CSE 444: Database Internals

Lecture 2
Review of the
Relational Model and SQL

Relation Definition

- **Database is collection of relations**
- **Relation R is subset of S_1 x S_2 x … x S_n**
  - Where S_i is the domain of attribute **i**
  - **n** is number of attributes of the relation
- Relation is basically a table with rows & columns
  - SQL uses word table to refer to relations

Properties of a Relation

- Each row represents an n-tuple of R
- Ordering of rows is immaterial
- All rows are distinct
- Ordering of columns is significant
  - Because two columns can have same domain
  - But columns are labeled so
  - Applications need not worry about order
  - They can simply use the names
- Domain of each column is a primitive type
- Relation consists of a **relation schema** and **instance**

More Definitions

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

Even More Definitions

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

Example

- Relation schema
  Supplier(<u>sno: integer</u>, sname: string, scity: string, sstate: string)
- Relation instance

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

Integrity Constraints

- **Integrity constraint**
  - 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

Key Constraints

- **Key constraint:** "certain minimal subset of fields is a unique identifier for a tuple"
- **Candidate key**
  - Minimal set of fields
  - That uniquely identify each tuple in a relation
- **Primary key**
  - One candidate key can be selected as primary key

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

Key Constraint SQL Examples

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


Key Constraint SQL Examples

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


Key Constraint SQL Examples

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

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
);
```


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
);
```


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

Relational Queries

- **Query inputs and outputs are relations**
- Query evaluation
  - Input: instances of input relations
  - Output: instance of output relation

Relational Algebra

- **Query language** associated with relational model
- **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**

Relational Operators

- Selection: $\sigma_{condition}(S)$
  - Condition is Boolean combination ($\wedge$,$\vee$) of terms
  - Term is: attr. op constant, attr. op attr.
  - Op is: <, <=, =, ≠, >=, or >
- Projection: $\pi_{list-of-attributes}(S)$
- Union ($\cup$), Intersection ($\cap$), Set difference (–),
- Cross-product or cartesian product ($\times$)
- Join: $R \bowtie_\theta S = \sigma_\theta(R \times S)$
- Division: R/S, Rename $\rho(R(F),E)$

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

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

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

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

zip
98120
98125

Relational Operators

- Selection: $\sigma_{condition}(S)$
  - Condition is Boolean combination ($\wedge$,$\vee$) of terms
  - Term is: attr. op constant, attr. op attr.
  - Op is: <, <=, =, ≠, >=, or >
- Projection: $\pi_{\text{list-of-attributes}}(S)$
- Union (∪), Intersection (∩), Set difference (–),
- Cross-product or cartesian product (×)
- Join: $R \bowtie_\theta S = \sigma_\theta(R \times S)$
- Division: R/S, Rename ρ(R(F),E)

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⋈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

Different Types of Join

- **Theta-join**: $R \bowtie_\theta S = \sigma_\theta(R \times S)$
  - Join of R and S with a join condition θ
  - Cross-product followed by selection θ
- **Equijoin**: $R \bowtie_\theta S = \pi_A(\sigma_\theta(R \times S))$
  - Join condition θ consists only of equalities
  - Projection π_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

Theta-Join Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$P \bowtie_{P.age=V.age \wedge P.zip=A.zip \wedge P.age < 50} V$

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

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

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

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

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

$P \overset{o}{\bowtie} V$

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

Example of Algebra Queries

Q1: Names of patients who have heart disease

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

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

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

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

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

(Many more examples in the book)

Extended Operators of Relational Algebra

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

Structured Query Language: SQL

- Influenced by relational calculus (see 344)
- Declarative query language
- Multiple aspects of the language
  - Data definition language
    - Statements to create, modify tables and views
  - Data manipulation language
    - Statements to issue queries, insert, delete data
  - More

SQL Query

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

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


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"

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"

Details

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

Eliminating Duplicates

```
SELECT DISTINCT category
FROM   Product
```

Category
Gadgets
Photography
Household

Compare to:

```
SELECT category
FROM   Product
```

Category
Gadgets
Gadgets
Photography
Household

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.

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
```


Tuple Variables

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

Which address ?

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

```
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
```


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)

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 one

correlation

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

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

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

Grouping and Aggregation

SELECT S
FROM $R_1,\ldots,R_n$
WHERE C1
GROUP BY $a_1,\ldots,a_k$
HAVING C2

Conceptual evaluation steps:

1. Evaluate FROM-WHERE, apply condition C1
2. Group by the attributes $a_1,\ldots,a_k$
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...