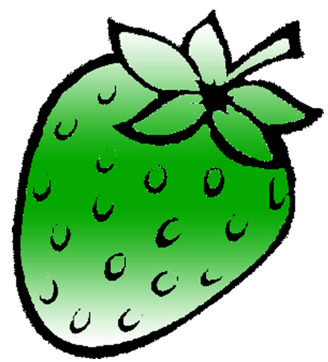
STRAWBERRY



/strawberrydevelopers

/strawberry_app

For more visit:

Strawberrydevelopers.weebly.com

UNIT II – B. **RELATIONAL MODEL**

Agenda

- Structure of Relational Databases
- Relational Algebra
- Extended Relational-Algebra-Operations
- Modification of the Database
- Views
- Tuple Relational Calculus
- Domain Relational Calculus
- Formation of Queries

Relational Model

- Relational Model includes: *Relations, Tuples, Attributes, keys and foreign keys.*
  - **Relation**: A two dimensional table make up of tuples (This is a simple definition that we will define more rigorously in a later chapter).
  - **Tuple**: A row of data in a relation made up of one or more attributes.
  - **Attribute**: A characteristic of the relation contained in a tuple.

Relational Model

attributes

column

SID	SName	SAge	SClass	SSection
1101	Alex	14	9	A
1102	Maria	15	9	A
1103	Maya	14	10	B
1104	Bob	14	9	A
1105	Newton	15	10	B

tuple

table (relation)

A Sample Relational Database

customer-id	*customer-name*	*customer-street*	*customer-city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

account-number	*balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

customer-id	*account-number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

Basic Structure

- Formally, given sets D_1, D_2, …. D_n , a **relation** *r* is a subset of $D_1 \times D_2 \times \ldots \times D_n$
  Thus a relation is a set of n-tuples (a_1, a_2, …, a_n) where each $a_i \in D_i$
- Example: if

  customer-name = {Jones, Smith, Curry, Lindsay}
  customer-street = {Main, North, Park}
  customer-city = {Harrison, Rye, Pittsfield}

  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

- 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. multivalued attribute values are not atomic
  - E.g. composite attribute values are not 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

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

  E.g. *Customer-schema* =
  (*customer-name, customer-street, customer-city*)
- $r(R)$ is a *relation* on the *relation schema R*

  E.g. *customer (Customer-schema)*

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

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

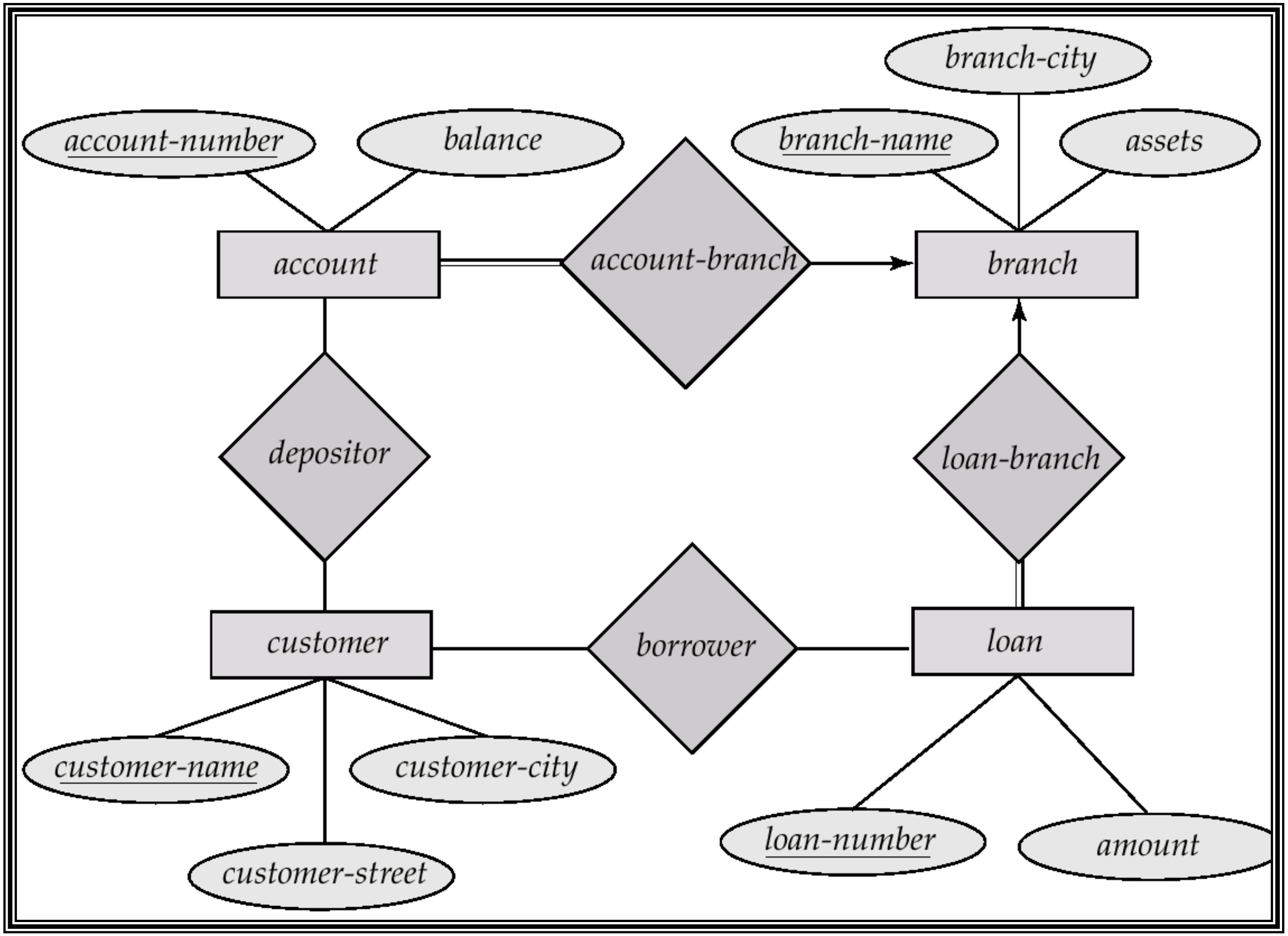
Keys

- 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)*
  - by "possible *r*" we mean a relation *r* that could exist in the enterprise we are modeling.
  - Example: {*customer-name, customer-street*} and {*customer-name*} are both superkeys of *Customer*, if no two customers can possibly have the same name.
- *K* is a ***candidate key*** if *K* is minimal
  Example: {*customer-name*} is a candidate key for *Customer*, since it is a superkey (assuming no two customers can possibly have the same name), and no subset of it is a superkey.

