Relational Algebra

- By

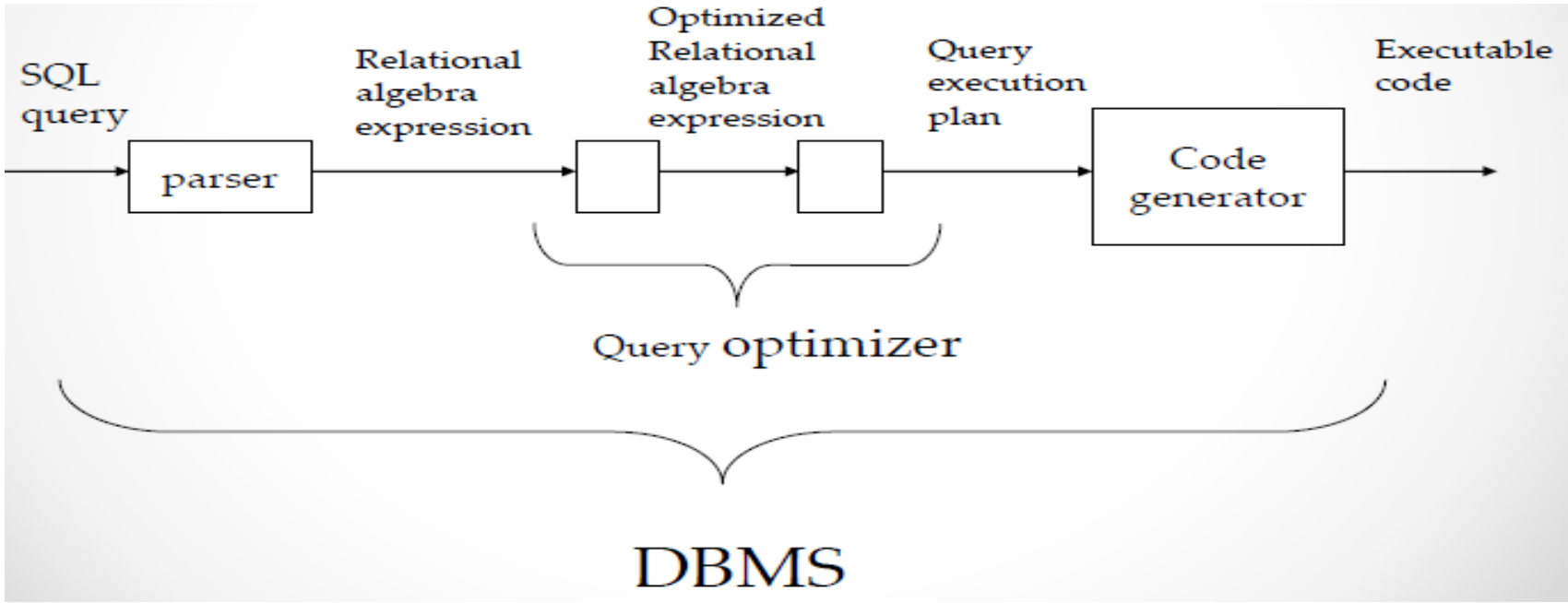
Jyoti Tryambake

Outline

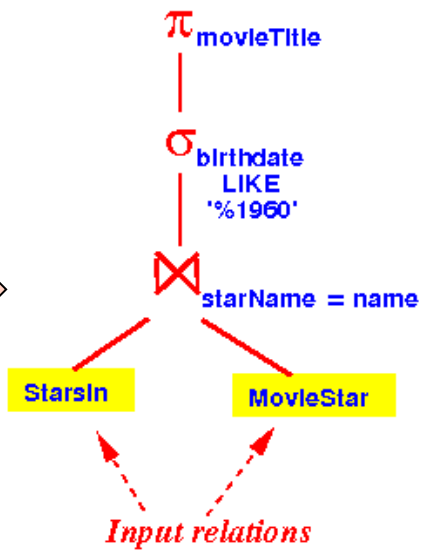
- Relational Algebra - the basis of the widely used SQL query language.
- Tuple Relational Calculus
- Domain Relational Calculus

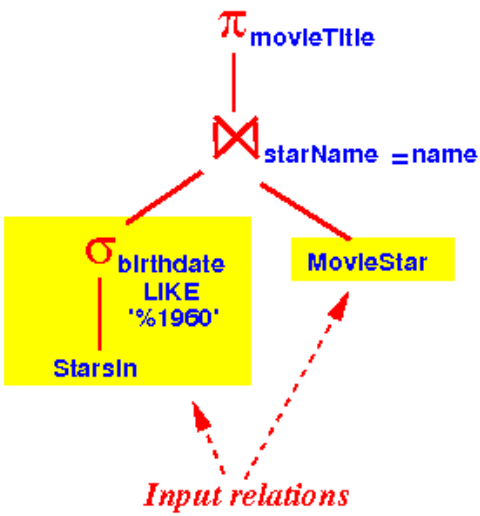
} declarative query languages based on mathematical logic

