CS315: Database Systems
Relational Algebra

Arnab Bhattacharya
arnabb@cse.iitk.ac.in

Computer Science and Engineering,
Indian Institute of Technology, Kanpur
http://web.cse.iitk.ac.in/~cs315/

2nd semester, 2018-19

Mon 12:00-13:15, Tue 9:00-10:15

Relational Algebra

- *Procedural language* to specify database queries
- Operators are functions from one or two input relations to an output relation
  1. Select: σ
  2. Project: Π
  3. Union: $\cup$
  4. Set Difference: $-$
  5. Cartesian Product: $\times$
  6. Rename: ρ

Relational Algebra

- *Procedural language* to specify database queries
- Operators are functions from one or two input relations to an output relation
  1. Select: σ
  2. Project: Π
  3. Union: $\cup$
  4. Set Difference: $-$
  5. Cartesian Product: $\times$
  6. Rename: ρ
- Uses *propositional calculus* consisting of *expressions* connected by
  1. and: $\wedge$
  2. or: $\vee$
  3. not: $\neg$
- Each term is of the form

  `<attr/const> comparator <attr/const>`

  where `comparator` is one of $=, \neq, >, \geq, <, \leq$

Select

- $\sigma_p(r) = \{t | t \in r \text{ and } p(t)\}$
- p is called the selection predicate
- Select all tuples from r that satisfies the predicate p
- Schema is

Select

- $\sigma_p(r) = \{t | t \in r \text{ and } p(t)\}$
- p is called the selection predicate
- Select all tuples from r that satisfies the predicate p
- Schema is not changed
- Applying $\sigma_{A=B \wedge D>5}$ on

A	B	C	D
1	1	2	7
1	2	5	7
2	2	9	3
2	2	8	6

returns

Select

- $\sigma_p(r) = \{t | t \in r \text{ and } p(t)\}$
- p is called the selection predicate
- Select all tuples from r that satisfies the predicate p
- Schema is not changed
- Applying $\sigma_{A=B \wedge D>5}$ on

A	B	C	D
1	1	2	7
1	2	5	7
2	2	9	3
2	2	8	6

returns

A	B	C	D
1	1	2	7
2	2	8	6

Project

- $\Pi_{A_1,\ldots,A_k}(r)$
- A_i, etc. are attributes of r
- Select only the specified attributes $A_1, \ldots, A_k$ from all tuples of r
- Duplicate rows are removed, since relations are sets

Project

- $\Pi_{A_1,\ldots,A_k}(r)$
- A_i, etc. are attributes of r
- Select only the specified attributes $A_1, \ldots, A_k$ from all tuples of r
- Duplicate rows are removed, since relations are sets
- Schema is

Project

- $\Pi_{A_1,\ldots,A_k}(r)$
- A_i, etc. are attributes of r
- Select only the specified attributes $A_1, \ldots, A_k$ from all tuples of r
- Duplicate rows are removed, since relations are sets
- Schema is changed
- Applying $\Pi_{A,C}$ on

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8

returns

Project

- $\Pi_{A_1,\ldots,A_k}(r)$
- A_i, etc. are attributes of r
- Select only the specified attributes $A_1, \ldots, A_k$ from all tuples of r
- Duplicate rows are removed, since relations are sets
- Schema is changed
- Applying $\Pi_{A,C}$ on

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8

returns

A	C
1	5
2	5
2	8

Set Union

- $r \cup s = \{t | t \in r \text{ or } t \in s\}$
- Relations r and s must have the same *arity* (i.e., number of attributes)
- They must have same *type* of attribute in each column as well, i.e., attribute domains must be *compatible*
- If attribute names are not same, renaming should be used
- Schema is

Set Union

- $r \cup s = \{t | t \in r \text{ or } t \in s\}$
- Relations r and s must have the same *arity* (i.e., number of attributes)
- They must have same *type* of attribute in each column as well, i.e., attribute domains must be *compatible*
- If attribute names are not same, renaming should be used
- Schema is not changed
- Applying $\cup$ on

A	B
1	1
1	2
2	1

and

A	B
1	2
2	3

returns

Set Union

- $r \cup s = \{t | t \in r \text{ or } t \in s\}$
- Relations *r* and *s* must have the same *arity* (i.e., number of attributes)
- They must have same *type* of attribute in each column as well, i.e., attribute domains must be *compatible*
- If attribute names are not same, renaming should be used
- Schema is not changed
- Applying ∪ on

A	B
1	1
1	2
2	1

and

A	B
1	2
2	3

returns

A	B
1	1
1	2
2	1
2	3

Set Difference

- $r - s = \{t | t \in r \text{ and } t \notin s\}$
- Relations *r* and *s* must have the same *arity* (i.e., number of attributes)
- They must have same *type* of attribute in each column as well, i.e., attribute domains must be *compatible*
- If attribute names are not same, renaming should be used
- Schema is

Set Difference

- $r - s = \{t | t \in r \text{ and } t \notin s\}$
- Relations r and s must have the same *arity* (i.e., number of attributes)
- They must have same *type* of attribute in each column as well, i.e., attribute domains must be *compatible*
- If attribute names are not same, renaming should be used
- Schema is not changed
- Applying – on

A	B
1	1
1	2
2	1

and

A	B
1	2
2	3

returns

Set Difference

- $r - s = \{t | t \in r \text{ and } t \notin s\}$
- Relations r and s must have the same *arity* (i.e., number of attributes)
- They must have same *type* of attribute in each column as well, i.e., attribute domains must be *compatible*
- If attribute names are not same, renaming should be used
- Schema is not changed
- Applying – on

