Relational Algebra

Objectives

- Introduction to Relational Algebra +
- Relational Algebra Operations +
- Summary +
- Example Queries +

- Introduction To Relational Algebra ...

- Relational algebra is a set of operations that enable the user to specify basic retrieval requests. Each operation produces results which is a relation.
- Relational algebra expression is a sequence of relational algebra operations whose result will also be a relation.
- There are two groups of relational algebra operations:
  - Operations developed specifically for relational database, such as SELECT, PROJECT, and JOIN.
  - Operations from mathematical set theory, such as UNION, SET DIFFERENCE, INTERSECTION, CARTESIAN PRODUCT, and DIVISION

… - Introduction To Relational Algebra

- Set theoretic operations are used to merge the tuples of two relations. These are binary operations.
- Some set theoretic operations require both relations must be union compatible.
- Union compatible relations should have the same degree and each pair of corresponding attribute should have the same domain. These include:
  - UNION
  - SET DIFFERENCE
  - INTERSECTION
- CARTESIAN PRODUCT is another set theoretic operation which doesn’t require union compatibility.

- Relational Algebra Operations

- Select +
- Project +
- Rename +
- Union +
- Difference +
- Intersection +
- Division +
- Assignment +
- Cartesian Product +
- Join +
- Outer Union +
- Composition of Operators +
- Aggregate Functions +
- Null Values +

-- Select Operation

- SELECT operation is used to select a subset of the tuples from the relation that satisfies the select condition.
- It is denoted by: $\sigma_p(r)$
- p is called the selection predicate (SELECT condition)
- 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: $=, \neq, >, \geq, <, \leq$

- Example of selection: $\sigma_{name=\text{"Adil"}}(EMPLOYEE)$

--- Select Operation – Example

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

r

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

A	B	C	D
α	α	1	7
β	β	23	10

--- Characteristics of SELECT Operation

- The select condition is applied independently to each tuple *t* in *r*. If the condition is true, then tuple *t* is selected and will appear in the resulting relation.
- The SELECT operation is unary, it is applied to a single relation.
- The degree of resulting relation is the same as *r*.
- The cardinality of resulting relation is less than or equal to *r*.
- The SELECT operation is cumulative. A sequence of SELECT operations can be applied in any order.
  - $\sigma<\text{cond1}>(\sigma<\text{cond2}>(r)) = \sigma<\text{cond2}>(\sigma<\text{cond1}>(r))$
- A cascade of SELECT operations can be combined into a single SELECT operation with a conjunctive (^) condition.
  - $\sigma<\text{cond1}>(\ldots(\sigma<\text{condn}>(r)) = \sigma<\text{cond1}>\wedge<\text{cond2}>\ldots<\text{condn}>(r)$

-- Project Operation

- Is used to select some attributes from a relation.
- Is denoted by:

$$\Pi_{\text{<attribute list>}}(r)$$

where <attribute list> are attribute names and *r* is a relation algebra expression

- The result is defined as the relation of <attribute list> columns obtained by erasing the columns that are not listed
- Example: To eliminate the *name* attribute of *DEPARTMENT*

$$\Pi_{number}(DEPARTMENT)$$

--- Project Operation – Example

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

r

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

A	C
α	1
β	1
β	2

Duplicates Removed

--- Characteristics of PROJECT Operation

- The result of a PROJECT operation will be a relation consisting of the attributes specified in the <attribute list> in the same order.

- The degree is equal to the number of attributes in the list.

- The projection operations removes any duplicates.

- The cardinality of the resulting relation is always less than or equal to the cardinality of r.