Relational Algebra in DBMS



Query tree example





Relational Algebra

- Relational algebra is the basic set of operations for the relational model.
- These operations enable the user to specify basic retrieval requests (or queries).
- The operators take one or two relations as inputs and produce a new relation as a result.
- Procedural language
- Six basic operators
  - select: σ
  - project: Π
  - rename: ρ

  } Unary operations

  - union: $\cup$
  - set difference: –
  - Cartesian product: x

  } Binary operations

Select Operation

- Selects tuples that satisfy a given predicate
- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Defined as:

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

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

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

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

Select Operation

- Examples of selection:
  - Select those tuples of the *instructor relation where the instructor is in the "Physics"* department

$$\sigma_{dept\_name="Physics"}(instructor)$$

  - Select the employee tuples whose salary is greater than $30,000:

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

  - to find the instructors in Physics with a salary greater than $90,000

$$\sigma_{dept\_name = "Physics" \wedge salary > 90000}(instructor)$$

  - comparisons between two attributes, To find all departments whose name is the same as their building name

$$\sigma_{dept\_name = building}(department)$$

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.
- This operation keeps certain *columns* (attributes) from a relation and discards the other columns.
- PROJECT creates a vertical partitioning
  - The list of specified columns (attributes) is kept in each tuple
  - The other attributes in each tuple are discarded
- Example: To eliminate the *dept_name* attribute of *instructor*

$$\Pi_{ID,\ name,\ salary}(instructor)$$

Combining Select and Project

- Find the name of all instructors in the Physics department
  - $\Pi_{name}(\sigma_{dept\ name = \text{“Physics”}}(instructor))$

Examples of applying SELECT and PROJECT operations

(a) $\sigma_{(Dno=4\ \textbf{AND}\ Salary>25000)\ OR\ (Dno=5\ \textbf{AND}\ Salary>30000)}(EMPLOYEE)$.

Results of SELECT and PROJECT operations ⇨ (b) $\pi_{Lname,\ Fname,\ Salary}(EMPLOYEE)$.

(c) $\pi_{Sex,\ Salary}(EMPLOYEE)$.

(a)

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5

(b)

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

(c)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Union Operation

- Takes the set of rows in each table and combines them, eliminating duplicates.
- Participating relations must be compatible, i. e. have the same number of columns, and the same column names, domains, and data types.

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*)

Union Operation

- Example 1: Find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) \cup \Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

- Example 2: List all cities where there is either a branch office or a property for rent.
  - $\Pi_{city}(\text{Branch}) \cup \Pi_{city}(\text{PropertyForRent})$

Union Operation

- Example 1 output

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Figure 6.4 The *section* relation.

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Figure 6.5 Courses offered in either Fall 2009, Spring 2010 or both semesters.

Set Difference Operation

- Takes the set of rows in the first relation but not the second
- Participating relations must be compatible.
- 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
- Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) - \Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

Set Difference Operation

- Example 1

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Figure 6.4 The *section* relation.

course_id
CS-347
PHY-101

Figure 6.6 Courses offered in the Fall 2009 semester but not in Spring 2010 semester.

Set Difference Operation

- Example 2

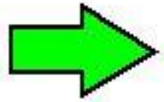
X1(Name, Age)

Name	Age
Anoop	22
Saurav	22
Rakesh	20
Pritesh	19

—

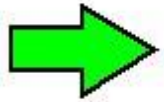
X2(Name, Age)

Name	Age
Anoop	22
Anurag	23
Ganesh	21
Saurav	22
Rakesh	20



Query Output(X1 - X2)

Name	Age
Pritesh	19



Query Output(X2 - X1)

Name	Age
Anurag	23
Ganesh	21

Set-Intersection Operation

- Takes the set of rows that are common to each relation
- Participating relations must be compatible
- 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

Set-Intersection Operation

- Example:
  - Find the set of all courses taught in both the Fall 2009 and the Spring 2010 semesters.

$$\Pi_{course\_id}\left(\sigma_{semester=\text{“Fall”} \wedge year=2009}\left(section\right)\right) \cap \Pi_{course\_id}\left(\sigma_{semester=\text{“Spring”} \wedge year=2010}\left(section\right)\right)$$

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Figure 6.4 The *section* relation.

course_id
CS-101

Figure 6.13 Courses offered in both the Fall 2009 and Spring 2010 semesters.

Cartesian-Product Operation

- It defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S.
- 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 $r(R)$ and $s(S)$ are not disjoint, then renaming must be used.

