

# Relational Model

Dr. Odelu Vanga

Indian Institute of Information Technology Sri City

<http://www.iiits.ac.in/people/regular-faculty/dr-odelu-vanga/>

# 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.

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

- $E$  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:

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

# Aggregate Functions and Operations

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

**avg**: average value

**min**: minimum value

**max**: maximum value

**sum**: sum of values

**count**: number of values

- **Aggregate operation** in relational algebra

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

- $E$  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  $r$ :

$A$	$B$	$C$
$\alpha$	$\alpha$	7
$\alpha$	$\beta$	7
$\beta$	$\beta$	3
$\beta$	$\beta$	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*  $g_{sum(balance)}$  (*account*)

<i>branch-name</i>	<i>balance</i>
Perryridge	1300
Brighton	1500
Redwood	700

# Aggregate Functions (Cont.)

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

*branch-name* ***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.



# Joined Relations – Datasets for Examples

- 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

Note: borrower information missing for L-260  
and loan information missing for L-155

# Inner 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

- **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

# 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

- Left Outer Join**

*loan*  $\sqcup\bowtie$  *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>

# 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

- Right Outer Join

*loan* ⋈<sub>⊃</sub> *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

# 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

- Full 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-260	Perryridge	1700	<i>null</i>
L-155	<i>null</i>	<i>null</i>	Hayes

# Joined Relations

- Join operations take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the **from** clause
- Join condition – defines which tuples in the two relations match, and what attributes are present in the result of the join.
- Join type – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

Join Types
<b>inner join</b> <b>left outer join</b> <b>right outer join</b> <b>full outer join</b>

Join Conditions
<b>natural</b> <b>on</b> <predicate> <b>using</b> ( $A_1, A_2, \dots, A_n$ )

# Joined Relations – Datasets for Examples

- 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

Note: borrower information missing for L-260  
and loan information missing for L-155

# Joined Relations – Examples

*loan* **inner join** *borrower* **on**

*loan.loan-number = borrower.loan-number*

<i>loan-number</i>	<i>branch-name</i>	<i>amount</i>	<i>customer-name</i>	<i>loan-number</i>
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230

*loan* **left inner join** *borrower* **on**

*loan.loan-number = borrower.loan-number*

<i>loan-number</i>	<i>branch-name</i>	<i>amount</i>	<i>customer-name</i>	<i>loan-number</i>
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	<i>null</i>	<i>null</i>



# Joined Relations – Examples

*loan* **natural inner join** *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

*loan* **natural right outer join** *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	null	null	Hayes

# Joined Relations – Examples

*loan* **full outer join** *borrower* **using** (*loan-number*)

<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

Find all customers who have either an account or a loan (but not both) at the bank.

**select** *customer-name*  
**from** (*depositor* **natural full outer join** *borrower*)  
**where** *account-number* **is** *null* **or** *loan-number* **is** *null*

**BREAK**

# Null Values

- *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
- 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  $\text{not } (A < 5)$  would not be equivalent to  $A \geq 5$
- Three-valued logic using the truth value *unknown*:
  - OR:  $(\text{unknown} \textbf{ or } \text{true}) = \text{true}$ ,  
 $(\text{unknown} \textbf{ or } \text{false}) = \text{unknown}$   
 $(\text{unknown} \textbf{ or } \text{unknown}) = \text{unknown}$
  - AND:  $(\text{true} \textbf{ and } \text{unknown}) = \text{unknown}$ ,  
 $(\text{false} \textbf{ and } \text{unknown}) = \text{false}$ ,  
 $(\text{unknown} \textbf{ and } \text{unknown}) = \text{unknown}$
  - NOT:  $(\textbf{not } \text{unknown}) = \text{unknown}$
  - 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*

**THANK YOU**