- For a cascade of PROJECT operations, only the outermost need to be considered for evaluation. If $<list1> \subseteq <list2> \subseteq \ldots \subseteq <listn> \subseteq r$, then
  - $\prod_{<list1>}(\prod_{<list2>}(\ldots (\prod_{<listn>}(r))) = \prod_{<list1>}(r)$

-- Rename Operation

- The rename operation (ρ) 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:

  $$\rho_s(r)$$

  returns the expression *r* under the name *s*

- If a relational-algebra expression *r* has arity *n*, then

  $$\rho_{s\,(A1,\, A2,\, \ldots,\, An)}(r)$$

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

--- Rename Operations: Example

r

A	B	C
α	a	α
α	a	γ
α	a	γ

$\rho_s(r)$

s

A	B	C
α	a	α
α	a	γ
α	a	γ

$\rho_{(D,\ E,\ F)}(s)$

$\rho_{s(D,\ E,\ F)}(r)$

s

D	E	F
α	a	α
α	a	γ
α	a	γ

-- Union Operation

- Is denoted by: $r \cup s$
- Is defined as:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- The result of $r \cup s$ will include all tuples which are either in r or in s or in both.
- For $r \cup s$ to be valid r and **s** must be union compatible
- Union operation is:
  - Commutative: $r \cup s = s \cup r$
  - Associative: $r \cup (s \cup w) = (r \cup s) \cup w$
- E.g. to find all the names of faculty and students in the FACULTY and STUDENT tables: $\Pi_{name}(FACULTY) \cup \Pi_{name}(STUDENT)$

--- Union Operation – Example

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

No Duplicates

-- Set Difference Operation

- Is denoted by: $r - s$
- IS defined as: $r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$
- The result of $r - s$ will include all the tuples that are in r but not in s.
- r and s must be union compatible
- this operation operation is neither Commutative nor Associative.

--- Set Difference Operation – Example

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

r – s

A	B
α	1
β	1

-- Set-Intersection Operation

- Is denoted by: $r \cap s$
- Is defined as: $r \cap s = \{ t \mid t \in r \text{ and } t \in s \}$
- The result of $r \cap s$ will include all the tuples that are in both *r* and *s*.
- *r* and *s* must be union compatible.
- Intersection is:
  - Commutative: $r \cap s = s \cap r$
  - Associative: $r \cap (s \cap w) = (r \cap s) \cap w$
- Note: $r \cap s = r - (r - s)$

--- Set-Intersection Operation - Example

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 …

- Is denoted by: $r \div s$
- 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 ÷ s is a relation on schema $R - S = (A_1, \ldots, A_m)$

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

--- Division Operation – Example1

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

r

B
1
2

s

$r \div s$ →

A
α
β

--- Division Operations: Example 2

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* ÷ *s $\longrightarrow$

A	*B*	*C*
α	a	γ
γ	a	γ

… -- Division Operation

- 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

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

  - 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.

--- Assignment Operation – Example

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

r

$$Temp1 \leftarrow \Pi_{A,C}\,(r)$$

A	C
α	1
β	1
β	2

Temp1

-- Cartesian-Product Operation

- Is denoted by: $r \times s$
- Is defined as:

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

- The result of $r \times s$ will combine tuples from both r and s in a combinatorial fashion.
- Assume that attributes of r(A) and s(B) are disjoint. (That is, $A \cap B = \varnothing$).
- If attributes of $r(A)$ and $s(B)$ are not disjoint, then renaming must be used.

--- Cartesian-Product Operation-Example

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

--- Characteristics of Cartesian-Product Operation

- Degree r X s = degree(r) + degree(s)
- Cardinality of r X s = cardinality(r) * cardinality(s)
- Generally the result of CARTESIAN PRODUCT is meaningless unless is followed by SELECT, and is called JOIN.

-- JOIN

- Join Operation combine related tuples from two relations into a single tuple based on join condition.
- Its is denoted by: r $\bowtie_{<join\ condition>}$ s

--- Characteristic of the Join Operation

- Degree of the $r \bowtie s$ = degree(r) + degree(s).
- Cardinality of $r \bowtie s$ is between 0 and cardinality (r) * cardinality(s).
- The order of attributes in $r \bowtie s$ is { $A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m$} where $A_1, A_2, \ldots, A_n$ attributes of *r* and $B_1, B_2, \ldots, B_m$ are attributes of *s*.
- The resulting relation has one tuple for each combination of tuples – one from r and one for s – whenever the combination satisfies the join condition.

