# COMP 5143
# Advanced
# Database Management System

*Fall 2015*

Computer Science Department
Prairie View A&M University

Mary Heejin Kim, Ph.D.

# Chapter 5

## Relational Algebra and
## Relational Calculus

# Chapter 5 - Objectives

- **Meaning of the term relational completeness.**

- **How to form queries in relational algebra.**

- **How to form queries in tuple relational calculus.**

- **How to form queries in domain relational calculus.**

- **Categories of relational DML.**

# Introduction

- **Relational algebra and relational calculus are formal languages associated with the relational model.**
- **Informally, relational algebra is a (high-level) procedural language and relational calculus a non-procedural language.**
- **However, formally both are equivalent to one another.**
- **A language that produces a relation that can be derived using relational calculus is <u>relationally complete</u>.**
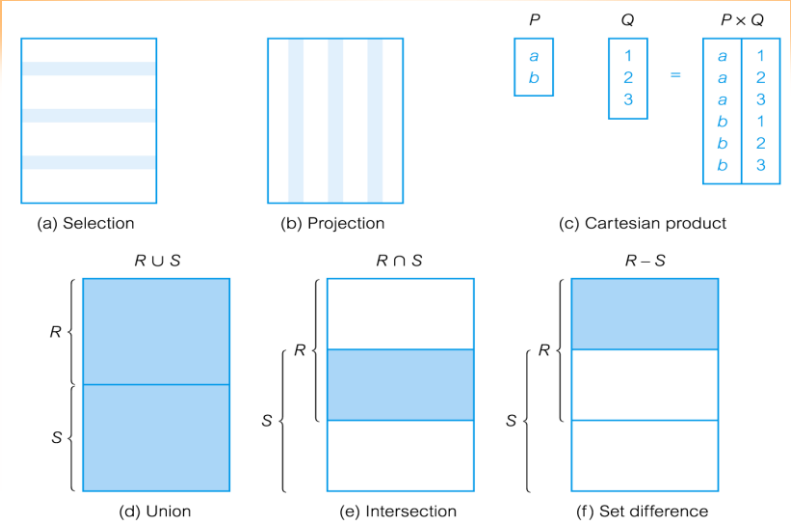
# Relational Algebra

- **Relational algebra operations work on one or more relations to define another relation without changing the original relations.**
- **Both operands and results are relations, so output from one operation can become input to another operation.**
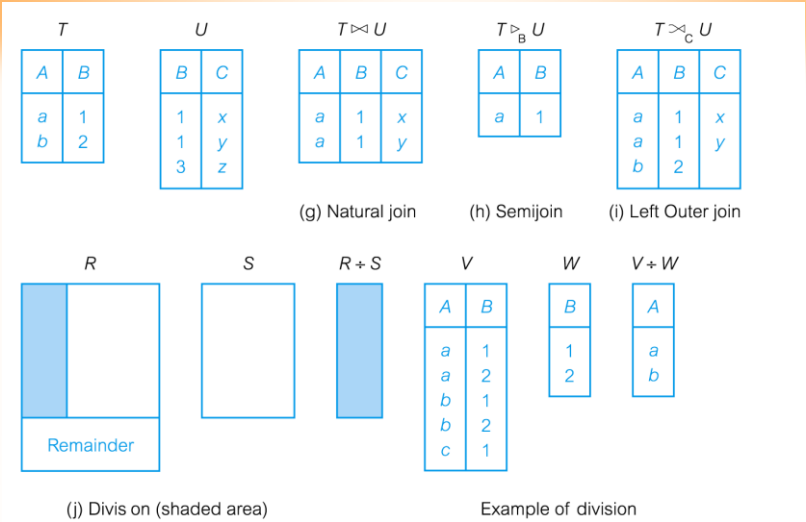- **Allows expressions to be nested, just as in arithmetic. This property is called <u>closure</u>.**

# Relational Algebra

- **Five basic operations in relational algebra: Selection, Projection, Cartesian product, Union, and Set Difference.**

- **These perform most of the data retrieval operations needed.**

- **Also have Join, Intersection, and Division operations, which can be expressed in terms of 5 basic operations.**

