contains the answer, but not in a form that the database can access. Also, the database cannot perform any form of type checking on the BLOB.

If the BLOB represents a data structure containing pointers, the application must transform these pointers in a "relocatable" form that makes sense outside the process context before dumping the data structure to a BLOB.

The “impedance mismatch”

- Mismatch between application and DBMS
- App creates complex data structures
- App wants to access them in specific ways
- Mismatch worse if app itself is OO

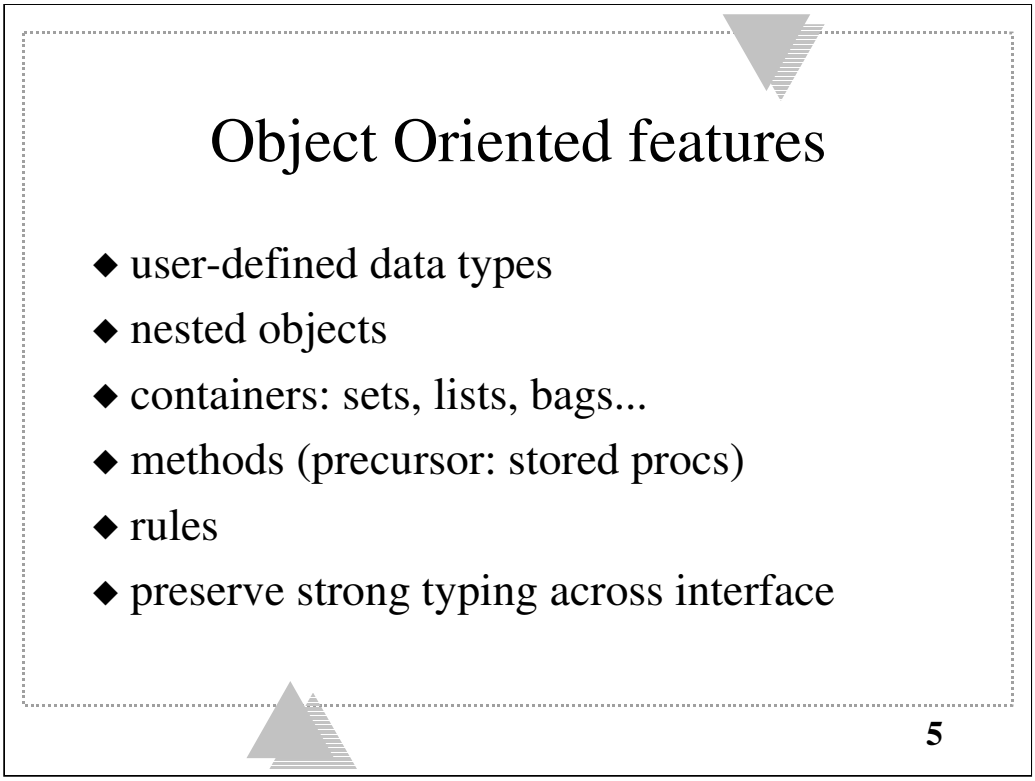
To solve this:

- Bring app and db type systems closer
- Integrate programming language and db

The example just shown is representative of a problem of relational databases described metaphorically in the literature as “impedance mismatch”. The two components whose impedances do not match are the application (in this case the CAD system) and the database.

The application is procedural, deals with one item of data at a time and creates complex data structures; the database, conversely, is declarative, deals with tuples in sets instead of individually and holds data in flat tables. Both approaches have their good reasons and strengths in their respective fields, but bringing the two paradigms together in an application that uses the database exposes this mismatch.

The solution proposed by the supporters of object oriented databases is to bring the type systems of the application and database closer to each other, so that a data item defined in the application can be stored in the database without being first “unwound”, and to integrate the programming language of the application with the database’s DML.



Relational databases don't normally let you define your own data types (although some authors, most notably [date95], argue that this is a deficiency of current implementations and not a prescription of the relational model); object databases, instead, let you define arbitrarily complex, data types like their programming language counterparts, possibly nested in is-a and/or has-a hierarchies.

Sets, lists, bags (i.e. multisets) and other containers are used, among other things, to represent the result of a query which returns several objects.

A method is a piece of code associated with an object that has privileged access to the object's state. Checking whether an object has or doesn't have a property which is "derived" from its state may sometimes be done more efficiently by invoking a specially written method than by running a query on the attributes. Stored procedures (stored in the database, that is) which can be executed on the server side were for certain aspects a precursor of this idea.