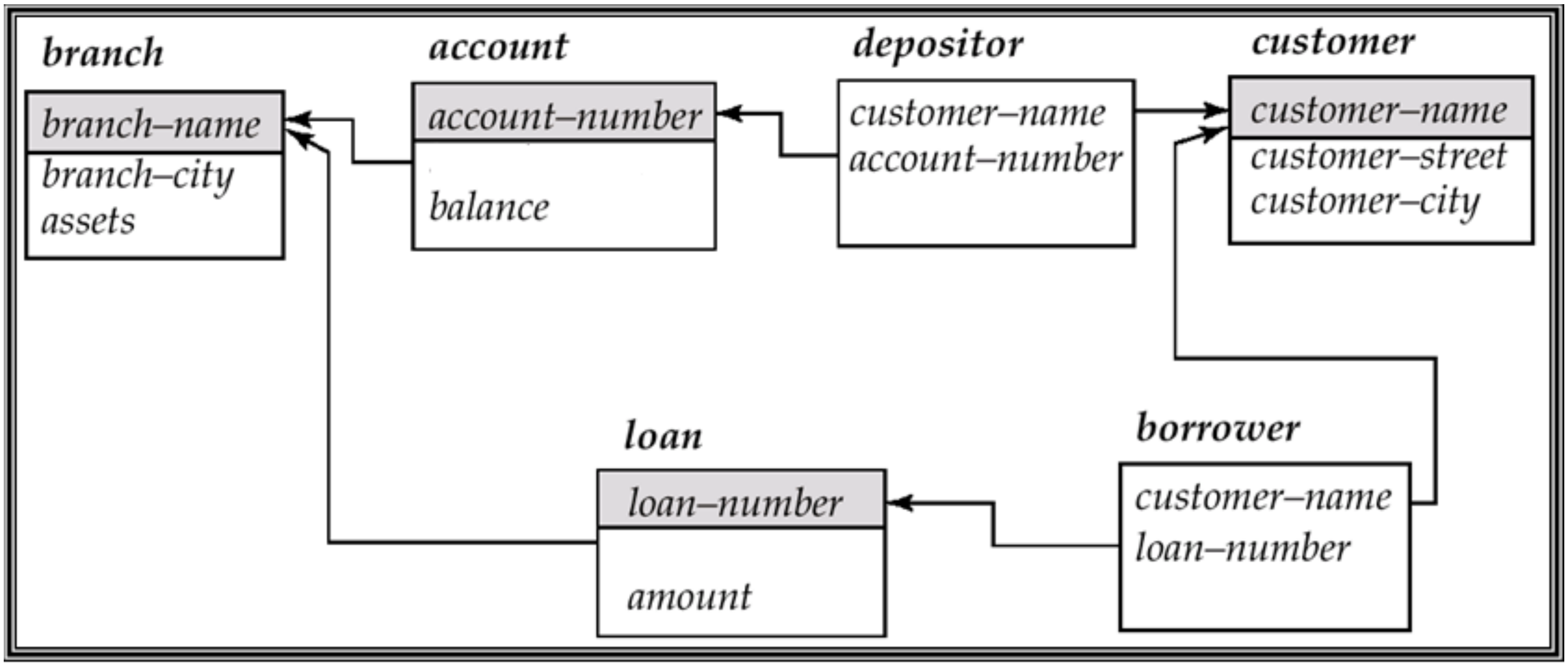
Determining Keys from E-R Sets

- **Strong entity set**. The primary key of the entity set becomes the primary key of the relation.
- **Weak entity set**. The primary key of the relation consists of the union of the primary key of the strong entity set and the discriminator of the weak entity set.
- **Relationship set**. The union of the primary keys of the related entity sets becomes a super key of the relation.
  - For binary many-to-one relationship sets, the primary key of the "many" entity set becomes the relation's primary key.
  - For one-to-one relationship sets, the relation's primary key can be that of either entity set.
  - For many-to-many relationship sets, the union of the primary keys becomes the relation's primary key