# Relational Algebra Operations



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

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

# Relational Algebra Operations



(g) Natural join     (h) Semijoin     (i) Left Outer join

(j) Divis on (shaded area)     Example of division

# Selection (or Restriction)

- $\sigma_{predicate}$ **(R)**
  - **Works on a single relation R and defines a relation that contains only those tuples (rows) of R that satisfy the specified condition (*predicate*).**

# Example - Selection (or Restriction)

- **List all staff with a salary greater than £10,000.**

  $\sigma_{salary > 10000}$ **(Staff)**

| staffNo | fName | lName | position | sex | DOB | salary | branchNo |
|---------|-------|-------|----------|-----|-----|--------|----------|
| SL21 | John | White | Manager | M | 1-Oct-45 | 30000 | B005 |
| SG37 | Ann | Beech | Assistant | F | 10-Nov-60 | 12000 | B003 |
| SG14 | David | Ford | Supervisor | M | 24- Mar-58 | 18000 | B003 |
| SG5 | Susan | Brand | Manager | F | 3-Jun-40 | 24000 | B003 |

# Projection

- $\Pi_{col1,\ldots,coln}(R)$
  - Works on a single relation R and defines a relation that contains a vertical subset of R, extracting the values of specified attributes and eliminating duplicates.

# Example - Projection

- **Produce a list of salaries for all staff, showing only staffNo, fName, lName, and salary details.**

$\Pi_{staffNo, fName, lName, salary}(Staff)$

| staffNo | fName | lName | salary |
|---------|-------|-------|--------|
| SL21 | John | White | 30000 |
| SG37 | Ann | Beech | 12000 |
| SG14 | David | Ford | 18000 |
| SA9 | Mary | Howe | 9000 |
| SG5 | Susan | Brand | 24000 |
| SL41 | Julie | Lee | 9000 |

# Union

- **R ∪ S**

  –**Union of two relations R and S defines a relation that contains all the tuples of R, or S, or both R and S, duplicate tuples being eliminated.**

  –**R and S must be union-compatible.**

- **If R and S have *I* and *J* tuples, respectively, union is obtained by concatenating them into one relation with a maximum of (*I* + *J*) tuples.**

# Example - Union

- **List all cities where there is either a branch office or a property for rent.**

  $\Pi_{city}(\text{Branch}) \cup \Pi_{city}(\text{PropertyForRent})$

| city |
| --- |
| London |
| Aberdeen |
| Glasgow |
| Bristol |

# Set Difference

- **R – S**
  - Defines a relation consisting of the tuples that are in relation R, but not in S.
  - R and S must be union-compatible.

# Example - Set Difference

- **List all cities where there is a branch office but no properties for rent.**

$$\Pi_{city}(\text{Branch}) - \Pi_{city}(\text{PropertyForRent})$$

| city |
| --- |
| Bristol |

# Intersection

- **$R \cap S$**
  - Defines a relation consisting of the set of all tuples that are in both R and S.
  - R and S must be union-compatible.

- **Expressed using basic operations:**
  **$R \cap S = R - (R - S)$**

# Example - Intersection

- **List all cities where there is both a branch office and at least one property for rent.**

  $\Pi_{city}(Branch) \cap \Pi_{city}(PropertyForRent)$

| city |
|------|
| Aberdeen |
| London |
| Glasgow |

# Cartesian product

- **R X S**
  - **Defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S.**

# Example - Cartesian product

- **List the names and comments of all clients who have viewed a property for rent.**

$$(\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \; X \; (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing}))$$

