3B. Relational Algebra

Slides adapted from Pearson Ed.

Recall from week 1:

- Relational database management systems (RDBMS).
  - **Formalized** by relational algebra/calculus.
  - **Standardized** by SQL (Structured Query Language) standard.
  - **Implemented** by various off-the-shelf products (MySQL, Oracle, MariaDB, SQLite, PostgreSQL Microsoft SQL Server, IBM Db2, etc.).

Basic mathematical definitions

- **Set**: an *unordered* collection of data, without repeats.
  - Denoted using curly brackets {}.
- **Tuple**: an *ordered* list of data (2-tuple = ordered pair; 3-tuple = triple; etc.).
  - Denoted using brackets ().
- **Cartesian product**: the set of all possible combinations of the elements of the sets involved in the product.
  - Cartesian product of two sets (A x B) will be a set containing ordered pairs like (a, b).

Relations in mathematics

- A mathematical relation describes a connection between elements of two sets (for a binary relation).
  - If a pair (a, b) is in the relation, it means element *a* is related to *b.*
  - If a pair (c, d) is missing from the relation, it means *c* and *d* don’t have this particular relationship.
- For an *n*-ary relationship: Given sets $A_1, A_2, \ldots, A_n$, a **relation** is a particular subset of the set $A_1 \times A_2 \times \ldots \times A_n$.
  - **Domain**: the set each member of the tuple is drawn from (A_1, A_2, etc.).

In relational databases, all tables are a mathematical relation

- Attributes = columns = domains that data is drawn from.
  - For Staff, the columns are staffNo (set of all possible staff numbers), fName (set of all possible strings maybe with an upper character limit), etc.
- Records = rows = tuples that are *included* in the relation.
  - For Staff, a tuple in the relation might be (“SG37”, “John”, “Smith”, …).
- Table = relation = a set of tuples.
- All parts of a relational database are in a simple 2-D form (relation) that can be mathematically manipulated and analyzed.

Relational algebra

- Can perform transformations on relations to produce new relations.
- Has the property of **closure**: applying an operation to a type (relation) yields the same type (relation).
  - Allows multiple operations to be strung together.

Five basic operations

- Selection,
- Projection,
- Union,
- Set difference,
- Cartesian product.

- Plus 3 derived operations: Intersection, Join (several types), and Division.

Selection (or Restriction) - σ

- $\sigma_{predicate}(R)$
- Row selection:
  - Given a relation R and a predicate, produces a new relation R' consisting only of tuples satisfying the predicate.
  - Ex: $\sigma_{salary > 10000}(Staff)$
- Like the WHERE clause in SQL.

Projection – Π

- $\Pi_{a_1,\ldots,a_n}(R)$
- Column selection:
  - Given a relation R and a list of attributes, produces a new relation R’ that has all of R’s tuples but with attributes other than the ones listed masked out
  - Ex: $\Pi_{\text{staffNo, fName, lName, salary}}$(Staff) → produces a four-column table (a 4-tuple relation).
- Like the SELECT [list] clause in SQL.

Union – ∪

- $R \cup S$
- Table combination; vertical join:
  - Combine (without duplicates) the tuples of two relations R and S that have identical **schemas** (list of (attribute name, domain) pairs)
  - Ex: List all cities where there is either a branch office or a property for rent:
    - Π_{city}(Branch) ∪ Π_{city}(PropertyForRent)
- Same as UNION function in SQL.

