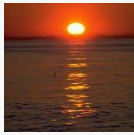
Das Bild kann zurzeit nicht angezeigt werden.

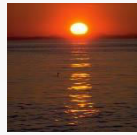
Chapter 2: Relational Model
Tablosal (İlişkisel) Model

Database System Concepts, 5th Ed.

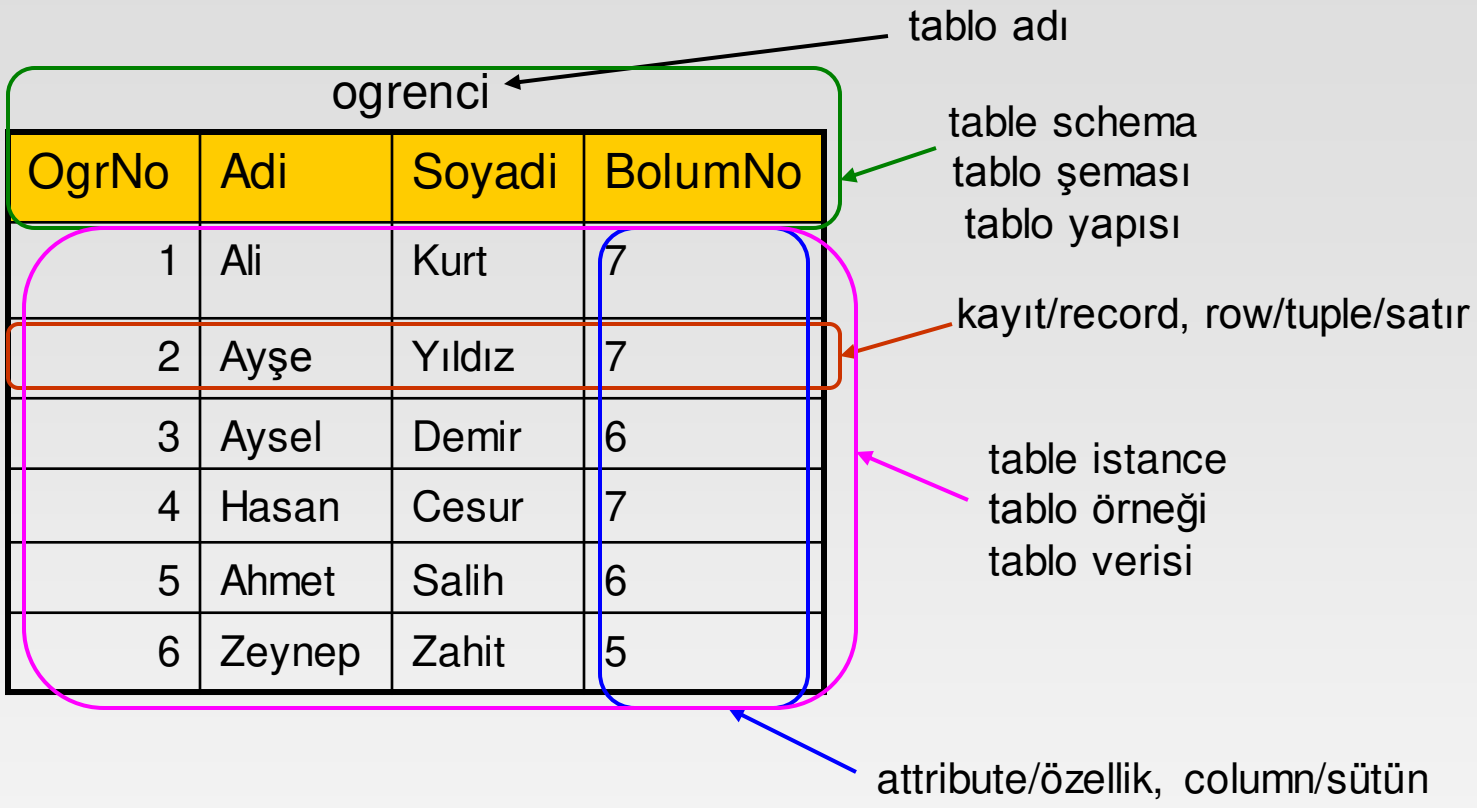


Chapter 2: Relational Model

- Structure of Relational Databases İlişkisel Veritabanı Yapısı
- Fundamental Relational-Algebra-Operations Temel İlişkisel Cebir İşlemleri
- Additional Relational-Algebra-Operations Ek İlişkisel Cebir İşlemleri
- Extended Relational-Algebra-Operations Genişletilmiş İlişkisel Cebir İşlemleri
- Null Values Null Değerler
- Modification of the Database Veritabanı Güncelleme İşlemleri



Tablo/İlişki (relation) örneği



OgrNo	Adi	Soyadi	BolumNo
1	Ali	Kurt	7
2	Ayşe	Yıldız	7
3	Aysel	Demir	6
4	Hasan	Cesur	7
5	Ahmet	Salih	6
6	Zeynep	Zahit	5

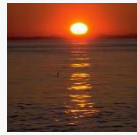
Basic Structure

- Formally, given sets D_1, D_2, …. D_n a **relation** r is a subset of
  D_1 x D_2 x … x D_n
  Thus, a relation is a set of n-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$
- Matematiksel olarak, bir r ilişkisi D_1 x D_2 x … x D_n kartezyen çarpımının bir alt kümesidir. Diğer bir ifade ile r ilişkisi $a_i \in D_i$ olmak üzere n-değerden oluşan $(a_1, a_2, \ldots, a_n)$ kayıtlarının kümesidir.
- Example: If
  - *customer_name* = {Jones, Smith, Curry, Lindsay, …}
    /* Set of all customer names */
  - *customer_street* = {Main, North, Park, …} /* set of all street names*/
  - *customer_city* = {Harrison, Rye, Pittsfield, …} /* set of all city names */

  Then r = { (Jones, Main, Harrison),
  (Smith, North, Rye),
  (Curry, North, Rye),
  (Lindsay, Park, Pittsfield) }

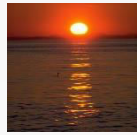
  is a relation over

  customer_name x customer_street x customer_city



Attribute Types (Veri Türleri)

