# Contents

**Relational Algebra** **1**

**Operations** **2**

Selection . . . . . 2

Selection in SQL . . . . . 5

Projection . . . . . 5

Projection in SQL . . . . . 6

Duplicate Elimination . . . . . 6

Combining Select and Project . . . . . 6

Nesting Operations . . . . . 6

Immediate Relations . . . . . 7

Combining Select and Project . . . . . 7

**Set Operators** **7**

Union . . . . . 8

Intersection . . . . . 8

Union and Intersection . . . . . 8

Difference . . . . . 8

Set Operators in SQL . . . . . 8

Join . . . . . 9

Joins in SQL . . . . . 9

# Relational Algebra

- A data model must include a set of operations to manipulate the database
  - Relational Algebra is the basic set of operations for the Relational Model
- Relational Algebra is important, as it:
  - Provides a formal foundation for relational model operations
  - Is used as a basis for implementing and optimising queries
  - Is incorporated into the SQL language for RDBMS

- Relational Algebra is a collection of operations on relations which fall into two groups
  - Set operations from mathematical set theory
    * Union, intersection, etc.
    * Applicable as each relation is defined to be a set of tuples in the relational model
  - Relational operations
    * Defined specifically for relational databases
    * Selection, projection, join, etc.

# Operations

- Set operations
  - Standard mathematical operations on sets
  - UNION, INTERSECTION, SET DIFFERENCE
- Relational Database Operations
  - Unary Operations
    * Operate on a single relation
    * SELECT and PROJECT
  - Binary Operations
    * A JOIN is used to combine related tuples across two relations using a *join condition*

## Selection

- The SELECT operation is used to identify the subset of tuples from a relation that satisfy a selection condtion
  - Acts as a filter on a relation
  - Horizonal partition of a relation
- Formal Notation
- $\sigma_{(selection\ condition)}$ (R)
- So to identify all the employeers in department 4

**EMPLOYEE**

| Fname | Minit | Lname | *Ssn* | Dno |
|---|---|---|---|---|
| John | B | Smith | 123456789 | 5 |

| Fname | Minit | Lname | *Ssn* | Dno |
|---|---|---|---|---|
| Franklin | T | Wong | 333445555 | 5 |
| Alicia | J | Zelaya | 999887777 | 4 |
| Jennier | S | Wallace | Z987654321 | 4 |

- $\sigma_{(Dno=4)}$ (EMPLOYEE)
- The result of SELECT operation is a new relation
  - Made up of those tuples that satisfied the selection condition

| Fname | Minit | Lname | *Ssn* | Dno |
|---|---|---|---|---|
| Alicia | J | Zelaya | 999887777 | 4 |
| Jennier | S | Wallace | Z987654321 | 4 |

- This new relation has the same attributes as the relation `R`, upon which the selection was performed
- Boolean Expression specified in the (selection condition) is made up of a number of clauses:
  - `<attribute name><comparison op><constant value>`
- `<attribute name>`
  - name of an attribute of R
- `<comparison op>`
  - operator such as $= < \leq > \geq \neq$
- `<constant value>`
  - a constant value from the attribute domain
- Selection conditions can be joined by boolean operators *and*, *or* and *not*
  - (cond1 AND cond2) is TRUE if both (cond1) and (cond2) are TRUE; otherwise, it is FALSE
  - (cond1 OR cond2) is TRUE if either (cond1) or (cond2) or both are TRUE; otherwise, it is FALSE
  - (NOT cond) is TRUE if cond is FALSE; otherwise, it is FALSE
  - Example
    * Select all employees who either work in department 4 and make over €25,000 a year or work in department 5 and make over €30,000 a year

