Review of Basic Database Concepts

CPS 296.1

Topics in Database Systems

What's a database system?

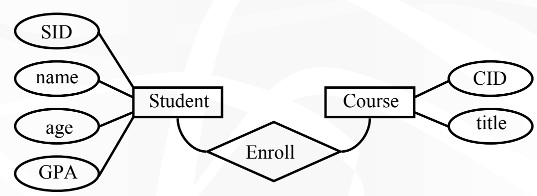
- According to Oxford Dictionary
  - Database: an organized body of related information
  - Database system, DataBase Management System, or DBMS: a software system that facilitates the creation and maintenance and use of an electronic database
- More precisely, a DBMS should support
  - Efficient and convenient querying and updating of large amounts of persistent data
  - Safe, multi-user access

Two important questions

- What is the right API for a DBMS?
  - Data model
    - How is the data structured conceptually?
  - Query language
    - How do users ask queries about the data?
- How does the DBMS support the API?
  - Query processing and optimization
    - What is the most efficient way to answer a query?
  - Transaction processing
    - How are atomicity, consistency, isolation, and durability of transaction ensured?

Entity-relationship (E/R) diagram

- Entities: students and courses
- Relationships: students enroll in courses



- Widely used for database design by humans
- DBMS does not need a graphical data model

Before the relational "revolution"

- Hierarchical and network data models
  - Relationships are modeled as pointers
  - Queries require explicit pointer following
- Example: a simplified CODASYL query

```
Student.GPA := 4.0
FIND Student RECORD BY CALC-KEY
FIND OWNER OF CURRENT Student-Course SET
IF Course.CID = "CPS 296" THEN
   PRINT Student.name
```

Assume that we can quickly find student records by GPA

Assume there is a pointer from students to courses

How about navigating from courses to students?

Physical data independence

- Problems with hierarchical and network data models
  - Access to data is not declarative
  - Whenever data is reorganized, applications must be reprogrammed!
- ➢ Physical data independence
  - Applications should not need to worry about how data is physically structured and stored
  - Applications should work with a logical data model and declarative query language
  - Leave the implementation details and optimization to DBMS

Relational data model

- A database is a collection of relations (or tables)
- Each relation has a list of attributes (or columns)
- Each relation contains a set of tuples (or rows)
  - Duplicates not allowed

Student

SID	name	age	GPA
142	Bart	10	2.3
123	Milhouse	10	3.1
857	Lisa	8	4.3
456	Ralph	8	2.3
...	...	...	...

Course

CID	title
CPS 296	Topics in Database
CPS 216	Advanced Database
CPS 116	Intro. to Database
...	...

Enroll

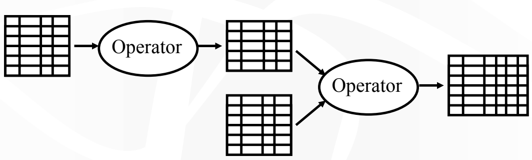
SID	CID
142	CPS 296
142	CPS 216
123	CPS 296
857	CPS 296
857	CPS 116
456	CPS 116
...	...

Schema versus instance

- Schema (metadata)
  - Structure and constraints over data
    - *Student* (*SID* integer, *name* string, *age* integer, *GPA* float)
    - *Course* (*CID* string, *title* string)
    - *Enroll* (*SID* integer, *CID* integer)
    - *Student.SID* is a key, *Enroll.SID* is a foreign key referencing *Student.SID*, etc.
  - Changes infrequently
- Instance
  - Actual contents that conform to the schema
    - { <142, Bart, 10, 2.3>, <123, Milhouse, 10, 3.1>, ...}
    - { <CPS 296, Topics in Database Systems>, ...}
    - { <142, CPS 296>, <142, CPS 216>, ...}
  - Changes frequently

Relational algebra

- Core set of operators:
  - Selection, projection, cross product, union, difference, and renaming
- Additional, derived operators:
  - Join, etc.