"Rules" is a generic name for code that is automatically activated when certain events occur. A form of constraint programming.

Finally, one of the most important features of object databases is that the commonality between the application's and the database's type systems preserves the strong typing in the communication between the two.

Strengths

- rich type system
- better at modelling complex objects
- better performance on certain data structures
- no impedance mismatch

The advantages of the object database approach are that applications which define a rich type system for their data structures can carry this over to the database when these data structures are made persistent. The impedance mismatch is eliminated as the database becomes a “persistence extension” to the language rather than an external service that one has to talk to through a narrow and limited interface.

Weaknesses

- procedural navigation
- querying breaks encapsulation (or is limited)
- what about closure?
- no mathematical foundation

Object systems introduce pointers linking objects to each other. This in turn promotes a procedural style of navigation among the data items that can be seen as a step backwards from the higher-level declarative approach introduced by relational systems.

While encapsulation (the idea that an object's internal state is only accessible through its designated methods) is a valuable modular technique, it presents problems with queries. Even if the Employee class has an extensive set of methods for hire_employee, change_salary, fire_employee etc., I might still not be able to answer a simple question like "who is the manager of this employee?" unless an appropriate method has been written. There is a tradeoff between giving up encapsulation (by exposing all of the object's state through methods or by allowing queries to violate encapsulation under certain circumstances) and severely restricting the class of possible queries.

Closure of the relational algebra (the property that operators take relations as their arguments and give out relations as their results, thus allowing nested queries) is an important property that is normally lost in object systems.

Finally, but perhaps most importantly, object databases do not rest on a formal mathematical base like their relational predecessors. A powerful instrument of analysis, deduction and insight is lost.

What is an OODBMS anyway?

Different things to different people

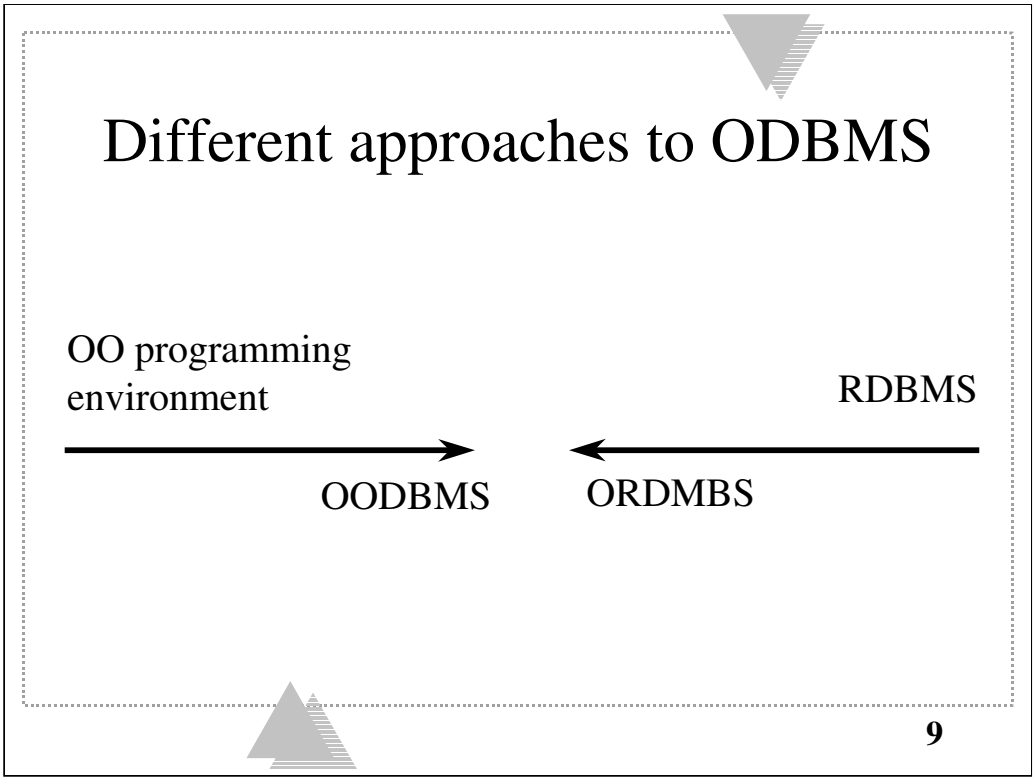
- **1989: The OO DBS Manifesto**
  - relational is old and inadequate
- **1990: Third Generation DBS Manifesto**
  - Wrong! Add classes and inheritance, but keep declarative queries and SQL
- **1995: The Third Manifesto**
  - Wrong again! Extend relational, but dump SQL

What is an OODBMS? Or, as some used to claim that no real one exists yet, what should it be? Ask this question to different people and you'll get wildly different answers. The three papers listed above were important milestones that attempted to define the issue.