E-R Diagram for the Banking Enterprise

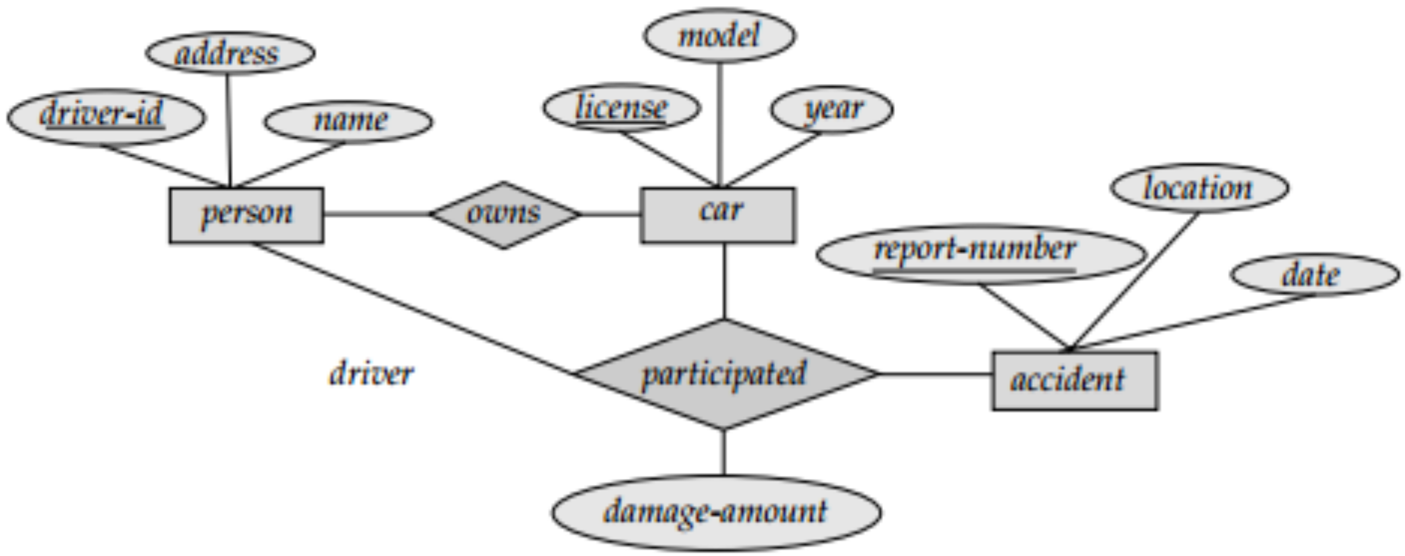


Schema Diagram for the Banking Enterprise



Example

- Design a relational database corresponding to the E-R diagram given below.



- Sol. The relational database schema is given below.
  - person (<u>driver-id</u>, name, address)
  - car (<u>license</u>, year, model)
  - accident (<u>report-number</u>, location, date)
  - owns (<u>driver-id</u>, <u>license</u>)
  - participated (<u>report-number, driver-id, license</u>, damage-amount)

Query Languages

- Language in which user requests information from the database.
- Categories of languages
  - procedural
  - non-procedural
- "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
  - project
  - Union
  - Intersection
  - set difference
  - Cartesian product
  - rename
- The operators take one or more relations as inputs and give a new relation as a result.

Select Operation

Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<SELECTION CONDITION>}}(R)$

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

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

Where *p* is a formula in propositional calculus consisting of terms connected by :

∧ (and), ∨ (or), ¬ (not)

Each term is one of:

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

where *op* is one of: =, ≠, >, ≥. <. ≤

- Example of selection:

$$\sigma_{\textit{branch-name="Perryridge"}}(\textit{account})$$

Select Operation – Example1

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

Select Operation – Example 2 & 3

- Select Electrical Engineers from Employee Relation.
- Sol.

$\sigma_{\text{TITLE='Elect. Eng.'}}(\text{EMP})$

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E6	L. Chu	Elect. Eng.

EMP

	ENO	ENAME	TITLE
☺	E1	J. Doe	Elect. Eng.
	E2	M. Smith	Syst. Anal.
☺	E3	A. Lee	Mech. Eng.
	E4	J. Miller	Programmer
	E5	B. Casey	Syst. Anal.
☺	E6	L. Chu	Elect. Eng.
☺	E7	R. Davis	Mech. Eng.
	E8	J. Jones	Syst. Anal.

- Select Electrical or mechanical engineers from Employee Relation.
- Sol.

$\sigma_{\text{TITLE='Elect. Eng.'} \vee \text{TITLE='Mech.Eng'}}(\text{EMP})$

Select Operation – Example4

- Find the projects with budget less than equal to $200,000 & greater than $200,000 from the relation PROJ using select operation. Define the relations $PROJ_1$ & $PROJ_2$ based on Budget.

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

- **Sol:** $PROJ_1 = \sigma_{BUDGET_{<=200000}}(PROJ)$

  $PROJ_2 = \sigma_{BUDGET_{>200000}}(PROJ)$

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York

$PROJ_2$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

Select Operation – Example5

- If a new tuple with a BUDGET value of, $600,000 is to be inserted into PROJ of previous example. Define the relations $PROJ_1$, $PROJ_2$ & $PROJ_3$ based on Budget.
- **Sol:** $PROJ_1 = \sigma_{BUDGET_{<=200000}}(PROJ)$

  $PROJ_2 = \sigma_{200000<BUDGET_{<=500000}}(PROJ)$

  $PROJ_3 = \sigma_{BUDGET_{>500000}}(PROJ)$

Select Operation – Example6

- Consider the relation PROJ. Using select operation define the relations $PROJ_1$, $PROJ_2$ & $PROJ_3$ based on Location.

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

- **Sol:** $PROJ_1 = \sigma_{LOC = \text{“Montreal”}}(PROJ)$

  $PROJ_2 = \sigma_{LOC = \text{“New York”}}(PROJ)$

  $PROJ_3 = \sigma_{LOC = \text{“Paris”}}(PROJ)$

Select Operation – Example6 contd.

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal

$PROJ_2$

PNO	PNAME	BUDGET	LOC
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York

$PROJ_3$

PNO	PNAME	BUDGET	LOC
P4	Maintenance	310000	Paris

Company Database Schema

Figure Schema diagram for the COMPANY relational database schema; the primary keys are underlined.

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

P.Query2

P.Query3

U.Query3

Select Operation – Example7 & 8

- Select the EMPLOYEE tuples whose department number is four.

Sol. $\sigma_{DNO = 4}$ **(EMPLOYEE)**

- Select the EMPLOYEE tuples whose salary is greater than $30,000.

Sol. $\sigma_{SALARY > 30,000}$ **(EMPLOYEE)**

Select Operation – Example9

- Select the EMPLOYEE tuples whose department number is four and whose salary is greater than $25,000 or those employees whose department number is five and whose salary is greater than $30,000.

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
John		Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin		Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia		Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer		Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh		Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce		English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad		Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James		Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

- Sol.

$$\sigma_{(\text{DNO}=4 \text{ AND SALARY}>25000) \text{ OR } (\text{DNO}=5 \text{ AND SALARY}>30000)}(\text{EMPLOYEE})$$

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
Franklin	T	Wong	333445555	1955-12-08	638 Voss,Houston,TX	M	40000	888665555	5
Jennifer		Wallace	987654321	1941-06-20	291 Berry,Bellaire,TX	F	43000	888665555	4
Ramesh		Narayan	666884444	1962-09-15	975 FireOak,Humble,TX	M	38000	333445555	5

Project Operation

PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<ATTRIBUTE LIST>}}(R)$

- Notation:

$$\prod_{A1,\ A2,\ \ldots,\ Ak}(r)$$

  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
- E.g. To eliminate the *branch-name* attribute of *account*

$$\prod_{account\text{-}number,\ balance}(account)$$

Project Operation – Example1

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 – Example2

- List each employee's first and last name and salary from Employee Relation.

