



Chapter 3

Relational Model and Relational Algebra

Introduction to the Relational Model, relational schema and concept of keys. Mapping the ER and EER Model to the Relational Model, Relational Algebra-operators, Relational Algebra Queries.



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Data Science

Example of a Relation

<i>account_number</i>	<i>branch_name</i>	<i>balance</i>
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



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Data Science

Basic Structure

- Formally, given sets D_1, D_2, \dots, D_n a **relation** r is a subset of

$$D_1 \times D_2 \times \dots \times D_n$$

Thus, a relation is a set of n -tuples (a_1, a_2, \dots, 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 \times *customer_street* \times *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
 - Note: multivalued attribute values are not atomic
 - Note: 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, \dots, A_n are *attributes*
- $R = (A_1, A_2, \dots, A_n)$ is a *relation schema*

Example:

Customer_schema = (*customer_name*, *customer_street*,
customer_city)

- $r(R)$ is a *relation* on the *relation schema* R

Example:

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

<i>customer_name</i>	<i>customer_street</i>	<i>customer_city</i>
Jones	Main	Harrison
Smith	North	Rye
Curry	North	Rye
Lindsay	Park	Pittsfield

customer

Diagram illustrating the components of a relation instance:

- Attributes (or columns):** Indicated by arrows pointing to the column headers *customer_name*, *customer_street*, and *customer_city*.
- Tuples (or rows):** Indicated by arrows pointing to the five rows of data.



Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *account* relation with unordered tuples

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



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

account : stores information about accounts

depositor : stores information about which customer
owns which account

customer : stores information about customers

- Storing all information as a single relation such as
bank(account_number, balance, customer_name, ...)

results in

- repetition of information (e.g., two customers own an account)
- the need for null values (e.g., represent a customer without an account)



The *customer* Relation

<i>customer_name</i>	<i>customer_street</i>	<i>customer_city</i>
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



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Data Science

The *depositor* Relation

<i>customer_name</i>	<i>account_number</i>
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Data Science

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.
- **Primary Key**



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Data Science

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.



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Data Science

Relational Algebra

- Procedural language
- Six basic operators
 - select: σ
 - project: Π
 - union: \cup
 - set difference: $-$
 - Cartesian product: \times
 - 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



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:

$\langle \text{attribute} \rangle \quad op \quad \langle \text{attribute} \rangle \text{ or } \langle \text{constant} \rangle$

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

- Example of selection:

$$\sigma_{\text{branch_name}=\text{"Perryridge"}(\text{account})}$$



Project Operation – Example

- Relation r :

	A	B	C
α	10	1	
α	20	1	
β	30	1	
β	40	2	

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

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2



Project Operation

- Notation:

$$\prod_{A_1, A_2, \dots, A_k} (r)$$

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*

$$\prod_{\text{account_number}, \text{balance}} (\text{account})$$



Union Operation – Example

- 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

- 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

$$\prod_{customer_name} (depositor) \cup \prod_{customer_name} (borrower)$$



Union Operation

Example:

DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17



PARSHVANATH CHARITABLE TRUST'S
A.P. SHAH INSTITUTE OF TECHNOLOGY
Department of Computer Science and Engineering
Data Science

Input:

\sqcap CUSTOMER_NAME (BORROW) \sqcup \sqcap CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Johnson
Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Mayes



Set Difference Operation – Example

- Relations r, s :

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r - s$:

A	B
α	1
β	1



Set Difference Operation

- Notation $r - s$
- Defined as:

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

- 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

Example:

DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Data Science

Set Difference Operation

Notation: R - S

Input:

$$\Pi \text{ CUSTOMER_NAME} (\text{BORROW}) - \Pi \text{ CUSTOMER_NAME} (\text{DEPOSITOR})$$

Output:

CUSTOMER_NAME
Jackson
Hayes
Willians
Curry

Jackson

Hayes

Willians

Curry



INTERSECTION Operation

Example:

DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17



PARSHVANATH CHARITABLE TRUST'S
A.P. SHAH INSTITUTE OF TECHNOLOGY
Department of Computer Science and Engineering
Data Science

Input:

Π CUSTOMER_NAME (BORROW) \cap Π CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME

Smith

Jones



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

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



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Data Science

Cartesian-Product Operation

- Notation $r \times s$
- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
- It is denoted by \times .
- Defined as:

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

- 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:

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal



Cartesian-Product Operation

Input:

EMPLOYEE X DEPARTMENT

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal



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

A	B	C	D	E
---	---	---	---	---

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

α	1	α	10	a
β	2	β	10	a
β	2	β	20	b



Rename Operation

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

$$\rho_x(E)$$

returns the expression E under the name X

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

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

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



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)



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Data Science

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

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



Example Queries

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

$$\Pi_{customer_name} (borrower) \cup \Pi_{customer_name} (depositor)$$

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

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



Example Queries

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

$$\Pi_{customer_name} (\sigma_{branch_name = "Perryridge"} \\ (\sigma_{borrower.loan_number = loan.loan_number}(borrower \times loan)))$$

- 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.

$$\Pi_{customer_name} (\sigma_{branch_name = "Perryridge"} \\$$
$$(\sigma_{borrower.loan_number = loan.loan_number}(borrower \times loan)) - \\ \Pi_{customer_name}(depositor)$$



Example Queries

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

- Query 1

$$\prod_{\text{customer_name}} (\sigma_{\text{branch_name} = \text{"Perryridge"} } (\sigma_{\text{borrower.loan_number} = \text{loan.loan_number}} (\text{borrower} \times \text{loan})))$$

- Query 2

$$\prod_{\text{customer_name}} (\sigma_{\text{loan.loan_number} = \text{borrower.loan_number}} (\sigma_{\text{branch_name} = \text{"Perryridge"} } (\text{loan}) \times \text{borrower}))$$



Example Queries

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

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



Formal Definition

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



Additional Operations

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

- Set intersection
- Natural join
- Division
- Assignment



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Data Science

Set-Intersection Operation

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



Set-Intersection Operation – Example

- Relation r, s :

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

Division Operation

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

The result of $r \div s$ is a relation on schema

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

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

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

Division Operation – Example

□ Relations r, s :

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

B
1
2

s

□ $r \div s$:

A
α
β

r

Another Division Example

□ Relations r, s :

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

r

D	E
a	1
b	1

s

□ $r \div s$:

A	B	C
α	a	γ
γ	a	γ

Division Operation (Cont.)

- Property

- Let $q = r \div s$
- Then q is the largest relation satisfying $q \times s \subseteq r$
- Definition in terms of the basic algebra operation

Let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$

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

To see why

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



Assignment Operation

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

```
temp1 ← ΠR-S(r)
temp2 ← ΠR-S((temp1 × s) - ΠR-S,S(r))
result = temp1 - temp2
```

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



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
- Example:

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

$$S = (E, B, D)$$

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

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



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

Data Science

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	ϵ

s

□ $r \bowtie s$

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



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

$$\Pi_{customer_name} (\sigma_{branch_name = "Downtown"} (depositor \bowtie account)) \cap$$
$$\Pi_{customer_name} (\sigma_{branch_name = "Uptown"} (depositor \bowtie account))$$
$$\Pi_{customer_name, branch_name} (depositor \bowtie account) \\ \div \rho_{temp(branch_name)} (\{("Downtown"), ("Uptown")\})$$

Note that Query 2 uses a constant relation.