| client.clientNo | fName | lName | Viewing.clientNo | propertyNo | comment |
|---|---|---|---|---|---|
| CR76 | John | Kay | CR56 | PA14 | too small |
| CR76 | John | Kay | CR76 | PG4 | too remote |
| CR76 | John | Kay | CR56 | PG4 | |
| CR76 | John | Kay | CR62 | PA14 | no dining room |
| CR76 | John | Kay | CR56 | PG36 | |
| CR56 | Aline | Stewart | CR56 | PA14 | too small |
| CR56 | Aline | Stewart | CR76 | PG4 | too remote |
| CR56 | Aline | Stewart | CR56 | PG4 | |
| CR56 | Aline | Stewart | CR62 | PA14 | no dining room |
| CR56 | Aline | Stewart | CR56 | PG36 | |
| CR74 | Mike | Ritchie | CR56 | PA14 | too small |
| CR74 | Mike | Ritchie | CR76 | PG4 | too remote |
| CR74 | Mike | Ritchie | CR56 | PG4 | |
| CR74 | Mike | Ritchie | CR62 | PA14 | no dining room |
| CR74 | Mike | Ritchie | CR56 | PG36 | |
| CR62 | Mary | Tregear | CR56 | PA14 | too small |
| CR62 | Mary | Tregear | CR76 | PG4 | too remote |
| CR62 | Mary | Tregear | CR56 | PG4 | |
| CR62 | Mary | Tregear | CR62 | PA14 | no dining room |
| CR62 | Mary | Tregear | CR56 | PG36 | |

# Example - Cartesian product and Selection

- Use selection operation to extract those tuples where Client.clientNo = Viewing.clientNo.

$$\sigma_{\text{Client.clientNo = Viewing.clientNo}}((\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \text{ X } (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing})))$$

| client.clientNo | fName | lName | Viewing.clientNo | propertyNo | comment |
|---|---|---|---|---|---|
| CR76 | John | Kay | CR76 | PG4 | too remote |
| CR56 | Aline | Stewart | CR56 | PA14 | too small |
| CR56 | Aline | Stewart | CR56 | PG4 | |
| CR56 | Aline | Stewart | CR56 | PG36 | |
| CR62 | Mary | Tregear | CR62 | PA14 | no dining room |

- Cartesian product and Selection can be reduced to a single operation called a *Join*.

# Join Operations

- **Join is a derivative of Cartesian product.**
- **Equivalent to performing a Selection, using join predicate as selection formula, over Cartesian product of the two operand relations.**
- **One of the most difficult operations to implement efficiently in an RDBMS and one reason why RDBMSs have intrinsic performance problems.**

# Join Operations

- **Various forms of join operation**
  - **Theta join**
  - **Equijoin (a particular type of Theta join)**
  - **Natural join**
  - **Outer join**
  - **Semijoin**

# Theta join ($\theta$-join)

- **R $\bowtie_F$ S**
  - **Defines a relation that contains tuples satisfying the predicate F from the Cartesian product of R and S.**
  - **The predicate F is of the form $R.a_i \; \theta \; S.b_i$ where $\theta$ may be one of the comparison operators $(<, \leq, >, \geq, =, \neq).$**

# Theta join (θ-join)

- **Can rewrite Theta join using basic Selection and Cartesian product operations.**

$$R \bowtie_F S = \sigma_F(R \times S)$$

- **Degree of a Theta join is sum of degrees of the operand relations R and S. If predicate F contains only equality (=), the term *Equijoin* is used.**

# Example - Equijoin

- **List the names and comments of all clients who have viewed a property for rent.**

$(\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \bowtie_{\text{Client.clientNo =}}$
$_{\text{Viewing.clientNo}} (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing}))$

| client.clientNo | fName | lName | Viewing.clientNo | propertyNo | comment |
|---|---|---|---|---|---|
| CR76 | John | Kay | CR76 | PG4 | too remote |
| CR56 | Aline | Stewart | CR56 | PA14 | too small |
| CR56 | Aline | Stewart | CR56 | PG4 | |
| CR56 | Aline | Stewart | CR56 | PG36 | |
| CR62 | Mary | Tregear | CR62 | PA14 | no dining room |

# Natural join

- **R ⋈ S**

  − **An Equijoin of the two relations R and S over all common attributes $x$. One occurrence of each common attribute is eliminated from the result.**

# Example - Natural join

- **List the names and comments of all clients who have viewed a property for rent.**

