

# CSE 444: Database Internals

## Lecture 2 Review of the Relational Model

CSE 444 - Winter 2017

1

# Announcements

- Lab 1 part 1 is due on Friday
  - Lab 1 is due Jan 18th
  - “git commit -a” and “git push” often!
- HW1 is due next week on Friday.
- 544M first paper review is due next week
  - Deadlines are flexible for graduate readings

CSE 444 - Winter 2017

2

# Agenda

- Review Relational Model
- Review Queries (will skip most slides)
  - Relational Algebra
  - SQL
- Review translation SQL → RA
  - Needed for HW1

CSE 444 - Winter 2017

3

# Database/Relation/Tuple

- A **Database** is collection of relations
- A **Relation**  $R$  is subset of  $S_1 \times S_2 \times \dots \times S_n$ 
  - Where  $S_i$  is the domain of attribute  $i$
  - $n$  is number of attributes of the relation
  - A relation is a set of tuples
- A **Tuple**  $t$  is an element of  $S_1 \times S_2 \times \dots \times S_n$ 
  - Other names: relation = **table**; tuple = **row**

CSE 444 - Winter 2017

4

# Discussion

- **Rows** in a relation:
  - Ordering immaterial (a relation is a set)
  - All rows are distinct – **set semantics**
  - Query answers may have duplicates – **bag semantics**
- **Columns** in a tuple:
  - Ordering is significant
  - Applications refer to columns by their names
- **Domain** of each column is a primitive type

Data independence!

CSE 444 - Winter 2017

5

# Schema

- **Relation schema**: describes column heads
  - Relation name
  - Name of each field (or column, or attribute)
  - Domain of each field
- **Degree (or arity) of relation**: # attributes
- **Database schema**: set of all relation schemas

CSE 444 - Winter 2017

6

## Instance

- **Relation instance:** concrete table content
  - Set of tuples (also called records) matching the schema
- **Cardinality of relation instance:** # tuples
- **Database instance:** set of all relation instances

CSE 444 - Winter 2017

7

## What is the schema? What is the instance?

### Supplier

sno	sname	scity	sstate
1	s1	city 1	WA
2	s2	city 1	WA
3	s3	city 2	MA
4	s4	city 2	MA

CSE 444 - Winter 2017

8

## What is the schema? What is the instance?

Relation schema

Supplier(sno: integer, sname: string, scity: string, sstate: string)

### Supplier

sno	sname	scity	sstate
1	s1	city 1	WA
2	s2	city 1	WA
3	s3	city 2	MA
4	s4	city 2	MA

} instance

CSE 444 - Winter 2017

9

## Integrity Constraints

- Condition specified on a database schema
- Restricts data that can be stored in db instance
- DBMS enforces integrity constraints
  - Ensures only legal database instances exist
- Simplest form of constraint is domain constraint
  - Attribute values must come from attribute domain

CSE 444 - Winter 2017

10

## Key Constraints

- **Super Key:** “set of attributes that functionally determines all attributes”
- **Key:** Minimal super-key; a.k.a. “candidate key”
- **Primary key:** One minimal key can be selected as primary key

CSE 444 - Winter 2017

11

## Foreign Key Constraints

- A relation can refer to a tuple in another relation
- **Foreign key**
  - Field that refers to tuples in another relation
  - Typically, this field refers to the primary key of other relation
  - Can pick another field as well

CSE 444 - Winter 2017

12

## Key Constraint SQL Examples

```
CREATE TABLE Part (  
  pno integer,  
  pname varchar(20),  
  psize integer,  
  pcolor varchar(20),  
  PRIMARY KEY (pno)  
);
```

CSE 444 - Winter 2017

13

## Key Constraint SQL Examples

```
CREATE TABLE Supply(  
  sno integer,  
  pno integer,  
  qty integer,  
  price integer  
);
```

```
CREATE TABLE Part (  
  pno integer,  
  pname varchar(20),  
  psize integer,  
  pcolor varchar(20),  
  PRIMARY KEY (pno)  
);
```

