CompSci 516
Database Systems

Lecture 3
Data Model
+
More SQL

Instructor: Sudeepa Roy

Announcements

- If you are enrolled to the class, but
  - have not received the email from Piazza, please email me
  - If you missed Thursday’s lab, please email me
- HW1 will be released this week
- Project ideas will be posted by next week

Today's topic

- Physical and Logical Data Independence
- Data Model and XML
- More SQL
  - Aggregates (Group-by)!
  - Creating/modifying relations/data
  - Constraints

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.

Physical and Logical Data Independence

What does a DBMS provide?

Why use a DBMS?

1. Data Independence
   - Application programs should not be exposed to the data representation and storage
   - DBMS provides an abstract view of the data

2. Efficient Data Access
   - A DBMS utilizes a variety of sophisticated techniques to store and retrieve data (from disk) efficiently

Why use a DBMS?

3. Data Integrity and Security
   - DBMS enforces "integrity constraints" – e.g. check whether total salary is less than the budget
   - DBMS enforces "access controls" – whether salary information can be accesses by a particular user

4. Data Administration
   - Centralized professional data administration by experienced users can manage data access, organize data representation to minimize redundancy, and fine tune the storage

Why use a DBMS?

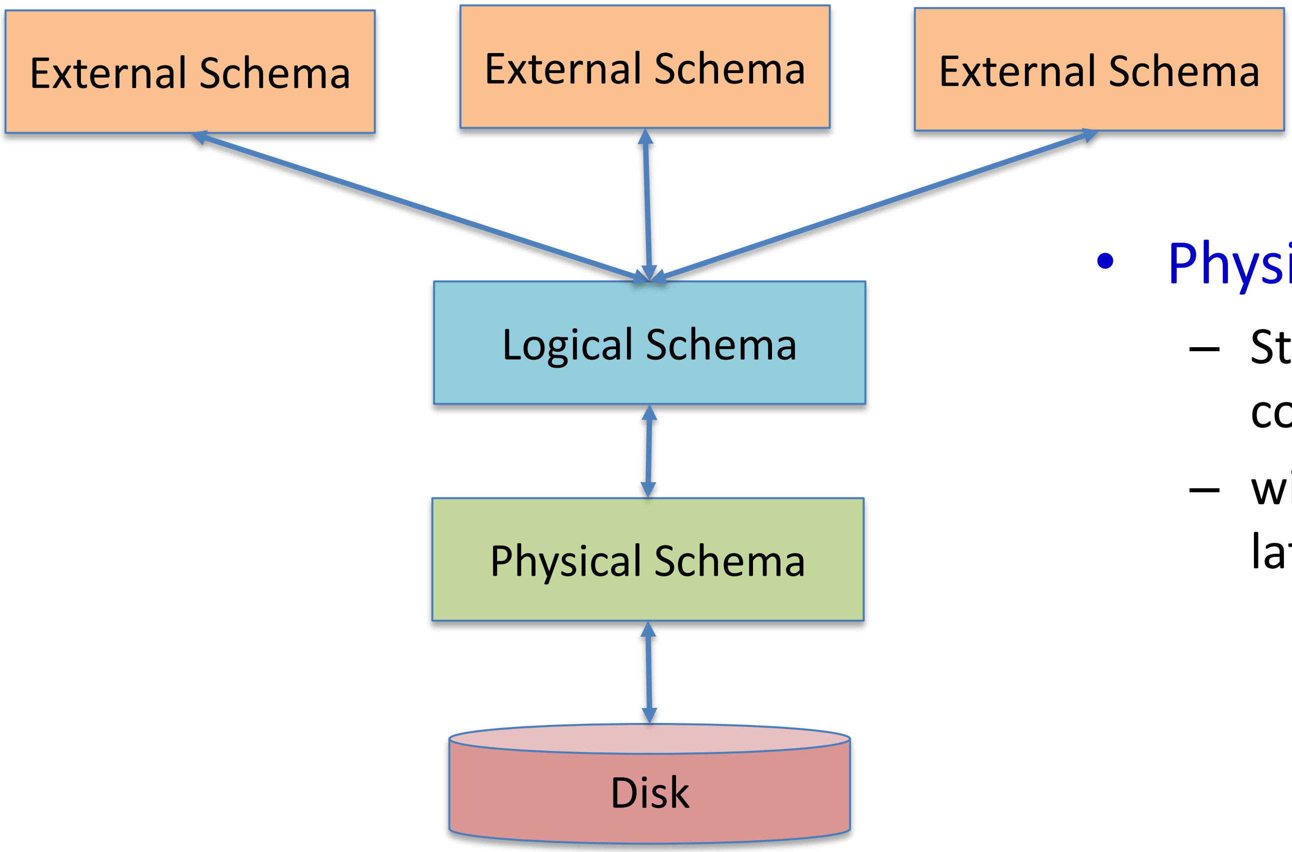
5. Concurrent Access and Crash Recovery
   - DBMS schedules concurrent accesses to the data such that the users think that the data is being accessed by only one user at a time
   - DBMS protects data from system failures

