Database Management System

Lecture 2

Relational Algebra and SQL

* Some materials adapted from R. Ramakrishnan, J. Gehrke

Today's Agenda

- Relational Algebra
- Complex SQL

Relational Algebra

Relational DB and Algebra

- SQL
- Practical definition of relational DB
  - Operates on Tables (bags)
- Operations
  - Keywords
  - Statements: SELECT, FROM, WHERE,…

- Relational Algebra
- Mathematical definition of Relational DB
  - Operates on Relations (Sets)
- Operations
  - set-based operations
  - Intersection, Union,...

- The default is to produce a bag of rows as a query result
- Want a set, use DISTINCT

Describing a relational DB mathematically

- Two ingredients
  - A relation is a set of tuples
  - Define query operators as a set functions

Recap: Cross product with Set

- Let A = {a, b, c} and B = {1, 2}
- Cross product in set theory is defined as ordered pairs (2-tuples) where each pair consists of an element from A and B

$$A \times B = \{(a, 1), (b, 1), (c, 1), (a, 2), (b, 2), (c, 2)\}$$

- How about A = {a, b, c}, B = {1, 2}, and C = {α, β}?

Defining Relations

Person(name, salary, num, status)

name = {all possible strings of 30 characters}
salary = {real numbers between 0 and 100,000,000}
num = {integer between 0 and 9999}
status = {“a”, “b”}

- Any instance of the relation is always a subset ($\subseteq$) of attributes
  - name × sal × num × status

- Each relation instance is a subset of the cross product of its domains
- one element of a relation is called tuple
- A relation is always a set by definition

Recap: Set Theory

$A = \{1, 3, 5, 7\}$ $B = \{1, 2, 3, 4\}$

- What do these return?
  - $A \cap B$
  - $A \cup B$
  - $A - B$
  - $A \times B$

Relational Algebra has Additional Operations

$A = \{1, 3, 5, 7\}$ $B = \{1, 2, 3, 4\}$

- Introducing new operators

  (C for condition, L for attribute list, R for renaming specification)

  - $A \bowtie_c B$
  - $A \div B$
  - $\boldsymbol{\sigma}_c (A)$
  - $\pi_L (A)$
  - $\rho_R (A)$

Relational Algebra as a Query Language

- We don’t normally use relational algebra directly
  - Products don’t allow you to write relational algebra queries
- But, it is used internally in a DBMS to represent a query plan
- It is also often used in theoretical work on databases
  - (although fragments of first order logic are frequently used as well ...)

Relational Algebra Queries w/out Operators

- What does the following SQL query return?

```
SELECT *
FROM Student;
```

Student

Student
John Cusack
Will Smith

- Answer: Student

  (It is called identity function)

- A relation name by itself is a valid relational algebra query
- Listing the relation name just returns the tuples in the relation

Relational Algebra: Selection operator (σ)

Account

Number	Owner	Balance	Type
7003001	Jane Smith	1,000,000	Savings
7003003	Alfred Hitchcock	4,400,200	Savings
7003005	Takumi Fujiwara	2,230,000	Checking
7003007	Brian Mills	1,200,000	Savings

- The relational algebra query

$$\sigma_{\text{Balance<3000}} (\text{Account})$$

- Is similar to the SQL query

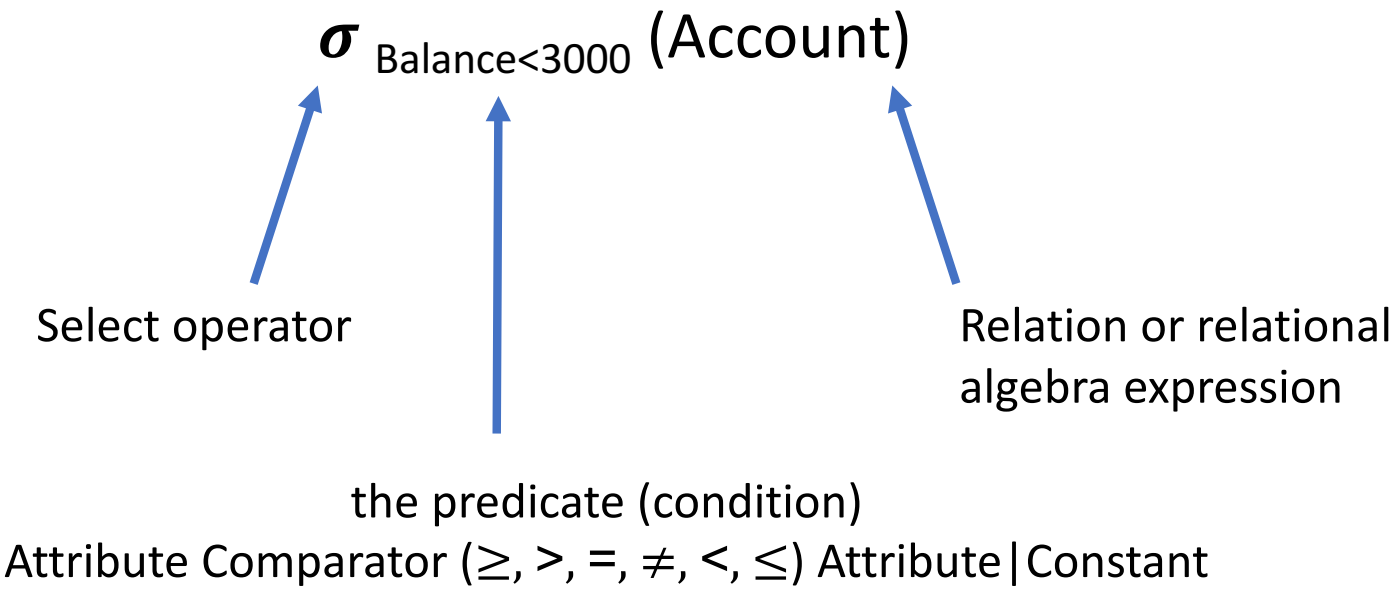
```
SELECT *
FROM Account
WHERE Balance < 3,000,000;
```

