Oral History of Donald Chamberlin

Interviewed by:
Paul McJones

Recorded: July 21, 2009
Mountain View, California

CHM Reference number: X5374.2009

Paul McJones: This is Paul McJones and I'm interviewing Don Chamberlin. It's the 21st of July 2009 and we are at the Computer History Museum in Mountain View, California. Hello, Don.

Don Chamberlin: Hi Paul, it's a pleasure to be here.

McJones: It's a pleasure to be interviewing you. I thought it would be good to start with your background, when and where you grew up, a little bit about your family and education.

Chamberlin: Sure. I was born here in the Bay Area in Campbell, California. My father was actually fighting in the South Pacific on my birthday and he didn't get home to see me until I was almost a year old. I grew up in Campbell, which was then a little orchard town distinct from San Jose and the other towns that have since enveloped it. I went to Campbell High School, which no longer exists. It's now been turned into a community center for the town of Campbell. My father was a high school principal and my mother was a homemaker. After graduating from high school I went to college at Harvey Mudd College in Southern California. Harvey Mudd, as you probably know, is a very small college that's focused on science and technology. I think the thing that got me interested in science and technology as a career was all of the emphasis on technology that was going on in the late 1950s as a fall-out from the Sputnik launch, the Russian satellite. There was a sense in the country that technology was important and that we were falling behind and losing our edge. I had an idea that technology was a way to do something important and to make a contribution that would be respected, recognized, appreciated, and rewarded, and, by and large, I think that's turned out to be true.

McJones: Did you have specific teachers or other mentors around that time, high school or perhaps even earlier, that encouraged you in this direction?

Chamberlin: Not specifically. I had good teachers in high school.

McJones: But nobody in particular stood out that was...

Chamberlin: I don't think there's anybody I could name in particular along those lines. I was interested in Harvey Mudd because it was focused on excellence of undergraduate education. They're strictly an undergraduate school. They have small classes. Their professors are all dedicated primarily to the education process, to teaching and involving undergraduates in their research, and I thought that would be a really good environment to learn in, and I think it was. I enjoyed my time at Harvey Mudd very much.

McJones: As I understand it, you received a general engineering degree at that point?

Chamberlin: That's right. Harvey Mudd was such a small school that they didn't specialize in specific branches of engineering, so my degree from Harvey Mudd was in engineering, which was one of their four majors at the time. You could major in math, physics, chemistry or engineering. That's all there was.

McJones: How about the liberal arts subjects? Did they have good coverage?

Chamberlin: They did, actually. Harvey Mudd is a member of the Claremont Consortium of five undergraduate colleges. Harvey Mudd specializes in science and engineering, but there are other colleges that specialize in other aspects of learning, and you can cross register at any of the colleges, and you're encouraged to do that. So Harvey Mudd, I think, is actually kind of unique in the emphasis that it places on liberal arts as a part of an engineering education.

McJones: That sounds good. It seems to have worked well in your case.

Chamberlin: Yes.

McJones: So then you decided to go on. A bachelor's degree wasn't enough. You wanted to go on to grad school.

Chamberlin: Yes, I got lucky, and I did well at Harvey Mudd, and I got a full-ride fellowship from the National Science Foundation to go to Stanford University. So I went to Stanford. When I went to Stanford they were just starting their computer science program. It was the first year that computer science was a separate department. And I thought about computer science as a major in graduate school, but it seemed risky to me. This was a brand new discipline, and who knows whether it was going to have any staying power or not, whereas electric engineering was a well established department, with a well established pipeline, and they were turning out many PhDs every year. They were well recognized in industry, and I thought it might be safer to major in EE, so that's what I did. My advisor was Ed McCluskey, who had a joint appointment in EE and computer science—he was really kind of a computer science guy. He ran something called the Digital Systems Laboratory, and I was one of his first PhD students at Stanford after he came to Stanford from Princeton. He was mainly a hardware guy. He was interested in fault tolerant digital circuits and things like that. I was more interested in software and also in computer architecture in those days, so my thesis[1] at Stanford was a design for a parallel machine, what we'd call these days a data flow machine, that would have many processing units that would basically execute as soon as their data was ready. Data would flow from one task to another, and the processors would be dispatched to the tasks on the basis of the data flow.

McJones: I assume you weren't actually able to fabricate it.

Chamberlin: No.

McJones: But did you do simulations?

[1] D.D. Chamberlin. *Parallel Implementation of a Single-Assignment Language*. Doctoral Thesis. UMI Order Number: AAI7123494, Stanford University, 1971. See also: D.D. Chamberlin. The "single-assignment" approach to parallel processing. In *Proceedings of the 1971 Fall Joint Computer Conference* (Las Vegas, Nevada, November 16 - 18, 1971), ACM, New York, NY, 263-269. http://doi.acm.org/10.1145/1479064.1479114.

Chamberlin: We did simulations.

McJones: Yes, it's a more affordable approach.

Chamberlin: Yes.

McJones: Were there fellow graduate students that you remember from that time that you encountered later?

Chamberlin: Sure. Sam Fuller was a colleague of mine at Stanford. We were close friends in graduate school. He later became VP of research at DEC.

McJones: And now he's CTO and VP of R&D at Analog Devices.

Chamberlin: Dan Siewiorek, who's now a very well known professor at Carnegie Mellon University, was also a colleague of Sam's and mine in the Digital Systems Lab at Stanford at that time.

I spent summers working in a variety of places. I worked at Lockheed Research Lab at Hanover Street in Palo Alto. I worked at Hewlett Packard, where they were designing one of the early HP mini-computers. I worked one summer in the Digital Systems Lab at Stanford trying to interface a television camera to an IBM 1620, which is an early machine.

McJones: Interesting. And one of those summers didn't you also go to IBM?

Chamberlin: Yes. In the summer of 1970, after I'd been at Stanford for about four years, I was offered an opportunity to spend the summer at Watson Research Center at Yorktown Heights in New York State, in Westchester County, just north of New York City. This was a big adventure for me. It was the first time I'd been outside California for any extended period of time. My wife and I drove our 1962 Chevy across the country, and took two weeks to do that, and went to Yellowstone Park and Niagara Falls and had a wonderful trip. I really enjoyed my summer at IBM. I was in a group that was working on scheduling algorithms for a time sharing system. If you had many humans sharing a computing resource, and humans had an expectation of what kind of response time would make them happy, and you had to schedule the tasks and allocate the memory in such a way that all of the jobs got done on time—that was what we were working on. I was working on a simulator for this scheduling algorithm. It was written in PL/1. And I had a really good time that summer.

McJones: It introduced you to the company and the people and the style of work.

Chamberlin: Yes, I enjoyed working in the building at Yorktown Heights. It was a beautiful building made out of glass and stone on top of a hill. It was designed by Eero Saarinen, the famous architect, the same guy that designed the St. Louis Arch. And it was a beautiful place to work.

McJones: So then you came back, finished up your work, and you were on the job market, I guess.

Chamberlin: Yes. Actually by the time I was ready to leave Yorktown Heights at the end of my summer assignment, the person I had been working with, who was a manager named Leonard Liu, interviewed me and realized that I was nearing completion of my thesis and said he was going to try and generate an offer for me. And shortly after I got back to Palo Alto I did receive an offer from Yorktown to return there as a permanent employee. I had a few months to spend finishing up my thesis and did that. I graduated in winter quarter of 1971, and left California for New York in February of 1971. Moving to New York in February is not something that I'd recommend.

McJones: But you did consider a couple of other places?

Chamberlin: I interviewed at Burroughs. They had a lab in Santa Barbara, California. One of my advisors at Stanford, Bill McKeeman, was well connected to Burroughs and he thought I ought to go down there and talk to them, and they seemed to be doing interesting work. The Xerox Palo Alto Research Center was just getting started. They didn't have a building or hardly any staff, but they were interviewing some early candidates and I talked to them. But I had had such a good experience at Yorktown and had a kind of personal connection with the people, and knew what they were doing, and I thought I'd like to go back there, so that's what I chose to do.

McJones: That obviously worked out quite well. So then did you start off in the same project as you had been working on?

Chamberlin: Yes, I did. I was working for the same manager, Leonard Liu, on a timesharing system. In those days the IBM 370 had just been released and timesharing, believe it or not, was kind of an advanced concept. The idea that a computer was a big expensive resource, and lots of people could share it and you had to figure out a way for them to share the physical resources of the machine, and how you would schedule and allocate these resources—that was the problem we were working on. Nowadays it's the opposite, the human is expensive and machines are cheap, but the economics were quite different 40 years ago.

McJones: Right. So that was the System A Project?

Chamberlin: Yes, it was called System A and I was, again, involved in the virtual memory and the processor scheduling aspects of that project[2]. We worked on that for a year roughly after I rejoined the group. But in those days IBM had a lot of redundancy. It was a pretty big company. They often had competing projects in multiple locations. There would be rivalry between these projects, and the System

[2] These papers resulted from Chamberlin's work on the project:

- D.D. Chamberlin, S.H. Fuller, and L.Y. Liu. A page allocation strategy for multiprogramming systems. In *Proceedings of the Fourth ACM Symposium on Operating System Principles* SOSP '73. ACM, New York, NY, 66-72. http://doi.acm.org/10.1145/800009.808051
- Donald D. Chamberlin, Hans P. Schlaeppi, Irving Wladawski. Experimental Study of Deadline Scheduling for Interactive Systems. *IBM Journal of Research and Development* 17(3), 263-269, 1973.

A Project, which was in the research division, was to some extent duplicating the work of an operating system project in one of the development divisions in Poughkeepsie, New York, and at some point management made a decision to consolidate this work So the operating system work went to Poughkeepsie[3]. I liked it in Yorktown Heights—I didn't want to move. Research is a very nice place in IBM so I decided to stay in the Research Division. I had a certain personal loyalty to this manager, Leonard, who had hired me. Leonard was looking around for work for the people that stayed behind, and he felt that database systems were an important area, an area of increasing importance--the management of persistent data that didn't just go away when your program stopped executing but stuck around.

I think over the last 40 years this idea of managing persistent data by a database system has probably been the area of computer science that's had the most direct impact on people's everyday lives. Nowadays we take it for granted that we can stick our ATM card into a machine anywhere in the world and out will come some money in the local currency, or we can just charge things on our credit card, or we can use the web to make airline reservations or to buy things in online auctions. Every time you do any of those things you're interacting with persistent data that's stored in some kind of a database. And database management was just in its infancy in the early 1970s, late 1960s, and this was an area that Leonard could see was going to be of increasing importance and he sort of wanted to put his oar in and see if he could make a contribution to that area. And I think he had good judgment and good vision, and I'm glad I followed in his footsteps.

McJones: Yes, again, it seems to have worked out pretty well. I guess disk drives were about 15 years old by that point. They were getting fairly reliable and relatively affordable, and companies were starting to see that the data itself could span multiple applications. Are these some of things that were leading up to this?

Chamberlin: Yes, the invention of the disk drive, the IBM RAMAC in 1957 that was released out of the IBM San Jose Laboratory, revolutionized the management of this persistent data. Before disks came along data was stored on tape, which was a sequential medium. It would take maybe three minutes to read a tape from beginning to end, and that made it necessary to dedicate data to one particular application and to process it in a sequential way. So you could have a tape that had your inventory on it, and if you had some orders to process against the inventory they'd be on a stack of cards, and you'd run through the stack of cards and through the tape sequentially and make a new tape. The random access property of disks meant that you didn't have to read them from the beginning to the end but you could skip around and access any piece of information as easily as any other piece, basically, and access it quickly. This made it possible to manage persistent data in brand new ways. Data began to turn into a kind of a corporate resource that could be shared among many different applications, and could be managed by a layer of software that would protect the data, and authorize access to it, and provide services like backup and recovery so it wouldn't get lost, and provide transaction semantics so that multiple applications that wanted to access the same piece of data at the same time would not interfere with each other. All of these were new concepts that were evolving in the early 1970s and that were basically made possible by this breakthrough development of the disk drive. The IBM RAMAC that was introduced in 1957 stored five megabytes of data and cost $50,000 dollars, so that was $10,000 dollars a

[3] The Poughkeepsie project continued to evolve and later merged with other projects to form part of FS (Future System). See: Emerson W. Pugh, *Building IBM: Shaping an Industry and its Technology*. MIT Press, 1995.

megabyte for data stored on rotating magnetic storage. Nowadays, of course, magnetic storage costs a few hundredths of a cent per megabyte.

McJones: Right. It's a really incredible rate of development. So that's a very nice background on why you folks felt that this was a good area to jump into. Your backgrounds were not in data management, but very few people had a background in data management so it felt quite reasonable to jump a little bit sideways into this new area.

Chamberlin: That's right. It's not like there wasn't any work that had been done in database management at that point. Of course there had. IBM had a very successful database management product called IMS, Information Management System. IMS had been developed jointly by IBM and one of their customers in the aerospace industry, and it had been generalized and was now in fairly widespread use. There were also a number of other companies that had database management products. There was one very well known product called IDS, or Integrated Data Store, that had been developed originally at GE and later at Honeywell. Charles Bachman was the architect of the Integrated Data Store, and he was very well known in the industry. In fact he was awarded the ACM Turing Award in 1973. Bachman gave a very well known talk on the occasion of his Turing Award called "The Programmer as Navigator"[4]. He observed that we are used to thinking of the computer as a stationary object with data flowing through it, interacting with the program as it flows, as a tape is read sequentially from the beginning to the end. Bachman proposed that we think of data instead as a space, cyber space if you like, where data records are objects that are connected together by pointers and links. The job of retrieving data from a database then becomes a job of navigating through these pathways to find the data records that contain the information that you want.