A	B
1	1
1	2
2	1

and

A	B
1	2
2	3

returns

A	B
1	1
2	1

Cartesian Product

- $r \times s = \{t\,q | t \in r \text{ and } q \in s\}$
- Attributes of relations r and s should be disjoint
- If attributes are not disjoint, renaming should be used
- Schema is

Cartesian Product

- $r \times s = \{t\, q | t \in r \text{ and } q \in s\}$
- Attributes of relations r and s should be disjoint
- If attributes are not disjoint, renaming should be used
- Schema is changed
- Applying $\times$ on

A	B
1	1
2	2

and

C	D	E
1	2	7
2	6	8
5	7	9

returns

Cartesian Product

- $r \times s = \{t\,q | t \in r \text{ and } q \in s\}$
- Attributes of relations *r* and *s* should be disjoint
- If attributes are not disjoint, renaming should be used
- Schema is changed
- Applying $\times$ on

A	B
1	1
2	2

and

C	D	E
1	2	7
2	6	8
5	7	9

returns

A	B	C	D	E
1	1	1	2	7
1	1	2	6	8
1	1	5	7	9
2	2	1	2	7
2	2	2	6	8
2	2	5	7	9

Rename

- $\rho_N(E)$ returns E, but under the new name N
- For n-ary relations, $\rho_{N(A_1,\ldots,A_n)}(E)$ returns the result of expression E, but under the new name N and the attributes renamed to A_1, etc.
- Schema is

Rename

- $\rho_N(E)$ returns E, but under the new name N
- For n-ary relations, $\rho_{N(A_1,\dots,A_n)}(E)$ returns the result of expression E, but under the new name N and the attributes renamed to A_1, etc.
- Schema is changed although its meaning is not
- Applying $\rho_{s(C,D)}$ on $r(A, B)$

A	B
1	1
1	2
2	3
2	4

returns

Rename

- $\rho_N(E)$ returns E, but under the new name N
- For n-ary relations, $\rho_{N(A_1,\ldots,A_n)}(E)$ returns the result of expression E, but under the new name N and the attributes renamed to A_1, etc.
- Schema is changed although its meaning is not
- Applying $\rho_{s(C,D)}$ on $r(A, B)$

A	B
1	1
1	2
2	3
2	4

returns

C	D
1	1
1	2
2	3
2	4

Additional Operations

- Additional operators have been defined
  1. Set Intersection: $\cap$
  2. Join: $\bowtie$
  3. Division: $\div$
  4. Assignment: $\leftarrow$
- These do not add any power to the basic relational algebra
  - They can be defined using the six basic operators
- However, they simplify queries

Set Intersection

- $r \cap s = \{t | t \in r \text{ and } t \in s\}$
- Relations r and s must have the same *arity* (i.e., number of attributes)
- They must have same *type* of attribute in each column as well, i.e., attribute domains must be *compatible*
- If attribute names are not same, renaming should be used
- Schema is

Set Intersection

- $r \cap s = \{t | t \in r \text{ and } t \in s\}$
- Relations r and s must have the same *arity* (i.e., number of attributes)
- They must have same *type* of attribute in each column as well, i.e., attribute domains must be *compatible*
- If attribute names are not same, renaming should be used
- Schema is not changed
- Applying $\cap$ on

A	B
1	1
1	2
2	1

and

A	B
1	2
2	3

returns

Set Intersection

- $r \cap s = \{t | t \in r \text{ and } t \in s\}$
- Relations r and s must have the same *arity* (i.e., number of attributes)
- They must have same *type* of attribute in each column as well, i.e., attribute domains must be *compatible*
- If attribute names are not same, renaming should be used
- Schema is not changed
- Applying $\cap$ on

A	B
1	1
1	2
2	1

and

A	B
1	2
2	3

returns

A	B
1	2

Set Intersection

- $r \cap s = \{t | t \in r \text{ and } t \in s\}$
- Relations r and s must have the same *arity* (i.e., number of attributes)
- They must have same *type* of attribute in each column as well, i.e., attribute domains must be *compatible*
- If attribute names are not same, renaming should be used
- Schema is not changed
- Applying $\cap$ on

A	B
1	1
1	2
2	1

and

A	B
1	2
2	3

returns

A	B
1	2

- $r \cap s = r - (r - s)$

Join

- $r \bowtie_\theta s = \sigma_\theta(r \times s)$
- Join is too common a query to not have its own operator
- Schema is

Join

- $r \bowtie_\theta s = \sigma_\theta(r \times s)$
- Join is too common a query to not have its own operator
- Schema is same as $r \times s$ but (potentially) less number of tuples
- The above form is the most general, called the theta join

Join

- $r \bowtie_\theta s = \sigma_\theta(r \times s)$
- Join is too common a query to not have its own operator
- Schema is same as $r \times s$ but (potentially) less number of tuples
- The above form is the most general, called the theta join
- Equality join: When the join condition only contains equality
  - $r \bowtie_{B=C} s$

Join

- $r \bowtie_\theta s = \sigma_\theta(r \times s)$
- Join is too common a query to not have its own operator
- Schema is same as $r \times s$ but (potentially) less number of tuples
- The above form is the most general, called the theta join
- Equality join: When the join condition only contains equality
  - $r \bowtie_{B=C} s$
- Natural join: If two relations share an attribute (also its *name*), equality join on that common attribute
  - Denoted by $*$ or simply $\bowtie$ without any predicate
  - Changes schema by retaining only one copy of common attribute
  - $r * s = r \bowtie s = r \bowtie_{r.A=s.A} s$
- Applying $\bowtie$ on