A1(Name, Roll No)

Name	Roll No
Anoop	1
Anurag	2

X

A2(Name, Roll No)

Name	Roll No
Anoop	1
Anurag	2
Ganesh	3

Query Output(A1 X A2)

Name	Roll No	Name	Roll No
Anoop	1	Anoop	1
Anoop	1	Anurag	2
Anoop	1	Ganesh	3
Anurag	2	Anoop	1
Anurag	2	Anurag	2
Anurag	2	Ganesh	3

Cartesian-Product Operation

- Example - find the names of all instructors in the Physics department together with the course id of all courses they taught
  - need the information in both the instructor relation and the teaches relation

$$\sigma_{dept\_name = \text{“Physics”}}(instructor \times teaches)$$

inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
22222	Einstein	Physics	95000	10101	CS-437	1	Fall	2009
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2010
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2010
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2009
22222	Einstein	Physics	95000	32343	HIS-351	1	Spring	2010
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
33456	Gold	Physics	87000	10101	CS-437	1	Fall	2009
33456	Gold	Physics	87000	10101	CS-315	1	Spring	2010
33456	Gold	Physics	87000	12121	FIN-201	1	Spring	2010
33456	Gold	Physics	87000	15151	MU-199	1	Spring	2010
33456	Gold	Physics	87000	22222	PHY-101	1	Fall	2009
33456	Gold	Physics	87000	32343	HIS-351	1	Spring	2010
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Cartesian-Product Operation

$$\sigma_{instructor.ID = teaches.ID}(\sigma_{dept\_name = \text{“Physics”}}(instructor \times teaches))$$

- The names of all instructors in the Physics department together with the *course id of all courses they taught*

$$\Pi_{name,\ course\_id}\ (\sigma_{instructor.ID = teaches.ID}(\sigma_{dept\_name = \text{“Physics”}}(instructor \times teaches)))$$

- Instructor Gold is in the Physics department, he does not teach any course (as recorded in the *teaches relation), and therefore does* not appear in the result.

name	*course_id*
Einstein	PHY-101

Cartesian-Product Operation

- Are following expressions are equivalent:

$$\Pi_{name,\ course\_id}\left(\sigma_{instructor.ID = teaches.ID}\left(\sigma_{dept\_name = \text{“Physics”}}\left(instructor \times teaches\right)\right)\right)$$

or

$$\Pi_{name,\ course\_id}\left(\sigma_{instructor.ID = teaches.ID}\left(\left(\sigma_{dept\_name = \text{“Physics”}}\left(instructor\right)\right) \times teaches\right)\right)$$

Division Operation

- Division operator A÷B can be applied if and only if:
  - Attributes of B is proper subset of Attributes of A.
  - The relation returned by division operator will have attributes = (All attributes of A – All Attributes of B)
  - The relation returned by division operator will return those tuples from relation A which are associated to every B's tuple.

Division Operation

- Suited to queries that include the phrase "**for all**".
- Let *r* and *s* be relations on schemas *R* and *S* respectively

$$r \div s$$

  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$

The result of r ÷ s is a relation on schema

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

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

* ***u*** *representing any tuple in s*

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

* *for every tuple in R-S (called t), there are a set of tuples in R, such that for all tuples (such as u) in s, the tu is a tuple in R.*

Division Operation – Example

- Relations *r, s*:

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

r

B
1
2

s

- $r \div s$:

A
α
β

e.g.

A is customer name

B is branch-name

1and 2 here show two specific branch-names

(Find customers who have an account in all branches of the bank)

Examples of Division A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

sno
s1
s2
s3
s4

A/B1

Which have p2 in A

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 $A_1, A_2, \ldots, A_n$.

Rename Operation

- Example - Find the highest salary in the university
- Strategy is to
  - (1) compute first a temporary relation consisting of those salaries that are *not the largest*
  - (2) take the set difference between the relation salary (instructor) and the temporary relation just computed, to obtain the result.

Rename Operation

- Example - Find the highest salary in the university
- Strategy is to

  (1) compute first a temporary relation consisting of those salaries that are *not the largest*

  - computing the Cartesian product instructor × instructor and forming a selection to compare the value of any two salaries appearing in one tuple
  - the rename operation to rename one reference to the instructor relation; to reference the relation twice without ambiguity
  - write the temporary relation that consists of the salaries that are not the largest:

$$\Pi_{instructor.salary} (\sigma_{instructor.salary < d.salary} (instructor \times \rho_d (instructor)))$$

Rename Operation

- Example - Find the highest salary in the university
- Strategy is to
  - (1) compute first a temporary relation consisting of those salaries that are *not the largest*

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

salary
65000
90000
40000
60000
87000
75000
62000
72000
80000
92000

Figure 6.11 Result of the subexpression $\Pi_{instructor.salary}(\sigma_{instructor.salary < d.salary}(instructor \times \rho_d(instructor)))$.

Rename Operation

- Example - Find the highest salary in the university
- Strategy is to
  - (2) take the set difference between the relation salary (instructor) and the temporary relation just computed, to obtain the result.
  - The query to find the largest salary in the university can be written as:

$$\Pi_{salary}\ (instructor) - \Pi_{instructor.salary}\ (\sigma_{instructor.salary\ <\ d.salary}\ (instructor\ \times\ \rho_d\ (instructor)))$$

salary
95000

Figure 6.12 Highest salary in the university.

