Pontifícia Universidade Católica de São Paulo

PUC-SP

Programa de Estudos Pós-Graduados em
História da Ciência

Odécio Souza

Edgar Frank Codd e o Banco de Dados Relacional: Uma contribuição para a História da Computação.

Dissertação apresentada à
Banca Examinadora da
Pontifícia Universidade Católica de São Paulo,
como exigência parcial para obtenção do título de
MESTRE em História da Ciência,
sob orientação do Prof. Dr. José Luiz Goldfarb

São Paulo
2014

Banca Examinadora

_____

_____

_____

Agradecimento

A sinceridade me obrigaria a criar aqui uma lista infindável, desde meus ancestrais, indígenas brasileiros, portugueses, italianos e austríacos; colegas, críticos e, sobretudo, meus mestres. Para ser menos exaustivo e destacar aqueles diretamente responsáveis pela presente conquista, agradeço aos colegas, corpo administrativo e professores da História da Ciência. Quatro pessoas, entretanto, agiram de forma direta e fundamental para qualidade deste trabalho: *Alessandro Menegat*, companheiro de trincheira nesta batalha de aprendiz de historiador da ciência, assim como os Professores Doutores que participaram da minha banca de qualificação, lançando luzes imprescindíveis sobre o texto de então, *Eli Banks*, *Thomás Haddad* e em especial (imensa paciência, dedicação e – a qualidade das qualidades – boa vontade) meu orientador, *José Luiz Goldfarb*.

Dedico

este trabalho à operária Maria Assunção Rissutti – minha queridíssima vó Gija - e aos seus descendentes e agregados, que me ensinaram, especialmente pelo exemplo de suas vidas, que o trabalho levado a sério, a honra, a dignidade e o respeito aos direitos do próximo, são os mais altos valores do ser humano.

Quanto mais eu vivo,
mais eu aprendo.
Quanto mais eu aprendo,
mais eu percebo quanto menos sei.
Cada passo que eu dou,
cada página que viro,
cada milha que ando, significa apenas
o quanto mais eu tenho que andar.
O que há de errado em querer mais?
Se você pode voar,
voe ainda mais para cima!
Com tudo o que há, porque contentar-se
com apenas um pedaço do céu?

Trecho de "A piece of sky" do filme "Yentel" (1983)
Composição: Alan Bergman, Marilyn Bergman e Michel Legrand
Intérprete: Barbra Streisand

Autor: Odécio José Fernandes de Souza Junior.

Título: Edgar Frank Codd e o Banco de Dados Relacional: Uma contribuição para a História da Computação.

Resumo: No presente ensaio, pretendemos analisar a participação de Edgar Frank "Ted" Codd na organização das bases teóricas que permitiram a criação do "Objeto Tecnológico" conhecido como "Sistema Gerenciador de Banco de Dados Relacional", oferecendo assim uma contribuição para a História da Computação.

Palavras Chave: História da Ciência; História da Computação; Informática; Computadores; Banco de Dados; Base de Dados; Relacional; Codd.

Author: Odécio José Fernandes de Souza Junior.

Title: Edgar Frank Codd and the Relational Database: A contribution to the History of Computing.

Abstract: In this essay, we intend to analyze the participation of Edgar Frank "Ted" Codd in the organization of the theoretical foundations that allowed the creation of the "Technological Object" known as "Relational Data Base Management System", thus offering a contribution to the History of Computing.

Key Words: History of Science; History of Computing; Computers; Data Base; Database; Data Bank; Relational; Codd.

Sumário

1.Introdução

Segundo dados da União Internacional de Telecomunicações, apresentados na edição de 05 de maio de 2014 d'O Estado de São Paulo, o número de celulares chegará até o fim deste ano de 2014 a 7 bilhões, perto do número de habitantes no planeta. Sob a ponta dos dedos de (em média) cada ser humano haverá então uma *máquina que processa dados*. Mesmo se forem desconsideradas todas as outras *máquinas que processam dados* no âmbito comercial, técnico ou pessoal (como a injeção eletrônica ou o dispositivo GPS incorporados aos veículos automotores, a televisão digital, ou o painel de controle do aparelho de micro-ondas bastante comum nas cozinhas), podemos observar a significância desse tipo de tecnologia atualmente.

Múltiplos componentes se agregam para permitir a existência de tal tecnologia, desde os físicos (como baterias e manipuladores de imagens: mostradores, captadores, lentes) aos lógicos (dados e algoritmos que os manipulam), os quais compõem ou são compostos por elementos advindos de ciências como a Física, da Informação, Econômica e Sociológica, dentre tantos outros saberes.

Escolhemos trabalhar uma gota desse oceano denominada *dado*. Para sermos precisos, do bojo da ciência que participa diretamente da construção da *máquina que processa dados* (genericamente designada *computador*, pelo menos desde a segunda metade do século XX, por volta de 1945, segundo explica Ceruzzi[1]), ciência esta conhecida comumente como

[1] Ceruzzi, *History of Modern Computing*, 1 e 6: no âmbito deste ensaio, o termo *Computador* é utilizado como uma máquina, equipamento ou sistema, construído com base na assim designada "arquitetura de John von Neumann". Na seção denominada *SGBDRs* voltaremos a esta citação para oferecer ao leitor um quadro integrado destas questões.

Computação[2], uma parte dessa gota, um componente particular destinado a guardar de uma determinada forma e permitir certas particularidades de recuperação de *dados*, denominado *Sistema Gerenciador de Banco de Dados Relacional*, doravante *SGBDR*.

Dentre os múltiplos elementos que compõem um *SGBDR*, procuraremos analisar a participação de Edgar Frank "Ted" Codd, na organização das bases teóricas que permitiram a criação desse *objeto tecnológico*[3], a partir de 1969.

A obra conhecida como inicial de Codd sobre *SGBDRs* intitulada "A Relational Model of Data for Large Shared Data Banks"[4] constituirá o principal documento a ser analisado e, considerando que Codd trabalhava nessa época para a *IBM*, faremos diversas referências a essa instituição, assim como à sua concorrente direta no desenvolvimento e comercialização de *bancos de dados*, a *Oracle*[5].

Esta última empresa citada, assim como a designação comercial do *objeto tecnológico* do qual detém os direitos comerciais – *Oracle* – pode ser utilizada para entendermos pelo menos uma das motivações de Codd (ver

---

[2] Muitos outros termos, como "Informática" e "Processamento de Dados", diversos de seus ramos como "Engenharia ou Ciência da Computação", do mesmo modo que a os papéis dos profissionais quanto ao seu relacionamento com os computadores, como "Analista de Sistemas", "DBA" [Database Administrador, Administrador de Bancos de Dados em português], "Programador de Computador", "Matemático de Programação" ou simplesmente "Programador" aparecem neste ensaio como se fossem sinônimos. Explicar a diferenciação ou coincidência deles não é uma possibilidade aqui.

[3] Em Lawson, "Technology, Technological Determinism", pode-se observar "objetos técnicos [...] como a condição e consequência da atividade técnica." Muito embora Lawson esteja discutindo questões mais amplas, sua utilização desta ideia será tomada no contexto do presente ensaio, para que entenda-se tal "objeto técnico ou tecnológico" como um conjunto de algoritmos computacionais destinado a prover uma certa funcionalidade, a qual pode ser entendida como um sistema de informação qualquer, na sua interface com o usuário final, ou destinado a completar uma tarefa em lotes, como a atividade de fechamento de uma folha de pagamentos, ou um *SGBDR*.

[4] Codd, "A Relational Model", 65. Nossa análise aponta pelo menos um trabalho anterior sobre SGBDRs, além de trabalhos relativos a outros assuntos.

[5] Neste ponto do ensaio, utilizaremos as expressões "banco de dados", "base de dados" e "depósitos de dados" como se fossem sinônimos e do modo mais simples possível: Imagine-se um catálogo telefônico impresso em papel, ou um arquivo metálico contendo fichas de cartolina que permitam ao usuário de uma biblioteca a localização de um item de seu acervo, ainda que neste texto se faça referência a uma forma eletrônica de tal ideia. Na sua obra *Introdução a Sistemas de Bancos de Dados*, C. J. Date, 3, oferece uma imagem semelhante. *IBM* se refere à International Business Machines Corporation [ver www.ibm.com] e *Oracle* à Oracle Corporation [ver www.oracle.com] cuja importância da participação se demonstra pelo gráfico exposto como Anexo 1.

as seções designadas, *Codd* e *Antecedentes e Motivações* adiante): por exemplo, quem tem uma conta em uma Instituição Financeira, normalmente conhecida como *Banco Comercial*, está usualmente limitado a somente ter acesso ao seu próprio dinheiro lá depositado se souber informar os números de sua agência e conta corrente. Tal restrição é imposta pelo *depósito de dados* utilizado, ou seja, o *sistema de informação,* através de uma determinada *linguagem de interface com usuários*[6], permite somente esta pergunta. Se por um lado, um *oráculo grego* era uma entidade capaz de responder a qualquer questionamento, por outro o *oráculo da tecnologia* permite a criação de *depósitos de dados* capazes de fornecer respostas a perguntas que contemplem quaisquer das suas características.

Observemos questões importantes para a História da Ciência, citadas por Alfonso-Goldfarb et al., tais como a *biblioteca ideal*[7] ou *classificação*[8]*.* Um aspecto da *biblioteca ideal* é a possibilidade de constituir-se num *labirinto*[9], assim como para a questão da *classificação* é vital um conceito, um método sólido, para seu estabelecimento. A conexão deste trabalho com tal questão, a *classificação*, pode ser verificada também em outro artigo, de autoria das Professoras Alfonso-Goldfarb, Waisse e Ferraz, *From Shelves to Cyberspace*[10].

---

[6] Entenda-se por *Sistema de Informação* uma coleção de *objetos tecnológicos* utilizada pelos *computadores* de uma pessoa física ou instituição, com a finalidade de permitir a manipulação dos *depósitos de dados* que lhes interessa. Entenda-se *linguagem de interface com usuários*, como a expressão do "modelo de usabilidade, conteúdo da mensagem e atividade de produção de signos apoiada por um sistema semiótico", conforme, por exemplo, debatido por Leite em "Modelos e Formalismos para a Engenharia Semiótica de Interfaces de Usuário", 2.

[7] Alfonso-Goldfarb et al., "Uma 'viagem' entre documentos e fontes." v-viii.

[8] Alfonso-Goldfarb, "Centenário Simão Mathias." 5-9.

[9] O termo *labirinto* deverá incomodar ao leitor. Optamos por utiliza-lo especialmente para evidenciar o cenário complexo com que se está lidando, ao utilizar-se uma base de dados através de um computador: um caminho aparentemente óbvio e seguro que pode levar repentinamente a um beco sem saída.

[10] Alfonso-Goldfarb et al., "From Shelves to Cyberspace", 551-60.

Acreditamos ser importante assinalar que um *computador*, salvo se muito bem instruído, constitui-se como mera amálgama de silício, cobre e plástico, assim como os caracteres que congrega não deixam de ser agrupamentos desordenados de dados. Elementos do *labirinto*.

Assim como o pesquisador em um trabalho encetado com a devida qualidade, mergulhando apropriadamente em suas fontes sem risco de afogamento, citando ainda Alfonso-Goldfarb et al.[11], para transformar dados em um conjunto aproveitável, há que se constituir apropriadamente uma *base de dados*. O *computador* acumula aparentemente de forma ideal os *dados*, mas isso pode esconder inconsistências como um livro sem autor ou sem assunto, ou um assunto erroneamente classificado, para citar algumas poucas dentre tantas possíveis dificuldades aí escondidas. *Labirinto*. Problemas decorrentes de equívocos de *classificação*.

É deste tipo de contexto que emerge na *indústria da tecnologia da informação*[12] a necessidade da criação de um *objeto tecnológico* capaz de conter dados de uma forma íntegra, através de um método que alguém, mesmo não sendo especialista em *informática*, possa entender e principalmente do qual possa obter respostas consistentes. Tal *objeto tecnológico* é designado *SGBDR*.

Pretendemos oferecer uma visão do papel da *IBM* e de Codd na tentativa de resolução desse problema. Devemos evidenciar que o presente trabalho deverá ser capaz tão somente de estabelecer a transição entre *Teoria Axiomática dos Conjuntos* e a interpretação e utilização desta por

[11] Alfonso-Goldfarb et al., "Uma 'viagem' entre documentos e fontes.", v-viii.

[12] Por *indústria da tecnologia da informação* desejamos referenciar as instituições ou pessoas físicas que participam ou participaram da pesquisa, desenvolvimento, produção, comercialização ou utilização de produtos e conceitos destinados direta ou indiretamente à criação ou uso de *Computadores*: equipamentos construídos com base na assim designada arquitetura de John von Neumann, constituída basicamente de dispositivos de entrada, processamento e saída de dados, conforme Ceruzzi, *History of Modern Computing*, 6.

computadores ao manipular um *SGBDR*. O problema como um todo, se insere num contexto muito maior, do mesmo modo que o entendimento do desenvolvimento da *indústria da tecnologia da informação* e da participação da *IBM* nessa indústria, história esta que já percorre pouco mais de um século, é um objeto muito mais vasto[13].

Para não deixarmos a ideia totalmente no ar, observemos como Latour exemplifica uma empresa de sucesso, em um tratado oriundo da Sociologia da Ciência: "a empresa pode quebrar, [ou] transformar-se na IBM do século XXI."[14]

Segundo nossa leitura do texto de Alfonso-Goldfarb, "qualquer historiador da ciência considera que a interligação (ou interdependência) das três esferas de análise [...] é sempre recomendável para um bom trabalho"; a saber, a esfera que considera "implicações epistemológicas e filológicas"; a esfera que abrange questões que flutuam entre o internalismo e o externalismo, que a autora aponta como "um influxo de ideias sobre ciência e sociedade"; assim como a "esfera de análises para verificação do contexto histórico."[15] Considerando tal norte, pretendemos oferecer um recorte do contexto histórico e social no qual tal desenvolvimento aconteceu, que deverá apontar para a influência da assim denominada *Guerra Fria*, especialmente no cenário norte-americano. Na seção denominada *Um modelo relacional*, através de tal exercício epistemológico (no sentido de analisar uma obra), indicaremos a participação de Edgar Frank "Ted" Codd na organização dos fundamentos teóricos que permitiram a criação do *objeto tecnológico* conhecido como *SGBDR*, apontando uma trajetória que,

[13] Obras como Ceruzzi, *A History of Modern Computing* e *Computing: a concise history*, além de IBM, *Tornando o Mundo Melhor* ou Gates, *A Estrada do Futuro*, apresentam visões desses aspectos.
[14] Latour, *Ciência em ação*, 52.
[15] Alfonso-Goldfarb, "Centenário Simão Mathias."

passando por Childs[16], se inicia na interpretação da *Teoria Axiomática dos Conjuntos* apresentada por Suppes, designada também como *sistema Zermelo-Fraenkel*[17]. Na seção denominada *SGBDRs*, estabeleceremos a importância, de tal *objeto tecnológico*, qual seja, onde e quando ele se apresentou e apresenta, seja na vida do leitor, destes autores, de cada ser humano e instituição.

Esse desenvolvimento teórico aconteceu a partir de 1969, sendo que em 1980 produtos como SQL-DS e DB2 (da IBM) e Oracle já eram plenamente comercializados[18].

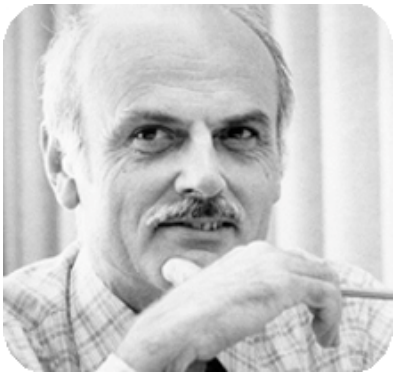
Podemos observar, envolvidas nesse contexto, entidades como as empresas que se apropriaram dessa tecnologia e a comercializam até hoje (2014), assim como o governo norte-americano, principalmente através de suas agências militares e aeroespacial. Melhores esclarecimentos desse cenário, como a análise do papel de instituições educacionais norte-americanas, assim como sobre a biografia de Codd serão necessários. Queremos crer não ser possível, neste ensaio, analisar mais profundamente o cenário socioeconômico norte-americano ou como e em que contexto, no resto do mundo, tecnologias convergentes ou similares se desenvolveram.

---

[16] Childs, D. L. "Feasibility of a set-theoretic data structure", 1-46 e "Description of a Set-Theoretic Data Structure", 557-64.
[17] Suppes, *Axiomatic Set Theory*. 2, 12.
[18] Sumathi, *Fundamentals of Relational Database Management Systems*. 5.

2.Codd

Edgard Frank "Ted" Codd nasceu na Inglaterra, na *Isle of Portland*, ligada por um istmo a Dorset, ao sul da Inglaterra, em 19 de agosto de 1923,



sétimo filho de um artesão coureiro e uma professora; incorporado à Real Força Aérea Inglesa, lutou na Segunda Grande Guerra Mundial e na Inglaterra concluiu sua graduação. Foi, entretanto, nos Estados Unidos da América pós-guerra, especialmente sob a égide da *Guerra Fria* que exerceu sua principal atividade intelectual e profissional, a maior parte do tempo como empregado da *IBM*[19].

Por este motivo, antes de oferecermos maiores detalhes sobre sua trajetória profissional, procuraremos oferecer uma visão da sociedade norte-americana dessa época, mais especificamente sobre o efeito da *Guerra Fria* no desenvolvimento tecnológico, particularmente na *Indústria da Informática*, também muito comumente referenciada como *Computação*.

Terminada a Segunda Grande Guerra Mundial, estabelece-se uma polarização de forças econômicas e bélicas que ficou conhecida como *Guerra Fria*, a qual perdurou até a queda do *Muro de Berlin* em 1989, ou até a extinção da União Soviética em 1991, dependendo do historiador consultado[20]. Tal cenário manteve de um lado os Estados Unidos da América e sob seu controle econômico a Europa Ocidental (vide o assim chamado

[19] ACM A.M.Turing Award. Inclusive sua fotografia. ACM é o acrônimo para "Association for Computing Machinery" (em português, Associação para Instrumentos de Computação), uma "sociedade informática educativa e científica [...] oferece recursos que promovem a computação como ciência e profissão." http://www.acm.org (acessado em 09 de novembro de 2014).
[20] McMahon, *Guerra Fria*, 187-90.

Plano Marshall[21]) e de outro a União das Repúblicas Socialistas Soviéticas

que controlava alguns países Europeus situados mais ao leste. A linha divisória, que inclusive estabeleceu a cisão da Alemanha e se concretizava em Berlim pela existência do *Muro*, delimitava a assim chamada *Cortina de Ferro*, ao leste.

Esta visão sucinta do fenômeno *Guerra Fria* foi estabelecida a partir da obra assim intitulada, de autoria de McMahon, assim como do trabalho de Knight, intitulado *Como começou a Guerra Fria*, a qual, para esta última autora, teria se estabelecido a partir da deserção de um criptógrafo russo[22]. McMahon aponta tal início a partir do final do conflito mundial contra o Eixo, demonstrando o ambiente de devastação provocado pela Segunda Guerra Mundial, terreno fértil para a imposição, ou tentativa de imposição, de novos condutores para o "sistema internacional eurocêntrico que havia dominado as relações mundiais nos 500 anos anteriores". Seja como for, ambos os eventos indicados datam de 1945.

Países sediados na Ásia, África, Europa, Oriente Médio e Américas, constituíram espaços disputados mais ou menos claramente por ambas as

[21] McMahon, *Guerra Fria*, 40.
[22] Knight, *Como começou a Guerra Fria*, 20. Ilustração obtida em www.historiadigital.org (acessado em 15 de outubro de 2014).

facções, através de atos de espionagem, fornecendo suporte financeiro, enviando recursos bélicos, participando efetivamente com tropas[23].

Para viabilizar e manter este jogo de forças, constantemente instável em certos territórios, foram estabelecidos tratados militares, como a OTAN (Organização do Tratado do Atlântico Norte, que congregou os Estados Unidos da América e os países da Europa Ocidental, ao qual se opôs o Pacto de Varsóvia, estabelecido entre a União Soviética e seus aliados[24]) e outros menos citados, como o Tratado Interamericano de Assistência Recíproca[25], que quase levou o Brasil a envolver-se no conflito militar entre a Argentina e a Inglaterra, conhecido como a *Guerra das Malvinas*, ou das *Falklands*, dependendo da nacionalidade do analista.

Esses tratados militares ora justificaram ora viabilizaram mesmo a predominância desse grupo social em alguns países. Na América Latina foram notórios os *Regimes de Exceção de Direita* que conhecemos no Brasil como *Revolução* (ou *Golpe*, novamente dependendo na tendência política do analista) *de 1964*.

Muito embora nos Estados Unidos da América o estado de direito não tenha sido alterado formalmente, ou seja, ainda que os militares não tenham tomado o poder explicitamente naquele país, obtiveram meios financeiros muito significativos, justificados pela necessidade de defender os interesses norte-americanos pelo mundo afora, assim como o próprio território. Para ilustrar tal assertiva, observemos que enquanto "o Plano Marchall [... forneceu] US$ 13 bilhões em assistência para a Europa Ocidental", "Carter autorizou um grande aumento nos gastos de defesa dos

[23] McMahon, *Guerra Fria*, 11.
[24] Ibid., 44 e 74.
[25] Conhecido como Tiar. http://acervo.estadao.com.br/procura/#!/Guerra+das+Malvinas+x+brasil/Acervo/acervo (acessado em 14 de outubro de 2014).

Estados Unidos: solicitou US$ 1,2 trilhão em gastos relacionados ao setor militar ao longo dos cinco anos seguintes [1977-81]."[26]

A História da Computação encontra-se severamente impregnada de descobertas, abandonos, avanços e recuos determinados por interesses armamentistas, ora financiando diretamente pesquisas em universidades, ora encomendando *objetos tecnológicos* e suas melhorias às empresas[27]. O estabelecimento da ARPANET, elaboração tecnológica que propiciou o desenvolvimento da rede mundial de computadores a partir dos anos 1990, então conhecida como INTERNET[28]; a miniaturização e aprimoramento de componentes necessários aos diversos projetos de mísseis e sistemas de defesa antimísseis, ou aos componentes das naves que levariam o homem à lua; transformações estas que modificaram a válvula a vácuo (que em sua gênese chegou a ter a altura de um homem), permitindo sua adaptação a partir dos transistores, estes últimos reduzidos e aglomerados aos milhares em componentes multitarefa designados chips e baseados em uma patente de 1948[29], são exemplos bastante típicos.

Por sua vez a *IBM* aparece como uma instituição bastante atuante quer por iniciativa própria, oferecendo ao mercado produtos que resolvera desenvolver, quer atendendo as demandas desse mercado.

As Universidades também tiveram papel semelhante, operando em ambos os sentidos, ao desenvolver projetos oriundos de pesquisas autônomas ou ao receber subsídios para projetos ligados diretamente a interesses militares. Uma das principais fontes indicadas por Codd, por

---

[26] McMahon, 41 e 164.
[27] Ibid., 138. Obras como Ceruzzi, *A History of Modern Computing* e *Computing: a concise history*, além de IBM, *Tornando o Mundo Melhor* ou Gates, *A Estrada do Futuro*, apresentam visões desses aspectos.
[28] Ceruzzi, *History of Modern Computing*, 284.
[29] Ibid.

exemplo, o ensaio de Childs, foi executado como um relatório técnico inserido em um projeto de pesquisa em uso de computadores em forma conversacional, nas dependências da Universidade de Michigan, patrocinado pela Agência de Projetos de Pesquisa Avançada do Departamento de Defesa norte-americano[30]. Uma sucessão de eventos um tanto mais complexa aconteceu em relação ao sistema operacional UNIX: desenvolvido a partir de pesquisas que aconteceram nas instalações da Bell Telephone Laboratories[31] e após ser apropriado e adaptado por diversos fabricantes de *computadores*, tornou-se fartamente utilizado pelos departamentos de ciência da computação das universidades, particularmente na Universidade da Califórnia em Berkeley[32], cujos trabalhos a partir de 1974 levaram à sua versão melhor conceituada. O *LINUX*, outra de suas versões, é um dos principais baluartes do movimento conhecido como *software livre*[33].

Os eventos motores, entretanto, não partiram sempre de necessidades bélicas: a partir de outra necessidade governamental americana, neste caso contar a população do senso de 1880, Herman Hollerith reuniu princípios encontrados em máquinas como o tear de Joseph Marie Jacquard e a *Difference Engine* de Charles Babbage e em 8 de janeiro de 1889 anunciou a criação da *máquina de tabulação*, a qual constituiu um passo importante em direção a computação automática[34]. Não fora por outra evidência de sua relevância, observemos que milhões de trabalhadores todos os meses

---

[30] Childs, "Feasibility of a set-theoretic data structure".
[31] Ceruzzi, *History of Modern Computing*, 106.
[32] Ibid., 283.
[33] *Versão* é um termo que acabou evoluindo para *distribuição* – ver http://www.vivaolinux.com.br/artigo/Historia-do-GNU-Linux-1965-assim-tudo-comecou/, http://br-linux.org/ e http://softwarelivre.org/ (acessados em 11 de novembro de 2014).
[34] IBM, *Tornando o Mundo Melhor*, 23-6. http://operamundi.uol.com.br/conteudo/noticias/19016/hoje+na+historia+1889+-+herman+hollerith+inventa+a+maquina+eletrica+de+contagem.shtml (acessado em 14 de outubro de 2014). Para uma visão mais detalhada sobre Jacquard e Babbage, vide Banks, "O invento de Jacquard" e "Charles Babbage".

recebem um demonstrativo de pagamento que se convencionou chamar de *hollerith*. Para fornecer tais equipamentos, fundou uma empresa que:

> em 1911, o financista Charles Flint [... uniria, a outra duas]; uma que fazia balanças calculadoras (que calculavam automaticamente o preço total de um item vendido por peso); outra que fazia relógios de ponto que os funcionários usavam nas fábricas. Inicialmente, a empresa resultante foi chamada de Computing-Tabulating-Recording-Company, ou C-T-R, à qual se associou, em 1914, Thomas Watson. Em 1924, ele mudou o nome para International Business Machines [*IBM*].[35]

"Aperfeiçoar a máquina de Hollerith e inventar outros dispositivos" [36] foi a estratégia perseguida por Watson, determinando uma trajetória empresarial que teve como parte importante do seu início a instalação de "um departamento de desenvolvimento de produtos em um edifício de arenito em frente à velha Penn Station, na cidade de Nova York"[37], onde Codd trabalharia mais tarde. Tal estratégia propiciou o fato da *IBM* ter se notabilizado pelo volume de patentes registradas, de produtos desenvolvidos exclusivamente em seus laboratórios, em cooperação com universidades, ou ainda sob contratos do governo norte-americano. Muito embora o conceito de sociedade Pós-Industrial[38] tenha sido documentado e mais intensamente utilizado na última década do século XX, estava influenciando, portanto as atividades da *Big Blue*, inserida em um cenário macroeconômico que se caracterizaria pelo fato da sociedade estar "descartando rapidamente as indústrias poluidoras e adotando tecnologias digitais como fonte de crescimento econômico e vantagem competitiva."[39]

> Nós também começávamos a entender que o conhecimento promove muito mais do que inovações na produção. Na verdade, numerosos estudos mostraram que não existe uma correlação simples entre desempenho

---

[35] IBM, *Tornando o Mundo Melhor*, 19.
[36] Ibid., 174.
[37] Ibid.
[38] Em IBM, *Tornando o Mundo Melhor*, 170-3, cita-se o trabalho de 1973 chamado The Coming of the Post-Industrial Society (A Chegada da Sociedade Pós-Industrial), de Daniel Bell, sociólogo da Universidade de Harvard.
[39] Ibid., 170-3. *Big Blue* foi e é constantemente utilizado como apelido da *IBM*.

> financeiro e o nível de gastos em P&D [Pesquisa & Desenvolvimento] em uma empresa. Mais exatamente, o que importa é a combinação de conhecimento, talento, ferramentas e forma de trabalho que as empresas utilizam. [...] Não importa como você a chame — sociedade pós-industrial, organização do conhecimento ou qualquer outro termo —, a mudança fundamental é clara. No entanto, por trás dessa megatendência, a questão continua sendo como as organizações realmente capturam o valor que criam. A história da IBM revela um padrão amplo de como a informação e o conhecimento se transformaram em dinheiro durante o último século — e esse padrão sugere uma trajetória que governará a evolução contínua da empresa moderna.[40]

Codd alistou-se na RAF, apesar de sua condição de estudante universitário permitir uma dispensa e em "1953 deixou os Estados Unidos (e a IBM), em protesto contra a caça às bruxas do senador Joseph McCarthy e mudou-se para o Canadá". Podemos justificar o primeiro destaque como simples entusiasmo patriótico, e o segundo pode ter seu valor diminuído quando quatro anos depois "um encontro casual com seu antigo gerente na IBM provocou sua volta para os EUA [e à *IBM*]",[41] mas são traços da personalidade dessa pessoa, que havia se tornado cidadão norte-americano por acreditar que lá havia mais oportunidades de sucesso para um bolsista integral da Universidade de Oxford (Colégio Exeter), que havia estudado química entre 1941 e 1942 e após a guerra, matemática, obtendo sua graduação em 1948.