Company database schema

Sol. $\pi_{LNAME,\ FNAME,SALARY}$(EMPLOYEE)

LNAME	FNAME	SALARY
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

Project Operation – Example3

- List each employee's sex and salary from Employee Relation.

Company database schema

Sol. $\pi_{SEX,SALARY}(EMPLOYEE)$

SEX	SALARY
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Project Example4

- Select PNO & BUDGET from the relation PROJ.

PROJ

PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

- Sol:

$\pi_{PNO,BUDGET}(PROJ)$

PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000
P5	500000

Selection with Projection Example

- List each employee's first and last name and salary from Employee Relation whose DNO is 5 from the employee relation.

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
John		Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin		Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia		Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer		Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh		Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce		English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad		Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James		Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

- Sol.

$$\pi_{\text{LNAME, FNAME, SALARY}}(\sigma_{\text{DNO}=5}(\text{EMPLOYEE}))$$

FNAME	LNAME	SALARY
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

Union Operation

UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$

- 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* (e.g., 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)
- E.g. to find all customers with either an account or a loan

$$\Pi_{customer\text{-}name}(depositor) \cup \Pi_{customer\text{-}name}(borrower)$$

Union Operation – Example1

Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

$r \cup s$:

A	B
α	1
α	2
β	1
β	3

Union Operation: Example2

- Find STUDENT ∪ INSTRUCTOR.

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

- Sol:

FN	LN
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

Union Operation: Example3

- To retrieve the social security numbers of all employees who either work in department 5 or directly supervise an employee who works in department 5, use the union operation.

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
John		Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin		Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia		Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer		Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh		Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce		English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad		Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James		Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

Sol. **DEP5_EMPS ← $\sigma_{DNO=5}$ (EMPLOYEE)**

RESULT1 ← π_{SSN}(DEP5_EMPS)

RESULT2(SSN) ← $\pi_{SUPERSSN}$(DEP5_EMPS)

RESULT ← RESULT1 ∪ RESULT2

Intersection Operation

INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$

Intersection Operation: Example1

- Find STUDENT ∩ INSTRUCTOR.

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

- Sol:

FN	LN
Susan	Yao
Ramesh	Shah

Set Difference Operation

DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$

- 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

Set Difference Operation – Example1

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 – Example2

- Find (a) STUDENT − INSTRUCTOR

  (b) INSTRUCTOR – STUDENT

STUDENT

FN	LN
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

FNAME	LNAME
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

- Sol: (a)

FN	LN
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(b)

FNAME	LNAME
John	Smith
Ricardo	Browne
Francis	Johnson

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.

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

Composition of Operations

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

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

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

A	B	C	D	E
α	1	α	10	a
β	2	β	20	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

$$\rho_{X(A1, A2, \ldots, An)}(E)$$

returns the result of expression *E* under the name *X*, and with the attributes renamed to *A1, A2, …., An*.

Example1: Queries

- Consider the relational database given below where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries:

employee (<u>person-name</u>, street, city)
works (<u>person-name</u>, company-name, salary)
company (<u>company-name</u>, city)
manages (<u>person-name</u>, manager-name)

a. Find the names of all employees who work for First Bank Corporation.

Sol. $\Pi_{\text{person-name}} (\sigma_{\text{company-name}} = \text{“First Bank Corporation”} (\text{works}))$

b. Find the names and cities of residence of all employees who work for First Bank Corporation.

Sol. $$\Pi_{person\text{-}name,\, city} (employee \bowtie (\sigma_{company\text{-}name = \text{“First Bank Corporation”}} (works)))$$

Example1: Queries contd.

c. Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000 per annu

Sol.

$$\Pi_{person\text{-}name,\ street,\ city}\left(\sigma_{(company\text{-}name\ =\ \text{"First Bank Corporation"}\ \wedge\ salary\ >\ 10000)}\ works \bowtie employee\right)$$

d. Find the names of all employees in this database who live in the same city as the

Sol.

$$\Pi_{person\text{-}name}\ (employee \bowtie works \bowtie company)$$

Example2: Banking Queries

branch (branch-name, branch-city, assets)

customer (customer-name, customer-street, customer-only)

account (account-number, branch-name, balance)

loan (loan-number, branch-name, amount)

depositor (customer-name, account-number)

borrower (customer-name, loan-number)

Example Queries

- Select all loans of over $1200

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

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

$$\Pi_{loan\text{-}number}\ (\sigma_{amount > 1200}\ (loan))$$

Example Queries

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

$$\Pi_{customer\text{-}name}\ (borrower) \cup \Pi_{customer\text{-}name}\ (depositor)$$

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

$$\Pi_{customer\text{-}name}\ (borrower) \cap \Pi_{customer\text{-}name}\ (depositor)$$

Example Queries

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

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

- 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.

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

Example Queries

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

  - Query 1

    $\Pi_{\text{customer-name}}(\sigma_{\text{branch-name = "Perryridge"}}($
    $\sigma_{\text{borrower.loan-number = loan.loan-number}}(\text{borrower x loan})))$

  - Query 2

    $\Pi_{\text{customer-name}}(\sigma_{\text{loan.loan-number = borrower.loan-number}}($
    $(\sigma_{\text{branch-name = "Perryridge"}}(\text{loan})) \text{ x } \text{borrower}))$

Example Queries

Find the largest account balance

- Rename *account* relation as *d*
- The query is:

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

Summary: Relation Algebra

- 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
  - $\prod_s(E_1)$, 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

Additional operations

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

- Natural join
- Division
- Assignment

Natural-Join Operation

- Notation: $r \bowtie s$
- Let *r* and *s* be relations on schemas *R* and *S* respectively. Then, $r \bowtie s$ 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*

Natural Join Operation – Example1

- Example1:

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

  $S = (E, B, D)$

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

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

Natural Join Operation – Example1 contd..