--- Types of Join Operation

- Theta join +
- Equijoin +
- Natural join +
- Outer Join +
  - Left Outer Join +
  - Right Outer Join +
  - Full Outer Join +

--- Tables Used in Coming Examples

- The following two tables will be used in coming examples.

loan

loan-number	*branch-name*	*amount*
L-170	Khobar	3000
L-230	Riyadh	4000
L-260	Dammam	1700

borrower

customer-name	*loan-number*
Adel	L-170
Sami	L-230
Hashem	L-155

--- Theta Join

- Its is denoted by: $r \bowtie_{<r.A\ \theta\ s.B>} s$

  Where $\theta = \{=, \neq, <, >, \leq, \geq\}$

$$loan \bowtie_{loan\text{-}number = loan\text{-}number} Borrower$$

$$\downarrow$$

Loan-number	Branch-name	amount	Customer-name	Loan-number
L-170	Khobar	3000	Adel	L-170
L-230	Riyadh	4000	Sami	L-230

--- Equijoin

- The most common join involves join conditions with equality comparisons, where θ is =. This special type of Theta join is called Equijoin.

--- Equijoin

- The most common join involves join conditions with equality comparisons, where θ is {=}. This special type of Theta join is called Equijoin.

- Its is denoted by: r $\bowtie$ *<r.A = s.B>* s

$$loan \bowtie_{loan\text{-}number\, =\, loan\text{-}number} Borrower$$

Loan-number	Branch-name	amount	Customer-name	Loan-number
L-170	Khobar	3000	Adel	L-170
L-230	Riyadh	4000	Sami	L-230

- The problem with Equijoin is Pairs of attributes with identical values in evey tuple.

--- Natural-Join Operation

- Is denoted by: $r * s$
- Let r and s be relations on schemas R and S respectively. Then, $r * s$ is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from r and t_s from s.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple t to the result, where
    - t has the same value as t_r on r
    - t has the same value as t_s on s
- Example:

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

  $S = (E, B, D)$

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

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

---- Natural Join Operation – Example 1

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

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

s

r * s →

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

Look for: r.B=s.B ^ r.D = s.D

---- Natural Join – Example 2

*loan * Borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Khobar	3000	Adel
L-230	Riyadh	4000	Sami

- Unlike Equijoin, no pairs of attributes with identical values in evey tuple.

--- Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form 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.
    - Will study precise meaning of comparisons with nulls later

---- Left Outer Join – Example

loan ⟕ *Borrower*

↓

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Khobar	3000	Adel
L-230	Riyadh	4000	Sami
L-260	Dammam	1700	*null*

---- Right Outer Join – Example

loan $⟖$ *borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Khobar	3000	Adel
L-230	Riyadh	4000	Sami
L-155	*null*	*null*	Hashim

---- Full Outer Join – Example

$$loan \;⟗\; borrower$$

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Khobar	3000	Adel
L-230	Riyadh	4000	Sami
L-260	Dammam	1700	*null*
L-155	*null*	*null*	Hashim

-- OUTER UNION Operation

- Outer Union operation compute the union of two relations if the relations are partially union compatible.
- Characteristics:
  - The list of compatible attributes includes a key for both relations.
  - Tuples from the component relations with the same key are presented only once in the result and have values for all attributes in the result.
  - The attributes that are not union compatible from either relation are kept in the result.
  - Tuples that have no values for these attributes are padded with null values.
  - OUTER UNION is equivalent to a FULL OUTER JOIN if the join attributes are all the common attributes of the two relations.

--- OUTER UNION Operation: Example

Non-compatible Attributes

Name	SSN	Dept	Advisor
Ali	111	COE	Sami
Adel	222	EE	Khaled
Fahd	333	COE	Sami

U

Name	SSN	Dept	Rank
Sami	444	COE	FP
Khaled	555	EE	AP
Adel	222	EE	TA

Name	SSN	Dept	Advisor	Rank
Ali	111	COE	Sami	null
Adel	222	EE	Khaled	TA
Fahd	333	COE	Sami	null
Sami	444	COE	null	FP
Khaled	555	EE	null	AP

