Veri Tabanına Giriş

Bir veritabanı, bir dizi ilişkili veriden ibarettir. Örneğin bir telefon rehberi, belirli bir bölgede yaşayan tüm insanların adlarının, telefon numaralarının ve adreslerinin bulunduğu bir veri tabanıdır. Bir telefon rehberi her yerde bulunan ve sık kullanılan bir veri tabanı olsa da, aşağıdaki dezavantajları vadır:

- Özellikle telefon rehberi çok sayıda giriş içeriyorsa, bir kişinin telefon numarasını bulmak zaman alabilir.
- Bir telefon rehberi sadece soyadları/adları ile indekslenir, bu nedenle belirli bir adreste yaşayan kişilerin isimlerini bulmak teorik olarak mümkün olsa da bu veritabanı için pratik bir kullanım değildir.
- Telefon rehberi yazdırıldığı andan itibaren, insanlar bir bölgeye girip çıktıkça, telefon numaralarını değiştirdikçe veya aynı bölge içinde başka bir yere taşındıkça bilgiler giderek daha az doğru hale gelir.

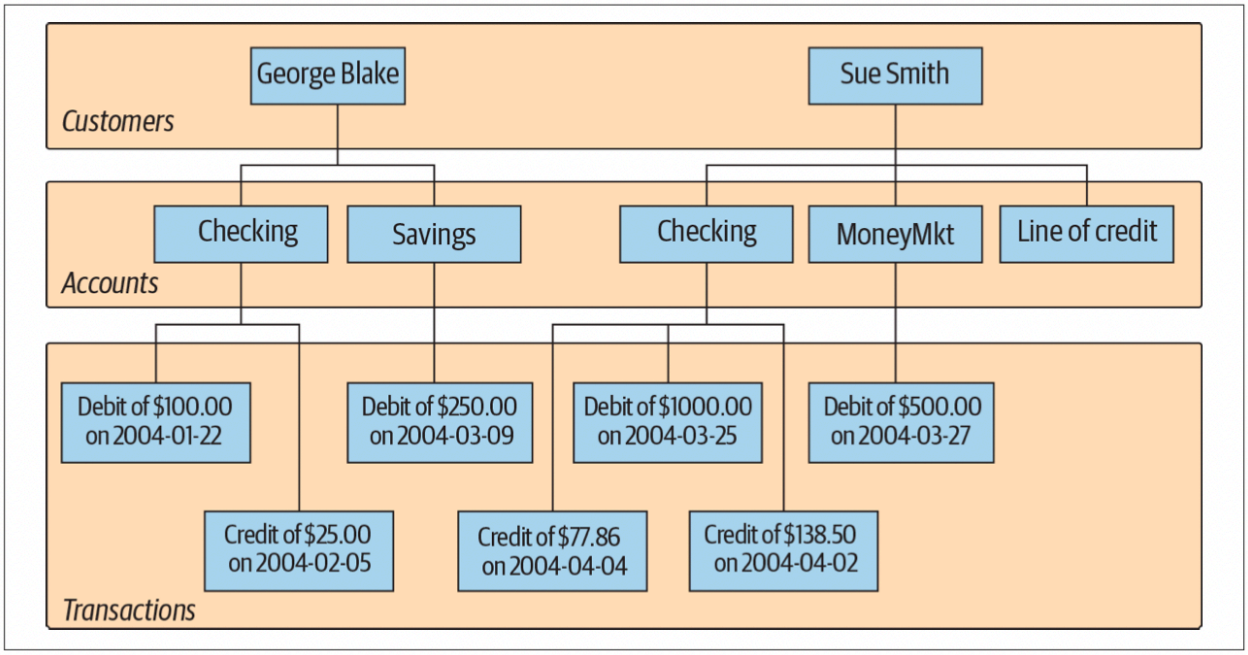
İlk Veritabanı

Telefon rehberlerine atfedilen aynı dezavantajlar, bir dosya dolabında saklanan hasta kayıtları gibi herhangi bir manuel veri depolama sistemi için de geçerli olabilir. Kağıt veri tabanlarının hantal doğası nedeniyle, geliştirilen ilk bilgisayar uygulamalarından bazıları, bilgisayarlı veri depolama ve okuma mekanizmaları olan veri tabanı sistemleriydi. Bir veritabanı sistemi verileri kağıt yerine elektronik olarak depoladığından, bir veritabanı sistemi verileri daha hızlı alabilir, verileri birden çok yolla indeksleyebilir ve kullanıcı topluluğuna en güncel bilgileri iletebilir.

