

# Re: OT: SQL

---

*Source:* <http://sci.tech-archive.net/Archive/sci.logic/2008-05/msg00150.html>

---

- *From:* Marshall <marshall.spight@xxxxxxxxxx>
  - *Date:* Sat, 3 May 2008 21:37:51 -0700 (PDT)
- 

Charlie-Boo wrote:

On May 3, 7:01pm, Marshall <marshall.spi...@xxxxxxxxxx> wrote:

On May 3, 2:24 pm, Charlie-Boo <shymath...@xxxxxxxxxx> wrote:

On May 3, 12:26 pm, Marshall  
<marshall.spi...@xxxxxxxxxx> wrote:

Codd noticed that databases contain relations. The rest of what he said was a totally failed attempt to formalize and automate database query processing. SQL is an example of that failure, by virtue of it being only a programming language rather than an automatic query processor.

"Totally failed" is too strong. Partly failed, partly succeeded. Still a definite step forward if you compare it to what came before.

Is there a procedure for asking "List all employees who earn more than their managers." (classic problem)

Um, yeah, that's pretty easy actually.

-- first, the data definition:

```
create table Employees
```

```
(
```

```
  ý EmployeeId int primary key,
```

```
  ý Salary Money
```

```
);
```

```
create table Management
```

```
(
```

```
  ý ManagerId int foreign key references Employees(EmployeeId),
```

```
  ý EmployeeId int foreign key references Employees(EmployeeId),
```

```
  ý primary key(ManagerId, EmployeeId)
```

```
);
```

-- then the actual query

-- list the employee id of all employees who have a

-- larger salary than their manager.

```
SELECT e.EmployeeId
```

```
FROM Employees e, Employees m, Management mgt
```

```
WHERE
```

```
  ý mgt.ManagerId = m.EmployeeId AND
```

```
  ý mgt.EmployeeId = e.EmployeeId AND
```

```
  ý e.Salary > m.Salary;
```

Notice that this solution specifies nothing whatsoever about how the result is to be obtained. No execution plan is mandated. This is a straightforward, unadorned declarative specification of what information is desired.

No. It says go through the Manager.Employee hierarchy and get the Manager and Employee Salaries.

The relations in the FROM clause and the predicates in the WHERE clause are unordered. There is no "go though" anywhere. Again, the query is a logical construct; it does not specify any physical data path or any physical schema; it does not depend on the presence or absence of any index. It doesn't specify any access path or depend on any physical data structure whatsoever. It depends only on the absolute minimum set of things it necessarily must depend on, which is the logical schema.

Re: OT: SQL

An alternate way would be to go through the Employee file, get the Employee Salary and Manager, using the Manager ID get the Manager Salary.

"File" is a filesystem concept; it is not a data management construct. You're not by chance an MV advocate, are you? They use that word "file" a lot.

Some systems are smart enough to check for both possibilities, but are hardwired to that specific 2nd possibility. "Go through an index" where index is Field => Key of Record is hardwired. But in general you can have any logical expression and any hierarchy of any conditions.

You could have a file of employees who make more than their manager and just go through that. Or a file of employees who don't make more than their managers and you go through all employees and don't list those in that file.

All of these possible files are expressible as Predicate Calculus wffs and will be seen as possibilities by the algorithm detailed in my ARXIV paper. How would your SQL query see them? It wouldn't.

The query *\*shouldn't\** see them. The things you describe are all physical performance tweaks. You don't want to pollute your logical schema with physical issues, such as the indexes or materialized views you describe. The schema should be independent of all of those consideration, and be a purely logical construct. This enables swapping physical constructs independent of queries; it increases modularity.

Again, higher end DBMS products can do what you describe: use materialized views in query planning.

You are hardwired for 1 or 2 special cases only.

My query is not hardwired at all.

There could be other indexes e.g. Salary.Employee and after we get the Employee we go through Salary and if Salary < Employee's get the Employees with that Salary and if any is his Manager then he passes.

Re: OT: SQL

There are lots of possibilities of how to go through possible databases and get that data, and there are lots of file structures that might exist besides the "relation" Employee and maybe an index of Manager.Employee.

The logical schema can be backed by whatever physical data structures. Usually row stores are used, but there are companies and academics advocating column stores these days. Hierarchical stores have mostly been discredited because of the problems with query bias.

The query can be expressed as standard Predicate Calculus

The relational algebra is expressively equivalent to predicate calculus. This result is due to Codd.

but signify  
input as e.g. I, J, K . . . and output as x, y, z, . . . and the  
single relation (regardless of how many database files you have and  
what their content or keys are)

The Universal Relation idea? Ullman advocated for that for many years, but it ultimately failed. It never seemed even remotely like a good idea to me.

is EMP(Emp#,Manager#,Salary):

$$(\text{existsMGR})(\text{existsEMPSAL})\text{EMP}(X,\text{MGR},\text{EMPSAL}) \wedge (\text{existsMGRSAL})$$
$$(\text{existsMGRMGR})\text{EMP}(\text{MGR},\text{MGRMGR},\text{MGRSAL}) \wedge \text{LT}(\text{MGRSAL},\text{EMPSAL})$$

(A higher level interface can walk the user through this and produce the Predicate Calculus wff from that.)

In <http://arxiv.org/html/cs/0003071> I show how this Predicate Calculus query can be reduced to Axioms that may be supplied as database files. Each time the axioms are all database files, you have a different solution.

Codd talked about using Logic but all that time (up to the present) he and the rest did not realize that standard Predicate Calculus suffices to express queries.

Re: OT: SQL

Re: OT: SQL

Again, Codd proved relational algebra expressively equivalent to the predicate calculus in, like, 1970 or so. There are other equivalent theories as well, like the tuple calculus.

All of the SQL syntax and semantics is not needed.

The syntax sucks ass, no question. The semantics are wrong in a whole bunch of different places, too, especially in the use of NULL and three-valued-logic. (3VL sucks ass.)

Codd never talked about listing prime numbers and Manna/Waldinger never talked about listing Employees. But they are all cases of reducing a Predicate Calculus wff down to primitive programs that consist of programming language constructs or database accesses contained within that programming language.

If you said that in comp.database.theory, you'd get "of course" as a reply.

They say that "a file is a relation" but that is 2 steps away from the truth.

I don't know who these "they" are. I don't know anyone in the database field outside of the fringe MV crowd that even uses the word "file".

Researchers created all sorts of query languages, but you can stay within standard Mathematics syntax and use an axiomatic framework to unite Program Synthesis, Database Query Processing and even Theory of Computation proofs using the same Rules of Inference but a handful of axioms particular to the Theory of Computation.

Sure. I've been working on that for years. So have lots of other people.

Re: OT: SQL

Re: OT: SQL

Combining these different problems, seeing the connections, and coming up with a general solution to all of them is exactly what we want to do.

Agreed.

Marshall

.