CSE 444 - Winter 2017

14

## Key Constraint SQL Examples

```
CREATE TABLE Supply(  
  sno integer,  
  pno integer,  
  qty integer,  
  price integer,  
  PRIMARY KEY (sno,pno)  
);
```

```
CREATE TABLE Part (  
  pno integer,  
  pname varchar(20),  
  psize integer,  
  pcolor varchar(20),  
  PRIMARY KEY (pno)  
);
```

CSE 444 - Winter 2017

15

## Key Constraint SQL Examples

```
CREATE TABLE Supply(  
  sno integer,  
  pno integer,  
  qty integer,  
  price integer,  
  PRIMARY KEY (sno,pno),  
  FOREIGN KEY (sno) REFERENCES Supplier,  
  FOREIGN KEY (pno) REFERENCES Part  
);
```

```
CREATE TABLE Part (  
  pno integer,  
  pname varchar(20),  
  psize integer,  
  pcolor varchar(20),  
  PRIMARY KEY (pno)  
);
```

CSE 444 - Winter 2017

16

## Key Constraint SQL Examples

```
CREATE TABLE Supply(  
  sno integer,  
  pno integer,  
  qty integer,  
  price integer,  
  PRIMARY KEY (sno,pno),  
  FOREIGN KEY (sno) REFERENCES Supplier  
  ON DELETE NO ACTION,  
  FOREIGN KEY (pno) REFERENCES Part  
  ON DELETE CASCADE  
);
```

```
CREATE TABLE Part (  
  pno integer,  
  pname varchar(20),  
  psize integer,  
  pcolor varchar(20),  
  PRIMARY KEY (pno)  
);
```

CSE 444 - Winter 2017

17

## General Constraints

- Table constraints serve to express complex constraints over a single table

```
CREATE TABLE Part (  
  pno integer,  
  pname varchar(20),  
  psize integer,  
  pcolor varchar(20),  
  PRIMARY KEY (pno),  
  CHECK ( psize > 0 )  
);
```

Note: Also possible to create constraints over many tables  
Best to use database triggers for that purpose

CSE 444 - Winter 2017

18

## Relational Query Languages

CSE 444 - Winter 2017

19

## Relational Query Language

- **Set-at-a-time:**
  - Query inputs and outputs are relations
- Two variants of the query language:
  - Relational algebra: specifies order of operations
  - Relational calculus / SQL: declarative

CSE 444 - Winter 2017

20

## Note

- We will go very quickly in class over the Relational Algebra and SQL
- Please review at home:
  - Read the slides that we skipped in class
  - Review material from 344 as needed

CSE 444 - Winter 2017

21

## Relational Algebra

- **Queries specified in an operational manner**
  - A query gives a step-by-step procedure
- **Relational operators**
  - Take one or two relation instances as argument
  - Return one relation instance as result
  - Easy to compose into **relational algebra expressions**

CSE 444 - Winter 2017

22

## Five Basic Relational Operators

- **Selection:**  $\sigma_{\text{condition}}(S)$ 
  - Condition is Boolean combination ( $\wedge, \vee$ ) of atomic predicates ( $<, <=, =, \neq, >=, >$ )
- **Projection:**  $\pi_{\text{list-of-attributes}}(S)$
- **Union** ( $\cup$ )
- **Set difference** ( $-$ ),
- **Cross-product/cartesian product** ( $\times$ ),  
**Join:**  $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$

Other operators: anti-semijoin, renaming

CSE 444 - Winter 2017

23

```
Supplier(sno, sname, scity, sstate)
Supply(sno, pno, qty, price)
Part(pno, pname, psize, pcolor)
```

## Logical Query Plans

CSE 444 - Winter 2017

24

Supplier (sno, sname, scity, sstate)  
 Supply (sno, pno, qty, price)  
 Part (pno, pname, psize, pcolor)

### Logical Query Plans

```

    graph TD
      S[Supplier] --> J1(( ))
      P[Part] --> J1
      J1 --> J2(( ))
      S --> J2
      S --> S_sel[σ sno=sno]
      P --> P_sel[σ psize > 10]
      J2 --> Proj[π sname, scity]
  
```

What does this query compute?