Essas oportunidades oferecidas pela economia norte-americana no pós-guerra foram assim evidenciadas por McMahon: "o produto interno bruto da nação dobrou entre 1941 e 1945 [...] Os salários reais elevaram-se rápida e dramaticamente [...] e os americanos do front doméstico viram-se inundados numa cornucópia de bens de consumo agora acessíveis ao seu poder de compra." Citando "o Diretor do Departamento de Mobilização e

[40] IBM, *Tornando o Mundo Melhor*, 170-3.
[41] ACM A.M.Turing Award.

Reconversão de Guerra, 'está na agradável situação de ter de aprender a viver de um modo 50% melhor do que jamais experimentara antes.'"[42]

"Em 1961, com uma bolsa da IBM, [... Cod ...] frequentou a Universidade de Michigan e obteve um Mestrado e um [Doutorado] Ph.D. em ciências da comunicação."[43]

Desde sua admissão na *IBM* em 1949, como *matemático de programação,*[44] lidou com questões ligadas diretamente à criação de *algoritmos computacionais,* trabalhando principalmente nos laboratórios de pesquisa da empresa, em Nova York, Poughkeepsie, Ann Arbour e San Jose até que voltou sua atenção para as questões do banco de dados.[45]

Notemos que durante o autoexílio canadense, "dirigiu [em Ottawa] o departamento de processamento de dados da 'Dispositivos de Computação do Canadá Limitada' (que esteve envolvida no desenvolvimento do programa de mísseis guiados canadense)."[46]

Somente entre 1981 e 1983 a *IBM* lançaria seus *SGBDRs*, respectivamente SQL/DS para o ambiente VSE e DB2 para o ambiente MVS. Tendo permitindo que "outros fornecedores, incluindo Relational Software Inc. (mais tarde renomeada Oracle Corporation) e Relational Technology Inc. (mais tarde renomeada Ingres Corporation)" se estabelecessem nesse mercado muito antes, "talvez por ter investido fortemente em seu banco de

---

[42] McMahon, *Guerra Fria*, 14.

[43] ACM A.M.Turing Award. Sua tese – que foi publicada pela Academic Press, em 1968, sob o título "Cellular Automata" - representou uma continuação e simplificação do trabalho de von Neumann na auto-reprodução de autômatos, em que Codd mostra que os 29 estados exigidos pelo esquema de von Neumann, poderiam ser reduzidos para apenas oito.

[44] Designação que se aproxima mais do conjunto de atribuições que por volta do ano 2000 equivalia a programador de computadores e atualmente (2014) a desenvolvedor de aplicativos.

[45] Ibid. Codd afirmou que o que inicialmente motivou nesta pesquisa foi uma apresentação por um representante de uma empresa de banco de dados, que parecia - incrivelmente, tanto quanto Codd podia entender – não ter nenhum conhecimento ou compreensão da lógica de predicados. Vários produtos de banco de dados, de fato, existiam naquela época; no entanto, eles eram, sem exceção, ad hoc, pesados e difíceis de usar - eles poderiam realmente ser usados apenas por pessoas com competências técnicas altamente especializadas - e repousavam [os produtos de bancos de dados] sobre nenhuma base teórica sólida.

[46] Ibid.

dados não relacional - IMS - e estar ansiosa para preservar a receita com esse produto.”[47]

Codd recebeu o *Prêmio Turing* da ACM em 1981, tendo sido selecionado por seu Comitê Geral de Realização Técnica em função de sua "fundamental e continuada contribuição para a teoria e prática de sistemas de gerenciamento de banco de dados”.[48] Demitiu-se da *IBM* em 1984, mas manteve-se ativo nesse campo, lidando através de empreendimentos próprios com o produto cuja concepção marcara tanto. Faleceu em 18 de abril de 2003.

Acreditamos ser relevante observar estes trechos:

> ... foi o homem que, isoladamente, colocou o campo do gerenciamento de bancos de dados sobre uma base científica sólida. Toda a indústria de banco de dados relacional, faturando muitos bilhões de dólares por ano, deve o fato de sua existência ao trabalho original de Ted, e o mesmo é verdade para todo o enorme número de programas de ensino e pesquisa em banco de dados relacionais em curso em todo o mundo [...] Na verdade, todos nós que trabalhamos nesta área devemos nossa carreira e meios de subsistência para o gigante Ted cujas contribuições foram feitas a partir do final da década de 1960 até o início de 1980. Todos nós temos para com ele uma enorme dívida. Esta homenagem a Ted e suas realizações é oferecida em reconhecimento dessa dívida.[49]

Observemos como a *IBM* o descreve:

> No final dos anos 1960, E. F. “Ted” Codd, pesquisador da IBM, observou o modo de os dados serem classificados e manuseados nos computadores e pensou em um modo melhor [...] ele tinha formação de matemático e trabalhou como *programador* nos primeiros *computadores*. Codd procurou uma definição matematicamente rigorosa de um banco de dados. A meta era uma descrição geral de como armazenar, atualizar e extrair dados, de modo que a resposta às consultas fosse precisa e que quaisquer alterações nos dados produzissem resultados consistentes.
>
> O trabalho de Codd deu margem a toda uma geração de empresas de bancos de dados, inclusive a Oracle e a Sybase.[50]

---

[47] ACM A.M.Turing Award.
[48] E.F.Codd. The 1981 ACM Turing Award Lecture.
[49] Date. "Edgar F. Codd … a tribute", 4.
[50] IBM. *Tornando o Mundo Melhor*, 77.

Em outra das fontes, a *Oracle* aponta:

> Dr. E. F. Codd publicou o "paper" "um modelo relacional de dados para grandes bancos de dados compartilhados", em junho de 1970 na revista "Comunicações da ACM". O modelo de Codd é agora aceito como o modelo definitivo para sistemas de gerenciamento de banco de dados relacionais (*SGBDR*s). A "Linguagem Estruturada de Pesquisa" [em inglês Structured Query Language (SEQUEL)] foi desenvolvida pela IBM Corporation, Inc., para usar o modelo de Codd [... e] tornou-se mais tarde SQL (ainda pronunciado como "sequel"). Em 1979, a Relational Software, Inc. (agora Oracle) introduziu a primeira implementação comercialmente disponível de SQL. Hoje, SQL é aceita como a linguagem padrão [para manipulação dos] *SGBDRs*.[51]

A perigosa sombra (ou *vírus*, como nos ensina Canguilhem[52]) de precursor paira sobre estas citações (apresentamos diversos outros trechos de conteúdo similar como *Anexo 3 - Outros documentos*). Nossa análise e pesquisa, entretanto não nos permite afirmar que tais assertivas estejam integralmente equivocadas. O contexto em que se insere nosso ensaio, por outro lado, nos leva a destacar tal questão, buscando estabelecer uma crítica, um questionamento, a respeito de tal saber científico. Afinal, segundo Bachelard, "se não há pergunta, não pode haver conhecimento científico [... e] um obstáculo epistemológico se incrusta no conhecimento não questionado."[53]

[51] Oracle. Database SQL Language Reference.
[52] Canguilhem, "Objeto da História da Ciência", 5.
[53] Bachelard, *A formação do espírito científico*, 18-9.

3.Um modelo relacional

O título desta seção 3 é retirado da nossa tradução do título do ensaio dito seminal de autoria de Codd, cuja íntegra e tradução constam deste trabalho como Anexo 2. Nela pretendemos realizar uma análise epistemológica deste nosso eixo principal. Optamos por manter exclusivamente nesse Anexo 2, juntamente como texto traduzido, comentários e esclarecimentos muito específicos, diretamente ligados a ele.

3.1.Antecedentes e Motivações

Em 1959 foi estabelecida uma organização com a finalidade de guiar o desenvolvimento de padrões para linguagens de programação que pudessem ser utilizadas em computadores. Tal organização foi designada *CODASYL* (Conference on Data Systems Languages, em português, Conferência em Linguagens de Sistemas de Dados), com membros voluntários oriundos da indústria e do governo, envolvidos na atividade de processamento de dados[54]. Membro da *CODASYL* desde 1960, por exemplo, Jean E. Sammet foi responsável pelo estabelecimento de conferências da *ACM* denominadas *History of Programming Language* (*HOPL*) (em português, Conferências sobre História das Linguagens de Programação)[55].

Originaram-se da *CODASYL* propostas amplas, como um modelo hierárquico para bancos de dados, designado *IDMS* (Integrated Database Management System, em português, Sistema Integrado de Gerenciamento de Banco de Dados), ou as diretrizes para a "COmmon Business Oriented Language" (em português, Linguagem Orientada a Negócios Regulares - ou Comuns -, conhecida como linguagem *COBOL*), assim como discussões de

[54] Universidade de Minnesota.
[55] Bergin, "A History of the History of Programming Languages," 69-74.

detalhes como os conceitos de conjuntos e subconjuntos de objetos (designados *schemas* e *subschemas*, em inglês), ou o alcance do *DDL* (*Data Definition Language*, em português, Linguagem para Definição de Dados)[56].

Mashey descreve que em cada classe de computador, *mainframe*, *mini* e *micro computadores*, diversos níveis de linguagem foram sendo utilizadas, desde aquelas que mais se aproximaram do funcionamento da máquina, designadas como de baixo nível (por exemplo, *Assembly*, *Macro-assembler*, *Higher-level algorithmic languages* como *Fortran* e *C*), até aquelas que procuram aproximar-se da linguagem utilizada pelas pessoas, estas consideradas de alto nível (por exemplo, *APL*, *Mathematica*, e *Planilhas de Cálculo*)[57].

Processar dados em computadores significa, pelo menos para nossa abrangência de análise, utilizar um equipamento construído com base na assim designada arquitetura de John von Neumann, constituída basicamente de dispositivos de entrada, processamento e saída de dados, que Ceruzzi datou como existente desde meados da década de 1940[58]. Essa atividade estabelecia (e ainda estabelece) desafios como o da inexistência, ou existência precária, de padrões que instituições como a *CODASYL* tentaram determinar.

Podemos observar desde antes dessa época (meados de 1940) e ainda hoje, a falta de certos padrões e as dificuldades daí decorrentes. Para citarmos um exemplo, observemos adiante as dificuldades em estabelecer-se um *byte* de 6, 8 ou 9 *bits*. Se tal imagem não estiver clara ao leitor, sugerimos que este observe a quantidade de dispositivos que

---

[56] Anthes, "Happy Birthday, RDBMS !", 16-7.
[57] Mashey, "Languages, Levels, Libraries, and Longevity", 33-8.
[58] Ceruzzi, *History of Modern Computing*, 6.

provavelmente possui, alguns destes obsoletos, destinados a manter carregada a bateria de seus dispositivos móveis (ditos *celulares* ou *tablets*). Via de regra, eles se tornam obsoletos porque um novo dispositivo adquirido, até mesmo do mesmo fornecedor, utiliza conectores diferentes, tornando os antigos inutilizáveis.

Nesse período, entre 1945 e 1980, antes que os microcomputadores começassem a se tornar populares, prevalecem duas alternativas de trabalho: Processamento em Lotes, *Batch* (que veremos expresso no artigo de Codd como *noninferential* ou *nondeductive*), e Processamento Conversacional (*inferential*, *deductive*, para Codd). Mesmo hoje em dia (2014), tais alternativas existem e são largamente utilizadas, ainda que permeadas de diversas outras possibilidades.

Realizar um Processamento em Lote significava pré-agrupar volumes de dados em pilhas de cartões perfurados, diversos carretéis de fitas, ou extensas áreas de tambores ou discos, submetê-los a rotinas específicas de processamento que usualmente ao seu final geravam novos volumes de dados e grossas pilhas de papel impresso. Na década de 1960 praticamente não havia alternativa para esta rotina[59].

O *Trabalho em Terminais*, citado por Codd e descrito por Gates[60], aponta para uma situação mais próxima daquilo que é visto hoje como a utilização de um computador pessoal. Era possível utilizar um terminal para codificar um programa que se encarregaria de um processamento *Batch*, ou que executaria alguma pesquisa em uma base de dados, ou uma rotina de

[59] Ceruzzi, *History of Modern Computing*, 74. Vide ainda Fragomeni, *Dicionário Enciclopédico de Informática*, 502, assim como Hyman, *Computing*, 29.
[60] Gates, *A Estrada do Futuro*, 69.

calculo sendo, entretanto bastante rara nas empresas e instituições educacionais.

Segundo o próprio Codd:

> a motivação mais importante para o trabalho de investigação que resultou no modelo relacional foi o objetivo do fornecer uma fronteira nítida e clara entre a lógica e aspectos físicos de gestão de banco de dados (incluindo projeto de banco de dados, recuperação de dados e manipulação de dados).[61]

Codd classificará essa motivação como *Objetivo Independência de Dados*, ou seja, a não vinculação entre o acesso ao dado e como ele está armazenado, questão que exploraremos em detalhe em seções posteriores. Outros objetivos, com gradações diferentes de importância, estão descritos pelo autor no documento citado, porém gostaríamos de destacar o desenvolvimento da *Linguagem SQL*, dotada de uma semântica, sintaxe e morfologia muito próximas da linguagem inglesa, dada inclusive sua permanência até o presente (2014) [62].

Os usuários de *Grandes Bancos de Dados*[63] beneficiaram-se efetivamente da faceta relacional, subsidiária confessa do trabalho de Codd, mas não encontramos evidência de uma demanda prévia, ou explícita, por esse trabalho, dada inclusive a resistência ao *Modelo Relacional*, por parte da *IBM*. Ao traçar sua visão d'*Uma História da Computação Moderna*, especialmente ao tentar definir *computador*, Ceruzzi deixa claro, entretanto que a "Computação nos Estados Unidos desenvolveu-se após 1945 em um clima de prosperidade e forte mercado de consumo. Foi também durante a *Guerra Fria* com a *União Soviética*."[64] Através de todo o seu trabalho, podemos observar o quanto esta indústria foi subsidiária das necessidades

[61] E.F.Codd. "The 1981 ACM Turing Award Lecture".
[62] Oracle. "Database SQL Language Reference". Uma linguagem de acesso a dados organizados de forma relacional.
[63] Observar a caracterização de *dimensão* que é realizada nas seções posteriores do presente trabalho.
[64] Ceruzzi, *A History of Modern Computing*, 7. Grifo nosso.

bélicas desse período. Pretendemos ter demonstrado tais questões na seção denominada *Codd*.

3.2.Conceitos introdutórios

Modelo Relacional é o centro, para não dizer o conteúdo todo do ensaio, onde Codd aponta de que modo conceitos oriundos de uma *teoria axiomática dos conjuntos matemáticos* podem ser utilizados para manipulação de dados. Aparecem no título questões não desenvolvidas durante o ensaio, como *Grandes* e *Compartilhados*.

Lançaremos mais uma vez mão do trabalho de Ceruzzi para ilustrar a questão da *dimensão* (o que o termo *Grandes* poderia significar): ao relatar a criação da livraria *on line* Amazon.com em 1995, "em outubro estava processando 100 pedidos por dia [...] em Julho de 2000, 100 pedidos por minuto era normal."[65] É óbvio que o exemplo é anacrônico por ter ocorrido décadas depois, mas nos serve para projetar o que poderia estar sendo indicado no título do ensaio de Codd: um grande acervo de dados se referiria aos dados necessários para processar a folha de pagamento de uma empresa como a *Ford Motor Company*, ou ao registro do *Seguro Social* de milhões de cidadãos americanos; não a lista dos cinquenta estados que se unem naquela nação. O anacronismo se justifica porque os autores, de um modo geral em informática e particularmente em relação aos *SGBDRs*, costumam isentar-se de quantificar esse tipo de questão. Portanto utilizamos uma pouco comum quantificação (onde se verifica inclusive outra questão significativa, qual seja, o crescimento de um *acervo* de *dados* manipulados diariamente em quase uma mil e quinhentas vezes em menos de cinco anos). Rever a quantificação apresentada por Hardgrave pode ser útil por nos remeter a um universo, um tamanho, na ordem de uma

[65] Ceruzzi, *A History of Modern Computing*, 326.

centena de *Giga Bytes*, ao qual se referia o autor no final da década de 1970[66].

Quanto à questão do compartilhamento, ainda que perpasse muito da história contemporânea da *indústria da tecnologia da informação*[67], sua análise profunda se encontra muito além das possibilidades do nosso ensaio, mas pode ser esclarecida nos seguintes termos: suponhamos que em uma Instituição de Ensino um conjunto de procedimentos que permita a reserva de um auditório, por exemplo. Tal procedimento se constituiria em uma folha de papel com data e hora de início e término de um dado evento, assim como o nome do professor responsável. Dois atendentes recebem simultaneamente ligações telefônicas de dois professores que desejam reservar o mesmo auditório para uma mesma data e hora. Ambos os atendentes podem ao mesmo tempo ler a referida folha de papel para informar aos professores sobre a disponibilidade do auditório, mas somente um deles poderia realizar a inscrição da reserva na linha de papel correspondente. Se esta imagem, esta dificuldade, não estiver clara, sugerimos ao leitor que tente escrever na mesma linha do mesmo papel em que outra pessoa está no mesmo momento escrevendo.

Em termos de *computadores*, a mesma dificuldade de escrita se impõe e mesmo a leitura compartilhada oferece dificuldades. No ensaio de Codd em análise tais questões não são exploradas, apesar da importância que

---

[66] Hardgrave, "A Technique for Implementing a Set Processor", 86. Mais adiante, exploraremos em detalhes, dentre outras, a conceituação de *Byte*. Neste ponto, um leitor que não tiver intimidade com esse conceito, pode pensar em um *Byte* como sendo igual a um *caractere* ou letra. Convencionou-se que *Kilo Bytes* (KBytes ou KB) corresponderiam a 1024 *bytes*, *Mega Bytes* (MBytes ou MB) a 1024 KBs, *Giga Bytes* (GBytes ou GB) a 1024 MBs e assim sucessivamente para Peta, Exa, Zetta, Yotta. Ousamos aqui indicar um tipo de fonte pouco convencional, um fórum incluso no Site da Microsoft: http://social.msdn.microsoft.com/forums/vstudio/pt-BR/127c4869-1b15-4a55-a5f9-2f687f151a81/ converter-bytes-para-megabytes (acessado em 06 de novembro de 2014).

[67] Queremos reiterar que tal história pode ser encontrada em obras como Ceruzzi, *A History of Modern Computing* e *Computing: a concise history,* além de IBM, *Tornando o Mundo Melhor* ou Gates, *A Estrada do Futuro*.

tentamos elucidar, muito embora compareçam no título (através do termo *Shared*, Compartilhados em português) e ainda que os *SGBDR*s tenham efetivamente oferecido soluções eficazes para tais questões. Mais uma possibilidade de desenvolvimentos futuros.

Dedicaremo-nos agora à questão da *coleção* ou *acervo* de *dados*. Em um primeiro momento, solicitamos ao leitor que entendesse as expressões *banco de dados* e *depósitos de dados* como se fossem sinônimos e do modo mais simples possível. A tais conceitos, foram se somando no texto de Codd e em nosso ensaio: *coleção*, *acervo*, *depósito*, *banco*, *base*.

Da já referida questão da *classificação* surge uma importante motivação para a análise desta questão. Retomemos um exemplo citado anteriormente, onde "o *banco de dados*, por si só, pode ser considerado como o equivalente eletrônico de um armário de arquivamento, ou seja, ele é um *repositório* ou *recipiente* para uma *coleção* de *arquivos* de *dados* computadorizados."[68]

Nesse ponto, dois novos termos, *repositório* e *recipiente*, se somam à nossa lista, além do fato de se aplicarem a *arquivos*, segundo Date. Podemos perceber que nossa lista (agora constante de *coleção*, *acervo*, *depósito*, *banco, base, repositório, arquivo* e *recipiente*) carece de atenção. Ou *classificação* se desejarmos transitar devidamente através deste emaranhado.

Codd não explica como utiliza estes conceitos, mas acreditamos ser necessário caracteriza-los, ou seja, analisar epistemologicamente como ele os utiliza. Será útil também explorar o significado de *bit*, *byte*, *caractere* (ou

[68] Date, *Introdução a Sistemas de Bancos de Dados*, 3. Grifos nossos.

character em inglês), além de *data* e *datum*, os quais também são utilizados por Codd ou por nós.

Iniciemos por *bit.* Segundo Hyman, "BInary digiT. Unidade básica de informação em um sistema de informação codificado de forma binária: ou um 1 [numeral um] ou [um] 0 [numeral zero]."[69] Fragomeni complementa essa ideia afirmando que

> [*bit* ...] a menor unidade de informação em um sistema binário, ou elemento tecnológico que assegura a conservação, o tratamento ou a visualização de informação em forma binária, podendo assumir um entre dois estados (0 ou 1, marca ou espaço, sim ou não, verdadeiro ou falso, ligado ou desligado, etc.), e podendo ocupar, em um computador digital, uma posição ou célula de memória. Em teoria da informação, unidade de quantidade de informação, ou seja, a quantidade de informação que resulta da escolha entre duas possibilidades mutuamente exclusivas e que têm, cada uma, uma probabilidade de realização igual a ½.[70]

Acreditamos ser significativo, para devidamente respeitar as esferas dos trabalhos em História da Ciência, citarmos uma provável origem tecnológica e não matemática do *bit*. Encontramos na obra de Banks um elo entre a procura para:

> [...] resolver problemas de caráter imediatista na tecelagem de tecidos com desenhos, a solução encontrada teve repercussões claras nas técnicas de armazenamento e processamento de informações. Para ele [Jacquard], tratava-se de guardar comandos de posicionamento das agulhas do tear; ficou, entretanto, como um legado, uma série de conceitos que nos remetem à computação moderna.[71]

Dente as repercussões citadas aí, a mais fundamental, no sentido de inicial, ou aquela da qual as outras dependem: uma agulha passando ou não pelo orifício de um cartão, já que este tem ou não um furo naquela posição, coincide justamente com a escolha entre duas possibilidades mutuamente exclusivas que citamos anteriormente.

---

[69] Hyman, *Computing*, s.v. "Bit".
[70] Fragomeni, *Dicionário Enciclopédico de Informática,* s.v. "Bite". Grifo nosso. O verbete indica "bite", mas essa forma é raramente utilizada, ainda que expresse a forma aportuguesada. "Bit" aparece na mesma página e a autora aponta outra versão do significado, ligada diretamente a Shanon, em Teoria da Informação.
[71] Banks, "A máquina que pensa", 71.

Decorrência de *bit, byte*:

> Um *caractere* [*character* em inglês] consistindo de inicialmente 8 *bits* de informação. O *byte* de 8 *bits* [ou *8-bit*] tornou-se o principal padrão de tamanho de *caractere* com a introdução do system/360. Anteriormente, o *caractere* de 6 *bits* havia sido o padrão. Entretanto, o termo "*caractere*" está se tornando o significado de *caractere 8-bit* ou *byte* conforme o padrão de *8-bit* vai sendo adotado de forma geral.
>
> A história da adoção do padrão de *caractere* de *8-bit* é instrutiva. Quando ele [, o *byte*,] foi introduzido alguns fabricantes declararam que iriam manter-se no padrão *6-bit*. Como resultado de fusões [de empresas] e outras razões, virtualmente todo fabricante agora [1976] fornece sistemas com *8-* ou mais *bit*. A maioria dos sistemas de *6-bit* têm sido descartados e usuários com sistemas *6-bit* têm tido que enfrentar problemas de conversão; existe uma grande penalidade em tentar desviar dos padrões em sistemas de processamento de dados.
>
> Ocasionalmente "*byte*" é sinônimo de *caractere*: nesse uso um *byte* pode ter 6 *bits*, ou mesmo somente um *bit*. A possibilidade de um *grande-bit* [*big-byte* em inglês] com 9 *bits* foi muito discutida em conexão com o projeto de séries futuras da IBM, o sistema que se espera largamente que substitua o system/370 em algum momento dos anos 1970.[72]

Há diversas informações interessantes neste trecho. Podemos perceber que à época que Codd estava desenvolvendo seu ensaio, sequer a quantidade de uns e zeros que formam cada unidade, que enxergamos como caracteres nos dispositivos que hoje utilizamos, estava estabelecida. Esse particular detalhe, mais uma vez, nos remeteria para longe do escopo deste ensaio, assim como, outro aspecto seu, a aparente tentativa da *IBM* de ditar padrões de desenvolvimento para a *indústria da tecnologia da informação*. Reiteramos que obras como a de Ceruzzi podem satisfazer pelo menos em boa parte ao leitor que desejar penetrar nessa discussão. Alguns aspectos particulares da questão da padronização do tamanho do *byte* e seu significado estão expressos em seu livro[73]. Como Codd era funcionário da *IBM*, por outro lado, não podemos nos furtar a essas conexões. Mas voltemos ao *byte* para lembrar que ele acabou sendo designado também

[72] Hyman, *Computing*, s.v. "Byte".
[73] Ceruzzi, *History of Modern Computing*, 152.

como *octeto* (*byte* composto por 8 *bits*), sendo indicado por Fragomeni como o mais comumente usado, já em 1986.[74]

A instituição norte-americana conhecida como *ANSI* (American National Standards Institute, em português, Instituto Nacional Americano de Padrões):

> supervisiona a criação, promulgação e uso de milhares de normas e diretrizes que impactam diretamente as empresas em quase todos os setores: a partir de dispositivos acústicos de equipamentos de construção, da produção de leite e gado de distribuição de energia, e muitos mais. ANSI também está ativamente engajada em programas de certificação para avaliar a conformidade com os padrões - incluindo programas intersetoriais globalmente reconhecidos, como a ISO 9000 (qualidade) e ISO 14000 sistemas de gestão (ambiental).[75]

O esforço do *ANSI* que interessa ao contexto do nosso ensaio se direciona ao estabelecimento de elementos computacionais que possam ser comuns à indústria de computadores, em seu sentido mais amplo, ou seja, desde pesquisadores, passando pelas universidades e fornecedores de equipamentos e programas de computador até os usuários finais destes, agindo no sentido de pesquisar, normatizar e manter atualizada tal normatização. Um exemplo dessa atividade, *ASCII* (American Standard Code for Information Interchange, em português, Padrão Americano de Código para Troca de Informações) permite a codificação por parte dos computadores, que "só podem compreender números", fornecendo "um código ASCII [que] é a representação numérica de um caractere, como 'a' ou '@' ou uma ação de algum tipo." Existem diversas representações possíveis, expressas em tabelas (também conhecidas como *page codes*), que permitem, por exemplo, que o mesmo equipamento interprete uma parte de um texto em língua árabe, outra em língua chinesa ou ocidental.

---

[74] Fragomeni, *Dicionário Enciclopédico de Informática*, s.v. "Baite".
[75] http://www.ansi.org/ (acessado em 25 de outubro de 2014).

Alguns equipamentos, entretanto, especialmente os *mainframes* construídos pela *IBM*, adotaram um padrão específico, designado *EBCDIC* (Extended Binary Coded Decimal Interchange Code, em português Código Binário Estendido para Troca Decimal) desenvolvido para interpretação de cartões perfurados no início da década de 1960[76].

Neste parágrafo, é possível que o leitor acredite vir a encontrar uma dicotomia, já que queremos caracterizar *data* e *datum* logo depois de termos caracterizado *bit* e *byte* (através dos quais, *caractere* foi também caracterizado). Dicotomia porque alguns caminhos de análise podem apontar para características absolutamente distintas desses conceitos e ao final das contas, Codd sequer os cita. Por um lado, ao discutirmos adiante a questão de *Data Bases* versus *Data Banks*, no texto de Codd, acreditamos que o leitor perceberá o elo existente entre todos estes termos e nosso eixo principal de análise; por outro lado, desejamos demonstrar que todos eles, ainda que pudessem ser utilizados como similares, são elementos que se unem para estabelecer a base de uma construção que pode ser chamada de *Informática*, *Computação* (*Computing* em inglês) ou *Processamento de Dados*, porém, para não introduzir termos demais, elementos fundamentais da *indústria da tecnologia da informação*.

> *Data* [...] também tratado como singular ainda que a forma singular seja estritamente *datum*. 1 fatos conhecidos usados para inferir ou levar em consideração. 2 quantidades ou caracteres operados em um computador etc.[77]

> *Data* [o *dado*] a ser processado por máquinas IBM deve ser perfurado em um cartão de acordo com um padrão determinado. Consequentemente, colunas no cartão são agrupados e reservados para a gravação de cada fato sobre uma transação de negócio.[78]

---

[76] http://www.asciitable.com/ (acessado em 25 de outubro de 2014).
[77] *Oxford Dictionary of Current English*, ed.1998, s.v. "data".
[78] IBM punched card. From IBM corporation, "IBM Data Processing Functions," Brochure 224-8208-5, ca. 1963, citado em Ceruzzi, *A History of Modern Computing*, 17.