- Each attribute of a relation has a name. Her özellik bir isim ve tür'e sahiptir.
- The set of allowed values for each attribute is called the **domain** of the attribute Bir özelliğin alabildiği olası değerler kümesine tür/domain denir.
- Attribute values are (normally) required to be **atomic**; that is, indivisible Değerler normalde atomik yani bölünmez değerlerdir.
  - E.g. the value of an attribute can be an account number, but cannot be a set of account numbers
- Domain is said to be atomic if all its members are atomic. Eğer bir tür'ün tüm elemaları atomik ise tür atomik bir türdür
- The special value *null* is a member of every domain. Null her tür'ün elemanıdır.
- The null value causes complications in the definition of many operations. Null birçok işlemde problem oluşturur, bu yüzden null'lı işlem durumları özel olarak tanımlanmalıdır.
  - We shall ignore the effect of null values in our main presentation and consider their effect later. Şimdilik sunum sırasında null gözönüne alınmayacak olup ileride yeri geldiğinde null özerinde durulacaktır.



Relation Schema (Tablo Şeması)

- $A_1, A_2, \ldots, A_n$ are *attributes.*

$R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*

Example: C*ustomer_schema* = (*customer_name, customer_street, customer_city*)

r(*R*) denotes a *relation r* on the *relation schema R*

Example: c*ustomer (Customer_schema)*

$A_1, A_2, \ldots, A_n$ özelliklerine ve $D_1, D_2, \ldots. D_n$ türlerine sahip r adlı ilişkinin şeması $r(A_1, A_2, \ldots, A_n)$ veya $r(A_1\ D_1, A_2\ D_2, \ldots, A_n\ D_n)$ şeklinde ifade edilir. Örneğin

- *Ogrenci(ogrno int, adi char(10), soyadi char(10))*
- *Ogrenci(ogrno, adi, soyadi) – veri tipleri genellikle yazılmaz*

Relation Instance (Tablo Örneği/Verisi)

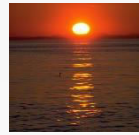
- The current values (*relation instance*) of a relation are specified by a table. Bir ilişkinin bir tabloyla ifade edilen o anki değerlerine ilişki örneği veya ilişki verisi denir
- An element *t* of *r* is a *tuple*, represented by a *row* in a table. Bir *r* ilişkisinin *t* kayıdı tabloda bir satırla gösterilir.

attributes (or columns)

customer_name	*customer_street*	customer_city
Jones	Main	Harrison
Smith	North	Rye
Curry	North	Rye
Lindsay	Park	Pittsfield

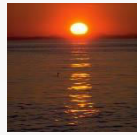
tuples (or rows)

customer



Tabloda kayıt sırası önemli değildir.

- **Tablo kayıt kümesidir. Kümede eleman sırası önemsizdir.**
- Kayıtların ilişkiye/tabloye eklenme sırası ile saklandığı dosyadaki fiziksel sırası aynı olmak zorunda değildir ve genellikle farklıdır.
- Buna göre, aksi söylenmediği sürece, kayıtların tablo içerisinde rastgele bir sırada tutulduğu söylenebilir.
- Bazen kayıtları tablo içinde sıralı tutmak istersek bunu bir SQL komutu (?) ile ayrıca ifade etmemiz gerekir.
- Ama kayıtları her zaman için erişirken sıralayabiliriz. Kayıtları belirli bir sırada listelemek için SELECT komutunda ORDER BY cümleciği kullanılır:
  - SELECT * FROM ogrenci ORDER BY adi, soyadı
- Kayıtları sıralamanın ve tablo içinde sıralı tutmanın sıralama algoritmasından ve/veya indislemeden dolayı zaman/alan maliyeti olduğunu unutmamalıyız.

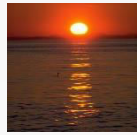


Database (Veritabanı)

- A database consists of multiple relations. Veritabanı ilişki kümesidir.
- Information about an enterprise is broken up into parts, with each relation storing one part of the information. Bir konuyla ilgili bilgi kısımlara (buna veritabanı tasarımı denir) ayrılıp ayrı ilişkilerde saklanır.

  account : stores information about accounts
  depositor : stores information about which customer owns which account
  customer : stores information about customers

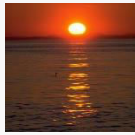
- Storing all information as a single relation such as
  bank(*account_number, balance, customer_name*, ..)
  results in
  - repetition of information
    - e.g.,if two customers own an account (What gets repeated?)
  - the need for null values
    - e.g., to represent a customer without an account
- Tüm bilgileri tek bir ilişkide(tabloda) saklama gereksiz veri tekrarına ve gereksiz null değerlerine sebep olur.
- Normalization theory (Chapter 7) deals with how to design relational schemas



Örnek Uygulama: Universite veritabanı

Veritabanı Şeması (Schema)
ogrenci (ogrNo, adi, soyadi, bNo) – öğrenci kayıtlarını tutar
hoca (hNo, adi, soyadi, bNo) – hoca kayıtlarını tutar
ders (kodu, adi, kredi, bNo) – ders kayıtlarını tutar
bolum (bNo, bAdi) – bolum bilgilerini tutar
dersAl (ogrNo, kodu, not) – hangi öğrencinin hangi dersi aldığı bilgisini tutar
dersVer (hNo, kodu) – hangi hocanın hangi dersi verdiği bilgisini tutar

Veritabanı Şeması
student (sid, fname, lname, did) – öğrenci kayıtlarını tutar
teacher (tid, fname, lname, did) – hoca kayıtlarını tutar
course (code, title, kredits, did) – ders kayıtlarını tutar
department (did, dname) – bolum bilgilerini tutar
take (sid, code, grade) – hangi öğrencinin hangi dersi aldığı bilgisini tutar
teach (tid, code) – hangi hocanın hangi dersi verdiği bilgisini tutar



Universite veritabanı *örneği*

ogrenci			
ogrNo	adi	soyadi	bNo
1	Ali	Kurt	7
2	Ayşe	Yıldız	7
3	Aysel	Demir	6

hoca			
hno	adi	soyadi	bno
10	Hasan	Kurt	7
20	Mehmet	Yıldız	7
30	Ozan	Eren	6

ders			
kodu	adi	kredi	bNo
ce101	Programlama	3	7
Ce201	Veri yapısı	4	7
ce301	Veritabanı	2	7