  $(\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \bowtie$
  $(\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing}))$

| clientNo | fName | lName | propertyNo | comment |
|----------|-------|---------|------------|----------------|
| CR76 | John | Kay | PG4 | too remote |
| CR56 | Aline | Stewart | PA14 | too small |
| CR56 | Aline | Stewart | PG4 | |
| CR56 | Aline | Stewart | PG36 | |
| CR62 | Mary | Tregear | PA14 | no dining room |

# Outer join

- **To display rows in the result that do not have matching values in the join column, use Outer join.**

- **R $⋈$ S**
  **−(Left) outer join is join in which tuples from R that do not have matching values in common columns of S are also included in result relation.**

# Example - Left Outer join

- **Produce a status report on property viewings.**

  $\Pi_{propertyNo, street, city}(\textbf{PropertyForRent}) ⋈$
  **Viewing**

| propertyNo | street | city | clientNo | viewDate | comment |
|---|---|---|---|---|---|
| PA14 | 16 Holhead | Aberdeen | CR56 | 24-May-01 | too small |
| PA14 | 16 Holhead | Aberdeen | CR62 | 14-May-01 | no dining room |
| PL94 | 6 Argyll St | London | null | null | null |
| PG4 | 6 Lawrence St | Glasgow | CR76 | 20-Apr-01 | too remote |
| PG4 | 6 Lawrence St | Glasgow | CR56 | 26-May-01 | |
| PG36 | 2 Manor Rd | Glasgow | CR56 | 28-Apr-01 | |
| PG21 | 18 Dale Rd | Glasgow | null | null | null |
| PG16 | 5 Novar Dr | Glasgow | null | null | null |

# Semijoin

- $R \rhd_F S$

    – **Defines a relation that contains the tuples of R that participate in the join of R with S.**

- **Can rewrite Semijoin using Projection and Join:**

    $R \rhd_F S \quad = \Pi_A(R \bowtie_F S)$

# Example - Semijoin

- **List complete details of all staff who work at the branch in Glasgow.**

    $\text{Staff} \rhd_{\text{Staff.branchNo=Branch.branchNo}}(\sigma_{\text{city='Glasgow'}}(\text{Branch}))$

| staffNo | fName | lName | position | sex | DOB | salary | branchNo |
|---------|-------|-------|----------|-----|-----|--------|----------|
| SG37 | Ann | Beech | Assistant | F | 10-Nov-60 | 12000 | B003 |
| SG14 | David | Ford | Supervisor | M | 24- Mar-58 | 18000 | B003 |
| SG5 | Susan | Brand | Manager | F | 3-Jun-40 | 24000 | B003 |

# Division

- $R \div S$
  - Defines a relation over the attributes C that consists of set of tuples from R that match combination of *every* tuple in S.

- **Expressed using basic operations:**

  $T_1 \leftarrow \Pi_C(R)$

  $T_2 \leftarrow \Pi_C((S \times T_1) - R)$

  $T \leftarrow T_1 - T_2$

# Example - Division

- **Identify all clients who have viewed all properties with three rooms.**

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

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

$\Pi_{clientNo,propertyNo}(\text{Viewing})$    $\Pi_{propertyNo}(\sigma_{rooms=3}(\text{PropertyForRent}))$    RESULT

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

| propertyNo |
|------------|
| PG4 |
| PG36 |

| clientNo |
|----------|
| CR56 |

# Aggregate Operations

- $\mathfrak{I}_{AL}(R)$

  – Applies aggregate function list, AL, to R to define a relation over the aggregate list.

  – AL contains one or more (<aggregate_function>, <attribute>) pairs .

- **Main aggregate functions are: COUNT, SUM, AVG, MIN, and MAX.**

# Example – Aggregate Operations

- **How many properties cost more than £350 per month to rent?**

  $\rho_R(myCount)\ \mathfrak{I}_{COUNT\ propertyNo}\ (\sigma_{rent > 350}\ (PropertyForRent))$

| myCount |
|---------|
| 5 |

(a)

# Grouping Operation

- $_{GA}\Im_{AL}(R)$

  – Groups tuples of R by grouping attributes, GA, and then applies aggregate function list, AL, to define a new relation.

  – AL contains one or more (<aggregate_function>, <attribute>) pairs.