Nestes dois trechos, oferecemos uma perspectiva de como o *dado* (conforme já explicitado, *data* ou *datum*), sendo existente no mundo físico como, por exemplo, o gênero de um aluno, masculino ou feminino, é manipulado em um computador: esse *dado* é interpretado como um *octeto* (como indicamos antes, mais frequentemente), mais provavelmente um "M" ou um "F", portanto, uma sequencia *8-bit*, que um ser humano visualiza como um *caractere*[79]. A máquina "lê" um furo em uma dada posição de um cartão e realiza uma tradução, retendo um dado conjunto de oito *bits* se for "M" ou outro se for "F" – uma possibilidade bastante comum em 1963, ocasião em foi escrito o texto utilizado para a nota imediatamente anterior a este parágrafo.

A trajetória entre *dado* e *informação* constitui um profundo *labirinto*, outra possibilidade para pesquisas futuras. Entretanto, acreditamos que a leitura do ensaio de Codd demonstra que ele procura uma solução para algumas das armadilhas de tal *labirinto*.

Codd utiliza por vezes o termo *coleção* (*collection* em inglês). Considerando que um sinônimo possível seria *conjunto* (*set* em inglês), do mesmo modo que utilizamos *acervo* como alternativa a esses dois termos mas, já que uma visão muito específica de conjunto é defendida no assim designado ensaio seminal, podemos perceber que cada termo designa uma ideia que, muito embora convergente, poderia nos levar à seguinte imagem: *coleção* ou *acervo* como conjuntos organizados de *entidades*[80] cuja *teoria axiomática de conjuntos* utilizada através de um *SGBDR* em um

[79] Um ser humano letrado em uma língua ocidental.

[80] Autores como Codd e Date trabalharão intensamente em momentos posteriores em um "acervo de conhecimentos" denominado "Modelagem de Dados", que utiliza o termo *entidade* de uma forma particular. Tais estudos, apesar de sua importância, estão também além do alcance do presente ensaio.

Computador, e lá defendida por Codd, ofereceria vantagens de organização e acesso. Tais *entidades* podem ser caracterizadas por um termo novo para este texto, *registro*, que ligaremos a *complex*, uma "característica adicional [...] de relações binárias", segundo Childs[81], conceito que Codd evoca no estabelecimento dessa organização. Exploraremos, entretanto estas conexões, em detalhe, mais adiante. Neste ponto, recorremos a Fragomeni para oferecer a caracterização de *registro*:

> *Record*, *registration* [em inglês] : Conjunto de *dados* (ou palavras) relacionados, tratados como um todo, em termos lógicos ou físicos. Ex.: um documento completo, os *dados* de um funcionário em um cadastro de pessoal, uma fatura em controle de estoques. Geralmente um conjunto de *registros* forma um *arquivo* [file em inglês].[82]

O termo *file* (arquivo em português) é utilizado por Codd sem comentários adicionais. O autor, portanto, o consideraria como algo "bem sabido" o que justifica esta nossa trajetória desde *bit*, da qual oferecemos aqui um resumo: um *byte* (ou *octeto*) é registrado em um *computador* como conjuntos de oito *bits*, cada *byte* representa um *caractere*, cada coleção de *caracteres* (vazia, unitária ou múltipla) uma *palavra*[83], enquanto *palavras* são agregadas em *campos*[84] os quais são reunidos em *registros* que, por sua vez, têm o potencial de fornecer informações. *Arquivos* são acervos de *registros*. Outros termos que se pode utilizar para o mesmo significado de *arquivo* são *depósito*, *repositório* ou *recipiente*. Existe alguma distância entre o significado de *recipiente* que utilizamos de forma ordinária (uma bolsa, um baleiro, um porta-lápis) e aquele que oferecemos aqui.

---

[81] Childs, "Feasibility of a set-theoretic data structure", V.
[82] Fragomeni, *Dicionário Enciclopédico de Informática*, s.v. "Registro".
[83] O termo palavra aqui está sendo utilizado em seu sentido "linguístico" apontado por Ferreira em *Novo Dicionário da Língua Portuguesa*. No âmbito da arquitetura de um computador, é possível encontrar outros significados para esse termo.
[84] O leitor com certeza já se deparou com um "Campo". Basta recordar qualquer tipo de ficha cadastral, formulário de matrícula ou de seguros que já tenha preenchido. Cada espaço a ser preenchido, como nome, gênero, data de nascimento constitui um *campo*.

Estamos nos referindo neste ponto a uma estrutura que contém *dados* e os mantém organizados de uma certa forma que o *Computador* pode interpretar, armazenar e recuperar (à qual Codd se refere, através dos termos *Store* e *Stored.* A*rmazenamento* e *armazenado*, em português):

> *Memória*; também *armazém*/*armazenamento*. Sistema ou dispositivo o qual é ou pode ser utilizado para manter informações em alguma forma que pode ser acessada, entendida e utilizada por um *computador* ou sistema de processamento de dados. É evidente que de maneira na qual *memórias* são incorporadas é fundamental aos sistemas de computação. *Memórias* podem ser classificadas de diferentes modos: (1) permanente (fixa), semi-permanente, alterável; (2) de acesso serial, de acesso aleatório [random em inglês]; (3) on-line, off-line; (4) de endereço fixo, com endereço determinado pelo conteúdo; (5) por tempo de acesso; (6) por capacidade; (7) volátil, não volátil; (8) em relação à posição na hierarquia: registrador, área de rascunho [scratch pad em inglês], memória principal, memória de suporte [ou para cópia de segurança, backing memory em inglês], *armazenamento* em massa; (9) tecnologia utilizada para construir a memória; (10) confiabilidade; (11) custo. Outras formas de classificação são utilizadas e a estrutura de dados lógica mantida em sistemas com grandes *memórias* é claro, um assunto muito complexo.[85]

Neste ponto queremos abandonar uma simplificação anterior, afirmando que *depósitos de dados* poderiam ser considerados como qualquer tipo de organização de dados que se pode encontrar em um *computador*, enquanto *banco de dados*, ao menos no sentido que utilizaremos doravante, e que acreditamos ser o sentido utilizado por Codd, seria um tipo específico de *depósitos de dados*, cuja organização privilegia a manipulação de grandes volumes de *dados* (observar comentários anteriores sobre a questão da *dimensão*).

Codd utiliza como título de seu ensaio a expressão *Data Banks* (*Banco de Dados*, em português) e na maioria das vezes que se refere a essa estrutura de dados, no corpo deste. Nas *key words*, aparece também o termo *Data Bases* (*Base de Dados*, em português), assim como em alguns

---

[85] Hyman, *Computing*, s.v. "Memory". Grifos nossos.

pontos do texto. Não localizamos, no ensaio estudado, qualquer informação que indique uma possível diferença entre essas duas formas.

Fragomeni, entretanto, oferece uma possibilidade de diferenciação:

> Banco de Dados (Data Banks): Arquivo de dados de diversas fontes, armazenado de forma a possibilitar o acesso por vários usuários. São muito importantes sua estrutura e organização, bem como os programas de acesso e tratamento. Atualmente [1986] grandes sistemas têm sido desenvolvidos especialmente para a gerência de bancos de dados.
>
> Nota: Um banco de Dados pode conter várias bases de Dados.
>
> *Base de Dados* (*Data Bases*) 1) Coleção de dados fundamental a um sistema, empresa ou empreendimento. 2) Conjunto de dados, formando parte ou o todo de um outro conjunto de dados, onde pelo menos um dos arquivos é definido para uma aplicação ou para um sistema de processamento de dados.
>
> Nota 1: Como os dados, em um computador, existem sob a forma de sequencias de bites armazenados, a recuperação da informação requer o conhecimento de sua estrutura lógica, cuja ponte com a estrutura física é a estrutura de alocação, o elemento da base de dados que lida com o mapeamento da estrutura lógica dos dados para o meio físico, um dos tipos de gerência de memória.
>
> Nota 2: O termo ainda não alcançou um significado padronizado que fosse largamente aceito. É muito aceito como um novo nome para o conceito de arquivo (*file*), que veio para a área de PD [Processamento de Dados] desde antes do computador. A diferença mais importante é que a base de dados deve estar numa memória de acesso direto, para que a UCP [Unidade Central de Processamento, ou CPU do inglês Central Processing Unit] possa utilizar seus dados e referências cruzadas em tempo razoável. Ao contrário, um arquivo com referências cruzadas pode, tecnicamente, estar em fita magnética. O termo Referência Cruzada (cross refeerence [sic]) não é muito usada para bases de dados, mas sim Relação (relation) (por ex., relacionamento (relationship) entre os tipos de registros).
>
> Nota 3: A diferença entre *Base* e *Banco de Dados* é que geralmente forma um sistema completo e estruturado especificamente, combinado com programas adequados para sua manipulação, e pode ser acessado por diversos usuários.
>
> Nota 4: existem basicamente quatro tipos de estruturas usadas: a) coleção de bases de dados independentes; b) base de dados centralizada (solitária); c) base de dados interfaciada; d)base de dados distribuída. [86]

[86] Fragomeni, *Dicionário Enciclopédico de Informática*, s.vv. "Banco de Dados", "Base de Dados".

Para concluir esta tarefa de entendimento e apropriação do título do ensaio de Codd, utilizaremos termos do próprio ensaio que apontam para a qualidade *Relacional* do modelo preconizado:

> o termo relação é usado aqui em seu aceito senso matemático. Dados os conjuntos S1, S2, ..., Sn (não necessariamente distintos), R é a relação desses n conjuntos se é um conjunto de n-tuplas cada qual possui seu primeiro elemento desde S1, seu segundo elemento desde S2 e assim por diante.[87]

Ao discutirmos adiante o conceito de *complex* e depois na seção denominada *SGBDRs*, acreditamos ter tido êxito em esclarecer devidamente o sentido *Relacional* dos *Bancos de Dados*. Gostaríamos também de assinalar que os termos *tupla*, *registro* e *complex* podem ser entendidos como apontando para o mesmo objeto, especialmente se considerarmos que uma *tupla*, enquanto resultado de uma atividade de recuperação de dados, os quais estariam originalmente organizados em *registros*, ou segundo o que ficará evidente mais adiante, *complex*, pode ser a origem de um novo registro.

[87] Codd, "A Relational Model of Data for Large Shared Data Banks", 377. *Tupla*, em Matemática pode ser considerado lista ordenada de elementos. Várias interpretações são oferecidas no âmbito dos *SGBDR*s, mas utilizaremos neste ponto como resultado de uma ação de recuperação de dados.

3.3.Construção e justificação do modelo

Queremos crer que o Sumário de Codd é suficiente em si, mas utiliza alguns termos que merecem ser elucidados. Antes, porém, acreditamos oportuno chamar a atenção para o entorno de seu desenvolvimento, qual seja, o "Laboratório de Pesquisa da IBM em San José, Califórnia":

> Em 1952, a IBM enviou Reynold Johnson a San José para dar início a um novo laboratório de pesquisas. Johnson estava na empresa desde 1934, depois de ter ensinado ciências em Ironwood, Michigan, [...] Ele era um administrador heterodoxo com uma intuição bem precisa sobre tecnologia. Quando a força aérea quis um sistema de estoque com acesso aleatório, Johnson colocou as 50 pessoas de seu laboratório em ação tentando de tudo – tiras, bastões, fitas, placas planas [...] O laboratório logo decidiu usar discos horizontais giratórios cobertos com material magnético [...] O produto, como foi apresentado em maio de 1955 [...] Nos anos 1980, armazenar um gigabyte de informação – o equivalente a aproximadamente uma hora de vídeo com qualidade padrão de TV – requeria uma máquina de quase 230 kg, do tamanho de uma geladeira. Em 2010, um gigabyte poderia ser armazenado em um disco menor do que uma moeda.[88]

Apesar do trecho destacado não fazer menção direta ao trabalho de Codd, podemos observar características interessantes. San José foi estabelecido em torno de um professor de uma universidade que há mais de uma década trabalhava na *IBM*. A partir de uma demanda de uma instituição militar norte-americana, desenvolve um produto tecnológico que abre espaço para a questão vital do ensaio em análise: uma vez que o acesso ao *dado* pode ser realizado pelo *computador* de maneira aleatória, faz-se necessário promover facilidades de interação com esses *dados*. No final da referência, mantivemos alguns dados quantitativos por constituírem informações que costumam ser raras, conforme já tivemos a oportunidade de apontar.

Ainda analisando o sumário, podemos observar o destaque às duas formas de interação citadas pelo autor. Ambas as formas se fazem

---

[88] IBM. *Tornando o Mundo Melhor*,46-9.

presentes até a atualidade ainda que a tecnologia hoje (2014) utilizada tenha permitido avanços significativos em termos de tamanho e velocidade dos *computadores*, o que pelo menos em parte está exemplificado no final da referência anterior. Codd se refere a *terminais* e *programas de aplicação*. As modernas *APPs*, muito embora explorando recursos gráficos e, portanto, oferecendo recursos de interatividade absolutamente diferentes daqueles do final da década de 1960[89], contêm ainda em seu interior amontoados de instruções organizadas, a qual se convencionou denominar *algoritmos*[90]. Possivelmente o leitor esteja utilizando um dispositivo móvel (em inglês, *Mobile*, popularmente, *Smart Phone* ou *Tablet* dependendo de suas capacidades) para a leitura deste ensaio, ou seu *computador* pessoal no formato *de mesa* ou *portátil*, mas provavelmente nunca viu um *terminal*, que pode ser descrito como "qualquer dispositivo capaz de enviar e/ou receber informação através de um canal de comunicação."[91] De posse dessa descrição, e observando – imaginemos – seu *Mobile*, o leitor imaginará um nosso equívoco. Ocorre que a descrição, colocada em sua época, deveria remeter o leitor a um objeto específico e absolutamente diferente daquele que modernamente costumamos encontrar em uso. Observe-se que, salvo alguns dispositivos isolados, a essa época (1969~70) e ainda experimentais[92], o trabalho profissional em Sistemas de Informação era

---

[89] Até que as interfaces gráficas pudessem contar com recursos tecnológicos adequados, o que se daria por volta das décadas de 1980 e 90, as aplicações na década de 1960, quando "conversavam" com as pessoas, o faziam através de telas monocromáticas compostas tão somente de caracteres – que eventualmente indicavam a necessidade de uso de alguma tecla especial, que ainda existem nos teclados dos computadores de mesa e portáteis (laptops ou notebooks), como "F1", "Shift" ou "Ctrl".

[90] Em Bispo, "Dos fundamentos da matemática ao surgimento da teoria da computação por Alan Turing", a questão *algoritmos* é largamente analisada.

[91] Fragomeni, *Dicionário Enciclopédico de Informática*, s.v. "terminal."

[92] Segundo Gates, em *A Estrada do Futuro*, 29, 59 e 69, a IBM lançaria seu PC - *Personal Computer* – em 1981, e a *Apple* começaria a comercializar seu *Macintosh* gráfico em 1984. Os kits para montagem dos primeiros computadores pessoais baseados no kit *Altair 8800*, vendido pela *MITS* com processador *Intel 8080*, seriam comercializados a partir de 1975. Por volta de 1977, *Apple*, *Commodore* e *Radio Shack* também já haviam entrado no ramo.

realizado em dispositivos que possuíam uma ligação direta e exclusiva com um computador central, ligação esta que era materializada por um par de fios que o leitor encontrará ainda ligando postes a residências e sendo utilizados em telefonia tradicional. Em função desse tipo de conexão com o *Mainframe,* tais dispositivos eram designadas *terminais*. Podiam se materializar como equipamentos bastante similares a *máquinas de escrever* e *impressoras* ou ainda como (mais modernamente) unidades providas de *teclado* (usualmente bastante semelhante àquele que se utiliza até nossos tempos) e *vídeo de raios catódicos,* normalmente não providos de capacidade de processamento. Bill Gates fornece uma descrição dessa configuração mais básica quando inicia o relato de sua trajetória[93].

Acreditamos ser oportuno oferecer uma caracterização de *Mainframe*. Muito embora Fragomeni apresente este termo como *deprecated* (caduco, obsoleto, em português), o termo era e é utilizado para designar um dispositivo central de processamento, que se entende como o componente do sistema com capacidade de processamento[94]. Hyman evidencia esta visão denominando-o como "centro de um computador"[95]. O termo *Mainframe* evoluiu, a partir da popularização dos computadores de uso pessoal, para designar sistemas de máquinas e programas reunidos para processar dados corporativos, como as folhas de pagamento das empresas, ou todos os dados relacionados às atividades dos clientes de uma instituição financeira.

Entendemos que Codd se apoia especialmente no ensaio de Childs para justificar o predicado *Relacional* do produto de seu trabalho. Estabelece

---

[93] Gates, *A Estrada do Futuro*, 11.
[94] Fragomeni, *Dicionário Enciclopédico de Informática*, s.vv. "main frame," "unidade central de processamento – UCP."
[95] Hyman, *Computing*, s.v. "main frame."

através desse termo a possibilidade de uma organização de arquivos que se opõe a uma organização ligada à hierarquia da informação a extrair, hierarquia esta subordinada ao método de armazenamento em si, detalhando tal dependência e porque a julga falha[96]. Dos conceitos expressos por Childs, este evidencia que parte de um ponto do qual também Codd partirá, qual seja, que:

> este trabalho é motivado por uma suposição de que muitos problemas ao lidar com dados arbitrariamente relacionados pode ser acelerada num computador digital por uma estrutura de armazenamento que permite rápida execução das operações dentro e entre conjuntos de nomes de datum.[97]

No início de sua introdução, Childs oferece detalhes dos problemas ao lidar com dados, aos quais pretende oferecer uma solução:

> O objetivo geral, do qual este artigo é uma parte, é o desenvolvimento de uma estrutura de dados independente da máquina, permitindo processamento rápido de dados relacionados com a atribuição arbitrária tais como: o conteúdo de um livro de telefone, arquivos de biblioteca, relatórios de censo, linhagem da família, redes, etc. Dados que não são intrinsecamente relacionados têm que ser expressos (armazenados) de modo a definir o modo em que eles estão relacionados antes que *qualquer* estrutura de dados seja aplicável.[98]

Os detalhes, os problemas, ou obstáculos, que Childs pretende superar seriam:

1. A dependência então existente entre os dados em si e a forma que eles eram armazenados na memória física do computador, e
2. decorrente do primeiro, a limitação de como tais dados podem ser acessados.

---

[96] Codd, "A Relational Model of Data for Large Shared Data Banks", especialmente na seção denominada "1.2.DEPENDÊNCIAS DE DADOS EM SISTEMAS PRESENTES [ATUAIS]".

[97] Childs, "Feasibility of a set-theoretic data structure", v. "Viabilidade de uma estrutura de dados [baseada na] teoria dos conjuntos. Uma estrutura geral baseada em uma definição reorganizada de relação", que foi apresentado ao Congresso da IFIP de 6 de agosto de 1968 em Edinburgh, Escócia, tendo sido desenvolvido na Universidade de Michigan, na qual Codd obteve seu doutorado em Ciências da Comunicação em 1965, sendo este o único elo de ligação entre estes dois personagens que se pôde localizar. IFIP é o acrônimo em língua inglesa para "International Federation for Information Processing", ou Federação Internacional de Processamento de Informações em português, vide http://www.ifip.org/.

[98] Ibid., 1. Grifos nossos.

Para tentar esclarecer as implicações desses obstáculos, imaginemos um sistema manual em uma biblioteca, qual seja, uma coleção de cartões que contenham o nome do autor, o nome e demais dados da obra e a localização do livro. Tais cartões seriam organizados seguindo a estrutura imposta por sua mídia, ou seja, por nome do autor. Não seria impossível localizar uma obra por seu nome, mas para tanto seria necessário ler cada um dos cartões existentes. Mesmo encontrando (em um hipotético lance de sorte) a obra desejada no primeiro cartão, seria necessário ler todos os demais, uma vez que qualquer outro cartão, inclusive o último cartão, poderia conter uma outra obra (provavelmente de outro autor), com o mesmo título. Tal esforço que, tanto manualmente quanto em termos computacionais, não pode ser considerado como desprezível, justifica a inexistência de certas opções de caminho para obtenção de dados em *sistemas de informação*, conforme exemplificamos na seção *Introdução*, ao comentarmos a limitação de acesso de um correntista.

Retornando ao resumo, observamos que a solução proposta por Childs também é convergente aquela posteriormente apresentada por Codd:

> Para que tal estrutura seja viável, dois problemas devem ser considerados:
>
> 1. A estrutura deve ser geral o suficiente para que os conjuntos envolvidos possam ser irrestritos, permitindo, assim, conjuntos de conjuntos de conjuntos...; conjuntos de pares ordenados, ordenados triplos...; conjuntos de comprimento variável n-tuplas, n-tuplas de conjuntos arbitrários; etc.;
> 2. As operações em conjuntos devem ser de natureza geral, permitindo que quaisquer das operações da teoria dos conjuntos habituais entre as séries como descrito acima, com a garantia de que estas operações serão executadas rapidamente.
>
> Uma condição suficiente para o último é a existência de uma bem-ordenada relação na união dos conjuntos participantes.
>
> Estes problemas são resolvidos neste trabalho, com a introdução do conceito de um 'complexo', [doravante *complex*] que tem um recurso

> adicional de permitir uma extensão natural das propriedades de relações binárias as propriedades de relações gerais.[99]

Convergente mas seguindo uma rota diversa. Childs estabelecerá princípios teóricos, especialmente caracterizando o *complex*, mas limitando-se a tal criação, "com o intuito de desenvolver uma estrutura de dados independente da máquina"[100], ou seja, de como os dados são mantidos na *memória* do *computador*, enquanto Codd parte desse princípio buscando estabelecer uma visão mais prática para sua proposição, qual seja, efetivamente estabelecendo as bases para um *Sistema Gerenciador de Banco de Dados*. Seria possível considerar que ambos alcançam o mesmo objetivo e têm o mesmo propósito somente se olvidarmos que Childs se refere a um objeto mais simples, que identifica como uma "estrutura de dados"[101]. Ao considerar que a "visão relacional [...] constitui uma base sólida para o tratamento de derivabilidade, redundância e consistência das relações - estes são discutidos na Seção 2"[102] o *SGBDR*, produto do trabalho de Codd, assume uma abrangência maior (vide a seção designada *SGBDR* adiante).

Considerando que qualificamos o estabelecimento do *complex* como principal contribuição de Childs, tentaremos caracterizá-lo de forma mais detalhada.

Tanto Codd quanto Childs utilizam os termos Array e Tupla como se fossem de domínio público. Acreditamos oportuno tentar localizar uma interpretação possível de tais ideias, assim, vamos caracterizar *Array como* "*uma s*érie bem ordenada, ou colocada em forma ordenada"[103]; a partir daí,

---

[99] Childs, "Feasibility of a set-theoretic data structure", v.
[100] Ibid., 1.
[101] Ibid.
[102] Codd, "A Relational Model of Data for Large Shared Data Banks", 377.
[103] *Oxford Dictionary of Current English*, ed.1998, s.v. "array."

par ordenado: "representação de dois objetos em uma dada ordem." Suppes, apresenta a "Definição 10", explicando, a partir do "axioma de emparelhamento", tal assertiva, chegando ao seu "Teorema 45"[104].

Para não sermos parciais, observamos que há outras interpretações possíveis. Por exemplo, para Fragomeni, *N-tuples* são "Diversos registradores que funcionam como um único registrador [...] pode ser usado [...] em divisão, para armazenar o quociente parcial e o resto."[105]

Neste contexto, parece ainda ser oportuno observar uma possível interpretação para *N-ary:* "N-ário, relativo à seleção, escolha ou condição que tem N diferentes valores ou estados possíveis."[106]

Childs refere-se a uma "relação geral" como aquela que se pode estabelecer entre quaisquer dois conjuntos, digamos A,B:

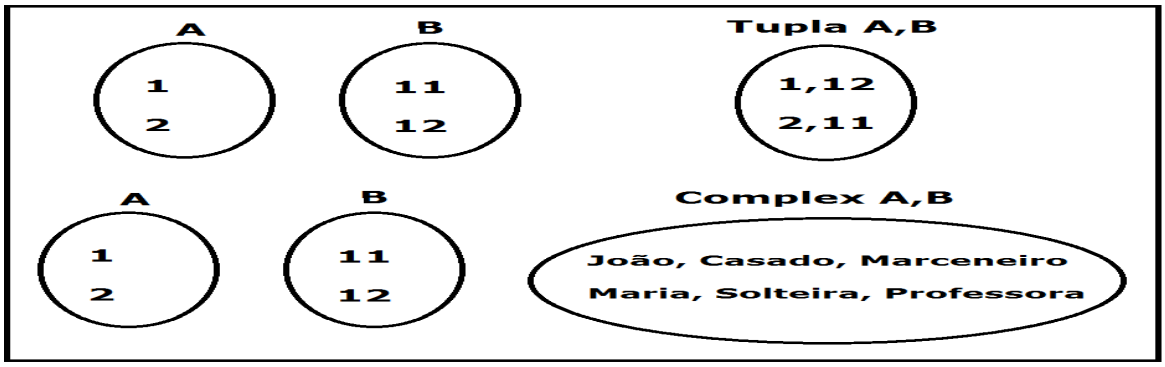
> [...] definir uma n-tupla em termos de uma função [de]
>
> N(n) = {1,... ,n} para um conjunto A por [by]:
>
> $<a_1,a_2,...,a_n> = \{<i,a_i>: i \in N(n) \& ai \in A\}$.
>
> Um problema aqui é justificar a ambiguidade de pares ordenados e 2-tuplas sob operações de conjunto; para pares ordenados
>
> $<a,b>0 \& <x,b>0, <a,b>0 \cap <x,b>0 = \beta$,
>
> mas para 2-tuplas $<a,b>2 \& <x,b>2, <a,b>2 \cap <x,b>2 = \{<2,b>0\}$.
>
> O objetivo deste trabalho é, então, estabelecer uma ampla gama de conjuntos permitidos para uma STDS[107], tal que:
>
> 1. conjuntos de n-tuplas sejam incluídos,
> 2. operações de conjuntos entre n-tuplas sejam significativos,
> 3. todos os conjuntos admissíveis possam ser representado em termos de elementos de Beta, e
> 4. para garantir a rápida execução das operações de conjunto em uma coleção de conjuntos, a união desses conjuntos deve preservar o ordenação.

---

[104] Suppes, Patrick. *Axiomatic Set Theory*, 30-2.
[105] Fragomeni, *Dicionário Enciclopédico de Informática*, s.vv. "N-tuple," "Registrador múltiplo," "Registrador duplo."
[106] Ibid., s.v. "N-ary."
[107] Childs, "Feasibility of a set-theoretic data structure", 1. Manteremos a sigla em Inglês STDS para "Set-Theoretic Data Structure" que se pode traduzir para "estrutura de dados baseada na teoria dos conjuntos".

> O desenvolvimento da teoria dos conjuntos até, mas não incluindo a introdução de, pares ordenados é assumida (ver [2]). Seja N representando os números naturais, e seja N(n) representando os números naturais até e incluindo n. Os números naturais não estão sendo assumidos por eles poderem ser desenvolvidos* num quadro conjunto da teoria, sem a utilização de pares ordenados. O primeiro é a definição de um complex.[108]

Assim, *Tupla* se caracteriza pelo resultado da relação entre dois conjuntos. A importância da definição de um *complex* se dá pela possibilidade de representar uma *Tupla* que se materialize por uma estrutura que seja constituída, que represente, diversos valores, como ilustrado em nesta figura de nossa autoria:



Outro exemplo da transição realizada por Codd entre o conceito matemático e sua utilização em *SGBDR*s pode ser observado ao caracterizar Join (em português, Junção ou União, qual seja, como estabelecer uma ligação útil e precisa entre relações). "Suponha que temos duas relações binárias, que têm algum domínio em comum. Em que circunstâncias é que podemos combinar essas relações para formar uma relação ternária que preserva toda a informação nas relações dadas?"[109], pergunta Codd, que então parte de um "*Produto Cartesiano"* – Relação resultante do *join* entre N relações sem que nenhuma função ou regra ou função de relacionamento

[108] Childs, "Feasibility of a set-theoretic data structure, 4. O * incluso no texto remete à uma referência de Childs à obra de Suppes para definir com exatidão um *complex* vazio.
[109] Codd, "A Relational Model of Data for Large Shared Data Banks", 383.

seja estabelecida. Encontraremos em Suppes, a caracterização de tal operação:

> "O Produto Cartesiano de dois conjuntos A e B (em símbolos: A X B) é o conjunto de todos os pares ordenados <x, y> de modo que x € A e y € B. Por exemplo, se
>
> A = {1,2}
>
> B = {Archimedes, Eudoxus}
>
> Então
>
> A X B = {<1, Archimedes>, <1, Eudoxus >,
>
> <2, Archimedes>, <2, Eudoxus>}.
>
> Formalmente, temos:
>
> Definição 17. A X B = {<x,y>:x € A & y € B}."[110]

Os termos *Forma Normal* e *Consistência* deveriam chamar a atenção do leitor que estivesse estudando os aspectos técnicos do artigo de Codd. Estes constituem dois aspectos capitais, sendo que o primeiro se encarrega de aspectos que permitem a construção de modelos de dados que permitem a representação dos *dados* a partir da "realidade", ou seja, como se organizam os diversos *campos* para constituir um *registro* e como cada *registro*, pertencente então a uma *relação*, se corresponde com as demais *relações*. A partir da *Normalização*, atividade que caracteriza a aplicação da *Forma Normal*, estabelece-se a *Integridade Referencial*, a qual impedirá, por exemplo, que um departamento seja eliminado se ainda tiver empregados subordinados a ele. Além disso, possibilita que não haja retrabalho quando a descrição de um item, por exemplo, é alterada, situação que converge também para uma utilização racional de espaço de *armazenamento*.