Relational Algebra: Selection operator ($\boldsymbol{\sigma}$)

- Select ($\boldsymbol{\sigma}$) is a unary operator:

  $\boldsymbol{\sigma} : R \rightarrow R$

  - It is always applied to a single relation



Exercises

- $\sigma_{\text{Balance<3,000,000}}$ (Account)
- $\sigma_{\text{Number<7003005}}$ (Account)
- $\sigma_{\text{Balance=Number}}$ (Account)
- $\sigma_{\text{Type="checking"}}$ ($\sigma_{\text{Balance<3,000,000}}$ (Account))

Account

Number	Owner	Balance	Type
7003001	Jane Smith	1,000,000	Savings
7003003	Alfred Hitchcock	4,400,200	Savings
7003005	Takumi Fujiwara	2,230,000	Checking
7003007	Brian Mills	1,200,000	Savings

Relational Algebra: Projection Operator(π)

Account

Number	Owner	Balance	Type
7003001	Jane Smith	1,000,000	Savings
7003003	Alfred Hitchcock	4,400,200	Savings
7003005	Takumi Fujiwara	2,230,000	Checking
7003007	Brian Mills	1,200,000	Savings

- The relational algebra query:

$$\pi_{\text{Number, Owner}}(\text{Account})$$

- Is similar to the SQL query

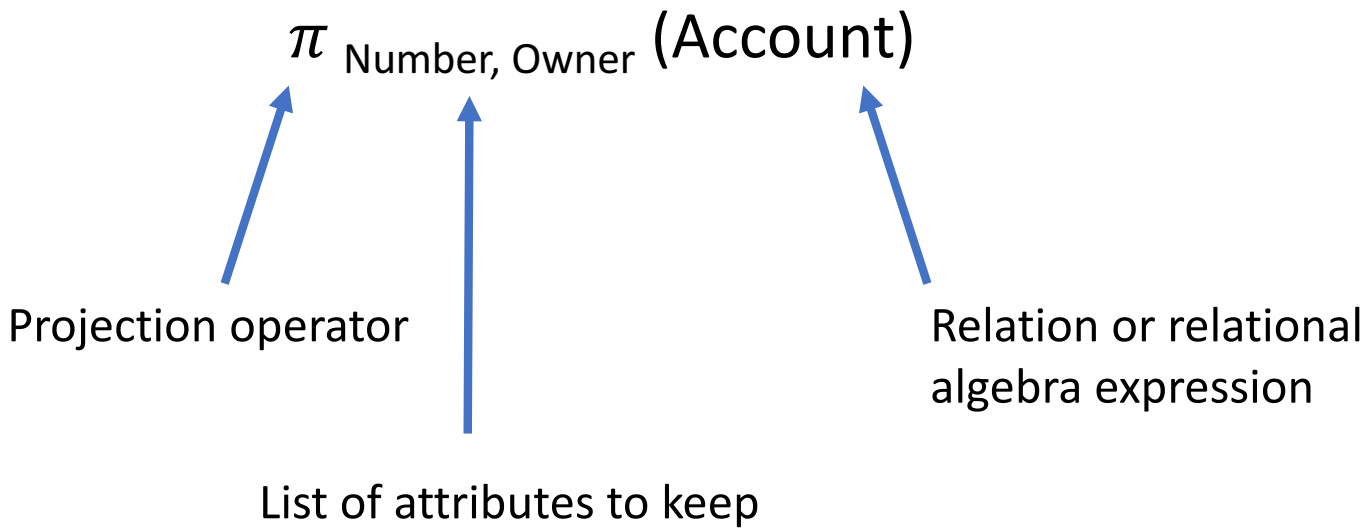
```
SELECT Number, Owner
FROM Account;
```

Relational Algebra: Projection operator (π)

- Projection (π) is a unary operator:

  $\pi : R \rightarrow R$

  - It is always applied to a single relation



Example

π_{Owner} (Account) Vs. SELECT Number FROM Account;

Account

Number	Owner	Balance	Type
7003001	Jane Smith	1,000,000	Savings
7003003	Alfred Hitchcock	4,400,200	Savings
7003005	Takumi Fujiwara	2,230,000	Checking
7003007	Brian Mills	1,200,000	Savings
7003009	Alfred Hitchcock	3,400,200	Checking

Owner
Jane Smith
Alfred Hitchcock
Takumi Fujiwara
Brian Mills

- Relations are always sets
- Query answer is a set of names
- and J. Smith appears just once in the answer

Number
7003001
7003003
7003005
7003007
7003009

Combining Select and Project

- Are any of these equivalent ?

$\pi_{\text{Owner}}(\sigma_{\text{Balance} < 3{,}000{,}000}(\text{Account}))$

$\sigma_{\text{Balance}<3{,}000{,}000}(\pi_{\text{Owner, Balance}}(\text{Account}))$

$\pi_{\text{Owner}}(\sigma_{\text{Balance}<3{,}000{,}000}(\pi_{\text{Owner, Balance}}(\text{Account})))$

$\sigma_{\text{Type} = \text{“checking”}}(\sigma_{\text{Balance}<3{,}000{,}000}(\pi_{\text{Owner, Balance}}(\text{Account})))$

Account

Number	Owner	Balance	Type
7003001	Jane Smith	1,000,000	Savings
7003003	Alfred Hitchcock	4,400,200	Savings
7003005	Takumi Fujiwara	2,230,000	Checking
7003007	Brian Mills	1,200,000	Savings
7003009	Alfred Hitchcock	3,400,200	Checking

Relational Algebra: Cross Product operator (×)