Formal Definition of the Relational 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
  - $\Pi_S(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_x (E_1)$, x is the new name for the result of E_1

Natural Join

- The natural join is a binary operation that allows us to combine certain selections and a Cartesian product into one operation.
- It is denoted by the **join symbol** $\bowtie$
- Example: Find the names of all instructors together with the course id of all courses they taught

$$\Pi_{name,\ course\_id}\ (instructor \bowtie teaches)$$

- the schemas for instructor and teaches have the attribute ID in common, the natural-join operation considers only pairs of tuples that have the same value on ID.

name	*course_id*
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

Natural Join – Formal Definition

- The **natural join of *r and s, denoted by r s, is a relation*** on schema *R ∪ S formally defined as follows:*

$$r \bowtie s = \Pi_{R \cup S} \left(\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n} \left(r \times s \right) \right)$$

- if r (R) and s(S) are relations without any attributes in common, that is, R ∩ S = ∅, then r ⋈ s = r × s.
- Example - Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach

$$\Pi_{name, title} \left(\sigma_{dept\_name = \text{“Comp. Sci.”}} \left(instructor \bowtie teaches \bowtie course \right) \right)$$

name	*title*
Brandt	Game Design
Brandt	Image Processing
Katz	Image Processing
Katz	Intro. to Computer Science
Srinivasan	Intro. to Computer Science
Srinivasan	Robotics
Srinivasan	Database System Concepts

Natural Join

- the natural join is **associative**

$$(instructor \bowtie teaches) \bowtie course$$
$$instructor \bowtie (teaches \bowtie course)$$

- The theta join operation is a variant of the natural-join operation that allows us to combine a selection and a Cartesian product into a single operation
- Consider relations *r (R) and s(S), and let be a predicate on attributes in the schema R ∪ S.*
- The **theta join operation *is defined as follows:***

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

Theta join example

List the names and comments of all clients who have viewed a property for rent.

$$(\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \bowtie_{\text{client.clientNo = Viewing.clientNo}} (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing}))$$

client.clientNo	fName	lName	Viewing.clientNo	propertyNo	comment
CR76	John	Kay	CR76	PG4	too remote
CR56	Aline	Stewart	CR56	PA14	too small
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR56	PG36	
CR62	Mary	Tregear	CR62	PA14	no dining room

The Assignment Operation

- The **assignment operation, denoted by ←,** works like assignment in a programming language.

$$\Pi_{R \cup S}\left(\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n}\left(r \times s\right)\right)$$

- Written as;

$$\begin{aligned} temp1 &\leftarrow R \times S \\ temp2 &\leftarrow \sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n}\left(temp1\right) \\ result &= \Pi_{R \cup S}\left(temp2\right) \end{aligned}$$

- Just a convenient way to express complex queries.

Outer Join

- The **outer-join operation is an extension of the join operation to deal with missing information**
- It preserves those tuples that would be lost in an join by creating tuples in the result containing null values
- Types-
  - Left outer
  - Right outer
  - Full outer

Left Outer Join

- All rows from R are retained and unmatched rows of S are padded with NULL

ID	name	dept_name	salary	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
33456	Gold	Physics	87000	*null*	*null*	*null*	*null*
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
58583	Califieri	History	62000	*null*	*null*	*null*	*null*
76543	Singh	Finance	80000	*null*	*null*	*null*	*null*
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Figure 6.17 Result of $instructor \supset\!\bowtie teaches$.

Right Outer Join

- All rows from S are retained and unmatched rows of R are padded with NULL

ID	course_id	sec_id	semester	year	name	dept_name	salary
10101	CS-101	1	Fall	2009	Srinivasan	Comp. Sci.	65000
10101	CS-315	1	Spring	2010	Srinivasan	Comp. Sci.	65000
10101	CS-347	1	Fall	2009	Srinivasan	Comp. Sci.	65000
12121	FIN-201	1	Spring	2010	Wu	Finance	90000
15151	MU-199	1	Spring	2010	Mozart	Music	40000
22222	PHY-101	1	Fall	2009	Einstein	Physics	95000
32343	HIS-351	1	Spring	2010	El Said	History	60000
33456	*null*	*null*	*null*	*null*	Gold	Physics	87000
45565	CS-101	1	Spring	2010	Katz	Comp. Sci.	75000
45565	CS-319	1	Spring	2010	Katz	Comp. Sci.	75000
58583	*null*	*null*	*null*	*null*	Califieri	History	62000
76543	*null*	*null*	*null*	*null*	Singh	Finance	80000
76766	BIO-101	1	Summer	2009	Crick	Biology	72000
76766	BIO-301	1	Summer	2010	Crick	Biology	72000
83821	CS-190	1	Spring	2009	Brandt	Comp. Sci.	92000
83821	CS-190	2	Spring	2009	Brandt	Comp. Sci.	92000
83821	CS-319	2	Spring	2010	Brandt	Comp. Sci.	92000
98345	EE-181	1	Spring	2009	Kim	Elec. Eng.	80000

Figure 6.18 Result of $teaches \bowtie instructor$.