Se na década de 1960 o custo de *armazenagem* dos *bytes* era extremo, por outro lado em 2014 esse custo parece desprezível, mas o volume de

[110] Suppes, Patrick. *Axiomatic Set Theory*, 49.

informação gerada e armazenada hoje é assombroso. Do mesmo modo que a *Normalização* se justificava em 1960, justifica-se hoje.

A segunda questão, *Consistência*, é conhecida também por *Integridade Referencial*, a qual, muito embora seja justificada pela *Normalização*, é estabelecida por um *SGBDR* como em nenhum outro sistema de *armazenamento de dados*. Ela será responsável por impedir que no registro de um bebê que acabou de nascer, por um equívoco qualquer, o dia 31 de fevereiro seja aceito, ou que tenha algo diferente do gênero "M" ou "F" (de masculino ou feminino) anotado.

Muito embora estas duas questões sejam vitais para a Computação, ou justamente por esse motivo, não ofereceremos maiores detalhes sobre elas. Isto constituiria diversos trabalhos adicionais, mas observemos que fornecem ao artigo de Codd o status de reunir uma visão que acolhe efetivamente muitos se não todos os fundamentos dos *SGBDR*s.

3.4.SGBDRs

Ainda que tenhamos já estabelecido o significado desta sigla, queremos aqui retomar seu significado, após explorar um possível caminho para sua criação, e assim evidenciar sua importância.

Sistemas Gerenciadores de Bancos de Dados Relacionais foram se tornando complexos conjuntos de algoritmos, que permitem a criação, manutenção e acesso a vastos conjuntos de dados, dos quais se pode depreender informações vitais para os indivíduos e à sociedade. Quando Herman Hollerith criou sua máquina ele permitiu que o trabalho de diversos anos fosse concluído em alguns meses. Quando Winston Churchill perdeu as eleições, depois de finalizada a Segunda Guerra Mundial, semanas foram necessárias para contagem dos votos, enquanto na última eleição presidencial brasileira, em pouco mais de três horas já se haviam apurado mais de 97% dos votos. Se o leitor tentar pagar novamente a mesma conta no mesmo banco, será notificado do equívoco ao invés de desperdiçar seu dinheiro. O Programa de Seguridade Social norte-americano somente foi tornado realidade após ter seus dados devidamente depositados em uma base de dados[111].

Poderíamos escrever alguns livros, como é possível encontrar diversos já escritos, para descrever onde e como os *SGBDR*s interferiram e interferem no quotidiano das instituições e indivíduos, que depositam seus dados com integridade e segurança em tal *objeto tecnológico.* Alguns deles estão enumerados na seção *Bibliografia* de nosso ensaio e são a base da presente seção. São Instituições financeiras, como Bancos Comerciais, Companhias

[111] As primeiras versões dessas bases de dados não se constituiriam como relacionais. Este é um outro exemplo de uma demanda não militar que forneceu recursos à *Indústria da Informática*, recebendo dela o que pode ser considerado como a própria possibilidade de existência.

Seguradoras e Bolsas de Valores; Governamentais como a Receita Federal, o IBGE e a Previdência Social; ou mesmo o controle de uma corrida de automóveis. Para citar mais um exemplo que se encontra presente na vida das pessoas, a "prevenção de fraudes" dos cartões de crédito, que tendem a barrar o uso do um cartão se ele não acontecer dentro de certos parâmetros: imagine o leitor que nunca esteve na cidade de Belém do Pará e que ao lá chegar pela primeira vez decide fazer uma compra em um supermercado no valor de mil reais; provavelmente o sistema da administradora de cartão de crédito emitirá um alerta e pedirá um contato telefônico com o leitor antes de liberar a operação, já que ela ocorre em uma região geográfica incomum e com um valor mais significativo que o usual.

A efetiva explosão do volume dos dados, que se tem nominado como Big Data[112], sugere novas ferramentas de extração e análise de dados, mas estas manipulações, frequentemente, encaminham os dados para depósitos relacionais, a partir do qual podem ser analisados.

Segundo a Forbes, dentre as 100 maiores empresas do mundo, encontramos a Oracle na 94ª posição, lembrando que tal empresa se notabilizou justamente por seu *SGBDR* [113].

Um *SGBDR* provê segurança de um lado controlando quem e quando se pode manipular ou acessar os dados e de outro provendo métodos de redundância de leitura e gravação destes, assim como de cópias (conhecidas como rotinas de Backup). As citadas redundâncias vão,

---

[112] Ver por exemplo o artigo "The Age of Big Data" na edição de 11 de fevereiro de 2012 do The New York Times.

[113] http://www.forbes.com/global2000/#page:1_sort:0_direction:asc_search:oracle_filter:All%20industries_filter:All%20countries_filter:All%20states. Informações de 2012 (acessado em 04 de novembro de 2014).

portanto, garantir a integridade física dos dados, enquanto a *Integridade Relacional* cuidará que a inferência de informações a partir desses dados respeite regras projetadas para tanto.

4.Considerações finais

Em trabalhos como o de Danilo Horta, podemos verificar um exemplo do tipo de trabalho que se apresenta em relação ao nosso objeto, demonstrando uma utilização prática de *SGBDRs*[114]. Em outro exemplo, muito embora apresentando uma visão divergente da de Codd, a autora se refere a este no mesmo sentido daqueles textos enumerados no Anexo 3 – *Outros documentos*[115], assim como podemos observar em uma dissertação que descreve a implementação do *Projeto Biblos*, o qual visava essencialmente atender as necessidades da biblioteca central da *Unicamp*[116].

Do conjunto de ensaios enumerados nesta nota[117], diversos aspectos citados estabelecem conexões entre a Matemática, especialmente a *Teoria Axiomática dos Conjuntos*, e o *Objeto* do presente trabalho, ainda que por vezes de maneira indireta. Análises mais aprofundadas destes ensaios, assim como pesquisas adicionais apontam para possíveis trabalhos futuros. Julgamos, entretanto importante assinalar que tais ensaios costumam apontar para os *SGBDR*s como algo existente, já estabelecido.

Hardgrave, por outro lado, como resultado de um trabalho patrocinado pela NASA, faz uma análise similar a de Codd, mas sem se referir a este, citando documentos do final da década de 1960 e primeira metade da

---

[114] Horta, "Abordagens evolutivas," ix, 1-2.
[115] Oliveira, "Sistema De Operações Em Álgebra Não-Normalizado", 10,14, 31, 137.
[116] Silva, "Implementação de um Sistema de Gerenciamento de Banco De Dados Relacional", 25, 56, 57,59, 85.
[117] Barendregt & Wiedijk, "The Challenge Of Computer Mathematics". Blanke et al., "Deploying General-Purpose Virtual Research Environments For Humanities Research". Campbell-Kelly, "David John Wheeler". Dowek, "The Physical Church–Turing Thesis And Non-Deterministic Computation Over The Real Numbers". Dzamonja, "Measure Recognition Problem". Grattan-Guinness, "Bertrand Russell (1872–1970), Man Of Dissent". Hall, David & Nigel, "The Evolution of The Web and Implications For Research". Kukla et al., "Integrating Open Grid Services Architecture Data Access And Integration With Computational Grid Workflows". Sumner, "Turing Today". Yaikhom et al., "Validation And Mismatch Repair Of Workflows Through Typed Data Streams". Zucker, "A grounded theory of abstraction in artificial intelligence". Os números das páginas não estão citados porque neste momento é o ensaio como um todo que está sendo considerado.

década de 1970[118]. Outro trabalho com características similares foi publicado por Beitz[119]. Ambos indicam possíveis contrapontos, ou seja, como haveria de esperar-se neste tipo de ensaio, os *SGBDR*s não caíram na cabeça de Codd como uma maçã; ele não é o único a lidar com tais questões aquela época.

Para terminar a análise em pauta, acreditamos ser oportuno assinalar que se deveria observar[120]:

> também a versão anterior "Derivabilidade, redundância e consistência das Relações armazenados em grandes bancos de dados," IBM Research Relatório RJ599 (19 de agosto de 1969); reimpresso em ACM SIGMOD Record, vol. 38, No. 1 (Maio de 2009) [...] artigo de 1969 foi primeiro paper de Codd sobre o modelo relacional; o mais amplamente lido (e referenciado), editado em 1970, que é geralmente creditado como sendo o paper seminal no campo, era essencialmente uma versão revisada do anterior.

Os argumentos de Codd no citado ensaio de 19 de agosto de 1969 parecem ser mais voltados para a própria *IBM*. Mesmo se considerarmos o ensaio de 1970 como uma revisão, foi este segundo o trazido a público, sendo o mais conhecido e citado.

Na presente análise pretendemos ter evidenciado, entretanto, que Codd realizou pelo menos uma significativa ordenação do conjunto de questões que envolvem a gênese dos *SGBDRs*. Considerando seu envolvimento com a *IBM,* acreditamos poder afirmar que seu trabalho obteve destaque em função do sucesso da implementação posterior de produtos *IBM* – SQL-DS, DB2 – intensamente fundamentados nas teorias de Codd, expressas a partir do ensaio que pretendemos ter devidamente analisado.

---

[118] Hardgrave, "A Technique for Implementing a Set Processor", 86-94. NASA – Agência Aeroespacial Norte-americana, vide http://www.nasa.gov/.
[119] Beitz, "Set-Theoretic View of Data-Base Representation", 477-94.
[120] ACM A.M.Turing Award.

5.Bibliografia[121]

ACM A.M.Turing Award. Todos os dados biográficos sobre Codd foram obtidos em pesquisas a partir do site da ACM, especialmente http://amturing.acm.org/award_winners/codd_1000892.cfm (acessado em 15 de outubro de 2014).

Alfonso-Goldfarb, Ana M. "Centenário Simão Mathias: Documentos, Métodos e Identidade da História da Ciência." *Circumscribere* 4 (2008): 5-9.

_____, Márcia H. M. Ferraz & Patrícia Aceves. "Uma 'viagem' entre documentos e fontes." *Circumscribere* 12 (2012): v-viii.

_____, Silvia Waisse, Márcia H. M. Ferraz. "From Shelves to Cyberspace: Organization of Knowledge and the Complex Identity of History of Science." *Isis* 104:3 (2013): 551-60.

Anthes, Gary. "Happy Birthday, RDBMS !" *Communications of the ACM*. Maio 2010/Vol. 53, N. 5: 16-7, http://delivery.acm.org/10.1145/1740000/1735223/communications201005dl.pdf?ip=200.144.147.216&id=1735223&acc=ACTIVE%20SERVICE&key= 344E943C9DC262BB%2E5A7B6855A7F5F1B1%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=587740440&CFTOKEN=45100978&__acm__=1413912396_abe74081c4cc2a415f14a423f0b246c1 (acessado em 21 de outubro de 2014).

Bachelard, Gaston. *A formação do espírito científico. Contribuição para uma Psicanálise do Conhecimento*. 5ª Reimpressão. Rio de Janeiro: Editora Contraponto, 1996.

[121] Considerando que a maioria dos documentos consultados foram escritos originalmente em idioma Inglês (especialmente norte-americano), esclarecemos que as traduções são de nossa responsabilidade, salvo especial menção apontada nesta bibliografia.

Banks, Eli Liberato da Costa. "'A máquina que pensa': alguns aspectos das origens da computação." In *História da Ciência. Tópicos Atuais 3*, org. Maria Helena Roxo Beltran, Fumikazu Saito & Laís dos Santos Pinto Trindade, 68-93. São Paulo: LF Editorial, 2014.

_____, "Charles Babbage (1791-1871) e a mecanização do cálculo: das engrenagens a 'máquina de pensar'". Tese de Doutorado. Pontifícia Universidade Católica de São Paulo, 2012.

_____, "O invento de Jacquard e os computadores: alguns aspectos das origens da programação no século XIX". Dissertação de Mestrado, Pontifícia Universidade Católica de São Paulo, 2008.

Barendregt, Henk & Freek Wiedijk. "The Challenge Of Computer Mathematics". *Philosophical Transactions of The Royal Society A – Mathematical, Physical & Engineering Sciences* (2005), http://royalsociety.org (acessado em 29 de outubro de 2013).

Beitz, E. Henry. "Set-Theoretic View of Data-Base Representation". *ACM*. Abril de 1974. 477-94 (acessado em 11 de novembro de 2013).

Bergin, Thomas J. (Tim). "A History of the History of Programming Languages." *Communications of the ACM*. Maio 2007/Vol. 50, N. 5: 69-74, http://cacm.acm.org/magazines/2007/5/5656-a-history-of-the-history-of-programming-languages/pdf (acessado em 21 de outubro de 2014).

Bispo, Danilo Gustavo. "Dos fundamentos da matemática ao surgimento da teoria da computação por Alan Turing." Dissertação de Mestrado, Pontifícia Universidade Católica de São Paulo, 2013.

Blanke, Tobias. Leonardo Candela, Mark Hedges, Mike Priddy, Fabio Simeoni. "Deploying General-Purpose Virtual Research Environments

For Humanities Research". *Philosophical Transactions of The Royal Society A – Mathematical, Physical & Engineering Sciences* (2010), http://royalsociety.org (acessado em 29 de outubro de 2013).

Campbell-Kelly, Martin. "David John Wheeler". *Biographical Memoirs of Fellows of The Royal Society* (2006), http://royalsociety.org (acessado em 29 de outubro de 2013).

Canguilhem, Georges. "Objeto da História da Ciência". Tradução para estudo, com divulgação restrita, de autoria de Sílvia Waisse, do texto "Objet de l'Histoire des Sciences". *Études d'histoire et de philosophie des sciences. Concernant les vivants et la vie*. 7a ed. Paris, J. Vrin, 1994, 9-23.

Ceruzzi, Paul E. *A History of Modern Computing*. Second edition. Cambridge, MA, London, England: The MIT Press, 2003.

_____. *Computing: a concise history*. Cambridge, MA, London, England: The MIT Press, 2012.

Champion, Justin. "Relational databases and the Great Plague in London 1665." *History of Computation* 5 (1993): 2-18.

Childs, D. L. "Feasibility of a set-theoretic data structure — a general structure Two remarks on set theory based on a reconstituted definition of relation". *IFIP Congress*. North Holland Pub. Co.: Amsterdam, 1968, 1-46.

_____. "Description of a Set-Theoretic Data Structure". *Technical Report 3, Concomp Project,* University of Michigan, March 1968, 557-64.

Codd, E. F. "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM* June, 1970. Volume 13 (6): 377-87, http://www.acm.org (acessado em 02 de setembro de 2013).

Reimpresso em milestones de trabalhos de pesquisa selecionados 1958-1982 (MCCA 25th Anniversary Issue), Comunicações do ACM, vol. 26, Nº 1 (Janeiro 1983).

_____. *Relational model for database management.* Reading, Massachusetts: Addison-Wesley, 1990.

_____. *The 1981 ACM Turing Award Lecture*. Delivered at ACM '81, Los Angeles, California, November 9, 1981 http://www.acm.org (acessado em 16 de outubro de 2014).

_____. *The Relational Model for Database Management: Version 2*, http://www.acm.org (acessado em 02 de setembro de 2013).

_____. & C. J. Date. *Interactive support for non-programmers: the relational and network approaches*. Yorktown Heights, New York: IBM Corp., 1974.

Date C. J. "Edgar F. Codd August 23rd, 1923 - April 18th, 2003 a tribute and personal memoir." *ACM's SIGMOD Record* (32):4-13, http://www.acm.org (acessado em 02 de setembro de 2013).

_____. *Bancos de Dados. Tópicos Avançados*. Trad. Newton Dias de Vasconcellos. Rio de Janeiro: Editora Campus, 1983.

_____. *Introdução a Sistemas de Bancos de Dados*. 8ª ed. Trad. Daniel Vieira. Rio de Janeiro: Editora Campus, 2004.

Dowek, Gilles. "The Physical Church–Turing Thesis And Non-Deterministic Computation Over The Real Numbers". *Philosophical Transactions of The Royal Society A – Mathematical, Physical & Engineering Sciences* (2012), http://royalsociety.org (acessado em 10 de novembro de 2013).

Dzamonja, Mirna. "Measure Recognition Problem". *Philosophical Transactions of The Royal Society A – Mathematical, Physical & Engineering Sciences* (2006), http://royalsociety.org (acessado em 29 de outubro de 2013).

Elmasri, Ramez. Shamkant B. Navathe. *Fundamentals of Database Systems*. 2a. Ed. Redwood City: The Benjamin/Cummings Publishing Company, Inc., 1994.

Ferreira, Aurélio Buarque de Holanda. *Novo Dicionário da Língua Portuguesa*. 2ª. Edição. Rio de Janeiro: Editora Nova Fronteira, 1986.

Fragomeni, Ana Helena. *Dicionário Enciclopédico de Informática*. São Paulo: Livraria Nobel, Rio de Janeiro: Editora Campus, 1986.

Gates, Bill. *A Estrada do Futuro*. Trad. Beth Vieira, Pedro Maia Soares, José Rubens Siqueira e Ricardo Rangel. São Paulo: Companhia das Letras, 1995.

Grattan-Guinness, Ivor. "Bertrand Russell (1872–1970), Man Of Dissent." *Notes & Records of The Royal Society* (2009), http://royalsociety.org (acessado em 29 de outubro de 2013).

Hall, Wendy, David De Roure & Nigel Shadbolt. "The Evolution of the Web and Implications for Research". *Philosophical Transactions of The Royal Society A – Mathematical, Physical & Engineering Sciences* (2008), http://royalsociety.org (acessado em 29 de outubro de 2013).

Hardgrave, W.T. "A Technique for Implementing a Set Processor". *ACM*. s.d. 86-94 (acessado em 11 de novembro de 2013).

Horta, Danilo. "Abordagens evolutivas para agrupamento relacional de dados". Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação (IMC-USP). São Carlos, 2010.

Hyman, Anthonyh. *Computing: A Dictionary of Terms,Concepts and Ideas.* London: Arrow Books, 1976.

IBM. *Tornando o Mundo Melhor: As Ideias que Moldaram um Século e uma Empresa* [Comemorativo ao centenário da IBM]. Trad. thebigword – Bia Peine e Daltony Nóbrega. Upper Saddle River, Nova Jersey: IBM Press, 2011.

Knight, Amy. *Como começou a Guerra Fria: O caso Igor Gouzenko e a caçada aos espiões soviéticos*. Trad. Carlos Duarte e Ana Duarte. Rio de Janeiro: Record, 2005.

Kukla, Tamas, Tamas Kiss, Peter Kacsuk & Gabor Terstyanszky. "Integrating Open Grid Services Architecture Data Access And Integration With Computational Grid Workflows". *Philosophical Transactions of The Royal Society A – Mathematical, Physical & Engineering Sciences* (2009), http://royalsociety.org (acessado em 29 de outubro de 2013).

Latour, Bruno. *Ciência em Ação. Como seguir cientistas e engenheiros sociedade afora.* Trad. Ivone C. Benedetti. São Paulo: UNESP, 2000.

Lawson, Clive. "An Ontology of Technology: Artefacts, Relations and Functions." *Techné: Research in Philosophy and Technology*. Ed. Joseph C. Pitt, Virginia Tech, Electronic Journals. Winter 2008. Volume 12. Number 1 - University of Cambridge. http://scholar.lib.vt.edu/ejournals/SPT/v12n1/lawson.html. (acessado em 29 de outubro de 2014).

_____. "Technology, Technological Determinism and the Trans-formational Model of Technical Activity." *Contributions to Social Ontology*. London: Routledge. http://www.google.com.br/

url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CCIQFjAA&url=http%3A%2F%2Fwww.csog.group.cam.ac.uk%2Fiacr%2Fpapers%2FLawsonC.doc&ei=LGNTVL7xFMGlgwSUuITIBw&usg=AFQjCNHgGHOknjj11jqAPaihGJk4E_7ntg&bvm=bv.78677474,d.eXY. (acessado em 31 de outubro de 2014).

Leite, Jair Cavalcanti. "Modelos e Formalismos para a Engenharia Semiótica de Interfaces de Usuário." Tese de doutorado, Pontifícia Universidade Católica do Rio de Janeiro, 1998.

Mahoney, Michael S. "The Structures of Computation and the Mathematical Structure of Nature"*. Rutherford Journal: The New Zealand Journal for the History and Philosophy of Science and Technology* 3 (2010).

Maier, David. *The Theory of Relational Databases*. Rockville: Computer Science Press, Inc., 1983.

Mashey, John R., "Languages, Levels, Libraries, and Longevity". *QUEUE dezembro/janeiro 2004-2005*: 33-8. http://delivery.acm.org/10.1145/1040000/1039532/p32-mashey.pdf?ip=200.144.145.4&id=1039532&acc=OPEN&key=344E943C9DC262BB%2E5A7B6855A7F5F1B1%2E4D4702B0C3E38B35%2E6D218144511F3437&CFID=587758291&CFTOKEN=61013909&__acm__=1413914776_38543b1bc01196d688d8d39206eeafb3 (acessado em 21 de outubro de 2014).

McMahon, Robert J. *Guerra Fria*. Trad. Rosaura Eichenberg. Porto Alegre: L&PM Pocket, 2012.

Nicoletti, Maria do Carmo. "Forma normal prenex". *Notas de Aula do Curso de Introdução a Lógica*. UFSCAR. http://www2.dc.ufscar.br/~carmo/intro_logica.htm (acessado em 05 de novembro de 2014).

Oliveira, Hilda Carvalho de. "Sistema De Operações Em Álgebra Não-Normalizado". Dissertação de Mestrado, Instituto de Matemática, Estatística e Ciência da Computação, Universidade de Campinas. 1991.

Oracle Corporation. Oracle Database SQL Language Reference 12c Release 1, E17209-14. 2013. http://docs.oracle.com/cd/E16655_01/server.121/e17209/toc.htm (acessado em 18 de outubro de 2013).

Oxford Dictionary of Current English, The. Revised edition. New York: Oxford University Press, 1998.

Panchenko, B., Dirk Leinders, & Jan Van Den Bussche. "Framework design of a domain-key schema of a relational database." *Cybernetics and Systems Analysis* 48:3 (2012): 469-78.

Ramakrishnan, Raghu. Johannes Gehrke. *Sistema de Gerenciamento de Banco de Dados*. 3ª ed. Trad. Célia Taniwake & João Eduardo Nóbrega Tartello. São Paulo: McGraw Hill, 2008.

Silberschatz, Abraham. Henry F. Korth. S. Sudarshan. *Sistema de Banco de Dados*. 5a ed. Trad. Daniel Vieira. Rio de Janeiro: Editora Campus, 2006.

Silva, Maria De Fátima R. O. P. da. "Implementação de um Sistema de Gerenciamento de Banco de Dados Relacional". Dissertação de Mestrado, Instituto de Matemática, Estatística d Ciência da Computação da Universidade Estadual de Campinas – Unicamp, 1984.

Sumathi S., & S. Esakkirajan. *Fundamentals of Relational Database Management Systems*. Berlin: Springer, 2007.

Sumner, James. "Turing Today". *Notes & Records of The Royal Society* (2012), http://royalsociety.org (acessado em 29 de outubro de 2013).

Suppes, Patrick. *Axiomatic Set Theory*. Van Nostrand: Princeton, 1960.

United States patent number 5.627.961 http://www.google.com.br/patents?hl=pt-BR&lr=&vid=USPAT5627961&id=ZBsgAAAAEBAJ&oi=fnd&dq=United+States+patent+number+5,627,961&printsec=abstract#v=onepage&q=United%20States%20patent%20number%205%2C627%2C961&f=false (acessado em 11 de setembro de 2013).

Universidade de Minnesota. Charles Babbage Institute Center for the History of Information Technology. *Conference on Data Systems Languages Records*, 1959-1987. https://www.lib.umn.edu/special. (acessado em 21 de outubro de 2014).

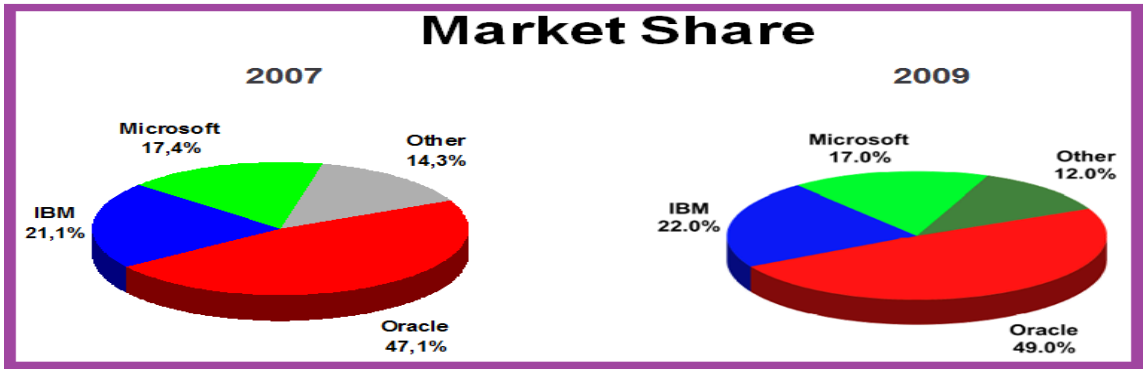
Yaikhom, Gagarine, Malcolm P. Atkinson, Jano I. Van Hemert, Oscar Corcho & Amy Krause. "Validation And Mismatch Repair Of Workflows Through Typed Data Streams". *Philosophical Transactions of The Royal Society A – Mathematical, Physical & Engineering Sciences* (2011), http://royalsociety.org (acessado em 29 de outubro de 2013).

Yong, Chi Shao. *Banco de Dados. Organização Sistemas e Administração.* São Paulo: Editora Atlas, 1988.

Zucker, Jean-Daniel. "A grounded theory of abstraction in artificial intelligence". *Philosophical Transactions of The Royal Society B – Biological Sciences* (2003), http://royalsociety.org (acessado em 29 de outubro de 2013).

Anexo 1 – Market Share de SGBDRs

Market Share [participação dos fornecedores no mercado de *SGBDRs*]. Fonte: Gerência de Produtos Oracle.



Anexo 2 – Eixo principal: Artigo seminal de Codd.

Neste anexo, apresentamos a versão em português que preparamos para auxiliar no entendimento daquele que é o principal eixo do presente ensaio, cujo título é **"Relational Model of Data for Large Shared Data Banks".**[122]

Apresentamos inicialmente a forma digitada desde o original, a qual foi executada da forma mais fiel possível, tentando preservar ao máximo a forma e conteúdo do documento publicado. Em seguida, a tradução por nós executada e de nossa inteira responsabilidade.

Forma digitada desde o original

Communications of the ACM 377-87 Volume 13/Number 6/June, 1970

Information Retrieval P. BAXENDALE, Editor

Relational Model of Data for Large Shared Data Banks

E. F. CODD

IBM Research Laboratory, San Jose, California

---

[122] Codd, "A Relational Model of Data for Large Shared Data Banks", 377-87.

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on n-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity

CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

1. Relational Model and Normal Form

1.1. INTRODUCTION

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to

data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

In contrast, the problems treated here are those of *data independence* - the independence of application programs and terminal activities from growth in data types and changes in data representation - and certain kinds of *data inconsistency* which are expected to become troublesome even in nondeductive systems.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only - that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations - these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the *"connection* trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in

various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. DATA DEPENDENCIES IN PRESENT SYSTEMS

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. Ordering Dependence.

Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is closely associated with the hardware-determined ordering of addresses. For example, the records of a file concerning parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a sub-ordering

of) the stored ordering. Those application programs which take advantage of the stored ordering of a file are likely to fail to operate correctly if for some reason it becomes necessary to replace that ordering by a different one. Similar remarks hold for a stored ordering implemented by means of pointers.

It is unnecessary to single out any system as an example, because all the well-known information systems that are marketed today fail to make a clear distinction between order of presentation on the one hand and stored ordering on the other. Significant implementation problems must be solved to provide this kind of independence.

1.2.2. Indexing Dependence.

In the context of formatted data, an index is usually thought of as a purely performance-oriented component of the data representation. It tends to improve response to queries and updates and, at the same time, slow down response to insertions and deletions. From an informational standpoint, an index is a redundant component of the data representation. If a system uses indices at all and if it is to perform well in an environment with changing patterns of activity on the data bank, an ability to create and destroy indices from time to time will probably be necessary. The question then arises: Can application programs and terminal activities remain invariant as indices come and go?