- Used in the basic definition of a relation
  - “An instance of a relation is a subset of the cross product of its domains”

- Is also an operator in the relational algebra

Example

- Suppose we have following two relations

Teacher(TID, Tname)

Teacher

TID	Tname
101	Emma Thompson
105	Billy Elliot
110	John Waine

Course(CID, Cname)

Course

CID	Cname
346	How to Act
491	How to Think

- The cross product produces every possible combinations of teacher and courses

Teacher X Course

SELECT * FROM Teacher, Course;

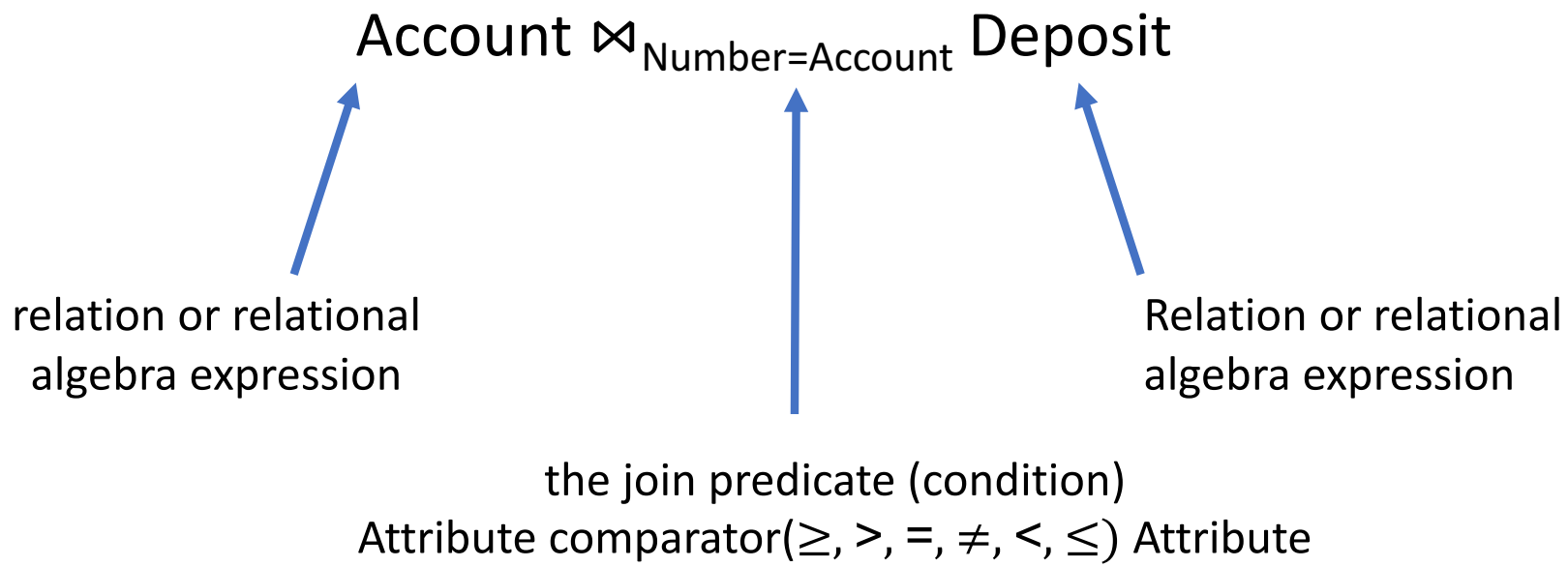
TID	Tname	CID	Cname
101	Emma Thompson	346	How to Act
101	Emma Thompson	491	How to Think
105	Billy Elliot	346	How to Act
105	Billy Elliot	491	How to Think
110	John Waine	346	How to Act
110	John Waine	491	How to Think

Relational Algebra: Join operator (⋈)

- Join () is a binary operator

  $\bowtie : R \times R \rightarrow R$

  - It is always applied to a two relations and returns one



Relational Algebra: Join operator (⋈)

Account

Number	Owner	Balance	Type

Deposit

Accnt	TxID	Date	Amount

- The relational algebra query

$$\text{Account} \bowtie_{\text{Number=Accnt}} (\text{Deposit})$$

- is equivalent to

$$\sigma_{\text{Number = Accnt}} (\text{Account} \times \text{Deposit})$$

Relational Algebra: Join operator (⋈)

- The join operator is defined for convenience

$$R1 \bowtie_{a1=a2} R2 \equiv \sigma_{a1=a2} (R1 \times R2)$$

- Any query with a join can always be rewritten into cross product followed by selection

Notes on Join

- Each simple Boolean predicate in the join condition must compare an attribute from one relation to an attribute in the other relation

$$\text{Account} \bowtie_{\text{Number = Account ^ type = “checking”}} \text{Deposit}$$

  - type=“checking” is not a join condition

- if you have a join with NO condition, then it is just a cross product

Examples

S instance of Student

sid	name	advisor	age
101	Bill	301	20
102	John	302	20
103	Edward	301	19
104	Albert	301	19
105	Thompson	302	19

F instance of Faculty

fid	name	age
301	Morrison	45
302	Groot	37

- $S \bowtie_{advisor=fid} (F)$
- $S \bowtie_{S.age < F.age} (F)$

- The most common join is called a equi-join (for equality condition)

$$R1 \bowtie_{A1 = A2} R2$$

SQL statement to an relational Algebra expression

SELECT DISTINCT attributes
FROM T1, T2, …
WHERE conditions

?