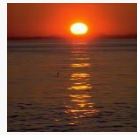
dersAl		
ogrNo	kodu	not
2	ce101	B
1	ce101	A
1	ce301	A

varlık kümesi

ilişki kümesi

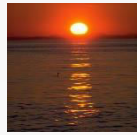
bolum	
bNo	bAdi
6	Bilgisayar
7	Elektrik

dersVer	
kodu	hNo
ce101	10
ce301	20



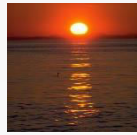
Keys

- Let K $\subseteq$ R
- *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)* *R r* ilişkisinin özellikleri kümesi ve *K* ise *R*'nin bir alt kümesi (K $\subseteq$ R) olmak üzere, eğer K özelliklerinin değerleri olası tüm r ilişkilerinin tekil(biricik, benzersiz) bir kaydını diğerlerinden ayırt etmeye yeterli ise K r için bir süper anahtardır denir. Başka bir ifadeyle, bir ilişkideki tüm olası kayıtları diğerlerinden ayırt edebilen değerlere sahip alanlar kümesine süper anahtar denir.
- Ogrenci(ogrNo, adi, soyadi, dogumTarihi, dogumYeri, tcKimlik, pasaportNo)
  - ogrNo, adi, soyadi
  - ogrNo, adi
  - ogrNo
  - adi, tcKimlik
  - tcKimlik, ogrNo
  - adi, soyadi
  - adi, soyadi, dogumTarihi, dogumTarihi



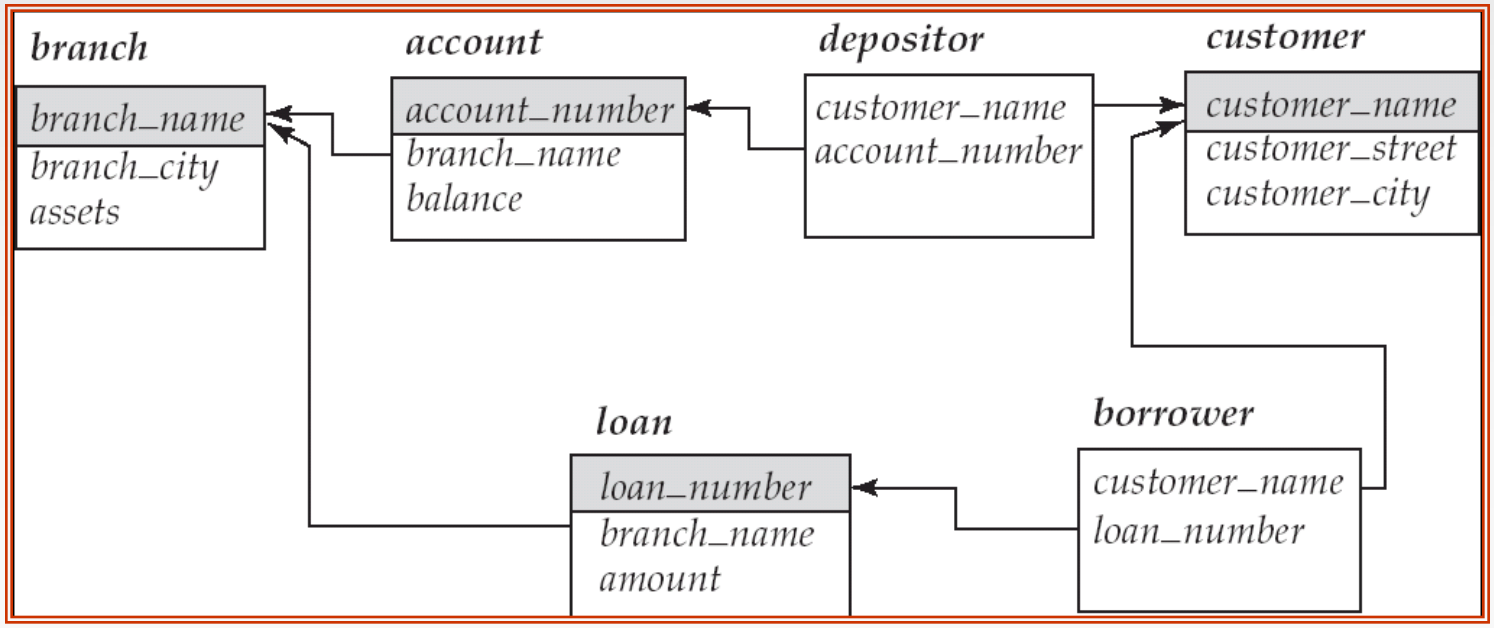
Keys (Cont.)

- *K* is a **candidate key** if *K* is minimal
  Eğer K süper anahtarı minimal (anahtar olmak için gereksiz alan içermiyorsa) ise, K bir aday anahtardır.
  - ogrNo
  - tcKimlik
  - pasaportNo
- **Primary key:** a candidate key chosen as the principal means of identifying tuples within a relation Aday anahtarlardan seçilen bir tanesine birincil anahtar denir. Birincil anahtarlar null değeri alamaz.
  - Should choose an attribute whose value never, or very rarely, changes. Değeri hiç değişmeyecek veya çok nadir değişen alanlar birincil anahtar seçilmelidir.
  - E.g. email address is unique, but may change. Eposta her kayıt için tekil(unique) olmakla birlikte değişebileceği için anahtar seçilmemelidir. Buna rağmen bazı uygulamalarda, uygulamanın özel ihtiyaçlarından dolayı anahtar olarak kullanılabilir.