Relations r, s:

r

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

s

B	D	E
1	a	α
~~3~~	~~a~~	~~β~~
1	a	γ
2	b	δ
~~3~~	~~b~~	~~∈~~

$r \bowtie s$

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 – Example2

r_1

Employee	Department
Smith	sales
Black	production
White	production

r_2

Department	Head
production	Mori
sales	Brown

$r_1 \bowtie r_2$

Employee	Department	Head
Smith	sales	Brown
Black	production	Mori
White	production	Mori

Natural Join Operation – Example3

Offences

Code	Date	Officer	Dept	Registartion
143256	25/10/1992	567	75	5694 FR
987554	26/10/1992	456	75	5694 FR
987557	26/10/1992	456	75	6544 XY
630876	15/10/1992	456	47	6544 XY
539856	12/10/1992	567	47	6544 XY

Cars

Registration	Dept	Owner	…
6544 XY	75	Cordon Edouard	…
7122 HT	75	Cordon Edouard	…
5694 FR	75	Latour Hortense	…
6544 XY	47	Mimault Bernard	…

Offences ⋈ Cars

Code	Date	Officer	Dept	Registration	Owner	…
143256	25/10/1992	567	75	5694 FR	Latour Hortense	…
987554	26/10/1992	456	75	5694 FR	Latour Hortense	…
987557	26/10/1992	456	75	6544 XY	Cordon Edouard	…
630876	15/10/1992	456	47	6544 XY	Cordon Edouard	…
539856	12/10/1992	567	47	6544 XY	Mimault Bernard	…

Natural Join Operation – Example4

Paternity

Father	Child
Adam	Cain
Adam	Abel
Abraham	Isaac
Abraham	Ishmael

Maternity

Mother	Child
Eve	Cain
Eve	Seth
Sarah	Isaac
Hagar	Ishmael

Paternity ⋈ Maternity

Father	Child	Mother
Adam	Cain	Eve
Abraham	Isaac	Sarah
Abraham	Ishmael	Hagar

Natural Join Operation – Example5

r_1

Employee	Project
Smith	A
Black	A
White	A

r_2

Project	Head
A	Mori
A	Brown

$r_1 \bowtie r_2$

Employee	Project	Head
Smith	A	Mori
Black	A	Brown
White	A	Mori
Smith	A	Brown
Black	A	Mori
White	A	Brown

- Cartesian product of r1 & r2.

Natural Joins: Can be incomplete – Example6

r_2

Department	Head
production	Mori
purchasing	Brown

r_1

Employee	Department
Smith	sales
Black	production
White	production

$r_1 \bowtie r_2$

Employee	Department	Head
Black	production	Mori
White	production	Mori

Natural Joins: Can be Null – Example7

r_1

Employee	Department
Smith	sales
Black	production
White	production

r_2

Department	Head
marketing	Mori
purchasing	Brown

$r_1 \bowtie r_2$

Employee	Department	Head

Division Operation

$$r \div s$$

- 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 $r \div s$ is a relation on schema

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

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

Division Operation

Relations *r*, *s*:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
ϵ	6
ϵ	1
β	2

r

B
1
2

s

$r \div s$:

A
α
β

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

Example Queries

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

Query 1

$$\Pi_{CN}(\sigma_{BN=\text{“Downtown”}}(depositor \bowtie account)) \cap$$
$$\Pi_{CN}(\sigma_{BN=\text{“Uptown”}}(depositor \bowtie account))$$

where CN denotes customer-name and BN denotes branch-name.

Query 2

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

Example Queries

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

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

Extended Relational-Algebra-Operations

- Generalized Projection
- Outer Join
- Aggregate Functions

Generalized Projection

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

$$\prod_{F1,\ F2,\ \ldots,\ Fn}(E)$$

- *E* is any relational-algebra expression
- Each of F_1, F_2, …, F_n 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:

$$\prod_{customer\text{-}name,\ limit\ -\ credit\text{-}balance}(credit\text{-}info)$$

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

$$_{G1,\, G2,\, \ldots,\, Gn}\, \mathcal{G}\, _{F1(A1),\, F2(A2),\ldots,\, Fn(An)} (E)$$

  — 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

$\mathcal{g}_{\mathbf{sum(c)}}(r)$

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

$_{\text{branch-name}}\,\mathcal{g}\,_{\text{sum(balance)}}\,(\text{account})$

branch-name	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

$$ {}_{\text{branch-name}}\,\mathcal{G}\,_{\textbf{sum}(\text{balance})\ \textbf{as}\ \text{sum-balance}}\ (\text{account}) $$

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that do 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.
    - Will 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

- **Inner 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
  - Is an arbitrary decision. Could have returned null as result instead.
  - We follow the semantics of SQL in its handling of null values
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same
  - Alternative: assume each null is different from each other
  - Both are arbitrary decisions, so we simply follow 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.

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

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

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

- Delete all accounts at branches located in Needham.

$$r_1 \leftarrow \sigma_{\text{branch-city} = \text{“Needham”}}(account \bowtie branch)$$
$$r_2 \leftarrow \prod_{\text{branch-name, account-number, balance}}(r_1)$$
$$r_3 \leftarrow \prod_{\text{customer-name, account-number}}(r_2 \bowtie depositor)$$
$$account \leftarrow account - r_2$$
$$depositor \leftarrow depositor - r_3$$

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.

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

- 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.

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

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

$$r \leftarrow \prod_{F1,\, F2,\, \ldots,\, Fl,} (r)$$

- 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.

$$account \leftarrow \prod_{AN,\ BN,\ BAL\ *\ 1.05} (account)$$

where AN, BN and BAL stand for account-number, branch-name and balance, respectively.

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

$$account \leftarrow \prod_{AN,\ BN,\ BAL\ *\ 1.06} (\sigma_{BAL\ >\ 10000} (account)) \cup \prod_{AN,\ BN,\ BAL\ *\ 1.05} (\sigma_{BAL\ \leq\ 10000} (account))$$

Views

- In some cases, it is not desirable for all users to see the entire logical model (i.e., all the actual relations stored in the database.)
- Consider a person who needs to know a customer's loan number but has no need to see the loan amount. This person should see a relation described, in the relational algebra, by

$$\Pi_{\textit{customer-name, loan-number}}\,(\textit{borrower} \bowtie \textit{loan})$$

- Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a **view**.