- SELECT-FROM-WHERE queries are sometimes described as equivalent to the Select-Project-Join (SPJ) subset of relational algebra

Complex SQL

More SQL query constructs

1. SELECT …
2. FROM …
3. WHERE …

(SELECT … FROM … WHERE …)

4. UNION

(SELECT … FROM … WHERE …)

ORDER BY …

5. GROUP BY …

HAVING …

1. Extensions: SUM, COUNT, MIN, AVG, etc
2. Extensions include various kinds of JOINs
3. Additional comparators, e.g. EXISTS, IN, ANY

4. Operators that takes two or more complete SQL queries as arguments, e.g., UNION and INTERSECT

5. Several additional clauses, e.g., ORDER BY, GROUP BY, and HAVING

More SQL query constructs

1. SELECT …
2. FROM …
3. WHERE …

(SELECT … FROM … WHERE …)

4. UNION

(SELECT … FROM … WHERE …)

ORDER BY …

5. GROUP BY …

HAVING …

1. Extensions: SUM, COUNT, MIN, AVG, etc
2. Extensions include various kinds of JOINs
3. Additional comparators, e.g. EXISTS, IN, ANY

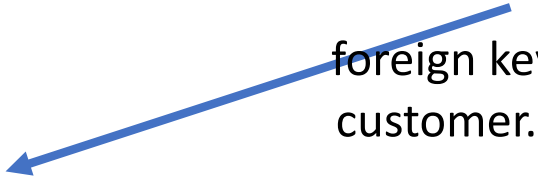
4. Operators that takes two or more complete SQL queries as arguments, e.g., UNION and INTERSECT

5. Several additional clauses, e.g., ORDER BY, GROUP BY, and HAVING

Sample Database

- Let's consider the following DB for the examples

Customer(Number, Name, Address, Crating,
Camount, Cbalance, Salesperson)



foreign key
customer.Salesperson ->Salesperson.Number

Salesperson(Number, Name, Address, Office)

- We are going to other DBs time to time

SELECT (1/4)

- Aggregate Operators: COUNT, SUM, MIN, MAX, and AVG

```
SELECT MIN(Cbalnace), MAX(Cbalance), AVG(Cbalance)
FROM Customer;
```

```
SELECT MIN(Cbalnace), MAX(Cbalance), AVG(Cbalance)
FROM Customer
WHERE age > 35;
```

- If one aggregate operator appears in the SELECT clause
  - ALL OF THE ENTRIES in the select clause MUST BE AN AGGREGATE OPERATOR
  - Unless the query includes a GROUP BY clause (more on later)

Stop to think

- What would/should the query result be?
- Is it allowed?

```
SELECT Name, Crating, AVG(Cbalance)
FROM Customer;
```

SELECT (2/4)

- What is the difference between these two queries?

```
SELECT COUNT(Name)
FROM Customer;
```

Vs.

```
SELECT DISTINCT Name
FROM Customer;
```

- When will these two queries return the same answer?
  - or what are the conditions for it to happen

SELECT (3/4)

- What is the implication of using DISTINCT
  - When computing the SUM or AVG of an attribute?

SUM(DISTINCT(AGE)) Vs. SUM(age)

  - When computing the MIN or MAX of an attribute?

MIN(DISTINCT(AGE)) Vs. MIN(age)

SELECT (4/4)

- SELECT clause list can also include simple arithmetic expressions using +, -, *, /

```
SELECT (Camount – Cbalance) AS AvailableCredit, Name
FROM Customer
WHERE Camount > 0
```

More SQL query constructs

1. SELECT …
2. FROM …
3. WHERE …

(SELECT … FROM … WHERE …)

4. UNION

(SELECT … FROM … WHERE …)

ORDER BY …

5. GROUP BY …

HAVING …

1. Extensions: SUM, COUNT, MIN, AVG, etc
2. Extensions include various kinds of JOINs
3. Additional comparators, e.g. EXISTS, IN, ANY

4. Operators that takes two or more complete SQL queries as arguments, e.g., UNION and INTERSECT

5. Several additional clauses, e.g., ORDER BY, GROUP BY, and HAVING

FROM: Syntactic Sugars and new operators

- There are a number of join types that can be expressed in FROM clause
  - Inner join (the regular join)
  - Cross join
  - natural join

  } syntactic sugars that can be expressed using SELECT-FROM-WHERE queries

  - left outer join
  - right outer join
  - full outer join

  } New operators

FROM

- These two queries are equivalent

1. SELECT C.Name, S.Name
   FROM Customer C JOIN Salesperson S ON C.Salesperson = S.Number
   WHERE C.Crating < 6;

2. SELECT C.Name, S.Name
   FROM Customer C, Salesperson S
   WHERE C.Salesperson = S.Number AND C.Crating < 6;

FROM: JOIN with USING clause

- JOIN with USING clause when attributes in the 2 tables have the same name

  Course(CNumber, CName, Description)
  Teacher(TNumber, TName, Phone)
  Offering(CNumber, TNumber, Time, Days, Room)

- These Two queries are equivalent

```
SELECT C.CNumber, C.CName, Room
FROM Course C JOIN Offering USING(CNumber);

SELECT C.CNumber, C.Name, Room
FROM Course C JOIN Offering O ON C.CNumber=O.CNumber;
```

- USING clause doesn't need (and can't have) a correlation name

FROM: Basic Join ≡ (INNER) JOIN

- For the INNER JOIN

```
SELECT C.Name, S.Name
FROM Customer C INNER JOIN Salesperson S ON C.Salesperson = S.Number;
```

- The query result includes all “matches” but excludes
  - customer rows that do not have a Salesperson
  - Salesperson rows that are not assigned to any customers

- The keyword “INNER” is optional
  - above query is equivalent to

```
SELECT C.Name, S.Name
FROM Customer C JOIN Salesperson S ON C.Salesperson = S.Number;
```

FROM: cross product ≡ CROSS JOIN

- The following queries are equivalent

```
SELECT *
FROM Customer, Salesperson;
```

```
SELECT *
FROM Customer CROSS JOIN Salesperson;
```

FROM: Equi-Jioin vs. Natual Join (1/3)

- When the join is based on equality of attributes, we always have two identical attributes in the result

