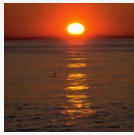
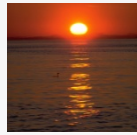
Chapter 2: Relational Model

Database System Concepts, 5th Ed.



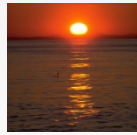
Chapter 2: Relational Model

- Structure of Relational Databases
- Fundamental Relational-Algebra-Operations
- Additional Relational-Algebra-Operations
- Extended Relational-Algebra-Operations
- Null Values
- Modification of the Database



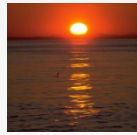
Example of a Relation

account_number	*branch_name*	*balance*
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350



Attribute Types

- Each attribute of a relation has a name
- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
  - 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
- The special value *null* is a member of every domain
- The null value causes complications in the definition of many operations
  - We shall ignore the effect of null values in our main presentation and consider their effect later

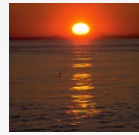


Relation Schema

- Formally, given domains D_1, D_2, …. D_n a **relation** r is a subset of

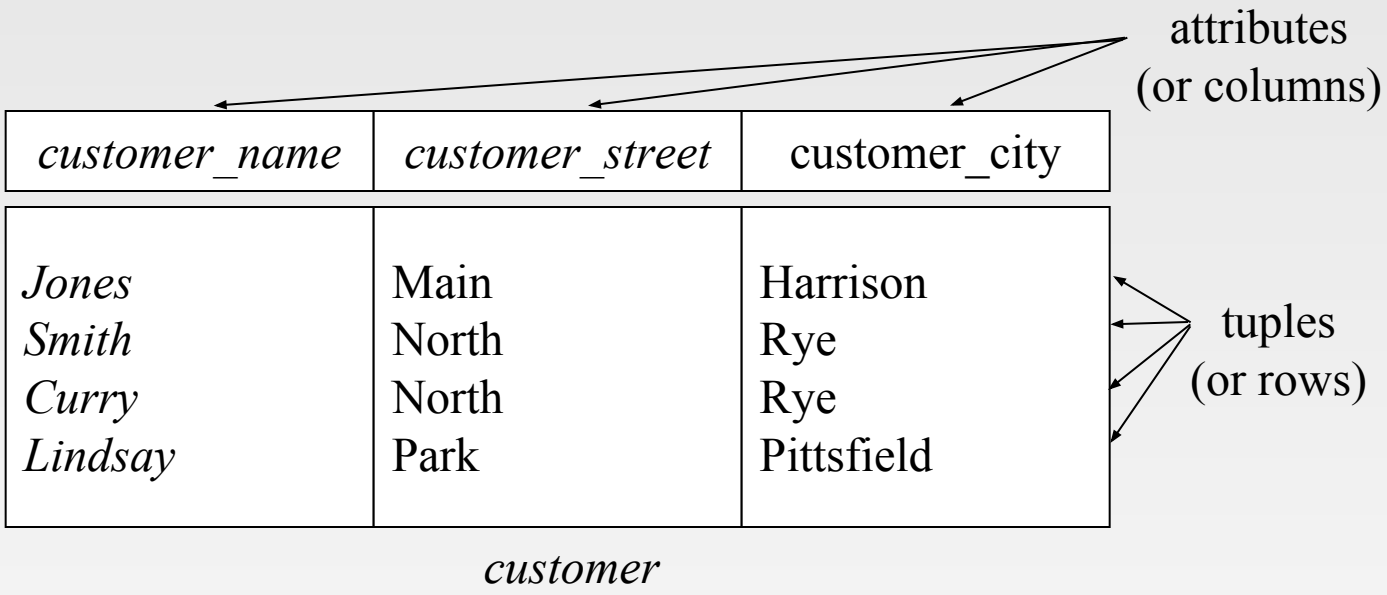
  D_1 x D_2 x … x D_n

  Thus, a relation is a set of n-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$
- Schema of a relation consists of
  - attribute definitions
    - name
    - type/domain
  - integrity constraints



Relation Instance

- The current values (*relation instance*) of a relation are specified by a table
- An element *t* of *r* is a *tuple*, represented by a *row* in a table
- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)



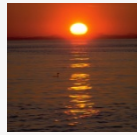
Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts, with each relation storing one part of the information
- E.g.