Erken veri tabanı sistemleri, manyetik bantlarda depolanan verileri yönetiyordu. Genellikle teyp okuyuculardan çok daha fazla teyp olduğu için, teknisyenlere belirli veriler istendiğinde teypleri yükleme ve boşaltma görevi verildi. O dönemin bilgisayarları çok az belleğe sahip olduğundan, aynı veriler için birden çok istek genellikle verilerin banttan birden çok kez okunmasını gerektiriyordu. Bu veritabanı sistemleri, kağıt veritabanlarına göre önemli bir gelişme olsa da, günümüz teknolojisinden çok uzaktır. (Modern veritabanı sistemleri, her biri onlarca gigabaytlık veriyi yüksek hızlı bellekte önbelleğe alan sunucu kümeleri tarafından erişilen petabaytlarca veriyi yönetebilir.)

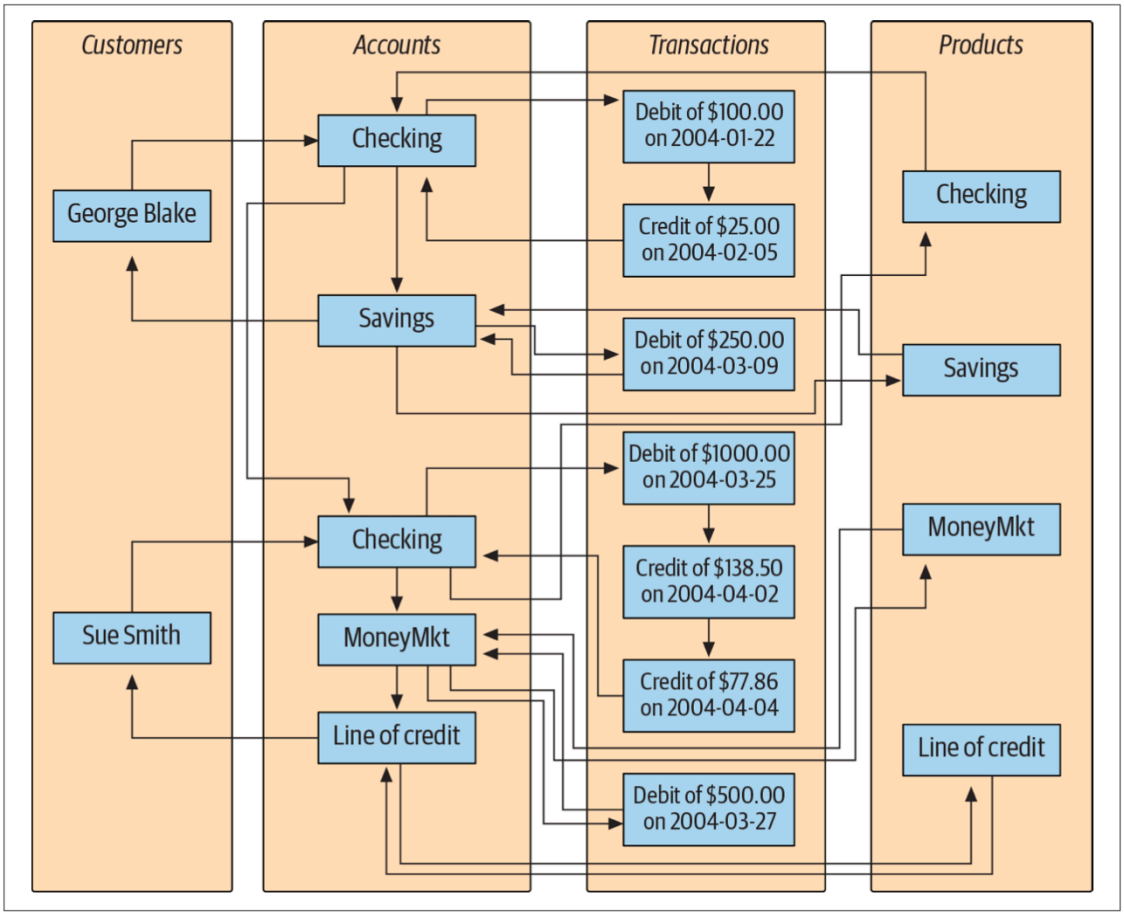
İlişkisel Olmayan Veritabanı Sistemleri

Bilgisayarlı veritabanı sistemlerine ilk birkaç on yılı boyunca, veriler depolandı ve kullanıcılara çeşitli şekillerde sunuldu. Hiyerarşik bir veritabanı sisteminde, veriler bir veya daha fazla ağaç yapısı olarak temsil edilir. Şekilde verilen ağaç veri tabanı sisteminde George ve Sue'nun her birinin kendi hesaplarını ve bu hesaplardaki işlemleri içeren kendi ağaçları vardır. Hiyerarşik veritabanı sistemi, belirli bir müşterinin ağacını bulmak ve ardından istenen hesapları ve/veya işlemleri bulmak ve ağaçta gezinmek için araçlar sağlar. Ağaçtaki her düğümün ya sıfır ya da bir ebeveyni ve sıfır, bir ya da çok sayıda çocuğu olabilir. Bu yapılandırma, tek ebeveynli hiyerarşi olarak bilinir.



Hiyerarşik Veritabanı

Ağ veritabanı sistemi adı verilen diğer bir yaygın yaklaşım, farklı kayıtlar arasındaki ilişkileri tanımlayan kayıt kümelerini ve bağlantı kümelerini ortaya çıkarır.