Full Outer Join

full outer join ($\overset{\cdot}{\supset\!\bowtie\!\subset}$)

- Does both the left and right outer join operations, padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adding them to the result of the join.

- Note: teaches tuples always have matching instructor tuples, and
- Will this $teaches \overset{\cdot}{\supset\!\bowtie} instructor$ give the same result as $teaches \bowtie instructor$?

Extended Relational-Algebra-Operations

- Generalized Projection
- Aggregate Functions

Generalized Projection

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

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

- E is any relational-algebra expression
- Each of $F_1, F_2, \ldots, F_n$ are arithmetic expressions such as +, −, ∗, and ÷ on numeric valued attributes, numeric constants, and on expressions that generate a numeric result in the schema of E.

Generalized Projection

Examples:

- Following example gives the ID, name, dept name, and the monthly salary of each instructor
  - $\Pi_{\text{ID,name,dept name,salary}\div 12}(\text{instructor})$
- Given relation credit-info(customer-name, limit, credit-balance), find how much more each person can spend:

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

Aggregate Functions and Operations

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

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

- **Aggregate operation** in relational algebra

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

  - 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

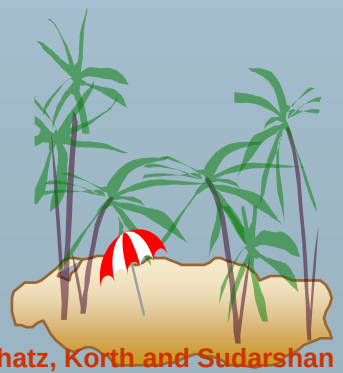
Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

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

sum-C
27



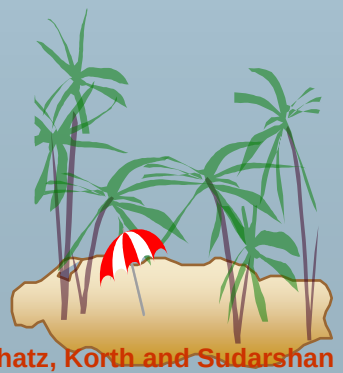
Aggregate Operation – Example

- 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

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

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.

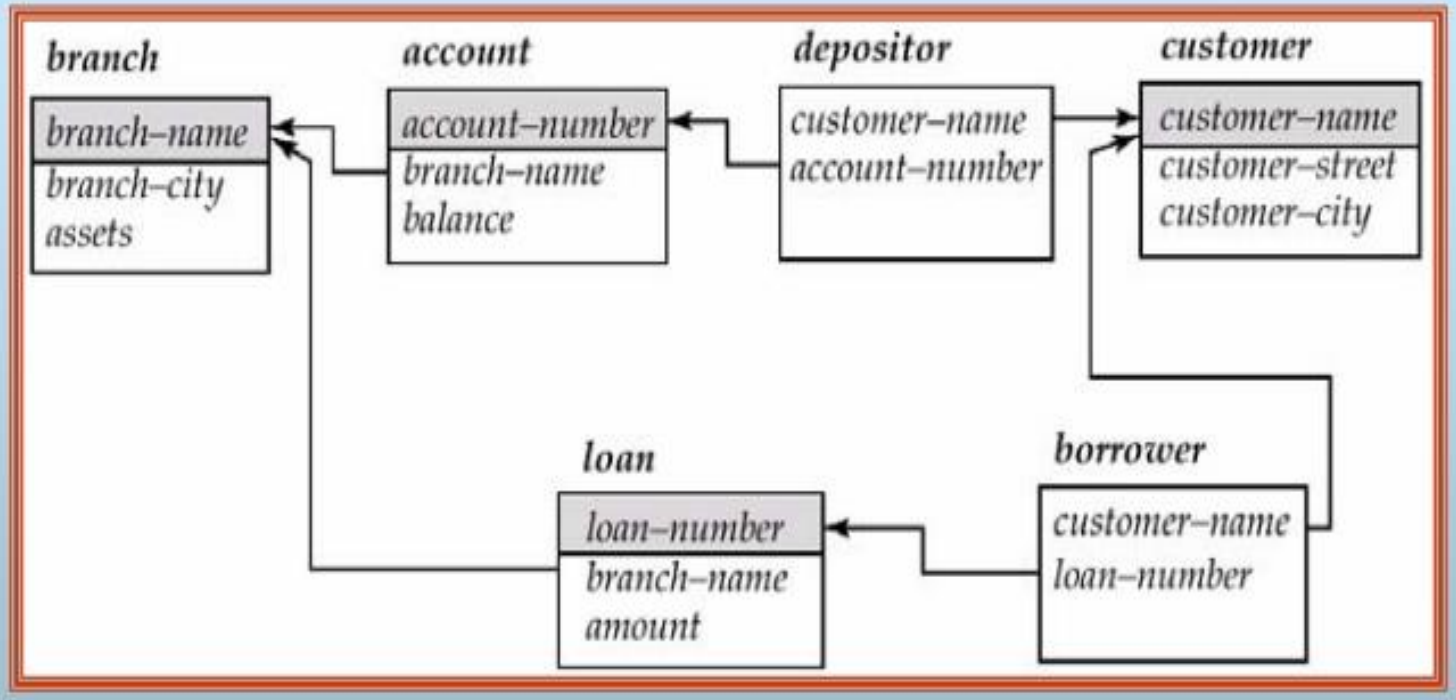
Deletion

- 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.
- A deletion is expressed in relational algebra by:

$$r \leftarrow r - E$$

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

Schema Diagram for the Banking Enterprise



Deletion Examples

- Delete all account records in the Perryridge branch.

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

- 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 (branch-city = Needham).

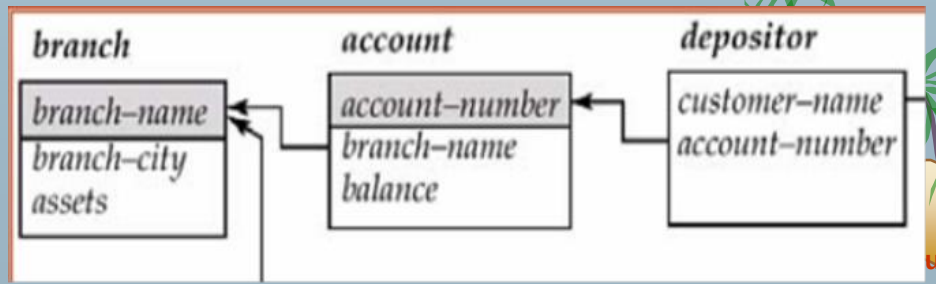
$$r_1 \leftarrow \sigma_{branch\text{-}city = \text{“Needham”}}(account \bowtie branch)$$

$$r_2 \leftarrow \Pi_{branch\text{-}name,\ account\text{-}number,\ balance}(r_1)$$

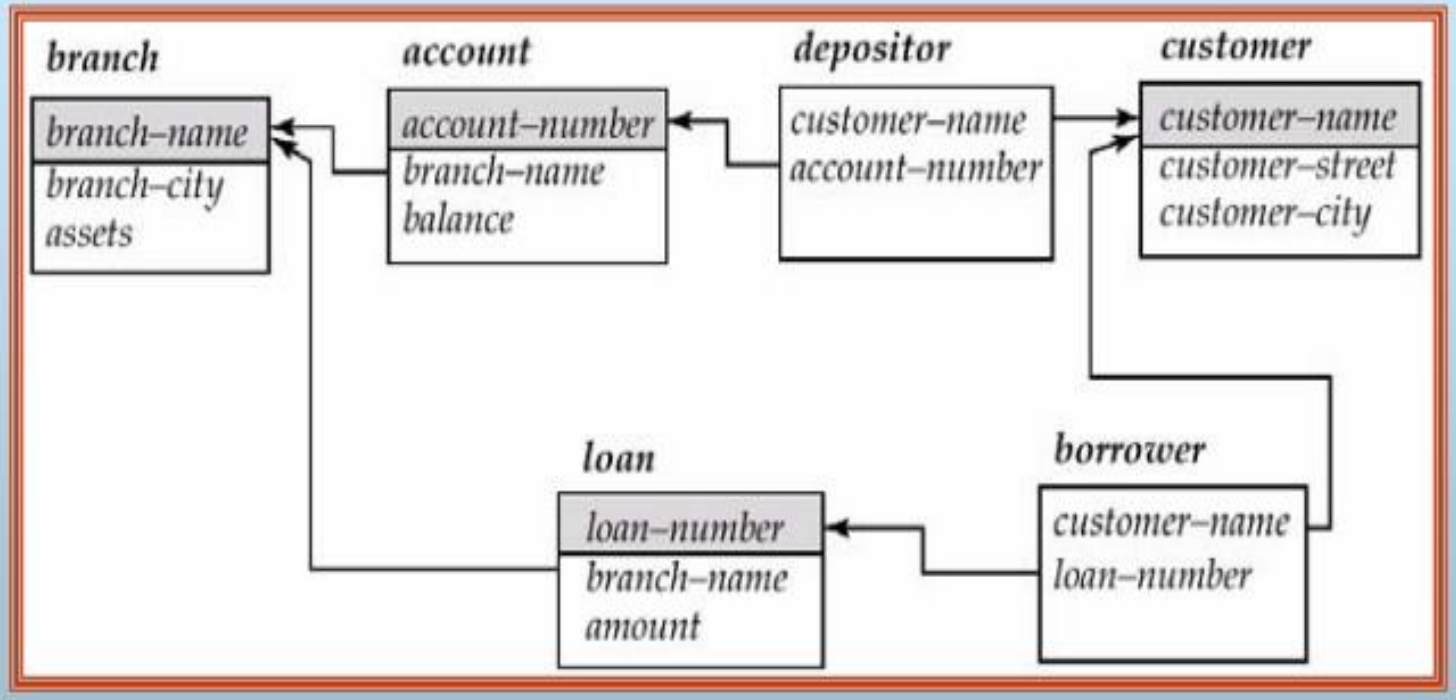
$$r_3 \leftarrow \Pi_{customer\text{-}name,\ account\text{-}number}(r_2 \bowtie depositor)$$