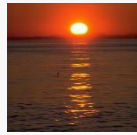
Foreign Keys

- A relation schema may have an attribute that corresponds to the primary key of another relation. The attribute is called a **foreign key**. Bir ilişki başka bir ilişkinin birincil anahtarını içerebilir. Bu durumda bu alana yabancı anahtar denir.
  - Ders-al (ogrNo, dersKodu) ilişkisinde ogrNo ve dersKodu alanları ogrenci ve ders tablolarından gelen yabancı anahtarlardır.
  - Only values occurring in the primary key attribute of the **referenced relation** may occur in the foreign key attribute of the **referencing relation.** Bu durumda yabancı anahtar alanlarında sadece diğer tablonun birincil anahtar alanlarında geçen değerler bulunabilir.
- **Schema diagram**



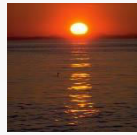
Query Languages

- Language in which user requests information from the database. Kullanıcıların veritabanındaki verilere ulaşmasını sağlayan dile sorgu dili denir.
- Categories of languages Sorgu dili Çeşitleri
  - Procedural İşlemin nasıl yapılacağını adım adım tarif eden diller
  - Non-procedural, or declarative Hangi verinin istendiğinin tarif edildiği tanımsal yani adım'sal olmayan diller
- "Pure" languages Saf yani matematiksel diller:
  - Relational algebra İlişkisel yani tablosal cebir
  - Tuple relational calculus Kayıta dayalı ilişkisel matematik
  - Domain relational calculus Tipe dayalı ilişkisel matematik
- Pure languages form underlying basis of query languages that people use. Matematiksel diller sorgu dillerinin altyapısını oluşturduğundan dolayı sorguların nasıl formulize edildiği, nasıl optimize edildiği ve nasıl çalıştırıldığı yönüyle önemlidir.



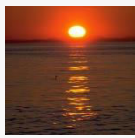
Relational Algebra

- Procedural language Adımsal bir dildir.
- Six basic operators 6 temel işlem
  - select: σ kayıt seçme operatörü
  - project: Π sütun seçme operatörü
  - union: $\cup$ ilişki/küme/tablo bileşimi operatörü
  - set difference: – ilişki/küme/tablo farkı operatörü
  - Cartesian product: x katrezyen çarpım operatörü
  - rename: ρ tablo ve alan isimlendirme operatörü
- The operators take one or two relations as inputs and produce a new relation as a result. Bu operatörler bir veya iki tabloyu giriş olarak alıp, sonuç olarak bir kayıt kümesi yani tablo döndürüler.



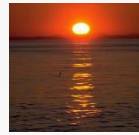
İlişkisel Cebir - SQL SELECT Bağıntısı

İlişkisel Cebir	SELECT komutu
Kayıt seçme $\sigma_{A=B \wedge D > 5}(r)$	WHERE cümleciği WHERE A=B ^ D > 5
Sütun seçme $\Pi_{A,C}(r)$	SELECT cümleciği SELECT A, C
Kartezyen çarpım r x s	FROM cümleciği FROM r, s
Birleşim $r \cup s$	UNION cümleciği r UNION s
Fark r – s	EXCEPT/MINUS cümleciği r EXCEPT s
Tablo ve alan isimlendirme $\rho_{x(a1, a2, \ldots an)}(r)$	FROM/SELECT cümleciği (AS) SELECT b1 AS a1, ... Bn AS an FROM r AS x



İlişkisel Cebir- SQL Bağıntısı

- İlişkisel cebir ve SQL kabaca aynı işi yapar. İkiside sorgu dilidir.
- Birbirlerine ifade gücü bakımından kabaca eşdeğer sayılabilirler.
- İlişkisel cebir ve SQL ifadeleri birbirlerine çevrilebilir.
- SQL komutları çalıştırılırken işlemleri optimize etmek için ve çalıştırmak için ilişkisel cebir kullanılır.
- İlişkisel cebirde ifadelerin sonucu daima bir ilişkidir; bundan dolayı sonuçtaki tekrarlı kayıtlar bir defa listelenir. Buna duplicate elimination denir.
- SQLde ise genellikle sorguların sonucundaki tekrarlı kayıtların hepsi listelenir. Tekrarlı kayıtları bir defa listelemek için SELECT cümleciğinde DISTINCT anahtar kelimesi kullanılır. DISTINCT kullanıldığında kayıtlar sıralanacağı için sonuçların daha geç geleceği bilinmelidir.



Select Operation – Example

- Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- Örne sorgu: r ilişkisinde A ve B alanlarında aynı değere ve D alanında 5'ten büyük değerler içeren kayıtları listeleyiniz.
- $\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Select Operation

- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Defined as:

$$\sigma_p(\boldsymbol{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

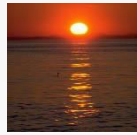
Where p is a formula in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
Each **term** is one of:

<attribute> *op* <attribute> or <constant>

where *op* is one of: $=, \neq, >, \geq, <, \leq$

- Example of selection:

$$\sigma_{branch\_name=\text{“Perryridge”}}(account)$$



Project Operation – Example

- Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$\Pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Project Operation

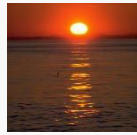
- Notation:

$$\Pi_{A_1, A_2, \ldots, A_k}(r)$$

  where A_1, A_2 are attribute names and r is a relation name.

- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- Example: To eliminate the *branch_name* attribute of *account*

$$\Pi_{account\_number,\ balance}(account)$$



Union Operation – Example

- Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r ∪ s:

A	B
α	1
α	2
β	1
β	3

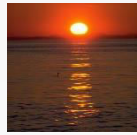
Union Operation

- Notation: $r \cup s$
- Defined as:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- For $r \cup s$ to be valid.
  1. *r, s* must have the *same* **arity** (same number of attributes)
  2. The attribute domains must be **compatible** (example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)
- Example: to find all customers with either an account or a loan

$$\Pi_{customer\_name}(depositor) \cup \Pi_{customer\_name}(borrower)$$



Set Difference Operation – Example

- Relations *r*, *s*:

A	*B*
α	1
α	2
β	1

r

A	*B*
α	2
β	3

s

- *r* – *s*:

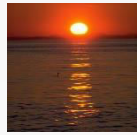
A	*B*
α	1
β	1

Set Difference Operation

- Notation $r - s$
- Defined as:

$$r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$$

- Set differences must be taken between **compatible** relations.
  - r and s must have the same arity
  - attribute domains of r and s must be compatible



Cartesian-Product Operation – Example

- Relations *r, s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r* x *s*:

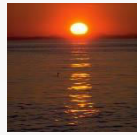
A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Cartesian-Product Operation

- Notation $r \times s$
- Defined as:

$$r \times s = \{t\, q \mid t \in r \textbf{ and } q \in s\}$$

- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \emptyset$).
- If attributes of *r(R)* and *s(S)* are not disjoint, then renaming must be used.



