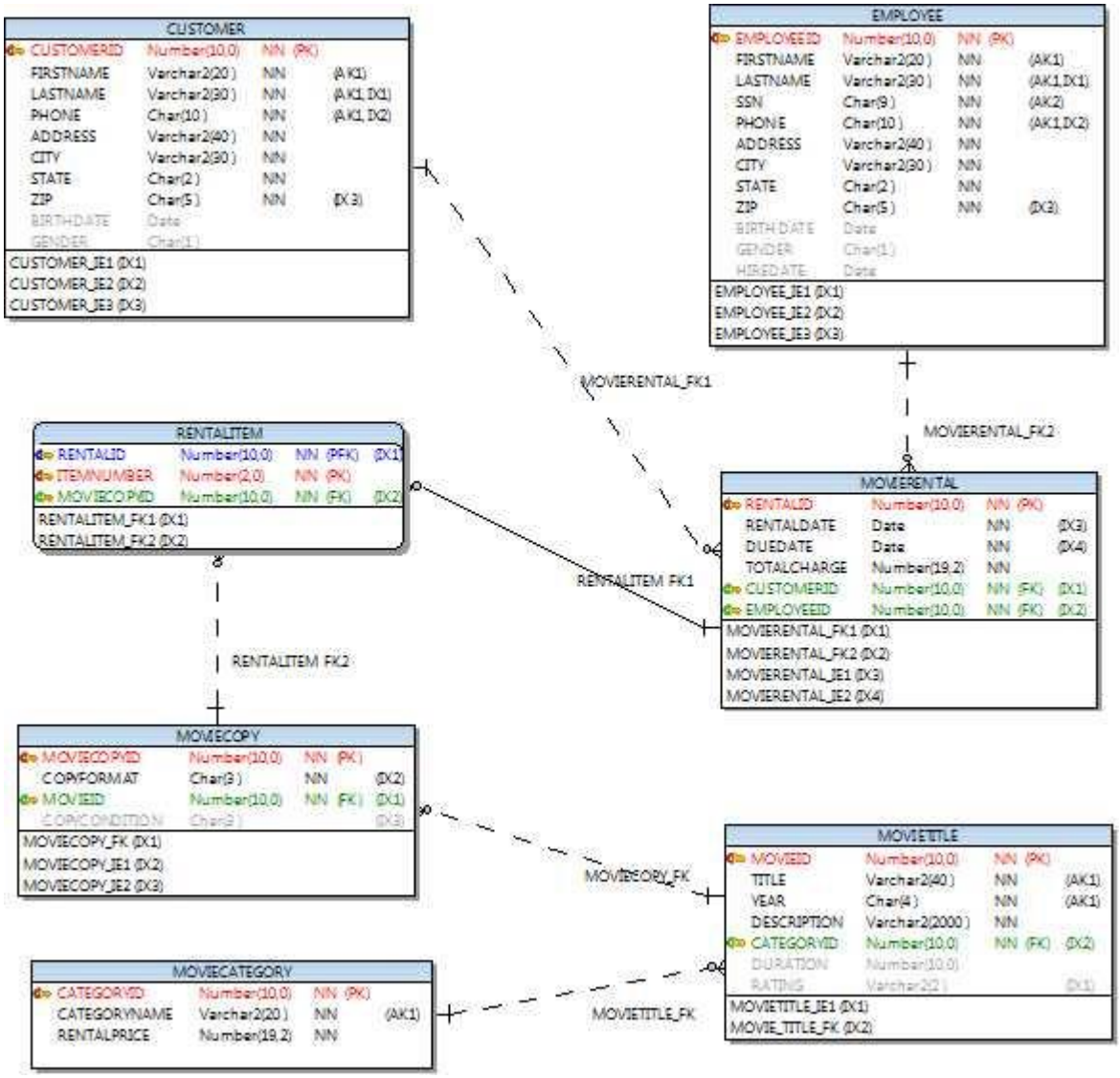
SQL Aint So Simple

SQL is not an overly complex language in theory. The data manipulation language or DML consist of four primary commands: INSERT, UPDATE, DELETE and SELECT. Newer versions of Oracle also offer the MERGE command, which is an *"upsert"* – meaning it tries an update followed by an insert as a single command. Since the SELECT command is the one we use most since queries are the majority of database operations, let's examine statement complexity in real life to see if the SQL language really is simple.

Let's use the following data model for a movie rental system as the basis for our SQL construction examples that follow.



What we want is a basic report that yields the rental date, total revenue per movie format made by employees with last name of "Smith" for customers whose address starts with the letter A, F, T or Z. The results would match the following.