CSE 444 - Winter 2017 25

### Selection & Projection Examples

**Patient**

no	name	zip	disease
1	p1	98125	flu
2	p2	98125	heart
3	p3	98120	lung
4	p4	98120	heart

$\pi_{zip, disease}(Patient)$

zip	disease
98125	flu
98125	heart
98120	lung
98120	heart

$\sigma_{disease='heart'}(Patient)$

no	name	zip	disease
2	p2	98125	heart
4	p4	98120	heart

$\pi_{zip}(\sigma_{disease='heart'}(Patient))$

zip
98120
98125

CSE 444 - Winter 2017 26

### Cross-Product Example

**AnonPatient P**

age	zip	disease
54	98125	heart
20	98120	flu

**Voters V**

name	age	zip
p1	54	98125
p2	20	98120

$P \times V$

P.age	P.zip	disease	name	V.age	V.zip
54	98125	heart	p1	54	98125
54	98125	heart	p2	20	98120
20	98120	flu	p1	54	98125
20	98120	flu	p2	20	98120

CSE 444 - Winter 2017 27

### Different Types of Join

- Theta-join:**  $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$ 
  - Join of R and S with a join condition  $\theta$
  - Cross-product followed by selection  $\theta$
- Equijoin:**  $R \bowtie_{\theta} S = \pi_A(\sigma_{\theta}(R \times S))$ 
  - Join condition  $\theta$  consists only of equalities
  - Projection  $\pi_A$  drops all redundant attributes
- Natural join:**  $R \bowtie S = \pi_A(\sigma_{\theta}(R \times S))$ 
  - Equijoin
  - Equality on **all** fields with same name in R and in S

CSE 444 - Winter 2017 28

### Theta-Join Example

**AnonPatient P**

age	zip	disease
50	98125	heart
19	98120	flu

**Voters V**

name	age	zip
p1	54	98125
p2	20	98120

$P \bowtie_{P.zip = V.zip \text{ and } P.age \leq V.age + 1 \text{ and } P.age \geq V.age - 1} V$

P.age	P.zip	disease	name	V.age	V.zip
19	98120	flu	p2	20	98120

CSE 444 - Winter 2017 29

### Equijoin Example

**AnonPatient P**

age	zip	disease
54	98125	heart
20	98120	flu

**Voters V**

name	age	zip
p1	54	98125
p2	20	98120

$P \bowtie_{P.age=V.age} V$

age	P.zip	disease	name	V.zip
54	98125	heart	p1	98125
20	98120	flu	p2	98120

CSE 444 - Winter 2017 30

## Natural Join Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$P \bowtie V$

age	zip	disease	name
54	98125	heart	p1
20	98120	flu	p2

CSE 444 - Winter 2017

31

## More Joins

- **Outer join**
  - Include tuples with no matches in the output
  - Use NULL values for missing attributes
- Variants
  - Left outer join
  - Right outer join
  - Full outer join

CSE 444 - Winter 2017

32

## Outer Join Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu
33	98120	lung

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$P \bowtie_J V$

age	zip	disease	name
54	98125	heart	p1
20	98120	flu	p2
33	98120	lung	NULL

CSE 444 - Winter 2017

33

## Example of Algebra Queries

Q1: Names of patients who have heart disease

$\pi_{\text{name}}(\text{Voter} \bowtie (\sigma_{\text{disease}=\text{heart}}(\text{AnonPatient})))$

CSE 444 - Winter 2017

34

## More Examples

### Relations

Supplier(sno, sname, scity, sstate)  
 Part(pno, pname, psize, pcolor)  
 Supply(sno, pno, qty, price)

Q2: Name of supplier of parts with size greater than 10

$\pi_{\text{sname}}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{\text{psize}>10}(\text{Part})))$

Q3: Name of supplier of red parts or parts with size greater than 10

$\pi_{\text{sname}}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{\text{psize}>10}(\text{Part}) \cup \sigma_{\text{pcolor}=\text{red}}(\text{Part})))$