A	B
1	1
1	2
2	1

and

A	C
1	2
2	3

returns

Join

- $r \bowtie_\theta s = \sigma_\theta(r \times s)$
- Join is too common a query to not have its own operator
- Schema is same as $r \times s$ but (potentially) less number of tuples
- The above form is the most general, called the theta join
- Equality join: When the join condition only contains equality
  - $r \bowtie_{B=C} s$
- Natural join: If two relations share an attribute (also its *name*), equality join on that common attribute
  - Denoted by $*$ or simply $\bowtie$ without any predicate
  - Changes schema by retaining only one copy of common attribute
  - $r * s = r \bowtie s = r \bowtie_{r.A=s.A} s$
- Applying $\bowtie$ on

A	B
1	1
1	2
2	1

and

A	C
1	2
2	3

returns

A	B	C
1	1	2
1	2	2
2	1	3

Division

- $r \div s = \{t | t \in \Pi_{R-S}(r) \text{ and } \forall u \in s (tu \in r)\}$
- $q = r \div s$ is the largest relation satisfying $q \times s \subseteq r$

Division

- $r \div s = \{t | t \in \Pi_{R-S}(r) \text{ and } \forall u \in s(tu \in r)\}$
- $q = r \div s$ is the largest relation satisfying $q \times s \subseteq r$
- Relation r must have a schema that is a proper superset of the schema of s, i.e., $S \subset R$
- Used for queries of the form "for all"
- Schema is

Division

- $r \div s = \{t | t \in \Pi_{R-S}(r) \text{ and } \forall u \in s(tu \in r)\}$
- $q = r \div s$ is the largest relation satisfying $q \times s \subseteq r$
- Relation r must have a schema that is a proper superset of the schema of s, i.e., $S \subset R$
- Used for queries of the form "for all"
- Schema is changed to $R - S$
- Applying $\div$ on

A	B
1	5
1	6
1	7
2	5
2	6
3	5
3	7
4	5

and

B
5
6

returns

Division

- $r \div s = \{t | t \in \Pi_{R-S}(r) \text{ and } \forall u \in s (tu \in r)\}$
- $q = r \div s$ is the largest relation satisfying $q \times s \subseteq r$
- Relation r must have a schema that is a proper superset of the schema of s, i.e., $S \subset R$
- Used for queries of the form "for all"
- Schema is changed to $R - S$
- Applying $\div$ on

A	B
1	5
1	6
1	7
2	5
2	6
3	5
3	7
4	5

and

B
5
6

returns

A
1
2

Division (contd.)

- Applying $\div$ on

A	B	C	D
1	5	2	7
1	5	3	7
1	6	3	7
2	6	2	7
2	6	3	7
3	6	2	7
3	6	3	7
3	5	3	7

and

C	D
2	7
3	7

returns

Division (contd.)

- Applying $\div$ on

A	B	C	D
1	5	2	7
1	5	3	7
1	6	3	7
2	6	2	7
2	6	3	7
3	6	2	7
3	6	3	7
3	5	3	7

and

C	D
2	7
3	7

returns

A	B
1	5
2	6
3	6

Division (contd.)

- Applying $\div$ on

A	B	C	D
1	5	2	7
1	5	3	7
1	6	3	7
2	6	2	7
2	6	3	7
3	6	2	7
3	6	3	7
3	5	3	7

and

C	D
2	7
3	7

returns

A	B
1	5
2	6
3	6

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

Assignment

- $s \leftarrow E(r)$ assigns the relation resulting from applying E on r to s
- Useful in complex queries to hold intermediate values
  - Can be used sequentially
- Schema is

Assignment

- $s \leftarrow E(r)$ assigns the relation resulting from applying E on r to s
- Useful in complex queries to hold intermediate values
  - Can be used sequentially
- Schema is not changed
- $\rho_{(A=B)}(\Pi_{A,B}(r))$ can be broken into $s \leftarrow \Pi_{A,B}(r)$ and $\rho_{(A=B)}(s)$

r

A	B	C
1	1	7
2	2	8
5	7	9

Assignment

- $s \leftarrow E(r)$ assigns the relation resulting from applying E on r to s
- Useful in complex queries to hold intermediate values
  - Can be used sequentially
- Schema is not changed
- $\rho_{(A=B)}(\Pi_{A,B}(r))$ can be broken into $s \leftarrow \Pi_{A,B}(r)$ and $\rho_{(A=B)}(s)$

r

A	B	C
1	1	7
2	2	8
5	7	9

s

A	B
1	1
2	2
5	7

and

$\rho_{(A=B)}(s)$

A	B
1	1
2	2

Composition of Operators

- Expressions can be built using multiple operators
- Applying $\sigma_{A=C}(r \times s)$ on

A	B
1	1
2	2

and

C	D	E
1	2	7
2	6	8
5	7	9

intermediately produces

A	B	C	D	E
1	1	1	2	7
1	1	2	6	8
1	1	5	7	9
2	2	1	2	7
2	2	2	6	8
2	2	5	7	9

and finally returns

A	B	C	D	E
1	1	1	2	7
2	2	2	6	8

Precedence and Associativity

- Precedence is generally assumed to be

Precedence	Operators
Highest	σ, Π, ρ
Medium	$\bowtie$, $\bowtie_\theta$, $\times$
Lowest	$\cup$, $\cap$, $-$

- Associativity is assumed to be left-to-right
- Not part of definition
- Therefore, best to use explicit brackets

Example Schema

- course (code, title, *ctype*, webpage)
- coursetype (ctype, *dept*)
- faculty (fid, name, *dept*, designation)
- department (deptid, name)
- semester (yr, half)
- offering (*coursecode, yr, half, instructor*)
- student (roll, name, *dept*)
- program (*roll*, ptype)
- registration (*coursecode, roll, yr, half*, grade)

