Relational Calculus and Relational Algebra Review DDL and DML SQL

Lesson 5

Northeastern University
Kathleen Durant

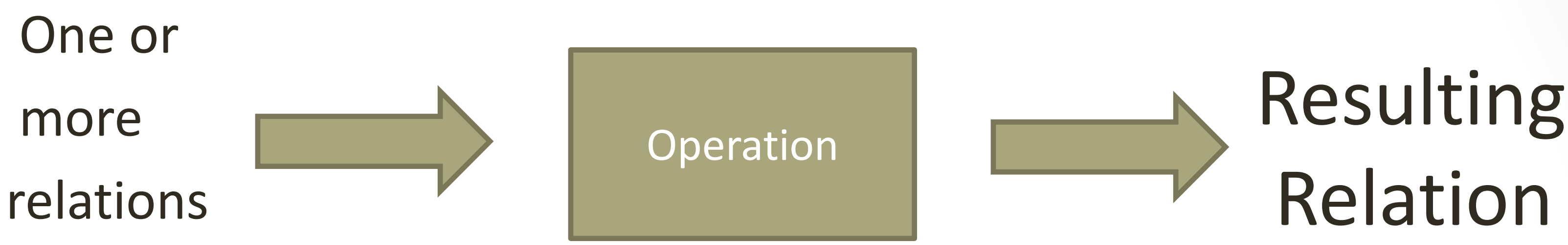
Outline for today

- Review of Chapter 4
  - Quantifiers
  - Relational Algebra & Relational Calculus
- Introduce examples from the text
  - Students, Courses, Lecturers
  - Boats, Sailors, Reservations
- Review of DDL SQL Queries
- Introduction to the SELECT command
  - Basics, Set operations, Nested Queries, Aggregation functions
- Additional information for the homework assignment

Data manipulation via Relational Algebra

- Data is represented as mathematical relations.
- Manipulation of data (query and update operations) corresponds to **operations** on relations
- Relational algebra describes those operations
- Relational algebra contains two kinds of operators:
  - common set-theoretic operators
  - operators specific to relations (for example projection of columns).

Relational Algebra



- A collection of operations that users can perform on relations to obtain a desired result (which is also a relation)
- For each operation (steps in the computation), both the operands and the result are relations
- Basic (Relational) operations:
  - Selection (σ): Selects a subset of tuples from a relation.
  - Projection (π): Selects columns from a relation.
  - Cross-product ($\times$): Allows us to combine two relations.
  - Set-difference ($-$): Tuples in relation 1, but not in relation 2.
  - Union ($\cup$): Tuples in relation 1 and in relation 2.
- Relational Algebra treats relations as sets: duplicates are removed

Example: Different solutions – same answer

S1

SID	Name	Login	DoB	GPA
55515	Smith	smith@ccs	Jan 10,1990	3.82
55516	Jones	jones@hist	Feb 11, 1992	2.98
55517	Ali	ali@math	Sep 22, 1989	3.11
55518	Smith	smith@math	Nov 30, 1991	3.32

Find the names of students registered for History 101

Solution1: $\pi_{Name}((\delta_{cid\ =\ 'History\ 101'}$ Courses) $\bowtie$S1)

Solution2: $\pi_{Name}(\delta_{cid\ =\ 'History\ 101'}$ (Courses $\bowtie$S1))

Solution3: ρ(Temp1, ($\delta_{cid\ =\ 'History\ 101'}$ Courses))

ρ(Temp2,(Temp1 $\bowtie$S1))

π_{Name}(Temp2)

C1

Sid	CId	Grade
55515	History 101	C
55516	Biology 220	A
55517	History 101	B
55518	Music 101	A

Answer

Name
Smith
Ali

Example: 3 Table join

Find the lecturers teaching History 101 Whose Students GPA >3.2

Solution1: π_{Name}
$((\pi_{Sid,GPA}(\delta_{GPA > 3.2}\ S1)) \bowtie ((\delta_{cid = 'History\ 101'}\ C1) \bowtie \delta_{cid = 'History\ 101'}\ L1))$

Solution2:

$\rho(Temp1, (\delta_{cid = 'History\ 101'}\ C1))$

$\rho(Temp2, (Temp1 \bowtie \delta_{cid = 'History\ 101'}\ L1))$

$\rho(Temp3, (\pi_{Sid,GPA}(\delta_{GPA > 3.2}\ S1) \bowtie Temp2))$

$\pi_{Name}(Temp3)$

S1

SID	Name	Login	DoB	GPA
55515	Smith	smith@ccs	Jan 10,1990	3.82
55516	Jones	jones@hist	Feb 11, 1992	2.98
55517	Ali	ali@math	Sep 22, 1989	3.11
55518	Smith	smith@math	Nov 30, 1991	3.32

L1

LID	Name	CID
45	Fisk	History 101
46	Alder	Biology 220
47	Wong	History 101
48	Foster	Music 101

C1

Sid	CId	LID	Grade
55515	History 101	45	C
55516	History 101	47	A
55517	History 101	45	B
55518	Music 101	48	A

Answer

Name
Fisk

Why did I need $\pi_{Sid,GPA}$ to use a natural join?
Any other solution?

Table Instances

- We will use these instances of the Sailors and Reserves relations in our examples.
- If the key for the Reserves relation contained only the attributes sid and bid, how would the semantics differ?

B1

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

R1

SID	BID	DAY
22	101	10/10/96
58	103	11/12/96

S1

SID	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

S2

SID	Sname	Rating	Age
28	Yuppy	9	35.0
31	Lubber	8	55.5
44	Guppy	5	35.0
58	Rusty	10	35.0

Relational calculus

- A formal, logical description, of what you want from the database
- Sometimes describing the set you desire is easier than figuring out the operations you need to do to get to the desired set
- Case in point: Division

Division Operation in RA A/B

- Given 2 relations A (courses) and B (students); A/B = let x, yA be two attributes in A and yB is an attribute in B with the same domain as the domain of yB
- A/B **= {<x> such that for all <y> in B there exists <x ,y> an element of A =** $\{< \boldsymbol{x} > | \forall < \boldsymbol{y} > \in \boldsymbol{B}\ \exists < \boldsymbol{x}, \boldsymbol{y} > \in \boldsymbol{A}\}$
- A/B contains all x tuples (courses) such that for every y tuple (students) in B, there is an xy tuple in A.
- Or: If the set of y values (courses) associated with an x value (students) in A contains all y values in B, the x value is in A/B.
  - In general, x and y can be any lists of attributes
  - y is the list of fields in B, and x U y is the list of fields of A.

Example of division

Table A

Student Id (x)	Course Id (y)
10	cs200
10	cs100
10	cs300
10	cs400
20	cs300
30	cs200
15	cs400
15	cs100
25	cs100
25	cs200

B

Course Id
cs200

A/B

Student Id
10
30
25

Basic operations for Division

- Compute all x values in A that are not disqualified
  - How is a value disqualified?
  - If by attaching a y value from B, we obtain a tuple NOT in A
  - $\pi_x((\pi_x(A) \text{x } B) - A)$
- $\pi_x(A) - \pi_x((\pi_x(A) \text{x } B) - A)$

Step by step process of Division

B

Course Id
cs200

A

Student Id (x)	Course Id (y)
10	cs200
10	cs100
10	cs300
10	cs400
20	cs300
30	cs200
15	cs400
15	cs100
25	cs100
25	cs200

$(\pi_x(A) \text{x } B)$

10, cs200
20, cs200
30, cs200
15,cs200
25, cs200

$(\pi_x(A) \text{x } B)$ - A

20, cs200
15,cs200

$\pi_x((\pi_x(A) \text{x } B) - A)$

20
15

$\pi_x(A) - \pi_x((\pi_x(A) \text{x } B) - A)$

Student Id
10
30
25

Division via Relational Calculus

- Select students who have taken all courses

- Algebra :
- $\pi_x(A) - \pi_x((\pi_x(A) \text{x}\, B) - A)$
- Calculus:
- $\{<N> \mid \exists I, L, D, G\ (< I, N, L, D, G > \in S1 \wedge$
  $\forall <I,C,G> \in C1\ (\exists <Ic,C,G> \in C1 \wedge$
  $(S1.I = C1.Ic))\}$

- $\{<I> \mid \exists C\ (< I, C > \in S1 \wedge$
  $\forall <C> \in C1\ (\exists <Ic> \in C1 \wedge$
  $(S1.I = C1.Ic))\}$