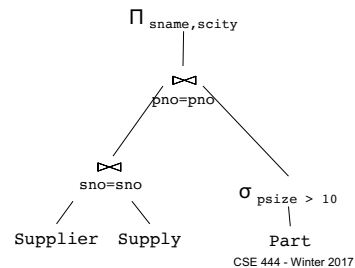
(Many more examples in the book)

CSE 444 - Winter 2017

35

Supplier(sno, sname, scity, sstate)  
 Part(pno, pname, psize, pcolor)  
 Supply(sno, pno, qty, price)

## Logical Query Plans



CSE 444 - Winter 2017

36

## Extended Operators of Relational Algebra

- Duplicate elimination ( $\delta$ )
  - Since commercial DBMSs operate on multisets not sets
- Aggregate operators ( $\gamma$ )
  - Min, max, sum, average, count
- Grouping operators ( $\rho$ )
  - Partitions tuples of a relation into “groups”
  - Aggregates can then be applied to groups
- Sort operator ( $\tau$ )

CSE 444 - Winter 2017

37

## Structured Query Language: SQL

- Declarative query language, based on the relational calculus (see 344)
- Data definition language
  - Statements to create, modify tables and views
- Data manipulation language
  - Statements to issue queries, insert, delete data

CSE 444 - Winter 2017

38

## SQL Query

Basic form: (plus many many more bells and whistles)

```
SELECT <attributes>
FROM <one or more relations>
WHERE <conditions>
```

CSE 444 - Winter 2017

39

```
Supplier(sno, sname, scity, sstate)
Supply(sno, pno, qty, price)
Part(pno, pname, psize, pcolor)
```

## Quick Review of SQL

CSE 444 - Winter 2017

40

```
Supplier(sno, sname, scity, sstate)
Supply(sno, pno, qty, price)
Part(pno, pname, psize, pcolor)
```

## Quick Review of SQL

```
SELECT DISTINCT z.pno, z.pname
FROM Supplier x, Supply y, Part z
WHERE x.sno = y.sno and y.pno = z.pno
and x.scity = 'Seattle' and y.price < 100
```

What does this query compute?

CSE 444 - Winter 2017

41

```
Supplier(sno, sname, scity, sstate)
Supply(sno, pno, qty, price)
Part(pno, pname, psize, pcolor)
```

## Quick Review of SQL

What about this one?

```
SELECT z.pname, count(*) as cnt, min(y.price)
FROM Supplier x, Supply y, Part z
WHERE x.sno = y.sno and y.pno = z.pno
GROUP BY z.pname
```

CSE 444 - Winter 2017

42

## Simple SQL Query

Product	PName	Price	Category	Manufacturer
	Gizmo	\$19.99	Gadgets	GizmoWorks
	Powergizmo	\$29.99	Gadgets	GizmoWorks
	SingleTouch	\$149.99	Photography	Canon
	MultiTouch	\$203.99	Household	Hitachi

```
SELECT *
FROM Product
WHERE category='Gadgets'
```



PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks

"selection"

CSE 444 - Winter 2017

43

## Simple SQL Query

Product	PName	Price	Category	Manufacturer
	Gizmo	\$19.99	Gadgets	GizmoWorks
	Powergizmo	\$29.99	Gadgets	GizmoWorks
	SingleTouch	\$149.99	Photography	Canon
	MultiTouch	\$203.99	Household	Hitachi

```
SELECT PName, Price, Manufacturer
FROM Product
WHERE Price > 100
```



PName	Price	Manufacturer
SingleTouch	\$149.99	Canon
MultiTouch	\$203.99	Hitachi

"selection" and  
"projection"

CSE 444 - Winter 2017

44

## Details

- Case insensitive:
  - Same: SELECT Select select
  - Same: Product product
  - Different: 'Seattle' 'seattle'
- Constants:
  - 'abc' - yes
  - "abc" - no

CSE 444 - Winter 2017

45

## Eliminating Duplicates

```
SELECT DISTINCT category
FROM Product
```

Category
Gadgets
Photography
Household

Compare to:

```
SELECT category
FROM Product
```

Category
Gadgets
Gadgets
Photography
Household

CSE 444 - Winter 2017

46

## Ordering the Results

```
SELECT pname, price, manufacturer
FROM Product
WHERE category='gizmo' AND price > 50
ORDER BY price, pname
```

Ties are broken by the second attribute on the ORDER BY list, etc

Ordering is ascending, unless you specify the DESC keyword.