IDS was one of the systems that embodied the notion of data as a corporate resource that's available, managed in common and shared across many applications. IDS was a very successful system and introduced a lot of ground breaking ideas.

McJones: There was somebody else in IBM who thought this still wasn't the ultimate direction.

Chamberlin: Yes, there sure was. There was Ted Codd, who was sitting out at the San Jose Research Lab doing some work on his own. He was a kind of an iconoclast. He wasn't a team player very much. He was educated as a mathematician at Oxford University. He served in the Royal Air Force. He joined IBM and worked in various places at IBM. By training he was a mathematician, and he thought that mathematical ideas could be brought to bear on the management of data to make it more rigorous and to raise the level of abstraction at which people interacted with databases. He thought this idea of navigating through data space that had been Charles Bachman's trademark and the basis of his Turing Award was a bad idea. He said that when you think of yourself as a navigator you're telling the computer what to do. Go here, then go there, then follow this link. Ted said that you shouldn't tell the computer what to do. You should tell the computer what you want and let the computer figure out what to do, and the best way to tell the computer what you want is to use an abstraction based on the mathematical concept of a relation.

[4] C.W. Bachman, C. W. The programmer as navigator. *Commun. ACM* 16, 11 (Nov. 1973), 653-658. http://doi.acm.org/10.1145/355611.362534

A relation is a well known mathematical construct that represents relationships between objects in domains. For example, if you have a domain of products and a domain of customers, information about which customer bought which product could be represented as a mathematical relation between these two domains. Well, this was a fairly well understood concept in mathematics and Ted thought that we should use this concept to allow users to interact with stored data in a way that was independent of the physical representation of that data. He called this notion data independence. And he said people shouldn't think about pointers, or indexes, or navigation, or any procedural things like that. They should think about the abstract relationship between the objects of interest like products and customers. So Ted published a paper on this subject in *Communications of the ACM* in June of 1970[5].

June of 1970, incidentally, was the month when my wife and I were driving across the country in our 1962 Chevy to take a summer job at Yorktown Heights, so I didn't get my copy of the *Communications of the ACM* on time because I was out of town. But that was the month that I joined IBM and that was the month that Codd published his famous paper.

Codd introduced the notion of relations as an abstraction for stored data, and he explained about data independence and why that was important, and he introduced a couple of languages that were described very briefly in this paper for manipulating stored data in the form of relations. You can think of relations as tables. They look like tables, with rows and columns. One of Codd's languages was called the Relational Calculus. It was based on the first-order predicate calculus, which is a logical notation that's used in mathematics, and it allows you to ask questions by binding variables. You would say something like “find X such that for all Y there exists a Z such that X is greater than Y and Y is less than Z” and things like that. You could write all this out in mathematical notation. It had a lot of quantifiers in it. Quantifiers are things like "for all" and "there exists", and the mathematical notation for those looks like, well, "for all" is an upside down "A" and "there exists" is a backwards "E". All of this is well recognized by mathematicians, but end users of computers found it a little bit esoteric. They weren't used to thinking in those terms. They were used to saying, "Well, go here and then go there." So the Relational Calculus described in Ted's paper certainly raised the level of abstraction that you could use to think about data. It made your queries independent of the physical storage of data, but it seemed a little bit esoteric and mathematical and unfamiliar, I think, to many computer users. The other language that Ted introduced was called the Relational Algebra. It was more of a procedural language. It operated on relations as objects, and it had a bunch of operators that would combine these relations and project things out of them, or select rows out of the table, or select columns out of the table, or join two tables and things like that. Once again, it was represented in a mathematical notation, and once again, some of the operators of that mathematical notation were a little bit esoteric and unfamiliar to many people.

So Codd's paper got some traction in the academic community where these mathematical concepts were very familiar. It didn't, at first, get much traction in the commercial database community. They had their navigational systems. They had IMS from IBM and they had IDS from Honeywell, and these systems were working just fine and doing the job. In fact there was a process underway to standardize something called CODASYL. CODASYL was a database interface that was based loosely on Bachman's IDS system, and that seemed to be where the action was in database management. In fact, my first job at

[5] E.F. Codd. A relational model of data for large shared data banks. *Commun. ACM* 13, 6 (Jun. 1970), 377-387. http://doi.acm.org/10.1145/362384.362685. See also Codd’s previously-unpublished 1969 research report: E.F. Codd. Derivability, redundancy and consistency of relations stored in large data banks. *SIGMOD Rec*. 38, 1 (Jun. 2009), 17-36. http://doi.acm.org/10.1145/1558334.1558336

Yorktown, when my boss wanted to get into the database area, was to study this CODASYL proposal and to suggest ways to improve it and extend it, and that seemed to be where the action was, the center of gravity in the database field at that time. We did read the papers that were coming out of Ted Codd's group in San Jose.

At some point, I believe it was in 1972, Ted organized a symposium in the Research Division at IBM. He chose to do it at Yorktown Heights even though he was located in California because Yorktown was the center of Research at IBM. There were several IBM Research Labs, but Yorktown was the biggest one and that's where the headquarters was. There were people, various little groups in IBM that were paying varying degrees of attention to Codd's ideas. There was a group in England that was developing something called the Peterlee Relational Test Vehicle, PRTV[6]. Another group at a different place in England was developing something called Business System 12[7]. These were early prototypes based on Codd's ideas of relations. At Cambridge Scientific Center, Raymond Lorie was working on something called XRM, which was Extended Relational Memory[8], another implementation of the relational model. There were some academics that were interested in this work. At UC Berkeley I'm sure you know Eugene Wong and Mike Stonebraker were beginning to study the implementation of the relational model. So there was a lot of work, but it was distributed here and there. Ted's idea was that IBM's efforts in implementing the relational model should be consolidated, or at least coordinated, so that we'd have critical mass to get something done.

Ted organized this symposium at Yorktown Heights, and this was really my first exposure to the power of this mathematical approach to database management. I had studied the CODASYL interface fairly extensively and I could visualize how you would represent a query if there was something that you wanted to retrieve from the database, customers who exceeded their credit limit in a certain year or something like that. I could visualize how you would represent that query using the CODASYL notation. Ted came and he would take some example queries off the top of his head and he'd write them on the blackboard using one of his relational languages. And they were very short and compact and easy to understand. So I began to appreciate the power of this very elegant mathematical approach to accessing data based on its logical properties rather than on its physical representation. This was a kind of conversion experience for me. After that I wasn't as interested in working on CODASYL any more, because I could see that the relational approach had a lot of power and a lot of potential.

There was another guy who was in my group at Yorktown at this time. His name was Ray Boyce. He had graduated from Purdue with a PhD in computer science and joined our group in 1972, just before Codd's symposium. Ray had written a thesis in Purdue on structured programming languages[9], and he was interested in user interfaces, languages, and database systems. Ray and I became very close colleagues. We were impressed by the power of Ted's ideas, but we weren't very impressed by their lack of friendliness to users who didn't have a mathematical background. They were framed in these rather esoteric mathematical concepts, terminology and notation. It was notation that you couldn't type on a

---

[6] S. Todd, S. PRTV, an efficient implementation for large relational data bases. In *Proceedings of the 1st international Conference on Very Large Data Bases* (Framingham, Massachusetts, September 22 - 24, 1975). VLDB '75. ACM, New York, NY, 554-556. http://doi.acm.org/10.1145/1282480.1282546.

[7] See http://en.wikipedia.org/wiki/IBM_Business_System_12.

[8] Raymond A. Lorie. XRM - An Extended (N-ary) Relational Memory. IBM Report G320-2096, 1974.

[9] Raymond F. Boyce. *Topological Reorganization as an Aid to Program Simplification*. Doctoral Thesis. UMI Order Number: AAI7230860, Purdue University, 1972.

keyboard because it had all of these upside down "A's" in it and Greek letters and stuff like that. So we thought, this is not going to catch on until we take these very powerful ideas and frame them in terms of more familiar concepts and notation. We recognized and respected the power of Ted's ideas, the notion of data independence, the relational data model, and the idea of representing data as tables. We loved all that stuff. What we didn't love was the notation with the Greek letters in it.

At first Ray and I worked on a language of our own called SQUARE[10]. SQUARE was an acronym that stood for Specifying Queries As Relational Expressions. We thought we had an angle on representing queries that was different from the relational algebra and different from the relational calculus. We thought of it in terms of mapping. We thought that, if you had a set of values, you could map that set of values into a different set of values by means of a relation. For example, if you had a department number and you wanted to find the names of employees who were in that department you would use an employee table and you would map your department number into names of employees by passing it through that table. And you could compose these mappings, so you could take a department number and find the set of employees in that department, and then you could find the average salaries of those employees, and so on. So we had this notion of a mapping-based language. I think that in retrospect all we were really doing was assimilating Ted's ideas and appreciating them. I think this notion of a mapping would have been just as clear as water to Ted. I think that's what he had in mind all along. But we were sort of disciples and we were learning as we went along, and we invented this language called SQUARE.

Well, at about this time research management in IBM really began to get tuned into Ted Codd's idea that to achieve critical mass with this notion of relational databases we were going to have to have some consolidation. It didn't make sense to have a few people in Yorktown, and one or two in Cambridge, and Ted's group in San Jose all working independently. So there was a reorganization at the end of 1972 and it was decided that people in the Research Division at IBM who were working on relational database systems should all go to San Jose and sit at the feet of Ted Codd and work together to try and create an industrial strength implementation of his ideas. So Ray Boyce and I and Leonard Liu, our mentor and boss, went to San Jose. Frank King went at that same time. Vera Watson, who was another researcher at Yorktown, was part of the same transfer, as was Robin Williams, who was interested in graphics. The set of us, Ray and I, Leonard, Frank, Vera, and Robin, six people, all came to California in early 1973 to join Ted together with Raymond Lorie from Cambridge and some others to form the nucleus of a group that was going to provide a robust industrial strength implementation of Ted's ideas.

Interestingly, Ted himself was a kind of a mentor in the actual construction of System R, but he was not deeply involved in the nuts and bolts of it. Ted was not really a nuts and bolts guy. He was a visionary and a guru. He was an architect but he wasn't a carpenter, and a lot of the people who came to San Jose to work on these ideas were carpenters. They could see that to build an industrial strength database system you had to have indexes, and buffer pools, and B-trees, and locks and things like that. Ted didn't like to work at that level. He liked to be the visionary. In recognition of Ted's work as the father of the relational model he was made an IBM Fellow. An IBM Fellow is somebody who can pretty much do anything he wants to do, and he wanted to do two things. First he wanted to be an evangelist and to promote these relational ideas within IBM and around the world. He did a lot of traveling around, speaking at conferences, visiting universities, arguing with bureaucrats and generally promoting the

[10] R.F. Boyce, D.D. Chamberlin, W.F. King, and M.M. Hammer. Specifying queries as relational expressions: the SQUARE data sublanguage. *Commun. ACM* 18, 11 (Nov. 1975), 621-628. http://doi.acm.org/10.1145/361219.361221.

relational model as a concept. And the other thing that he wanted to do was to work on natural language information retrieval. This is a problem that has been about to be solved for the last 40 years, and in the 1970s Ted wanted to take a shot at natural language information retrieval. So he had some people working directly for him and they were working on a program called Rendezvous[11], which was supposed to be a natural English question answering system. It was written in APL. Ted spent a lot of time on it, but it wasn't a very successful system. So Ted was kind of an advisor, and guru and mentor to the work that was going on in System R, but he wasn't a hands-on type of guy.

McJones: But would it be fair to say that the work he was also doing giving talks, talking about normalizing the relational data model and so on, was certainly helping to create the right environment for you folks to be able to do your work?

Chamberlin: Yes, it certainly was, and Ted would work with people in System R. He was very open and friendly and accessible to people.

McJones: He was a consultant.

Chamberlin: Sure. For example, my friend Ray Boyce who was working with me on languages was also interested in normalization, and he would work with Ted Codd on the theory of normal forms. Ted had defined first, second, third, and fourth normal forms, which were disciplines for designing relations that had good properties that minimize redundancy. Ray Boyce and Ted worked together on the normalization problem and came up with yet another normal form that has been called Boyce-Codd Normal Form. It's a fairly famous discipline that's taught in colleges and you can read about it in textbooks. That was one of the examples of Ted working with someone in the System R project on a specific problem.

<audio break>

Chamberlin: Two of the important figures of System R were Leonard Liu, who was the person who hired me into IBM, and Frank King, who was the manager of the Relational Database project. Leonard Liu became the manager of all computer science research at the San Jose Research Lab, and Frank King was the manager of the database work specifically. I think both of them were very good managers. The organization, focus, and resources that they provided for this project made it possible for us to be successful. One of the things that Frank thought was important was for our project to have a name so that we could make slides about it, write papers about it, and get some recognition. In order to get recognition for something it's good for it to have a name. So Frank called a meeting and said, "You guys are going to have to think of a name for your project." We thought, "Well, that's a waste of time. Don't bother us." But he persisted and said, "You guys are going to have to come up with a name." So for lack of anything else we said, "Well, we'll call ourselves System R." R stood for relations, or maybe it stood for research, or Franco Putzolu even thought it stood for Rufus, which was the name of his dog. It was a little bit of artful ambiguity what the R stood for, but that was the name of our project.