  account : information about accounts

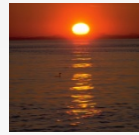
  depositor : which customer owns which account

  customer : information about customers

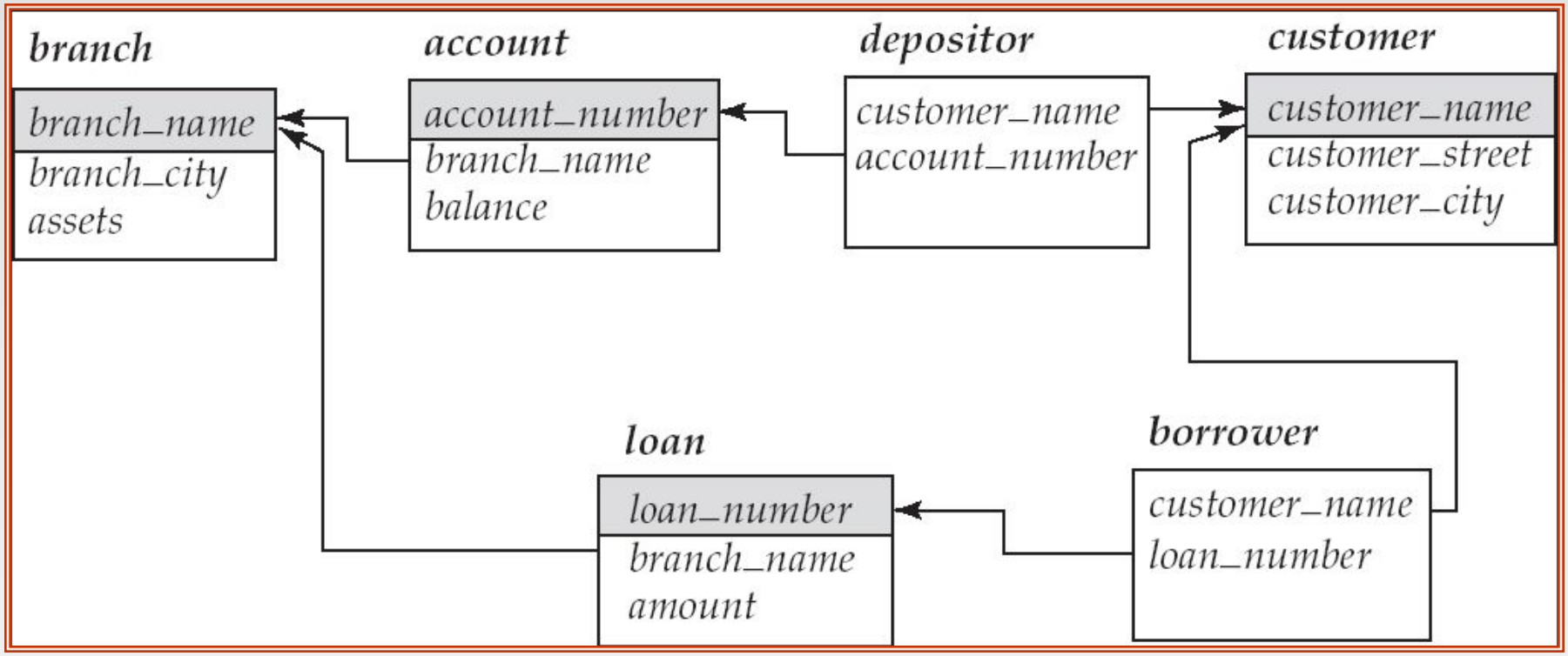


The *customer* Relation

customer_name	*customer_street*	*customer_city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

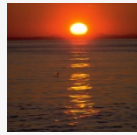


Schema Diagram



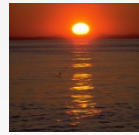
Query Languages

- Language in which user requests information from the database.
- Categories of languages
  - Procedural
  - Non-procedural, or declarative
- "Pure" languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- Pure languages form underlying basis of query languages that people use.



Relational Algebra

- Procedural language
- Six basic operators
  - select: σ
  - project: Π
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ
- The operators take one or two relations as inputs and produce a new relation as a result.



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*

- $\sigma_{A=B \wedge D > 5}(r)$

A	*B*	*C*	*D*
α	α	*1*	*7*
β	β	*23*	*10*

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

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

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

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*

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:

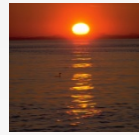
$$\rho_x(E)$$

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 $A_1, A_2, \ldots, A_n$.