Example Queries

- Find all courses offered in the year 2018

Example Queries

- Find all courses offered in the year 2018

$$\sigma_{\text{yr}=2018}(\text{offering})$$

Example Queries

- Find all courses offered in the year 2018

$$\sigma_{\text{yr}=2018}(\text{offering})$$

- Find the course codes for all courses offered in the year 2018

Example Queries

- Find all courses offered in the year 2018
$$\sigma_{\text{yr}=2018}(\text{offering})$$
- Find the course codes for all courses offered in the year 2018
$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$

Example Queries

- Find all courses offered in the year 2018
$$\sigma_{\text{yr}=2018}(\text{offering})$$
- Find the course codes for all courses offered in the year 2018
$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$
- Find the course codes for all the courses offered in either of the years 2017 and 2018

Example Queries

- Find all courses offered in the year 2018
$$\sigma_{\text{yr}=2018}(\text{offering})$$

- Find the course codes for all courses offered in the year 2018
$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$

- Find the course codes for all the courses offered in either of the years 2017 and 2018
$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2017}(\text{offering})) \cup \Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$

Example Queries

- Find all courses offered in the year 2018
$$\sigma_{\text{yr}=2018}(\text{offering})$$
- Find the course codes for all courses offered in the year 2018
$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$
- Find the course codes for all the courses offered in either of the years 2017 and 2018
$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2017}(\text{offering})) \cup \Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$
- Find the course codes for all the courses offered in the year 2017 but not in 2018

Example Queries

- Find all courses offered in the year 2018

$$\sigma_{\text{yr}=2018}(\text{offering})$$

- Find the course codes for all courses offered in the year 2018

$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$

- Find the course codes for all the courses offered in either of the years 2017 and 2018

$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2017}(\text{offering})) \cup \Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$

- Find the course codes for all the courses offered in the year 2017 but not in 2018

$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2017}(\text{offering})) - \Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$

Example Queries

- Find all courses offered in the year 2018
$$\sigma_{\text{yr}=2018}(\text{offering})$$
- Find the course codes for all courses offered in the year 2018
$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$
- Find the course codes for all the courses offered in either of the years 2017 and 2018
$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2017}(\text{offering})) \cup \Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$
- Find the course codes for all the courses offered in the year 2017 but not in 2018
$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2017}(\text{offering})) - \Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$
- Find the course codes for all the courses offered in both the years 2017 and 2018

Example Queries

- Find all courses offered in the year 2018
$$\sigma_{\text{yr}=2018}(\text{offering})$$
- Find the course codes for all courses offered in the year 2018
$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$
- Find the course codes for all the courses offered in either of the years 2017 and 2018
$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2017}(\text{offering})) \cup \Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$
- Find the course codes for all the courses offered in the year 2017 but not in 2018
$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2017}(\text{offering})) - \Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$
- Find the course codes for all the courses offered in both the years 2017 and 2018
$$\Pi_{\text{coursecode}}(\sigma_{\text{yr}=2017}(\text{offering})) \cap \Pi_{\text{coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}))$$

Example Queries

- Find the titles of all courses offered in the year 2018

Example Queries

- Find the titles of all courses offered in the year 2018

  $\Pi_{\text{title}}(\sigma_{\text{yr}=2018}(\sigma_{\text{offering.coursecode}=\text{courses.coursecode}}(\text{offering} \times \text{courses})))$

Example Queries

- Find the titles of all courses offered in the year 2018

$$\Pi_{\text{title}}(\sigma_{\text{yr}=2018}(\sigma_{\text{offering.coursecode}=\text{courses.coursecode}}(\text{offering}\times\text{courses})))$$

$$\Pi_{\text{title}}(\sigma_{\text{offering.coursecode}=\text{courses.coursecode}}(\sigma_{\text{yr}=2018}(\text{offering})\times\text{courses}))$$

Example Queries

- Find the titles of all courses offered in the year 2018

$\Pi_{\text{title}}(\sigma_{\text{yr}=2018}(\sigma_{\text{offering.coursecode}=\text{courses.coursecode}}(\text{offering}\times\text{courses})))$

$\Pi_{\text{title}}(\sigma_{\text{offering.coursecode}=\text{courses.coursecode}}(\sigma_{\text{yr}=2018}(\text{offering})\times\text{courses}))$

$$\Pi_{\text{title}}(\sigma_{\text{yr}=2018}(\text{offering} \bowtie \text{courses}))$$

Example Queries

- Find the titles of all courses offered in the year 2018

$\Pi_{\text{title}}(\sigma_{\text{yr}=2018}(\sigma_{\text{offering.coursecode}=\text{courses.coursecode}}(\text{offering} \times \text{courses})))$

$\Pi_{\text{title}}(\sigma_{\text{offering.coursecode}=\text{courses.coursecode}}(\sigma_{\text{yr}=2018}(\text{offering}) \times \text{courses}))$

$$\Pi_{\text{title}}(\sigma_{\text{yr}=2018}(\text{offering} \bowtie \text{courses}))$$

$$\Pi_{\text{title}}(\sigma_{\text{yr}=2018}(\text{offering}) \bowtie \text{courses})$$

Example Queries

- Find the titles of all courses offered in the year 2018

$$\Pi_{\text{title}}(\sigma_{\text{yr}=2018}(\sigma_{\text{offering.coursecode}=\text{courses.coursecode}}(\text{offering}\times\text{courses})))$$

$$\Pi_{\text{title}}(\sigma_{\text{offering.coursecode}=\text{courses.coursecode}}(\sigma_{\text{yr}=2018}(\text{offering})\times\text{courses}))$$

$$\Pi_{\text{title}}(\sigma_{\text{yr}=2018}(\text{offering} \bowtie \text{courses}))$$

$$\Pi_{\text{title}}(\sigma_{\text{yr}=2018}(\text{offering}) \bowtie \text{courses})$$