View Definition

- A view is defined using the **create view** statement which has the form

  create view *v* **as** <query expression>

  where <query expression> is any legal relational algebra query expression. The view name is represented by *v.*
- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

View Examples

- Consider the view (named *all-customer*) consisting of branches and their customers.

create view all-customer **as**

$$\Pi_{\text{branch-name, customer-name}}(\text{depositor} \bowtie \text{account})$$
$$\cup\ \Pi_{\text{branch-name, customer-name}}(\text{borrower} \bowtie \text{loan})$$

- We can find all customers of the Perryridge branch by writing:

$$\Pi_{\text{customer-name}}$$
$$(\sigma_{\text{branch-name} = \text{“Perryridge”}}(\text{all-customer}))$$

Updates Through View

- Database modifications expressed as views must be translated to modifications of the actual relations in the database.
- Consider the person who needs to see all loan data in the *loan* relation except *amount.* The view given to the person, *branch-loan,* is defined as:

create view *branch-loan* **as**

$$\Pi_{branch\text{-}name,\ loan\text{-}number}(loan)$$

- Since we allow a view name to appear wherever a relation name is allowed, the person may write:

$$branch\text{-}loan \leftarrow branch\text{-}loan \cup \{(\text{“Perryridge”},\ \text{L-37})\}$$

Updates Through Views (Cont.)

- The previous insertion must be represented by an insertion into the actual relation *loan* from which the view *branch-loan* is constructed.
- An insertion into *loan* requires a value for *amount*. The insertion can be dealt with by either.
  - rejecting the insertion and returning an error message to the user.
  - inserting a tuple (“L-37”, “Perryridge”, *null*) into the *loan* relation
- Some updates through views are impossible to translate into database relation updates
  - create view v as $\sigma_{branch\text{-}name\ =\ \text{“Perryridge”}}(account)$)

    $v \leftarrow v \cup$ (L-99, Downtown, 23)
- Others cannot be translated uniquely
  - $all\text{-}customer \leftarrow all\text{-}customer \cup$ {(“Perryridge”, “John”)}
    - Have to choose loan or account, and create a new loan/account number!

Views Defined Using Other Views

- One view may be used in the expression defining another view
- A view relation v_1 is said to *depend directly* on a view relation v_2 if v_2 is used in the expression defining v_1
- A view relation v_1 is said to *depend on* view relation v_2 if either v_1 depends directly to v_2 or there is a path of dependencies from v_1 to v_2
- A view relation v is said to be *recursive* if it depends on itself.

View Expansion

- A way to define the meaning of views defined in terms of other views.
- Let view v_1 be defined by an expression e_1 that may itself contain uses of view relations.
- View expansion of an expression repeats the following replacement step:

  repeat
  Find any view relation v_i in e_1
  Replace the view relation v_i by the expression defining v_i
  until no more view relations are present in e_1

- As long as the view definitions are not recursive, this loop will terminate

Tuple Relational Calculus

- A nonprocedural query language, where each query is of the form

$$\{t \mid P(t)\}$$

- It is the set of all tuples *t* such that predicate *P* is true for *t*
- *t* is a *tuple variable*, *t*[*A*] denotes the value of tuple *t* on attribute *A*
- $t \in r$ denotes that tuple *t* is in relation *r*
- *P* is a *formula* similar to that of the predicate calculus

Predicate Calculus Formula

1. Set of attributes and constants
2. Set of comparison operators: (e.g., $<$, $\leq$, $=$, $\neq$, $>$, $\geq$)
3. Set of connectives: and ($\wedge$), or ($\vee$), not ($\neg$)
4. Implication ($\Rightarrow$): $x \Rightarrow y$, if x is true, then y is true

$$x \Rightarrow y \equiv \neg x \vee y$$

5. Set of quantifiers:
   - $\exists\, t \in r\, (Q(t)) \equiv$ "there exists" a tuple in *t* in relation *r* such that predicate *Q(t)* is true
   - $\forall t \in r\, (Q(t)) \equiv Q$ is true "for all" tuples *t* in relation *r*

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 the *loan-number, branch-name,* and *amount* for loans of over $1200

$$\{t \mid t \in loan \wedge t\,[amount] > 1200\}$$

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

$$\{t \mid \exists\, s \in loan\ (t[loan\text{-}number] = s[loan\text{-}number] \wedge s\,[amount] > 1200)\}$$

Notice that a relation on schema [loan-number] is implicitly defined by the query

Example Queries

- Find the names of all customers having a loan, an account, or both at the bank

$$\{t \mid \exists s \in borrower(\ t[customer\text{-}name] = s[customer\text{-}name]) \vee \exists u \in depositor(\ t[customer\text{-}name] = u[customer\text{-}name])$$

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

$$\{t \mid \exists s \in borrower(\ t[customer\text{-}name] = s[customer\text{-}name]) \wedge \exists u \in depositor(\ t[customer\text{-}name] = u[customer\text{-}name])$$

Example Queries

- Find the names of all customers having a loan at the Perryridge branch

$$\{t \mid \exists s \in borrower(t[customer\text{-}name] = s[customer\text{-}name] \wedge \exists u \in loan(u[branch\text{-}name] = \text{"Perryridge"} \wedge u[loan\text{-}number] = s[loan\text{-}number]))\}$$

- Find the names of all customers who have a loan at the Perryridge branch, but no account at any branch of the bank

$$\{t \mid \exists s \in borrower(t[customer\text{-}name] = s[customer\text{-}name] \wedge \exists u \in loan(u[branch\text{-}name] = \text{"Perryridge"} \wedge u[loan\text{-}number] = s[loan\text{-}number])) \wedge \textbf{not}\ \exists v \in depositor\ (v[customer\text{-}name] = t[customer\text{-}name])\}$$