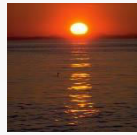
Composition of Operations

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$
- $r \times s$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

- $\sigma_{A=C}(r \times s)$

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b



Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
- Example:

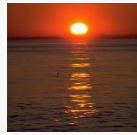
$$\rho_X(E)$$

returns the expression E under the name X

- If a relational-algebra expression E has arity n, then

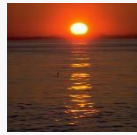
$$\rho_{x(A_1, A_2, \ldots, A_n)}(E)$$

returns the result of expression E under the name X, and with the attributes renamed to $A_1, A_2, \ldots, A_n$.



Alan seçme İşlemi

- Örnek sorgu: Üniversite veritabanındaki öğrencilerin soyadlarını listeleyiniz.
  - $\Pi_{fname,\ lname}$ (*student*) – tekil değerler(kayıtlar ir defa listeleinir
  - SELECT fname, lname FROM student
- Örnek sorgu:



Banking Example

branch (branch_name, branch_city, assets)

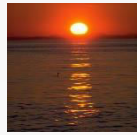
customer (customer_name, customer_street, customer_city)

account (account_number, branch_name, balance)

loan (loan_number, branch_name, amount)

depositor (customer_name, account_number)

borrower (customer_name, loan_number)



Example Queries

- Find all loans of over $1200

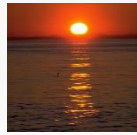
$$\sigma_{amount > 1200}(loan)$$

- Find the loan number for each loan of an amount greater than $1200

$$\Pi_{loan\_number}(\sigma_{amount > 1200}(loan))$$

- Find the names of all customers who have a loan, an account, or both, from the bank

$$\Pi_{customer\_name}(borrower) \cup \Pi_{customer\_name}(depositor)$$



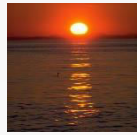
Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.

$$\Pi_{customer\_name}(\sigma_{branch\_name=\text{“Perryridge”}}(\sigma_{borrower.loan\_number = loan.loan\_number}(borrower \times loan)))$$

- Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

$$\Pi_{customer\_name}(\sigma_{branch\_name = \text{“Perryridge”}}(\sigma_{borrower.loan\_number = loan.loan\_number}(borrower \times loan))) - \Pi_{customer\_name}(depositor)$$



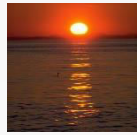
Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.
  - Query 1

$$\Pi_{customer\_name}(\sigma_{branch\_name = \text{“Perryridge”}}(\sigma_{borrower.loan\_number = loan.loan\_number}(borrower \times loan)))$$

  - Query 2

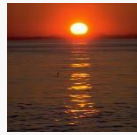
$$\Pi_{customer\_name}(\sigma_{loan.loan\_number = borrower.loan\_number}((\sigma_{branch\_name = \text{“Perryridge”}}(loan)) \times borrower))$$



Example Queries

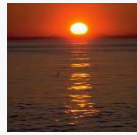
- Find the largest account balance
  - Strategy:
    - Find those balances that are *not* the largest
      - Rename *account* relation as *d* so that we can compare each account balance with all others
    - Use set difference to find those account balances that were *not* found in the earlier step.
  - The query is:

$$\Pi_{balance}(account) - \Pi_{account.balance}$$
$$(\sigma_{account.balance < d.balance} (account \times \rho_{d} (account)))$$



Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p(E_1)$, P is a predicate on attributes in E_1
  - $\Pi_s(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_x(E_1)$, x is the new name for the result of E_1

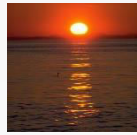


Additional Operations (Ek İşlemler)

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

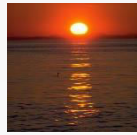
Be 4 ek işlem ilişkisel cebirin ifade gücünü arttırmasa da çoğu sorguları basitlerştirdikleri için tanımlanmıştır.

- Set intersection İlişki/küme/tablo kesişimi
- Natural join Doğal kartezyen/biRleşim (ortak alan ve değerle kartezyen çarpım)
- Division İlişki/Küme/tablo Bölümü
- Assignment Atama



Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \textbf{ and } t \in s \}$
- Assume:
  - r, s have the *same arity*
  - attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$



Set-Intersection Operation – Example

- Relation *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

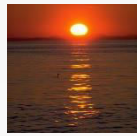
Natural-Join Operation

- Notation: r ⋈ s
- Let *r* and *s* be relations on schemas *R* and *S* respectively.
  Then, r ⋈ s is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from *r* and t_s from *s*.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result, where
    - *t* has the same value as t_r on *r*
    - *t* has the same value as t_s on *s*
- Example:

  $R = (A, B, C, D)$

  $S = (E, B, D)$
  - Result schema = (A, B, C, D, E)
  - $r \bowtie s$ is defined as:

$$\Pi_{r.A,\ r.B,\ r.C,\ r.D,\ s.E}\left(\sigma_{r.B = s.B \wedge r.D = s.D}(r \times s)\right)$$



Natural Join Operation – Example

- Relations r, s:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

s

- r ⋈ s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Division Operation

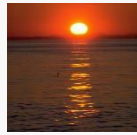
- Notation: $r \div s$
- Suited to queries that include the phrase “for all”.
- Let *r* and *s* be relations on schemas *R* and *S* respectively where
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$

  The result of r ÷ s is a relation on schema

  $R - S = (A_1, \ldots, A_m)$

  $$r \div s = \{ t \mid t \in \prod_{R-S}(r) \wedge \forall u \in s (tu \in r) \}$$

  Where *tu* means the concatenation of tuples *t* and *u* to produce a single tuple



Division Operation – Example

- Relations *r*, *s*:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
$\in$	6
$\in$	1
β	2

r

B
1
2

s

- $r \div s$:

A
α
β

Another Division Example

- Relations *r*, *s*:

A	*B*	*C*	*D*	*E*
α	a	α	a	*1*
α	a	γ	a	*1*
α	a	γ	b	*1*
β	a	γ	a	*1*
β	a	γ	b	*3*
γ	a	γ	a	*1*
γ	a	γ	b	*1*
γ	a	β	b	*1*

r

D	*E*
a	*1*
b	*1*

s

- $r \div s$:

A	*B*	*C*
α	a	γ
γ	a	γ

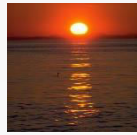
Division Operation (Cont.)