Present formatted data systems take widely different approaches to indexing. TDMS [7] unconditionally provides indexing on all attributes. The presently released version of IMS [5] provides the user with a choice for each file: a choice between no indexing at all (the hierarchic sequential organization) or indexing on the primary key only (the hierarchic indexed

sequential organization). In neither case is the user's application logic dependent on the existence of the unconditionally provided indices. IDS [8], however, permits the file designers to select attributes to be indexed and to incorporate indices into the file structure by means of additional chains. Application programs taking advantage of the performance benefit of these indexing chains must refer to those chains by name. Such programs do not operate correctly if these chains are later removed.

1.2.3. Access Path Dependence.

Many of the existing formatted data systems provide users with tree-structured files or slightly more general network models of the data. Application programs developed to work with these systems tend to be logically impaired if the trees or networks are changed in structure. A simple example follows.

Suppose the data bank contains information about parts and projects. For each part, the part number, part name, part description, quantity-on-hand, and quantity-on-order are recorded. For each project, the project number, project name, project description are recorded. Whenever a project makes use of a certain part, the quantity of that part committed to the given project is also recorded. Suppose that the system requires the user or file designer to declare or define the data in terms of tree structures. Then, any one of the hierarchical structures may be adopted for the information mentioned above (see Structures 1-5).

Structure 1. Projects Subordinate to Parts

File	Segment	Fields
F	PART	part #
		part name

		part description
		quantity-on-hand
		quantity-on-order
	PROJECT	project #
		project name
		project description
		quantity committed

Structure 2. Parts Subordinate to Projects

File	Segment	Fields
F	PROJECT	project #
		project name
		project description
	PART	part #
		part name
		part description
		quantity-on-hand
		quantity-on-order
		quantity committed

Structure 3. Parts and Projects as Peers Commitment Relationship Subordinate to Projects

File	Segment	Fields
F	PART	part #
		part name
		part description

		quantity-on-hand
		quantity-on-order
G	PROJECT	project #
		project name
		project description
	PART	part #
		quantity committed

Structure 4. Parts and Projects as Peers Commitment Relationship Subordinate to Parts

File	Segment	Fields
F	PART	part #
		part description
		quantity-on-hand
		quantity-on-order
	PROJECT	project #
		quantity committed
G	PROJECT	project #
		project name
		project description

Structure 5. Parts, Projects, and Commitment Relationship as Peers

File	Segment	Fields
F	PART	part #
		part name
		part description

		quantity-on-hand
		quantity-on-order
G	PROJECT	project #
		project name
		project description
H	COMMIT	part #
		project #
		quantity committed

Now, consider the problem of printing out the part number, part name, and quantity committed for every part used in the project whose project name is "alpha." The following observations may be made regardless of which available tree-oriented information system is selected to tackle this problem. If a program P is developed for this problem assuming one of the five structures above – that is, P makes no test to determine which structure is in effect - then P will fail on at least three of the remaining structures. More specifically, if P succeeds with structure 5, it will fail with all the others; if P succeeds with structure 3 or 4, it will fail with at least 1, 2, and 5; if P succeeds with 1 or 2, it will fail with at least 3, 4, and 5. The reason is simple in each case. In the absence of a test to determine which structure is in effect, P fails because an attempt is made to execute a reference to a nonexistent file (available systems treat this as an error) or no attempt is made to execute a reference to a file containing needed information. The reader who is not convinced should develop sample programs for this simple problem.

Since, in general, it is not practical to develop application programs which test for all tree structurings permitted by the system, these programs fail when a change in structure becomes necessary.

Systems which provide users with a network model of the data run into similar difficulties. In both the tree and network cases, the user (or his program) is required to exploit a collection of user access paths to the data. It does not matter whether these paths are in close correspondence with pointer-defined paths in the stored representation – in IDS the correspondence is extremely simple, in TDMS it is just the opposite. The consequence, regardless of the stored representation, is that terminal activities and programs become dependent on the continued existence of the user access paths.

One solution to this is to adopt the policy that once a user access path is defined it will not be made obsolete until all application programs using that path have become obsolete. Such a policy is not practical, because the number of access paths in the total model for the community of users of a data bank would eventually become excessively large.

1.3. A RELATIONAL VIEW OF DATA

The term *relation* is used here in its accepted mathematical sense. Given sets S_1, S_2, … , S_n (not necessarily distinct), *R* is a relation on these *n* sets if it is a set of n-tuples each of which has its first element from S_1, its second element from S_2, and so on.[123] We shall refer to as the j_{th} *domain* of R. As defined above, *R* is said to have *degree n.* Relations of degree 1 are often called *unary,* degree 2 *binary,* degree 3 *ternary,* and degree *n n-ary.*

[123] More concisely, R is a subset of the Cartesian product S1 X S2 X … X Sn.

For expository reasons, we shall frequently make use of an array representation of relations, but it must be remembered that this particular representation is not an essential part of the relational view being expounded. An array which represents an *n-ary* relation *R* has the following properties:

(1) Each row represents an n-tuple of R.

(2) The ordering of rows is immaterial.

(3) All rows are distinct.

(4) The ordering of columns is significant - it corresponds to the ordering $S_1, S_2, \ldots, S_n$ of the domains

on which *R* is defined (see, however, remarks below on domain-ordered and domain-unordered relations).

(5) The significance of each column is partially conveyed by labeling it with the name of the corresponding domain.

The example in Figure 1 illustrates a relation of degree 4, called *supply,* which reflects the shipments-in-progress of parts from specified suppliers to specified projects in specified quantities.

supply (supplier	part	project	quantity)
1	2	5	17
1	3	5	23
2	3	7	9
2	7	5	4
4	1	1	12

Fig. 1. A relation of degree 4

One might ask: If the columns are labeled by the name of corresponding domains, why should the ordering of columns matter? As the example in

Figure 2 shows, two columns may have identical headings (indicating identical domains) but possess distinct meanings with respect to the relation. The relation depicted is called *component.* It is a ternary relation, whose first two domains are called *part* and third domain is called *quantity.* The meaning of *component (x, y, z)* is that part *x* is an immediate component (or subassembly) of part *y*, and *z* units of part *x* are needed to assemble one unit of part *y*. It is a relation which plays a critical role in the parts explosion problem.

component (part	part	quantity)
1	5	9
2	5	7
3	5	2
2	6	12
3	6	3
4	7	1
6	7	1

Fig. 2. A relation with two identical domains

It is a remarkable fact that several existing information systems (chiefly those based on tree-structured files) fail to provide data representations for relations which have two or more identical domains. The present version of IMS/360 [5] is an example of such a system.

The totality of data in a data bank may be viewed as a collection of time-varying relations. These relations are of assorted degrees. As time progresses, each n-ary relation may be subject to insertion of additional n-tuples, deletion of existing ones, and alteration of components of any of its existing n-tuples.

In many commercial, governmental, and scientific data banks, however, some of the relations are of quite high degree (a degree of 30 is not at all uncommon). Users should not normally be burdened with remembering the domain ordering of any relation (for example, the ordering *supplier,* then *part,* then *project,* then *quantity* in the relation *supply).* Accordingly, we propose that users deal, not with relations which are domain-ordered, but with *relationships* which are their domain-unordered counterparts.[124] To accomplish this, domains must be uniquely identifiable at least within any given relation, without using position. Thus, where there are two or more identical domains, we require in each case that the domain name be qualified by a distinctive *role name,* which serves to identify the role played by that domain in the given relation. For example, in the relation *component* of Figure 2, the first domain *part* might be qualified by the role name *sub*, and the second by *super,* so that users could deal with the relationship *component* and *its domains - sub.part, super.part, quantity - without* regard to any ordering between these domains.

To sum up, it is proposed that most users should interact with a relational model of the data consisting of a collection of time-varying relationships (rather than relations). Each user need not know more about any relationship than its name together with the names of its domains (role qualified whenever necessary).[125] Even this information might be offered in menu style by the system (subject to security and privacy constraints) upon request by the user.

---

[124] In mathematical terms, a relationship is an equivalence class of those relations that are equivalent under permutation of domains (see Section 2.1.1).

[125] Naturally, as with any data put into and retrieved from a computer system, the user will normally make far more effective use of the data if he is aware of its meaning.

There are usually many alternative ways in which a relational model may be established for a data bank. In order to discuss a preferred way (or normal form), we must first introduce a few additional concepts (active domain, primary key, foreign key, nonsimple domain) and establish some links with terminology currently in use in information systems programming. In the remainder of this paper, we shall not bother to distinguish between relations and relationships except where it appears advantageous to be explicit.

Consider an example of a data bank which includes relations concerning parts, projects, and suppliers. One relation called *part* is defined on the following domains:

(1) part number

(2) part name

(3) part color

(4) part weight

(5) quantity on hand

(6) quantity on order

and possibly other domains as well. Each of these domains is, in effect, a pool of values, some or all of which may be represented in the data bank at any instant. While it is conceivable that, at some instant, all part colors are present, it is unlikely that all possible part weights, part names, and part numbers are. We shall call the set of values represented at some instant the *active domain* at that instant.

Normally, one domain (or combination of domains) of a given relation has values which uniquely identify each element (n-tuple) of that relation. Such a domain (or combination) is called a *primary key.* In the example

above, part number would be a primary key, while part color would not be. A primary key is *nonredundant* if it is either a simple domain (not a combination) or a combination such that none of the participating simple domains is superfluous in uniquely identifying each element. A relation may possess more than one nonredundant primary key. This would be the case in the example if different parts were always given distinct names. Whenever a relation has two or more nonredundant primary keys, one of them is arbitrarily selected and called the primary key of that relation.

A common requirement is for elements of a relation to cross-reference other elements of the same relation or elements of a different relation. Keys provide a user-oriented means (but not the only means) of expressing such cross-references. We shall call a domain (or domain combination) of relation *R* a *foreign key* if it is not the primary key of *R* but its elements are values of the primary key of some relation *S* (the possibility that *S* and *R* are identical is not excluded). In the relation *supply* of Figure 1, the combination of *supplier, part, project* is the primary key, while each of these three domains taken separately is a foreign key.

In previous work there has been a strong tendency to treat the data in a data bank as consisting of two parts, one part consisting of entity descriptions (for example, descriptions of suppliers) and the other part consisting of relations between the various entities or types of entities (for example, the *supply* relation). This distinction is difficult to maintain when one may have foreign keys in any relation whatsoever. In the user's relational model there appears to be no advantage to making such a distinction (there may be some advantage, however, when one applies

relational concepts to machine representations of the user's set of relationships).

So far, we have discussed examples of relations which are defined on simple domains - domains whose elements are atomic (nondecomposable) values. Nonatomic values can be discussed within the relational framework. Thus, some domains may have relations as elements. These relations may, in turn, be defined on nonsimple domains, and so on. For example, one of the domains on which the relation *employee is* defined might be *salary history.* An element of the *salary history* domain is a binary relation defined on the domain *date* and the domain *salary.* The *salary history* domain is the set of all such binary relations. At any instant of time there are as many instances of the *salary history* relation in the data bank as there are employees. In contrast, there is only one instance of the *employee* relation.

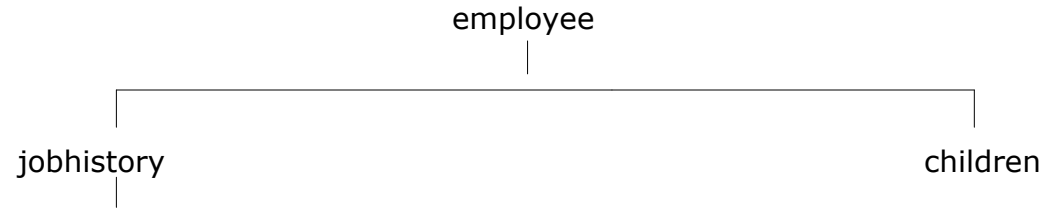
The terms attribute and repeating group in present data base terminology are roughly analogous to simple domain and nonsimple domain, respectively. Much of the confusion in present terminology is due to failure to distinguish between type and instance (as in "record") and between components of a user model of the data on the one hand and their machine representation counterparts on the other hand (again, we cite "record" as an example).

1.4. NORMAL FORM

A relation whose domains are all simple can be represented in storage by a two-dimensional column-homogeneous array of the kind discussed above. Some more complicated data structure is necessary for a relation with one or more nonsimple domains. For this reason (and others to be cited below) the possibility of eliminating nonsimple domains appears worth

investigating.[126] There is, in fact, a very simple elimination procedure, which we shall call normalization.

Consider, for example, the collection of relations exhibited in Figure 3 (a). *Job history* and *children* are nonsimple domains of the relation *employee. Salary history* is a non-simple domain of the relation *job history.* The tree in Figure 3 (a) shows just these interrelationships of the non-simple domains.



employee *(man#,* name, birthdate, jobhistory, children)

jobhistory *(jobdate*, title, salaryhistory)

salaryhistory *(salarydate*, salary)

children *(childname*, birthyear)

Fig. 3(a). Unnormalized set

employee' *(man#,* name, birthdate)

jobhistory' *(man#,* jobdate, title)

salaryhistory' *(man#,* jobdate, salarydate, salary)

children' *(man#,* childname, birthyear)

Fig. 3(b). Normalized set

Normalization proceeds as follows. Starting with the relation at the top of the tree, take its primary key and expand each of the immediately

[126] M. E. Sanko of IBM, San Jose, independently recognized the desirability of eliminating non-simple domains.

subordinate relations by inserting this primary key domain or domain combination. The primary key of each expanded relation consists of the primary key before expansion augmented by the primary key copied down from the parent relation. Now, strike out from the parent relation all nonsimple domains, remove the top node of the tree, and repeat the same sequence of operations on each remaining subtree.

The result of normalizing the collection of relations in Figure 3 (a) is the collection in Figure 3 (b). The primary key of each relation is italicized to show how such keys are expanded by the normalization.

If normalization as described above is to be applicable, the unnormalized collection of relations must satisfy the following conditions:

(1) The graph of interrelationships of the nonsimple domains is a collection of trees.

(2) No primary key has a component domain which is nonsimple.

The writer knows of no application which would require any relaxation of these conditions. Further operations of a normalizing kind are possible. These are not discussed in this paper.

The simplicity of the array representation which becomes feasible when all relations are cast in normal form is not only an advantage for storage purposes but also for communication of bulk data between systems which use widely different representations of the data. The communication form would be a suitably compressed version of the array representation and would have the following advantages:

(1) It would be devoid of pointers (address-valued or displacement-valued).

(2) It would avoid all dependence on hash addressing schemes.

(3) It would contain no indices or ordering lists.

If the user's relational model is set up in normal form, names of items of data in the data bank can take a simpler form than would otherwise be the case. A general name would take a form such as

R (g).r.d

where *R* is a relational name; *g* is a generation identifier (optional); *r* is a role name (optional); *d* is a domain name. Since *g* is needed only when several generations of a given relation exist, or are anticipated to exist, and *r* is needed only when the relation *R* has two or more domains named *d*, the simple form *R.d* will often be adequate.

1.5. SOME LINGUISTIC ASPECTS

The adoption of a relational model of data, as described above, permits the development of a universal data sublanguage based on an applied predicate calculus. A first order predicate calculus suffices if the collection of relations is in normal form. Such a language would provide a yardstick of linguistic power for all other proposed data languages, and would itself be a strong candidate for embedding (with appropriate syntactic modification) in a variety of host languages (programming, command- or problem-oriented). While it is not the purpose of this paper to describe such a language in detail, its salient features would be as follows.

Let us denote the data sublanguage by *R* and the host language by *H*. *R* permits the declaration of relations and their domains. Each declaration of a relation identifies the primary key for that relation. Declared relations are added to the system catalog for use by any members of the user community who have appropriate authorization. *H* permits supporting declarations which indicate, perhaps less permanently, how these relations

are represented in storage. *R* permits the specification for retrieval of any subset of data from the data bank. Action on such a retrieval request is subject to security constraints.

The universality of the data sublanguage lies in its descriptive ability (not its computing ability). In a large data bank each subset of the data has a very large number of possible (and sensible) descriptions, even when we assume (as we do) that there is only a finite set of function subroutines to which the system has access for use in qualifying data for retrieval. Thus, the class of qualification expressions which can be used in a set specification must have the descriptive power of the class of well-formed formulas of an applied predicate calculus. It is well known that to preserve this descriptive power it is unnecessary to express (in whatever syntax is chosen) every formula of the selected predicate calculus. For example, just those in prenex normal form are adequate [9].

Arithmetic functions may be needed in the qualification or other parts of retrieval statements. Such functions can be defined in *H* and invoked in *R*.

A set so specified may be fetched for query purposes only, or it may be held for possible changes. Insertions take the form of adding new elements to declared relations without regard to any ordering that may be present in their machine representation. Deletions which are effective for the community (as opposed to the individual user or sub-communities) take the form of removing elements from declared relations. Some deletions and updates may be triggered by others, if deletion and update dependencies between specified relations are declared in *R*.

One important effect that the view adopted toward data has on the language used to retrieve it is in the naming of data elements and sets.

Some aspects of this have been discussed in the previous section. With the usual network view, users will often be burdened with coining and using more relation names than are absolutely necessary, since names are associated with paths (or path types) rather than with relations.

Once a user is aware that a certain relation is stored, he will expect to be able to exploit[127] it using any combination of its arguments as "knowns" and the remaining arguments as "unknowns," because the information (like Everest) is there. This is a system feature (missing from many current information systems) which we shall call (logically) *symmetric exploitation* of relations. Naturally, symmetry in performance is not to be expected.

To support symmetric exploitation of a single binary relation, two directed paths are needed. For a relation of degree *n*, the number of paths to be named and controlled is *n* factorial.

Again, if a relational view is adopted in which every *n-ary* relation ($n > 2$) has to be expressed by the user as a nested expression involving only binary relations (see Feldman's LEAP System [10], for example) then $2n - 1$ names have to be coined instead of only $n + 1$ with direct *n-ary* notation as described in Section 1.2. For example, the 4-ary relation *supply* of Figure 1, which entails 5 names in *n-ary* notation, would be represented in the form

P (supplier, Q (part, R (project, quantity)))

in nested binary notation and, thus, employ 7 names.

A further disadvantage of this kind of expression is its asymmetry. Although this asymmetry does not prohibit symmetric exploitation, it certainly makes some bases of interrogation very awkward for the user to

[127] Exploiting a relation includes query, update, and delete.

express (consider, for example, a query for those parts and quantities related to certain given projects via *Q* and *R*).

1.6. EXPRESSIBLE, NAMED, AND STORED RELATIONS

Associated with a data bank are two collections of relations: the *named set* and the *expressible set.* The named set is the collection of all those relations that the community of users can identify by means of a simple name (or identifier). A relation *R* acquires membership in the named set when a suitably authorized user declares *R*; it loses membership when a suitably authorized user cancels the declaration of *R***.**

The expressible set is the total collection of relations that can be designated by expressions in the data language. Such expressions are constructed from simple names of relations in the named set; names of generations, roles and domains; logical connectives; the quantifiers of the predicate calculus;[128] and certain constant relation symbols such as =, >. The named set is a subset of the expressible set - usually a very small subset.

Since some relations in the named set may be time-independent combinations of others in that set, it is useful to consider associating with the named set a collection of statements that define these time-independent constraints. We shall postpone further discussion of this until we have introduced several operations on relations (see Section 2).

One of the major problems confronting the designer of a data system which is to support a relational model for its users is that of determining the class of stored representations to be supported. Ideally, the variety of

[128] Because each relation in a practical data bank is a finite set at every instant of time, the existential and universal quantifiers can be expressed in terms of a function that counts the number of elements in any finite set.

permitted data representations should be just adequate to cover the spectrum of performance requirements of the total collection of installations. Too great a variety leads to unnecessary overhead in storage and continual reinterpretation of descriptions for the structures currently in effect.

For any selected class of stored representations the data system must provide a means of translating user requests expressed in the data language of the relational model into corresponding - and efficient - actions on the current stored representation. For a high level data language this presents a challenging design problem. Nevertheless, it is a problem which must be solved - as more users obtain concurrent access to a large data bank, responsibility for providing efficient response and throughput shifts from the individual user to the data system.

2. Redundancy and Consistency

2.1. OPERATIONS ON RELATIONS

Since relations are sets, all of the usual set operations are applicable to them. Nevertheless, the result may not be a relation; for example, the union of a binary relation and a ternary relation is not a relation.

The operations discussed below are specifically for relations. These operations are introduced because of their key role in deriving relations from other relations. Their principal application is in noninferential information systems - systems which do not provide logical inference services - although their applicability is not necessarily destroyed when such services are added.

Most users would not be directly concerned with these operations. Information systems designers and people concerned with data bank control should, however, be thoroughly familiar with them.

2.1.1. Permutation.

A binary relation has an array representation with two columns. Interchanging these columns yields the converse relation. More generally, if a permutation is applied to the columns of an *n-ary* relation, the resulting relation is said to be a *permutation* of the given relation. There are, for example, 4! = 24 permutations of the relation *supply in* Figure 1, if we include the identity permutation which leaves the ordering of columns unchanged.

Since the user's relational model consists of a collection of relationships (domain-unordered relations), permutation is not relevant to such a model considered in isolation. It is, however, relevant to the consideration of stored representations of the model. In a system which provides symmetric exploitation of relations, the set of queries answerable by a stored relation is identical to the set answerable by any permutation of that relation. Although it is logically unnecessary to store both a relation and some permutation of it, performance considerations could make it advisable.

2.1.2. Projection.

Suppose now we select certain columns of a relation (striking out the others) and then remove from the resulting array any duplication in the rows. The final array represents a relation which is said to be a *projection* of the given relation.

A selection operator π is used to obtain any desired permutation, projection, or combination of the two operations. Thus, if *L* is a list of *k*

indices[129] $L = i_1, i_2, \ldots, i_k$ and R is an *n-ary* relation ($n \geq k$), then $\Pi_L(R)$ is the *k-ary* relation whose j_{th} column is column i_j of R ($j = 1, 2, \ldots, k$) except that duplication in resulting rows is removed. Consider the relation *supply* of Figure 1. A permuted projection of this relation is exhibited in Figure 4. Note that, in this particular case, the projection has fewer *n-tuples* than the relation from which it is derived.

2.1.3. Join.

Suppose we are given two binary relations, which have some domain in common. Under what circumstances can we combine these relations to form a ternary relation which preserves all of the information in the given relations?

The example in Figure 5 shows two relations *R*, S, which are joinable without loss of information, while Figure 6 shows a join of *R* with *S*. A binary relation *R* is *joinable* with a binary relation *S* if there exists a ternary relation *U* such that $\Pi_{12}(U) = R$ and $\Pi_{23}(U) = S$. Any such ternary relation is called a *join* of *R* with *S*. If *R*, *S* are binary relations such that $\Pi_2(R) = \Pi_1(S)$, then *R* is joinable with *S*. One join that always exists in such a case is the *natural join* of R with S defined by

R*S = {(a,b,c):R(a,b) ^ S(b,c)}

where *R* (*a, b*) has the value *true* if (*a, b*) is a member of *R* and similarly for *S* (*b, c*). It is immediate that

$\Pi_{12}(U) = R$

and

$\Pi_{23}(U) = S$

[129] When dealing with relationships, we use domain names (role-qualified whenever necessary) instead of domain positions.

Note that the join shown in Figure 6 is the natural join of *R* with *S* from Figure 5. Another join is shown in Figure 7.

π_{31}(supply) (project supplier)

5	1
5	2
1	4
7	2

Fig. 4. A permuted projection of the relation in Figure 1

R (supplier part) S (part project)

1	1	1	1
2	1	1	2
2	2	2	1

FIG. 5. Two joinable relations

R*S (supplier part project)

1	1	1
1	1	2
2	1	1
2	1	2
2	2	1

FIG. 6. The natural join of *R*, with *S* (from Figure 5)

U (supplier part project)

1	1	2
2	1	1

2 2 1

FIG. 7. Another join of R with S (from Figure 5)

Inspection of these relations reveals an element (element 1) of the domain *part* (the domain on which the join is to be made) with the property that it possesses more than one relative under *R* and also under *S*. It is this element which gives rise to the plurality of joins. Such an element in the joining domain is called a *point of ambiguity* with respect to the joining of *R* with *S*.

If either π_{21} (*R*) or S is a function,[130] no point of ambiguity can occur in joining *R* with *S*. In such a case, the natural join of *R* with *S* is the only join of *R* with *S*. Note that the reiterated qualification "of *R* with *S*" is necessary, because *S* might be joinable with *R* (as well as *R* with *S*), and this join would be an entirely separate consideration. In Figure 5, none of the relations *R*, $\pi_{21}(R)$, *S*, π_{21} (*S*) is a function.

Ambiguity in the joining of *R* with *S* can sometimes be resolved by means of other relations. Suppose we are given, or can derive from sources independent of *R* and *S*, a relation *T* on the domains *project* and *supplier* with the following properties:

(1) $\pi_1(T) = \pi_2(S)$,

(2) $\pi_2(T) = \pi_1(R)$,

(3) $T(j, s) \rightarrow \exists p(R(S, p) \wedge S(p , j))$,

(4) $R(s, p) \rightarrow \exists j(S(p,j) \wedge T(j, s))$,

(5) $S(p,j) \rightarrow \exists s(T(j, s) \wedge R(s, p))$,

then we may form a three-way join of *R*, *S*, *T*; that is, a ternary relation such that

[130] A function is a binary relation, which is one-one or many-one, but not one-many.

$\pi_{12}(U) = R$, $\pi_{23}(U) = S$, $\pi_{31}(U) = T$.

Such a join will be called a *cyclic 3-join* to distinguish it from a *linear 3-join* which would be a quaternary relation *V* such that

$\pi_{12}(V) = R$, $\pi_{23}(V) = S$, $\pi_{34}(V) = T$.

While it is possible for more than one cyclic 3-join to exist (see Figures 8, 9, for an example), the circumstances under which this can occur entail much more severe constraints

R (*s*	*p*)	*S* (*p*	*j*)	*T*(*j*	*s*)
l	a	a	d	d	l
2	a	a	e	d	2
2	b	b	d	e	2
B	e	e	2		

FIG. 8. Binary relations with a plurality of cyclic 3-joins

U (*s*	*p*	*j*)	U' (*s*	*p*	*j*)
l	a	d	l	a	d
2	a	e	2	a	d
2	b	d	2	a	e
2	b	e	2	b	d
2	b	e			

FIG. 9. Two cyclic 3-joins of the relations in Figure 8

than those for a plurality of 2-joins. To be specific, the relations *R, S, T* must possess points of ambiguity with respect to joining *R* with *S* (say point *x*), *S* with *T* (say *y*), and *T* with *R* (say *z),* and, furthermore, *y* must be a relative of *x* under *S, z* a relative of *y* under *T*, and *x* a relative of *z* under *R*. Note that in Figure 8 the points *x = a; y = d; z = 2* have this property.

The natural linear 3-join of three binary relations *R, S, T* is given by

R*S*T = {(a , b , c , d) : R (a , b) ^ S(b,c) ^ T(c,d)}

where parentheses are not needed on the left-hand side because the natural 2-join (*) is associative. To obtain the cyclic counterpart, we introduce the operator γ which produces a relation of degree *n* - 1 from a relation of degree *n* by tying its ends together. Thus, if *R* is an *n-ary* relation ($n \geq 2$), the *tie* of *R* is defined by the equation

$\gamma (R) = \{ (a_1, a_2, \ldots , a_{n-1}):R(a_1, a_2, \ldots , a_{n-1}, a_n)^ a_1 = a_n\}$.

We may now represent the natural cyclic 3-join of *R, S, T* by the expression

γ (R*S*T).

Extension of the notions of linear and cyclic 3-join and their natural counterparts to the joining of n binary relations (where $n \geq 3$) is obvious. A few words may be appropriate, however, regarding the joining of relations which are not necessarily binary. Consider the case of two relations *R* (degree *r*), *S* (degree *s*) which are to be joined on *p* of their domains ($p < r$, $p < s$). For simplicity, suppose these *p* domains are the last *p* of the *r* domains of *R*, and the first *p* of the *s* domains of *S*. If this were not so, we could always apply appropriate permutations to make it so. Now, take the Cartesian product of the first *r-p* domains of *R*, and call this new domain *A*. Take the Cartesian product of the last *p* domains of *R*, and call this *B*. Take the Cartesian product of the last *s-p* domains of *S* and call this *C*.

We can treat *R* as if it were a binary relation on the domains *A, B*. Similarly, we can treat *S* as if it were a binary relation on the domains *B, C*. The notions of linear and cyclic 3-join are now directly applicable. A similar approach can be taken with the linear and cyclic *n*-joins of *n* relations of assorted degrees.

2.1.4. Composition.

The reader is probably familiar with the notion of composition applied to functions. We shall discuss a generalization of that concept and apply it first to binary relations. Our definitions of composition and composability are based very directly on the definitions of join and joinability given above.

