Introduction to Database Systems

CSE 444

Lecture #2
Jan 8 2001

Enrollment Closed

Announcement: Homework

- HW#1 is being handed out
  - Due **Wed Jan 17**
  - Requires use of SQL Server
- Homework is <u>individual</u> work
  - Even when you are asked to share an account
- No late submission
  - You will lose entire credit
- In the future, we will only announce availability of homework/solutions
  - Download from the website

Announcement: Course Project

- Goal: Build end to end database application with web front-end
- Tasks
  - Find a database application
  - Model the data and define application requirements
  - Design and implement relational schema
  - Populate database
  - Build a web-based front end
- Your application should be nontrivial
  - Sample applications and other details available in course web pages

---

Announcement: Course Project

- Group Project
  - Important: Must work in Team of 3
  - Each member must have well-defined contribution
  - Send <u>yana@cs</u> team information **ASAP**
    - Latest by **Jan 12** by email
- Stages
  - Formation of Group
  - Informal Proposal and ASP Programming
  - Formal design (graded)
  - Project Report (graded)
  - Interview and Demo (graded)
    - March 7,9
- Requires significant design and implementation
  - Start now!
  - Get familiar with software

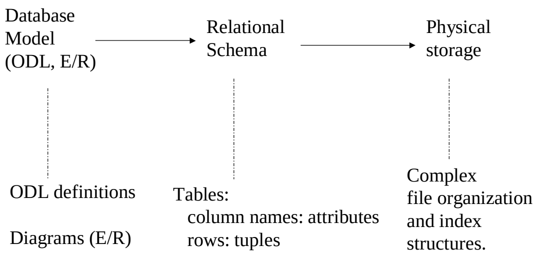
The Relational Data Model

Reading: 3.1, 3.5.1-3.5.3

Data Models

- A *data model* is a collection of concepts for describing data.
- The *relational model of data* is the most widely used model today.
  - Main concept: *relation*, basically a table with rows and columns.
  - Every relation has a *schema*, which describes the columns, or fields.

The Relational Data Model



Database Model (ODL, E/R) → Relational Schema → Physical storage

ODL definitions

Diagrams (E/R)

Tables:
column names: attributes
rows: tuples

Complex file organization and index structures.

Terminology

Table name

Attribute names

Products:

Name	Price	Category	Manufacturer
gizmo	$19.99	gadgets	GizmoWorks
Power gizmo	$29.99	gadgets	GizmoWorks
SingleTouch	$149.99	photography	Canon
MultiTouch	$203.99	household	Hitachi

tuples

Domains

- each attribute has a type
- must be atomic type called *domain*
- examples:
  - Integer
  - String
  - Real
  - …

Schemas

- Relational Schema:
  - Relation name plus attribute names
  - E.g. Product(Name, Price, Category, Manufacturer)
  - In practice we add the domain for each attribute
- Database Schema
  - Set of relational schemas
  - E.g. Product(Name, Price, Category, Manufacturer)
    Vendor(Name, Address, Phone)

Instances

- An instance of a relational schema R(A1,…,Ak), is a relation with k attributes with values of corresponding domains
- An instance of a database schema R1(…), R2(…), …, Rn(…), consists of n relations, each an instance of the corresponding relational schema.

Example

Relational schema: Product(Name, Price, Category, Manufacturer)
Instance:

Name	Price	Category	Manufacturer
gizmo	$19.99	gadgets	GizmoWorks
Power gizmo	$29.99	gadgets	GizmoWorks
SingleTouch	$149.99	photography	Canon
MultiTouch	$203.99	household	Hitachi

Schemas and Instances

- Analogy with programming languages:
  - Schema = type
  - Instance = value
- Important distinction:
  - Database Schema = stable over long periods of time
  - Database Instance = changes constantly, as data is inserted/updated/deleted

---

Integrity Constraints (ICs)

- IC: condition that must be true for *any* instance of the database; e.g., *domain constraints.*
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A *legal* instance of a relation is one that satisfies all specified ICs.
  - DBMS should allow only legal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids many data entry errors, too!

Keys

- Examples:

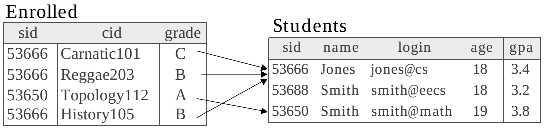
- “For a given student and course, there is a single grade.”
- “No two students have the same sid and no two students have the same login. Furthermore, any other table wishing to reference a student should reference the sid field if possible.”