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

Banking Example

branch (branch_name, branch_city, assets)

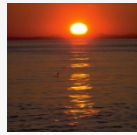
customer (customer_name, customer_street, customer_city)

account (account_number, branch_name, balance)

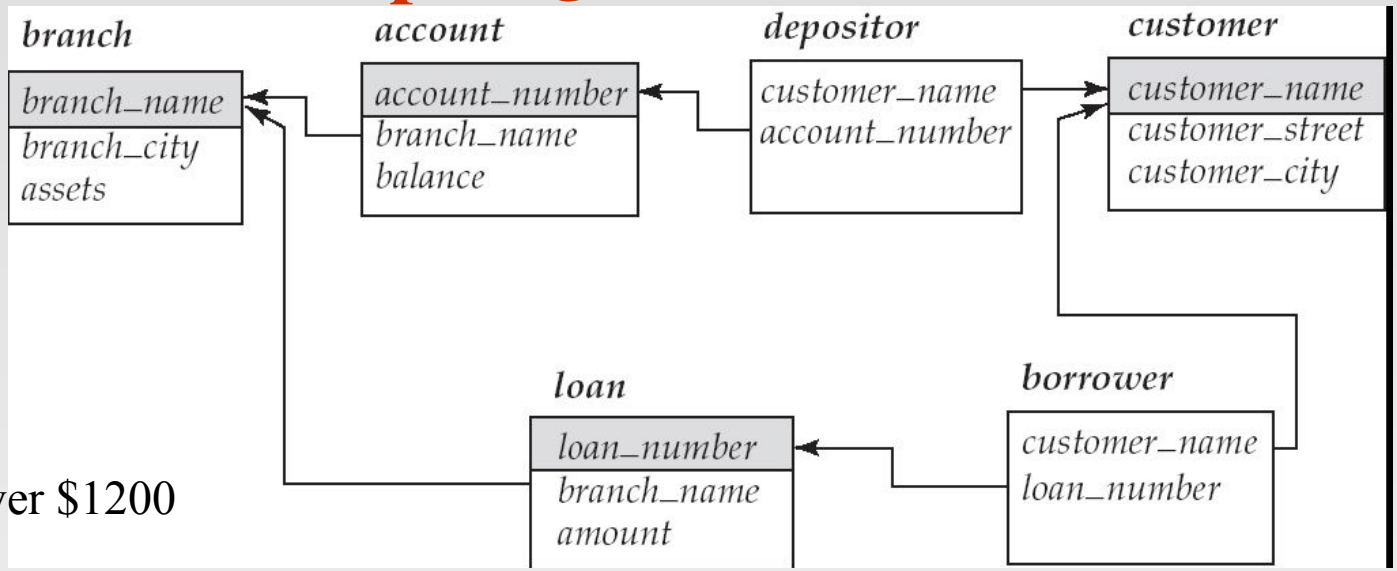
loan (loan_number, branch_name, amount)

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

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

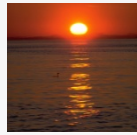
Example Queries

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

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

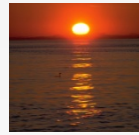
- Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

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



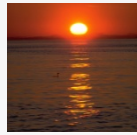
Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.
  - $\Pi_{customer\_name} (\sigma_{branch\_name = \text{“Perryridge”}} (\sigma_{borrower.loan\_number = loan.loan\_number} (borrower \times loan)))$
  - $\Pi_{customer\_name} (\sigma_{loan.loan\_number = borrower.loan\_number} ((\sigma_{branch\_name = \text{“Perryridge”}} (loan)) \times borrower))$



Additional Operations

- Additional Operations
  - Set intersection
  - Natural join
  - Aggregation
  - Outer Join
  - Division
- All above, other than aggregation, can be expressed using basic operations we have seen earlier



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 – 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

Natural-Join Operation

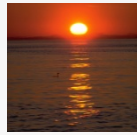
- Notation: r ⋈ s
- Let r and s be relations on schemas R and S respectively.
  Then, r ⋈ s is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from r and t_s from s.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple t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

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



Aggregate Functions and Operations

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

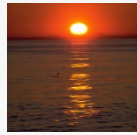
  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **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

  - $G_1, G_2 \ldots, G_n$ 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

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

sum(c)
27

- Question: Which aggregate operations cannot be expressed using basic relational operations?