- Find the years when *all* the courses of type 5 were offered

Example Queries

- Find the titles of all courses offered in the year 2018

$$\Pi_{\text{title}}(\sigma_{\text{yr}=2018}(\sigma_{\text{offering.coursecode}=\text{courses.coursecode}}(\text{offering}\times\text{courses})))$$

$$\Pi_{\text{title}}(\sigma_{\text{offering.coursecode}=\text{courses.coursecode}}(\sigma_{\text{yr}=2018}(\text{offering})\times\text{courses}))$$

$$\Pi_{\text{title}}(\sigma_{\text{yr}=2018}(\text{offering} \bowtie \text{courses}))$$

$$\Pi_{\text{title}}(\sigma_{\text{yr}=2018}(\text{offering}) \bowtie \text{courses})$$

- Find the years when *all* the courses of type 5 were offered

$$\text{ct} \leftarrow \rho_{\text{coursecode}}(\Pi_{\text{code}}(\sigma_{\text{ctype}=5}(\text{course})))$$

$$(\Pi_{coursecode,yr}(\text{offering})) \div \text{ct}$$

Extended Relational Algebra

- The power of relational algebra can be enhanced by
  1. Generalized Projection
  2. Grouping and Aggregate operations
  3. Outer Join

Generalized Projection

- Extends project operator by allowing arbitrary arithmetic functions in attribute list
- $\Pi_{F_1,\ldots,F_k}(E)$
- F_i, etc. are arithmetic expressions involving constants and attributes in schema of E
- Applying $\Pi_{B-A,2C}$ on r

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8

returns

Generalized Projection

- Extends project operator by allowing arbitrary arithmetic functions in attribute list
- $\Pi_{F_1,\ldots,F_k}(E)$
- F_i, etc. are arithmetic expressions involving constants and attributes in schema of E
- Applying $\Pi_{B-A,2C}$ on r

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8

returns

B-A	2C
0	10
1	10
2	16

Aggregate Operations

- Aggregate functions that can be used are *avg*, *min*, *max*, *sum*, *count*
- Can be applied on groups of tuples as well
- Aggregate operation is of the form ${}_{G_1,\ldots,G_k}\mathcal{G}_{F_1(A_1),\ldots,F_n(A_n)}(E)$ where
  - $G_1, \ldots, G_k$ is the list of attributes on which to group (may be empty)
  - Each F_i is an aggregate function that operates on the attribute A_i
- Applying $\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8

returns

Aggregate Operations

- Aggregate functions that can be used are *avg*, *min*, *max*, *sum*, *count*
- Can be applied on groups of tuples as well
- Aggregate operation is of the form $_{G_1,\dots,G_k}\mathcal{G}_{F_1(A_1),\dots,F_n(A_n)}(E)$ where
  - $G_1, \dots, G_k$ is the list of attributes on which to group (may be empty)
  - Each F_i is an aggregate function that operates on the attribute A_i
- Applying $\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8

returns

sum(C)
23

Aggregate Operations

- First, the tuples are grouped according to $G_1, \ldots, G_k$
- Then, aggregate functions $F_1(A_1), \ldots, F_n(A_n)$ are applied on each group
- Schema changes to $(G_1, \ldots, G_k, F_1(A_1), \ldots, F_n(A_n))$
- Applying ${}_A\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8
3	4	8

returns

Aggregate Operations

- First, the tuples are grouped according to $G_1, \ldots, G_k$
- Then, aggregate functions $F_1(A_1), \ldots, F_n(A_n)$ are applied on each group
- Schema changes to $(G_1, \ldots, G_k, F_1(A_1), \ldots, F_n(A_n))$
- Applying ${}_A\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8
3	4	8

returns

A	sum(C)
1	10
2	13
3	8

Aggregate Operations

- First, the tuples are grouped according to $G_1, \dots, G_k$
- Then, aggregate functions $F_1(A_1), \dots, F_n(A_n)$ are applied on each group
- Schema changes to $(G_1, \dots, G_k, F_1(A_1), \dots, F_n(A_n))$
- Applying ${}_{A}\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8
3	4	8

returns

A	sum(C)
1	10
2	13
3	8

- Applying ${}_{A,B}\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
1	2	4

returns

Aggregate Operations

- First, the tuples are grouped according to $G_1, \ldots, G_k$
- Then, aggregate functions $F_1(A_1), \ldots, F_n(A_n)$ are applied on each group
- Schema changes to $(G_1, \ldots, G_k, F_1(A_1), \ldots, F_n(A_n))$
- Applying ${}_{A}\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8
3	4	8

returns

A	sum(C)
1	10
2	13
3	8

- Applying ${}_{A,B}\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
1	2	4

returns

A	B	sum(C)
1	1	5
1	2	9

Outer Join

- Extension of the join to retain more information
- Computes join and then adds tuples to result that do not match
- Requires use of *null* values
- Left outer join $r ⟕_\theta$ s retains *every* tuple from left or first relation
  - If no matching tuple is found in right or second relation, values are padded with *null*
- Right outer join $r ⟖_\theta$ s is defined analogously
- Full outer join $r ⟗_\theta$ s retains all tuples from both relations
  - Non-matching fields are filled with *null* values

Outer Join

- Extension of the join to retain more information
- Computes join and then adds tuples to result that do not match
- Requires use of *null* values
- Left outer join $r ⟕_\theta s$ retains *every* tuple from left or first relation
  - If no matching tuple is found in right or second relation, values are padded with *null*
- Right outer join $r ⟖_\theta s$ is defined analogously
- Full outer join $r ⟗_\theta s$ retains all tuples from both relations
  - Non-matching fields are filled with *null* values
- Consequently, ordinary join is sometimes called inner join
- "Outer" word is sometimes dropped from join yielding left join, right join and full join