  – Resulting relation contains the grouping attributes, GA, along with results of each of the aggregate functions.

# Example – Grouping Operation

- **Find the number of staff working in each branch and the sum of their salaries.**

  $\rho_R(branchNo, myCount, mySum)$

  $_{branchNo}\Im_{COUNT\ staffNo,\ SUM\ salary}(Staff)$

| branchNo | myCount | mySum |
|----------|---------|-------|
| B003     | 3       | 54000 |
| B005     | 2       | 39000 |
| B007     | 1       | 9000  |

# Relational Calculus

- **Relational calculus query specifies *what* is to be retrieved rather than *how* to retrieve it.**
  **–No description of how to evaluate a query.**

- **In first-order logic (or predicate calculus), *predicate* is a truth-valued function with arguments.**

- **When we substitute values for the arguments, function yields an expression, called a *proposition*, which can be either true or false.**

# Relational Calculus

- **If predicate contains a variable (e.g. '*x* is a member of staff'), there must be a range for *x*.**

- **When we substitute some values of this range for *x*, proposition may be true; for other values, it may be false.**

- **When applied to databases, relational calculus has forms: *tuple* and *domain*.**

# Tuple Relational Calculus

- **Interested in finding tuples for which a predicate is true. Based on use of <u>tuple variables</u>.**

- **Tuple variable is a variable that 'ranges over' a named relation: i.e., variable whose only permitted values are tuples of the relation.**

- **Specify range of a tuple variable $S$ as the Staff relation as:**

  **Staff(S)**

- **To find set of all tuples S such that P(S) is true:**

  **{S | P(S)}**

# Tuple Relational Calculus - Example

- **To find details of all staff earning more than £10,000:**

  **{S | Staff(S) ∧ S.salary > 10000}**

- **To find a particular attribute, such as salary, write:**

  **{S.salary | Staff(S) ∧ S.salary > 10000}**

# Tuple Relational Calculus

- **Can use two *quantifiers* to tell how many instances the predicate applies to:**
  - **Existential quantifier ∃ ('there exists')**
  - **Universal quantifier ∀ ('for all')**

- **Tuple variables qualified by ∀ or ∃ are called *bound* variables, otherwise called *free* variables.**

# Tuple Relational Calculus

- **Existential quantifier used in formulae that must be true for at least one instance, such as:**

  **Staff(S) ∧ (∃B)(Branch(B) ∧ (B.branchNo = S.branchNo) ∧ B.city = 'London')**

- **Means 'There exists a Branch tuple with same branchNo as the branchNo of the current Staff tuple, *S*, and is located in London'.**

# Tuple Relational Calculus

- **Universal quantifier is used in statements about every instance, such as:**
  ($\forall$B) (B.city $\neq$ 'Paris')

- **Means 'For all Branch tuples, the address is not in Paris'.**

- **Can also use ~($\exists$B) (B.city = 'Paris') which means 'There are no branches with an address in Paris'.**

# Tuple Relational Calculus

- **Formulae should be unambiguous and make sense.**
- **A (well-formed) formula is made out of <u>atoms</u>:**
  - $R(S_i)$, where $S_i$ is a tuple variable and $R$ is a relation
  - $S_i.a_1 \; \theta \; S_j.a_2$
  - $S_i.a_1 \; \theta \; c$
- **Can recursively build up formulae from atoms:**
  - An atom is a formula
  - If $F_1$ and $F_2$ are formulae, so are their conjunction, $F_1 \wedge F_2$; disjunction, $F_1 \vee F_2$; and negation, $\sim F_1$
  - If $F$ is a formula with free variable $X$, then $(\exists X)(F)$ and $(\forall X)(F)$ are also formulae.