Selection

- Notation: $\sigma_p(R)$
  - p is called a selection condition/predicate
- Output: only rows that satisfy p
- Example: Students with GPA higher than 3.0

$\sigma_{GPA > 3.0}(Student)$

SID	name	age	GPA
142	Bart	10	2.3
123	Milhouse	10	3.1
857	Lisa	8	4.3
456	Ralph	8	2.3

→ $\sigma_{GPA > 3.0}$ →

SID	name	age	GPA
[redacted]			
123	Milhouse	10	3.1
857	Lisa	8	4.3
[redacted]			

Projection

- Notation: $\pi_L(R)$
  - L is a list of columns in R
- Output: only the columns in L
  - Duplicate rows are removed
- Example: age distribution of students

$\pi_{age}(Student)$

SID	name	age	GPA
142	Bart	10	2.3
123	Milhouse	10	3.1
857	Lisa	8	4.3
456	Ralph	8	2.3

→ π_{age} →

age
10
[redacted]
8
8

Cross product

- Notation: $R \times S$
- Output: for each row r in R and each row s in S, output a row rs (concatenation of r and s)
- Example: *Student* × *Enroll*

SID	name	age	GPA
142	Bart	10	2.3
123	Milhouse	10	3.1
...	...	...	...

×

SID	CID
142	CPS 296
142	CPS 216
123	CPS 296
...	...

↓

SID	name	age	GPA	SID	CID
142	Bart	10	2.3	142	CPS 296
142	Bart	10	2.3	142	CPS 216
142	Bart	10	2.3	123	CPS 296
123	Milhouse	10	3.1	142	CPS 296
123	Milhouse	10	3.1	142	CPS 216
123	Milhouse	10	3.1	123	CPS 296
...	...	...	...	...	...

Derived operator: join

- Notation: $R \bowtie_p S$ (shorthand for $\sigma_p (R \times S)$)
  - p is called a join condition/predicate
- Example: students and CIDs of their courses
  Student $\bowtie_{Student.SID = Enroll.SID}$ *Enroll*

SID	name	age	GPA
142	Bart	10	2.3
123	Milhouse	10	3.1
…	…	…	…

$\bowtie_{Student.SID = Enroll.SID}$

SID	CID
142	CPS 296
142	CPS 216
123	CPS 296
…	…

SID	name	age	GPA	SID	CID
142	Bart	10	2.3	142	CPS 296
142	Bart	10	2.3	142	CPS 216
[redacted]					
123	Milhouse	10	3.1	123	CPS 296
…	…	…	…	…	…

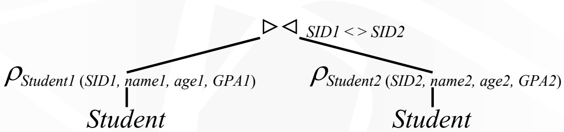
Union and difference

- Notation: $R \cup S$
  - R and S must have identical schema
- Output:
  - Same schema as R and S
  - Contains all rows in R and all rows in S, with duplicates eliminated

- Notation: $R - S$
  - R and S must have identical schema
- Output:
  - Same schema as R and S
  - Contains all rows in R that are not found in S

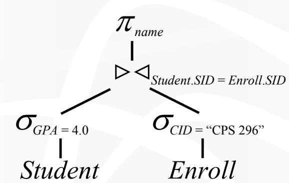
Renaming

- Notation: $\rho_S(R)$, or $\rho_{S(A_1, A_2, \ldots)}(R)$
- Purpose: rename a table and/or its columns
  - No real processing involved
  - Used to avoid confusion caused by identical column names
- Example: all pairs of (different) students



Relational algebra example

- Names of students in CPS 296 with 4.0 GPA



- Compare this query to the CODASYL version!

SQL

- SQL (Structured Query Language)
  - Pronounced "S-Q-L" or "sequel"
  - The query language of every commercial DBMS
- Simplest form: SELECT $A_1, A_2, \ldots, A_n$
  FROM $R_1, R_2, \ldots, R_m$
  WHERE *condition*;
  - Also called an SPJ (select-project-join) query
  - Equivalent (more or less) to relational algebra query
    $\pi_{A_1, A_2, \ldots, A_n} (\sigma_{condition} (R_1 \times R_2 \times \ldots \times R_m))$
  - Unlike relational algebra, SQL preserves duplicates by default

SQL example

- Names of students in CPS 296 with 4.0 GPA

```
SELECT Student.name
FROM Student, Enroll
WHERE Enroll.CID = 'CPS 296'
AND Enroll.SID = Student.SID
AND Student.GPA = 4.0;
```

- Compare this query to the CODASYL version!