- Property
  - Let $q = r \div s$
  - Then q is the largest relation satisfying $q \times s \subseteq r$
- Definition in terms of the basic algebra operation
  Let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

To see why

- $\Pi_{R-S,S}(r)$ simply reorders attributes of r
- $\Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$ gives those tuples t in $\Pi_{R-S}(r)$ such that for some tuple $u \in s$, $tu \notin r$.

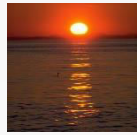


Assignment Operation

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries.
  - Write query as a sequential program consisting of
    - a series of assignments
    - followed by an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a temporary relation variable.
- Example: Write $r \div s$ as

$$temp1 \leftarrow \prod_{R-S} (r)$$
$$temp2 \leftarrow \prod_{R-S} ((temp1 \times s) - \prod_{R-S,S} (r))$$
$$result = temp1 - temp2$$

  - The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.
  - May use variable in subsequent expressions.



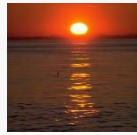
Bank Example Queries

- Find the names of all customers who have a loan and an account at bank.

$$\Pi_{customer\_name}(borrower) \cap \Pi_{customer\_name}(depositor)$$

- Find the name of all customers who have a loan at the bank and the loan amount

$$\Pi_{customer\_name,\ loan\_number,\ amount}(borrower \bowtie loan)$$



Bank Example Queries

- Find all customers who have an account from at least the “Downtown” and the Uptown” branches.
  - Query 1

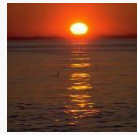
$$\Pi_{customer\_name}(\sigma_{branch\_name = \text{“Downtown”}}(depositor \bowtie account)) \cap$$

$$\Pi_{customer\_name}(\sigma_{branch\_name = \text{“Uptown”}}(depositor \bowtie account))$$

  - Query 2

$$\Pi_{customer\_name,\ branch\_name}(depositor \bowtie account) \div \rho_{temp(branch\_name)}(\{(\text{“Downtown”}), (\text{“Uptown”})\})$$

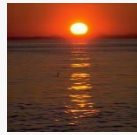
Note that Query 2 uses a constant relation.



Bank Example Queries

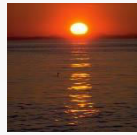
- Find all customers who have an account at all branches located in Brooklyn city.

$$\Pi_{customer\_name,\ branch\_name}\ (depositor \bowtie account)$$
$$\div \Pi_{branch\_name}\ (\sigma_{branch\_city\ =\ \text{“Brooklyn”}}\ (branch))$$



Extended Relational-Algebra-Operations
Genişletilmiş İlişkisel Cebir İşlemleri

- Generalized Projection Genelleşmiş sütun seçme
- Aggregate Functions Grup Fonksiyonları
- Outer Join Dış doğal kartezyen (dış birleşim)



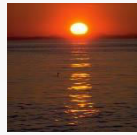
Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list.

$$\Pi_{F_1, F_2, \ldots, F_n}(E)$$

- E is any relational-algebra expression
- Each of F_1, F_2, …, F_n are are arithmetic expressions involving constants and attributes in the schema of E.
- Given relation *credit_info(customer_name, limit, credit_balance),* find how much more each person can spend:

$$\Pi_{customer\_name,\ limit - credit\_balance}(credit\_info)$$



Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.

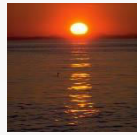
  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **Aggregate operation** in relational algebra

$$_{G_1,G_2,\ldots,G_n}\vartheta_{F_1(A_1),F_2(A_2,\ldots,F_n(A_n)}(E)$$

  E is any relational-algebra expression

  - G_1, G_2 …, G_n is a list of attributes on which to group (can be empty)
  - Each F_i is an aggregate function
  - Each A_i is an attribute name



Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

- $g_{\text{sum(c)}}(\text{r})$

sum(*c*)
27

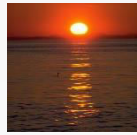
Aggregate Operation – Example

- Relation *account* grouped by *branch-name*:

branch_name	*account_number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$$branch\_name\, \mathcal{G}_{\mathbf{sum}(balance)}\, (account)$$

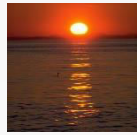
branch_name	**sum**(*balance*)
Perryridge	1300
Brighton	1500
Redwood	700



Aggregate Functions (Cont.)

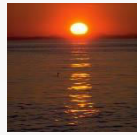
- Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation

$$branch\_name\, \mathcal{G}\ \textbf{sum}(balance)\ \textbf{as}\ sum\_balance\,(account)$$



Outer Join (Dış doğal kartezyen)

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
  - All comparisons involving *null* are (roughly speaking) **false** by definition.
    - We shall study precise meaning of comparisons with nulls later



Outer Join – Example

- Relation *loan*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

- Relation *borrower*

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

Outer Join – Example

- Join

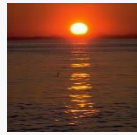
  loan ⋈ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

- Left Outer Join

  loan ⟕ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*



Outer Join – Example

- Right Outer Join

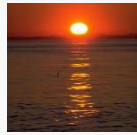
  loan ⟖ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

- Full Outer Join

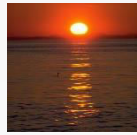
  loan ⟗ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes



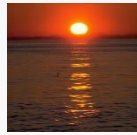
Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null.*
- Aggregate functions simply ignore null values (as in SQL)
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL)



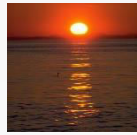
Null Values

- Comparisons with null values return the special truth value: *unknown*
  - If *false* was used instead of *unknown*, then *not (A < 5)* would not be equivalent to *A >= 5*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: (*true* **and** *unknown*) = *unknown,*
    (*false* **and** *unknown*) = *false,*
    (*unknown* **and** *unknown*) = *unknown*
  - NOT: (**not** *unknown*) = *unknown*
  - In SQL "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
- Result of select predicate is treated as *false* if it evaluates to *unknown*



Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations are expressed using the assignment operator.