Faculty

Name	DeptID
Smith	1
James	2
Brown	3
Johnson	1
Robert	

Department

DeptID	DeptName
1	Engineering
2	Communications
3	Marketing

```
SELECT *
FROM Faculty F INNER JOIN Department D
ON F.DeptID = D.DeptID;
```

Equi-Join

F.Name	F.DeptID	D.DeptID	D.DeptName
Smith	1	1	Engineering
Johnson	1	1	Engineering
James	2	2	Communication
Brown	3	3	Markeing

FROM: Equi-Jioin vs. Natual Join (1/3)

- Equi-Join with the USING construct: applicable with columns having same name

Faculty

Name	DeptID
Smith	1
James	2
Brown	3
Johnson	1

Department

DeptID	DeptName
1	Engineering
2	Communications
3	Marketing

SELECT *
FROM Faculty F INNER JOIN Department D
USING (DeptID);

Equi-Join with USING construct

Name	DeptID	DeptName
Smith	1	Engineering
Johnson	1	Engineering
James	2	Communication
Brown	3	Markeing

FROM: Equi-Jioin vs. Natual Join (3/3)

- NATURAL JOIN: Equi-Join with only one column for each equally named columns

Faculty

Name	DeptID
Smith	1
James	2
Brown	3
Johnson	1

Department

DeptID	DeptName
1	Engineering
2	Communications
3	Marketing

```
SELECT *
FROM Faculty NATURAL JOIN Department;
```

Name	DeptID	DeptName
Smith	1	Engineering
Johnson	1	Engineering
James	2	Communication
Brown	3	Markeing

NATURAL JOIN

If you don't specify which attributes to join on, natural join will join on *all attributes with the same name*

FROM: more on NATURAL JOIN (1/2)

- NATURAL JOIN is like a “macro” that joins tables with an equality condition for all attributes with the same name

  Course(CNumber, CName, Description)

  Teacher(TNumber, TName, Phone)

  Offering(CNumber, TNumber, Time, Days, Room)

- NATURAL JOIN drops one of duplicate columns automatically

FROM: more on NATURAL JOIN (2/2)

- List the course and teacher name for all course offerings
- This query can be expressed with the NATURAL JOIN or with an INNER JOIN
  - These two queries are equivalent

```
SELECT CName, TName
FROM Course C, Offering O, Teaching T
WHERE C.CNumber = O.CNumber AND O.TNumber = T.Tnumber
```

```
SELECT CName, TName
FROM Course NATURAL JOIN Offering NATURAL JOIN Teacher;
```

  - They are equivalent because the join attributes have the same attribute names

- But is it always useful?

FROM: INNER JOIN Vs. OUTER JOIN (1/2)

- For the INNER JOIN

  SELECT C.Name, S.Name
  FROM Customer C INNER JOIN Salesperson S ON C.Salesperson = S.Number

- the query result does not include (p.40)
  - a customer that does not have a salesperson
  - a salesperson that is not assigned to any customers

Customer

Number	Name	Address	Crating	Camount	Cbalance	Salesperson
1	Smith	1st Str.	700	10,000	9,000	55
2	Jones	2nd Str.	700	8,000	4,000	77
3	Mills	3rd Str.	700	11,000	8,000	NULL

Salesperson

Number	Name	Address	Office
55	Miller	5th Str.	101
77	Khan	7th Str.	102
83	Dunham	8th Str.	103

FROM: INNER JOIN Vs. OUTER JOIN (2/2)

- An INNER (regular) JOIN includes only those customers that have salespersons (only the matches)

```
SELECT C.Name, S.Name
FROM Customer as C INNER JOIN Salesperson as S
        ON C.Salesperson = S.Number;
```

- A LEFT OUTER JOIN will include all matches plus all – customers that do not have a Salesperson
- A RIGHT OUTER JOIN will include all matches plus all – salespersons that are not assigned to any customers
- A FULL OUTER JOIN will include all of these

FROM: LEFT OUTER JOIN

INNER JOIN on C.Salesperson = S.Number gives:

1	Smith	1st Str.	700	10,000	9,000	55	55	Miller	5th Str.	101
2	Jones	2nd Str.	700	8,000	4,000	77	77	Khan	7th Str.	102

LEFT OUTER JOIN on C.Salesperson = S.Number gives:

1	Smith	1st Str.	700	10,000	9,000	55	55	Miller	5th Str.	101
2	Jones	2nd Str.	700	8,000	4,000	77	77	Khan	7th Str.	102
3	Mills	3rd Str.	700	11,000	8,000	NULL	NULL	NULL	NULL	NULL

Customer

Number	Name	Address	Crating	Camount	Cbalance	Salesperson
1	Smith	1st Str.	700	10,000	9,000	55
2	Jones	2nd Str.	700	8,000	4,000	77
3	Mills	3rd Str.	700	11,000	8,000	NULL

Salesperson

Number	Name	Address	Office
55	Miller	5th Str.	101
77	Khan	7th Str.	102
83	Dunham	8th Str.	103

FROM: RIGHT OUTER JOIN

INNER JOIN on C.Salesperson = S.Number gives:

1	Smith	1st Str.	700	10,000	9,000	55	55	Miller	5th Str.	101
2	Jones	2nd Str.	700	8,000	4,000	77	77	Khan	7th Str.	102

RIGHT OUTER JOIN on C.Salesperson = S.Number gives:

1	Smith	1st Str.	700	10,000	9,000	55	55	Miller	5th Str.	101
2	Jones	2nd Str.	700	8,000	4,000	77	77	Khan	7th Str.	102
NULL	NULL	NULL	NULL	NULL	NULL	NULL	83	Dunham	8th Str.	103

Customer

Number	Name	Address	Crating	Camount	Cbalance	Salesperson
1	Smith	1st Str.	700	10,000	9,000	55
2	Jones	2nd Str.	700	8,000	4,000	77
3	Mills	3rd Str.	700	11,000	8,000	NULL