$$account \leftarrow account - r_2$$

$$depositor \leftarrow depositor - r_3$$



Schema Diagram for the Banking Enterprise



Insertion Examples

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

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

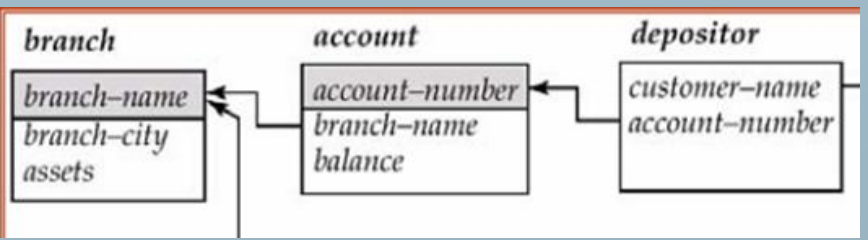
  $depositor \leftarrow depositor \cup \{(\text{“Smith”}, \text{A-973})\}$

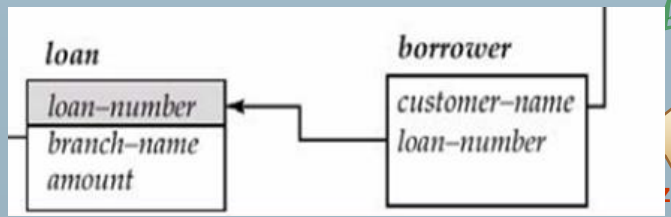
- Provide as a gift for all loan customers in the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account.

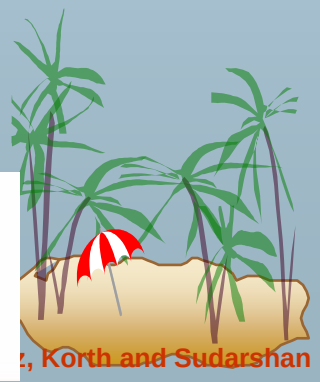
  $r_1 \leftarrow (\sigma_{branch\text{-}name = \text{“Perryridge”}} (borrower \bowtie loan))$

  $account \leftarrow account \cup \prod_{branch\text{-}name,\ loan\text{-}number,200} (r_1)$

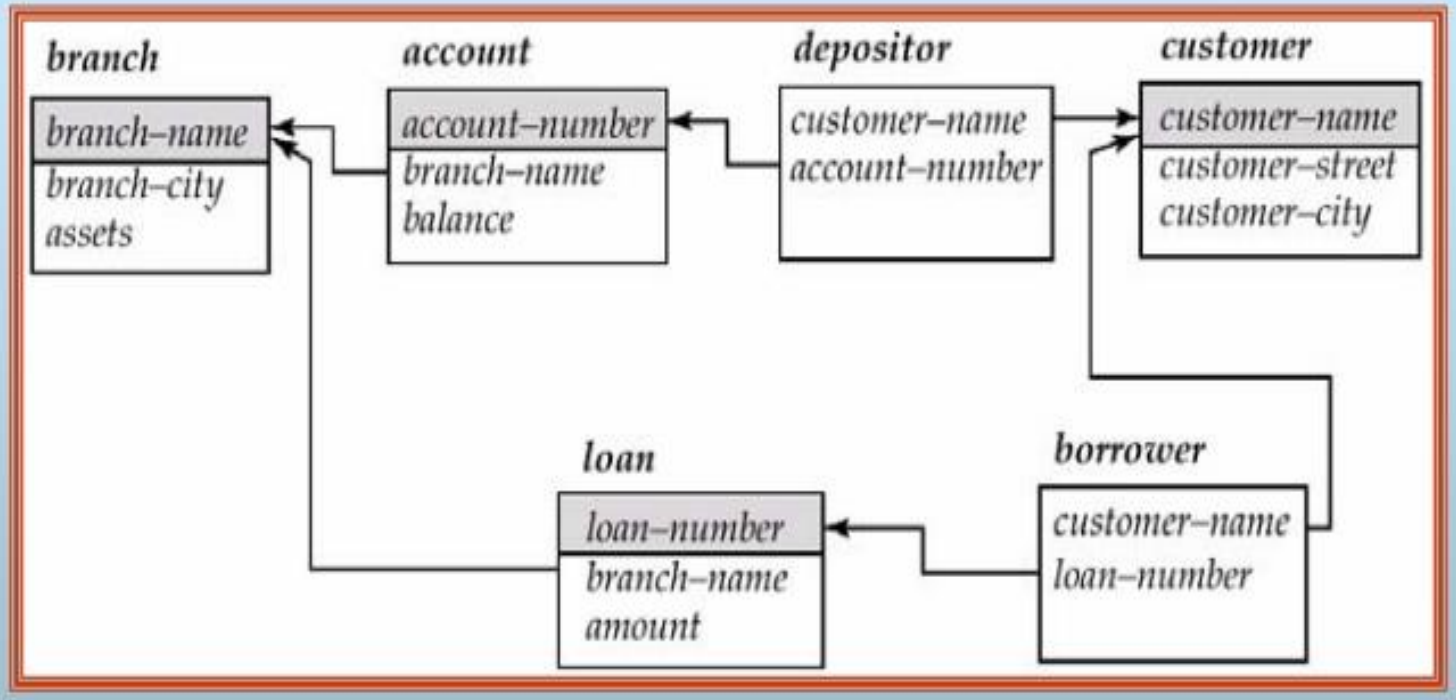
  $depositor \leftarrow depositor \cup \prod_{customer\text{-}name,\ loan\text{-}number} (r_1)$