    * $\delta_{(Dno=4 \wedge Salary>25000)}$ or $\delta_{(Dno=5 \wedge Salary>30000)}$(EMPLOYEE)
- Rather than using a constant value to evaluate the selection condition, another attribute can be used
  -
  - Example
    * Select all employees who earn more than the lower tax band
    * $\delta_{(Salary>lower_tax_band)}$(EMPLOYEE)
- Full set of comparison operators ($= < \leq > \geq \neq$) can be used on all *Ordered Domains*
  - Numeric, Currency, Dates
- Can also have *Unordered Domains*
  - i.e. Color = {Red, Blue, Green, Yellow, …}
  - Only valid comparison operators
    * {=, $\neq$}
    * Exceptions - substring
- The Select operator is Unary
  - Applied to a single relation
  - Select is applied to each tuple in turn
- Degree
  - The degree is the number of attributes in the relation
  - The degree of the Select operator is the same as the degree of the relation R
- The number of tuples returned for a Select operation, is always less than or equal to the number of tuples in R
  - $\mid \delta_c(R) \mid \leq \mid R \mid$
- The fraction of tuples in a relation selected by a condition is known as the *selectivity* of that condition
- The Select operation is *commutative*
  - $\delta_(condition1)$ ($\delta_{condition2}$(R))
  - $\delta_(condition2)$ ($\delta_{condition1}$(R))
- A "*cascade*" or sequence of select operations can be combined into a singple operation
  - $\delta_(condition1)$ ($\delta_{condition2}$(…(- $\delta_(conditionN)$)…))
  - $\delta_(condition1) \wedge (condition2) \wedge \cdots \wedge (conditionN)$(R)

**Selection in SQL**

- SELECT is a common command in SQL, directly based upon this Relational Algebra approach
- While the algebraic notation is:
  - $\delta_{Dno=4 \wedge Salary>25000}$(EMPLOYEE)
- The SQL syntax is:

```
SELECT * from EMPLOYEE
WHERE Dno=4
AND Salary > 25,000;
```

## Projection

- The Project operation selects certain attributes from the table, while discarding the others
  - Vertical partition of relation
- Formal Notation
  - $\pi_{attribute\ list}$(R)
- Example
  - To list all employees first and last names and their salary, we can use Project:
  - $\pi_{Fname,Lname,Salary}$(EMPLOYEE)
- The relation that is the result of the Project operation has only the attributes specified in the attribute list
  - Hence, the *degree* of the resulting relation is equal to the number of attributes in this list
- Dupicate Elimination
  - It is possible for a Project operation to specify a set of non-key attributes, i.e. forename and surname
  - In this instance, duplicate tuples are likely to occur
  - However, the result of a Project operation has to be a valid relation, and only contain distinct tuples
  - The operation removes (or merges) the duplicate tuples and only returns distinct tuples

### Projection in SQL

- The SELECT statement in SQL also handles the Project operation
- While the algebraic notation is
  - $\pi_{Fname,Lname,Salary}$(EMPLOYEE)
- The SQL syntax is:

```
SELECT Fname, Lname, Salary
from EMPLOYEE;
```

### Duplicate Elimination

- SQL does not automatically implement the duplicate elimination process of the Project operation
  - This is a departure from formal Relational Algebra
- To implement this, you need to use the DISTINCT keyword

```
SELECT DISTINCT firstname, surname, salary
from EMPLOYEE;
```

## Combining Select and Project

- Complex queries are often need to use combinations of Select and Project operations
- This can be achieved in two ways
  - Nesting of operations in a single relational algebra expression
  - One operation at a time using intermediate result relations

## Nesting Operations

- If we want a list of the names and salaries of all employees who earn more than €25,000 and work in the administration department
- This would require two operations
  - $\delta_{Dno=4 \wedge Salary>25000}$(EMPLOYEE)
  - $\pi_{Fname,Lname,Salary}$(EMPLOYEE)
- However, we can achieve the same result in a single relational algebra expression by nesting
  - $\pi_{Fname,Lname,Salary}(\delta_{Dno=4 \wedge Salary>25000}$(EMPLOYEE))