Enrolled
(*sid* CHAR(20)
cid CHAR(20),
grade CHAR(2))

Students
(*sid* CHAR(20)
login CHAR(10),
gpa REAL, …,)

Foreign Keys

- Only students listed in the Students relation should be allowed to enroll for courses.



Enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Foreign Keys, Referential Integrity

- Foreign key: Set of fields in one relation that is used to `refer’ to a tuple in another relation
  - Must correspond to primary key of the second relation
  - Like a `logical pointer’
- If all foreign key constraints are enforced, referential integrity is achieved
  - No dangling references

Integrity Constraints and Semantics

- ICs are based upon the semantics of the real-world enterprise
- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
- Key and foreign key ICs are the most common; more general ICs supported too.

Relational Operators and Relational Algebra

Reading: 4.1, 4.5-4.8

Set-Oriented Operations: Relational Algebra

- Basic operations:
  - *Selection* ( ) Selects a subset of rows
  - *Projection* ( ) Deletes unwanted columns
  - *Set-difference* ( ) Tuples in reln. 1, but not in reln. 2.
  - *Union* ( ) Tuples in reln. 1 and in reln. 2.
  - *Cross-product* ( ) Allows us to combine two relations
- Since each operation returns a relation, operations can be *composed*! (Algebra is “closed”)

Example

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

- “Sailors” and “Reserves” relations for our examples.
- Assume that names of fields in query results are `inherited’ from names of fields in query input relations.

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

Projection

- Retains only attributes that are in *projection list*.
- *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the input relation.
- Projection operator has to eliminate *duplicates*! (Why??)
  - Note: real systems typically don’t do duplicate elimination unless the user explicitly asks for it. (Why not?)

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$$\pi_{sname,rating}(S2)$$

age
35.0
55.5

$$\pi_{age}(S2)$$

Selection

- Selects rows that satisfy *selection condition.*
- No duplicates in result! (Why?)
- *Schema* of result identical to schema of input relation.
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition.*)

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$$\sigma_{rating>8}(S2)$$

sname	rating
yuppy	9
rusty	10

$$\pi_{sname,rating}(\sigma_{rating>8}(S2))$$

Union, Intersection, Set-Difference

- ⌘All of these operations take two input relations, which must be *union-compatible*:
  - ⌂Same number of fields.
  - ⌂`Corresponding' fields have the same
- ⌘What is the *schema* of result?

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

Cross-Product

- ⌘Each row of S1 is paired with each row of R1.

(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

Joins

- ⌘*Condition Join*: $R \bowtie_c S = \sigma_c (R \times S)$
- ⌘*Equi-Join*: A special case of condition join where the condition *c* contains only ***equalities***

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

$S1 \bowtie_{sid} R1$

- ⌘*Result schema* similar to cross-product, but only one copy of fields for which equality is specified.
- ⌘*Natural Join*: Equijoin on *all* common fields (fields with the same ***name***).

Example of Composition and Equivalence

- ⌘Find names of sailors who've reserved boat #103
- ⌘Solution:

$$\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie Sailors)$$

$$\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie Sailors))$$

Find names of sailors who've reserved a red boat

- ⌘Information about boat color only available in Boats; so need an extra join:

$$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie \text{Reserves} \bowtie Sailors)$$

- ❖ A more efficient solution:

$$\pi_{sname}(\pi_{sid}((\pi_{bid}\sigma_{color='red'} Boats) \bowtie \text{Res}) \bowtie Sailors)$$

SQL

Reading: Sec 5 (all subsections, except 5.10)

Why yet another Language?

- Built-in support for set-oriented retrieval of data from a "large" database.
- Query Languages **!=** programming languages!
  - QLs not expected to be "Turing complete"
  - QLs not intended to be used for complex computation

Basic SQL Query

```
SELECT   [DISTINCT] targetlist
FROM     relation-list
WHERE    qualification
```

- *relation-list* A list of relation names (possibly with a *range-variable* after each name).
- *target-list* A list of attributes of relations in *relation-list*
- *qualification* Comparisons: Attr *op* const or Attr1 *op* Attr2, where *op* is one of $<, >, =, \leq, \geq, \neq$ connected using AND, OR and NOT.
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates.
  - Default is that duplicates are *not* eliminated!