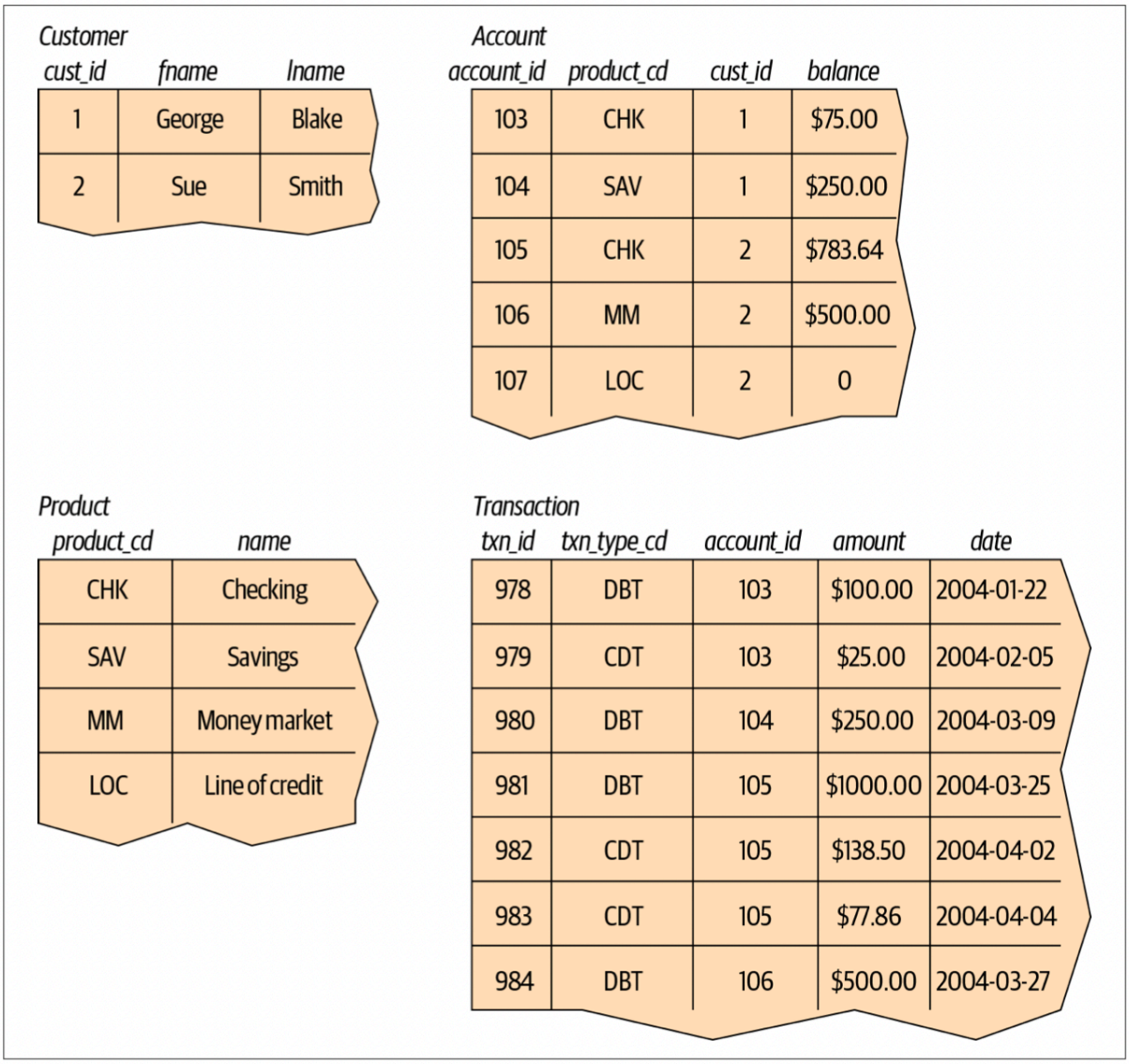


Ağ Veritabanı

Kaynak: Learning SQL, 3rd Edition, Alan Beaulieu, O'Reilly.

İlişkisel Model

1970 yılında, IBM'in araştırma laboratuvarından Dr. E. F. Codd, verilerin tablo kümeleri olarak temsil edilmesini öneren “A Relational Model of Data for Large Shared Data Banks” başlıklı bir makale yayınladı. İlişkili varlıklar arasında gezinmek için işaretçiler kullanmak yerine, farklı tablolardaki kayıtları bağlamak için artık veriler kullanılır.



Customer

cust_id	fname	lname
1	George	Blake
2	Sue	Smith

Account

account_id	product_cd	cust_id	balance
103	CHK	1	$75.00
104	SAV	1	$250.00
105	CHK	2	$783.64
106	MM	2	$500.00
107	LOC	2	0

Product

product_cd	name
CHK	Checking
SAV	Savings
MM	Money market
LOC	Line of credit

Transaction

txn_id	txn_type_cd	account_id	amount	date
978	DBT	103	$100.00	2004-01-22
979	CDT	103	$25.00	2004-02-05
980	DBT	104	$250.00	2004-03-09
981	DBT	105	$1000.00	2004-03-25
982	CDT	105	$138.50	2004-04-02
983	CDT	105	$77.86	2004-04-04