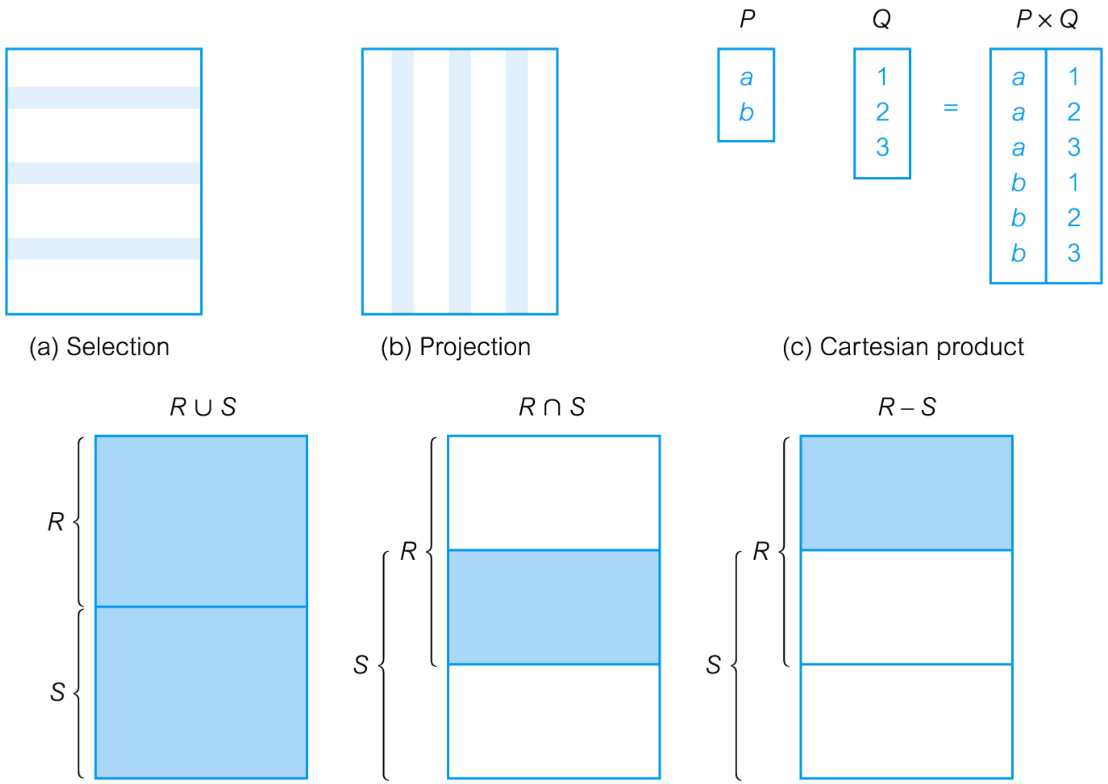
Set difference – -

- R – S (or R \ S)
- Removal of elements:
  - Remove from R all tuples identical to the ones that exist in S, producing a new relation R'.
  - Like union, must be union compatible (have same schemas).
  - Ex: List all cities where there is a branch office but no properties for rent.
    - $\Pi_{city}(Branch) - \Pi_{city}(PropertyForRent)$
- Same as EXCEPT function in SQL.

Cartesian product – x

- $R \times S$
- Horizontal join; pairwise concatenation of tuples.
  - Combine every tuple in R with every tuple in S to get a table with |R|x|S| tuples with numcols(R) + numcols(S) columns.
- Same as listing two or more tables after FROM clause in SQL.

Visualizing basic operations



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

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

Derived operations

- Intersection - ∩ $R \cap S$
  - Equivalent to R – (R – S).
- Join: frequently-used operation that consist of a Cartesian product followed by a selection (selection differs by type of join).
- Division - ÷ $R \div S$
  - Sub-tuples of R that span all the values in S.

Theta join (θ-join) - ⋈

- $R \bowtie_{predicate} S$
  - Equivalent to $\sigma_{predicate}(R \times S)$
- Predicate involves a comparison (<, ≤, >, ≥, =, ≠).
  - Ones involving ‘=‘ are called equijoins.
- Same as using the FROM… JOIN… ON clause in SQL.
  - Flexible and all-purpose.

Natural join - ⋈

- $R \bowtie S$
- An equijoin that checks for equality on all attributes with the same name and removes the duplicate column(s) in the final table.
  - Equivalent to $\Pi_{reduced\ list}(\sigma_{R.c1=S.c1,etc.}(R \times S))$
- Similar to FROM… JOIN… USING clause in SQL.

Outer joins

- By default, joins are *inner* joins, meaning they only contain entries from the tables where the condition was fulfilled.
- Can also specify *outer* joins, which retain rows/tuples in one or both tables/relations even if they didn't match with anything.
  - Missing attributes that would have been filled by the match(es) in the other table are filled with nulls.
  - Left join / left outer join preserves all rows from the first table.
  - Right join / right outer join preserves all rows from the second table.
  - Full outer join does both.