[11] E. F. Codd. Seven Steps to Rendezvous with the Casual User. *IFIP Working Conference Data Base Management* 1974: 179-200, (IBM Research Report RJ 1333, San Jose, California).

More people came out from Yorktown in another wave to join this project, including Franco Putzolu, who was a very gifted programmer, one of the most prolific and accurate system implementers that I've ever met. Also Mike Blasgen came out from Yorktown and he was very interested in the index component of System R and how to adapt B-tree indexes for associative retrieval to objects on a disk. We organized ourselves into two groups. There was what you might call the lower-level group that was interested in physical implementation of the database. They had to worry about issues like how to manage the bytes on the rotating magnetic storage. They were managing the buffer pools, and they were locking records to preserve data consistency, and they were working on algorithms for recovery from system crashes and power failures and things like that. They were working on physical access paths to the data like B-trees and so on. They defined an interface that made all of those services available through a programming interface called the Relational Storage System or RSS. Then the second group was working on more user-facing issues relating to the use of the relational model, the ways in which the relational abstractions could be presented to the user and explained in the form of simple things like rows and columns of tables. This group developed languages for creating and manipulating these relations, and for storing data in them, and for asking questions about them, and for controlling access to them, and for doing database administration tasks: maintaining the data and extending it and changing its properties.

Frank King was the manager of the whole project, but these two subgroups had lower level managers. I was the manager of the language-oriented group, which was called the Relational Data System, and Irv Traiger was the manager of the more physically oriented group called the Relational Storage System. These groups had about six people each in them, so this was not a large project. If you think about some of the projects with hundreds of people participating, this was a very modest effort compared to that. And we were somewhat isolated from the things that were being done in the development divisions of IBM. IBM has several labs in the Research Division, and the Research Division's charter is to develop technology rather than to develop products. IBM has many development divisions that produce products. The mission of research is not to produce products per se but to develop technology that will influence IBM's products, and that's what System R was. It was not a product that was going to be released commercially. It was a feasibility demonstration. It was proof of the concept that Ted Codd's ideas actually made sense and could be implemented in a robust way. And there was a lot of skepticism about that.

It was kind of a replay of what happened when John Backus came out with FORTRAN and the early high level languages were developed[12]. There were a lot of programmers in those days who were very adept at programming the low-level registers of the machine. If they wanted to add two numbers they would write assembly language instructions to load the numbers into registers and add them. This was regarded as a highly developed skill. If you could write really tight code you were a master programmer and you could write very efficient applications. And then John Backus came along and said, "Come on, if you want to evaluate some formula, you just write it out like algebra and let the computer figure out how to allocate its own registers. We can write compilers that'll do that for you. People shouldn't waste their time on that." These highly skilled programmers who were recognized for their ability to optimize computer programs were horrified by this concept: "A machine taking over my job? How could a machine allocate registers as well as I can? I'm very smart." There was a lot of resistance to this idea. John Backus had to develop a FORTRAN compiler that generated very good code from these algebraic

[12] John Backus. The history of Fortran I, II, and III. In *History of Programming Languages I*, R. L. Wexelblat, Ed. History of Programming Languages. ACM, New York, NY, 25-74. DOI= http://doi.acm.org/10.1145/800025.1198345

expressions to prove the concept that you could interact with a machine at a higher level and not have to be concerned with all these physical details.

Well, that was back in the 1950s, but the same thing was happening now in the 1970s with respect to database management. There were database systems available in the world. There was IMS, there was IDS, and there were some other ones too. And these systems were mostly based around these navigational ideas of Charles Bachman and others who basically thought that retrieving data from a database was a highly developed human skill—much like allocating registers, you had to get it right and machines weren't smart enough to do that. But Ted would say, "Tell me what you want, not how to find it, and let the computer figure out how to find it." And "Tell me what you want" was to be done in a non-procedural way. It was not, “Do this, then do that”; it was, “Evaluate this mathematical formula,” such as an expression in the relational calculus, which would then be compiled by an optimizing compiler, which would say, "Oh, well, in order to answer that question I think I'll use this index to find departments, and then I'll use another index to find employees, and then I'll merge departments with employees by sorting them and merging them." That would be some algorithm that the optimizer might choose in order to implement this mathematical abstraction that came from the end user. Well, there was a lot of skepticism that computers could do that, because this is a harder problem than just a high level language like FORTRAN. In FORTRAN you may have an algebraic expression like $x^2 + y^2 - 3z$ or something like that, but the steps that you need to go through in order to evaluate that expression are fairly straightforward. It was widely believed by this time in the 1970s that computers were smart enough to do that. But if you had several tables full of data, and these tables had many columns, and you had a question to ask like “Find programmers in Poughkeepsie who understand FORTRAN and earn more than their managers”, there were many different ways to address that problem. You could start with a fixed point like Poughkeepsie, or you could start with a fixed point like FORTRAN, or you could start by merging the skills table with the salaries table. There was an almost unlimited array of possibilities, and it was hard to enumerate those possibilities, and it was hard to evaluate them. How do you know which one is going to be the most efficient? You had to gather statistics. Well, how many programmers are there, and out of those programmers how many of them know FORTRAN, and how many of them are in Poughkeepsie? The space of possibilities over which you had to optimize was much greater in the database world than it was in the programming language world, and there was a lot of skepticism that it could be done. So one of the things that we had to do, and this came down primarily in my group, in the language-oriented group, was to build a very smart optimizing compiler. We had to choose a language that we could present to users that would enable them to express these abstract concepts. And then we had to build an optimizing compiler that would translate those logical queries or declarative queries onto this physical storage level that was being built in Irv Traiger's group called the RSS. So I think the two most important things that we were working on were the language, which had to be something that users could understand, and the optimizing compiler, which had to bridge the gap, and it was a big gap, between this user-oriented language and this physical representation.

McJones: To prove that Ted's original conception was right you folks had to build a system.

Chamberlin: Yes.

Chamberlin: Just to repeat what I was saying, in the Relational Data System group, the language-oriented group in System R, I think we had two important tasks. One was to develop a language that framed Ted Codd’s concepts in a notation that users could easily understand and exploit, even though

they didn't have formal mathematical training. The second important task was to build an optimizing compiler that would compile that language into the lower-level interface, the physical representation that was supported by Irv Traiger's group, the Relational Storage System. We had to somehow bridge the gap between our user-oriented language and the physical representation. And this was a big job. This was a bigger gap to bridge than was present in programming languages. When you have an expression in a FORTRAN program, that looks like algebra—it looks like $x^2 + y^2 - 2z$—you can pretty well visualize the physical operations that you have to go through in order to evaluate that expression. But in the database world, there are many more possibilities to choose from, many more algorithms and access paths and alternative approaches to evaluating an expression in a database query language. So translating that high level abstraction into the lower level interface is a bigger job.

McJones: Because this basically relates to the fact that FORTRAN was a Turing-complete language. There was only so much analysis and optimization that could be done in that setting, whereas SQL is not Turing-complete, so that gave you much more latitude to consider a wide variety of implementation approaches. Is that a fair statement?

Chamberlin: Yes, I think that's right. Queries often involve joins of multiple relations, and each one of these relations has to be accessed in some way. So one dimension of choice is the order in which you want to do the joins. If you have tables A, B, and C, do you want to join tables A and B first, and then come to C? Or do you want to join B and C first and then come to A? There are many possibilities for the join order. And which one is most efficient depends a lot on things that are not part of the query. There are statistics that the optimizer has access to, like which one of the tables is the biggest, and which one has the most selective predicate. If you're looking for people by their jobs and you're only looking for anthropologists, that may be a very selective predicate, because out of all of the employees, only a few of them are anthropologists. So by applying that predicate early in the process, you can reduce the data space quickly and then you only have to join all the rest of the tables to the anthropologists and not to all the rest of the employees. So there are a lot of opportunities for, you might also say, artificial intelligence here. The objective is to find the answer to the question while doing as few accesses to the disk as strictly necessary. You don't want to rattle that disk because IOs are expensive. So it's kind of an operations research problem, if you like. Your figure of merit is to minimize the number of IOs, and your objective is to find the answer to the question, and there's a very large space to explore in order to do that optimization.

McJones: Right. Basically there are these algebraic properties such as that the joins are associative, so you could do them in any order, for example, and that leads to this explosion of possibilities.

Chamberlin: Yes, that's right. Some of our people worked very hard on the optimization problem—Pat Selinger[13] was one. Pat Selinger had joined our group from Harvard. She had a PhD[14] in applied math, and she came to join the System R project in 1975. She quickly got interested in the optimization problem and just did a wonderful job of organizing it; she wrote a groundbreaking paper about cost-based

[13] Her maiden name was Griffiths.

[14] Patricia Griffiths. SYNVER: An Automatic System for the Synthesis and Verification of Synchronous Processes, Ph.D. Thesis, Harvard University, June 1975.

optimization that's still taught in universities[15]. But she wasn't the only person who was working in this area. Raymond Lorie, Tom Price, and Morton Astrahan all had important contributions to make to the success of this optimizing compiler.

McJones: Maybe now would be a good time to come back to the language, now that we have an understanding of something about the system that it was controlling. So the last you'd said was SQUARE.

Chamberlin: Yes. Ray and I, when we were in Yorktown, attended this symposium that Codd had organized, and we became converts to the relational approach, and we used to play something called a query game. We would dream up questions and challenge each other to find syntax in which these questions could be expressed in a way that might be processable by computer. Natural language is a terrible way to retrieve information from a computer because it's ambiguous. If you say, "Find me all the policemen and firemen," you use the word "and," but you really don't mean "and;" you really mean "or," because you want to find all the people who are either a policeman or a fireman. But the computer doesn't understand that. The computer will see the word "and" and it'll find all the people who are both policemen and firemen, and maybe there aren't any such people. That's just an example of the many kinds of ambiguities and misunderstandings that you get into when you try to use natural language as a computer interface, as Ted Codd was discovering with his work on the Rendezvous project. So we wanted to build a language that was more structured and more rigorous and less ambiguous than English, but we had some criteria for this language, and we became aware that our first attempt, which was this language called SQUARE, was not meeting the criteria. SQUARE was based on the notion of mapping a set of values into another set by means of a relation. The way that we represented these mappings was by a table name with subscripts that represented the columns that we wanted to map. By means of this table, we wanted to map from column A to column B, so A and B would be subscripts on the table name. This wasn't working out because, first of all, it's hard to type on a keyboard. We didn't have subscripts on the keyboard, and although that's not an insurmountable problem, it was an annoyance. It was awkward. And also, by looking at the notation in the SQUARE language, you really didn't get a sense at an intuitive level of what this query meant.

So we decided we'd make a second try, Ray and I, after we both arrived at San Jose and bought houses and moved in our furniture and got settled. We began work on a SEQUEL, if you like, to the SQUARE notation—another relational language, but this time it was based on English keywords. You could spell them out, like SELECT, and other words like that. We tried to retain the notion of a mapping, which we thought was useful, but to frame the concept of a mapping in terms of something called a query block, which was made out of English phrases. So you would say, "SELECT name, salary FROM employee WHERE skill = 'Fortran' AND salary < 50000," or something like that. These short English phrases—SELECT, FROM, WHERE—would introduce the table, or possibly multiple tables that were involved in the query. That would be the FROM clause. And the SELECT clause would identify the columns from those tables that you wanted to retrieve; that's where the answer was. And then the WHERE clause would contain predicates that would govern how these tables were joined together, and how rows were selected out of the tables based on the properties that were of interest. For example, a row from the

[15] P.G. Selinger, M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, and T.G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international Conference on Management of Data* (Boston, Massachusetts, May 30 - June 01, 1979). SIGMOD '79. ACM, New York, NY, 23-34. http://doi.acm.org/10.1145/582095.582099

employee table, a row from the customer table, and a row from the products table might be joined by matching their values in certain columns, and then you might apply a predicate that would do further restrictions, keeping only the employees with a certain job or a certain salary. These operations—of merging tables together and choosing some of the rows and keeping some of the columns—were basically the operators from Ted Codd's original paper. They were joins and selections and projections, but they weren't written in mathematical notation anymore. They were written in English keyword notation. So you could see SELECT and you'd say, "Oh, I sort of know what that means." And WHERE—that would mean a filtering operation, where you'd keep the rows that satisfied the logical condition and you'd throw away the ones that didn't. This wasn't anything revolutionary. This was taking Codd's ideas, which were revolutionary, and wrapping them up in a prettier box, so you didn't have to use notations like upside-down "A's" that non-mathematicians thought were kind of strange and threatening. You would use keywords, which everybody understood and which were easy to type on your keyboard. Really, they meant the same things that Codd's mathematical notation did, but they were easier to teach people and easier to understand, and they preserved the things that we liked about the relational approach. You didn't find anywhere in this language any reference to navigation or pointers or indexes or access paths or locking, or any of these physical concepts. We retained the essential parts, the breakthrough ideas from Ted Codd, which were data independence and non-procedural access, and we found a way to retain these concepts without retaining the mathematical notation that Ted originally framed them in. We were just framing them in a friendlier way. That's where the language came from that we called SEQUEL.