The first manifesto, [atkinson89], said: ok, no one agrees on what an OODBMS is; so we are giving a set of golden rules. If a database system complies with these rules, you can call it OO. The rules stressed mostly the OO properties and considered relational databases as legacy systems not suited to the new crop of applications.

The second manifesto, [cadf90], which presented its own set of rules, was a counter-manifesto to the first. It stressed that relational systems had introduced two major developments (non procedural access and data independence) which it would be foolish to abandon in any future system. SQL, defined as "intergalactic dataspeak", had to be supported.

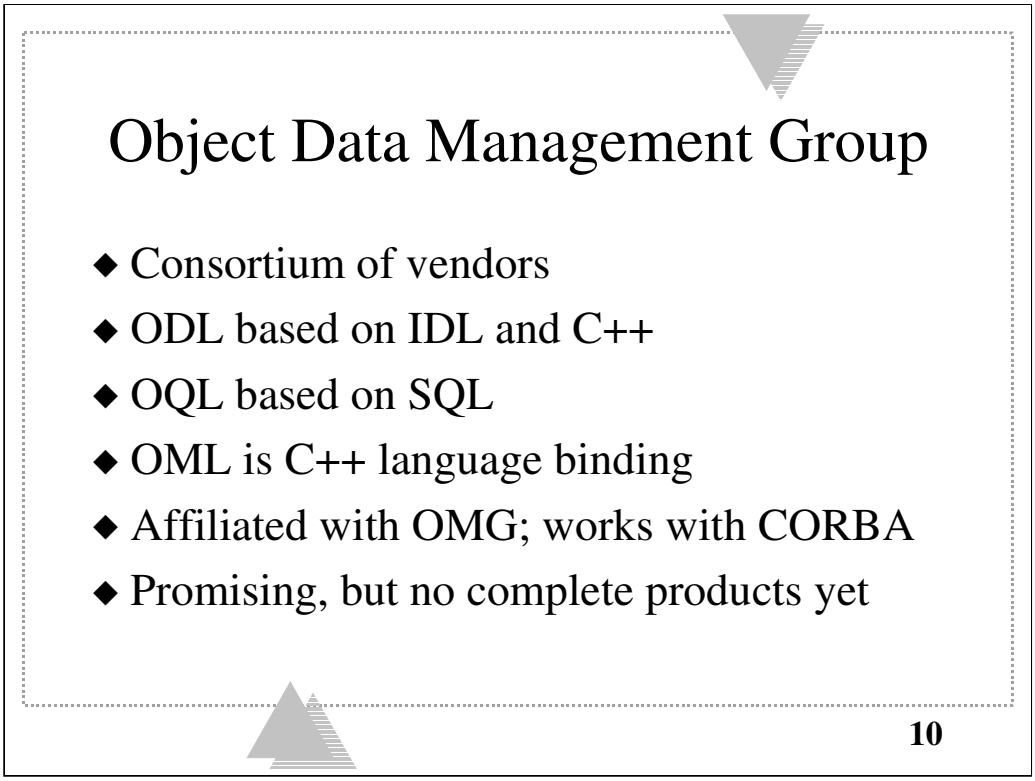
The third manifesto, [darwen95], more formal and technical than the first two, presented a system firmly rooted in the relational model. [atkinson89] got it wrong, they said, in ignoring relational; [cadf90] got it right on that point, but wrong when they supported SQL, which is a flawed perversion of the relational ideals. The system defined in [darwen 95] extends the relational model by allowing the definition of new data types for domains.



The first object databases (left portion of the diagram) restarted from scratch, without any ties to relational systems; they based their type systems on object oriented programming languages and went on from there to add database features. This is the approach described in the first manifesto referred to in the previous slide.

Another strain of object databases started from relational systems and added object features to them, as in the approaches described in the second and third manifestos. [manola94] describes this approach as "object relational DBMS", using the name "object oriented DBMS" for the first style and "object DBMS" as a collective name for the two.

The two approaches tend to a common point, but they reach it from different routes. The OODBMS approach starts from an object oriented framework and adds facilities to express declarative queries. The ORDBMS approach starts from a relational framework and extends the type system with objects.