SQL: Outer join example

- Task: List branches and properties in the same city, along with unmatched [branches | properties | branches or properties].

  - SELECT b.*, p.*
    FROM Branch b [LEFT | RIGHT | FULL] JOIN Property p
    ON b.bCity = p.pCity;

Semijoin - ▷

- R ▷$_{\text{predicate}}$ S
- Same as θ-join except the final result only contains R's attributes.
  - Equivalent to $\Pi_{R's\ attr\ list}(\sigma_{predicate}(R \times S))$
  - Use S to filter R's rows (tuples).

Division - ÷

- $R \div S \ (= T)$
- Selects (a part of) tuples in R that match every possible value in S.
  - Attributes in result relation T is attr(R) – attr(S).
  - S x T will not generate any tuples that weren't already in R.
  - Equivalent to $\Pi_{attr(T)}(R) - \Pi_{attr(T)}\left(\left(\Pi_{attr(T)}(R) \times S\right) - R\right)$

Start with all tuples in R

Remove the ones that don't fully cover S

After this op, only rows that don't fully cover S will remain

Division - ÷

- $R \div S \; (= T)$
- Selects (a part of) tuples in R that match every possible value in S.
  - Attributes in result relation T is attr(R) – attr(S).
  - S x T will not generate any tuples that weren't already in R.
  - Equivalent to $\Pi_{attr(T)}(R) - \Pi_{attr(T)}\left(\left(\Pi_{attr(T)}(R) \times S\right) - R\right)$
  - Alternative using assignment:

$$T_1 \leftarrow \Pi_{attr(T)}(R)$$
$$T_2 \leftarrow \Pi_{attr(T)}(T_1 \times S) - R$$
$$T \leftarrow T_1 \; - T_2$$

  - S ⊇ R

Division example

- Identify all clients who have viewed all properties with three rooms.
  - $(\Pi_{clientNo,\ propertyNo}(Viewing)) \div (\Pi_{propertyNo}(\sigma_{rooms\ =\ 3}\ (PropertyForRent)))$

$\Pi_{clientNo,propertyNo}(Viewing)$

clientNo	propertyNo
CR56	PA14
CR76	PG4
CR56	PG4
CR62	PA14
CR56	PG36

$\Pi_{propertyNo}(\sigma_{rooms=3}(PropertyForRent))$

propertyNo
PG4
PG36

RESULT

clientNo
CR56

Visualizing derived operations

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

$T \triangleright_B U$

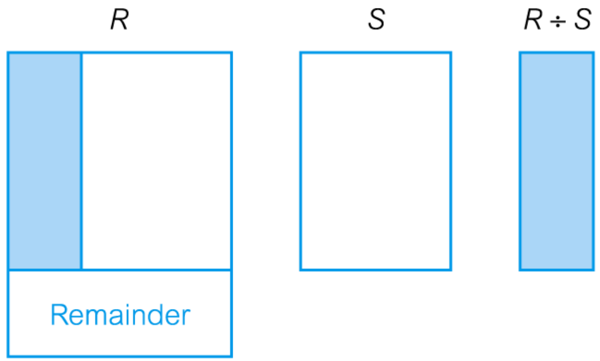
A	B
a	1

(h) Semijoin

$T ⟕_C U$

A	B	C
a	1	x
a	1	y
b	2	

(i) Left Outer join



(j) Divis on (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

Proposed operations

- Aggregation ($\Im_{\text{list of <aggfunc, attr> pairs}}(R)$)
  - Aggregation functions: COUNT, SUM, AVG, MIN, and MAX as in SQL.
- Grouping ($_{\text{grouping attr(s)}}\Im_{\text{list of <aggfunc, attr> pairs}}(R)$)
  - Forms groups based on the list on the left and calculates aggregate function on all groups.
- Produces a scalar, 1-D, or 2-D table (i.e. another relation).
  - Give new relation better name using rename function (ρ).
  - Example: $\rho_R(\text{myCount})\ \Im_{\text{COUNT propertyNo}}(\sigma_{\text{rent} > 350}(\text{PropertyForRent}))$.