Suppose we are given two relations *R, S*. T is a *composition* of *R* with *S* if there exists a join *U* of *R* with *S* such that $T = \pi_{13}(U)$. Thus, two relations are composable if and only if they are joinable. However, the existence of more than one join of *R* with *S* does not imply the existence of more than one composition of *R* with *S*.

Corresponding to the natural join of R with S is the *natural composition*[131] of R with S defined by

$R \bullet S = \pi_{13}(R*S)$.

Taking the relations *R, S* from Figure 5, their natural composition is exhibited in Figure 10 and another composition is exhibited in Figure 11 (derived from the join exhibited in Figure 7).

R•S	(project	supplier)
	1	1
	1	2
	2	1
	2	2

Fig. 10. The natural composition of *R* with *S* (from Figure 5)

T	(project	supplier)

[131] Other writers tend to ignore compositions other than the natural one, and accordingly refer to this particular composition as the composition - see, for example, Kelley's "General Topology."

1	2
2	1

Fig. 11. Another composition of *R* with *S* (from Figure 5)

When two or more joins exist, the number of distinct compositions may be as few as one or as many as the number of distinct joins. Figure 12 shows an example of two relations which have several joins but only one composition. Note that the ambiguity of point *c* is lost in composing *R* with *S*, because of unambiguous associations made via the points *a, b, d, e*.

R (supplier part)		S (part project)	
1	a	a	g
1	b	b	f
1	c	c	f
2	c	c	g
2	d	d	g
2	e	e	f

Fig. 12. Many joins, only one composition

Extension of composition to pairs of relations which are not necessarily binary (and which may be of different degrees) follows the same pattern as extension of pairwise joining to such relations.

A lack of understanding of relational composition has led several systems designers into what may be called the *connection trap.* This trap may be described in terms of the following example. Suppose each supplier description is linked by pointers to the descriptions of each part supplied by that supplier, and each part description is similarly linked to the descriptions of each project which uses that part. A conclusion is now drawn which is, in general, erroneous: namely that, if all possible paths are followed from a

given supplier via the parts he supplies to the projects using those parts, one will obtain a valid set of all projects supplied by that supplier. Such a conclusion is correct only in the very special case that the target relation between projects and suppliers is, in fact, the natural composition of the other two relations - and we must normally add the phrase "for all time," because this is usually implied in claims concerning path-following techniques.

2.1.5. Restriction.

A subset of a relation is a relation. One way in which a relation *S* may act on a relation *R* to generate a subset of *R* is through the operation *restriction* of *R* by *S*. This operation is a generalization of the restriction of a function to a subset of its domain, and is defined as follows.

Let *L*, *M* be equal-length lists of indices such that $L = i_1,i_2,\ldots, i_k$, $M = j_1,j_2,\ldots, i_k$ where $k \leq$ degree of *R* and $k \leq$ degree of *S*. Then the *L, M* restriction of *R* by *S* denoted ${}_R L|_M S$ is the maximal subset *R′* of *R* such that

$\pi_L (R') = \pi_M (S)$.

The operation is defined only if equality is applicable between elements of $\pi_{ih} (R)$ on the one hand and $\pi_{jh} (S)$ on the other for all h = 1, 2, …, *k*.

The three relations *R, S, R'* of Figure 13 satisfy the equation $R' = R_{(2,3)|(1,2)}S$.

R (s p j)	S (p j)	R' (s p j)
1 a A	a A	1 a A
2 a A	c B	2 a A
2 a B	b B	2 b B
2 b A		
2 b B		

FIG. 13. Example of restriction

We are now in a position to consider various applications of these operations on relations.

2.2. REDUNDANCY

Redundancy in the named set of relations must be distinguished from redundancy in the stored set of representations. We are primarily concerned here with the former. To begin with, we need a precise notion of derivability for relations.

Suppose θ is a collection of operations on relations and each operation has the property that from its operands it yields a unique relation (thus natural join is eligible, but join is not). A relation *R* is θ-*derivable* from a set *S* of relations if there exists a sequence of operations from the collection θ which, for all time, yields *R* from members of *S*. The phrase "for all time" is present, because we are dealing with time-varying relations, and our interest is in derivability which holds over a significant period of time. For the named set of relationships in noninferential systems, it appears that an adequate collection θ_1 contains the following operations: projection, natural join, tie, and restriction. Permutation is irrelevant and natural composition need not be included, because it is obtainable by taking a natural join and then a projection. For the stored set of representations, an adequate collection θ_2 of operations would include permutation and additional operations concerned with sub-setting and merging relations, and ordering and connecting their elements.

2.2.1. Strong Redundancy.

A set of relations is *strongly redundant* if it contains at least one relation that possesses a projection which is derivable from other projections of

relations in the set. The following two examples are intended to explain why strong redundancy is defined this way, and to demonstrate its practical use. In the first example the collection of relations consists of just the following relation:

employee (serial #, name, manager#, managername)

with *serial#* as the primary key and *manager#* as a foreign key. Let us denote the active domain by Δ and suppose that

Δ_t (manager#) $\subset$ Δ_t (serial#)

and

Δ_t (managername) $\subset$ Δ_t (name)

for all time *t*. In this case the redundancy is obvious: the domain *managername* is unnecessary. To see that it is a strong redundancy as defined above, we observe that

π_{34} (employee) = π_{12} (employee) $_1|_1$ π_3 (employee).

In the second example the collection of relations includes a relation *S* describing suppliers with primary key *s#*, a relation *D* describing departments with primary key *d#*, a relation *J* describing projects with primary key *j#*, and the following relations:

P (s#, d#, . . .), Q (s#, j#, .-.), R (d#, j#, . . .),

where in each case ... denotes domains other than *s#, d#, j#*. Let us suppose the following condition *C* is known to hold independent of time: supplier *s* supplies department *d* (relation *P*) if and only if supplier *s* supplies some project *j* (relation *Q*) to which *d* is assigned (relation *R*). Then, we can write the equation

π_{12} (P) = π_{12} (Q)• π_{21} (R)

and thereby exhibit a strong redundancy.

An important reason for the existence of strong redundancies in the named set of relationships is user convenience. A particular case of this is the retention of semi-obsolete relationships in the named set so that old programs that refer to them by name can continue to run correctly. Knowledge of the existence of strong redundancies in the named set enables a system or data base administrator greater freedom in the selection of stored representations to cope more efficiently with current traffic. If the strong redundancies in the named set are directly reflected in strong redundancies in the stored set (or if other strong redundancies are introduced into the stored set), then, generally speaking, extra storage space and update time are consumed with a potential drop in query time for some queries and in load on the central processing units.

2.2.2. Weak Redundancy.

A second type of redundancy may exist. In contrast to strong redundancy it is not characterized by an equation. A collection of relations is *weakly redundant* if it contains a relation that has a projection which is not derivable from other members but is at all times a projection of *some* join of other projections of relations in the collection.

We can exhibit a weak redundancy by taking the second example (cited above) for a strong redundancy, and assuming now that condition *C* does not hold at all times.

The relations π_{12} (*P*), π_{12} (*Q*), π_{12} (*R*) are complex[132] relations with the possibility of points of ambiguity occurring from time to time in the potential joining of any two. Under these circumstances, none of them is derivable from the other two. However, constraints do exist between them, since

[132] A binary relation is complex if neither it nor its converse is a function.

each is a projection of some cyclic join of the three of them. One of the weak redundancies can be characterized by the statement: for all time, $\pi_{12}(P)$ is *some* composition of $\pi_{12}(Q)$ with $\pi_{12}(R)$. The composition in question might be the natural one at some instant and a nonnatural one at another instant.

Generally speaking, weak redundancies are inherent in the logical needs of the community of users. They are not removable by the system or data base administrator. If they appear at all, they appear in both the named set and the stored set of representations.

2.3. CONSISTENCY

Whenever the named set of relations is redundant in either sense, we shall associate with that set a collection of statements which define all of the redundancies which hold independent of time between the member relations. If the information system lacks - and it most probably will – detailed semantic information about each named relation, it cannot deduce the redundancies applicable to the named set. It might, over a period of time, make attempts to induce the redundancies, but such attempts would be fallible.

Given a collection *C* of time-varying relations, an associated set *Z* of constraint statements and an instantaneous value *V* for *C*, we shall call the state *(C, Z, V) consistent* or *inconsistent* according as *V* does or does not satisfy *Z*. For example, given stored relations *R, S, T* together with the constraint statement "π_{12} *(T)* is a composition of π_{12}(R) with π_{12}(S)", we may check from time to time that the values stored for *R, S, T* satisfy this constraint. An algorithm for making this check would examine the first two

columns of each of *R, S, T* (in whatever way they are represented in the system) and determine whether

(1) $\pi_1(T) = \pi_1(R)$,

(2) $\pi_2(T) = \pi_2(S)$,

(3) for every element pair (*a, c*) in the relation $\pi_{12}(T)$ there is an element *b* such that (*a, b*) is in $\pi_{12}(R)$ and (*b, c*) is in $\pi_{12}(S)$.

There are practical problems (which we shall not discuss here) in taking an instantaneous snapshot of a collection of relations, some of which may be very large and highly variable.

It is important to note that consistency as defined above is a property of the instantaneous state of a data bank, and is independent of how that state came about. Thus, in particular, there is no distinction made on the basis of whether a user generated an inconsistency due to an act of omission or an act of commission. Examination of a simple example will show the reasonableness of this (possibly unconventional) approach to consistency.

Suppose the named set *C* includes the relations *S, J, D, P, Q, R* of the example in Section 2.2 and that *P, Q, R* possess either the strong or weak redundancies described therein (in the particular case now under consideration, it does not matter which kind of redundancy occurs). Further, suppose that at some time *t* the data bank state is consistent and contains no project *j* such that supplier 2 supplies project *j* and *j* is assigned to department 5. Accordingly, there is no element (2, 5) in π_{12}(P). Now, a user introduces the element (2, 5) into π_{12}(P) by inserting some appropriate element into *P*. The data bank state is now inconsistent. The inconsistency could have arisen from an act of omission, if the input (2, 5) is correct, and

there does exist a project *j* such that supplier 2 supplies *j* and *j* is assigned to department 5. In this case, it is very likely that the user intends in the near future to insert elements into *Q* and *R* which will have the effect of introducing (2, *j*) into $\pi_{12}(Q)$ and (5, *j*) in $\pi_{12}(R)$. On the other hand, the input (2, 5) might have been faulty. It could be the case that the user intended to insert some other element into *P* - an element whose insertion would transform a consistent state into a consistent state. The point is that the system will normally have no way of resolving this question without interrogating its environment (perhaps the user who created the inconsistency).

There are, of course, several possible ways in which a system can detect inconsistencies and respond to them. In one approach the system checks for possible inconsistency whenever an insertion, deletion, or key update occurs. Naturally, such checking will slow these operations down. If an inconsistency has been generated, details are logged internally, and if it is not remedied within some reasonable time interval, either the user or someone responsible for the security and integrity of the data is notified. Another approach is to conduct consistency checking as a batch operation once a day or less frequently. Inputs causing the inconsistencies which remain in the data bank state at checking time can be tracked down if the system maintains a journal of all state-changing transactions. This latter approach would certainly be superior if few non-transitory inconsistencies occurred.

2.4. SUMMARY

In Section 1 a relational model of data is proposed as a basis for protecting users of formatted data systems from the potentially disruptive

changes in data representation caused by growth in the data bank and changes in traffic. A normal form for the time-varying collection of relationships is introduced.

In Section 2 operations on relations and two types of redundancy are defined and applied to the problem of maintaining the data in a consistent state. This is bound to become a serious practical problem as more and more different types of data are integrated together into common data banks.

Many questions are raised and left unanswered. For example, only a few of the more important properties of the data sublanguage in Section 1.4 are mentioned. Neither the purely linguistic details of such a language nor the implementation problems are discussed. Nevertheless, the material presented should be adequate for experienced systems programmers to visualize several approaches. It is also hoped that this paper can contribute to greater precision in work on formatted data systems.

Acknowledgment. It was C. T. Davies of IBM Poughkeepsie who convinced the author of the need for data independence in future information systems. The author wishes to thank him and also F. P. Palermo, C. P. Wang, E. B. Altman, and M. E. Senko of the IBM San Jose Research Laboratory for helpful discussions.

RECEIVED SEPTEMBER, 1969; REVISED FEBRUARY, 1970

REFERENCES

1. CHILDS, D.L. Feasibility of a set-theoretical data structure --a general structure based on a reconstituted definition of relation. Proc. IFIP Cong., 1968, North Holland Pub. Co., Amsterdam, p. 162-172.

2. LEVEIN, R. E., AND MARON, M. E. A computer system for inference execution and data retrieval. Comm. ACM 10, 11 (Nov. 1967), 715--721.

3. BACHMAN, C. W. Software for random access processing. Datamation (Apr. 1965), 36-41.

4. McGEE, W. C. Generalized file processing. In Annual Review in Automatic Programming 5, 13, Pergamon Press, New York, 1969, pp. 77-149.

5. Information Management System/360, Application Description Manual H20-0524-1. IBM Corp., White Plains, N. Y., July 1968.

6. GIS (Generalized Information System), Application Description Manual H20-0574. IBM Corp., White Plains, N. Y., 1965.

7. BLEIER, R.E. Treating hierarchical data structures in the SDC time-shared data management system (TDMS). Proc. ACM 22nd Nat. Conf., 1967, MDI Publications, Wayne, Pa., pp. 41---49.

8. IDS Reference Manual GE 625/635, GE Inform. Sys. Div., Pheonix, Ariz., CPB 1093B, Feb. 1968.

9. CHURCH, A. An Introduction to Mathematical Logic I. Princeton U. Press, Princeton, N.J., 1956.

10. FELDMAN, J. A., AND ROVNER, P.D. An Algol-based associative language. Stanford Artificial Intelligence Rep. AI-66, Aug. 1, 1968.

Versão para o português

Nossa experiência demonstra que a tradução de determinados termos, no interior dos textos, tende a causar mais confusão que benefício. Costumam ser palavras cuja forma em língua inglesa adquire um determinado significado, quando utilizado no âmbito dos *SGBDRs* especificamente, assim como genericamente nos textos que tratam de aspectos da "era da informação digital"[133]. Tais termos foram mantidos então, como no original, destacados de forma sublinhada, para se distinguir dos termos em *itálico* originalmente assim grafados por Codd. Outros termos, ainda que traduzidos, foram assim assinalados (de forma sublinhada) por entendermos que merecem maior esclarecimento sobre seu significado, ou simplesmente constituir uma ideia a destacar. Reiteramos ter investigado não o significado contemporâneo, mas aquele que acreditamos ser coerente com o entendimento de Codd, ou melhor, os termos que localizamos nas fontes mais próximas possíveis à época em que Codd escreveu. Observe-se que Codd utiliza duas notações distintas de referencia: [N] para a "N" referência bibliográfica elencada no final do artigo e [M] para o "M" comentário ao seu próprio texto, o que aparece então como nota de rodapé. Nestas, aparecem as indicações [Codd], para indicar uma nota do autor do artigo, e [nossa] para evidenciar alguma intervenção que julgamos apropriada.

Comunicações da ACM 377-87 Volume 13 Número 6/junho de 1970

Recuperação de Informação - P. Baxendale, editor

Modelo Relacional de Dados para Grandes Bancos de Dados Compartilhados

---

[133] Ceruzzi, Computing: a concise history, xvi.

E. F. CODD

Laboratório de Pesquisa da IBM em San José, Califórnia

Os futuros usuários de grandes bancos de dados devem ser protegidos de ter que saber como os dados são organizados na máquina (a representação interna). Um serviço de ajuda que forneça tal informação não é uma solução satisfatória. Atividades de usuários nos terminais assim como maioria dos programas de aplicação não deveriam ser afetadas quando a representação interna de dados é alterada e mesmo quando alguns aspectos da representação externa são alterados. Mudanças na representação de dados, muitas vezes, são necessárias, como resultado de mudanças na consulta, atualização e relatório de tráfego e crescimento natural nos tipos de informações armazenadas.

Sistemas de dados formatados, não inferenciais, existentes fornecem aos usuários arquivos com estrutura de dados em árvore ou um pouco mais gerais, modelos em rede destes. Na Seção 1, são discutidas as insuficiências destes modelos. Um modelo baseado em relações *n-ary*, uma forma normal para as relações de base de dados, bem como o conceito de um subidioma de dados universal são introduzidos. Na Seção 2, certas operações sobre as relações (exceto inferência lógica) são discutidos e aplicados aos problemas de redundância e de coerência em modelos de [dados de] usuário.

PALAVRAS E FRASES CHAVE: banco de dados, base de dados, estrutura de dados, organização de dados, hierarquias de dados, redes de dados, relações, derivabilidade, redundância, consistência, composição, *join*, linguagem de recuperação, cálculo de predicados, segurança, integridade de dados.

CATEGORIAS *CR*: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29.

1. Modelo Relacional e Forma Normal

1.1. INTRODUÇÃO

Este artigo está preocupado com a aplicação da teoria elementar de relações aos sistemas que permitem o acesso compartilhado a grandes bancos de dados formatados. Com exceção de um artigo de Childs [1], a principal aplicação de relações a sistemas de dados tem sido para sistemas dedutivos de perguntas e respostas. Levein e Maron [2] oferecem inúmeras referências aos trabalhos nesta área.

Em contraste, os problemas tratados aqui são aqueles [relativos a] *independência de dados* - a independência dos programas de aplicação e atividades em terminais desde crescimento em tipos de dados e mudanças na representação dos dados - e certos tipos de *inconsistência de dados* que deverão tornar-se problemático, mesmo em sistemas nondeductive [não dedutivos].

A visão (ou modelo) relacional de dados descrita na Seção 1 parece ser superior em vários aspectos, contraposta ao modelo gráfico ou de rede [3, 4] atualmente em voga para sistemas noninferential. Ele fornece um meio de descrever os dados com a sua estrutura natural apenas - ou seja, sem sobreposição de qualquer estrutura adicional para efeito de representação destes na máquina. Assim, ele fornece uma base para uma linguagem de dados de alto nível que irá produzir independência máxima entre programas de um lado e a representação e organização dos dados na máquina por outro.

Uma outra vantagem da visão relacional é que ele constitui uma base sólida para o tratamento de derivabilidade, redundância e consistência das

relações - estes são discutidos na Seção 2. O modelo de rede, por outro lado, tem gerado uma série de confusões, não menos do que está confundindo a derivação de conexões para a derivação de relações (ver observações na Seção 2 sobre a "armadilha da conexão").

Finalmente, a visão relacional permite uma avaliação mais clara do alcance e limitações lógicas de sistemas de dados formatados presentes [conforme se apresentam no momento], e também os méritos relativos (de um ponto de vista lógico) de representações concorrentes de dados dentro de um único sistema. Exemplos dessa perspectiva mais clara são citados em várias partes deste documento. Implementações de sistemas para apoiar o modelo relacional não são discutidas.

1.2. DEPENDÊNCIAS DE DADOS EM SISTEMAS PRESENTES [ATUAIS]

O fornecimento de tabelas de descrição de dados em sistemas de informação desenvolvidos recentemente representa um grande avanço em direção à meta de independência de dados [5, 6, 7]. Tais tabelas facilitam a mudança de certas características da representação dos dados armazenados num banco de dados. No entanto, a variedade de características de representação de dados que pode ser alterado *sem prejudicar logicamente alguns programas de aplicação* ainda é bastante limitada. Além disso, o modelo de dados com o qual os usuários interagem ainda está cheio de propriedades representacionais, particularmente no que diz respeito à representação de conjuntos de dados (em oposição a itens individuais). Três dos principais tipos de dependências de dados que ainda precisam ser removidos são: dependência de ordenação, dependência de indexação e dependência caminho de acesso. Em alguns sistemas essas dependências não são claramente separáveis uma da outra.

1.2.1. Dependência de ordenação.

Elementos de dados em um banco de dados podem ser armazenados em uma variedade de maneiras, algumas envolvendo nenhuma preocupação com ordenação, algumas permitindo a cada elemento participar em somente uma [sequencia de] ordenação, outras permitindo a cada elemento participar em várias [sequencias de] ordenações. Vamos considerar aqueles sistemas existentes que tanto exijam quanto permitam que elementos de dados sejam armazenados em pelo menos uma [sequencia de] ordenação total a qual esteja intimamente associada com a ordenação dos endereços determinados pelo hardware. Por exemplo, os registros [records] de um arquivo sobre peças podem ser armazenados em ordem crescente por número de série da peça. Tais sistemas normalmente permitem que programas aplicativos suponham que a ordem de apresentação dos registros de tal arquivo é idêntico a (ou é uma sub-ordenação da) ordenação armazenada. Esses programas aplicativos que aproveitam a ordenação de um arquivo armazenado irão provavelmente não funcionar corretamente se por algum motivo tornar-se necessário substituir essa ordenação por uma diferente. Observações similares são válidas para [dados] armazenados [com] uma ordenação implementada por meio de ponteiros.

Não é necessário destacar qualquer outro sistema como um exemplo, porque todos os sistemas de informação bem conhecidos que são comercializados hoje [1969] não conseguem fazer uma distinção clara entre a ordem de apresentação, de um lado e ordenação armazenadas por outro. Problemas de implementação significativos devem ser resolvidos para fornecer este tipo de independência.

1.2.2. Dependência de indexação.

No contexto de dados formatados, um índice é geralmente considerado como um componente da representação de dados puramente orientada para o desempenho. Ele tende a melhorar a resposta às consultas e atualizações e, ao mesmo tempo, diminuir a resposta para inserções e deleções. Do ponto de vista informativo, um índice é um componente redundante da representação de dados. Se um sistema utiliza índices em tudo e se for para um bom desempenho em um ambiente com a mudança dos padrões de atividade no banco de dados, a capacidade de criar e destruir índices ao longo do tempo, provavelmente será necessária. Surge então a questão: programas de aplicação e atividades [em] terminal podem permanecer estáveis conforme os índices vêm e vão?

Sistemas de dados formatados atuais tomam muito diferentes [diversas] abordagens para indexação. TDMS [7] fornece incondicionalmente indexação em todos os atributos. A versão atualmente comercializada do IMS [5] oferece ao usuário uma opção para cada arquivo: uma escolha entre não indexar nada (a organização sequencial hierárquica) ou indexação somente através da chave primária (a organização sequencial indexada hierárquica). Em nenhum dos casos a lógica da aplicação do usuário depende da existência dos índices fornecidos incondicionalmente. IDS [8], no entanto, permite que os projetistas dos arquivos selecionem atributos a serem indexados e incorporem índices na estrutura de arquivos por meio de cadeias adicionais. Os programas de aplicação que se aproveitam do benefício do desempenho dessas cadeias de indexação devem se referir a essas cadeias pelo nome. Tais programas não funcionarão corretamente se essas cadeias forem posteriormente removidas.

1.2.3. Dependência de Caminho de Acesso.

Muitos dos sistemas de dados formatados existentes fornecem aos usuários arquivos com estruturas um pouco mais gerais dos dados [em forma] de árvore ou [em forma de] modelos em rede. Os programas aplicativos desenvolvidos para trabalhar com esses sistemas tendem a ser prejudicados logicamente se as árvores ou redes forem alterados em sua estrutura. Segue um exemplo simples.

Suponha que o banco de dados contém informações sobre peças e projetos. Para cada peça, o número da peça, nome da peça, descrição peça, quantidade-a-mão, e a quantidade-em-pedido são gravadas. Para cada projeto, o número do projeto, nome do projeto [e] descrição do projeto são gravadas. Sempre que um projeto faz uso de uma determinada peça, a quantidade de peças comprometidas com o projeto dado também é gravada. Suponha que o sistema exige que o usuário ou projetista de arquivo declare ou defina os dados em termos de estruturas em árvore. Então, qualquer uma das estruturas hierárquicas podem ser adotadas para a informação mencionada acima (ver Estruturas 1-5).

Estrutura 1. Projetos subordinados as Peças

Arquivo	Segmento	Campos
F	PEÇA	#peça
		nome-da-peça
		descrição-peça
		quantidade-a-mão
		quantidade-em-pedido
	PROJETO	#projeto
		nome-projeto

descrição-projeto

quantidade-peças-comprometidas

Estrutura 2. Peças subordinados aos Projetos

Arquivo	Segmento	Campos
F	PROJETO	#projeto
		nome-projeto
		descrição-projeto
	PEÇA	#peça
		nome-da-peça
		descrição-peça
		quantidade-a-mão
		quantidade-em-pedido
		quantidade-peças-comprometidas

Estrutura 3. Peças e Projetos como Pares Comprometidos em Relacionamento Subordinado a Projetos

Arquivo	Segmento	Campos
F	PEÇA	#peça
		nome-da-peça
		descrição-peça
		quantidade-a-mão
		quantidade-em-pedido
G	PROJETO	#projeto
		nome-projeto
		descrição-projeto

	PEÇA	#peça
		quantidade-peças-comprometidas

Estrutura 4. Peças e Projetos como Pares Comprometidos em Relacionamento Subordinado a Peças

Arquivo	Segmento	Campos
F	PEÇA	#peça
		[nome-da-peça][134]
		descrição-peça
		quantidade-a-mão
		quantidade-em-pedido
	PROJETO	#projeto
		quantidade-peças-comprometidas
G	PROJETO	#projeto
		nome-projeto
		descrição-projeto

Estrutura 5. Peças, Projetos e Relacionamento Comprometido como Pares

Arquivo	Segmento	Campos
F	PEÇA	#peça
		nome-da-peça
		descrição-peça
		quantidade-a-mão
		quantidade-em-pedido

[134] Em nenhuma versão encontrada do documento de Codd se encontra este campo. Acreditamos entretanto ser necessário incluí-lo para manter a coerência da ideia em discussão [nossa].

G	PROJETO	#projeto
		nome-projeto
		descrição-projeto
H	COMPROMETIMENTO	#peça
		#projeto
		quantidade-peças-comprometidas

Agora, considere o problema de imprimir o número da peça, nome da peça, quantidade e comprometida para cada peça utilizada no projeto cujo nome do projeto é "alfa". As seguintes observações podem ser feitas independentemente de qual sistema de informação orientado-em-árvore disponível é selecionado para resolver este problema. Se um programa P é desenvolvido para este problema assumindo uma das cinco estruturas acima - isto é, P não fizer qualquer teste para determinar qual a estrutura está em vigor - então P falhará em pelo menos três das estruturas restantes. Mais especificamente, se P tiver sucesso com a estrutura 5, ele falhará com todos os outros; se P tiver sucesso com estrutura 3 ou 4, ele falhará com pelo menos 1, 2 e 5; se P tiver sucesso com 1 ou 2, falhará com, pelo menos, 3, 4 e 5 . A razão é simples, em cada caso. Na ausência de um teste para determinar qual a estrutura está em vigor, P falha porque é feita uma tentativa de executar uma referência para um arquivo inexistente (sistemas disponíveis tratam isso como um erro) ou não é feita qualquer tentativa de executar uma referência para um arquivo contendo a informação necessária. O leitor que não está convencido deveria desenvolver programas de amostra para este problema simples.

Uma vez que, em geral, não é prático desenvolver programas de aplicação que testem todos as estruturas em árvore permitidas pelo

[possíveis ao] sistema, estes programas falharão quando uma mudança na estrutura tornar-se necessária.

Sistemas que fornecem aos usuários um modelo de dados em rede colidem com dificuldades semelhantes. Em ambos os casos, [dados organizados em forma] de árvore ou [em forma de] modelos em rede, o usuário (ou o seu programa) deve explorar uma coleção de caminhos de acesso do usuário aos dados. Não importa se esses caminhos estão em estreita correspondência com caminhos definidos pelo ponteiro na representação armazenada - em IDS a correspondência é extremamente simples, em TDMS é exatamente o oposto. A consequência, independentemente da representação armazenada, é que as atividades em terminal e programas se tornam dependentes da existência contínua [permanente] dos caminhos de acesso do usuário.

Uma solução para isso é a adoção da política de que uma vez que um caminho de acesso do usuário é definido, não será tornado obsoleto até que todos os programas aplicativos que usam esse caminho se tornem obsoletos. Tal política não é prática, porque o número total de caminhos de acesso no modelo para a comunidade de usuários de um banco de dados acabaria por se tornar excessivamente grande.

1.3. UMA VISÃO RELACIONAL DE DADOS

A termo *relação* é usado aqui no seu sentido matemático aceito. Dado conjuntos S_1, S_2, … , S_n (não necessariamente distintos), R é uma relação destes n conjuntos se é um conjunto de n-tuples cada uma das quais tendo o seu primeiro elemento a partir de S_1, o segundo elemento de S_2, e assim

por diante.[135] Referimo-nos a S_1, como o j_{th} domínio de R. Tal como acima definido, diz-se que *R* tem grau *n*. As relações de grau 1 são frequentemente chamados de *unárias*, grau 2 *binárias*, grau 3 *ternárias* e grau *n n-árias*.

Por razões expositivas, vamos frequentemente fazer uso de uma representação de relações em forma de array, mas é preciso lembrar que esta representação particular não é uma parte essencial da visão relacional que está sendo exposta. Um array que representa uma relação *R n-ary* [enária] tem as seguintes propriedades:

(1) Cada linha representa uma n-tupla de R.