Example Queries

- Find the names of all customers having a loan from the Perryridge branch, and the cities they live in

$$
\begin{aligned}
\{t \mid \exists s \in loan(&s[branch\text{-}name] = \text{“Perryridge”} \\
&\wedge \exists u \in borrower\ (u[loan\text{-}number] = s[loan\text{-}number] \\
&\qquad \wedge\ t\ [customer\text{-}name] = u[customer\text{-}name]) \\
&\quad \wedge \exists\ v \in customer\ (u[customer\text{-}name] = v[customer\text{-}name] \\
&\qquad\qquad \wedge\ t[customer\text{-}city] = v[customer\text{-}city])))\}
\end{aligned}
$$

Example Queries

- Find the names of all customers who have an account at all branches located in Brooklyn:

$$\{t \mid \exists\ c \in \text{customer}\ (t[\text{customer.name}] = c[\text{customer-name}]) \wedge$$
$$\forall\ s \in \text{branch}(s[\text{branch-city}] = \text{“Brooklyn”} \Rightarrow$$
$$\exists\ u \in \text{account}\ (\ s[\text{branch-name}] = u[\text{branch-name}]$$
$$\wedge\ \exists\ s \in \text{depositor}\ (\ t[\text{customer-name}] = s[\text{customer-name}]$$
$$\wedge\ s[\text{account-number}] = u[\text{account-number}]\ ))\ )\}$$

Safety of Expressions

- It is possible to write tuple calculus expressions that generate infinite relations.
- For example, $\{t \mid \neg\ t \in r\}$ results in an infinite relation if the domain of any attribute of relation *r* is infinite
- To guard against the problem, we restrict the set of allowable expressions to safe expressions.
- An expression $\{t \mid P(t)\}$ in the tuple relational calculus is *safe* if every component of *t* appears in one of the relations, tuples, or constants that appear in *P*
  - NOTE: this is more than just a syntax condition.
    - E.g. $\{ t \mid t[A]=5 \lor$ **true** $\}$ is not safe --- it defines an infinite set with attribute values that do not appear in any relation or tuples or constants in *P*.

Domain Relational Calculus

- A nonprocedural query language equivalent in power to the tuple relational calculus
- Each query is an expression of the form:

$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

  - $x_1, x_2, \ldots, x_n$ represent domain variables
  - P represents a formula similar to that of the predicate calculus

Example Queries

- Find the *loan-number, branch-name,* and *amount* for loans of over $1200

$$\{< l, b, a > \mid < l, b, a > \in loan \wedge a > 1200\}$$

- Find the names of all customers who have a loan of over $1200

$$\{< c > \mid \exists\ l, b, a\ (< c, l > \in borrower \wedge < l, b, a > \in loan \wedge a > 1200)\}$$

- Find the names of all customers who have a loan from the Perryridge branch and the loan amount:

$$\{< c, a > \mid \exists\ l\ (< c, l > \in borrower \wedge \exists b(< l, b, a > \in loan \wedge b = \text{“Perryridge”}))\}$$

or $\{< c, a > \mid \exists\ l\ (< c, l > \in borrower \wedge < l, \text{“Perryridge”}, a > \in loan)\}$

Example Queries

- Find the names of all customers having a loan, an account, or both at the Perryridge branch:

$$
\begin{aligned}
\{< c > \mid \exists\, l\ (\{< c, l > \in \text{borrower} \\
\quad\quad \wedge \exists\, b,a(< l, b, a > \in \text{loan} \wedge b = \text{“Perryridge”})) \\
\quad \vee \exists\, a(< c, a > \in \text{depositor} \\
\quad\quad \wedge \exists\, b,n(< a, b, n > \in \text{account} \wedge b = \text{“Perryridge”}))\}
\end{aligned}
$$

- Find the names of all customers who have an account at all branches located in Brooklyn:

$$
\begin{aligned}
\{< c > \mid \exists\, s, n\ (< c, s, n > \in \text{customer}) \wedge \\
\quad \forall\, x,y,z(< x, y, z > \in \text{branch} \wedge y = \text{“Brooklyn”}) \Rightarrow \\
\quad\quad \exists\, a,b(< x, y, z > \in \text{account} \wedge < c,a > \in \text{depositor})\}
\end{aligned}
$$

Safety of Expressions

$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

is safe if all of the following hold:

1. All values that appear in tuples of the expression are values from *dom*(*P*) (that is, the values appear either in *P* or in a tuple of a relation mentioned in *P*).
2. For every "there exists" subformula of the form $\exists\ x\ (P_1(x))$, the subformula is true if and only if there is a value of x in *dom*(P_1) such that $P_1(x)$ is true.
3. For every "for all" subformula of the form $\forall_x\ (P_1\ (x))$, the subformula is true if and only if $P_1(x)$ is true for all values x from *dom* (P_1).

End of Chapter 3

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

loan-number	*branch-name*	*amount*
L-15	Perryridge	1500
L-16	Perryridge	1300

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

Names of All Customers Who Have Either a Loan or an Account

customer-name
Adams
Curry
Hayes
Jackson
Jones
Smith
Williams
Lindsay
Johnson
Turner

Customers With An Account But No Loan

customer-name
Johnson
Lindsay
Turner

Result of *borrower* × *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

Result of $\sigma_{\text{branch-name} = \text{"Perryridge"}}$ (*borrower* × *loan*)

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

Result of $\Pi_{customer\text{-}name}$

customer-name
Adams
Hayes

Result of the Subexpression

balance
500
400
700
750
350

Largest Account Balance in the Bank

balance
900

Customers Who Live on the Same Street and In the Same City as Smith

customer-name
Curry
Smith

Customers With Both an Account and a Loan at the Bank

customer-name
Hayes
Jones
Smith

Result of $\Pi_{customer\text{-}name,\ loan\text{-}number,\ amount}$ (*borrower* $\bowtie$ *loan*)