Row#	LASTNAME	FIRSTNAME	RENTALDATE	COPYFORMAT	REVENUE
1	Smith	John	1999-10-07	DVD	1045.48
2	Smith	John	1999-10-07	VHS	1031.36
3	Smith	Mary	1999-10-07	DVD	1024.7
4	Smith	Mary	1999-10-07	VHS	1048.3

So here's the way a typical person might write the query:

```
select  a.LASTNAME, a.FIRSTNAME, b.RENTALDATE,
        d.COPYFORMAT, sum(b.TOTALCHARGE) REVENUE
from employee a, movierental b, rentalitem c, moviecopy d
where a.EMPLOYEEID = b.EMPLOYEEID
  and b.RENTALID = c.RENTALID
  and c.MOVIECOPYID = d.MOVIECOPYID
  and a.LASTNAME = 'Smith'
  and b.CUSTOMERID IN (
          select customerid
          from customer e
          where e.address like 'A%' or
                e.address like 'F%' or
                e.address like 'T%' or
                e.address like 'Z%'
          )
   group by a.LASTNAME, a.FIRSTNAME, b.RENTALDATE, d.COPYFORMAT
```

What are all of the *"main stream"* or normally expectably ways that one can rewrite this query and yet retain the exact same results? A tool like Quest's SQL Optimizer might return hundreds or even thousands of rewrites in an effort to find the most efficient code. But what are the primary rewrites a human might be expected to write?

First, we know that using non-ANSI syntax for JOIN's is to generally be avoided. So we can change the SELECT as follows.

```
select  a.LASTNAME, a.FIRSTNAME, b.RENTALDATE,
        d.COPYFORMAT, sum(b.TOTALCHARGE) REVENUE
```

```
from employee a JOIN movierental b using(EMPLOYEEID)
                    JOIN rentalitem c using (RENTALID)
                    JOIN moviecopy d using(MOVIECOPYID)
where a.LASTNAME = 'Smith'
  and b.CUSTOMERID IN (
          select customerid
          from customer e
          where e.address like 'A%' or
                  e.address like 'F%' or
                  e.address like 'T%' or
                  e.address like 'Z%'
          )
    group by a.LASTNAME, a.FIRSTNAME, b.RENTALDATE, d.COPYFORMAT
```

That makes the WHERE clause much easier to read since the JOIN code is now in the FROM clause with the tables. This exposes that we have an IN clause – which sometimes might not be the best way to code it. So we'll next switch to using an EXISTS clause. Yes – some newer versions of the Oracle optimizer might automatically make this switch internally for us. But when in doubt, it's always better to code it the proper or most efficient way for all cases. So now we can change the SELECT as follows.

```
select  a.LASTNAME, a.FIRSTNAME, b.RENTALDATE,
                        d.COPYFORMAT, sum(b.TOTALCHARGE) REVENUE
from employee a JOIN movierental b using(EMPLOYEEID)
                    JOIN rentalitem c using (RENTALID)
                    JOIN moviecopy d using(MOVIECOPYID)
where a.LASTNAME = 'Smith'
  and EXISTS (
          select 1
          from customer e
          where e.CUSTOMERID = b.CUSTOMERID
            and (e.address like 'A%' or
                  e.address like 'F%' or
                  e.address like 'T%' or
                  e.address like 'Z%'
                )
          )
    group by a.LASTNAME, a.FIRSTNAME, b.RENTALDATE, d.COPYFORMAT
```

Looking better – but we could also code that correlated sub-select as an *"inline view"*. So yet once again we can change the SELECT as follows.

```
select  a.LASTNAME, a.FIRSTNAME, b.RENTALDATE,
        d.COPYFORMAT, sum(b.TOTALCHARGE) REVENUE
from employee a JOIN movierental b using(EMPLOYEEID)
                    JOIN rentalitem c using (RENTALID)
                    JOIN moviecopy d using(MOVIECOPYID),
      ( select customerid
          from customer e
          where e.address like 'A%' or
                  e.address like 'F%' or
```

```
                    e.address like 'T%' or
                    e.address like 'Z%'
        ) e
where a.LASTNAME = 'Smith'
  and b.CUSTOMERID = e.CUSTOMERID

   group by a.LASTNAME, a.FIRSTNAME, b.RENTALDATE, d.COPYFORMAT
```

But we're still not quite done yet. SQL also offers the very useful WITH clause that can often be very efficient in some cases. So finally we can change the SELECT as follows.

```
with f as ( select customerid
              from customer e
               where e.address like 'A%' or
                     e.address like 'F%' or
                     e.address like 'T%' or
                     e.address like 'Z%'
          )
select  a.LASTNAME, a.FIRSTNAME, b.RENTALDATE,
        d.COPYFORMAT, sum(b.TOTALCHARGE) REVENUE
from employee a JOIN movierental b using(EMPLOYEEID)
                    JOIN rentalitem c using (RENTALID)
                    JOIN moviecopy d using(MOVIECOPYID),
                    f
where a.LASTNAME = 'Smith'
  and b.CUSTOMERID = f.CUSTOMERID

   group by a.LASTNAME, a.FIRSTNAME, b.RENTALDATE, d.COPYFORMAT
```

Wow – that's five major ways to rewrite this query. Since ANSI joins are the recommended way to go, the question remains which of these four coding styles is the more correct? The academia reply is using the WITH clause, as it leverages all the latest and greatest SQL language constructs available. But the correlated sub-query and inline view solutions are very typical of what someone might write – and there is seemingly nothing wrong with those approaches either. So I would not recommend a forced rewrite as long as the code is like one of the last three examples.

So which one yields the best performance? We'll leave that discussion to next time ☺