SEQUEL was an acronym for Structured English Query Language. It was put together by Ray Boyce and myself in this group which was the language-oriented part of System R. Ray and I had such a close working relationship that I don't think either of us can take more credit than the other for the initial concepts that came out of the SEQUEL work. The person who deserves the credit for these concepts is Ted Codd, obviously. SEQUEL was just a way developed by Ray and me to make Ted's concepts easier to swallow. The first SEQUEL paper[16] was published—Ray and I were coauthors of it—in May of 1974 at the ACM SIGFIDET conference in Ann Arbor, Michigan. SIGFIDET stood for Special Interest Group on File Definition and Editing. It was the precursor to SIGMOD, the SIG on Management of Data, which is the principal professional organization in database management today. But it was just getting started back in the 1970s because database was a brand new thing. SIGMOD didn't exist yet, but its precursor, SIGFIDET, had an annual conference, and in 1974 it was at the University of Michigan at Ann Arbor. At this meeting was where the first SEQUEL paper, authored by Ray and myself, was first presented.

But the main thing that was of interest at this conference was a showdown. It was a much publicized debate between Charlie Bachman, the principal promoter of navigational databases and winner of the Turing Award for that concept, versus, in this corner, Ted Codd, whose relational data model was disruptive technology, contradicting the notion of programmer as navigator, substituting the notion of data independence, and operating on data in terms of a nonprocedural abstraction. So there was this famous debate[17] between Ted and Charlie Bachman, and to my lasting regret, I wasn't there. Ray and I had

[16] D.D. Chamberlin and R.F. Boyce. SEQUEL: A structured English query language. In *Proceedings of the 1974 ACM SIGFIDET (Now Sigmod) Workshop on Data Description, Access and Control* (Ann Arbor, Michigan, May 01 - 03, 1974). FIDET '74. ACM, New York, NY, 249-264. http://doi.acm.org/10.1145/800296.811515

[17] See session: Data models: Data-structure-set versus relational In *Proceedings of the 1974 ACM SIGFIDET (Now Sigmod) Workshop on Data Description, Access and Control* (Ann Arbor, Michigan, May 01 - 03, 1974). FIDET '74. ACM, New York, NY, 1-144. http://portal.acm.org/toc.cfm?id=800297

coauthored the SEQUEL paper, and because of budgetary considerations, only one of us got a ticket to Ann Arbor, and it turned out to be Ray. So Ray went to Ann Arbor in May of 1974 and got to be a witness to the debate between Ted Codd and Charles Bachman. I think that this was the debate that turned the tide. I think after Ann Arbor, serious people didn't dispute any longer the advantages of data independence and the relational abstraction. There was still some debate about performance. Could you really implement these slick new ideas in a way that would be useful to the Bank of America? Everybody recognized by now that the relational approach made it easier for users to write queries, but what if your business depended on a database being up 24 hours a day, 7 days a week, processing thousands of transactions per minute—could relations do that? We had navigational systems that were doing that, and companies that depended on them. Were they really going to bet their company on this idea from a theoretical mathematician in San Jose? It was a risky business. So there was still some of that kind of skepticism. But I think Ann Arbor put the idea to rest that navigation was a good human interface.

McJones: So at least, for example, the research community from that point on really focused on the relational model?

Chamberlin: Yes, I think so. After 1974, I think the work at the Ingres project at UC Berkeley—Mike Stonebraker's group—and the System R group at San Jose were well focused and deeply committed to the notion that relations were the way to go from an end user's point of view, and that our job was to make them work efficiently.

McJones: It's interesting that you have said "end users" several times. As I understand it, you really had a focus initially that you could even have business people doing decision support kinds of applications using your query language. Is that correct?

Chamberlin: Yes, it is. What Ray and I thought we were doing was making access to databases available to a new class of people that hadn't accessed databases before. We thought they were going to be engineers and architects and city planners and economists, and people like that who were not computer people, not programmers. We thought that these people shouldn't have to turn themselves into computer experts or programmers in order to ask questions. They should ask questions in this English-keyword notation that we called SEQUEL. So with words like SELECT and FROM and WHERE and GROUP BY and ORDER BY and so on, maybe you could teach that kind of notation to people who didn't have any computer training at all, and weren't much interested in computers. They were interested in their domain of expertise. They wanted to study economics or something like that, and they needed to retrieve statistical information about economics, and they needed a language to do that, but they didn't need to be programmers. That's what SEQUEL was intended to do. I don't think it was very successful at doing that. If you look back on it from a perspective of 30 years later, I think you don't find a lot of architects and economists and city planners writing SEQUEL queries. SEQUEL queries are mostly written by experienced programmers who are trained and specialized in doing that, and a lot of them are generated by machines through some kind of interface that's specialized to some domain. For example, there may be some interface that architects like to use. It's built around what architects know how to do, and it uses SEQUEL as a backend when it needs to store and retrieve data and process complex queries.

To some extent, much later, you find non-computer specialists interacting with databases. Certainly using Google, you can retrieve all sorts of information with no training at all. But that's a different kind of problem than we were working on. That's unstructured data and it's what you might call "fuzzy search." It's approximate answers. You can ask questions like, "Find me information about the special theory of relativity," and Google will find lots of relevant information sources. But that's not necessarily all of the sources about the theory of relativity; it's just the ones that Google thought were important, and that's kind of a subjective judgment. So that question doesn't have a precise answer. It has a good answer. Google will find you a good answer and Google is very effective at doing that, and other search engines are too. That's certainly wonderful technology and it's transformed our society. But it's not exactly the same as what SEQUEL was trying to do. A SEQUEL question does have a well-defined answer. It has a correct answer. The correct answer might be 35, and other answers are not correct. So that's different from the kind of approximate search that you find in search engines. Maybe that's a little bit of a diversion.

McJones: No, I think that's very good. It seems people are perhaps more comfortable with a one-table database, such as using an Excel spreadsheet as a database. When you go to multiple tables, maybe that's the point where it takes a professional programmer to make the jump.

Chamberlin: Could be. That's a reasonable theory.

McJones: So now we have a language, we have a system. What happened next? Obviously there were a couple years of really building the code. You guys designed the language, but it took a while to write the optimizer and write the underlying storage system.

Chamberlin: Yes. There was a brief period in the project, lasting about a year, called Phase Zero, which was an interesting period for me. When we organized the project around these two groups—the language-oriented group and the storage-oriented group—we were both starting from scratch. But the language-oriented group needed to have a target for our optimizing compiler, and the storage-oriented group hadn't produced their work product yet, so we didn't have anything to map our SEQUEL language into. So we built first something called Phase Zero[18], with the philosophy you build one and throw it away and learn from your experience. I think that philosophy was from Fred Brooks in the *Mythical Man Month*[19]. He was one of the gurus from the IBM 360 and the OS/360 operating system. Anyway, we believed in that philosophy. We thought, "While we're waiting for the RSS guys to build their physical storage interface, let's do some experiments with these optimizing algorithms for our SEQUEL language." So the first SEQUEL implementation was this Phase Zero implementation, and it was done on top of a very primitive access method called XRM, which Raymond Lorie had developed at Cambridge Scientific Center. The people who worked on the Phase Zero implementation of SEQUEL were Morton Astrahan, Ray Boyce, Jim Mehl, Pat Selinger, Brad Wade and myself. It was a small group, interested in an early implementation of SEQUEL. After about a year, the storage level of the system, the Relational Storage System, began to be well enough defined and robust enough that we did in fact throw away the original

[18] M.M. Astrahan and D.D. Chamberlin. Implementation of a structured English query language. *Commun. ACM* 18, 10 (Oct. 1975), 580-588. http://doi.acm.org/10.1145/361020.361215

[19] F.P. Brooks. *The Mythical Man-Month (Anniversary Ed.)*. Addison-Wesley Longman Publishing Co., Inc., 1995.

SEQUEL implementation and reimplement it on top of the RSS in a much more robust way. I think this was a beneficial exercise for the project. I think we learned a lot by doing this.

The next thing that happened chronologically, at about the time that we were transitioning from the Phase Zero work into our second implementation of SEQUEL, was a tragic thing. Ray Boyce, my close collaborator on the development of the SEQUEL language and my carpool buddy, rode to work with me one day in June of 1974, and I began to hear rumors later that day that Ray had just collapsed at lunchtime in the cafeteria and was taken away in an ambulance. At first I found that hard to believe. I mean, I drove Ray to work that morning and he seemed to be just fine. But it was true. Ray had been taken to Valley Medical Center in an ambulance, and nobody quite knew why. I went to visit him; it turned out that he'd been a victim of an aneurysm of the brain, which is a swollen blood vessel that bursts inside your brain, and he'd lost consciousness, and he was in the hospital, in critical condition. Well, this was a hard thing to get your arms around. Ray was 27 years old. As I told you, he'd come out to California at the same time that I had. Both of us had young wives, and Ray's daughter Kristin was born just before the move to California. At the time when Ray was stricken with this aneurysm, Kristin was less than a year old. His wife Sanndy was working as a nurse, and his parents came out from New York to be with him. We visited him in the hospital, and he had surgery to try and repair the aneurysm. The surgery was not successful, and Ray lived for several days, and then he passed away. Obviously this was a great loss to us personally, and it was a great blow to our work. I've stayed in touch with Ray's family, with his widow, Sanndy. She decided to stay in California. She kept working as a nurse for a while; then she went back to San Jose State and got her master's degree in social work and became a social worker. She finally wound up as a vice principal of a middle school in Mountain View, where she still is. Her daughter Kristin is now—I don't know exactly how old; I'll say 35—and working in the computer industry. She works for a company that does consulting for users on security issues, and she flies all over the world doing this kind of world; does a lot of traveling. Every year Sanndy and I and our families go up to Pinecrest Lake in the Sierras and put our boat in the water and spend a week together at a cabin on the lake. So I'm still in pretty close touch with Ray's widow and his daughter, and we've tried to keep in touch over the years.

McJones: A really sad story. I certainly wish we had Ray beside you right now.

Chamberlin: Yes, so do I. Half of everything that came out of the SEQUEL work belongs to Ray.

McJones: So I guess that put a little more responsibility on your shoulders at that point.

Chamberlin: Yes, it did. Ray and I were good friends. We had similar tastes and similar interests, and I think the main difference between us was that Ray was a lot more interested in management than I was. He enjoyed coordinating things and managing people. So when the RSS and RDS group were originally formed, Ray was the first manager of the language-oriented group. I worked for Ray; he was my boss. When Ray passed away with this brain aneurysm in 1974, I had to step into his shoes. I became the manager of the language-oriented work. At about this time, we were wrapping up the Phase Zero work and doing our second implementation of SEQUEL. At about this time, we had to change the name of the language. The word "SEQUEL" turned out to be somebody's trademark. So I took all the vowels out of it and turned it into SQL. That didn't do too much damage to the acronym; it could still be the Structured Query Language.

Also at about this time, we began to acquire some users. Our manager, Frank King, who was politically astute, said, "The way you get attention in IBM is by interacting with customers, getting some customers on your side. You want some customers to stand up and recognize the value of this technology and demand it, basically, and then IBM will pay attention to it." So we did that. We entered into an agreement called a joint study with three prominent IBM customers. The idea was that we would give them this prototype, and we would tell them, "This is not a product. It's not for sale. Don't take it too seriously. We reserve the right to change it. We don't guarantee that it does anything in particular, but we'd like you to try it out and tell us how you like it." The first installation was at the Pratt & Whitney jet engine manufacturing plant in Hartford, Connecticut. The Pratt & Whitney people wanted to do an inventory control application. You can imagine how many parts it takes to build a jet engine, and all the different suppliers that are involved in acquiring those parts, and keeping track of them and ordering new ones. So that was a very good database application. It was just the sort of thing that relations were good at, and they did a very good job of exercising our early relational prototype in this inventory management task. The second installation was at the Upjohn drug company in Kalamazoo, Michigan. What they were doing was keeping track of clinical trials. They had a lot of experimental drugs that they were working on, and they wanted to keep track of the experimental subjects and their dosage levels and their outcomes, and everything related to the application of testing to the development of new drugs. That was another good application. The third one was with the Boeing aircraft company in Seattle. They were doing engineering design on, I think, the 757 and 767 generation of aircraft. That was another quite different application from the other two.

From each of these applications, we learned from the customers' experience. So I did a certain amount of traveling in those days, visiting these customers. We'd have quarterly meetings with them. They would write reports about how they were using the system and what their experiences were, what problems they'd encountered, what new features they wished for. We'd keep track of all these things. There wasn't any money involved in this at all; it was all for free. Both sides were, I think, benefiting from this experience, and learning more about how to develop relational applications and how to support relational interfaces that were useful. In addition to these three external customers, we had a number of places internal to IBM that were using System R for various different things. The Federal Systems Division in upstate New York was using System R on some kind of a military application. There were probably eight or ten other internal sites that were using this system experimentally. So we would have a little release process twice a year. We'd make a new release and put new features in it. I was the guy that had to write the manuals for all this. I would write the documentation of the user interface and how to write SQL queries and how to create new tables, and so on.

One of the things that we found to be important was building database administration tasks into the query language. Up until now, database administration had been considered a separate process that was done by some designated expert, and it usually involved shutting down the system. If you wanted to create a new table, let's say, in our terminology—our database was made out of tables—the traditional way that this was done in existing commercial database systems was to write a memo to the database administrator and ask him to do it for you, and then he had to schedule a time to do it, and he had to shut everything down and stop the system. He'd do it in the middle of the night, and the next day, if he did everything just right, you'd have your new table. Well, we thought that database administration should be a much more fluid task than that. It shouldn't be necessary to stop the system to create new data structures. In fact, that's one of the things that is a major advantage of the relational data model. The use of the data doesn't depend so much on its physical structure. So if you want to create, say, a new index on a table to speed up access to employees by their skills, you can create a skill index on the employees and that really shouldn't directly affect the way that you ask questions. You ask questions in

the same way. The questions that you asked yesterday will still work, but just now maybe they'll work faster if they're questions about skills. And next week, you may decide, "Well, that skill index really wasn't very useful. It didn't help anything very much. Let's get rid of it and create another index on location." And you can do that. These physical changes in the infrastructure of the database shouldn't require the system to stop, and they shouldn't be done only by some specialist; they should be built into the language.