6. Reduced Application Development Time
   - Supports many functions that are common to a number of applications accessing data
   - Provides high-level interface
   - Facilitates quick and robust application development

When NOT to use a DBMS?

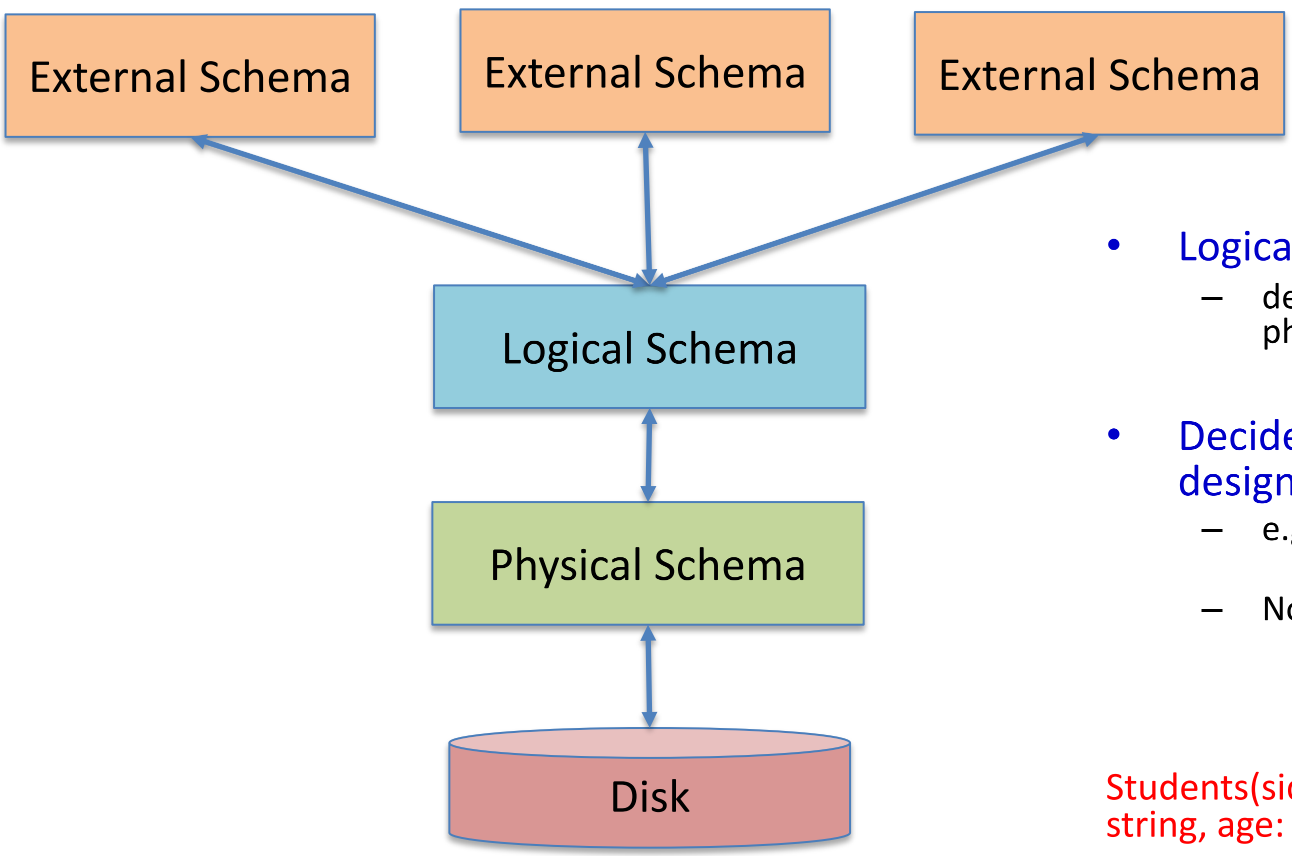
- DBMS is optimized for certain kind of workloads and manipulations
- There may be applications with tight real-time constraints or a few well-defined critical operations
- Abstract view of the data provided by DBMS may not suffice
- To run complex, statistical/ML analytics on large datasets