Salesperson

Number	Name	Address	Office
55	Miller	5th Str.	101
77	Khan	7th Str.	102
83	Dunham	8th Str.	103

FROM: FULL OUTER JOIN

* not supported in mysql

INNER JOIN on C.Salesperson = S.Number gives:

1	Smith	1st Str.	700	10,000	9,000	55	55	Miller	5th Str.	101
2	Jones	2nd Str.	700	8,000	4,000	77	77	Khan	7th Str.	102

RIGHT OUTER JOIN on C.Salesperson = S.Number gives:

1	Smith	1st Str.	700	10,000	9,000	55	55	Miller	5th Str.	101
2	Jones	2nd Str.	700	8,000	4,000	77	77	Khan	7th Str.	102
3	Mills	3rd Str.	700	11,000	8,000	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	83	Dunham	8th Str.	103

Customer

Number	Name	Address	Crating	Camount	Cbalance	Salesperson
1	Smith	1st Str.	700	10,000	9,000	55
2	Jones	2nd Str.	700	8,000	4,000	77
3	Mills	3rd Str.	700	11,000	8,000	NULL

Salesperson

Number	Name	Address	Office
55	Miller	5th Str.	101
77	Khan	7th Str.	102
83	Dunham	8th Str.	103

FROM: a form of subquery

- You can put a complete query expression in the FROM clause
  - also known as nested queries or subqueries
  - Parentheses are important

```
SELECT ...
FROM Employee E, (SELECT ... FROM ... WHERE ...)
WHERE ...
```

Relational Algebra Operators

Eight standard relational algebra operators

- π project — We have seen already
- $\boldsymbol{\sigma}$ select — We have seen already
- $\cup$ union — From set theory
- $\cap$ intersect — From set theory
- $-$ difference — From set theory

} can only used with union-compatible relations

- $\times$ cross product — We have seen already
- $\bowtie$ join — We have seen already
- $\div$ divide
- ρ renaming

Union-compatible relations

- Two relations are union-compatible if
  - have same number of attributes
  - have same domains

- Example

  Checking(*CNum*: int, *COwner*: string, *CBalance*: int)

  Savings(*SNum*: int, *SOwner*: string, *SBalance*: int)

Example: ∪ union

Checking

Cnum	Cowner	Cbalance
101	Smith	1000
102	Mills	2000
104	Jones	1000
105	Schwab	3000

Savings

Snum	Sowner	Sbalance
103	Smith	5000

Checking ∪ Savings

Cnum	Cowner	Cbalance
101	Smith	1000
102	Mils	2000
104	Jones	1000
105	Schwab	3000
103	Smith	5000

note that attributes are from the first relation in the query

Example: ∩ intersection

Checking ∩ Savings

?

$\pi_{Cowner}(Checking) \cap \pi_{Sowner}(Savings)$

Checking

Cnum	Cowner	Cbalance
101	Smith	1000
102	Mils	2000
104	Jones	1000
105	Schwab	3000

Savings

Snum	Sowner	Sbalance
103	Smith	5000

Example: – difference

* not supported in mysql

- Find all tuples *that are* in the Checking relation but *are not* in the Savings relation

- Everyone in Checking *except* Smith

Workaround for difference operation
example query
SELECT * FROM p LEFT OUTER JOIN q ON p.id = q.id WHERE q.id IS NULL

More SQL query constructs

1. SELECT …
2. FROM …
3. WHERE …

(SELECT … FROM … WHERE …)

4. UNION

(SELECT … FROM … WHERE …)

ORDER BY …

5. GROUP BY …

HAVING …

1. Extensions: SUM, COUNT, MIN, AVG, etc
2. Extensions include various kinds of JOINs
3. Additional comparators, e.g. EXISTS, IN, ANY

4. Operators that takes two or more complete SQL queries as arguments, e.g., UNION and INTERSECT

5. Several additional clauses, e.g., ORDER BY, GROUP BY, and HAVING

UNION and INTERSECTION

- Two complete queries with UNION in between

```
(SELECT C.Name
FROM Customer C
WHERE C.Name LIKE “B%”)
UNION
(SELECT S.Name
FROM Salesperson S
WHERE S.Name LIKE “B%”);
```

- Two complete queries with INTERSECT in between

```
(SELECT C.Name
FROM Customer C)
INTERSECT
(SELECT S.Name
FROM Salesperson S);
```

- Two complete queries with EXCEPT (i.e., DIFFERENCE) in between
  - MySQL doesn’t support EXCEPT

```
(SELECT C.Name
FROM Customer C)
EXCEPT
(SELECT S.Name
FROM Salesperson S);
```

ALL in UNION, INTERSECT, and EXCEPT

- If you don’t specify ALL, the result is computed on sets
  - Eliminate duplicates from first operand
  - Eliminate duplicates from second operand
  - Compute operation
  - Eliminate duplicates from result

- Note the difference and chose wisely
  - UNION Vs. UNION ALL
  - INTERSECT Vs. INTERSECT ALL
  - EXCEPT Vs. EXCEPT ALL

More SQL query constructs

1. SELECT …
2. FROM …
3. WHERE …

(SELECT … FROM … WHERE …)

4. UNION

(SELECT … FROM … WHERE …)

ORDER BY …

5. GROUP BY …

HAVING …

1. Extensions: SUM, COUNT, MIN, AVG, etc
2. Extensions include various kinds of JOINs
3. Additional comparators, e.g. EXISTS, IN, ANY

4. Operators that takes two or more complete SQL queries as arguments, e.g., UNION and INTERSECT

5. Several additional clauses, e.g., ORDER BY, GROUP BY, and HAVING

GROUP BY

- Any SQL query can have the answer “grouped”
  - one output row for each group

```
SELECT Salesperson, COUNT(*)
FROM Customer;
```

```
SELECT Salesperson, COUNT(*)
FROM Customer
GROUP BY Salesperson;
```