Different applications may want different views of data. One reason for this is for authorization purposes. Maybe I'm authorized to see salaries but you're not. So you and I have different views of employees. One of us sees a more complete view than the other. Or maybe you can only see the employees that you manage, or maybe you can only see the employees in a certain location, or maybe you can see only the average salary of each department, but not the individual salaries. Well, those are all different views of data. How do you create those views? Well, once again, in the past this had always been done by some specialized administrator. If you needed a new view of data, you would write a memo to the administrator and say, "Well, I need a view that only shows average salaries by department," and he would have to do some mysterious things. We thought about this. Defining a view that groups employees by department and only shows the average salaries? Well, you can think of that as a view definition, but you can also think of it as a query. It's just a way of taking the complete database and abstracting it into an aggregated view, where you only see part of it. That's what queries do. They extract part of the data and return it to you. So we said, "Gee, we don't need a new language for defining views. Our language is plenty fine for extracting things out of stored databases, and view definition is just another aspect of that."

There's also something called integrity assertions. You want to make sure that, for example, your bank balance is always a positive number, or that your charges don't exceed your credit limit, or that no employee earns more than a certain amount. You may have some rules that define the semantics of data that make that data valid in your company, or in your domain—things that you want the database system to enforce for you, to make sure that things don't get out of control.

McJones: So they come from the business logic.

Chamberlin: Yes, they come from the business logic. Once again, we thought, "Gee, this is something that ought to be built into the language, because these are just like predicates." You can say, "Balance due is less than credit limit;" that's like a predicate. That's a true or false statement that can be applied to a row of data that represents your credit card balance. Well, that's the same kind of predicate that you find in queries, and that predicate may be more complex than the example that I gave you. It may have sub-queries in it that go and compute something on the side and then bring it back and compare it to some value in this row. The kinds of things that you need to do to test these assertions to make sure they're true are the same kinds of thing that you need to do to process queries. We thought, "Gee, the SQL language has predicates in it for the purpose of retrieval. We should exploit the same notation for integrity assertions and for view definitions and for authorization, and for other tasks of a database administrator, so the language becomes unified to do all of these different things." Any authorized user—you don't want just anybody changing the structure of the database; you should have some control over who gets to do it—but the control should be a matter of policy. I mean, you should decide who you want to be able to perform database administration tasks, and it doesn't have to be a single person, and you don't have to shut down the system to do it. That was a big hit with our joint study users, because they

were used to database administration being a difficult, bureaucratic, time-consuming process that people couldn't do for themselves. Making it possible for end-users to create their own data and manipulate it, create indexes on it, define assertions on it, and define views of it anytime they wanted, on the fly—that was a big breakthrough, I think, something that made relational systems very attractive to application developers.

McJones: So were those language features, for the definition and authentication and so on, back in the Phase Zero point, or did they come a little bit later?

Chamberlin: Some of them were, and others were developed in later stages of the project. This was an ongoing development, and governed to some extent by feedback from our experimental users.

McJones: At this point, I think there was one more thread of outside activity that had started somewhere along, maybe during your joint studies or perhaps even slightly before. I know they're one of the first commercial participants in the relational database scene. Relational Technology[20]?

Chamberlin: Sure.

McJones: Would this be a good time to say a few words about them? Were you even aware of them? Probably not much, but they were working...

Chamberlin: Sure. There's that, which is an interesting thing to talk about, and there's—Query By Example [QBE] might be worth mentioning at some point. They're kind of contemporaneous, so we'll talk about them both in whatever order you'd like.

McJones: Either one.

Chamberlin: Let me do the QBE thing first, and then we'll do the Relational Technology thing later, because that's the order in which they came, sequentially. They both have something to do with the relationship between our research project and IBM. IBM is a very big company, and had a database product called IMS that was very commercially successful, and they were doing just fine. We were a tiny group of 12 people off in the corner of a research lab trying to implement some ideas that were considered a little bit far out. There were two things that were happening in IBM in the database field. The first thing was they had IMS; they had lots of customers of IMS; they were making lots of money with IMS; and all of those IMS customers were wishing for new features. They wanted this little thing added; they wanted this little thing fixed; they wanted this thing to go faster. Everybody was very busy working on all those requests. Relational technology was viewed as disruptive technology. The attitude was, "We have a system that works just fine. Our customers love it. They'd love it even more if we did this and that, which they asked us to do, and it's taking us lots of time and money and people to do these

[20] At this point the interviewer intended to say "Software Development Laboratories", the original name for Oracle; instead he said, "Relational Technology", which Chamberlin presumably recognized as the later commercial spin-off from the Ingres project.

important things, and we really haven't got time to talk to you, so have a nice day." That was basically the attitude of the IMS group, and it's a perfectly reasonable attitude. That's the attitude I would have had if I was in their job. Then there was another group in the development division of IBM that didn't have a product in the market yet. They were sort of the ad-tech group, and it was their job to think about what the next generation of IBM database products would be like. You might think that research would have some influence on that, and in principle they were supposed to, but IBM is a big company; it had lots of people in it, and there were a lot of different fiefdoms in the company, and in the development divisions there were ad-tech groups that were thinking about future products and had their own ideas about how those should be done. This was back in the 1970s, when IBM was more of a dominant force in the computer industry. So there was a lot of internal competition, more I think than there is today, because if you had the franchise to build the next IBM product, you were in a commanding position, and there were lots of people who would like to be in that position, and so there were lots of internal rivalries. In my view—this is a personal observation, and others might disagree—the ad-tech groups in IBM development in the 1970s suffered from a disease that I will call "biggerism." It was the notion that bigger is better, and in the initial stage of a project, the way you get credit and recognition is by shoveling requirements into the project. So nobody would be writing any code or even any serious designs. What they would be doing was collecting requirements, and the requirements collecting phase tended to expand without limit. So people would be producing these giant documents—the next database system was not only going to be compatible with IMS, but it would also do all of the network-related things that Charlie Bachman's IDS system would do, and it would also do all the relational things that Ted Codd liked, and it would do some other things too. Basically anything that you could imagine that anybody might ever dream up became a requirement. These projects would go through cycles in which the requirements would build and build, and they'd get bigger and bigger, and finally they'd collapse from their own weight, and there'd be a reorganization, and then the process would start over. That was the cycle that was going on.

What my group was doing and what Irv Traiger's group was doing was building a relational database system. We weren't much interested in company politics . Our management was a lot more interested in it than we were, and rightly so, because we were in the business of influencing IBM, and in order to influence IBM, you had to get their attention and you had to gain some credibility, and you had to get your ideas taken seriously. So we were negotiating, painfully in some cases, with basically these two development groups in IBM—the IMS people, who were very busy doing important, productive work, and didn't have time for us, and the ad-tech people, who were creating sort of inflationary dreams, of which they were only too happy to add our requirements to the end of their thousand-page document, but they weren't very serious about actually getting down and building something.

Another thing was happening inside IBM at this time also, and that was that there was another group in the research division that was interested in user interfaces to relational systems. This group was back in Yorktown Heights, where Ray and I had come from. The principal person who was working in that, who had great ideas in that group, was Moshé Zloof, who was proposing an interface called Query By Example[21]. Query By Example was something that nowadays we would call a graphical user interface—a GUI. It was based on the idea that, in order to ask a question, you would give an example of the answer, and then the computer would take that example and learn from it, and go find more examples like that. This was a very powerful notion. It was based on the relational model of data, and it was done in a graphical form on a display terminal, and it made it possible for people, much like the people we were

[21] Moshé M. Zloof. Query by example. In *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition* (Anaheim, California, May 19 - 22, 1975). AFIPS '75. ACM, New York, NY, 431-438. http://doi.acm.org/10.1145/1499949.1500034

going after—people without any mathematical training and who didn't consider themselves to be computer experts, but had some questions they needed to ask—to represent these questions in a very intuitive interface. It wasn't a syntax. It wasn't a language in the sense that SQL was a language. It was more of a graphical interface, and it was a very attractive one, and a lot of people liked it. They'd built a prototype of this Query By Example interface and installed it in a number of experimental sites, much like we were doing, and by and large, people liked it pretty well.

It's my personal opinion that the Query By Example group wasn't working on some of the hard, underlying physical database problems to the same extent that we were. We were building a multi-user system where many users could access shared data at once, but the Query By Example prototype didn't have that property. We were also more worried about efficient access paths and transaction semantics. Some of the more difficult physical database design problems that were going in Irv Traiger's group I think were being addressed more robustly in San Jose because we had more resources to focus on them. We were a larger group. Anyway, what you might suppose is that there would be synergy between these two groups and they could get together and benefit from each other, and that's certainly the way it should have worked out. But in practice, there was a certain amount of rivalry between the two groups. I won't place blame on one side more than the other. I'll just say that each of these groups thought that the ideal outcome would be for the other one to go away and stop bothering them. We're out of tape?

<audio skips>

Chamberlin: At some point, around 1978, IBM management decided that the System R and Query By Example groups should cooperate around a common strategy rather than working independently. Two research managers, Dick Case and Ashok Chandra, were assigned to conduct a review of the two groups. They visited the Query By Example group in Yorktown, and then they visited the System R group in San Jose. They listened to what each group had to say, and when they came to San Jose, we were waiting for them. We had set up a prototype of the Query By Example system that we'd obtained from Yorktown, and we set up a prototype of System R, running on two terminals side by side in a room. We had collected a set of example queries, and they also brought some queries along with them. We created the databases on both systems and we ran the queries side by side, on the Query By Example system and on System R, for the visitors when they came out to do this review. This became known in our group as the "shootout at the OK Corral". We put a sign on the door where this was done and called it the OK Corral. We did a relative performance comparison between the two systems, and System R was a pretty clear winner. I think in all of the queries that both of the systems could do, System R would do it much faster, and there were some queries for which the QBE system would just crash. In hindsight, this really wasn't surprising and in a sense it wasn't fair, because performance wasn't what Query By Example was about. Query By Example was about a user interface, and it was a good user interface, for certain applications.

In my view, queries that are very simple—"Find Paul McJones' salary"—are best done in a graphical interface like Query By Example, which makes them very easy to express. You just write "Paul McJones" in one column and a question mark in the salary column, and that's all you have to do, and doing that visually is a more appealing user interface than writing English keywords. On the other hand, some of our users, it turned out—in System R and later as relational systems evolved—need to write very complex queries that involve many tables joined together in complex ways with complex grouping and

things like that. My experience has been that a formal syntax like SQL, that's based on English keywords, scales better to more complex queries than a graphical user interface does, because you quickly run out of space on the screen. When you apply complex kinds of queries that involve grouping and things like that, you lose the visual appeal. Representing them in a syntax I think scales better for more complex applications. That's a personal view and possibly a biased one about the respective advantages of these two user interfaces.

Anyway, learning that System R ran faster than the Query By Example prototype from Yorktown I don't think was a big surprise, but it led to the conclusion that the way these two projects should cooperate, and the way we should coordinate our transfer of technology to IBM, was that Query By Example should be converted to be an interface to System R, that the semantics of Query By Example should be made consistent with the semantics of SQL, and they should be implemented as two human interfaces to a common substrate so you could have the advantages of both of them. That was the conclusion from this audit that Dick Case and Ashok Chandra arrived at, and I think it was a reasonable conclusion. To some extent, eventually, in the fullness of time, that came to pass. It didn't happen right away. The auditors went back where they came from, and there wasn't any immediate revolutionary impact. Both of the two research groups went back to doing what they'd been doing before, and both of them kept on trying to sell their technology to influence future IBM products, and the IMS people kept right on ignoring us because they were too busy, and the ad-tech groups kept right on saying, "This is wonderful. Let's do both. Let's do everything," and in fact not doing anything. So life continued as before.

<break>

McJones: Okay, Don. As I remember, we had talked about both the development of System R and some of the rivalry between this joint project, QBE, before our break. Could you say a little bit more about how System R and the ideas that are inside of it finally impacted IBM products?