ODMG, formed in 1991, is a consortium of object database vendors led by a small and focused group of experts. Their aim is to produce an open standard for object database management systems. Among the requirements for voting members is that the parent company must commercially ship an object database product and must commit to implement the ODMG specification.

The ODMG system has a strong CORBA flavour; although not derived from OMG, ODMG is affiliated with OMG and OMG has approved ODMG-93 as a standard interface for a Persistence Service.

The ODL (Object Definition Language) is an extension of CORBA's IDL and as such has a C++ flavour.

The OQL (Object Query Language) has a syntax based on SQL.

The OML (Object Manipulation Language) is a C++ language binding which provides access to database functions and persistence to objects defined with ODL. Among other things it defines standard container classes.

The standard has been formalised in a book [cattell94] but there are no full implementations just yet. A private communication from the ODMG executive director received while compiling these notes (1995 11 22) said "*A listing of ODMG-93 compliant products will be posted at http://www.odmg.org within the next two weeks*".

Summary

- General consensus on impedance mismatch
- No consensus on what an ODBMS must be
- Advantages of ODBMS:
  - rich, flexible type system
  - good interfacing to existing object environments
  - better performance in specific domains
- ODBMS don't have math bg of RDBMS
- ODBMS don't have maturity of RDBMS (yet?)

Field currently in wild evolution

Almost everyone in the field acknowledges the "impedance mismatch" problem and agrees that relational databases are insufficient for certain classes of applications. However, while object technology seems to provide a solution for improvement, there is no consensus on what the ideal ODBMS should look like, not even from a theoretical standpoint (see the three manifestos).

The main advantages of object databases are their rich type system which matches the expressive power and preserves the strong typing of its counterpart in OO programming languages, and the performance advantage they offer in certain applications to which relational systems are not suited.

The main drawbacks are that, unlike relational systems, they don't rest on a mathematical foundation and, partly as a consequence of this, they haven't currently attained the same maturity.

New products are being brought out and refined continuously, both in academia and industry, but there is no universally agreed benchmark (like Codd's rules for RDBMS) against which to compare them.

Bibliography

- The three manifestos (E/P)
- OO chapters of Date's textbook (P)
- Ullman's lecture notes (E/P)
- The Manola report (E/P)
- ODMG-93 (P)
- Cattell's *Object Data Management*

The capital letters in the above are mainly of "local" (Olivetti Research Laboratory) interest:

E = I've got an electronic copy of it;

P = I've got a paper copy of it.

The annotated bibliography, which includes the complete bibliographical references which could not fit on this page as well as a few more texts and some online resources, is available as a separate document at the following URL, where this same set of slides and all the "E" documents are available too:

`http://www.cam-orl.co.uk/~fms/db`

In the annotated bibliography, the above texts are referred to as follows:

•[atkinson89], [cadf90], [darwen95]

•[date95]

•[ullman94]

•[manola94]

•[cattell94]

•[cattell91]

An annotated bibliography on object oriented databases

Frank Stajano
Olivetti Research Limited

This bibliography is the outcome of a literature survey on object oriented databases that resulted in my talk *Object Oriented Databases: an overview*, to which this is an appendix. Annotated slides for that talk, as well as an online version of this bibliography (with live links where appropriate) are available from `http://www.cam-orl.co.uk/~fms/db`.

Books and papers

[atkinson89]

M. ATKINSON, F. BANCILHON, D. DEWITT, K. DITTRICH, D. MAIER, S. ZDONIK,
The Object-Oriented Database System Manifesto.
In *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, pages 223-40, Kyoto, Japan, December 1989.

Because there isn't a clear consensus on what an OODBMS is (a clear definition and a solid theoretical framework are both missing), the authors present this manifesto as a starting point for discussion. The paper presents a set of rules that a DBMS must comply with if it is to be called an OODBMS. The rules are subdivided in three categories: mandatory, optional, open. Only rules in the mandatory category (aka "the golden rules") are presented in detailed form.

The 13 golden rules are mostly of the form "it must support property X", where X is either a typical property of OO programming systems (e.g. encapsulation, classes, inheritance, polymorphism) or a typical property of existing databases (persistence, indexing, concurrency, recovery). The paper says little about the integration of the OO part with the DB part.