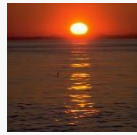


Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- Can delete only whole tuples; cannot delete values on only particular attributes
- A deletion is expressed in relational algebra by:

$$r \leftarrow r - E$$

where r is a relation and E is a relational algebra query.



Deletion Examples

- Delete all account records in the Perryridge branch.

$$account \leftarrow account - \sigma_{branch\_name = \text{“Perryridge”}}(account)$$

- Delete all loan records with amount in the range of 0 to 50

$$loan \leftarrow loan - \sigma_{amount \geq 0 \text{ and } amount \leq 50}(loan)$$

- Delete all accounts at branches located in Needham.

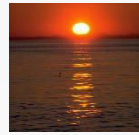
$$r_1 \leftarrow \sigma_{branch\_city = \text{“Needham”}}(account \bowtie branch)$$
$$r_2 \leftarrow \Pi_{account\_number,\ branch\_name,\ balance}(r_1)$$
$$r_3 \leftarrow \Pi_{customer\_name,\ account\_number}(r_2 \bowtie depositor)$$
$$account \leftarrow account - r_2$$
$$depositor \leftarrow depositor - r_3$$

Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted
- in relational algebra, an insertion is expressed by:

$$r \leftarrow r \cup E$$

  where r is a relation and E is a relational algebra expression.
- The insertion of a single tuple is expressed by letting E be a constant relation containing one tuple.



Insertion Examples

- Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.

$$account \leftarrow account \cup \{(\text{“A-973”}, \text{“Perryridge”}, 1200)\}$$

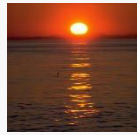
$$depositor \leftarrow depositor \cup \{(\text{“Smith”}, \text{“A-973”})\}$$

- Provide as a gift for all loan customers in the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account.

$$r_1 \leftarrow (\sigma_{branch\_name = \text{“Perryridge”}}(borrower \bowtie loan))$$

$$account \leftarrow account \cup \prod_{loan\_number,\ branch\_name,\ 200}(r_1)$$

$$depositor \leftarrow depositor \cup \prod_{customer\_name,\ loan\_number}(r_1)$$

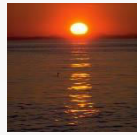


Updating

- A mechanism to change a value in a tuple without charging *all* values in the tuple
- Use the generalized projection operator to do this task

$$r \leftarrow \prod_{F_1, F_2, \ldots, F_l}(r)$$

- Each F_i is either
  - the I^{th} attribute of r, if the I^{th} attribute is not updated, or,
  - if the attribute is to be updated F_i is an expression, involving only constants and the attributes of r, which gives the new value for the attribute



Update Examples

- Make interest payments by increasing all balances by 5 percent.

$$account \leftarrow \prod_{account\_number,\ branch\_name,\ balance\ *\ 1.05} (account)$$

- Pay all accounts with balances over $10,000 6 percent interest and pay all others 5 percent

$$account \leftarrow \prod_{account\_number,\ branch\_name,\ balance\ *\ 1.06} (\sigma_{BAL > 10000} (account)) \cup \prod_{account\_number,\ branch\_name,\ balance\ *\ 1.05} (\sigma_{BAL \leq 10000} (account))$$

Das Bild kann zurzeit nicht angezeigt werden.

End of Chapter 2

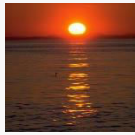


Figure 2.3. The *branch* relation

branch_name	*branch_city*	*assets*
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

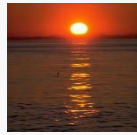


Figure 2.6: The *loan* relation

loan_number	*branch_name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

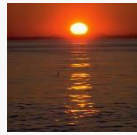


Figure 2.7: The *borrower* relation

customer_name	*loan_number*
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

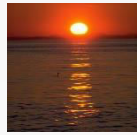


Figure 2.9

Result of $\sigma_{branch\_name = \text{“Perryridge”}}$ (*loan*)

loan_number	*branch_name*	*amount*
L-15	Perryridge	1500
L-16	Perryridge	1300

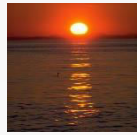


Figure 2.10: Loan number and the amount of the loan

loan_number	*amount*
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1000
L-23	2000
L-93	500

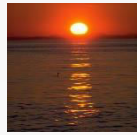


Figure 2.11: Names of all customers who have either an account or an loan

customer_name
Adams
Curry
Hayes
Jackson
Jones
Smith
Williams
Lindsay
Johnson
Turner

Figure 2.12:
Customers with an account but no loan

customer_name
Johnson
Lindsay
Turner

Figure 2.13: Result of *borrower* |X| *loan*

customer_name	*borrower. loan_number*	*loan. loan_number*	*branch_name*	*amount*
Adams	L-16	L-11	Round Hill	900
Adams	L-16	L-14	Downtown	1500
Adams	L-16	L-15	Perryridge	1500
Adams	L-16	L-16	Perryridge	1300
Adams	L-16	L-17	Downtown	1000
Adams	L-16	L-23	Redwood	2000
Adams	L-16	L-93	Mianus	500
Curry	L-93	L-11	Round Hill	900
Curry	L-93	L-14	Downtown	1500
Curry	L-93	L-15	Perryridge	1500
Curry	L-93	L-16	Perryridge	1300
Curry	L-93	L-17	Downtown	1000
Curry	L-93	L-23	Redwood	2000
Curry	L-93	L-93	Mianus	500
Hayes	L-15	L-11		900
Hayes	L-15	L-14		1500
Hayes	L-15	L-15		1500
Hayes	L-15	L-16		1300
Hayes	L-15	L-17		1000
Hayes	L-15	L-23		2000
Hayes	L-15	L-93		500
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
Smith	L-23	L-11	Round Hill	900
Smith	L-23	L-14	Downtown	1500
Smith	L-23	L-15	Perryridge	1500
Smith	L-23	L-16	Perryridge	1300
Smith	L-23	L-17	Downtown	1000
Smith	L-23	L-23	Redwood	2000
Smith	L-23	L-93	Mianus	500
Williams	L-17	L-11	Round Hill	900
Williams	L-17	L-14	Downtown	1500
Williams	L-17	L-15	Perryridge	1500
Williams	L-17	L-16	Perryridge	1300
Williams	L-17	L-17	Downtown	1000
Williams	L-17	L-23	Redwood	2000
Williams	L-17	L-93	Mianus	500