Chamberlin: Sure. I'll try and do that. In a way, we impacted IBM products partly because of our impact outside IBM, and there are some interesting stories around that. As you know, we were working together with some joint study partners to evaluate this technology, and that's one way to get IBM's attention. But another way to get IBM's attention is for some competitive company to be making some money and having some success with a new form of disruptive technology. In the summer of 1978, I was sitting at my desk. We were working on System R and the next release of it for our joint study customers. I'd been seeing some things in the trade press once in a while about a company called Software Development Laboratories that claimed to be developing a relational database system. I hadn't paid much attention to it, but in the summer of 1978, I got a phone call. It was from a guy named Larry Ellison, and he said he was the president of Software Development Laboratories, and they were developing an implementation of the SQL language. Since we were in the research division of IBM, our philosophy of research was to publish our results in the open literature. As you know, many papers[22] came out of the System R project that were published in conferences and journals, describing the language and the internal interfaces of the system and some of the optimization technology and so on. The project was not a secret and, in fact, we'd been telling everybody about it that would listen. And one of the people that had listened and had read some of our papers was Larry. So he called me up and said that he was interested in implementing the SQL language in the UNIX environment. IBM wasn't interested in UNIX at all. We were primarily a

[22] For a bibliography, see: http://www.sigmod.org/sigmod/dblp/db/systems/r.html .

mainframe company at that time. We had some minicomputer products, but they really weren't robust enough to manage a relational database, and there was little, if any, attention being paid to the UNIX platform. But Larry was really interested in the UNIX platform. He had a PDP-11, I think, that he was using as the basis for his SQL implementation, and he wanted to exchange visits with us and learn whatever he could about what we were doing, and in particular, he wanted to make sure that his implementation was consistent with ours so that there would be a common interface with compatible error codes and everything else. I was very pleased to get this call. I thought, "Terrific. This is somebody in the world who is interested in our work." But I had some constraints on what I could do because of my position in IBM. I had to get management approval to talk to somebody on the outside, even though there was nothing secret about our project. Everything that Larry had access to was perfectly available in the open literature. So I went to talk to my boss, Frank King, and Frank talked to some lawyers—there were always plenty of lawyers in IBM that could think of a reason not to do most anything. Sure enough, they said, "You better not talk to other companies who are building products that are competitive with ours. We really don't want you exchanging visits with these guys, so just tell them 'Thanks for your interest, and have a nice day.'" So that's what I did. I told Larry that, unfortunately, due to the constraints of the company, we wouldn't be able to exchange information other than in the public literature. But that didn't slow down Software Development Laboratories. They released their implementation of SQL. In fact, it was the first commercial implementation of SQL to go on the market. It was delivered by Larry Ellison's company, initially called Software Development Laboratories, which later changed its name, I think, to Relational Software Incorporated, and later took on the name of the product, which was called Oracle[23]. As you know, if you drive along Highway 101 in Redwood City and look at the giant 100-foot-tall disk drives over there on the edge of the bay, Larry's had some success with these ideas. And rightly so. He brought a lot of energy and marketing expertise and a completely independent implementation, and was very successful with it, and had a major impact on the industry.

Larry's success had a particularly beneficial impact on our research group, because I think this was the way that we finally got IBM's attention for this technology. Here was an outside company that had beat us to the punch as far as releasing a commercial implementation of these ideas, which were based on Ted Codd's concepts and our language coming out of the research division at IBM, and people were eating it up. They loved it. They were paying money for it. That's a good way to get IBM's attention. So after that, when the ad-tech groups in the development divisions were thinking what future database products would be like, there was more emphasis placed on the relational approach, and they seemed to take it more seriously.

McJones: So that started happening with things like SQL/DS, and eventually DB2.

Chamberlin: Yes. There were actually three groups in IBM that eventually decided to build commercial products based on System R technology. The first one was what you've alluded to, called SQL/DS, or SQL/Data System[24]. This was a low-end database system, not on IBM's flagship platform, the mainframe MVS. It was on a lower range operating system called DOS/VSE, I think, that was designed for smaller machines and for department-level business applications. It needed a database system, and the database mission was owned by the Silicon Valley Laboratory of IBM, which in those days was known as

[23] Later, Chamberlin was sent a copy of version 0.2 of an introductory Oracle manual, accompanied by a cover letter signed by Ellison; he donated it to the Computer History Museum as Lot # X5502.2010.
[24] E. F. Codd. The Significance of the SQL/Data System Announcement. *Computerworld* 15(7): 1981, 27-30.

the Santa Teresa Laboratory. They had identified this piece of work, building a database system for this midrange platform, as work that they couldn't afford to do. They didn't have enough resources in their laboratory to do that work. But there was a group in the Endicott Laboratory of IBM that had some loose resources. They just had a project cancelled and they were looking for work, and they found out that the Santa Teresa Lab had placed this on their out list, and they said, "We can do that. And as a matter of fact, System R would be just the thing to release as a database product on our platform." There was a manager—Bob Jolls—who came to see us with a completely different attitude from any IBM manager that I had ever seen. He came and said, "We want to ship System R. Help us do it. What do we have to do between now and shipping the product? Is there anything we need to clean up or polish up or document?" He didn't want to make thousand-page specification documents about blue-sky ideas. He wanted to take our code and ship it. I had never seen anything like that before. I think it came about partly because you could see at this point that these ideas were getting some traction in the industry. There were spinoffs from the Ingres[25,26] project at Berkeley that were turning into commercial products. Larry Ellison's Oracle company was getting a good deal of attention in the trade press. People began to get the impression that relational database systems were real. They weren't an academic subject; they were a way to make money. Companies like to make money, and Bob Jolls didn't want to think of an excuse why it wasn't time yet to do this; he wanted to actually do it. So the mission to build the midrange database system went to Endicott, and the people there did a nice job. They took the System R code, they adapted it to their platform, they cleaned it up a little bit to make it more robust, they wrote user manuals for it, and they shipped it out the door. That was the first group.

The second group was a query group. They wanted to build a query product; a decision support system. It had an internal name of VS/Query, but it was eventually shipped under a different name—QMF (Query Management Facility). What these people eventually did was what the research audit committee had recommended, which was to take the Query By Example interface, adapt it to have common semantics with SQL, and ship a decision support product based on System R code that had dual user interfaces: SQL and QBE. It took them a while to do this. SQL and QBE didn't have identical semantics to start with, and they had to figure out how to reconcile them, and there was a lot of resistance to taking research code, because it didn't have enough comments and it didn't have the right conventions for the names of variables, and it hadn't been reviewed by the correct process, and because of all these reasons, there were endless delays until all these things could be fixed. But eventually they did ship this query product.

The third group that was interested in commercializing SQL inside IBM came out of the ad-tech project that I alluded to, which for a while was named Eagle. They went through one of their boom-and-bust cycles, and it became embarrassing that they had spent so many resources over such a long time making plans and didn't have anything to show for it. So this group, prompted I think partially by the commercial success of relational systems outside IBM, also got serious, in the early 1980s, about shipping a relational product. The quickest way to do that was to adapt the System R code to the mainframe, to the MVS platform. Once they got onto that paradigm, they also did much the same as the Endicott group did, and produced a product which became known as DB2, which was shipped in 1983. So Oracle beat us to the marketplace with our own ideas by a couple of years, but when IBM got serious about commercializing relational databases, the System R algorithms and code and technology made it

[25] G.D. Held, M.R. Stonebraker, and E. Wong. INGRES: a relational data base system. In Proceedings of the May 19-22, 1975, National Computer Conference and Exposition (Anaheim, California, May 19 - 22, 1975). AFIPS '75. ACM, New York, NY, 409-416. http://doi.acm.org/10.1145/1499949.1500029

[26] Later in the interview, Chamberlin expands on the impact of the Ingres project.

into three different IBM platforms, which were released in quick succession in the early 1980s, and which have become quite successful. [27]

McJones: Indeed they have. It was around this time that you took a different turn in your career. Is that correct?

Chamberlin: Yes. I was the guy who wrote the manuals for System R. I personally wrote pretty much all of the user documentation for System R customers, both internally and externally. I got interested in this process of producing documentation, of producing publishable material, and the tools that I had to work with were just awful. I didn't like them at all. At about this time, personal computers were beginning to appear. This was 1980, around the time the IBM PC was released.[28] The Apple II had been out there in the world for a couple of years, and people were interacting with these systems on a very direct personal basis. It seemed to me that the production of documents should be an interactive process, and it wasn't in the way that I was doing it. I was using mainframe-based systems where you had a markup language; you'd write a little program that told the computer how to produce your document: skip a line, change the font, indent five spaces, and so on. You'd have little procedural commands like that, that told the computer what to do. And I thought, "This is like the old days of database systems, where you retrieve data by telling the computer what to do: go to this record, use this index, follow this link." It's all at a physical level. In database systems, we raised that level. We made it more logical. We made it more abstract, and the same thing could be done in document systems. Instead of saying, "Skip two lines and indent five spaces," you could say, "This is a heading," and the computer would know what headings look like. And in fact, maybe headings look different on different devices. If you're displaying something on a screen, you do one thing, and if you're printing it on a printer, you do a different thing, depending on what kind of a printer it is. Maybe information retrieval systems would process headings in a different way. They would help you to categorize documents according to their subject, and so on.

So I thought that documents should be marked up using logical markup rather than physical markup, much the same way that data should be retrieved by saying what you want to know rather than how to find it. You might describe this approach as, "Tell me what it is, not what it looks like." This wasn't a brand new idea of mine. This was an idea called generic markup, and there were several threads that had led to this idea. One of them was in IBM. Charles Goldfarb, Raymond Lorie, and Ed Mosher at IBM's Cambridge Scientific Center had developed a language called Generalized Markup Language—GML. The joke was that GML stood for Goldfarb, Mosher, and Lorie. There were some products based around this language, but they weren't interactive products. You would mark up your document with GML and process the whole thing in a big batch job, and out would come a print file. And if you made a mistake, you got to go back and fix the mistake in your source code, and then run it again. Well, I could see that personal computers were more interactive than that, or they could be. So I wanted to have an incremental reformatter, where you would see the document, and if you wanted to change something, you'd edit the source code and see the changes right away without having to reprint the whole document. The problem was that we didn't have good hardware to do that on, in IBM at least. There were people at Xerox PARC [Palo Alto Research Center] who were working on the same problem, and they were braver and had a better vision, I think, than my little group did. They developed their own hardware. They developed bitmap displays and laser printers and things like that. We weren't that ambitious, but when the System R project ended, I decided to do some research in developing tools for document processing.

[27] Later in the interview, Chamberlin reflects on what made System R such a successful project.
[28] The original IBM PC, model 5150, was released in August 1981.

The System R project came to an end at the end of 1979. We formally transferred our technology to the product groups that were interested in it. We made a bunch of videotapes[29] about everything that we had learned from System R and we gave them to the product divisions, and then we split up and went in different ways. System R was viewed as a modest success in those days, so I sort of had a chit to do what I wanted to do. What I wanted to do was to try and create more interactive, high-level tools for processing documents. So some of the System R people—Brad Wade and Don Slutz and a couple others—went with me and formed a little group to work on document processing. But we were seriously constrained by the kinds of displays that we had available. The IBM PC had a green screen with 24 lines and 80 characters, and all the characters were the same size. You couldn't do anything serious with that. Even the mainframe displays weren't bitmap displays like you find today. The best we could do was to use something called a graphic attachment. You had a mainframe terminal and on the side a second display on which you could draw vector graphics. And with vectors you could draw fonts if you wanted to. So we developed a system called Janus[30]. It was called Janus because it had two faces. It had the mainframe terminal on which you would write your generic markup and it had the graphic display on which you would see the formatted page. Janus would incrementally reformat one page at a time. If you edited something on page 68 and then you pressed enter, then right away you'd see page 68 appear on the graphic display so you could see what it looked like. We were very enthusiastic about this. We thought it was pretty neat and we wanted IBM to pay some attention to it. But once again we ran into some of the same problems that we had had in the database world, trying to get somebody to take this technology seriously. Eventually, we got it released as something called an FDP. An FDP is a Field Developed Program. It's something that IBM ran across that somebody produced somewhere, maybe a customer or maybe some crazy guy in a research lab. It's a kind of product that you can buy if you know about it and demand it but nobody is going to promote it for you. So Janus was released as an FDP. We had 30 or 35 customers. We also had internal use all over IBM. People used it to make slide slows. We had a mathematical formula processor in it, so it was really easy to produce mathematical papers using Janus and you could see your mathematical formula right on the display. Scientists loved it but IBM never took it seriously. And that was what I was working on during the 1980s.

In the latter part of the 1980s it really became clear that technology was outrunning this vector graphic display that we were working on. There wasn't anybody else in the industry that had a display like that. Bitmap displays were now finally becoming available even inside IBM. And in particular, in our research lab we had a new generation of personal workstations that had lots of storage and lots of processing power and bitmap displays, following on from the work that was done at Xerox PARC, which had finally penetrated into the rest of the world and was filtering down to us. So we redid the Janus system. We changed its name. We called it Quill,[31] but it was still based on the idea that you had both a logical representation of the document and a physical representation, what it was and what it looked like. And you had a style definition language that enabled you to control the mapping between the logical and the physical representation. And with Quill, you could edit either one of them. You could edit the physical

[29] Copies of these videotapes, along with the System R source code and documentation, were donated by IBM to the Computer History Museum; see Lot X4095.2007.

[30] D.D. Chamberlin, J.C. King, D.R. Slutz, S.J. Todd, and B.W. Wade. JANUS: An interactive system for document composition. In *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation* (Portland, Oregon, United States, June 08 - 10, 1981). ACM, New York, NY, 82-91. http://doi.acm.org/10.1145/800209.806458

[31] D.D. Chamberlin, H.F. Hasselmeier, A.W. Luniewski, D.P. Paris, B.W. Wade, and M.L. Zolliker. Quill: An extensible system for editing documents of mixed type. In *Proceedings of the Twenty-First Annual Hawaii international Conference on Software Track* (Kailua-Kona, Hawaii, United States), 1988, B. D. Shriver, Ed. IEEE Computer Society Press, Los Alamitos, CA, 317-326.

representation or you could edit the logical representation. And we had a collection of specialized editors. We had an editor for tables and an editor for graphics and an editor for mathematics. And all of these editors would cooperate with each other—we had something called the plug interface, which basically said that every editor had to be prepared to give a frame to some other editor in which that other editor could do its thing. So when you're editing a table, you use an editor that knows a lot about tables and has commands for editing tables. But inside a cell of a table maybe there's a graphic picture or maybe there's a mathematical formula. And while you're editing the formula, you use a mathematical editor that knows a lot about math and has a different set of commands. And you can nest these objects to any number of levels so you could have math inside graphics inside tables inside text or something like that. And whatever level you were editing at, you would interact with the specialized editor for that kind of medium. We built a working prototype of Quill in the late 1980s, and once again we tried to get IBM interested in it. There was a moderate level of interest in a development group in Bethesda, Maryland that had the mission for building composite editors for IBM. They were interested to some extent in our technology. They invited me to come and spend the summer in Bethesda, which I did and I had a wonderful summer. My kids were 12 and 15. We spent every weekend going to the Smithsonian Museum and to the Naval Academy and we toured the whole Washington metropolitan area and had a great time. During the weekdays I would try to transfer some of this Quill technology to the composite editor product that was under development in Bethesda.