More SQL features

SELECT [DISTINCT] *list_of_output_columns*
FROM *list_of_tables*
WHERE *where_condition*
GROUP BY *list_of_group_by_columns*
HAVING *having_condition*
ORDER BY *list_of_order_by_columns*;

Operational semantics

- FROM: take the cross product of *list_of_tables*
- WHERE: apply $\sigma_{where\_condition}$
- GROUP BY: group result tuples according to *list_of_group_by_columns*
- HAVING: apply $\sigma_{having\_condition}$ to the groups
- SELECT: apply $\pi_{list\_of\_output\_columns}$ (preserve duplicates)
- DISTINCT: eliminate duplicates
- ORDER BY: sort the result by *list_of_order_by_columns*

SQL example with aggregation

- Find the average GPA for each age group with at least three students

```
SELECT age, AVG(GPA)
FROM Student
GROUP BY age
HAVING COUNT(*) >= 3;
```

SID	*name*	*age*	*GPA*
142	Bart	10	2.3
857	Lisa	8	4.3
123	Milhouse	10	3.1
456	Ralph	8	2.3
789	Jessica	10	4.2

GROUP BY

SID	*name*	*age*	*GPA*
142	Bart	10	2.3
123	Milhouse	10	3.1
789	Jessica	10	4.2
857	Lisa	8	4.3
456	Ralph	8	2.3

HAVING

SID	*name*	*age*	*GPA*
142	Bart	10	2.3
123	Milhouse	10	3.1
789	Jessica	10	4.2

[redacted]

SELECT

age	*AVG(GPA)*
10	3.2

Summary: relational query languages

- Not your general-purpose programming language
  - Not expected to be Turing-complete
  - Not intended to be used for complex calculations
  - Amenable to much optimization
- More declarative than languages for hierarchical and network data models
  - No explicit pointer following
    - Replaced by joins that can be easily reordered
- Next: How do we support relational query languages efficiently?

Access paths

- Store data in ways to speed up queries
  - Heap file: unordered set of records
  - B+-tree index: disk-based balanced search tree with logarithmic lookup and update
  - Linear/extensible hashing: disk-based hash tables that can grow dynamically
  - Bitmap indexes: potentially much more compact
  - And many more…
- One table may have multiple access paths
  - One primary index that stores records directly
  - Multiple secondary indexes that store pointers to records

Query processing methods

- The same query operator can be implemented in many different ways
- Example: $R \bowtie_{R.A=S.B} S$
  - Nested-loop join: for each tuple of *R*, and for each tuple of *S*, join
  - Index nested-loop join: for each tuple of *R*, use the index on *S.B* to find joining *S* tuples
  - Sort-merge join: sort *R* by *R.A*, sort *S* by *S.B*, and merge-join
  - Hash join: partition *R* and *S* by hashing *R.A* and *S.B*, and join corresponding partitions
  - And many more…

Motivation for query optimization

- The same query can have many different execution plans
- Example:

```
SELECT Student.name
FROM Student, Enroll
WHERE Enroll.CID = 'CPS 296'
AND Enroll.SID = Student.SID
AND Student.GPA = 4.0;
```

  - Plan 1: evaluate $\sigma_{GPA = 4.0}$(*Student*); for each result *SID*, find the *Enroll* tuples with this *SID* and check if *CID* is CPS 296
  - Plan 2: evaluate $\sigma_{CID = \text{'CPS 296'}}$(*Enroll*); for each result *SID*, find the *Student* tuple with this *SID* and check if *GPA* is 4.0
  - Plan 3: evaluate both $\sigma_{GPA = 4.0}$(*Student*) and $\sigma_{CID = \text{'CPS 296'}}$(*Enroll*), and join them on *SID*
  - Any many more…

Query optimization

- A huge number of possible execution plans
  - With different access methods, join order, join methods, etc.
- Query optimizer's job
  - Enumerate candidate plans
    - Query rewrite: transform queries or query plans into equivalent ones
  - Estimate costs of plans
    - Use statistics such as histograms
  - Pick a plan with reasonably low cost
    - Dynamic programming
    - Randomized search

Optimizing for I/O

Location	Cycles	Location	Time
Registers	1	My head	1 min.
Memory	100	Washington D.C.	1.5 hr.
Disk	10^6	Pluto	2 yr.

(source: AlphaSort paper, 1995)

➢ I/O costs dominate database operations
  - DBMS typically optimizes the number of I/O's
- Example: Which of the following is a more efficient way to process SELECT * FROM R ORDER BY R.A;?
  - Use an available secondary B+-tree index on R.A: follow leaf pointers, which are already ordered by R.A
  - Just sort the table