Selections

Company(sticker, name, country, stockPrice)

Find all US companies whose stock is > 50:

```
SELECT   *
FROM     Company
WHERE    country="USA" AND stockPrice > 50
```

Output schema: R(sticker, name, country, stockPrice)

Selections

What you can use in WHERE:

- attribute names of the relation(s) used in the FROM.
- comparison operators: =, <>, <, >, <=, >=
- apply arithmetic operations: stockprice*2
- operations on strings (e.g., "||" for concatenation).
- Lexicographic order on strings.
- Pattern matching: s LIKE p
- Special stuff for comparing dates and times.

The LIKE operator

- s **LIKE** p: pattern matching on strings
- p may contain two special symbols:
  - % = any sequence of characters
  - _ = any single character

Company(sticker, name, address, country, stockPrice)
Find all US companies whose address has prefix "Mountain":

```
SELECT   *
FROM     Company
WHERE    country="USA" AND
         address LIKE "Mountain%"
```

Projections

Select only a subset of the attributes

```
SELECT   name, stockPrice
FROM     Company
WHERE    country="USA" AND stockPrice > 50
```

Input schema: Company(sticker, name, country, stockPrice)
Output schema: R(name, stock price)

Projections

Rename the attributes in the resulting table

```
SELECT   name AS company, stockprice AS price
FROM     Company
WHERE   country=“USA” AND stockPrice > 50
```

Input schema: Company(sticker, name, country, stockPrice)

Output schema: R(company, price)

Ordering the Results

```
SELECT   name, stockPrice
FROM     Company
WHERE   country=“USA” AND stockPrice > 50
ORDERBY  country, name
```

Ordering is ascending, unless you specify the DESC keyword.

Ties are broken by the second attribute on the ORDERBY list, etc.

Removing Duplicates

Product(pid, name, maker, category, price)

```
SELECT   DISTINCT category
FROM      Product
WHERE    price > 100
```

Aggregation

```
SELECT  Sum(price)
FROM     Product
WHERE   maker=“Toyota”
```

SQL supports several aggregation operations:

SUM, MIN, MAX, AVG, COUNT

Aggregation: Count

Except COUNT, all aggregations apply to a single attribute

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

Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

```
SELECT  Count(name, category)         same as Count(*)
FROM     Product
WHERE   year > 1995
```

Better:

```
SELECT  Count(DISTINCT name, category)
FROM     Product
WHERE   year > 1995
```

Simple Aggregation

Purchase(product, date, price, quantity)

Example 1: **find total sales for the entire database**

```
SELECT  Sum(price * quantity)
FROM      Purchase
```

Example 1’: **find total sales of bagels**

```
SELECT  Sum(price * quantity)
FROM      Purchase
WHERE   product = ‘bagel’
```

Grouping and Aggregation

Usually, we want aggregations on certain parts of the relation.

Purchase(product, date, price, quantity)

Example 2: **find total sales after 9/1 per product.**

```
SELECT       product, Sum(price*quantity) AS TotalSales
FROM          Purchase
WHERE       date > “9/1”
GROUPBY  product
```

Grouping and Aggregation

1. Compute the relation (i.e., the FROM and WHERE).
2. Group by the attributes in the GROUPBY
3. Select one tuple for every group (and apply aggregation)

SELECT can have (1) grouped attributes or (2) aggregates.

First compute the relation (date > “9/1”) then group by product:

Product	Date	Price	Quantity
Banana	10/19	0.52	17
Banana	10/22	0.52	7
Bagel	10/20	0.85	20
Bagel	10/21	0.85	15

Then, aggregate

Product	TotalSales
Bagel	$29.75
Banana	$12.48

```
SELECT       product, Sum(price*quantity) AS TotalSales
FROM          Purchase
WHERE       date > “9/1”
GROUPBY  product
```

Example

Product	SumSales	MaxQuantity
Banana	$12.48	17
Bagel	$29.75	20

For every product, what is the total sales and max quantity sold?

```
SELECT       product, Sum(price * quantity) AS SumSales
                         Max(quantity) AS MaxQuantity
FROM          Purchase
GROUP BY product
```

Example

```
SELECT   name,  max(stockPrice)
FROM     Company
WHERE   country=“USA” AND stockPrice > 50
GROUP BY name
HAVING Min(stockprice) > 25
```

- Partition by stockname
- One aggregation per partition
- Number of output tuples = Number of Partitions