CS315: Principles of 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, 2013-14

Tue, Fri 1530-1700 at CS101

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
  `<attribute> comparator <attribute/constant>`
  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
- Does not change the schema
- 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
- Changes the schema
- 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

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
- Does not change the schema
- Applying $\cup$ 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
- Does not change the schema
- 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
- Changes the schema
- 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.
- Changes the schema but does not change the meaning of it
- 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

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

Banking example

- branch(bname, bcity, assets)
- customer(cname, cstreet, ccity)
- account(ano, bname, bal)
- loan(lno, bname, amt)
- depositor(cname, ano)
- borrower(cname, lno)

Example queries

- Find all loans of over Rs 100

Example queries

- Find all loans of over Rs 100
  - $\sigma_{amt>100}(loan)$
- Find the loan numbers for each loan of over Rs 100

Example queries

- Find all loans of over Rs 100
  - $\sigma_{amt>100}(loan)$
- Find the loan numbers for each loan of over Rs 100
  - $\Pi_{lno}(\sigma_{amt>100}(loan))$
- Find names of all customers having a loan at "IIT" branch

Example queries

- Find all loans of over Rs 100
  - $\sigma_{amt>100}(loan)$
- Find the loan numbers for each loan of over Rs 100
  - $\Pi_{lno}(\sigma_{amt>100}(loan))$
- Find names of all customers having a loan at "IIT" branch
  - $\Pi_{cname}(\sigma_{bname=''IIT''}(\sigma_{borrower.lno=loan.lno}(borrower \times loan)))$
  - $\Pi_{cname}(\sigma_{loan.lno=borrower.lno}(\sigma_{bname=''IIT''}(loan)) \times borrower)))$
- Find names of all customers having a loan at "IIT" branch but not any account

Example queries

- Find all loans of over Rs 100
  - $\sigma_{amt>100}(loan)$
- Find the loan numbers for each loan of over Rs 100
  - $\Pi_{lno}(\sigma_{amt>100}(loan))$
- Find names of all customers having a loan at "IIT" branch
  - $\Pi_{cname}(\sigma_{bname="IIT"}(\sigma_{borrower.lno=loan.lno}(borrower \times loan)))$
  - $\Pi_{cname}(\sigma_{loan.lno=borrower.lno}(\sigma_{bname="IIT"}(loan)) \times borrower))$
- Find names of all customers having a loan at "IIT" branch but not any account
  - $\Pi_{cname}(\sigma_{bname="IIT"}(\sigma_{borrower.lno=loan.lno}(borrower \times loan))) - \Pi_{cname}(depositor)$

Additional operations

- Additional operators have been defined
  1. Intersection: $\cap$
  2. Join: $\bowtie$
  3. Division: $\div$
  4. Assignment: $\leftarrow$
- These do not add any power to the relational algebra
  - They can be defined using the 6 basic operators
- However, they simplify queries

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
- Does not change the schema
- Applying $\cap$ on

A	B
1	1
1	2
2	1

and

A	B
1	2
2	3

returns

A	B
1	2

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
- Does not change the schema
- 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
- Has the same schema 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
- Has the same schema 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
- Has the same schema 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

- $r \div s = \{t | t \in \Pi_{R-S}(r) \text{ and } \forall u \in s(tu \in r)\}$
- 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"
- Changes the schema 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.)

- It chooses those tuples from $r(R-S)$ such that when its Cartesian product is taken with $s(S)$, *all* the resulting tuples are in $r(R)$
- $q = r \div s$ is the largest relation satisfying $q \times s \subseteq r$
- 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.)

- It chooses those tuples from $r(R - S)$ such that when its Cartesian product is taken with $s(S)$, *all* the resulting tuples are in $r(R)$
- $q = r \div s$ is the largest relation satisfying $q \times s \subseteq r$
- 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.)

- It chooses those tuples from $r(R - S)$ such that when its Cartesian product is taken with $s(S)$, *all* the resulting tuples are in $r(R)$
- $q = r \div s$ is the largest relation satisfying $q \times s \subseteq r$
- 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
- Does not change the schema
- Example
  - $s \leftarrow \Pi_{cname}(\sigma_{bname=\text{''}IIT\text{''}}(\sigma_{borrower.lno=loan.lno}(borrower \times loan)))$
  - $q \leftarrow \Pi_{cname}(depositor)$
  - $r \leftarrow s - q$

Banking example

- branch(bname, bcity, assets)
- customer(cname, cstreet, ccity)
- account(ano, bname, bal)
- loan(lno, bname, amt)
- depositor(cname, ano)
- borrower(cname, lno)

Example queries

- Find all customers having both a loan and an account

Example queries

- Find all customers having both a loan and an account
  - $\Pi_{cname}(borrower) \cap \Pi_{cname}(depositor)$
- Find names of all customers having a loan and the corresponding loan amounts

Example queries

- Find all customers having both a loan and an account
  - $\Pi_{cname}(borrower) \cap \Pi_{cname}(depositor)$
- Find names of all customers having a loan and the corresponding loan amounts
  - $\Pi_{cname,lno,amt}(borrower \bowtie loan)$
- Find all customers who have an account at all branches in “Kanpur”

Example queries

- Find all customers having both a loan and an account
  - $\Pi_{cname}(borrower) \cap \Pi_{cname}(depositor)$
- Find names of all customers having a loan and the corresponding loan amounts
  - $\Pi_{cname,lno,amt}(borrower \bowtie loan)$
- Find all customers who have an account at all branches in "Kanpur"
  - $\Pi_{cname,bname}(depositor \bowtie account) \div \Pi_{bname}(\sigma_{bcity=''Kanpur''}(branch))$

Extended relational algebra

- The power of relational algebra can be enhanced by
  1. Generalized projection
  2. 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 $\Pi_{B-A,C}$ on r

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

returns

B-A	C
0	5
1	5
2	8

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

sum(C)
23

Aggregate operations (contd.)

- 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

returns

A	sum(C)
1	10
2	13

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

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 (contd.)

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

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

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

Null values (contd.)

- 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

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

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

Updating

- 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

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
- Updating 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
- Updating 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