Customer

Number	Name	Address	Crating	Camount	Cbalance	Salesperson
1	Smith	1st Str.	700	10,000	9,000	55
2	Jones	2nd Str.	700	8,000	4,000	77
3	Mills	3rd Str.	700	11,000	8,000	NULL

Salesperson	COUNT(*)
55	1
77	1
NULL	1

GROUP BY

```
SELECT Salesperson, COUNT(*)
FROM Customer
GROUP BY Salesperson;
```

Customer

Number	Name	Address	Crating	Camount	Cbalance	Salesperson
1	Smith	1st Str.	700	10,000	9,000	55
2	Jones	2nd Str.	700	8,000	4,000	77
3	Mills	3rd Str.	700	11,000	8,000	NULL
4	Bill	4th Str.	700	13,000	5,000	55
5	Jane	5th Str.	800	3,000	3,000	55
6	Harley	8th Str.	700	2,000	8,000	20
7	Khale	9th Str.	900	6,000	1,000	77

Example: GROUP BY

```
SELECT Salesperson, COUNT(*)
FROM Customer
GROUP BY Salesperson;
```

1. Make groups resulting in 4 Groups
2. Evaluate
   "SELECT Salesperson, Count(*)" for each group

Customer

Number	Name	Address	Crating	Camount	Cbalance	Salesperson
1	Smith	1st Str.	700	10,000	9,000	55
2	Jones	2nd Str.	700	8,000	4,000	77
3	Mills	3rd Str.	700	11,000	8,000	NULL
4	Bill	4th Str.	700	13,000	5,000	55
5	Jane	5th Str.	800	3,000	3,000	55
6	Harley	8th Str.	700	2,000	8,000	20
7	Khale	9th Str.	900	6,000	1,000	77

Salesperson	COUNT(*)
55	3
NULL	1
77	2
20	1

SQL HAVING

- HAVING clause specifies a predicate evaluated against each group
- A group is in the result if it satisfies the HAVING condition

```
SELECT Salesperson, COUNT(*)
FROM Customer
GROUP BY Salesperson HAVING COUNT(*) > 1;
```

Customer

Number	Name	Address	Crating	Camount	Cbalance	Salesperson
1	Smith	1st Str.	700	10,000	9,000	55
2	Jones	2nd Str.	700	8,000	4,000	55
3	Mills	3rd Str.	700	11,000	8,000	NULL

Salesperson	COUNT(*)
55	2

Example: GROUP BY

```
SELECT Salesperson, COUNT(*)
FROM Customer
GROUP BY Salesperson
HAVING COUNT(*) > 1;
```

1. Make groups resulting in 4 Groups
2. Check if COUNT(*) >1 holds
3. Evaluate
   "SELECT Salesperson, Count(*)" for each group

Customer

Number	Name	Address	Crating	Camount	Cbalance	Salesperson
1	Smith	1st Str.	700	10,000	9,000	55
2	Jones	2nd Str.	700	8,000	4,000	77
3	Mills	3rd Str.	700	11,000	8,000	NULL
4	Bill	4th Str.	700	13,000	5,000	55
5	Jane	5th Str.	800	3,000	3,000	55
6	Harley	8th Str.	700	2,000	8,000	20
7	Khale	9th Str.	900	6,000	1,000	77

Salesperson	COUNT(*)
55	3
NULL	1
77	2
20	1

Salesperson	COUNT(*)
55	3
77	2

Note on GROUP BY, HAVING

- The only attribute that can appear in a “grouped” query are
  - the grouping attributes
  - aggregate operators that are applied to the group

- Thus, the following is not legal

```
SELECT Name
FROM Customer GROUP BY Salesperson;
```

  - Because ther can be more than one name for each group

Exercise

Team(Name, Games, Wins, Losses, Conference)

Player(Name, Hits, AtBats, HomeRuns, Team)

Player.Team -> Team.Name

- Write SQL queries for the following
  - Average number of wins and losses across teams
  - Average number of wins and losses per conference
  - Batting average for each player, where batting average is the number of hits divided by at bats

ORDER BY

- Sort the result of a query

```
SELECT Number, Name, Salesperson
FROM Customer
ORDER BY Name;
```

Customer

Number	Name	…	Salesperson
1	Smith	…	55
2	Jones	…	77
3	Mills	…	NULL
4	Bill	…	55
5	Jane	…	55
6	Harley	…	20
7	Khale	…	77

Customer

Number	Name	…	Salesperson
4	Bill	…	55
6	Harley	…	20
5	Jane	…	55
2	Jones	…	77
7	Khale	…	77
3	Mills	…	NULL
1	Smith	…	55

ORDER BY

- Sort the result of a query

```
SELECT Number, Name, Salesperson
FROM Customer
ORDER BY Name DESC;
```

Customer

Number	Name	…	Salesperson
1	Smith	…	55
2	Jones	…	77
3	Mills	…	NULL
4	Bill	…	55
5	Jane	…	55
6	Harley	…	20
7	Khale	…	77

Customer

Number	Name	…	Salesperson
1	Smith	…	55
3	Mills	…	NULL
7	Khale	…	77
2	Jones	…	77
5	Jane	…	55
6	Harley	…	20
4	Bill	…	55

ORDER BY

- Sort the result of a query

```
SELECT Number, Name, Salesperson
FROM Customer
ORDER BY Name, Salesperson;
```

Customer

Number	Name	...	Salesperson
1	Smith	...	55
2	Jones	...	77
3	Mills	...	NULL
4	Bill	...	55
5	Jane	...	55
6	Harley	...	20
7	Khale	...	77
8	Bill	...	20