But there were bad times on the horizon for IBM. The late 1980s and around 1990 were not a prosperous period for IBM as a company. The company had made some bets on OS/2 and other technologies that might have been very good from a technological point of view but weren't well accepted in the marketplace. Microsoft was in ascendancy and IBM went through a period of retrenchment during which they focused on their core businesses and abandoned a lot of non-essential things, and the population shrank to approximately half of its former size. One of the things that went away was the composite editor group in Bethesda. And my research project on document processing called Quill pretty much vanished without a trace. My job in 1990, which was probably the most discouraging year of my career, was to find jobs for all of the people that had been working on my project and then find a job for myself. I'm proud to say that everybody that worked for me that wanted to stay in IBM had an opportunity to do that. One person went to Rochester, Minnesota and one went to Boulder, Colorado, and some of them stayed in San Jose but everybody found good jobs. A couple of people got disgusted with IBM and found better jobs on the outside and left, but that was their choice.

What I wound up doing was going back to the database world. I had been completely out of the database business for 11 years from 1980 through 1990 inclusive. The database people were downstairs and I had sort of kept in touch with them at lunch but I hadn't been directly involved in their work. When I had left the database business, we were in the preliminary stages of technology transfer to some IBM products that were promising to use our technology, and maybe they would but they had made a lot of promises before. I was kind of skeptical at the time, tired of the process and ready to try something new. Eleven years later I went back down the stairs, and oh my God, I was considered a kind of local celebrity. During the 11 years that I had been away from the database world, SQL had become adopted by IBM and Oracle and DEC [Digital Equipment Corporation] and Sybase and Microsoft and many other database vendors.

McJones: So it became a standard.

Chamberlin: It had been adopted by ISO [International Organization for Standardization] and ANSI [American National Standards Institute] as an international standard. Relational database systems had become ubiquitous. Most of the world's business data was either in relational systems or was in the process of transitioning into relational systems. And I was remembered as that guy who used to work downstairs and had originated the SQL language. So they not only welcomed me back into the database department but they gave me a nice office and treated me like a person who had made a contribution that they could remember and appreciate. I was very grateful that I had a place back in the database world. I went back to the database business at the end of 1990 and basically I've been working there ever since. IBM at that point in time was finally getting serious about the Unix environment, where Larry Ellison's company, Oracle, had pretty much taken over that whole platform. It was widely used in government and in universities and in commercial Unix applications. IBM now wanted a piece of that pie, so they needed to adapt their relational products to run on the Unix platform. There was a big project going on to do that. This time, the project was being done cooperatively between the research group and the product developers in IBM's Toronto and Silicon Valley Laboratories. There was a three-way cooperation to develop this technology and release it on the Unix platform. There weren't any "not invented here" problems anymore. There wasn't any resistance to this technology. Everyone was in a big hurry to finish up this code and get it shipped. People were working long hours and in a very good spirit of cooperation.

I was given very pragmatic jobs to do. I was supposed to work on the SQL parser to bring it up to compliance with the international standard because our parser didn't have all of the features that the latest version of the standard had. I worked on the catalogs, which contain the metadata that describes the structure of the database. I ran the conformance test that showed that IBM's DB2 product conformed to the current level of the international SQL standard. We had the guys from the National Institute of Standards and Technology come to my office and we ran the test and got certified. That was gratifying for me, that I was able to do that. I was given a very plum assignment, I thought. I wanted to write a book about DB2, especially about the new features that we were putting into DB2 in support of the standard. IBM was very generous with me; they gave me permission to spend time and resources writing this book. I did it partly on my own time and partly on IBM's time. They gave me an assignment to be a DB2 power user and to exercise every aspect of the system that I could think of and put challenging applications on it and try to break it and report all of the bugs and problems that I found to the developers, and to develop sample applications and write about them in my book. I just loved doing this because I'm sort of an end user application-oriented person. I like to be involved in languages. I like to do application development. It was a very good synergy between the book I wanted to write and the sort of user-directed, user-centered testing that the product groups needed that I could contribute to. So I worked on that for a while.

When the product was shipped and my books were done—I wrote two of them[32, 33]—I got involved in what's now called object relational work. The object-oriented paradigm for application development was in ascendancy and people had become convinced that you should process data in terms of objects. Relational systems were based on Ted Codd's original ideas of rows and columns. The things that were in the rows and columns were mostly small, simple things like numbers and little bits of text. But it was obviously possible to store more complex things in tables. You could store objects in tables. And objects could have complex internal state and behavior—they could have methods associated with them that you

[32] Don Chamberlin. *Using the New DB2: IBM's Object-Relational Database System.* Morgan Kaufmann Publishers, 1996. ISBN 1-55860-373-5.
[33] Don Chamberlin. *A Complete Guide to DB2 Universal Database.* Morgan Kaufmann Publishers, 1998. ISBN 1-55860-482-0.

could invoke. When you process these objects using a language like SQL, you could invoke their methods, or you could retrieve a whole object out of a database table and deliver it into some object-oriented programming language like Java. Interfaces like JDBC (Java Database Connectivity) were being developed for interfacing object-oriented languages like Java to database systems. So I worked on these object-relational ideas for a while because I was kind of into the object-oriented paradigm. I could see that this was another way to raise the level of abstraction at which users would interact with databases, from rows and columns of simple text to more complex objects.

Then in the late 1990's, after I had been back in the database game for maybe eight or nine years, people began to be interested in a new data format called XML, Extensible Markup Language. I thought this was a wonderful development because extensible markup language was something that had come from the heritage of the publishing industry. It was a markup language, sort of like the generalized markup language that I had used back in my Janus and Quill days, where you identified the logical structure of a document. But people were saying, "We could use this not just for documents but also for data." By representing data with markup you make the data self-describing. And so you'd have a large data object like a purchase order and the parts of the purchase order would be labeled with tags using this extensible markup language. Every purchase order might be a little bit different. They all didn't have exactly the same parts, but you could tell the structure of the purchase order by looking at the tags. The tags would represent metadata that was mixed together with the data. There would be a lot of numbers and things on this purchase order, but you could tell how to use them because the metadata, the self-describing part, was mixed right together with the data. So this brought together the two main threads of my career. I'd been interested in database management and raising the level of abstraction there and I'd been interested in document processing and raising the level of abstraction there. What XML was doing was pulling these two things together and saying that data and documents are really the same thing. They could be represented in a common form and they could be made self-describing. And what we need now is a query language for this new self-describing data format. As it turned out, there were a number of people in the industry who were interested in this, in particular, the W3C, the World Wide Web Consortium that was responsible for Web-based standards like HTML. They convened a workshop at the end of 1999 in Boston to think about what a query language for XML would look like. They decided that the world needed another query language and they formed a working group to develop a query language for this new data format. Well, I found this pretty exciting because it combined the things that I was interested in and I liked working on languages. So I got myself appointed to be the IBM representative to this W3C working group to design a query language for XML. There were people in the group from IBM and Oracle and Microsoft and Bell Labs and several other places. There were probably 20 people in the group. We had weekly phone calls and a meeting about once every two months face-to-face somewhere. And I really enjoyed doing this work. This was standards work. It was language development work. It was interesting because I enjoyed working with the people. They had different agendas and different points of view and different areas of expertise. Working with them, I thought we had a pleasing synergy, especially in the early days of this work.

McJones: So has that reached a good state of completion, XQuery?

Chamberlin: Yes. The first thing that the working group had to do was to convince ourselves that we were justified in inventing a new language. The representatives from Oracle, in particular, didn't think that a new language was justified. They said the world has a standard widely-used query language, which is SQL. SQL by this time had a specification that you had to carry in both hands, that had been developed by the international standards organization. And they said, "Well, there are lots of features in SQL now,

so let's add some more so that we can process XML data." So I was in the very curious position of arguing that the language that Ray Boyce and I had invented in 1974 was not the right answer to querying XML databases—that we needed a new language and we should stop working on that one. So with some help from other people in the working group who agreed with me, we were able to identify all of the reasons why XML is semantically quite different from relational data and justified the development of a new language that came to be known as XQuery. I worked in particular with another member of the working group, Jonathon Robie. Jonathan was working for Software AG, a German company. In the middle of the process he left Software AG and went to DataDirect Technologies. Anyway, Jonathon and I had a working relationship that reminded me of my relationship with Ray Boyce 25 years before. We were sort of on the same wave length about what a language ought to look like. We eventually teamed up with a third person, Daniela Florescu, who was working at INRIA [The French National Institute for Research in Computer Science and Control] but she kept changing jobs also. The three of us made a proposal to the working group of a particular syntax for an XML query language. We called it Quilt[34,35] because we were trying to draw together good ideas from several sources. Mary Fernández at Bell Labs had designed a query language called XML-QL[36] that we liked very much. We stole a lot of stuff from that. There was an existing standard called XPath[37] for retrieving data out of tree-like structures and we stole quite a lot of stuff from that. We tried to paste these things together, to do some cherry picking and put together concepts into a language that we thought had some coherence. And because of its multifarious sources we called it Quilt. The three of us—myself, Daniela Florescu, and Jonathon Robie—brought this Quilt proposal to the W3C working group as one of the candidates to be the syntax for the new language to be called XQuery.

There were actually three proposals that were brought to the working group as candidates and they were all discussed and evaluated and debated over a period of time. Ultimately, the Quilt proposal was adopted as the basis for XQuery. That didn't mean it was done. This was just the beginning of the process. There was a lot of refinement that was done after that and many generations of changes. But we began making working drafts based initially on the Quilt proposal and later on refinements to it as they evolved. A couple of times a year we would publish on the Web another working draft of the XQuery language as it would evolve. The XQuery language eventually became a W3C—they don't call them standards, they call them recommendations, but it's the same thing. It's a W3C recommendation that was finally adopted in January, 2007[38]. And that work is actually ongoing. They're adding a new version with additional query features and an update language, text search, and other things. So I enjoyed this process very much. I enjoyed the people. I enjoyed the material we were working on.

Working on XQuery was a very different experience than the development of SQL. SQL was done sort of in a closet. It was done by Ray Boyce and myself with some help from Ted Codd and some other people in the research division at IBM. Nobody cared about it. Nobody tried to influence it. It just belonged to us and we could do anything we wanted to with it. We didn't have any constraints. We didn't have any guidance. We made a lot of mistakes that eventually were corrected over the years as the standard

[34] Donald D. Chamberlin, Jonathan Robie, Daniela Florescu. Quilt: A Query Language for XML. *Proceedings of XML Europe,* Paris, France, June 2000.
[35] Donald D. Chamberlin, Jonathan Robie, Daniela Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. Invited paper, *WebDB 2000 Conference*, published in *Lecture Notes in Computer Science*, Springer-Verlag, 2000.
[36] See http://www.w3.org/TR/NOTE-xml-ql/ .
[37] See http://www.w3.org/TR/xpath .
[38] See http://www.w3.org/TR/xquery/ .

continued to evolve. So it was a smaller and more personal process and we had more control over it. XQuery wasn't like that at all. XQuery was done in the full glare of international publicity. It had representatives from 20 different companies. Everything was done in public on the Web. A lot of people were interested in it and there were criticisms of it including some scathing denunciations of it in the trade press every time we came out with a new version. There were violent arguments all of the time between people with different points of view about what it should look like. And so nobody had control over it. A lot of people had influence on it. I was the editor of the standard because I liked to write, and so I wrote most of these things, the working drafts that got published on the Web as the standard evolved over the years. I was kind of in the thick of it and taking a lot of the flack, but it was an exciting process. And I think it was a process that took longer but produced more consensus. I think there were more people who bought into it. By the time we published the W3C recommendation for XQuery in 2007 all of the companies involved had already implemented it. We knew they were implementing it as we went along. IBM had an implementation, Oracle had one, Microsoft had one, DataDirect had one. There were half a dozen start up companies that had one because it was all out there in the open. Everybody could see how it was evolving and everybody was keeping pace with the language as it evolved. So in a sense it was a more exciting process because there were more people involved in it. There was more give and take and you could see that it was going to have an impact. In System R, you always wondered whether this was going to just sink without a trace because it was hard to get anybody interested in it. But with XQuery, you could see that people were investing and spending money on it, and it was probably going to go somewhere.

McJones: Very interesting comparison. Yes, with SQL, you had some people like Larry [Ellison] jumping on board right away but you had a lot of people resisting; doing QBE. The Ingres people did QUEL and even the standardization process, I think, was a pretty raucous thing with people proposing different things.

Chamberlin: That's right.

McJones: And yet, I guess the power of Ted's relational model meant that there was just this tremendous traction there.