- SO MUCH EASIER

Unsafe queries

- Queries that have an infinite number of responses yet are syntactically correct
- Simple example – all students not in the table

$$\left\{ S \middle| \neg\left(S \in Students \right) \right\}$$

- Expressive theorem: every query that can be expressed in relational algebra can be expressed as a safe predicate calculus formula
- Relational completeness of a query language: every query that can be expressed in relational algebra can be expressed in the language

Summary

- The relational model has rigorously defined query languages — simple and powerful.
- Relational algebra is more operational
  - useful as an internal representation for query evaluation plans.
- Relational calculus is non-operational
  - Users define queries in terms of what they want, not in terms of how to compute it. (*Declarative*)
- Several ways of expressing a given query
  - a *query optimizer* should choose the most efficient version.
- Algebra and safe calculus have same *expressive power*
  - leads to the notion of *relational completeness*.

Onto SQL

- Review of DDL
- Introduction to DML (SELECT command)

SQL

- SQL provides
  - A data definition language (DDL)
  - A data manipulation language (DML)
  - A data control language (DCL)
- SQL can be used from other languages
- SQL Is often extended to provide common programming constructs (such as if then tests, loops, variables, etc.) Example T-SQL

DDL – CREATE TABLE

CREATE TABLE

<table name> (<col-def-1>,

<col-def-2>, …

<col-def-n>,

<constraint-1>, …

<constraint-k>)

- You supply
  - name for the table
  - A list of column definitions
  - A list of constraints (such as keys)

DDL – What is a Column Definition?

<col-name> <type>
[NULL|NOT NULL]
[DEFAULT <value>]
[constraint-1],
[constraint-2], [...]]]

- Each column has a
  - Name
  - Data Type
- Common data types
  - **INT**
  - **REAL**
  - **CHAR(n)**
  - **VARCHAR(n)**
  - **DATE**

DDL: Column Specifications

- Columns can be specified as **NULL** or **NOT NULL**
- **NOT NULL** columns cannot have missing values
  - If neither is given, then columns are allowed to have **NULL** values
- Columns can be given a default value
  - You just use the keyword DEFAULT followed by the value, e.g.: **fieldnum INT DEFAULT 0**
- **Example: CREATE TABLE Student (stuID INT NOT NULL,**
  stuName VARCHAR(50) NOT NULL,
- **stuAddress VARCHAR(50),**
  stuYear INT DEFAULT 2017)

DDL: Constraints

- **CONSTRAINT <name> <type> <details>**
  - Common **<type>**s
    - **PRIMARY KEY**
    - **UNIQUE**
    - **FOREIGN KEY**
    - **INDEX**
- Each constraint may be given a name –

  Most RDMS requires a name, but some others don't
- Constraints which refer to single columns can be included in the column definition

DDL: Primary Keys

- Primary Keys are defined through constraints
- A **PRIMARY KEY** constraint also includes a **UNIQUE** constraint and makes the columns involved **NOT NULL**
- The **<details>** for a primary key is a list of columns which make up the key
- **CONSTRAINT <name> PRIMARY KEY (col1, col2, …)**

DDL : UNIQUE

- Any set of columns can be specified as **UNIQUE**
  - This has the effect of making candidate keys in the table
  - The **<details>** for a unique constraint are a list of columns which make up the candidate key
- **CONSTRAINT <name> UNIQUE (col1, col2, …)**

- **Example: CREATE TABLE Student**
- **(stuID INT NOT NULL,**
- **stuName VARCHAR(50) NOT NULL,**
- **stuAddress VARCHAR(50),**
- **stuYear INT DEFAULT 2017,**
- **CONSTRAINT pkStudent PRIMARY KEY (stuID),**
- **CONSTRAINT uniqueName stuName)**

DDL: Foreign Keys

- Foreign Keys are also defined as constraints
  - You need to provide:
  - The columns which make up the Foreign Key
  - The referenced table
  - The columns which are referenced by the Foreign Key
- **CONSTRAINT <name> FOREIGN KEY (col1, col2,…) REFERENCES <table> [(ref1, ref2,…)]**
- If the Foreign Key references the Primary Key of **<table>** you don’t need to list the columns

DDL: Example with constraints

- **CREATE TABLE Enrollment (stuID INT NOT NULL,**
- **modCode CHAR(6) NOT NULL,**
- **enrAssignment INT,**
- **enrExam INT,**
- **CONSTRAINT enrPK PRIMARY KEY (stuID, modCode),**
- **CONSTRAINT enrStu FOREIGN KEY (stuID) REFERENCES Student (stuID),**