Outer Join

- Extension of the join to retain more information
- Computes join and then adds tuples to result that do not match
- Requires use of *null* values
- Left outer join $r ⟕_\theta s$ retains *every* tuple from left or first relation
  - If no matching tuple is found in right or second relation, values are padded with *null*
- Right outer join $r ⟖_\theta s$ is defined analogously
- Full outer join $r ⟗_\theta s$ retains all tuples from both relations
  - Non-matching fields are filled with *null* values
- Consequently, ordinary join is sometimes called inner join
- "Outer" word is sometimes dropped from join yielding left join, right join and full join
- When no θ condition is specified, it is natural outer join

Outer Join Examples

$$\begin{array}{cc} A & B \\ \hline 1 & 5 \\ 2 & 6 \\ 3 & 7 \end{array} \bowtie \begin{array}{cc} A & C \\ \hline 1 & 7 \\ 2 & 8 \\ 4 & 9 \end{array} =$$

Outer Join Examples

A	B
1	5
2	6
3	7

$\bowtie$

A	C
1	7
2	8
4	9

$=$

A	B	C
1	5	7
2	6	8

A	B
1	5
2	6
3	7

$⟕$

A	C
1	7
2	8
4	9

$=$

Outer Join Examples

$$\begin{array}{cc} A & B \\ \hline 1 & 5 \\ 2 & 6 \\ 3 & 7 \end{array} \bowtie \begin{array}{cc} A & C \\ \hline 1 & 7 \\ 2 & 8 \\ 4 & 9 \end{array} = \begin{array}{ccc} A & B & C \\ \hline 1 & 5 & 7 \\ 2 & 6 & 8 \end{array}$$

$$\begin{array}{cc} A & B \\ \hline 1 & 5 \\ 2 & 6 \\ 3 & 7 \end{array} ⟕ \begin{array}{cc} A & C \\ \hline 1 & 7 \\ 2 & 8 \\ 4 & 9 \end{array} = \begin{array}{ccc} A & B & C \\ \hline 1 & 5 & 7 \\ 2 & 6 & 8 \\ 3 & 7 & \text{null} \end{array}$$

$$\begin{array}{cc} A & B \\ \hline 1 & 5 \\ 2 & 6 \\ 3 & 7 \end{array} ⟖ \begin{array}{cc} A & C \\ \hline 1 & 7 \\ 2 & 8 \\ 4 & 9 \end{array} =$$

Outer Join Examples

A	B
1	5
2	6
3	7

$\bowtie$

A	C
1	7
2	8
4	9

$=$

A	B	C
1	5	7
2	6	8

A	B
1	5
2	6
3	7

⟕

A	C
1	7
2	8
4	9

$=$

A	B	C
1	5	7
2	6	8
3	7	null

A	B
1	5
2	6
3	7

⟖

A	C
1	7
2	8
4	9

$=$

A	B	C
1	5	7
2	6	8
4	null	9

Outer Join Examples

$$\begin{array}{cc} A & B \\ \hline 1 & 5 \\ 2 & 6 \\ 3 & 7 \end{array} \bowtie \begin{array}{cc} A & C \\ \hline 1 & 7 \\ 2 & 8 \\ 4 & 9 \end{array} =$$

Outer Join Examples

$$\begin{array}{cc} A & B \\ \hline 1 & 5 \\ 2 & 6 \\ 3 & 7 \end{array} \bowtie \begin{array}{cc} A & C \\ \hline 1 & 7 \\ 2 & 8 \\ 4 & 9 \end{array} = \begin{array}{ccc} A & B & C \\ \hline 1 & 5 & 7 \\ 2 & 6 & 8 \end{array}$$

$$\begin{array}{cc} A & B \\ \hline 1 & 5 \\ 2 & 6 \\ 3 & 7 \end{array} ⟗ \begin{array}{cc} A & C \\ \hline 1 & 7 \\ 2 & 8 \\ 4 & 9 \end{array} =$$

Outer Join Examples

A	B
1	5
2	6
3	7

$\bowtie$

A	C
1	7
2	8
4	9

$=$

A	B	C
1	5	7
2	6	8

A	B
1	5
2	6
3	7

⟗

A	C
1	7
2	8
4	9

$=$

A	B	C
1	5	7
2	6	8
3	7	null
4	null	9

Example Queries

- Find the total number of courses offered in the year 2018

Example Queries

- Find the total number of courses offered in the year 2018

$$\mathcal{G}_{count(\text{coursecode})}(\sigma_{\text{yr}=2018}(\text{offering}))$$

Example Queries

- Find the total number of courses offered in the year 2018

$$\mathcal{G}_{count(\text{coursecode})}(\sigma_{\text{yr}=2018}(\text{offering}))$$

- For each instructor, find the total number of courses offered by her in the year 2018

Example Queries

- Find the total number of courses offered in the year 2018

$$\mathcal{G}_{count(\text{coursecode})}(\sigma_{\text{yr}=2018}(\text{offering}))$$

- For each instructor, find the total number of courses offered by her in the year 2018

$${}_{\text{instructor}}\mathcal{G}_{count(\text{coursecode})}(\sigma_{\text{yr}=2018}(\text{offering}))$$

Example Queries

- Find the total number of courses offered in the year 2018

$$\mathcal{G}_{count(\text{coursecode})}(\sigma_{\text{yr}=2018}(\text{offering}))$$

- For each instructor, find the total number of courses offered by her in the year 2018

$$_{\text{instructor}}\mathcal{G}_{count(\text{coursecode})}(\sigma_{\text{yr}=2018}(\text{offering}))$$

- For each instructor, find the total number of courses offered by her per year

Example Queries

- Find the total number of courses offered in the year 2018

$$\mathcal{G}_{count(\text{coursecode})}(\sigma_{\text{yr}=2018}(\text{offering}))$$