[cadf90]

THE COMMITTEE FOR ADVANCED DBMS FUNCTION,
Third Generation Database System Manifesto,
In *Computer Standards and Interfaces* 13 (1991), pages 41-54. North Holland.
Also appears in *SIGMOD Record* 19:3 Sept, 1990.

A counter-manifesto to [atkinson89].

1st generation of databases: hierarchical and network; 2nd generation, which superseded the 1st: relational; 3rd generation (note that we don't call it OO): yet to come, but here are some tenets and propositions for it. The basic idea is: the relational systems were good, especially with their non procedural DML and data independence, and 3rd generation systems should keep these advantages,

while the rush to object oriented systems might break them and should be carefully considered. Things that need adding are richer object structures and rules. The claim is that richer type systems can be added to relational databases without necessarily going all the way to OODB. It is not always entirely clear where the dividing line exactly is between “relational with richer type system” (including inheritance, methods etc) and OODB. One of the points made is that access to the database should be only through a non procedural SQL-type query language instead of a navigational, pointer-oriented interface.

[cattell91]

CATTELL, R.G.G.,
Object data management: object-oriented and extended relational database systems,
Addison-Wesley, 1991

I haven’t been able to get hold of this book yet but I’ve seen it quoted sufficiently often that I consider it worth a look. Apparently the first book-lenght tutorial on OODBMS.

[cattell94]

R.G.G. CATTELL (ED),
The Object Database Standard: ODMG-93, Release 1.1
The Morgan Kaufmann Series in Data Management Systems, 1994

The Object Database Management Group is a recently established consortium of object-oriented database companies whose aim is to produce an inter-vendor standard for OO databases. This book is the second draft (release 1.1) of this standard, called ODMG-93. The ODMG is affiliated with CORBA’s OMG and in 1994 the OMG has adopted a Persistence Service endorsing ODMG-93 for storing object state. For a representative to become a voting member of the committee the sponsoring company must, among other things, commit to the development and support of a compliant database product.

ODMG follows the OMG architecture and has a general CORBA/C++ flavour. The ODL (Object Definition Language) is designed to be compatible with IDL, which it extends. The OQL (Object Query Language) is a non-procedural query language similar to SQL but with support for objects. The result of a query is typically a container (set, bag, array, list...) of objects. The OML (Object Manipulation Language) supports transactions with commit and rollback. In the C++ binding (a Smalltalk binding is also defined but appears to have secondary importance), the ODL’s containers are mapped into appropriate template-based collection classes. The C++ binding consists mostly of a class library.

Interworking with CORBA is part of the design specification. Interworking goes in both directions: the ORB is a client of the ODBMS for the implementation of object persistence, while the ODBMS is a client of the ORB for location, naming, and access to other ODBMSs. The ODBMS is optimised to provide better performance than the ORB for managing “millions of fine-grained objects”. The lower-grain objects are registered with the ODBMS and not with the ORB, thus bypassing the BOA/RPC overheads. Other objects are registered with both the ODBMS and the ORB.

[darwen95]

HUGH DARWEN AND C.J. DATE,
The Third Manifesto,
SIGMOD RECORD 24(1):39-49, March 1995.

The third manifesto aims to supercede the first two, [atkinson89] and [cadf90]. A rather technically oriented document, it formally defines the main properties of a hypotetical database language D. This language has a strong foundation in the relational model and has other characteristics that enrich its type system by providing user-defined types for domains. The belief of the authors is that the relational model can accommodate the required object oriented extensions without being corrected or perverted. The paper firmly states that SQL is “a perversion” of the relational model and makes it

clear that D should be based on more formal and abstract grounds. D accesses the database only through declarative queries.
The paper is more formal and precise than the other two manifestos and it describes an appealing, robust system. Unfortunately it doesn't exist yet!

[date95]

DATE, C. J. (CHRISTOPHER JOHN)
An introduction to database systems, Vol. 1, 6th ed,
Addison-Wesley, 1995

The latest revised edition of a classic database textbook. The last four chapters are dedicated to object oriented database systems. I find Date's explanations very clear and useful. He is a rather strongly opinionated author (although I personally find I tend to agree with him) and a strong supporter of the relational model (see [darwen95]); his main theme in these last chapters seems to be the rational destruction of the myth that object oriented databases are the designated successors to relational databases. His presentation of OODBMS is based on GemStone/OPAL as in [ullman82], so not all of Date's criticisms would be fair against the new generations of products such as those reviewed in [manola94]. Nevertheless, his coverage of the subject is clear, interesting and thought-provoking.