CONSTRAINT enrMod FOREIGN KEY (modCode) REFERENCES Module (modCode))

DDL Language: Alter Table

- ALTER TABLE can
  - Add a new column
  - Remove an existing column
  - Add a new constraint
  - Remove an existing constraint
- To add or remove columns use command
  - **ALTER TABLE <table> ADD COLUMN <colname, type>**
  - **ALTER TABLE <table> DROP COLUMN <name>**
- Examples
  - **ALTER TABLE Student ADD COLUMN Degree VARCHAR(50)**
  - **ALTER TABLE Student DROP COLUMN Degree**

DDL: Add constraint using ALTER

- Used when you want to add or drop a constraint after the table has been created
- **ALTER TABLE <table> ADD CONSTRAINT <definition>** (as defined previously)
- **ALTER TABLE <table> DROP CONSTRAINT <name>** (only need name of constraint to drop)
- Examples
  - **ALTER TABLE Module ADD CONSTRAINT ck UNIQUE (title)**
  - **ALTER TABLE Module DROP CONSTRAINT ck**

Other DDL Commands

- **DROP** - deletes a table
- **INSERT** - add a row to a table
- **UPDATE** – change row(s) in a table
- **DELETE** – remove row(s) from a table
- **UPDATE** and **DELETE** use '**WHERE** clauses' to specify which rows to change or remove
  - BE CAREFUL with these - an incorrect **WHERE** clause can destroy lots of data

Chapter 5: SELECT command

Basic DML SQL command for retrieval

SELECT [DISTINCT] target-list FROM
relation-list WHERE qualification

- Relation-list: List of tables names [possibly with a range variable (alias) after each name]
  - You can also specify a database name
  - Databasename.tablename
- Target-list: list of attributes wanted from the relation-list
  - Databasename.tablename.fieldname
- Qualification: comparisons (Attribute op const or Attribute op Attribute2, where op is one of (<,>,=,<=,>=,<>) can combine with AND, OR and NOT
- DISTINCT: Optional keyword indicating that the answer should not have duplicates
  - Default: duplicates are not eliminated

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy
  - Compute the cross product of relation-list
  - Discard resulting tuples if they fail qualifications
  - Delete attributes that are not in target-list
  - If distinct is specified, eliminate duplicate rows
- This strategy is probably the LEAST EFFICIENT way to compute a query
- Query optimizer should find more efficient strategies to compute the same answer

Example of Conceptual Evaluation

SELECT S.sname from sailors S, Reserves R where S.sid = R.sid and R.bid = 103

(sid)	sname	rating	age	(sid)	bid	day
22	Dustin	7	45.0	22	101	10/10/96
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	22	101	10/10/96
31	Lubber	8	55.5	58	103	11/12/96
58	Rusty	10	35.0	22	101	10/10/96
58	Rusty	10	35.0	58	103	11/12/96

(sid)	sname	rating	age	(sid)	bid	day
58	Rusty	10	35.0	58	103	11/12/96

Range variables or Aliases

- Are only necessary when you need to distinguish items within the query
  - Same named field or table
- Shows good coding practice
  - Less typing

- SELECT S.sname from sailors S, Reserves R where S.sid = R.sid and R.bid = 103

Equivalent

- SELECT sname, from Sailors, Reserves where sailors.sid = Reserves.sid and bid = 103

Find sailors who have reserved at least one boat

- What affect would adding a DISTINCT make on this query
- What is the effect of replacing S.sid by S.name in the SELECT clause? Do we still need a DISTINCT?

```
SELECT S.sid from
Sailors, Reserves R
where S.sid = R.sid
```

```
SELECT S.sid from
Sailors S Join Reserves R
on S.sid = R.sid
```

Expressions and Strings

- SELECT S.age, Age1=s.age-5, 2*S.age as Age2 from Sailors S where S.name like ‘B_%B’
- Returns triples of Ages for sailors whose names begin and end with B that are at least 3 characters long
- Can do computation within a SELECT statement
- Can assign variables to that computation using 2 different syntax
- Can do pattern matching using the like operator
  - % 0 or more characters
  - _ Any one character

Find sids of sailors who have reserved a red **or** green boat

- UNION computes the union of any two union-compatible sets
  - typically intermediate results
- Other set operator EXCEPT returns tuples in the first set that are not found in the Second Set
  - Not supported by MySQL – there is a workaround use NOT EXIST
- What happens if we replace the OR in the qualifier with and AND ?