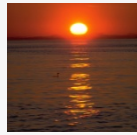
Aggregate Operation – Example

- 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

$$branch\_name\ g\ \mathbf{sum}(balance)\ (account)$$

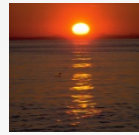
branch_name	**sum**(*balance*)
Perryridge	1300
Brighton	1500
Redwood	700



Aggregate Functions (Cont.)

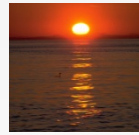
- Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation

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

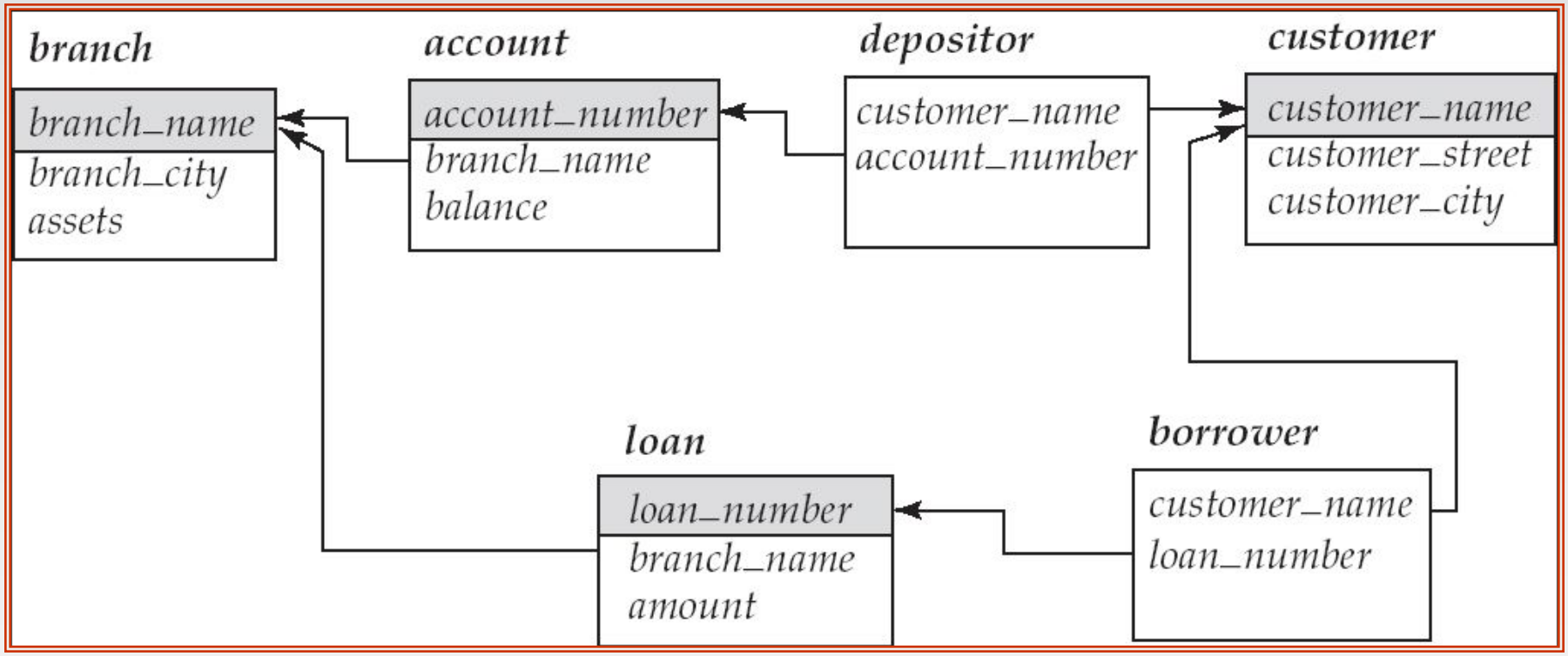


Null Values

- 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)



Schema Diagram



Natural Join ($\bowtie$):

The natural Join is a binary operation that allows us to combine certain operations like selections and a cartesian product.

Q: Find the names of all customers who have a loan at the bank, along with loan number and the loan amount.

$$\Pi_{cname,\ loan.loanno,\ amt}\left(\sigma_{borrower.loanno = loan.loanno}\left(borrower \times loan\right)\right)$$

Cartesian product

Selection

⇩ Transformed with natural Join as below.

$$\Pi_{cname,\ loanno,\ amt}\left(borrower \bowtie loan\right)$$

Conclusion ⇒ $r \bowtie_\theta s = \sigma_\theta (r \times s)$

θ :- Theta Join

θ: predicate.