984	DBT	106	$500.00	2004-03-27

İlişkisel veritabanı modeli

Dört tablo, şimdiye kadar tartışılan dört varlığı temsil etmektedir: müşteri, ürün, hesap ve işlem. Şekildeki müşteri tablosunun üst kısmına baktığınızda, üç sütun görebilirsiniz: cust_id (müşterinin kimlik numarasını içerir), fname (müşterinin adını içerir) ve lname (müşterinin soyadını içerir). Müşteri tablosunun yan tarafına baktığınızda, biri George Blake'in verilerini ve diğeri Sue Smith'in verilerini içeren iki satır görebilirsiniz. Bir tablonun içerebileceği sütun sayısı sunucudan sunucuya farklılık gösterir, ancak genellikle sorun olmayacak kadar büyüktür (örneğin, Microsoft SQL Server, tablo başına 1.024'e kadar sütuna izin verir). Bir tablonun içerebileceği satır sayısı, veritabanı sunucusundan çok fiziksel sınırlar (yani, ne kadar disk sürücüsü alanı kullanılabilir) ve sürdürülebilirlik (yani, bir tablonun çalışması zorlaşmadan önce ne kadar büyük olabileceği) meselesidir.

Kaynak: Learning SQL, 3rd Edition, Alan Beaulieu, O’Reilly.