SELECT s.SID FROM Sailors S, Boats B, Reserves R where S.sid = R.sid and R.bid = B.bid and (B.color = 'red' OR B.color = 'green')

SELECT s.SID FROM Sailors S, Boats B, Reserves R where S.sid = R.sid and R.bid = B.bid and (B.color = 'red')

UNION

SELECT s.SID FROM Sailors S, Boats B, Reserves R where S.sid = R.sid and R.bid = B.bid and (B.color = 'green')

Find the sid's of sailors who have reserved a red boat **and** a green boat

- Solution using a join
- Can also be solved using Intersect
  - MySql does not support Intersect
  - Workaround is to use EXISTS involves a subquery – will cover when we discuss subqueries

- SELECT S.sid from Sailors S, Boats B1, Reserves R1, Boats B2, Reserves R2 where S.sid=R1.sid and R1.bid=B1.bid and S.sid=R2.sid and R2.bid=B2.bid and (B1.color = 'red' and B2.color = 'green')

Nested queries

- Find names of sailors who have reserved boat #103
- SELECT S.name from Sailors S where S.sid in (SELECT R.sid from Reserves R where R.bid = 103)
- For each Sailor tuple check the Sid against the return of the nested query
- Where clause can be a complete query
  - Also true for FROM clause and HAVING clause
- In clause can be negated
  - Variable not in (…)
- Semantics for a nested query similar for a nested loop in programming

Nested queries with correlation

- SELECT names of sailors who have reserved boat #103
- SELECT S.sname from Sailors S where **exists**

    (SELECT * from Reserves R where R.bid = 103 and **S.sid** = R.sid)

Exists tests to see if the return set is empty

Set Operations

- IN, EXISTS, ANY as well as negation of these
- Missing Unique and Intersect in My SQL
- An **ANY** example:
- SELECT S.name from Sailors S where rating >

  any (SELECT S2.rating from Sailors S2

  where S2.name = ‘Horatio’)

- Find sailors with a higher rating than Horatio

Getting around no INTERSECT operator in MySQL

- Find sailor ids that have reserved a red boat and also a green boat
- SELECT s.sid from Sailors s, Boats B, Reserves R

  where S.sid =R.sid and R.sid=B.sid

  and **B.color='red'** and S.sid in

  (SELECT S2.sid Sailors S2, Boats B2, Reserves R2

  where S2.sid=R2.sid and R2.bid = B2.bid

  and **B2.color = 'green')**
- Use IN to define the opposing set

Division in SQL - MYSQL

- SELECT S.name, from Sailors S where not exists

  (SELECT B.bid from Boats B where not exists

   (SELECT R.bid from Reserves R

   where R.bid = B.bid and R.sid =S.sid)

Find sailors such that (line 1)

There is no boat without (line 2)

a Reserves tuple showing that sailor S reserved boat B

DML: Aggregate operators

- Significant extension to Relational Algebra
  - Operators: **count, avg, stdev, min, max, sum**
  - Examples count(*) , count([DISTINCT] FIELD), SUM([DISTINCT]FIELD), AVG([DISTINCT]FIELD), MIN(A), MAX(A)
  - SELECT COUNT(*) FROM Sailors S
  - SELECT AVG(S.age) from Sailors S where S.rating = 10
  - SELECT S.name from Sailors S where S.rating = (SELECT MAX(S2.rating) from Sailors S2)

DML: Examples of Aggregators

- SELECT AVG(Distinct S.Age) from Sailors S where S.rating=10
- Interpretation of Query?
- A particular age can only contribute once to the average
- SELECT AVG(S.Age) from Sailors S where S.rating=10
- Interpretation of Query?
- Every person's age contributes to the average (50 Sailors – 50 numbers contribute to the average

Complete SELECT command

- **SELECT [DISTINCT | ALL] <column-list> FROM <table-names> [WHERE <condition>] [ORDER BY <column-list>] [GROUP BY <column-list>] HAVING <condition>] [ORDER BY <column-list>]**
  - (*optional [], **|** - or)*
- Still need to introduce group by, order by and having
  - Next meeting