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Data Science

Example Queries

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

$$\begin{aligned} & \prod_{customer_name, branch_name} (depositor \bowtie account) \\ & \div \prod_{branch_name} (\sigma_{branch_city = "Brooklyn"} (branch)) \end{aligned}$$



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Data Science

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.

$$\prod_{F_1, F_2, \dots, F_n}(E)$$

- E is any relational-algebra expression
- Each of F_1, F_2, \dots, 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:

$$\prod_{customer_name, limit - credit_balance}(credit_info)$$



Aggregate Functions and Operations

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

avg: average value

min: minimum value

max: maximum value

sum: sum of values

count: number of values

- **Aggregate operation** in relational algebra

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

E is any relational-algebra expression

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



Aggregate Operation – Example

- Relation r :

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

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

sum(c)

27



Aggregate Operation – Example

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

<i>branch_name</i>	<i>account_number</i>	<i>balance</i>
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

branch_name $\text{g sum}(\text{balance})$ (*account*)

<i>branch_name</i>	$\text{sum}(\text{balance})$
Perryridge	1300
Brighton	1500
Redwood	700



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Data Science

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 g sum(balance) as sum_balance (account)



Outer Join

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



Outer Join – Example

- Relation *loan*

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

- Relation *borrower*

<i>customer_name</i>	<i>loan_number</i>
Jones	L-170
Smith	L-230
Hayes	L-155



Outer Join – Example

- Inner Join

loan ⚡ *Borrower*

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>	<i>customer_name</i>
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

- Left Outer Join

loan ☐☒ *Borrower*

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>	<i>customer_name</i>
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	null



Outer Join – Example

- Right Outer Join

loan \bowtie *borrower*

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>	<i>customer_name</i>
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	<i>null</i>	<i>null</i>	Hayes

- Full Outer Join

loan $\bowtie\bowtie$ *borrower*

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



Null Values

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



Null Values

- Comparisons with null values return the special truth value: *unknown*
 - If *false* was used instead of *unknown*, then $\text{not } (A < 5)$ would not be equivalent to $A \geq 5$
- 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.



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Data Science

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.



PARSHVANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

Data Science

Deletion Examples

- Delete all account records in the Perryridge branch.

$account \leftarrow account - \sigma_{branch_name = "Perryridge"}(account)$

- 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.

$r_1 \leftarrow \sigma_{branch_city = "Needham"}(account \bowtie branch)$

$r_2 \leftarrow \Pi_{branch_name, account_number, balance}(r_1)$

$r_3 \leftarrow \Pi_{customer_name, account_number}(r_2 \bowtie depositor)$

$account \leftarrow account - r_2$

$depositor \leftarrow depositor - r_3$



Insertion

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

$$r \leftarrow r \cup E$$

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

- The insertion of a single tuple is expressed by letting E be a constant relation containing one tuple.



Insertion Examples

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

$account \leftarrow account \cup \{("Perryridge", A-973, 1200)\}$

$depositor \leftarrow depositor \cup \{("Smith", A-973)\}$

- 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.

$r_1 \leftarrow (\sigma_{branch_name = "Perryridge"}(borrower \bowtie loan))$

$account \leftarrow account \cup \Pi_{branch_name, loan_number, 200}(r_1)$

$depositor \leftarrow depositor \cup \Pi_{customer_name, loan_number}(r_1)$



Updating

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

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

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



Update Examples

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

$$\text{account} \leftarrow \prod_{\text{account_number}, \text{branch_name}, \text{balance}} \text{balance} * 1.05 (\text{account})$$

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

$$\begin{aligned} \text{account} \leftarrow & \prod_{\text{account_number}, \text{branch_name}, \text{balance}} \text{balance} * 1.06 (\sigma_{BAL > 10000} (\text{account})) \\ & \cup \prod_{\text{account_number}, \text{branch_name}, \text{balance}} \text{balance} * 1.05 (\sigma_{BAL \leq 10000} (\text{account})) \end{aligned}$$

Figure . The *branch* relation

<i>branch_name</i>	<i>branch_city</i>	<i>assets</i>
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 : The *loan* relation

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
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 : The *borrower* relation

<i>customer_name</i>	<i>loan_number</i>
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17