- For each instructor, find the total number of courses offered by her in the year 2018

$$_{\text{instructor}}\mathcal{G}_{count(\text{coursecode})}(\sigma_{\text{yr}=2018}(\text{offering}))$$

- For each instructor, find the total number of courses offered by her per year

$$_{\text{instructor,yr}}\mathcal{G}_{count(\text{coursecode})}(\text{offering})$$

Example Queries

- Find the total number of courses offered in the year 2018

$$\mathcal{G}_{count(\text{coursecode})}(\sigma_{\text{yr}=2018}(\text{offering}))$$

- For each instructor, find the total number of courses offered by her in the year 2018

$$_{\text{instructor}}\mathcal{G}_{count(\text{coursecode})}(\sigma_{\text{yr}=2018}(\text{offering}))$$

- For each instructor, find the total number of courses offered by her per year

$$_{\text{instructor,yr}}\mathcal{G}_{count(\text{coursecode})}(\text{offering})$$

- For each course, indicate the most recent year it was offered

Example Queries

- Find the total number of courses offered in the year 2018

$$\mathcal{G}_{count(\text{coursecode})}(\sigma_{\text{yr}=2018}(\text{offering}))$$

- For each instructor, find the total number of courses offered by her in the year 2018

$$_{\text{instructor}}\mathcal{G}_{count(\text{coursecode})}(\sigma_{\text{yr}=2018}(\text{offering}))$$

- For each instructor, find the total number of courses offered by her per year

$$_{\text{instructor,yr}}\mathcal{G}_{count(\text{coursecode})}(\text{offering})$$

- For each course, indicate the most recent year it was offered

$$\text{course-offering} \leftarrow {}_{\text{coursecode}}\mathcal{G}_{max(\text{yr})}(\text{offering})$$

$$\text{course-year} \leftarrow \rho_{(\text{code,yr})}(\Pi_{\text{coursecode},max(\text{yr})}(\text{course-offering}))$$

$$\text{course} ⟕ \text{course-year}$$

Null Values

- Null denotes an unknown or missing value
- Arithmetic expressions involving null evaluate to null
- Aggregate functions ignore null
- Duplicate elimination and grouping treats null as any other value, i.e., two null values are same
    - null = null evaluates to true

Truth Tables with Null Values

- Comparison with null otherwise returns *unknown*, not false
- If false is used, consider two expressions $not(A < 5)$ and $A \geq 5$ and when attribute contains null
  - They will not be the same
- Three-valued logic with *unknown*
  - Or
    - unknown or true = true
    - unknown or false = unknown
    - unknown or unknown = unknown
  - And
    - unknown and true = unknown
    - unknown and false = false
    - unknown and unknown = unknown
  - Not
    - not unknown = unknown
- Select operation treats unknown as false

Database Modification

- Contents of a database may be modified by
  1. Deletion
  2. Insertion
  3. Updating
- Assignment operator is used to express these operations

Deletion

- $r \leftarrow r - E$ deletes tuples in the result set of the query E from the relation r
- Only whole tuples can be deleted, not some attributes
- Applying $r \leftarrow r - \sigma_{A=1}(r)$ on

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8

returns

Deletion

- $r \leftarrow r - E$ deletes tuples in the result set of the query E from the relation r
- Only whole tuples can be deleted, not some attributes
- Applying $r \leftarrow r - \sigma_{A=1}(r)$ on

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8

returns

A	B	C
2	3	5
2	4	8

Insertion

- $r \leftarrow r \cup E$ inserts tuples in the result set of the query E into the relation r
- Only whole tuples can be inserted, not some attributes
- If a specific tuple needs to be inserted, E is specified as a relation containing only that tuple
- Applying $r \leftarrow r \cup \{(1, 2, 5)\}$ on

A	B	C
1	1	5
2	3	5
2	4	8

returns

Insertion

- $r \leftarrow r \cup E$ inserts tuples in the result set of the query E into the relation r
- Only whole tuples can be inserted, not some attributes
- If a specific tuple needs to be inserted, E is specified as a relation containing only that tuple
- Applying $r \leftarrow r \cup \{(1, 2, 5)\}$ on

A	B	C
1	1	5
2	3	5
2	4	8

returns

A	B	C
1	1	5
2	3	5
2	4	8
1	2	5

Updation

- Updates allow values of only some attributes to change
- $r \leftarrow \Pi_{F_1,\ldots,F_n}(r)$ where each F_i is
  - Either the *i*th attribute of r if it is not to be changed
  - Or the result of the expression F_i involving constants and attributes resulting in the new value of the *i*th attribute
- Applying $r \leftarrow \Pi_{A,2*B,C}(r)$ on

A	B	C
1	2	5
1	1	5
2	4	8

returns

Updation

- Updates allow values of only some attributes to change
- $r \leftarrow \Pi_{F_1,\ldots,F_n}(r)$ where each F_i is
  - Either the ith attribute of r if it is not to be changed
  - Or the result of the expression F_i involving constants and attributes resulting in the new value of the ith attribute
- Applying $r \leftarrow \Pi_{A,2*B,C}(r)$ on

A	B	C
1	2	5
1	1	5
2	4	8

returns

A	B	C
1	4	5
1	2	5
2	8	8

- Applying $r \leftarrow \Pi_{A,2*B,C}(\sigma_{A=1}(r))$ on

A	B	C
1	2	5
1	1	5
2	4	8

returns

Updation

- Updates allow values of only some attributes to change
- $r \leftarrow \Pi_{F_1,\ldots,F_n}(r)$ where each F_i is
  - Either the ith attribute of r if it is not to be changed
  - Or the result of the expression F_i involving constants and attributes resulting in the new value of the ith attribute
- Applying $r \leftarrow \Pi_{A,2*B,C}(r)$ on