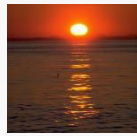


Figure 2.14

customer_name	*borrower. loan_number*	*loan. loan_number*	*branch_name*	*amount*
Adams	L-16	L-15	Perryridge	1500
Adams	L-16	L-16	Perryridge	1300
Curry	L-93	L-15	Perryridge	1500
Curry	L-93	L-16	Perryridge	1300
Hayes	L-15	L-15	Perryridge	1500
Hayes	L-15	L-16	Perryridge	1300
Jackson	L-14	L-15	Perryridge	1500
Jackson	L-14	L-16	Perryridge	1300
Jones	L-17	L-15	Perryridge	1500
Jones	L-17	L-16	Perryridge	1300
Smith	L-11	L-15	Perryridge	1500
Smith	L-11	L-16	Perryridge	1300
Smith	L-23	L-15	Perryridge	1500
Smith	L-23	L-16	Perryridge	1300
Williams	L-17	L-15	Perryridge	1500
Williams	L-17	L-16	Perryridge	1300

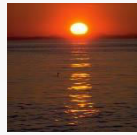
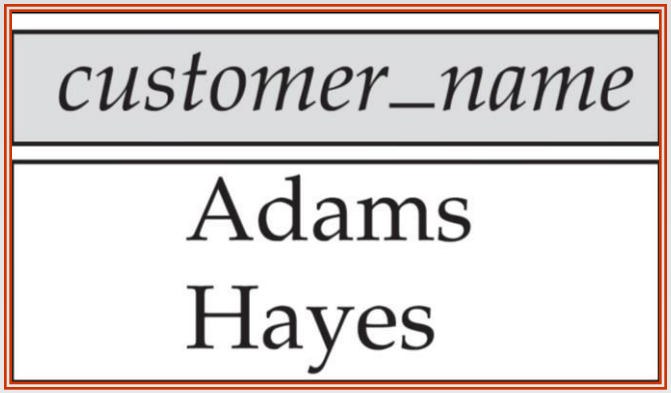


Figure 2.15



customer_name
Adams
Hayes

Figure 2.16

balance
500
400
700
750
350

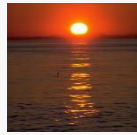


Figure 2.17
Largest account balance in the bank

balance
900

Figure 2.18: Customers who live on the same street and in the same city as Smith

customer_name
Curry
Smith

Figure 2.19: Customers with both an account and a loan at the bank

customer_name
Hayes
Jones
Smith

Figure 2.20

customer_name	*loan_number*	*amount*
Adams	L-16	1300
Curry	L-93	500
Hayes	L-15	1500
Jackson	L-14	1500
Jones	L-17	1000
Smith	L-23	2000
Smith	L-11	900
Williams	L-17	1000

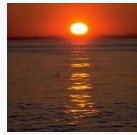


Figure 2.21

branch_name
Brighton
Perryridge

Figure 2.22

branch_name
Brighton
Downtown

Figure 2.23

customer_name	*branch_name*
Hayes	Perryridge
Johnson	Downtown
Johnson	Brighton
Jones	Brighton
Lindsay	Redwood
Smith	Mianus
Turner	Round Hill

Figure 2.24: The *credit_info* relation

customer_name	*limit*	*credit_balance*
Curry	2000	1750
Hayes	1500	1500
Jones	6000	700
Smith	2000	400

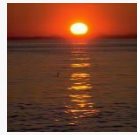


Figure 2.25

customer_name	*credit_available*
Curry	250
Jones	5300
Smith	1600
Hayes	0

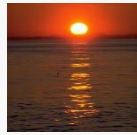


Figure 2.26: The *pt_works* relation

employee_name	*branch_name*	*salary*
Adams	Perryridge	1500
Brown	Perryridge	1300
Gopal	Perryridge	5300
Johnson	Downtown	1500
Loreena	Downtown	1300
Peterson	Downtown	2500
Rao	Austin	1500
Sato	Austin	1600

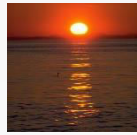


Figure 2.27
The *pt_works* relation after regrouping

employee_name	*branch_name*	*salary*
Rao	Austin	1500
Sato	Austin	1600
Johnson	Downtown	1500
Loreena	Downtown	1300
Peterson	Downtown	2500
Adams	Perryridge	1500
Brown	Perryridge	1300
Gopal	Perryridge	5300

Figure 2.28

branch_name	*sum of salary*
Austin	3100
Downtown	5300
Perryridge	8100

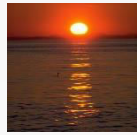


Figure 2.29

branch_name	*sum_salary*	*max_salary*
Austin	3100	1600
Downtown	5300	2500
Perryridge	8100	5300

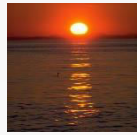


Figure 2.30
The *employee* and *ft_works* relations

employee_name	*street*	*city*
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Death Valley
Williams	Seaview	Seattle

employee_name	*branch_name*	*salary*
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

Figure 2.31

employee_name	*street*	*city*	*branch_name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500

Figure 2.32

employee_name	*street*	*city*	*branch_name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death Valley	*null*	*null*

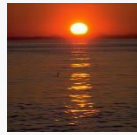


Figure 2.33

employee_name	*street*	*city*	*branch_name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Gates	*null*	*null*	Redmond	5300

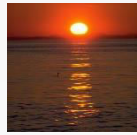


Figure 2.34

employee_name	*street*	*city*	*branch_name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death Valley	*null*	*null*
Gates	*null*	*null*	Redmond	5300