# Example - Tuple Relational Calculus

- **List the names of all managers who earn more than £25,000.**

  {S.fName, S.lName | Staff(S) ∧
  S.position = 'Manager' ∧ S.salary > 25000}

- **List the staff who manage properties for rent in Glasgow.**

  {S | Staff(S) ∧ (∃P) (PropertyForRent(P) ∧
  (P.staffNo = S.staffNo) ∧ P.city = 'Glasgow')}

# Example - Tuple Relational Calculus

- **List the names of staff who currently do not manage any properties.**

  {S.fName, S.lName | Staff(S) ∧ (~(∃P)
  (PropertyForRent(P)∧(S.staffNo = P.staffNo)))}

**Or**

  {S.fName, S.lName | Staff(S) ∧ ((∀P)
  (~PropertyForRent(P) ∨
  ~(S.staffNo = P.staffNo)))}

# Example - Tuple Relational Calculus

- **List the names of clients who have viewed a property for rent in Glasgow.**

  **{C.fName, C.lName | Client(C) $\wedge$ (($\exists$V)($\exists$P)**
  **(Viewing(V) $\wedge$ PropertyForRent(P) $\wedge$**
  **(C.clientNo = V.clientNo) $\wedge$**
  **(V.propertyNo=P.propertyNo) $\wedge$**
  **P.city ='Glasgow'))}**

# Tuple Relational Calculus

- **Expressions can generate an infinite set. For example:**
  **{S | ~Staff(S)}**

- **To avoid this, add restriction that all values in result must be values in the domain of the expression.**

# Domain Relational Calculus

- Uses variables that take values from <u>domains</u> instead of tuples of relations.

- If $F(d_1, d_2, \ldots, d_n)$ stands for a formula composed of atoms and $d_1, d_2, \ldots, d_n$ represent domain variables, then:

  $\{d_1, d_2, \ldots, d_n \mid F(d_1, d_2, \ldots, d_n)\}$

  is a general domain relational calculus expression.

# Example - Domain Relational Calculus

- Find the names of all managers who earn more than £25,000.

  {fN, lN | (∃sN, posn, sex, DOB, sal, bN)
   (Staff (sN, fN, lN, posn, sex, DOB, sal, bN) ∧
   posn = 'Manager' ∧ sal > 25000)}

# Example - Domain Relational Calculus

- **List the staff who manage properties for rent in Glasgow.**

  {sN, fN, lN, posn, sex, DOB, sal, bN |
  (∃sN1,cty)(Staff(sN,fN,lN,posn,sex,DOB,sal,bN) ∧
  PropertyForRent(pN, st, cty, pc, typ, rms,
    rnt, oN, sN1, bN1) ∧
  (sN=sN1) ∧ cty='Glasgow')}

# Example - Domain Relational Calculus

- **List the names of staff who currently do not manage any properties for rent.**

  {fN, lN | (∃sN)
   (Staff(sN,fN,lN,posn,sex,DOB,sal,bN) ∧
   (~(∃sN1) (PropertyForRent(pN, st, cty, pc, typ,
           rms, rnt, oN, sN1, bN1) ∧ (sN=sN1))))}

# Example - Domain Relational Calculus

- **List the names of clients who have viewed a property for rent in Glasgow.**

  {fN, lN | ($\exists$cN, cN1, pN, pN1, cty)
  (Client(cN, fN, lN,tel, pT, mR) $\wedge$
  Viewing(cN1, pN1, dt, cmt) $\wedge$
  PropertyForRent(pN, st, cty, pc, typ,
          rms, rnt,oN, sN, bN) $\wedge$
  (cN = cN1) $\wedge$ (pN = pN1) $\wedge$ cty = 'Glasgow')}

# Domain Relational Calculus

- **When restricted to safe expressions, domain relational calculus is equivalent to tuple relational calculus restricted to safe expressions, which is equivalent to relational algebra.**

- **Means every relational algebra expression has an equivalent relational calculus expression, and vice versa.**

# Other Languages

- **Transform-oriented languages are non-procedural languages that use relations to transform input data into required outputs (e.g. SQL).**

- **Graphical languages provide user with picture of the structure of the relation. User fills in example of what is wanted and system returns required data in that format (e.g. QBE).**

# Other Languages

- **4GLs can create complete customized application using limited set of commands in a user-friendly, often menu-driven environment.**

- **Some systems accept a form of *natural language*, sometimes called a 5GL, although this development is still at an early stage.**