(2) A ordem de linhas é imaterial.

(3) Todas as linhas são distintas.

(4) A ordem de colunas é importante - que corresponde a ordenação S_1, S_2, … , S_n dos domínios em que *R* é definido (ver, no entanto, observações abaixo sobre as relações domínio-ordenadas e de domínio-não-ordenadas).

(5) O significado de cada coluna é parcialmente transmitido através da marcação com o nome do domínio correspondente.

O exemplo da Figura 1 ilustra uma relação de grau 4, chamada de *abastecimento*, o que reflete os embarques em progresso de peças desde fornecedores específicos para projetos específicos em quantidades especificadas.

abastecimento (fornecedor	peça	projeto	quantidade)
1	2	5	17
1	3	5	23
2	3	7	9

[135] Mais concisamente, R é um subconjunto do produto Cartesiano S1 X S2 X … X Sn [Codd].

2	7	5	4
4	1	1	12

Fig. 1. Uma relação de grau 4

Alguém poderia perguntar: Se as colunas são rotuladas com o nome de domínios correspondentes, por que a ordenação das colunas importa? Conforme o exemplo da figura 2 mostra, duas colunas podem ter posições idênticas (indicando os domínios idênticos), mas possuem significados diferentes no que diz respeito à relação. A relação descrita é chamada *componente*. É uma relação ternária, cujo primeiro dois domínios são chamados *peça* e terceiro domínio é chamado *quantidade*. O significado do *componente* (*x, y, z*) é que a peça *x* é um componente imediato (ou subconjunto) da peça *y* e *z* unidades da peça *x* são necessárias para montar uma unidade da peça *y*. É uma relação que desempenha um papel crítico no problema de explosão de peças [explicação de como as peças são montadas].

Componente (peça	peça	quantidade)
1	5	9
2	5	7
3	5	2
2	6	12
3	6	3
4	7	1
6	7	1

Fig. 2. Uma relação com os dois domínios idênticos

É um fato notável que vários sistemas de informação existentes (principalmente aqueles baseados em arquivos de árvore-estruturadas) não

fornecem representações de dados para as relações que têm dois ou mais domínios idênticos. A versão atual da IMS/360 [5] é um exemplo de tal sistema.

A totalidade dos dados de um banco de dados pode ser visualizada como uma série de relações que variam no tempo. Essas relações são de graus variados. Conforme o tempo passa, cada relação n-ária pode estar sujeito a inserção de n-tuplas adicionais, exclusão dos já existentes, e alteração de componentes de qualquer de suas n-tuplas existentes.

Em muitos bancos de dados comerciais, governamentais e científicos, no entanto, algumas das relações são de muito alto grau (um grau de 30 não é de todo incomum). Os usuários não deveriam ser normalmente sobrecarregados com [a necessidade de] recordar a ordenação de domínios de qualquer relação (por exemplo, o *fornecedor* de pedidos, então *peça*, então *projeto*, então *quantidade* na relação *fornecimento*). Assim, propomos que os usuários lidem, não com as relações que são domínio-ordenadas, mas com *relacionamentos* que são suas contrapartidas domínio-não-ordenadas.[136] Para conseguir isso, os domínios devem ser unicamente identificáveis pelo menos dentro de qualquer relação dada , sem o uso de posição. Assim, quando há dois ou mais domínios idênticos, que requerem em cada caso, que o nome de domínio ser qualificado por um *nome de função* distintiva, que serve para identificar o papel desempenhado por esse domínio na relação dada. Por exemplo, na relação *componente* da Figura 2, o primeiro domínio *peça* pode ser qualificado pelo nome da [sua] função *sub*, e o segundo por *super*, de modo que os usuários possam lidar com o

[136] Em termos matemáticos, um relacionamento é uma classe de equivalência dessas relações que são equivalentes sob permutação de domínios (ver Seção 2.1.1) [Codd].

relacionamento *componente* e seus *domínios* - *sub.peça, super.peça, quantidade* - sem levar em conta qualquer ordenação entre esses domínios.

Em suma, propõe-se que a maioria dos usuários deveria interagir com um modelo relacional dos dados consistindo de uma coleção de relacionamentos variáveis no tempo (ao invés de relações). Cada usuário não precisa saber mais sobre qualquer relacionamento do que o seu nome juntamente com os nomes de seus domínios (papel qualificado sempre que necessário).[137] Mesmo esta informação pode ser oferecida em estilo de menu do sistema (sujeito a restrições de segurança e privacidade), a pedido do usuário.

Normalmente existem muitas formas alternativas em que um modelo relacional pode ser estabelecido para um banco de dados. A fim de discutir uma forma preferida (ou forma normal), é preciso primeiro apresentar alguns conceitos adicionais (de domínio ativo, chave primária, chave estrangeira, domínio não simples) e estabelecer alguns laços com a terminologia em uso atualmente na programação de sistemas de informação. No restante deste artigo, não se deu ao trabalho de distinguir entre relações e relacionamentos, exceto onde parece vantajoso ser explícito.

Considere um exemplo de um banco de dados que inclui as relações relativas a peças, projetos e fornecedores. Uma relação denominada *peças* é definida nos seguintes domínios:

(1) número da peça

(2) nome da peça

---

[137] Naturalmente, como com todos os dados inseridos e recuperados a partir de um sistema de computador, o usuário normalmente fazem uso muito mais eficiente dos dados, se ele tem consciência do seu significado [Codd].

(3) cor da peça

(4) peso da peça

(5) quantidade à mão

(6) quantidade em pedido

e possivelmente outros domínios do mesmo modo. Cada um destes domínios será, com efeito, um conjunto de valores, alguns ou todos dos quais podem ser representadas no banco de dados em qualquer instante. Embora seja possível que, em algum momento, todas as cores de peças estejam presentes, é improvável que todos os possíveis pesos de peças, nomes de peças e números de peças estejam. Vamos chamar o conjunto de valores representados em algum instante de *domínio ativo* naquele instante.

Normalmente, um domínio (ou a combinação de domínios) de uma dada relação tem valores que identificam cada elemento (n-tupla) daquela relação. Tal domínio (ou combinação) é chamado uma *chave primária*. No exemplo acima, número da peça seria uma *chave primária*, enquanto cor da peça não seria. Uma chave primária é *não redundante* se é um domínio simples (não uma combinação) ou uma combinação de tal forma que nenhum dos domínios simples participantes é supérfluo para identificar exclusivamente cada elemento. Uma relação pode possuir mais de uma chave primária não redundante. Este seria o caso, no exemplo, se diferentes peças foram sempre designadas com nomes diferentes. Sempre que uma relação tem duas ou mais chaves primárias não redundantes, uma delas é arbitrariamente selecionada e designada *a* chave primária dessa relação.

Um requisito comum é que os elementos de uma relação façam referência cruzada com outros elementos da mesma relação ou elementos de uma relação diferente. Chaves fornecem meios orientados para o usuário (mas não o único meio) de expressar tais referências cruzadas. Chamaremos um domínio (ou a combinação de domínios) da relação *R* uma *chave estrangeira*, se não for a *chave primária* de *R*, mas seus elementos são os valores da chave primária de uma relação *S* (a possibilidade de que *S* e *R* serem idênticos não é excluída). Na relação *abastecimento* da Figura 1, a combinação de *fornecedor, peça, projeto* é a chave primária, enquanto que cada um destes três domínios considerados separadamente é uma chave estrangeira.

Em trabalhos anteriores, tem havido uma forte tendência para tratar os dados de um banco de dados como consistindo em duas partes, uma parte composta por descrições de entidade (por exemplo, as descrições de fornecedores) e a outra parte consistindo de relações entre as diversas entidades ou tipos de entidades (por exemplo, a relação de *abastecimento*). Esta distinção é difícil de manter quando se pode ter chaves estrangeiras em qualquer relação. No modelo relacional do usuário, parece haver nenhuma vantagem em fazer tal distinção (pode haver alguma vantagem, no entanto, quando se aplicam os conceitos relacionais para representações de máquinas dos conjuntos de relacionamentos do usuário).

Até agora, discutimos exemplos de relações que são definidas em domínios simples - domínios cujos elementos são valores atômicos (não decomponíveis[138]). Valores não atômicos podem ser discutidos no âmbito relacional. Assim, alguns domínios podem ter relações como elementos.

[138] Ferreira, s.v. decomponível: Que se pode decompor. [nossa]

Estas relações podem, por sua vez, ser definidas em domínios não simples, e assim por diante. Por exemplo, um dos domínios em que a relação *empregado* é definida pode ser *histórico salarial*. Um elemento do domínio *histórico salarial* é uma relação binária definida no domínio *data* e do domínio *salário*. O domínio *histórico salarial* é o conjunto de todas essas relações binárias. A qualquer instante de tempo há tantas instâncias da relação *histórico salarial* no banco de dados quanto há empregados. Em contraste, há apenas um instância da relação *empregado*.

Os termos atributo e grupo de repetição estão presentes na terminologia de base de dados[139] [de forma] aproximadamente análoga a domínio simples e domínio não simples, respectivamente. Grande parte da confusão na terminologia atual é devida a incapacidade de distinguir entre tipo e instância (como em "record" [registro]) e entre os componentes de um modelo de usuário dos dados, por um lado e os sua contrapartida de [no nível da] representação da máquina, por outro (novamente, citamos "record" como um exemplo).

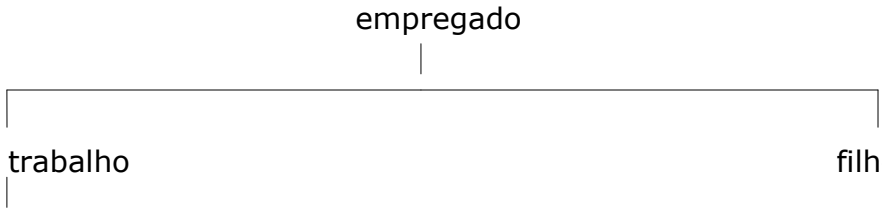
1.4. FORMA NORMAL

Uma relação cujos domínios são simples pode ser representada em [no nível do] armazenamento por uma matriz bidimensional coluna-homogênea, do tipo discutido acima. Algumas estruturas de dados mais complicadas são necessárias para uma relação com um ou mais domínios não simples. Por esta razão (e outras a serem citados abaixo) parece valer a pena investigar

[139] Não foi possível identificar algum significado ou motivo para aqui mudar de banco de dados para base de dados [nossa].

a possibilidade de eliminar domínios não simples.[140] Há, de fato, um processo de eliminação muito simples, que vamos chamar de normalização.

Considere-se, por exemplo, a coleção de relações exibida na Figura 3 (a). *Histórico de trabalho* e *filhos* são domínios não simples da relação *empregado*. *Histórico de salário* é um domínio não simples da relação *histórico de trabalho*. A árvore na Figura 3 (a) mostra apenas a estas inter-relações dos domínios não simples.



empregado (*#homen*, nome, data-de-nascimento, histórico-de-trabalho, filhos)

histórico-de-trabalho (*data-cargo*, título, histórico-de-salário)

histórico-de-salário (*data-do-salário*, salário)

filhos (*nome-do-filho*, ano-do-nascimento)

Fig. 3 (a). conjunto não normalizado

Empregado' (*#homen*, nome, data-de-nascimento)

histórico-de-trabalho' (*#homen, data-cargo*, título)

histórico-de-salário' (#homen , data-cargo , data-do-salário, salário)

filhos' (*#homen , nome-do-filho*, ano-do-nascimento)

Fig. 3 (b). conjunto normalizado

Procede-se a normalização como segue. Começando com a relação no topo da árvore, obter [determinar] sua chave primária e expandir cada uma

[140] ME Sanko da IBM em San Jose, reconheceu de forma independente a conveniência de eliminar domínios não simples [Codd].

das relações imediatamente subordinados inserindo esse domínio ou combinação de domínio da chave primária. A chave primária de cada relação expandida compreende à chave primária antes da expansão aumentada pela chave primária copiada para baixo a partir da relação pai. Agora, atacar a partir da relação pai todos os domínios não simples, remover o nó superior da árvore, e repetir a mesma sequência de operações em cada subárvore remanescente.

O resultado de normalizar a coleção de relações na Figura 3 (a) é a coleção da Figura 3 (b). A chave primária de cada relação está em itálico para mostrar como essas chaves são expandidas pela normalização.

Se normalização como descrito acima for aplicável, a coleção desnormalizada das relações deve satisfazer as seguintes condições:

(1) O gráfico de inter-relações dos domínios não simples é uma coleção de árvores.

(2) Nenhuma chave primária tem um componente de domínio que é não simples.

O autor não conhece nenhum aplicativo que exigiria qualquer relaxamento dessas condições. Operações adicionais de normalização são possíveis. Estas não são discutidas neste trabalho.

A simplicidade da representação matricial que se torna viável quando todas as relações são expressos em forma normal não é apenas uma vantagem para fins de armazenamento, mas também para a comunicação de dados em massa entre sistemas que utilizam amplamente diferentes representações dos dados. A forma de comunicação seria uma versão devidamente compactado da representação da matriz e teria as seguintes vantagens:

(1) Seria desprovido de ponteiros (endereçamento-por-valor ou endereçamento-por-deslocamento).

(2) Evitaria qualquer dependência de esquemas de endereçamento aleatórios.

(3) Não conteria índices ou listas de ordenação.

Se o modelo relacional do usuário é criado em forma normal, nomes de itens de dados no banco de dados podem ter uma forma mais simples do que seria em circunstâncias diferentes. Um nome geral tomaria uma forma como

R (g).r.d

onde *R* é um nome relacional [ou *nome da Relação*]; *g* é um identificador de geração [*versão*] (opcional); *r* é um nome de papel (opcional), *d* é um nome de domínio. Como *g* é necessária apenas quando várias gerações de uma dada relação existem, ou que estejam previstos para existir, e *r* é apenas necessário quando à relação *R* tem dois ou mais domínios denominados *d*, a forma simples *R.d* será frequentemente adequada [suficiente].

1.5. ALGUNS ASPECTOS LINGÜÍSTICOS

A adoção de um modelo de dados relacional, como descrito acima, permite o desenvolvimento de uma sublinguagem universal para dados, com base em um cálculo de predicados aplicado. Um cálculo de predicados de primeira ordem é suficiente se a coleção de relações está em forma normal. Tal linguagem forneceria um poder de comparação linguística para todas as outras linguagens de dados propostas, e em si seria um forte candidata para a incorporação (com modificação sintática for o caso) em uma variedade de linguagens hospedeiras (orientada à programação,

comando ou problema). Embora não seja o objetivo deste artigo descrever tal linguagem em detalhe, suas principais características seriam as seguintes.

Vamos denominar a sublinguagem de dados por *R* e a linguagem hospedeira por *H*. *R* permite a declaração de relações e de seus domínios. Cada declaração de uma relação identifica a chave primária para essa relação. Relações declaradas são adicionadas ao catálogo do sistema para ser utilizadas por todos os membros da comunidade de usuários que têm autorização apropriada. *H* permite apoiar as declarações que indica, talvez menos permanente, como essas relações são representados no armazenamento. *R* permite a especificação para recuperação de qualquer subconjunto de dados do banco de dados. Essa ação de pedido de recuperação [de dados] está sujeita a restrições de segurança.

A universalidade da sublinguagem de dados reside na sua capacidade descritiva (não na sua capacidade de computação). Em um grande banco de dados, cada subconjunto de dados tem um número muito grande de possíveis (e sensíveis) descrições, mesmo quando assumimos (como nós fazemos) que existe apenas um conjunto finito de funções de sub-rotinas que o sistema tem acesso para uso em qualificação de dados para recuperação. Assim, a classe de expressões de qualificação que pode ser usada na especificação de conjunto deve ter a capacidade descritiva da classe de fórmulas bem formadas de um cálculo de predicados aplicado. É bem conhecido que para preservar este poder descritivo, não é necessário expressar (em qualquer que seja a sintaxe escolhida) cada fórmula de

cálculo de predicados selecionado. Por exemplo, apenas aqueles na forma normal prenex[141] são adequados [9].

Funções aritméticas podem ser necessárias na qualificação ou em outras partes dos comandos de recuperação. Tais funções podem ser definidas em *H* e invocadas em *R*.

Um conjunto especificado desse modo pode ser obtido para fins de consulta apenas, ou pode ser mantido para possíveis mudanças. Inserções assumem a forma de adicionar novos elementos para as relações declaradas sem consideração qualquer ordem que podem estar presentes em sua representação na máquina [sem levar em consideração a ordem em que se encontram armazenados os registros]. Exclusões que são eficazes para a comunidade (ao contrário do usuário individual ou sub-comunidade) assumem a forma de remoção de elementos de relações declaradas. Algumas exclusões e atualizações podem ser desencadeadas por outros, se as dependências de exclusão e atualização entre as relações especificadas são declarados em *R*.

Um efeito importante que a visão adotada para dados tem sobre a linguagem usada para recuperá-lo é na nomeação de elementos de dados e conjuntos. Alguns aspectos desta foram discutidos na seção anterior. Com a visão de rede usual, os usuários, muitas vezes, são sobrecarregados com a cunhagem e uso de mais nomes de relações do que são absolutamente necessárias, uma vez que os nomes estão associados com caminhos (ou tipos de caminho) e não com as relações.

---

[141] Nicoletti, "Forma Normal Prenex", oferece uma visão da aplicação deste método de organização dos *depósitos de dados* [nossa].

Uma vez que o usuário está ciente de que uma certa relação é armazenada, ele vai esperar ser capaz de explorá-la[142] usando qualquer combinação de seus argumentos como "fatos conhecidos" e os argumentos restantes como "fatos não conhecidos", porque a informação (como o Everest) está lá. Esta é uma característica do sistema (inexistentes em muitos sistemas de informação atuais) que podemos chamar (logicamente) de *exploração simétrica* das relações. Naturalmente, *simetria* no desempenho não é prevista.

Para apoiar a exploração simétrica de uma única relação binária, são necessários dois caminhos direcionados. Para uma relação de grau *n*, o número de caminhos a serem nomeados e controlados é *n* fatorial.

Novamente, se uma visão relacional é adotada em que toda relação *n-ária* (*n*> 2) tem de ser expressa pelo usuário como uma expressão aninhada envolvendo apenas relações binárias (ver LEAP Sistema de Feldman [10], por exemplo), então 2*n* - 1 nomes têm de ser cunhados em vez de apenas *n* + 1 com a notação *n-ary* direta como descrito na Seção 1.2. Por exemplo, a relação 4-ária *abastecimento* da Figura 1, o que implica 5 nomes na notação *n-ary*, seria representada sob a forma

P (fornecedor, Q (peça, R (projeto, quantidade)))

em notação binária aninhada e, portanto, emprega 7 nomes.

Uma outra desvantagem deste tipo de expressão é a sua assimetria. Embora essa assimetria não proíba a exploração simétrica, ela certamente torna algumas bases de pesquisa muito estranho para o usuário expressar (considere, por exemplo, uma consulta para as peças e quantidades relacionadas com determinados projetos através de Q e R).

---

[142] Explorar uma relação inclui consulta, atualização e exclusão [Codd].

1.6. RELAÇÕES EXPRIMÍVEIS, NOMEADAS E ARMAZENADAS

Associado a um banco de dados existem duas coleções de relações: o *conjunto nomeado* e o *conjunto exprimível*. O conjunto nomeado é a coleção de todas aquelas relações que a comunidade de usuários pode identificar, por meio de um nome simples (ou identificador). A relação *R* adquire participação no conjunto nomeado quando um usuário devidamente autorizado declara *R*; perde adesão quando um usuário devidamente autorizado cancela a declaração de *R*.

O conjunto exprimível é a coleção total de relações que podem ser designadas por expressões na linguagem de dados. Tais expressões são construídas a partir de nomes simples de relações no conjunto nomeado; nomes de gerações, papéis e domínios; conectivos lógicos; os quantificadores do cálculo de predicados;[143] e certos símbolos constantes de relação, como =,>. O conjunto nomeado é um subconjunto do conjunto exprimível - geralmente um subconjunto muito pequeno.

Uma vez que algumas relações no conjunto nomeado podem ser combinações independentes do tempo de outros naquele conjunto, é útil considerar se associar com o nome definido um conjunto de instruções que definem essas restrições independentes do tempo. Vamos adiar uma discussão mais aprofundada deste até que tenhamos introduzido algumas operações sobre relações (ver Seção 2).

Um dos principais problemas com que se confrontam o projetista de um sistema de dados que suportará um modelo relacional para os seus usuários é determinar a classe de representações armazenadas a ser suportada. O

---

[143] Porque cada relação de um banco de dados prático é um conjunto finito em cada instante de tempo, os quantificadores existenciais e universais podem ser expressos em termos de uma função que conta o número de elementos em qualquer conjunto finito [Codd].

ideal é que a variedade de representações de dados permitidos deve ser apenas suficiente para cobrir o espectro de requisitos de desempenho da coleta total das instalações. Uma variedade muito grande leva à sobrecarga desnecessária no armazenamento e contínua reinterpretação de descrições para as estruturas atualmente em vigor.

Para qualquer classe selecionada de representações armazenadas o sistema de dados deve fornecer um meio de traduzir as solicitações do usuário expressas em linguagem de dados do modelo relacional em correspondentes - e eficientes - ações sobre a representação atual armazenada. Para uma linguagem de dados de alto nível isso apresenta um problema de projeto desafiador. No entanto, é um problema que deve ser resolvido [:] - conforme mais usuários obtêm acesso simultâneo a um grande banco de dados, a responsabilidade de dar uma resposta eficiente e mudanças de taxa de transferência do usuário individual para o sistema de dados.

2. Redundância e Consistência

2.1. OPERAÇÕES SUPORTADAS POR RELAÇÕES

Desde que as relações são conjuntos, todos os conjuntos de operações usuais são aplicáveis a elas. No entanto, o resultado pode não ser uma relação, por exemplo, a união de uma relação binária e uma relação ternária não é uma relação.

As operações analisadas em seguida são especificamente para as relações. Estas operações são introduzidas por causa de seu papel fundamental em derivar relações de outras relações. Sua aplicação principal é em sistemas de informação noninferential - sistemas que não prestam

serviços de inferência lógica - embora a sua aplicabilidade não seja necessariamente destruído quando tais serviços são adicionados.

A maioria dos usuários não seriam diretamente relacionado com essas operações. Projetistas de sistemas de informação e pessoas envolvidas com o controle de banco de dados deveriam, no entanto, estar totalmente familiarizados com eles.

2.1.1. Permutação.

Uma relação binária tem uma representação matricial com duas colunas. A troca destas colunas produz a relação recíproca. De modo mais geral, se uma permutação é aplicada para as colunas de uma relação *n-ary*, a relação resultante é denominada como sendo uma permutação da relação dada. Existem, por exemplo, 4! = 24 permutações da relação *abastecimento* na Figura 1, se incluirmos a permuta de identidade o que deixaria o ordenamento das colunas inalterada.

Como o modelo relacional do usuário consiste de uma coleção de relações (relações de domínio não-ordenado), permutação não é relevante para esse modelo considerado isoladamente. É, no entanto, pertinente para a consideração das representações armazenadas do modelo. Em um sistema que fornece exploração simétrica das relações, o conjunto de consultas respondíveis por uma relação armazenado é idêntico ao conjunto responsável por qualquer permutação dessa relação. Embora seja logicamente desnecessário para armazenar uma relação e alguns permutação dessa [relação], considerações de desempenho poderiam fazer isso aconselhável.

2.1.2. Projeção.

Suponha agora que selecionamos determinadas colunas de uma relação (ignorando as outras) e então removemos da matriz resultando qualquer duplicação nas linhas. A matriz final representa uma relação que se diz ser uma projeção da relação dada.

Um operador de seleção π é usado para obter qualquer permutação, projeção, ou combinação desejada das duas operações. Assim, se *L* é uma lista de *k* índices[144] $L = i_1, i_2, \ldots, i_k$ e *R* é uma relação *n-ary* ($n \geq k$), então $\pi_L(R)$ é a relação *k-ária* cuja coluna j_{th} é coluna i_j de *R* ($j = 1, 2, \ldots, k$), exceto que a duplicação em linhas resultantes for removida. Considere a relação *abastecimento* da Figura 1. Uma projeção permutada desta relação é exibido na Figura 4. Note-se que, neste caso particular, a projeção possui menos *n-tuples* do que a relação da qual é derivada.

2.1.3. Join.

Suponha que temos duas relações binárias, que têm algum domínio em comum. Em que circunstâncias é que podemos combinar essas relações para formar uma relação ternária que preserva toda a informação nas relações dadas?

O exemplo na Figura 5 mostra duas relações *R*, *S*, as quais são acopláveis [joinable, passíveis de serem compostas em *join*] sem perda de informações, enquanto que a Figura 6 mostra o *join* de *R* com *S*. Uma relação binária *R* é *joinable* [acoplável] com uma relação binária *S* se existe uma relação ternária *U* tal que $\pi_{12}(U) = R$ e $\pi_{23}(U) = S$. Qualquer relação de tal ternário é chamada de *join* de *R* com *S*. Se *R*, *S* são relações binárias em

[144] Ao lidar com relacionamentos, usamos nomes de domínio (qualificando seus papéis sempre que necessário) em vez de posições de domínio [Codd].

que π_2 (*R*) = π_1 (S), então *R* é *joinable* com *S*. Um *join* que sempre existe, neste caso, é o natural *join* de *R* com *S* definido por

R*S = {(a,b,c):R(a,b) ^ S(b,c)}

onde R (a, b) tem o valor *verdadeiro* se *(a, b)* é um membro de *R* e de forma semelhante *S* (*b, c*). É imediato que

$\pi_{12}(U) = R$

e

$\pi_{23}(U) = S$

Note-se que o *join* mostrado na Figura 6 é natural *join* de *R* com *S* a partir da Figura 5. Outro *join* é mostrado na Figura 7.

π_{31}(abastecimento) (projeto fornecedor)

5	1
5	2
1	4
7	2

Fig. 4. Uma projeção permutada desde a relação na Figura 1

R (fornecedor peça) S (peça projeto)

1	1	1	1
2	1	1	2
2	2	2	1

FIG. 5. Duas relações joinable

R*S (fornecedor peça projeto)

1	1	1

1	1	2
2	1	1
2	1	2
2	2	1

FIG. 6. O natural join de *R* com *S* (desde a Figura 5)

U (fornecedor peça projeto)

1	1	2
2	1	1
2	2	1

FIG. 7. Outro join de *R* com *S* (desde a Figura 5)

Uma inspeção destas relações revela um elemento (elemento 1) do domínio *peça* (domínio sobre o qual o join deve ser feito) com a propriedade que possui mais de um parente em *R* e também sob *S*. É este elemento que possibilita a pluralidade de joins. Um tal elemento no domínio de join é chamado de *ponto de ambiguidade* em relação ao join de *R* com *S*.

Se qualquer π_{21} (*R*) ou S é uma função,[145] nenhum ponto de ambiguidade pode ocorrer no join de *R* com *S*. Nesse caso, o natural join de *R* com *S* é o único join de *R* com *S*. Note-se que a reiterada qualificação "de *R* com *S*" é necessária, porque *S* poderia ser joinable com *R* (assim como *R* com *S*), e esta junção seria uma consideração completamente separada. Na Figura 5, nenhuma das relações *R*, $\pi_{21}(R)$, *S*, π_{21} (*S*) é uma função.

Ambiguidades no join de *R* com *S* as vezes podem ser resolvidas por meio de outras relações. Suponha que são dadas, ou pode derivar de fontes

[145] Uma função é uma relação binária, que é um a um ou vários-a-um, mas não um para muitos [Codd].

independentes de *R* e *S*, a relação *T* no domínios *projeto* e *fornecedor* com as seguintes propriedades:

(1) $\pi_1(T) = \pi_2(S)$,

(2) $\pi_2(T) = \pi_1(R)$,

(3) T(j, s) –>∃p(R(S, p) ^ S (p , j)) ,

(4) R(s, p) –>∃j(S(p,j) ^ T(j, s)),

(5) S(p,j) –>∃s(T(j, s) ^ R(s, p)),

então podemos formar um join de três modos de *R, S, T;* isto é, uma relação ternária tal que

$\pi_{12}(U) = R$, $\pi_{23}(U) = S$, $\pi_{31}(U) = T$.

Tal join será chamado *cyclic 3-join* para distingui-lo de um *linear 3-join* o que seria uma relação quaternária *V* de tal forma que

$\pi_{12}(V) = R$, $\pi_{23}(V) = S$, $\pi_{34}(V) = T$.