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

Result of $\Pi_{branch\text{-}name}(\sigma_{customer\text{-}city = \text{"Harrison"}}(customer \bowtie account \bowtie depositor))$

branch-name
Brighton
Perryridge

Result of $\Pi_{branch\text{-}name}(\sigma_{branch\text{-}city\ =\ \text{“Brooklyn”}}(branch))$

branch-name
Brighton
Downtown

Result of $\Pi_{customer\text{-}name,\ branch\text{-}name}$(*depositor* ⋈ *account*)

customer-name	*branch-name*
Hayes	Perryridge
Johnson	Downtown
Johnson	Brighton
Jones	Brighton
Lindsay	Redwood
Smith	Mianus
Turner	Round Hill

The *credit-info* Relation

customer-name	*branch-name*
Hayes	Perryridge
Johnson	Downtown
Johnson	Brighton
Jones	Brighton
Lindsay	Redwood
Smith	Mianus
Turner	Round Hill

Result of $\Pi_{\textit{customer-name, (limit – credit-balance) as credit-available}}$*(credit-info).*

customer-name	*credit-available*
Curry	250
Jones	5300
Smith	1600
Hayes	0

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

The *pt-works* Relation After Grouping

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

Result of $_{\textit{branch-name}}\,\mathcal{G}\,_{\text{sum}(\textit{salary})}$ (*pt-works*)

branch-name	*sum of salary*
Austin	3100
Downtown	5300
Perryridge	8100

Result of $_{\textit{branch-name}}\,\mathcal{G}_{\text{sum } \textit{salary}, \text{ max}(\textit{salary}) \text{ as } \textit{max-salary}}$ *(pt-works)*

branch-name	*sum-salary*	*max-salary*
Austin	3100	1600
Downtown	5300	2500
Perryridge	8100	5300

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

The Result of *employee* ⟗ *ft-works*

employee-name	*street*	*city*	*branch-name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500

The Result of *employee* ⟗ *ft-works*

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*

Result of *employee* $⟖$ *ft-works*

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

Result of *employee* ⟗ *ft-works*

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

Tuples Inserted Into *loan* and *borrower*

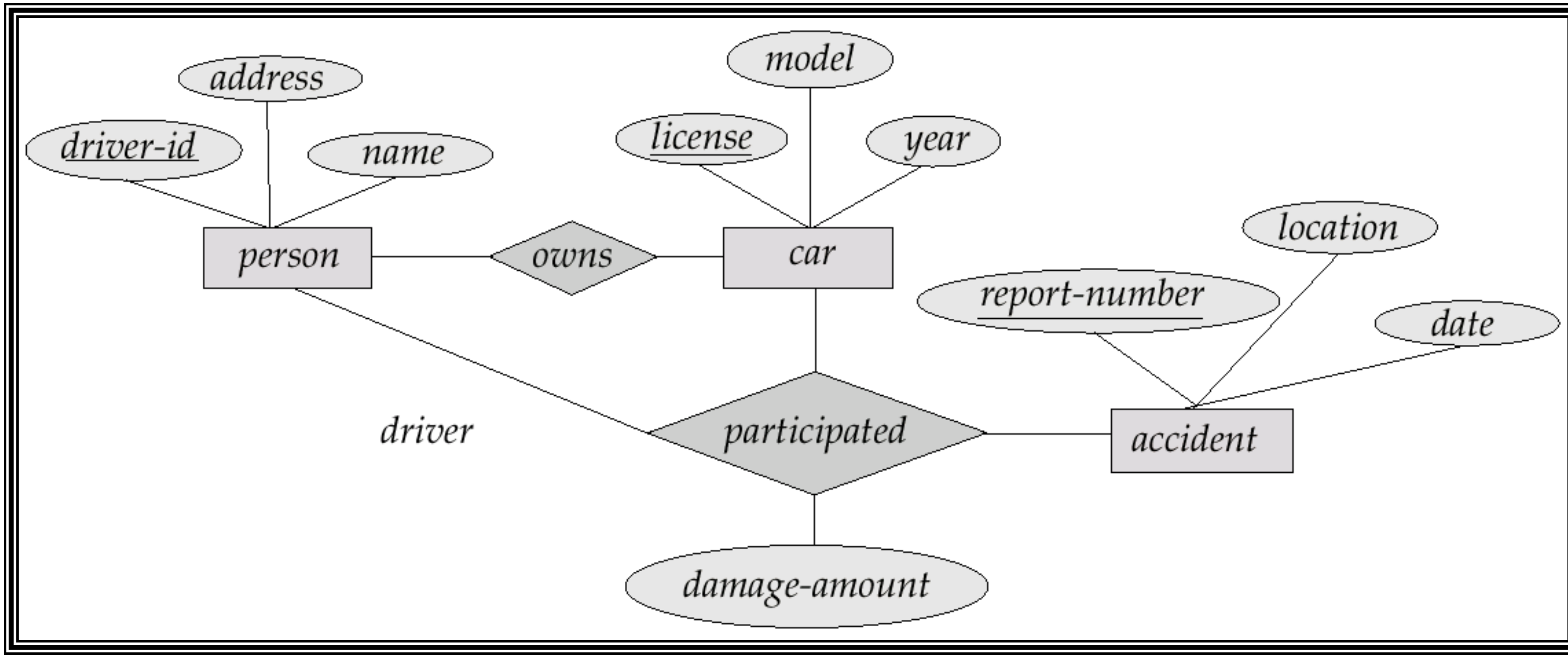
loan-number	*branch-name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500
null	*null*	1900

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
Johnson	*null*

Names of All Customers Who Have a Loan at the Perryridge Branch

customer-name
Adams
Hayes

E-R Diagram



address
driver-id
name
person
owns
model
license
year
car
location
report-number
date
driver
participated
accident
damage-amount

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

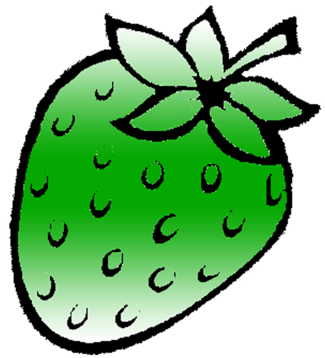
The *loan* Relation

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

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

STRAWBERRY



/strawberrydevelopers

/strawberry_app

For more visit:

Strawberrydevelopers.weebly.com