### Immediate Relations

- Operations can be applied one at a time using intermediate relations to store the results
- All intermediate tables must be named so that they can be referred to
  - $EMPS \leftarrow \delta_{(Dno=4 \wedge Salary>25000)}$(EMPLOYEE)
  - $RESULTS \leftarrow \pi_{Fname,Lname,Salary}$(EMPS)
- It can sometimes be simpler to break complex queries into a series in this manner

### Combining Select and Project

- The Select and Project operations can be combined in the SQL SELECT statement
- Consider the following operations
  - $\delta_{Dno=4 \wedge Salary>25000}$(EMPLOYEE)
  - $\pi_{Fname,Lname,Salary}$(EMPLOYEE)
- These can be combined in the following SQL

```
SELECT Fname, Lname, Salary
from EMPLOYEE
WHERE Dno=4
AND Salary>25,000
```

## Set Operators

- Standard mathematical operations used to merge the elements of two Sets
  - Union
  - Intersection
  - Set Difference
- Binary Operations
  - Each operation is applied to two sets
- Two use these operations, two relations must be "union compatible"
  - They must be of the same number of attributes
  - Each corresonding pair of attributes has the same domain
- Two relations $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots, B_n)$ are said to be union compatible if
  - They have the same degree `n`
  - $dom(A_i) = dom(B_i)$ for $1 \leq i \leq n$

## Union

- Denoted $R \cup S$
- The result of a *Union* operation is a new relation which contains all tuples that are either in `R` or `S` or in both `R` and `S`
- Duplicate tuples are discarded
- Commutative operation
  - $R \cup S = S \cup R$

## Intersection

- Denotes $R \cap S$
- The result of an *Intersection* operation is a new relation which contains all tuples that are in both `R` and `S`
- Commutative operation
  - $R \cap S = S \cap R$

## Union and Intersection

- Can both be treated as `n-ary` operations
  - Applicable to any number of relations
  - As they are *associative operations*
  - $R \cup (S \cup T) = (R \cup S) \cup T$
  - $R \cap (S \cap T) = (R \cap S) \cap T$

## Difference

- Denoted by `R - S`
- The result of a *Set Difference* (or *Minus*) operation is a new relation that includes all tuples that are in `R` but not in `S`
- Not commutative
  - $R - S \neq S - R$

### Set Operators in SQL

- There are three SQL commands which correspond to these Set Operations
  - `UNION` - Union
  - `INTERSECT` - Intersection
  - `EXCEPT` - Set Difference

- Additionally, there are multi-set operators in SQL tha do not eliminate duplicates
  - `UNION ALL`
  - `INTERSECT ALL`
  - `EXCEPT ALL`

## Join

- Denoted by $R \bowtie_{join\ condition} S$
- Used to combine related tuples from two relations, into a single tuple
  - Important as it allows for the processing of relationships between relations
- Makes use of Foreign Keys and Referential Integrity constraints that have been defined
- Consider two relations $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots, B_m)$
- The result of $R \bowtie S$ is a new relation `Q` with
  - `n+m` attributes
  - $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$
  - A tuple for each combination of tuples (one from `R` and one from `S`) that satisfy the join condition
- The join condition is specified on attributes from both relations, `R` and `S`
  - It is evaluated for every combination of tuples from the two relations
  - Each tuple combination for which the join condition is TRUE is included in the result relation is a single, combined tuple
- Suppose we want to return the name of the manager of each department
  - DEPARTMENT$\bowtie_{Mgr_ssn=Ssn}$EMPLOYEE
- Can then be combined with a Project operation to only return the required attributes
  - $\pi_{Dname,Fname,Lname}$(DEPARTMENT$\bowtie_{Mgr_ssn=Ssn}$EMPLOYEE)

### Joins in SQL

```
SELECT employee.name, job, department.name
FROM employee, department
WHERE employee.deptno = department.deptno;
```

or

```
SELECT employee.name, job, department.name
FROM employee
INNER JOIN department
ON employee.deptno = department.deptno
```