-- Composition of Operation – Example

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

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

r

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

A	B	C	D
α	α	1	7
β	β	23	10

$\prod_{A,C}($ $)$

A	C
α	1
β	23

-- 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 1

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

r

$$\mathcal{G}_{\text{sum}(c)}(r) \longrightarrow$$

sum-C
27

--- Aggregate Operation – Example 2

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

branch-name	***account-number***	***balance***
Dammam	**A-102**	**400**
Dammam	**A-201**	**900**
Khobar	**A-217**	**750**
Khobar	**A-215**	**750**
Hafuf	**A-222**	**700**

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

branch-name	***balance***
Dammam	**1300**
Khobar	**1500**
Hafuf	**700**

--- Aggregate Functions: Renaming

- 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

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

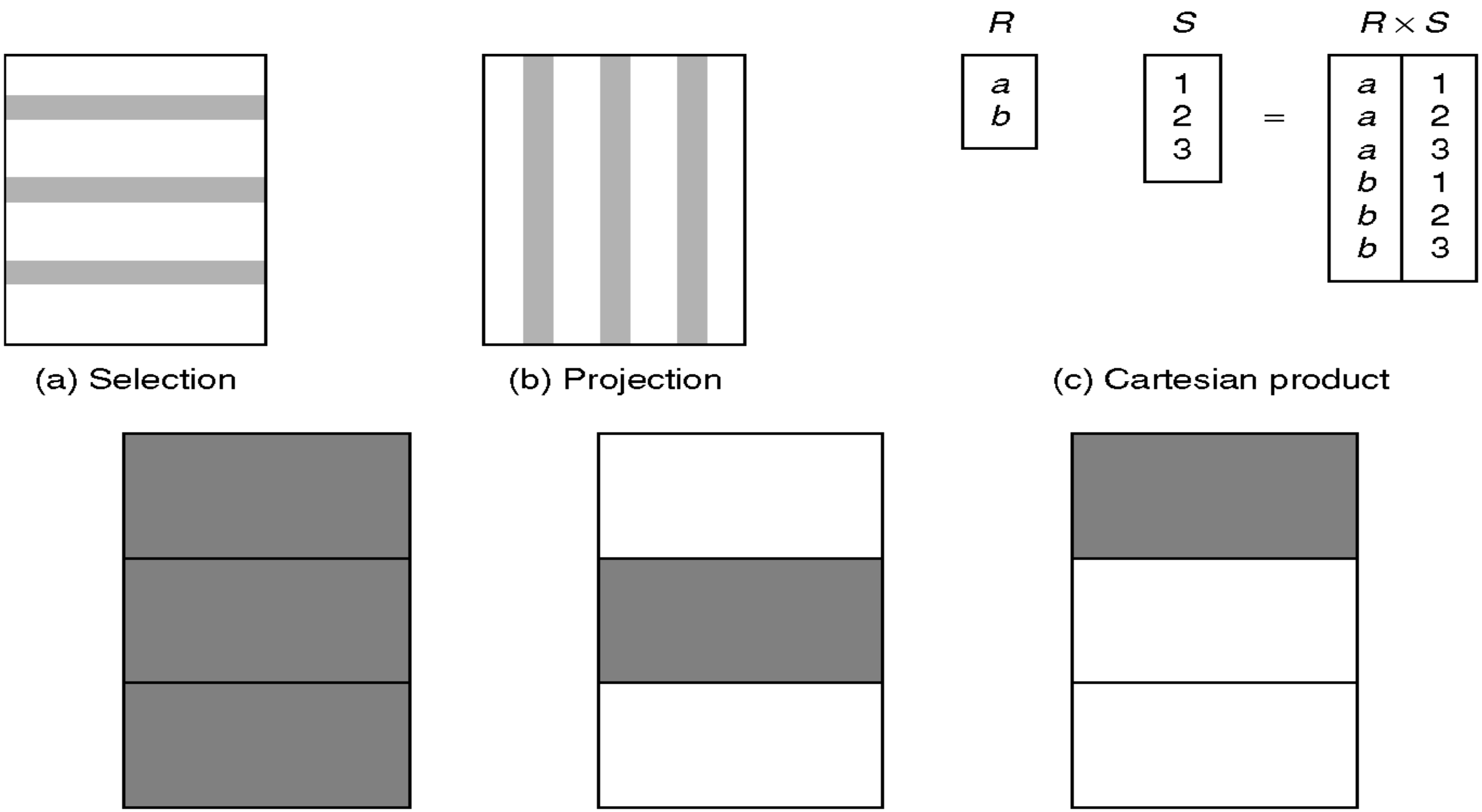
-- Null Values …

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null.*
- Aggregate functions simply ignore null values
  - 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*