Chamberlin: The hero of this story is Ted Codd. It was Ted's ideas that formed the basis for everything that we did. He was the guy who had the vision to see that you could raise the level of abstraction at which people interacted with databases. Once that idea was in place it was a matter of plumbing. It was complicated plumbing. We needed a lot of plumbers and carpenters to put all of the pieces together and make it work and package it in a way that people could relate to it and not be frightened away by a funny looking notation. But that was a matter of packaging and implementation based on the breakthrough ideas of Ted Codd[39].

McJones: But you took it to be a system-describing language. Ted had a data model and a query language and you turned it into a language for data manipulation, data definition, data authorization and control.

[39] For an early but detailed survey of relational database principles and implementations, see: D.D. Chamberlin, Relational Data-Base Management Systems. *ACM Comput. Surv*. 8, 1 (Mar. 1976), 43-66. http://doi.acm.org/10.1145/356662.356665

Chamberlin: That's true. We extended the ideas to a broader domain than Ted had done in his original papers.[40] The original query languages that were published by Ted didn't have, as you say, update operations. And of course, if you're the Bank of America, you're going to want to update your data from time to time. So that was an important part of the work.

McJones: Don, before we continue, I'd like to loop back around to the 1970's and ask a couple more questions about that period.

Chamberlin: Sure.

McJones: As the System R work was going on, there was, as you mentioned earlier, this Ingres project going on about the same time. Could you just say a little bit more about that project and its impact?

Chamberlin: Sure. Ingres had a huge impact on the acceptance of relational ideas, especially in the academic environment. Probably the two groups that were most active in the 1970s in developing relational technology were the System R group at IBM and the Ingres group at UC Berkeley under the direction of Professors Michael Stonebraker and Gene Wong and some others. They had very similar goals, and similar approaches and similar sizes. There were many parallels between these two groups, and there was some exchange of people and ideas between the two groups. Jim Gray had come from Berkeley and he was very prominent in the System R work. Bruce Lindsay came from Berkeley and joined System R. There were a number of people from Berkeley[41], including Randy Katz who spent a summer with us. I couldn't even name all the Berkeley people who came to IBM on a temporary basis to work during the summer, or for a postdoc or something of that sort. Ingres was developing a relational database implementation on the UNIX platform and distributing it, I believe, on an open source basis[42] to other universities, and ultimately made a spin-off into a company that commercialized their product. It was based on a high-level language called QUEL which was, I think, more closely modeled on Codd's relational calculus than SQL was, because it was based on the notion of bound variables and quantifiers, much like Codd's calculus language. There was exchange between the two groups in the sense of mutual visits in both directions and seminars, and papers were published by both groups and things like that. Once in awhile, there was some rivalry between the two groups. I think that was on a friendly basis, for the most part. People were a little bit guarded about their ideas some of the time, because we thought we had a great idea that we were about to publish, and they were working on similar things and vice versa, so you wanted to make sure that credit went where it belonged. But I don't think that was a serious problem. I think it was, for the most part, a friendly association. And certainly, Ingres is one of the threads that led to the commercial success of relational databases[43]. Mike Stonebraker didn't stop

---

[40] For Chamberlin's account of the contributions of IBM researchers to the database field, on the occasion of the 15th anniversary of the IBM Almaden Laboratory, see: Donald Chamberlin. Relational Database: Putting the World Online, June 21, 2001. Appendix to: Donald D. Chamberlin, OH 329. Oral history interview by Philip L. Frana, 3 October 2001, San Jose, California. Charles Babbage Institute, University of Minnesota, Minneapolis. http://www.cbi.umn.edu/oh/display.phtml?id=317

[41] The interviewer, who worked in the System R RSS group during 1975-1976, attended Berkeley, where he had worked with Gray and Lindsay on a timesharing system.

[42] University INGRES, as it was known, was placed in the public domain and was developed up through Version 8.9, which is archived here: http://s2k-ftp.cs.berkeley.edu/ingres/ .

[43] For a discussion of the origins of many of the relational database systems, see: Paul McJones, editor. The 1995 SQL Reunion: People, Projects, and Politics. Technical Note 1997-018, Systems Research

with the end of Ingres. He went on to Postgres and to Illustra, and he's had several startup companies and done very well, elected to the National Academy of Engineering and become quite well-known in the database field.

McJones: Since Ingres was an academic project, one of its purposes was to help turn out more Ph.D. students. Those people went out and had a lot of impact, too.

Chamberlin: There are alumni from UC Berkeley everywhere you look in the database industry. The original products that spun off from the Ingres project implemented the QUEL language. Eventually, because of the commercial success of the SQL-based projects from IBM and Oracle, and because of the international standard that grew around SQL, the Ingres spin-offs adopted SQL, first as an alternative interface and later as their primary interface. So over a period of time, their user interfaces converged to ours, but their technology, of course, was developed independently.

McJones: A final question on the 1970s era: could you reflect on what made System R a really successful project?

Chamberlin: Okay. It was a very great privilege for me to be involved in this work. I think I'm a lucky guy, because I was in a propitious place and time, and a lot of our success had to do with being in a propitious place and time. Part of that came from the power of Ted Codd's ideas. I think Ted was an authentic genius. As I said, he wasn't a carpenter, but he laid out the direction for all of us carpenters to come along and finish the work. So the first thing you needed was a great idea, and that great idea came from Ted Codd. The second thing you needed was astute management to assemble the resources, focus the work, keep us all marching in the same direction, fight the political battles, and generally attract attention and respect and credibility. I think that came from the managers of the System R group, Frank King and Leonard Liu, and I think we wouldn't have been successful without that. Another thing that made this work successful was that it was the right time for this to happen in the industry. People were putting their data online. They didn't keep their ledgers in loose-leaf binders with a ballpoint pen anymore. Everybody in the world who was using persistent data in their business was ready, at this point in history, to put it online, and they needed a way to do that. And we were in the right place at the right time to make that possible. We produced a system that allowed users with a very low entry cost to learn how to interact with online data, and how to administer it, and how to create databases with tables, and put them on their computers, and load them up with data, and manipulate the data and query it and generate reports from it. It wasn't that hard to learn. We made it easier than it had been before. People needed to do this, and we gave them a way to do it and they ate it up.

I think the fact that SQL was, in its early stages, standardized through ANSI and ISO, made a big impact. I wasn't directly involved in that process, because during the 1980s, when this was going on, I was off doing my work on document processing. But some people from IBM and other companies, who were interested in SQL, the Oracle company being one, got together and made a very formal, well-specified, rigorous international standard out of SQL. This was a *de jour* standard, published by an international standards organization. Conformance with the standard was required by the United States government—

they had a Federal Information Processing Standard, FIPS 127[44] , which said that if you wanted to sell database systems to the government, you had to implement SQL. That standard created a framework within which people could share resources. If you wrote tools for relational database design, or if you wrote books about relational databases, or if you taught courses on relational databases, or if you had anything to do with the database industry and you were doing it within the framework of the SQL standard, you knew that your products would be interoperable with other peoples' products. There was an established marketplace for that, so you were not out there on your own. I think the standard was very helpful in coalescing the database industry around the SQL language, and it also had a very beneficial effect in improving the language. There's been a fair amount of criticism about the early specifications of SQL. It had some problems with lack of orthogonality and with some other things, and the standard provided a controlled way in which the language could evolve. And people who had grievances, and suggestions, and complaints and wishes could have an avenue to bring these things forward and have them heard. And there were responsible people who could control the evolution of the language. They didn't exercise a lot of restraint, in my opinion, about what they put in the language. The language grew from our 20-page research paper to about 3,000 pages of specifications now in the International Standard. But I still think it was very beneficial to have that growth occur in a controlled way, administered by a recognized body that gave people a voice and a greater influence of the evolution of the language.

So I would say to be successful, you need two things. You need to have an opportunity, and you need to be prepared to exploit the opportunity. And these two things came together in a very synergistic way. I think the state of the industry being at the right time, receptive to these ideas, and Ted Codd's breakthrough ideas provided the opportunity. And I think the brilliant people that were assembled by IBM Research to work together, Jim Gray working on transactions, Pat Selinger working on the optimizer, you know, each of the individuals who was involved in the System R Project—and there were, over time, probably 20 of them who made spectacular intellectual contributions to this work—proved that these ideas were not daydreams, but were of practical significance. I've never been in a place where there were so many smart people working so productively to take advantage of this historic opportunity. You need the opportunity and you need the execution, and neither one of those is sufficient without the other. And I think System R was successful because we had them both.

McJones: Okay, to wrap up this interview, why don't we talk about what you've been doing since the XML work?

Chamberlin: Okay, yes. I came back to the database business and worked on XML query in the W3C working group for a number of years. When XQuery was adopted as a W3C recommendation, I began to think that I'd been doing the same thing for quite a while now, and it was time to try something new. I retired from IBM in the fall of 2008. It seemed to me that what I would like to do next was to go back to what interested me in computers when I was in college. I think computers are fun to play with. I like writing little computer programs, and I would like to communicate that sense of excitement to young people. So I went over to the University of California at Santa Cruz and asked them if they'd let me teach a course. I wanted to teach a programming course to undergraduates. And they said okay. They gave me an appointment as a Regents Professor. That's sort of a visiting professor from industry. They said, "You can teach our introductory Java course in the winter of 2009." So I did that and I had a wonderful time. I've also been participating as a judge in the ACM International Collegiate Programming Contest.

[44] See http://www.itl.nist.gov/fipspubs/fip127-2.htm .

This is an annual contest. Universities around the world send programming teams to regional contests at different parts of the world where they solve problems. The winning programming team from each region advances to the World Finals, which are held once a year. This year, in April of 2009, the World Finals were held in Stockholm. There are 100 teams that participate. I've been contributing problems to these ACM programming contests and serving as a judge at the World Finals. This spring was my 12th year in a row of doing that, and it's something that I enjoy very much.

What I would like to do at this point in my career is to try and pass along some of the enthusiasm that I have for computing as something that's fun to do, and a powerful influence on the world, to young people. I think it's alarming and difficult to understand why enrollments in computer science programs in the United States are declining, and why so many of the students in those programs are coming from outside the United States. When you look at what's happening in our world right now, the pace of technical change is increasing. It took us ten years to build System R, but look what's happening now. Google has come along and organized all of human knowledge and made it all accessible just by clicking on the web. ITunes has come along and revolutionized the music industry, so any song you've ever heard of can be downloaded with a click. These are revolutionary cultural changes. These are things that are having a profound impact on our society and our way of life. Everything in the world that is based on information being at a particular place or a particular time, so you have to go to that place and time to get the information, is obsolete now. Think about that! Universities, libraries, and publishers are all based on the idea that information is scarce and expensive, and you have to go to a store and buy it. But that's no longer true. Information is now ubiquitous and free, and we've only just begun to see the impact of that change in paradigm. So this is an exciting time to be involved, and computer science is where it's happening. SIGMOD [ACM Special Interest Group on Management of Data] and the data management industry is at the center of these spectacular changes, so how could you not be excited about that? So I think young people should understand this. They should understand that these important ideas are having a big impact on our world, and we'd like that impact to move the world in a good direction. People have pretty strong opinions about data privacy, and about the way that information should be managed, used, and made accessible, and this is something that I think young people should be more interested in than they are.

In addition to the importance of computer technology, I think young people should feel that playing with computers is fun. When I was in college at Harvey Mudd, when I should have been working on my calculus homework, I spent a month or so writing a program to play tic-tac-toe. This was in the era of punch cards and the 1620 computer which had, I think, 20K of RAM [random-access memory] or something like that. You programmed it in FORTRAN. So I tried to take this, what at the time I thought was a pretty sophisticated system, but in retrospect it looks kind of primitive, and tried to teach it to play games. I had a wonderful time doing that. I have a nice time thinking up these little problems for the ACM Programming Contest, or thinking up homework assignments for my Java class at Santa Cruz. I think that it's fun to do that kind of work and I'd like to communicate some of that enthusiasm to the next generation.

McJones: I think that's one of the roles of the museum that we're sitting in here, the Computer History Museum, to preserve the stories about the past so we can find out how we got here, and also to try to pass that excitement on. I'd love to hear some of your views of the museum. You've been coming in here for a couple years.

Chamberlin: I have been coming to events here. I think that this museum is a fabulous resource. I think we're preserving artifacts here at the museum that our society can't afford to lose. We need to remember where we came from and how we got here. I put together a little slide show called “50 Years of Data”, and a lot of the slides in this slide show are based on exhibits here at the museum. It starts with Herman Hollerith's punch card tabulator. In the late 1800s, Herman Hollerith was an MIT professor, and he made a bid to the government to process the U.S. Census of 1890 using his breakthrough technology of cards with holes in them, and little fingers that went through the holes into pools of mercury to advance counters. So if you wanted to know how many farmers there were in New York, you could run the punch cards through this machine and it would count them for you. Well this was a startup company in 1890. Herman Hollerith patented this punch card idea. He made a company around it. That company merged together with some other companies that made butcher scales, and clocks and various other things like that and eventually changed its name to International Business Machines Corporation. So if you want to know where IBM came from, you can go to the Computer History Museum and see the machine that started it all. It was Herman Hollerith's punch card processor that's down on the ground floor of the Computer History Museum. And from there, you can go through the early digital machines up to the commercial machines of the 1960s when I was in college. They have a 1620 in there. They have some of the IBM mainframes, the 360's and 370's that the System R project was based on. It's a wonderful way to see how this technology has evolved and matured over the years. So I'm very honored and excited by being invited by the museum to join their fellow program. It's a humbling experience to be included in this set of individuals, and I want to thank the museum for giving me this opportunity.

END OF INTERVIEW