Q. Find all customers who have both a loan and an account at the bank.

Ans:- By using set Intersection ($\cap$)

$$\Pi_{cname}(borrower) \cap \Pi_{cname}(depositor)$$

Ans:- By using Natural Join operation ($\bowtie$)

$$\Pi_{cname}(borrower \bowtie depositor)$$

Q. Find the names of all branches with customers who have an account in the bank and who live in Harrison.

Ans: $\Pi_{bname}(\sigma_{customer\_city = "Harisson"}(customer \bowtie account \bowtie depositor))$

$(Customer \bowtie account) \bowtie depositor$

$(customer \bowtie (account \bowtie depositor)$

Q Suppose that we wish to find all customers who have an account at all the branches located in Brooklyn city.

step 1 : We can obtain all branches in brooklyn city by the expression

$$r_1 = \Pi_{bname}(\sigma_{bcity = \text{"brooklyn"}}(branch))$$

sample o/p ⇒

bname
brighton
downtown

step 2 : We can find all (customername, branchname) pairs for which customer has an account at a branch by following expression.

$$r_2 = \Pi_{cname,\ bname}(depositor \bowtie account)$$

The sample output of r_2 is

cname	bname
Hayes	perryside
Johnson	downtown
Johnson	brighton
Jones	brighton
Lindsay	Redwood
Smith	Mianus
Turner	Round Hill

Final Ans is $r_2 \div r_1$

cname
Johnson

Π cname, bname (depositor ⋈ account)
$\div$ Π bname ($\sigma_{b\_city = \text{"brooklyn"}}$ (branch))

$R \div S$

- The division operation is a binary operation.
- It is suited to queries that include the phrase "**for all**".
- Formally let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$; that is, every attribute of schema S is also in schema R. The relation $r \div s$ is a relation on schema $R-S$ [i.e., on the schema containing all attributes of schema R that are not in schema S).

A tuple t is in $r \div s$ if and only if both of 2 conditions hold:

1. t is in $\Pi_{R-S}(r)$
2. For every tuple t_s in s, there is a tuple t_r in r satisfying both of the following:
   a. $t_r[S] = t_s[S]$
   b. $t_r[R-S] = t$

Bank Example Queries

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

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

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

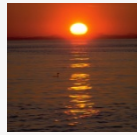
Bank Example Queries

- 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

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

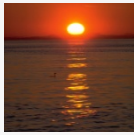
Note that Query 2 uses a constant relation.

Bank Example Queries

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

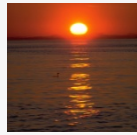
$$\prod_{customer\_name,\ branch\_name} (depositor \bowtie account)$$
$$\div \prod_{branch\_name} (\sigma_{branch\_city = \text{“Brooklyn”}} (branch))$$

End of Chapter 2



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 x E_2
  - $\sigma_p (E_1)$, P is a predicate on attributes in E_1
  - $\prod_s (E_1)$, 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



Select Operation

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

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

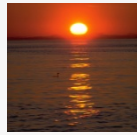
Where p is a formula in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
Each **term** is one of:

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

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

- Example of selection:

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



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*

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

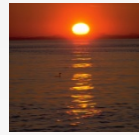
Union Operation

- 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

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

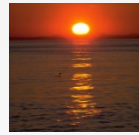


Set Difference Operation

- 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

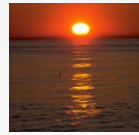


Cartesian-Product Operation

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

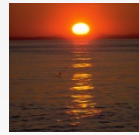
  $r \times s = \{t\, q \mid t \in r \textbf{ and } q \in s\}$

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



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: $r \cap s = r - (r - s)$



Assignment Operation

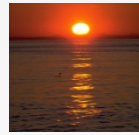
- The assignment operation (←) provides a convenient way to express complex queries.
  - Write query as a sequential program consisting of
    - a series of assignments
    - followed by an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a temporary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

  $temp1 \leftarrow \prod_{R-S} (r)$

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

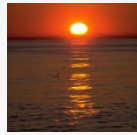
  $result = temp1 - temp2$

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



Extended Relational-Algebra-Operations

- Generalized Projection
- Aggregate Functions
- Outer Join



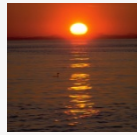
Generalized Projection

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

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

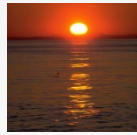
- E is any relational-algebra expression
- Each of $F_1, F_2, \ldots, F_n$ are are arithmetic expressions involving constants and attributes in the schema of E.
- Given relation *credit_info(customer_name, limit, credit_balance),* find how much more each person can spend:

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



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.