Schema Diagram for the Banking Enterprise



Modification of the Database – Insertion

- Gift for all loan customers of the Perryridge branch: a $200 savings account. Let the loan number serve as the account number for the new savings account

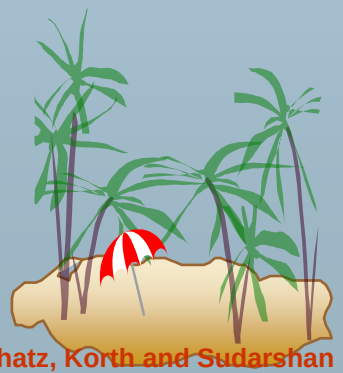
```
insert into account
    select loan_number, branch_name, 200
    from loan
    where branch_name = 'Perryridge'
insert into depositor
    select customer_name, loan_number
    from loan, borrower
    where branch_name = 'Perryridge'
    and loan.account_number = borrower.account_number
```

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 ith attribute of r, if the ith 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

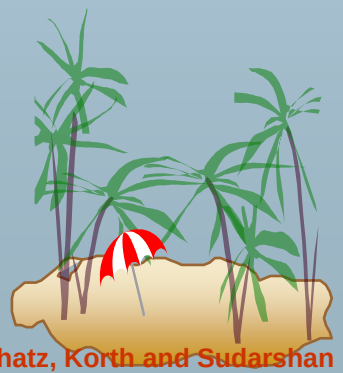
- Make interest payments by increasing all balances by 5 percent.

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

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

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

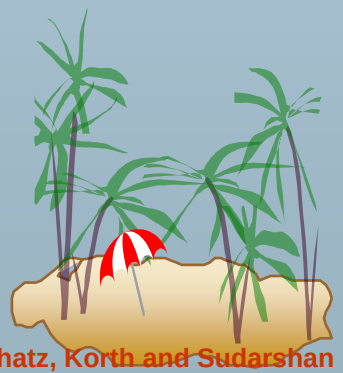


Views

- 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

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

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

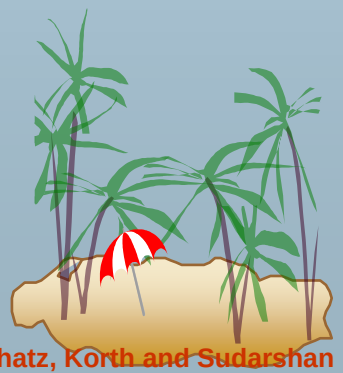


View Examples

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

create view *all-customer* **as**

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



Exercise

- Assume the following relations:

BOOKS(DocId, Title, Publisher, Year)

STUDENTS(StId, StName, Major, Age)

AUTHORS(AName, Address)

borrows(DocId, StId, Date)

has-written(DocId, AName)

describes(DocId, Keyword)

1. List the year and title of each book.
2. List all information about students whose major is CS.
3. List all books published by McGraw-Hill before 1990.
4. List the name of those authors who are living in Davis.
5. List the names of all students who have borrowed a book and who are CS majors.

Exercise

- Assume the following relations:

BOOKS(DocId, Title, Publisher, Year)

STUDENTS(StId, StName, Major, Age)

AUTHORS(AName, Address)

borrows(DocId, StId, Date)

has-written(DocId, AName)

describes(DocId, Keyword)

6. List the title of books written by the author 'Silberschatz'.

7. List the title of books written by the author 'Silberschatz‘ but not books that have the keyword 'database'.

8. Find the name of the youngest student.

9. List the title of books written by the author 'Ullman'.

10. List the authors of the books the student 'Smith' has borrowed.

11. Which books have both keywords 'database' and 'programming'?

References

- Korth