Benzersiz değerlerin önemi

İlişkisel bir veritabanındaki her tablo varlığı tamamen açıklamak için gereken ek bilgilerle birlikte, o tablodaki bir satırı (birincil anahtar olarak bilinir) benzersiz şekilde tanımlayan bilgileri içerir. Müşteri tablosuna tekrar bakıldığında, cust_id sütunu her müşteri için farklı bir numara tutar; Örneğin George Blake, müşteri kimliği 1 ile benzersiz bir şekilde tanımlanabilir. Bu tanımlayıcı hiçbir zaman başka bir müşteriye atanmaz ve George Blake'in müşteri tablosundaki verilerini bulmak için başka hiçbir bilgiye ihtiyaç yoktur.

- **Her veritabanı sunucusu, birincil anahtar değerleri olarak kullanmak üzere benzersiz sayı kümeleri oluşturmak için bir mekanizma sağlar, böylece hangi numaraların atandığını takip etme konusunda endişelenmenize gerek kalmaz.**

Birincil anahtar olarak fname ve lname sütunlarının birleşimi de bir benzersiz değer oluşturabilir. (iki veya daha fazla sütundan oluşan birincil anahtar, bileşik anahtar olarak bilinir).

Bu örnekte, birincil anahtar olarak fname/lname seçilmesi doğal anahtar olarak, cust_id seçimi ise vekil anahtar olarak anılır. Doğal veya vekil anahtarların kullanılıp kullanılmaması kararı veritabanı tasarımcısına bağlıdır, ancak bu özel durumda seçim açıktır, çünkü bir kişinin soyadı değişebilir (örneğin, bir kişinin bir eşin soyadını benimsemesi gibi) ve birincil anahtar sütunları bir değer atandıktan sonra asla değişmesine izin verilmemelidir.

Tablolar arası gezinti

Tablolardan bazıları, başka bir tabloya gitmek için kullanılan bilgileri de içerir; burada daha önce bahsedilen "gereksiz veriler" devreye girer. Örneğin, hesap tablosu, hesabı açan müşterinin benzersiz tanımlayıcısını içeren cust_id adlı bir sütun ile hesabın uygun olacağı ürünün benzersiz tanımlayıcısını içeren product_cd adlı bir sütun içerir. Bu sütunlar yabancı anahtarlar olarak bilinir ve hesap bilgilerinin hiyerarşik ve ağ sürümlerinde varlıkları birbirine bağlayan satırlarla aynı amaca hizmet eder. Belirli bir hesap kaydına bakıyorsanız ve hesabı açan müşteri hakkında daha fazla bilgi edinmek istiyorsanız, cust_id sütununun değerini alır ve müşteri tablosunda uygun satırı bulmak için kullanırsınız.

Aynı verileri birçok kez depolamak israf gibi görünebilir, ancak ilişkisel model, hangi gereksiz verilerin depolanabileceği konusunda oldukça açıktır. Örneğin, hesap tablosunda, hesabı açan müşterinin benzersiz tanımlayıcısı için bir sütun bulunması uygun olmakla birlikte, müşterinin adı ve soyadının da hesap tablosunda yer alması doğru değildir. Örneğin, bir müşteri adını değiştirecekse, veritabanında müşterinin adını tutan tek bir yer olduğundan emin olmak istersiniz; aksi takdirde veriler bir yerde değişip başka bir yerde değişemez ve bu da veritabanındaki verilerin güvenilmez olmasına neden olabilir. Bu veriler için uygun yer müşteri tablosudur ve diğer tablolarda sadece cust_id değerleri yer almalıdır. Aynı zamanda, bir kişinin hem adını hem de soyadını içeren bir ad sütunu veya sokak, şehir, eyalet ve posta kodu bilgilerini içeren bir adres sütunu gibi, tek bir sütunun birden çok bilgi içermesi de uygun değildir. Bu her bağımsız bilgi parçasının (yabancı anahtarlar hariç) tek bir yerde olmasını sağlamak için bir veritabanı tasarımının iyileştirilmesi işlemi, normalizasyon olarak bilinir.