Embora seja possível existir mais que de um *cyclic 3-join* (ver Figuras 8, 9, para um exemplo), as circunstâncias em que isso pode ocorrer implica muito mais graves restrições

R (*s*	*p*)	*S* (*p*	*j*)	*T*(*j*	*s*)
l	a	a	d	d	l
2	a	a	e	d	2
2	b	b	d	e	2
		b	e	e	2

FIG. 8. Relações binárias com uma pluralidade de *cyclic 3-join*

U	(*s*	*p*	*j*)	U' (*s*	*p*	*j*)
	1	a	d	1	a	d
	2	a	e	2	a	d
	2	b	d	2	a	e

2	b	e	2	b	d
2	b	e			

FIG. 9.Dois *cyclic 3-join* desde as relações da Figura 8

que aquelas advindas de uma pluralidade de 2-joins. Para ser mais específico, as relações *R, S, T* devem possuir pontos de ambiguidade em relação a join *R* com *S* (digamos ponto *x*), *S* com *T* (digamos *y*) e *T* com *R* (digamos *z*) e, além disso, *y* deve ser um parente de *x* em *S*, *z* um parente de *y* em *T*, e *x* um parente de *z* em *R*. Note que na Figura 8 os pontos *x = a, y = d, z = 2* têm essa propriedade.

O natural linear 3-join de três relações binárias *R, S, T* é dado por

R*S*T = { (a , b , c , d) : R (a , b) ^ S(b,c) ^ T(c,d)}

onde os parênteses não são necessários no lado esquerda, pois o natural 2-join (*) é associativo. Para obter a contrapartida cíclica, introduzimos o operador γ que produz uma relação de grau *n* - 1 a partir de uma relação de grau *n*, mantendo seus extremos juntos. Assim, se *R* é uma relação *n-ary* (n ≥ 2), o *laço* de R é definido pela equação

γ (R) = { $(a_I, a_2, \ldots, a_{n-1})$:R$(a_I, a_2, \ldots, a_{n-1}, a_n)$^ $a_I = a_n$}.

Podemos agora representar o natural cyclic 3-join de *R, S, T* pela expressão

γ (R*S*T).

Extensão das noções de linear e cyclic 3-join e os seus homólogos naturais para o join de *n* relações binárias (onde *n* ≥ 3) é óbvia. Algumas palavras podem ser apropriadas, no entanto, sobre a união de relações que não são necessariamente binárias. Consideremos o caso de duas relações de *R* (grau *r*), *S* (grau *s*) que estão joined através de *p* dos seus domínios (*p <r, p <s*). Para simplificar, vamos supor que esses domínios *p* são o

último *p* dos *r* domínios de *R* e o primeiro *p* dos *s* domínios de *S*. Se não fosse assim, poderíamos sempre aplicar permutações necessárias para fazê-lo assim. Agora, pegue o produto Cartesiano dos primeiros *r-p* domínios de R, e nomeie este novo domínio como *A*. Pegue o produto Cartesiano dos últimos *p* domínios de *R*, e nomeie-o como *B*. Pegue o produto Cartesiano dos últimos *s-p* domínios de S e nomeie-o como *C*.

Podemos tratar *R* como se fosse uma relação binária nos domínios *A, B*. Da mesma forma, podemos tratar *S* como se fosse uma relação binária nos domínios *B, C*. As noções de linear e cyclic 3-join agora são diretamente aplicáveis. Uma abordagem semelhante pode ser tomada com o linear e cyclic *n*-join de *n* relações de graus variados.

2.1.4. Composição.

O leitor é provavelmente está familiarizado com a noção de composição aplicada a funções. Vamos discutir uma generalização desse conceito e aplicá-lo primeiro a relações binárias. Nossas definições de composição e *composability* baseiam-se muito diretamente sobre as definições de *join* e *joinability* fornecidas acima.

Suponhamos que temos duas relações *R, S. T* é uma composição de *R* com *S*, se existe um *join U* de *R* com *S* de tal modo que $T = \pi_{13}(U)$. Assim, duas relações são *combináveis*, se e somente se eles são *joinable*. No entanto, a existência de mais do que uma junção de *R* com *S* não implica a existência de mais do que uma composição de *R* com *S*.

Correspondente a natural join de *R* com *S* é a *natural composition*[146]de *R* com *S* definido por

[146] Outros escritores tendem a ignorar outras compositions que não sejam [classificáveis como] natural e, portanto, referem-se a essa composição particular [simplesmente] como a composição - ver, por exemplo, de Kelley, "Topologia Geral" [Codd].

$R \bullet S = \pi_{13}(R*S)$.

Tomando as relações *R, S* a partir da Figura 5, a sua natural composition, é exibida na Figura 10, e uma outra composição está exposta na Figura 11 (derivada a partir do join exibido na Figura 7).

R•S	(projeto	fornecedor)
	1	1
	1	2
	2	1
	2	2

Fig. 10. Uma natural composition de *R* com *S* (desde a Figura 5)

T	(projeto	fornecedor)
	1	2
	2	1

Fig. 11. Outra composition de R com S (desde a Figura 5).

Quando dois ou mais joins existem, o número de diferentes compositions podem ser tão poucos como um ou quantos o número de distintos joins. A figura 12 mostra um exemplo de duas relações que têm vários joins, mas apenas uma composition. Note-se que a ambiguidade de ponto *c* é perdida [desaparece] na composição *R* com *S*, por causa das associações inequívocas realizadas através dos pontos *a, b, d, e*.

R (fornecedor peça)		S (peça projeto)	
1	a	a	g
1	b	b	f
1	c	c	f
2	c	c	g

2	d	d	g
2	e	e	f

Fig. 12. Muitos joins, somente uma composition

A extensão de composition a pares de relações que não são necessariamente binários (e que podem ser de diferentes graus), segue o mesmo padrão da extensão de pairwise joining destas relações.

A falta de compreensão da composition relacional levou vários designers de sistemas para o que pode ser chamado de *armadilha de conexão*. Esta armadilha pode ser descrita em termos do exemplo seguinte. Suponha que a descrição de cada fornecedor está ligada por ponteiros as descrições de cada peça fornecida por esse fornecedor, e cada descrição de peça é igualmente ligada as descrições de cada projeto que usa essa peça. A conclusão obtida agora que é, em geral, errônea: a saber que, se todos os caminhos possíveis são seguidos de um determinado fornecedor através das peças que ele fornece aos projetos que utilizam as peças, se vai obter um conjunto válido de todos os projetos fornecidos por esse fornecedor. Tal conclusão está correta apenas em caso muito especial que a relação entre os projetos-alvo e fornecedores é, de fato, a composição natural das outras duas relações - e devemos normalmente adicionar a frase "para sempre", porque esta é geralmente implícita em reclamações relativas técnicas de seguir caminho.

2.1.5. Restrição.

Um subconjunto de uma relação é uma relação. Uma maneira em que uma relação *S* pode atuar sobre uma relação *R* para gerar um subconjunto de *R* é através da operação *restrição* de *R* por *S*. Esta operação é uma

generalização da restrição de uma função a um subconjunto do seu domínio, e é definida como se segue.

Sejam *L, M* listas de mesmo comprimento de índices tais que $L = i_1,i_2,\ldots, i_k$, $M = j_1,j_2,\ldots, i_k$ onde $k \leq$ grau de *R* e $k \leq$ grau de *S.* Então a restrição *L, M de R por S* denota ${}_R L|_M S$ é o subconjunto máximo *R'* de *R* tal que

$\pi_L (R') = \pi_M (S)$.

A operação é definida somente se a igualdade é aplicável entre os elementos de π_{ih} (R), por um lado e π_{jh} (S) noutro, para todos h = 1, 2, …, *k*.

As três relações *R, S, R'* da Figura 13 satisfazem a equação $R' = R_{(2,3)|(1,2)}S$.

R (s p j)	S (p j)	R' (s p j)
1 a A	a A	1 a A
2 a A	c B	2 a A
2 a B	b B	2 b B
2 b A		
2 b B		

FIG. 13. Exemplo de restrição

Estamos agora em posição de considerar várias aplicações dessas operações sobre as relações.

2.2. REDUNDÂNCIA

Redundância no conjunto nomeado de relações deve ser diferenciada de redundância no conjunto armazenado de representações. Estamos principalmente preocupados com o primeiro. Para começar, precisamos de uma noção precisa de derivabilidade para as relações.

Suponha que θ é um conjunto de operações sobre as relações e cada operação tem a propriedade de que a partir de seus operandos produz uma relação única (assim natural join é elegível, mas join não é). A relação *R* é θ-*derivável* a partir de um conjunto *S* de relações se existe uma sequência de operações da collection θ que, para todos os tempos [sempre], resulta *R* desde membros de *S*. A frase "para todos os tempos [sempre]" está presente, porque temos que lidar com relações que variam no tempo, e nosso interesse é em derivabilidade que detém durante um período de tempo significativo. Para o conjunto nomeado de relacionamentos em sistemas noninferential, parece que uma coleção adequada θ_1 contém as seguintes operações: projection, natural join, tie, e restriction. Permutation é irrelevante e natural composition não precisa ser incluída, porque é obtida tomando um natural join e então uma projection. Para o conjunto armazenado de representações, uma coleção adequada de operações θ_2 deveria incluir permutation e operações adicionais envolvidas com [geração de] sub-conjuntos, merging, ordenação e conexão de seus elementos.

2.2.1. Redundância forte.

Um conjunto de relações é fortemente redundante se contiver pelo menos uma relação que possui uma projeção que é derivável a partir de outras projeções de relações no conjunto. Os dois exemplos a seguir destinam-se a explicar por que a redundância forte é definida desta maneira e para demonstrar a sua utilização prática. No primeiro exemplo, o conjunto de relações consiste em apenas a seguinte relação:

empregado (#serie, nome, #gerente, nomegerente)

com *#serie* como chave primária e *#gerente* como uma chave estrangeira. Vamos denotar o domínio ativo por Δ e supor que

Δ_t (#gerente) $\subset \Delta_t$ (#serie)

and

Δ_t (nomegerente) $\subset \Delta_t$ (nome)

por todo o tempo *t*. Neste caso, a redundância é óbvia: o domínio *nomegerente* é desnecessário. Para ver que esta é uma redundância forte, tal como definido anteriormente, observa-se que

π_{34} (empregado) = π_{12} (empregado) $_1|_1$ π_3 (empregado).

No segundo exemplo, a coleção de relações inclui uma relação *S* descrevendo *fornecedores* com chave primária *s#*, uma relação *D* descrevendo departamentos com chave primária *d#*, uma relação *J* descrevendo projetos com chave primária *j#* e as seguintes relações:

P (s#, d#, . . .), Q (s#, j#, .-.), R (d#, j#, . . .),

onde em cada caso ... denota domínios diferentes de *s#*, *d#*, *j#*. Vamos supor a seguinte condição *C* é conhecida por manter-se independente do tempo: fornecedor *s* departamento de suprimentos *d* (relação *P*) se e somente se o fornecedor *s* fornece suprimentos a alguns projetos *j* (relação *Q*), ao qual é atribuído *d* (relação *R*). Então, podemos escrever a equação

π_{12} (P) = π_{12} (Q)• π_{21} (R)

e, assim, apresentam um forte redundância.

Uma importante razão para a existência de fortes redundâncias no conjunto nomeado de relacionamentos é a conveniência do usuário. Um caso particular deste é a retenção de relações semi-obsoletas no conjunto nomeado de modo que programas antigos que se referem a eles pelo nome podem continuar a funcionar corretamente. O conhecimento da existência de fortes redundâncias no conjunto nomeado permite que um sistema ou o administrador da base de dados uma maior liberdade na escolha de

representações armazenadas para lidar de forma mais eficiente com o tráfego atual. Se as fortes redundâncias no conjunto nomeado são refletidas diretamente como fortes redundâncias no conjunto armazenado (ou se outras fortes redundâncias são introduzidos no conjunto armazenado), então em geral, o espaço de armazenamento extra e tempo de atualização são consumidos com uma queda de potencial no tempo de consulta para algumas consultas e da carga sobre as unidades de processamento central.

2.2.2. Fraca redundância.

Um segundo tipo de redundância pode existir. Em contraste com forte redundância que não é caracterizado por uma equação. Uma coleção de relações é *fracamente redundante* se ele contém uma relação que tem uma projeção que não é derivável de outros membros, mas é sempre uma projeção de *algum* join de outras projeções de relações na coleção.

Podemos exibir uma fraca redundância, tomando o segundo exemplo (citado acima) para uma forte redundância, e assumindo agora que a condição *C* não se mantém em todos os momentos.

As relações π_{12} (*P*), π_{12} (*Q*), π_{12} (*R*) são relações complex[147] com a possibilidade de pontos de ambiguidade ocorrendo de tempos em tempos para o joining potencial de quaisquer duas. Nestas circunstâncias, nenhuma delas é derivável a partir das outras duas. No entanto, as restrições que existem entre eles, uma vez que cada uma é uma projeção de algum cyclic join de três delas. Uma das fracas redundâncias pode ser caracterizada pela instrução: para todo o tempo, $\pi_{12}(P)$ é uma composição de $\pi_{12}(Q)$ com $\pi_{12}(R)$. A composição em questão pode ser a natural, em algum instante e um não natural em outro instante.

[147] Uma relação binária é complexa, se nem ela nem o seu inverso é uma função [Codd].

De um modo geral, as fracas redundâncias são inerentes as necessidades lógicas da comunidade de usuários. Elas não são removíveis pelo administrador do sistema ou base de dados. Se elas aparecem no geral, aparecem tanto o conjunto nomeado quanto no conjunto armazenado de representações.

2.3. CONSISTÊNCIA

Sempre que o conjunto nomeado de relações é redundante em qualquer sentido, vamos associar a esse conjunto uma coleção de declarações que definem todas as redundâncias que mantêm independente do tempo entre as relações de membros. Se o sistema de informação carece - e muito provavelmente irá – de informação semântica detalhada sobre cada relação nomeada, não pode deduzir as redundâncias aplicáveis ao conjunto nomeado. Ele pode, ao longo de um período de tempo, fazer tentativas para induzir as redundâncias, mas tais tentativas seriam falíveis.

Dada uma coleção *C* de relações variáveis no tempo, um conjunto associado *Z* de instruções de restrição e um valor instantâneo *V* para *C*, chamaremos o estado (*C, Z, V*) *consistente* ou *inconsistente* conforme *V* satisfaz ou não satisfaz *Z*. Por exemplo, dadas relações armazenado *R, S, T*, juntamente com a indicação de restrição "π_{12}*(T)* é uma composição de π_{12}(R) com π_{12}(S)", pode-se verificar ao longo do tempo que os valores armazenados para *R, S, T* atendem à essa restrição. Um algoritmo para fazer esta verificação examinará as duas primeiras colunas de cada uma *R, S, T* (de qualquer maneira que eles são representados no sistema) e determinará se

(1) $\pi_1(T) = \pi_1(R)$,

(2) $\pi_2(T) = \pi_2(S)$,

(3) para todo elemento pareado (*a, c*) na relação $\pi_{12}(T)$ existe um elemento *b* de modo que (*a, b*) esteja em $\pi_{12}(R)$ e (*b, c*) esteja em $\pi_{12}(S)$.

Existem problemas práticos (que não serão discutidos aqui) em tomar uma imagem instantânea de um conjunto de relações, algumas dos quais podem ser muito grandes e muito variáveis.

É importante notar que a consistência como acima definido é uma propriedade do estado instantâneo de um banco de dados, e é independente de como esse estado surgiu. Assim, em particular, não há nenhuma distinção feita com base em se um usuário gerou uma inconsistência devido a um ato de omissão ou de um ato de comissão [com autoridade, por dever]. O exame de um exemplo simples vai mostrar a razoabilidade desta abordagem (possivelmente não convencional) para a consistência.

Suponha que o conjunto nomeado *C* inclui as relações *S, J, D, P, Q, R* do exemplo da Seção 2.2 e que *P, Q, R* possuem tanto fortes ou fracas dependências nele descritas (no caso particular, agora sob consideração, não importa qual o tipo de redundância ocorre). Além disso, suponha que em algum momento *t* do estado do banco de dados é consistente e não contém nenhum projeto *j* tal que fornecedor 2 fornece ao projeto *j* e *j* é atribuído ao departamento 5. Por conseguinte, não há nenhum elemento (2, 5) na $\pi_{12}(P)$. Agora, um usuário introduz o elemento (2, 5) na $\pi_{12}(P)$ através da inserção de algum elemento apropriado em *P*. O estado do banco de dados é agora inconsistente. A inconsistência poderia ter surgido a partir de um ato de omissão, se a entrada (2, 5) está correta, e existe um projeto *j* de tal forma que o fornecedor 2 fornece ao projeto *j* e *j* é atribuído ao departamento 5. Neste caso, é muito provável que o usuário pretende, no

futuro próximo, inserir elementos em *Q* e *R*, que terá o efeito de introduzir (2, *j*) em $\pi_{12}(Q)$ e (5, *j*) em $\pi_{12}(R)$. Por outro lado, a entrada (2, 5) pode ter sido danificada [errônea]. Poderia ser o caso de que o usuário pretendia inserir algum outro elemento em *P* - um elemento cuja inserção transformaria um estado consistente em um estado [in]consistente.[148] O ponto é que o sistema, normalmente, não têm nenhuma maneira de resolver esta questão, sem interrogar seu ambiente (talvez o usuário que criou a inconsistência).

Existem, naturalmente, vários modos possíveis, em que um sistema pode detectar inconsistências e responder a elas. Em uma abordagem o sistema verifica para uma possível inconsistência sempre que uma inserção, exclusão ou atualização da chave ocorre. Naturalmente, essa verificação vai abrandar estas operações para baixo. Se uma inconsistência foi gerada, os detalhes são registrados internamente, e se não for sanada no prazo de algum intervalo de tempo razoável, o usuário ou alguém responsável pela segurança e integridade dos dados é notificado. Outra abordagem é a realização de verificação de consistência como uma operação de lote, uma vez por dia ou menos frequentemente. Entradas fazendo com que as inconsistências permaneçam no banco de dados no momento da verificação podem ser rastreadas se o sistema mantém um diário de todas as operações de mudança de estado. Esta última abordagem seria certamente superior se algumas inconsistências não transitórias ocorreram.

2.4. RESUMO

[148] A oração "transformaria um estado consistente em um estado consistente" parece ser incoerente. Outras edições do documento estão grafadas do mesmo modo. Nossa intervenção, indicaria uma ideia mais pertinente, coerente inclusive com a oração de Codd que segue [nossa].

Na Seção 1 um modelo relacional de dados é proposto como base para proteger os usuários de sistemas de dados formatados a partir das mudanças potencialmente perturbadoras em representação de dados causados pelo crescimento do banco de dados e mudanças no tráfego. Uma forma normal para coleções de relacionamentos variáveis no tempo é introduzida.

Na Seção 2 operações sobre as relações e dois tipos de redundância são definidos e aplicados ao problema de manter os dados em um estado consistente. Este deve tornar-se um problema prático grave a medida que mais e mais tipos diferentes de dados são integrados juntos em bancos de dados comuns.

Muitas questões são levantadas e deixadas sem resposta. Por exemplo, apenas algumas das propriedades mais importantes da sublinguagen de dados na Seção 1.4 são mencionados. Nem são discutidos os detalhes puramente linguísticos de tal linguagem, nem os problemas de implementação. No entanto, o material apresentado deve ser adequado para programadores de sistemas experientes para visualizarem várias abordagens. Espera-se também que este trabalho pode contribuir para uma maior precisão nos trabalhos sobre sistemas de dados formatados.

Agradecimento. Foi C. T. Davies da IBM Poughkeepsie que convenceu o autor da necessidade da independência de dados em futuros sistemas de informação. O autor gostaria de agradecer a ele e também F. P. Palermo, C. P. Wang, E. B. Altman e M. E. Senko do Laboratório de Pesquisa da IBM San Jose pelas discussões úteis.

RECEBIDO em SETEMBRO de 1969; REVISADO em FEVEREIRO de 1970

REFERÊNCIAS

1.CHILDS, D.L. Feasibility of a set-theoretical data structure - a general structure based on a reconstituted definition of relation. Proc. IFIP Cong., 1968, North Holland Pub. Co., Amsterdam, p. 162-172.

2. LEVEIN, R. E., AND MARON, M. E. A computer system for inference execution and data retrieval. Comm. ACM 10, 11 (Nov. 1967), 715--721.

3. BACHMAN, C. W. Software for random access processing. Datamation (Apr. 1965), 36-41.

4. McGEE, W. C. Generalized file processing. In Annual Review in Automatic Programming 5, 13, Pergamon Press, New York, 1969, pp. 77-149.

5. Information Management System/360, Application Description Manual H20-0524-1. IBM Corp., White Plains, N. Y., July 1968.

6. GIS (Generalized Information System), Application Description Manual H20-0574. IBM Corp., White Plains, N. Y., 1965.

7. BLEIER, R.E. Treating hierarchical data structures in the SDC time-shared data management system (TDMS). Proc. ACM 22nd Nat. Conf., 1967, MDI Publications, Wayne, Pa., pp. 41---49.

8. IDS Reference Manual GE 625/635, GE Inform. Sys. Div., Pheonix, Ariz., CPB 1093B, Feb. 1968.

9. CHURCH, A. An Introduction to Mathematical Logic I. Princeton U. Press, Princeton, N.J., 1956.

10. FELDMAN, J. A., AND ROVNER, P.D. An Algol-based associative language. Stanford Artificial Intelligence Rep. AI-66, Aug. 1, 1968.

Anexo 3 – Outros documentos

Indicamos aqui diversos outros textos que apresentam a mesma informação, qual seja, o papel absoluto de Codd na concepção dos *SGBDR*s. Observe-se que com grande frequência os termos se repetem, constituindo uma frequente reafirmação de uma pretensa verdade, digamos, absoluta demais para não causar estranheza, ao menos para o tipo de ensaio que pretendemos realizar.

Date C. J. [149]

p.27:

Codd foi o inventor do modelo relacional [...] tem uma certa razão ao afirmar ser o inventor do conceito de modelo de dados em geral, bem como do modelo de dados relacional em particular.

p.147:

E. F. Codd: "A Relational Model of Data for Large Shared Data Banks" [...] O artigo que deu início a tudo. Embora já tenha mais de 30 anos, ele aceita notavelmente bem a leitura repetida [...] Codd passou grande parte do final da década de 1980 revisando e estendendo seu modelo original (ao qual ele agora se refere como "o Modelo Relacional Versão1" ou RM/V1).

Elmasri, Shamkant.[150]

p.137:

O modelo relacional de dados foi introduzido por Codd (1970). É baseado em uma simples e uniforme estrutura de dados – a relação – e tem uma fundação teórica sólida.

p.184:

---

[149] Date, *Introdução a Sistemas de Bancos de Dados*.
[150] Elmasri, *Fundamentals of Database Systems*.

O modelo relacional foi introduzido por Codd (1970) em um ensaio clássico. Codd também introduziu a álgebra relacional e estabeleceu os fundamentos teóricos para o modelo relacional em uma série de ensaios [...] foi dado a ele o prêmio Turing, a mais alta honra da ACM, por seu trabalho no modelo relacional [...] incorporando NULLS [nulos] a álgebra relacional. O modelo resultante é conhecido como RM/T. Um trabalho mais recente de Childs (1968) utilizou a teoria de conjuntos para modelar bancos de dados.

Maier.[151]

p.10:

O modelo de dados relacional foi originalmente exposto em uma série de ensaios por Codd.

p.24:

Os operadores relacionais seleção, projeção e junção na forme que foram introduzidas por Codd [...] embora analogamente tenham sido dados por Childs para um modelo ligeiramente diferente.

p.70:

FDs [Functional Dependencies – Dependências Funcionais] foram apresentadas quando Codd introduziu inicialmente o modelo relacional, em forma de chaves [...] mais tarde, introduziu FDs que não dependem [follow from] de chaves, com o propósito de normalização.

Ramakrishnan, p.5.[152]

Em 1970, Edgar Codd, do Laboratório de Pesquisa de San Jose, da IBM, propôs uma nova estrutura de representação de dados chamada *modelo de*

---

[151] Maier, *The Theory of Relational Databases*.
[152] Ramakrishnan, *Sistema de Gerenciamento de Banco de Dados*.

dados relacional, que veio AA ser um marco histórico no desenvolvimento de sistemas de banco de dados. Ele impulsionou o rápido desenvolvimento de vários SGBDs baseados no modelo relacional, juntamente com um rico conjunto de resultados teóricos que consolidaram firmemente a área. Codd ganhou o Prêmio Turing de 1981 pelo seu trabalho original. Os sistemas de banco de dados amadureceram como uma disciplina acadêmica, e a popularidade dos SGBDRs alterou o cenário comercial. Seus benefícios foram amplamente reconhecidos, e o uso de SGBDs para gerenciar dados corporativos tornou-se uma prática padrão.

Silberschatz, p.19.[153]

Um documento revolucionário de Codd [1970] definiu o modelo relacional e os métodos voltados para procedimento de consulta a dados no modelo relacional, dando origem aos bancos de dados relacionais. A simplicidade do modelo relacional e a possibilidade de ocultar completamente os detalhes de implementação do programador eram propostas realmente tentadoras. Mais tarde, Codd ganhou o prestigiado prêmio da Association of Computing Machinery [ACM] Turing pelo seu trabalho.

Silva.[154]

p.21:

Em uma série de artigos, Codd introduziu o modelo relacional de dados e discutiu suas vantagens em termos de simplicidade, simetria, independência dos dados e de completeza semântica ("semantic completeness").

p.53-4:

---

[153] Silberschatz, *Sistema de Banco de Dados*.
[154] Silva, "Implementação de um Sistema de Gerenciamento de Banco de Dados Relacional".

Além da introdução da estrutura relacional de dados, Codd definiu dois modelos de linguagem, baseados em cálculo relacional e álgebra relacional, que têm como objetivo servir como base para linguagens de manipulação de dados.

O resultado de qualquer consulta é um conjunto, ou seja, é sempre uma relação derivada,de alguma maneira, das relações que compõem o modelo de dados. De uma maneira geral, o resultado pode ser extraído de uma Única relação ou pode envolver duas ou mais relações do modelo de dados; portanto a linguagem de consulta deve permitir ao usuário especificar quais as relações que devem ser manipuladas a fim de se conseguir o resultado desejado.

Sumathi, p.728.[155]

Ted Codd foi um verdadeiro pioneiro da computação. Ele também foi uma inspiração para todos nós que tiveram a sorte de conhecê-lo e trabalhar com ele [...] na década de 1960, ele voltou sua atenção ao problema da gestão de grandes bases de dados comerciais - e nos próximos anos, ele criou, sozinho, a invenção com que seu nome será para sempre associada: o modelo relacional de dados.

O modelo relacional é amplamente reconhecido como uma das grandes inovações técnicas do século XX. Codd descreveu e explorou sua implicações em uma série de trabalhos de pesquisa [...] Que publicou ao longo do período 1969-1979 . O efeito dos ensaios era duplo: eles mudaram para melhor a maneira da Tecnologia da Informação (TI) no mundo (incluindo a componente acadêmica de que o mundo, em particular) percebia o problema de gerenciamento de banco de dados, e eles lançaram

[155] Sumathi, *Fundamentals of Relational Database Management Systems*.

as bases para toda uma nova indústria, a indústria de banco de dados relacional, que agora vale muitos bilhões de dólares por ano. Na verdade, não só modelo relacional de Codd definiu toda a disciplina de gerenciamento de banco de dados sobre uma base científica sólida, mas também serviu de base para uma tecnologia que tem tido e continua a ter, um grande impacto sobre a própria estrutura da nossa sociedade. Não é exagero dizer que Ted Codd é o pai intelectual do campo de banco de dados moderno.

Realização suprema da Codd com o modelo relacional não deve ser permissão para eclipsar o fato de que ele fez grandes contribuições originais em vários outras áreas importantes, bem como, incluindo multiprogramação, linguagem natural processamento, e, mais recentemente, da empresa Delta (uma abordagem relacional para negócios gerenciamento de regras), para o qual a ele e sua esposa foram concedidos uma patente nos EUA.

A profundidade e a amplitude de suas contribuições foram reconhecidas pela longa lista de honras e cargos eletivos que foram conferidos a ele durante a sua vida, incluindo IBM Fellow, eleito ACM Fellow, eleito Fellow da Sociedade de computadores da Grã-Bretanha, membro eleito da Academia Nacional de Engenharia e membro eleito da Academia Americana de Artes e Ciências Cinematográficas. Em 1981, ele recebeu o ACM Turing Award, o prêmio de maior prestígio na área da Ciência da Informática. Ele também recebeu um prêmio de reconhecimento excepcional do IEEE: a primeira conquista prêmio anual a partir do Grupo de Usuários Internacionais do DB2, e um outro prêmio pelo conjunto anual de DAMA em 2001. Computerworld, em comemoração ao 25º aniversário de sua publicação, foi selecionado como um dos 25 indivíduos em ou relacionados à área de

computação que têm teve o maior efeito sobre a nossa sociedade. E a revista Forbes, que em dezembro de 2002 publicou uma lista das inovações e contribuições mais importantes para cada um dos 85 anos de sua existência, foi selecionado para o ano de 1970 o modelo relacional de dados por E. F. Codd.

Yong, p.383.[156]

Este artigo de Codd influiu decididamente no interesse geral sobre o modelo relacional. Contém explicações gerais sobre o modelo relacional e operações com a álgebra relacional.

[156] Yong, *Banco de Dados. Organização Sistemas e Administração*.