Levels of Abstractions in a DBMS



- Physical schema
  - Storage as files, row vs. column store, indexes
  - will discuss these in later lectures

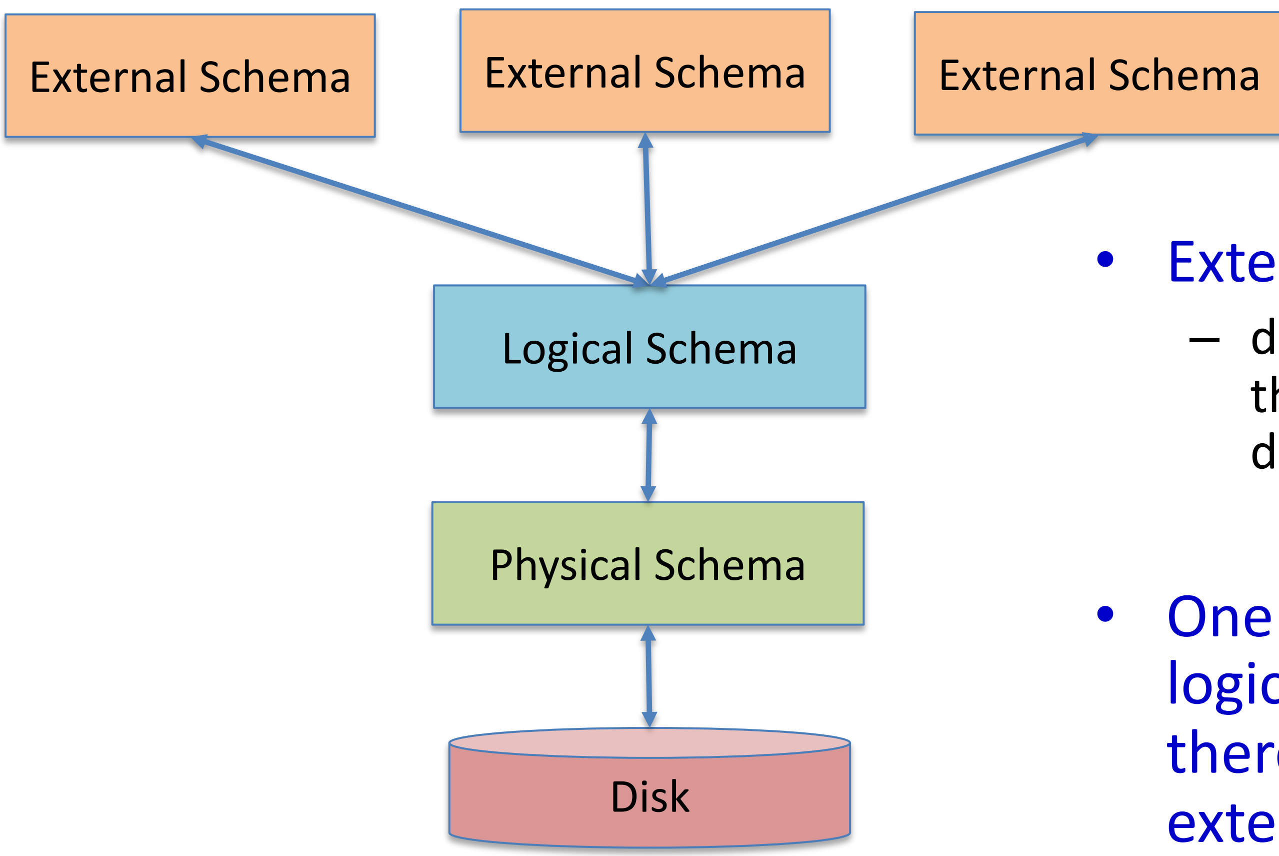
Levels of Abstractions in a DBMS



- Logical/Conceptual schema
  - describes the stored data in the physical schema
- Decided by conceptual schema design
  - e.g. ER Diagram
    - not covered in this course
  - Normalization
    - will be covered

Students(sid: string, name: string, login: string, age: integer, gpa: real)

Levels of Abstractions in a DBMS



- External schema
  - different "views" of the database to different users (later)
- One physical and logical schema but there can be multiple external schemas

Data Independence

- Application programs are insulated from changes in the way the data is structured and stored

- A very important property of a DBMS

- Logical and Physical

Logical Data Independence

- Users can be shielded from changes in the logical structure of data
- e.g. Students:

  Students(sid: string, name: string, login: string, age: integer, gpa: real)
- Divide into two relations

  Students_public(sid: string, name: string, login: string)

  Students_private(sid: string, age: integer, gpa: real)
- Still a "view" Students can be obtained using the above new relations
  - by "joining" them with sid
- A user who queries this view Students will get the same answer as before

Physical Data Independence

- The logical/conceptual schema insulates users from changes in physical storage details
  - how the data is stored on disk
  - the file structure
  - the choice of indexes

- The application remains unaltered
  - But the performance may be affected by such changes

Data Model and XML
(an overview)

Data Model

- Applications need to model some real world units
- Entities:
  - Students, Departments, Courses, Faculty, Organization, Employee, …
- Relationships:
  - Course enrollments by students, Product sales by an organization
- A data model is a collection of high-level data description constructs that hide many low-level storage details