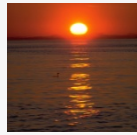


Deletion

- 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.

$$r_1 \leftarrow \sigma_{branch\_city = \text{“Needham”}}(account \bowtie branch)$$

$$r_2 \leftarrow \prod_{account\_number,\ branch\_name,\ balance}(r_1)$$

$$r_3 \leftarrow \prod_{customer\_name,\ account\_number}(r_2 \bowtie depositor)$$

$$account \leftarrow account - r_2$$

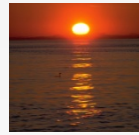
$$depositor \leftarrow depositor - r_3$$

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.



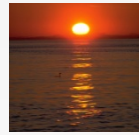
Insertion Examples

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

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

- 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.

$$r_1 \leftarrow (\sigma_{branch\_name = \text{“Perryridge”}}(borrower \bowtie \text{loan}))$$
$$account \leftarrow account \cup \prod_{loan\_number,\ branch\_name,\ 200}(r_1)$$
$$\text{depositor} \leftarrow depositor \cup \prod_{customer\_name,\ loan\_number}(r_1)$$

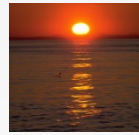


Updating

- 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

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

- 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

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

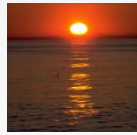


Figure 2.3. The *branch* relation

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

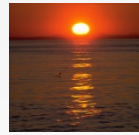


Figure 2.6: The *loan* relation

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

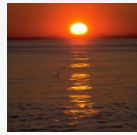


Figure 2.7: The *borrower* relation

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

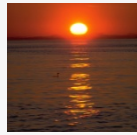


Figure 2.9

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

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

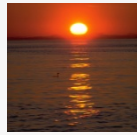


Figure 2.10:
Loan number and the amount of the loan

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

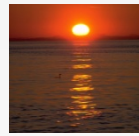


Figure 2.11: Names of all customers who have either an account or an loan

customer_name
Adams
Curry
Hayes
Jackson
Jones
Smith
Williams
Lindsay
Johnson
Turner

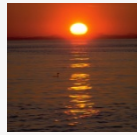


Figure 2.12:
Customers with an account but no loan

customer_name
Johnson
Lindsay
Turner

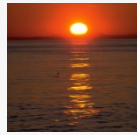


Figure 2.13: Result of *borrower* |X| *loan*

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

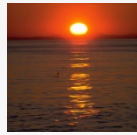


Figure 2.14

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

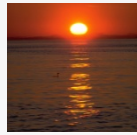


Figure 2.15

customer_name
Adams
Hayes

Figure 2.16

balance
500
400
700
750
350

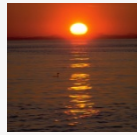


Figure 2.17
Largest account balance in the bank

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

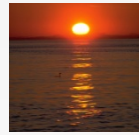


Figure 2.23

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

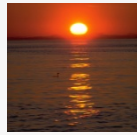


Figure 2.24: The *credit_info* relation

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

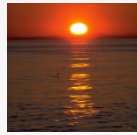


Figure 2.25

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

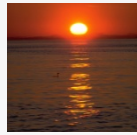


Figure 2.26: The *pt_works* relation

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

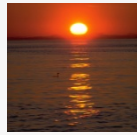


Figure 2.27
The *pt_works* relation after regrouping

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

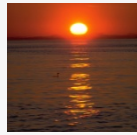


Figure 2.28

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

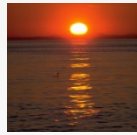


Figure 2.29

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

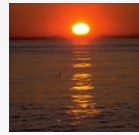


Figure 2.30
The *employee* and *ft_works relations*

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

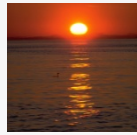


Figure 2.31

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

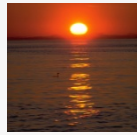


Figure 2.32

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*

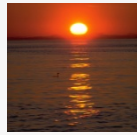


Figure 2.33

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

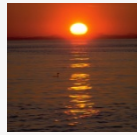


Figure 2.34

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