- Summary …



(a) Selection (b) Projection (c) Cartesian product

(d) Union (e) Intersection (f) Set difference

… - Summary …

T

A	B
a	1
b	2

U

B	C
1	x
1	y
3	z

$T \bowtie U$

A	B	C
a	1	x
a	1	y

(g) Natural join

$T \ltimes_B U$

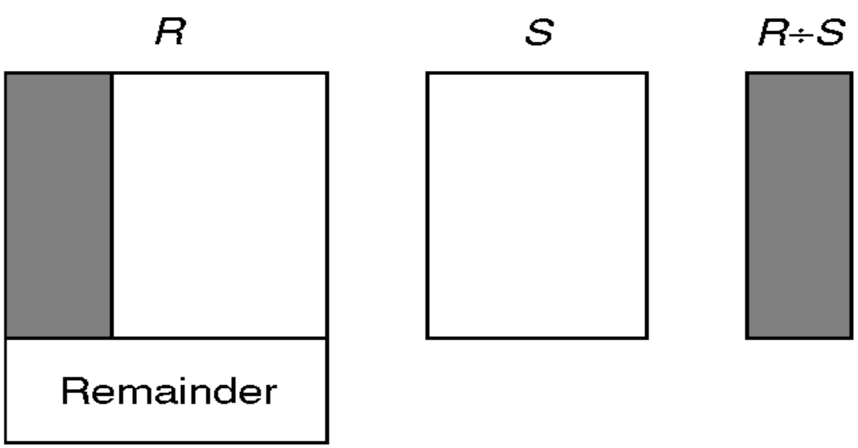
A	B
a	1

(h) Semi-join

$T ⟕_C U$

A	B	C
a	1	x
a	1	y
b	2	

(i) Left outer-join



(j) Division (shaded area)

V

A	B
a	1
a	2
b	1
b	2
c	1

W

B
1
2

$V \div W$

A
a
b

Example of division

... - Summary

Select	σ
Project	Π
Rename	ρ
Union	$\cup$
Difference	$-$
Intersection	$\cap$
Division	$\div$
Assignment	$\leftarrow$
Cartesian Product	X
Join	$\bowtie$
Natural Join	*
Left Outer Join	⟕
Right Outer Join	⟖
Full Outer Join	⟗
Aggregate Function	$\mathcal{g}$

- Example Queries …

- **The following Relations are used for the coming Examples.**
  - 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 …

- Find all loans of over $1200

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

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

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

… - 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 KFUPM branch.

$$\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name="KFUPM"}(borrower * loan))$$

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

$$\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name = "KFUPM"}(\text{borrower} * \text{loan})) - \Pi_{customer\text{-}name}(\text{depositor})$$

... - Example Queries ...

- Find the names of all customers who have a loan at the KFUPM branch.
  - Query 1

$$\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name\,=\,\text{“KFUPM”}}\ (borrower * loan))$$

… - Example Queries …

- Find the largest account balance. Rename *account* relation as *d*

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

… - Example Queries …

- Find all customers who have an account from at least the "Dammam" and the "Khobar" branches.

Query 1

$$\Pi_{CN}(\sigma_{BN=\text{"Dammam"}}(depositor * account)) \cap$$
$$\Pi_{CN}(\sigma_{BN=\text{"Khobar"}}(depositor * account))$$

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

Query 2

$$\Pi_{customer\text{-}name,\ branch\text{-}name}\ (depositor * account)$$
$$\div\ \rho_{temp}(branch\text{-}name)\ (\{(\text{"Dammam"}), (\text{"Khobar"})\})$$

… - Example Queries

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

$$\prod_{customer\text{-}name,\ branch\text{-}name} (depositor * account)$$
$$\div \prod_{branch\text{-}name} (\sigma_{branch\text{-}city = \text{“Dammam”}} (branch))$$