Customer

Number	Name	...	Salesperson
8	Bill	...	20
4	Bill	...	55
6	Harley	...	20
5	Jane	...	55
2	Jones	...	77
7	Khale	...	77
3	Mills	...	NULL
1	Smith	...	55

Subqueries

- It can be used in the where clause (in addition to the FROM clause)

```
SELECT C1.Number, C1.Name
FROM Customer C1
WHERE C1.CRating = (SELECT MAX(C2.Crating)
                    FROM Customer C2);
```

Outer query

Inner query

- Inner query returns
  - A single value that represents max credit rating

- Outer query returns
  - The name and number of the customer with the highest credit ratings

Example

```
SELECT C1.Number, C1.Name
FROM Customer C1
WHERE C1.CRating = (SELECT MAX(C2.Crating)
                    FROM Customer C2);
```

1. FROM clause in outer query
2. Take a row from the Customer table
3. Check if the row satisfies the WHERE clause
4. Evaluate the inner query (result: 800)
5. Evaluate if Crating is equal to the result

Customer

Number	Name	Address	Crating	Camount	Cbalance	Salesperson
1	Smith	1st Str.	200	10,000	9,000	55
2	Jones	2nd Str.	800	8,000	4,000	55
3	Mills	3rd Str.	700	11,000	8,000	NULL

Subqueries

- Subqueries can be used in the where clause (in addition to the from clause)

```
SELECT C1.Number, C1.Name
FROM Customer C1
WHERE C1.CRating = (SELECT MAX(C2.Crating)
                    FROM Customer C2);
```

- Six Comparators: =, >, < >=, <=, <> (not equal)
  - inner query must return a single value
- If the inner query does not mention any attributes from the outer query (C1 not mentioned in the inner query)
  - Then you only need to evaluate the inner query once
  - The inner (sub) query is *NOT correlated*

Subqueries: SOME/ALL comparison

```
SELECT S.Name
FROM Salesperson S
WHERE S.Name = SOME (SELECT C.Salesperson
                     FROM Customer C
                     WHERE C.CRating = 700);
```

- For SOME, the expression must be true for at least one row in the subquery answer
  - "ANY" is equivalent to SOME
- What does this query return?

Subqueries: SOME/ALL comparison

```
SELECT S.Name
FROM Salesperson S
WHERE S.Name = ALL (SELECT C.Salesperson
                    FROM Customer C
                    WHERE C.CRating = 700);
```

- For ALL, the expression must be true for all rows in the subquery answer

- What does this query return?

Subqueries: IN/NOT IN comparison (1/4)

```
SELECT C1.Number, C1.Name
FROM Customer C1
WHERE C1.Name IN (SELECT Name
                  FROM Salesperson);
```

- With IN, the attribute matches at least one value returned from the subquery
  - Same as "= SOME"

Subqueries: IN/NOT IN comparison (2/4)

```
SELECT C1.Number, C1.Name
FROM Customer C1
WHERE C1.Name NOT IN (SELECT Name
                      FROM Salesperson);
```

- With NOT IN, the attribute matches none of the values returned from the subquery
  - Same as “<> ALL”

Subqueries: IN/NOT IN comparison (3/4)

- Are these equivalent?
- Do we need to use DISTINCT for these to be equivalent?
- Is the subquery correlated?

```
SELECT S.Number, S.Name
FROM Salesperson S
WHERE S.Number IN (SELECT C.Salesperson
                   FROM Customer C);

SELECT DISTINCT S.Number, S.Name
FROM Salesperson S, Customer C
WHERE S.Number = S.Salesperson;
```

Subqueries: IN/NOT IN comparison (4/4)

```
SELECT S.Number, S.Name
FROM Salesperson S
WHERE S.Number IN (SELECT C.Salesperson
                   FROM Customer C
                   WHERE C.Name = S.Name);
```

- Because the subquery mentions an attribute from a table in the outer query
  - The subquery must be (re-)evaluated for each row in the outer query (each time the WHERE clause is evaluated)
  - Correlated subqueries can be very expensive!

Subqueries: EXISTS/NOT EXISTS (1/2)

```
SELECT C.Name
FROM Customer C
WHERE EXISTS (SELECT *
                FROM Salesperson S
                WHERE S.Number = C.Salesperson AND
                      S.Name = C.Name);
```

- If the answer to the subquery is not empty ... then the EXISTS predicate returns TRUE
  - Is this subquery correlated?
  - What does this query return?

Subqueries: EXISTS/NOT EXISTS (2/2)

```
SELECT C.Name
FROM Customer C
WHERE EXISTS (SELECT *
                FROM Salesperson S
                WHERE S.Number = C.Salesperson AND
                      S.Name = C.Name);
```

- Four predicates can be applied to a subquery
  - EXISTS : is the subquery answer non-empty?
  - NOT EXISTS : is the subquery answer empty?
  - UNIQUE : does the subquery return just one row?
  - NOT UNIQUE : does the subquery return multiple rows?

Missing Relational Algebra Operator

Divide

Divide Operator (p. 54)

- Suppose we have a extra table in our database

Account

Number	Owner	Balance	Type
7003001	Jane Smith	1,000,000	Savings
7003003	Alfred Hitchcock	4,400,200	Savings
7003005	Takumi Fujiwara	2,230,000	Checking
7003007	Brian Mills	1,200,000	Savings

AccountTypes

Type
Checking
Savings

- How do we find customers that have at least one account of each account type?

$$\pi_{\text{Owner,Type}}(\text{Account}) \div \text{AccountTypes}$$

Find account owners who have ALL types of accounts

For Next Week

- Review – Quiz on the material
  - Ch. 4 to 4.2
  - Ch. 5.5
- Reading assignments
  - Ch. 2-2.5
  - Ch. 3.5
- Be sure you understand
  - Aggregate operations
  - how join operates
  - set operators
  - GROUP BY, HAVING, ORDER BY, Subqueries