George Blake'in çek hesabının işlemlerini bulmak için ilk yapılması gereken müşteri tablosundan George Blake ismine karşılık gelen benzersiz tanımlayıcı (cust_id) bulunur. Bu değer ile hesap tablosunda product_cd değerlerine bakılır ve ürün tablosunda değeri "Checking" olan isimlerin product_cd değerlerine göre accound_id seçilir, Son olarak bu hesap numarasına karşılık gelen işlemler Transaction tablosundan belirlenir.

SQL

Codd'un ilişkisel model tanımıyla birlikte, ilişkisel tablolardaki verileri işlemek için DSL/ Alpha adlı bir dil önerdi. Codd'un makalesi yayınlandıktan kısa bir süre sonra IBM, Codd'un fikirlerine dayalı bir prototip oluşturmak için bir grup görevlendirdi. Bu grup, SQUARE adını verdikleri basitleştirilmiş bir DSL/Alpha sürümü oluşturdu. SQUARE'deki iyileştirmeler, sonunda SQL olarak kısaltılan SEQUEL adlı bir dile yol açtı. SQL, ilişkisel veritabanlarında verileri işlemek için kullanılan bir dil olarak başlarken, çeşitli veritabanı teknolojileri arasında verileri işlemek için bir dil olarak gelişti. SQL'in yaşı artık 40 'dan fazla ve bu süreçte büyük bir değişim geçirdi. 1980'lerin ortalarında, Amerikan Ulusal Standartlar Enstitüsü (ANSI), 1986'da yayınlanan SQL dili için ilk standart üzerinde çalışmaya başladı. Daha sonraki iyileştirmeler, 1989, 1992, 1999, 2003'te SQL standardının yeni sürümlerine yol açtı. 2006, 2008, 2011 ve 2016. Çekirdek dilde yapılan iyileştirmelerin yanı sıra, nesne yönelimli işlevselliği dahil etmek için SQL diline yeni özellikler eklendi. Sonraki standartlar, genişletilebilir biçimlendirme dili (XML) ve JavaScript nesne gösterimi (JSON) gibi ilgili teknolojilerin entegrasyonuna odaklanır. SQL, ilişkisel modelle el ele gider çünkü bir SQL sorgusunun sonucu bir tablodur (bu bağlamda sonuç kümesi olarak da adlandırılır). Böylece, bir sorgunun sonuç kümesini depolayarak ilişkisel bir veritabanında yeni bir kalıcı tablo oluşturulabilir. Benzer şekilde, bir sorgu hem kalıcı tabloları hem de diğer sorgulardan elde edilen sonuç kümelerini girdi olarak kullanabilir.

SQL Deyimleri

SQL dili birkaç farklı bölüme ayrılmıştır: veritabanında depolanan veri yapılarını tanımlamak için kullanılan SQL şema ifadeleri; SQL şema ifadeleri kullanılarak önceden tanımlanmış veri yapılarını işlemek için kullanılan SQL veri ifadeleri ve işlemleri başlatmak, bitirmek ve geri almak için kullanılan SQL işlem deyimleri. Örneğin, veritabanınızda yeni bir tablo oluşturmak için SQL şema deyimlerini kullanırsınız, yeni tablonuzu verilerle doldurma işlemi için de SQL veri deyimleri kullanılır

```
CREATE TABLE corporation
     (corp_id SMALLINT,
      name VARCHAR(30),
      CONSTRAINT pk_corporation PRIMARY KEY (corp_id)
     );
```

Bu ifade, corp_id ve name olmak üzere iki sütunlu bir tablo oluşturur ve corp_id sütunu tablonun birincil anahtarı olarak tanımlanır.

```
INSERT INTO corporation (corp_id, name)
    VALUES (27, 'Acme Paper Corporation');
```

Bu ifade, corp_id sütunu için 27 değerine ve ad sütunu için Acme Paper Corporation değerine sahip şirket tablosuna bir satır ekler.

Kaynak: Learning SQL, 3rd Edition, Alan Beaulieu, O'Reilly.

Son olarak, az önce oluşturulan verileri almak için basit bir seçim ifadesi:

```
mysql< SELECT name
        -> FROM corporation
        -> WHERE corp_id = 27;
    +------------------------+
    | name                   |
    +------------------------+
    | Acme Paper Corporation |
    +------------------------+
```