CSE 444 - Winter 2017

47

## Joins

Product (pname, price, category, manufacturer)  
Company (cname, stockPrice, country)

Find all products under \$200 manufactured in Japan;  
return their names and prices.

```
SELECT PName, Price
FROM Product, Company
WHERE Manufacturer=CName AND Country='Japan'
AND Price <= 200
```

CSE 444 - Winter 2017

48



## Tuple Variables

Person(pname, address, worksfor)  
Company(cname, address)

```
SELECT DISTINCT pname, address
FROM Person, Company
WHERE worksfor = cname
```

Which address ?

```
SELECT DISTINCT Person.pname, Company.address
FROM Person, Company
WHERE Person.worksfor = Company.cname
```

```
SELECT DISTINCT x.pname, y.address
FROM Person AS x, Company AS y
WHERE x.worksfor = y.cname
```

49

## Nested Queries

- **Nested query**
  - Query that has another query embedded within it
  - The embedded query is called a **subquery**
- Why do we need them?
  - Enables to refer to a table that must itself be computed
- Subqueries can appear in
  - **WHERE** clause (common)
  - **FROM** clause (less common)
  - **HAVING** clause (less common)

CSE 444 - Winter 2017

50

## Subqueries Returning Relations

Company(name, city)  
Product(pname, maker)  
Purchase(id, product, buyer)

Return cities where one can find companies that manufacture products bought by Joe Blow

```
SELECT Company.city
FROM Company
WHERE Company.name IN
  (SELECT Product.maker
   FROM Purchase , Product
   WHERE Product.pname=Purchase.product
   AND Purchase .buyer = 'Joe Blow');
```

## Subqueries Returning Relations

You can also use:  $s > ALL R$   
 $s > ANY R$   
**EXISTS R**

Product ( pname, price, category, maker)

Find products that are more expensive than all those produced By "Gizmo-Works"

```
SELECT name
FROM Product
WHERE price > ALL (SELECT price
                   FROM Purchase
                   WHERE maker='Gizmo-Works')
```

## Correlated Queries

Movie (title, year, director, length)

Find movies whose title appears more than once.

```
SELECT DISTINCT title
FROM Movie AS x
WHERE year <> ANY
  (SELECT year
   FROM Movie
   WHERE title = x.title);
```

correlation

Note (1) scope of variables (2) this can still be expressed as single SFW

CSE 444 - Winter 2017

53

## Aggregation

```
SELECT avg(price)
FROM Product
WHERE maker="Toyota"
```

```
SELECT count(*)
FROM Product
WHERE year > 1995
```

SQL supports several aggregation operations:  
sum, count, min, max, avg

Except count, all aggregations apply to a single attribute

CSE 444 - Winter 2017

54

## Grouping and Aggregation

```

SELECT S
FROM R1,...,Rn
WHERE C1
GROUP BY a1,...,ak
HAVING C2
    
```

Conceptual evaluation steps:

1. Evaluate FROM-WHERE, apply condition C1
2. Group by the attributes a<sub>1</sub>,...,a<sub>k</sub>
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

Read more about it in the book...

CSE 444 - Winter 2017

55

## From SQL to RA

CSE 444 - Winter 2017

56

## From SQL to RA

Product(pid, name, price)  
 Purchase(pid, cid, store)  
 Customer(cid, name, city)

```

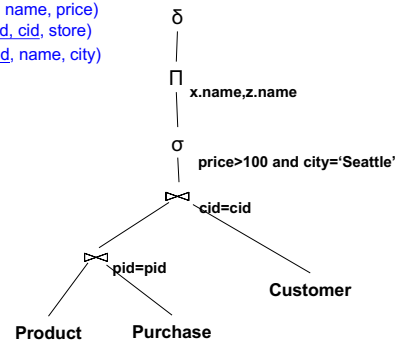
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
      x.price > 100 and z.city = 'Seattle'
    
```