A	B	C
1	2	5
1	1	5
2	4	8

returns

A	B	C
1	4	5
1	2	5
2	8	8

- Applying $r \leftarrow \Pi_{A,2*B,C}(\sigma_{A=1}(r))$ on

A	B	C
1	2	5
1	1	5
2	4	8

returns

A	B	C
1	4	5
1	2	5

Example Queries

- Create a new department “Astronomy” with id 18

Example Queries

- Create a new department “Astronomy” with id 18

$$\text{department} \leftarrow \text{department} \cup \{18, \text{“Astronomy”}\}$$

Example Queries

- Create a new department "Astronomy" with id 18

$$\text{department} \leftarrow \text{department} \cup \{18, \text{"Astronomy"}\}$$

- Delete the course whose code is "CS200"

Example Queries

- Create a new department "Astronomy" with id 18

$$\text{department} \leftarrow \text{department} \cup \{18, \text{"Astronomy"}\}$$

- Delete the course whose code is "CS200"

$$\text{course} \leftarrow \text{course} - \sigma_{\text{code}=\text{"CS200"}}(\text{course})$$

Example Queries

- Create a new department “Astronomy” with id 18

$$\text{department} \leftarrow \text{department} \cup \{18, \text{“Astronomy”}\}$$

- Delete the course whose code is “CS200”

$$\text{course} \leftarrow \text{course} - \sigma_{\text{code}=\text{“CS200”}}(\text{course})$$

- Update the title of the course “DBMS” to “Database Systems”

Example Queries

- Create a new department "Astronomy" with id 18

$$\text{department} \leftarrow \text{department} \cup \{18, \text{“Astronomy”}\}$$

- Delete the course whose code is "CS200"

$$\text{course} \leftarrow \text{course} - \sigma_{\text{code}=\text{“CS200”}}(\text{course})$$

- Update the title of the course "DBMS" to "Database Systems"

$$\text{old-course} \leftarrow \sigma_{\text{title}=\text{“DBMS”}}(\text{course})$$

$$\text{updated-course} \leftarrow \Pi_{\text{code},\text{“Database Systems”},\text{ctype},\text{webpage}}(\text{old-course})$$

$$\text{course} \leftarrow (\text{course} \cup \text{updated-course}) - \text{old-course}$$

Integrity Constraints Violations

- Deletion may violate

Integrity Constraints Violations

- Deletion may violate
  - Referential integrity: If a primary key is deleted, the corresponding foreign referencing key becomes orphan
    - Should be restricted (rejected) or cascaded or set to null
- Insertion may violate

Integrity Constraints Violations

- Deletion may violate
  - Referential integrity: If a primary key is deleted, the corresponding foreign referencing key becomes orphan
    - Should be restricted (rejected) or cascaded or set to null
- Insertion may violate
  - Referential integrity: If a foreign key is inserted, the corresponding primary referenced key must be present
    - Should be restricted

Integrity Constraints Violations

- Deletion may violate
  - Referential integrity: If a primary key is deleted, the corresponding foreign referencing key becomes orphan
    - Should be restricted (rejected) or cascaded or set to null
- Insertion may violate
  - Referential integrity: If a foreign key is inserted, the corresponding primary referenced key must be present
    - Should be restricted
  - Domain constraint: Value is outside the domain
    - Should be restricted or domain updated

Integrity Constraints Violations

- Deletion may violate
  - Referential integrity: If a primary key is deleted, the corresponding foreign referencing key becomes orphan
    - Should be restricted (rejected) or cascaded or set to null
- Insertion may violate
  - Referential integrity: If a foreign key is inserted, the corresponding primary referenced key must be present
    - Should be restricted
  - Domain constraint: Value is outside the domain
    - Should be restricted or domain updated
  - Key constraint: If an insertion violates the property of being a key
    - Should be restricted or design modified

Integrity Constraints Violations

- Deletion may violate
  - Referential integrity: If a primary key is deleted, the corresponding foreign referencing key becomes orphan
    - Should be restricted (rejected) or cascaded or set to null
- Insertion may violate
  - Referential integrity: If a foreign key is inserted, the corresponding primary referenced key must be present
    - Should be restricted
  - Domain constraint: Value is outside the domain
    - Should be restricted or domain updated
  - Key constraint: If an insertion violates the property of being a key
    - Should be restricted or design modified
  - Entity integrity: If the primary key of the inserted tuple is null
    - Should be restricted
- Updation may violate

Integrity Constraints Violations

- Deletion may violate
  - Referential integrity: If a primary key is deleted, the corresponding foreign referencing key becomes orphan
    - Should be restricted (rejected) or cascaded or set to null
- Insertion may violate
  - Referential integrity: If a foreign key is inserted, the corresponding primary referenced key must be present
    - Should be restricted
  - Domain constraint: Value is outside the domain
    - Should be restricted or domain updated
  - Key constraint: If an insertion violates the property of being a key
    - Should be restricted or design modified
  - Entity integrity: If the primary key of the inserted tuple is null
    - Should be restricted
- Updation may violate
  - *Referential integrity*
  - *Domain constraint*
  - *Key constraint*
  - *Entity integrity*

Drawbacks of Relational Algebra

Drawbacks of Relational Algebra

- First-order propositional logic
- Do not support recursive closure operations
  - Find supervisors of A at *all* levels
- Needs specifying multiple queries, each solving only one level at a time

Multiset Variant

- Relations are multisets or bags of tuples
  - Duplicate tuples are allowed
  - Select, project, set operations change

Multiset Variant

- Relations are multisets or bags of tuples
  - Duplicate tuples are allowed
  - Select, project, set operations change
- Distinct or duplicate elimination operator: δ
  - Removes duplicate tuples
  - Reduces relation to sets