[kim89]

W. KIM AND LOCHOVSKY (EDS),
Object-Oriented Concepts, Databases, and Applications,
Addison-Wesley (Reading MA), 1989

A collection of unrelated papers by different authors. As such it is rather fragmentary. The various bits are narrowly focused and I wouldn't recommend the volume as an introduction to the field. Chapters that were supposed to give a general perspective were rather generic and didn't offer any outstandingly clear insights.

[manola94]

FRANK MANOLA,
An Evaluation of Object-Oriented DBMS Developments, 1994 Edition,
GTE Laboratories technical report TR-0263-08-94-165, 1994

A very thorough and up-to-date review of the major commercial products in the object database arena including detailed coverage (approximately 10 pages each) of over a dozen products. This is a good text which combines a clear introductory perspective, a panorama of the currently available commercial offerings and a reasonably unbiased presentation of the many different approaches to the problem. The report opens with an enlightening introductory chapter that explains the motivations for the development of object database technology and highlights the main characteristics of ODBMS, taking into account the prescriptions of the first two manifestos, [atkinson89] and [cadf90]. It also highlights the different perspectives offered by the object-oriented DMBS approach versus the object-relational DBMS approach. After the product evaluation sections comes a chapter on standards covering ODMG-93 (see [cattell94]), SQL3 and OMG. The report then investigates key ODBMS features and issues in providing them. Finally, application considerations are investigated.

[ullman82]

JEFFREY D. ULLMAN,
Principles of Database and Knowledge-based Systems, Vol.1, 2nd edition,
Computer Science Press, 1982

(1st volume of 2-volume textbook.)
(See the annotation to [ullman89] first.) This first volume had something, but it was 1982 material and as such not comparable with the recent advances in the field. E.g. the simple fact of having object identity was considered as a feature suggesting object orientation. Volume 1 has a 20-page

description of an early object-oriented DMBS, OPAL. This turns out to be a system with OO-style class definitions where queries cannot be expressed declaratively and have to be implemented by writing methods for the objects they apply to.

[ullman89]

JEFFREY D. ULLMAN,
Principles of Database and Knowledge-based Systems, Vol.2,
Computer Science Press, 1989

(2^{nd} volume of 2-volume textbook.)

I looked at this because it was the textbook for the [ullman94] lectures, but there wasn't any material on OODBMS in this second volume. See [ullman82] instead.

[ullman94]

JEFFREY D. ULLMAN,
Informal lecture notes for the author's university course CS345A, Principles of Database Systems, 1994.

Lectures 14 - 16 contain a brief and easy to read introduction to OODBMS from an ODMG perspective, including some CORBA coverage. OODB = OO+DB; ODL is like CORBA's IDL, plus database features like domains, keys, relationships. Types are either atomic or built with constructors like set, bag, list, array, structure. OQL is a Select-From-Where query language that returns a set of values.

Useful web resources

`http://www.cs.cmu.edu/afs/cs.cmu.edu/user/clamen/OODBMS/README.html`

Contains copies of the three manifestos and a few pointers to books and online resources.

`http://www.odmg.org/`

Home page of the ODMG. Contains background on ODMG, a list of members, amendments to the standard and, sometime soon, a list of compliant commercial products.

`http://www-db.stanford.edu/~ullman/index.html`

Home page of Jeffrey D. Ullman, professor at Stanford University, USA. Contains his lecture notes and the list of his publications (careful with that one! 9 pages, 177 items... ☺)

`http://web.cs.ualberta.ca/~ozsu/`

Home page of M. Tamer Özsu, professor at the Laboratory for Database Systems Research at the University of Alberta, Canada. Contains a list of his publications and postscript for many of them, including entire book chapters.

Useful web meta-resources

`http://web.cs.city.ac.uk/homes/akmal/info.html`

A well organised list of bibliographical references and online resources on object databases.

`http://www.lpac.ac.uk/SEL-HPC/Articles/DBArchive.html`

An archive of technical articles on databases with various searchable indexes.

http://www.informatik.uni-trier.de/~ley/db/index.html

A bibliography server on database systems and logic programming. Covers conferences, journals, books etc.

http://src.doc.ic.ac.uk/computing/bibliographies/Karlsruhe/Database/index.html

An extensive set of bibliographies on database systems, part of a larger collection of computer science bibliographies.

http://ibd.ar.com/ger/comp.databases.object.html

An uncommented list of links to http and ftp resources on object databases, compiled from postings to the Usenet newsgroup `comp.database.object`.