CSE 444 - Winter 2017

57

## From SQL to RA

Product(pid, name, price)  
 Purchase(pid, cid, store)  
 Customer(cid, name, city)

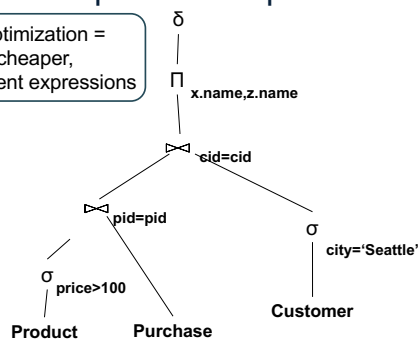


CSE 444 - Winter 2017

58

## An Equivalent Expression

Query optimization =  
 finding cheaper,  
 equivalent expressions



CSE 444 - Winter 2017

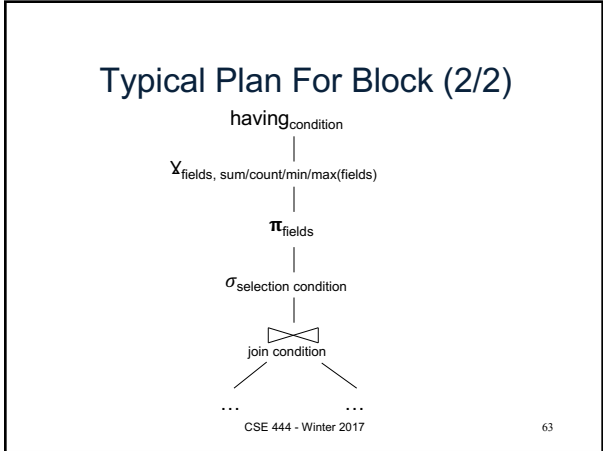
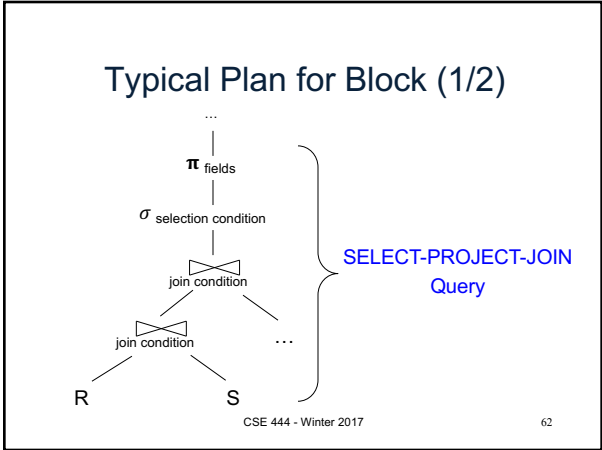
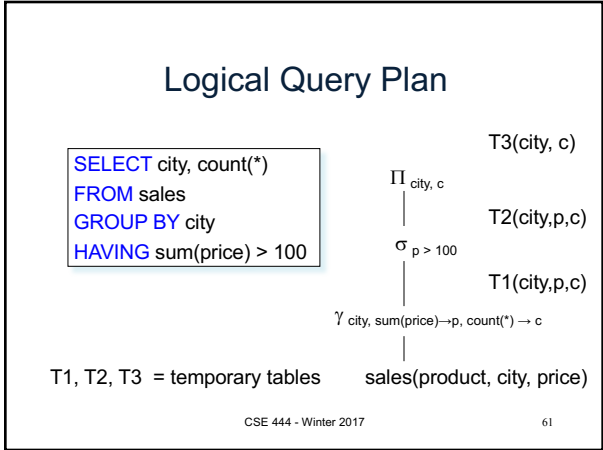
59

## Extended RA: Operators on Bags

- Duplicate elimination  $\delta$
- Grouping  $\gamma$
- Sorting  $\tau$

CSE 444 - Winter 2017

60



- ### Benefits of Relational Model
- **Physical data independence**
    - Can change how data is organized on disk without affecting applications
  - **Logical data independence**
    - Can change the logical schema without affecting applications (not 100%... consider updates)
- CSE 444 - Winter 2017 64