Data Model

Can Specify:

1. Structure of the data
   - like arrays or structs in a programming language
   - but at a higher level (conceptual model)

2. Operations on the data
   - unlike a programming language, not any operation can be performed
   - allow limited sets of queries and modifications
   - a strength, not a weakness!

3. Constraints on the data
   - what the data can be
   - e.g. a movie has exactly one title

Important Data Models

- Structured Data
- Semi-structured Data
- Unstructured Data

What are these?

Important Data Models

- Structured Data
  - All elements have a fixed format
  - Relational Model (table)
- Semi-structured Data
  - Some structure but not fixed
  - Hierarchically nested tagged-elements in tree structure
  - XML
- Unstructured Data
  - No structure
  - text, image, audio, video

Semi-structured Data and XML

- XML: Extensible Markup Language
- Will not be covered in detail in class, but many datasets available to download are in this form
  - You will download the DBLP dataset in XML format and transform into relational form (in HW1!)
- Data does not have a fixed schema
  - "Attributes" are part of the data
  - The data is "self-describing"
  - Tree-structured

XML: Example

Attributes

```
<article mdate="2011-01-11” key="journals/acta/Saxena96">
    <author>Sanjeev Saxena</author>
    <title>Parallel Integer Sorting and Simulation Amongst CRCW
        Models.</title>
    <pages>607-619</pages>
    <year>1996</year>
    <volume>33</volume>
    <journal>Acta Inf.</journal>
    <number>7</number>
    <url>db/journals/acta/acta33.html#Saxena96</url>
    <ee>http://dx.doi.org/10.1007/BF03036466</ee>
</article>
```

Elements

Attribute vs. Elements

- Elements can be repeated and nested
- Attributes are unique and atomic

XML vs. Relational Databases

+ Serves as a model suitable for integration of databases containing similar data with different schemas
    - e.g. try to integrate two student databases: S1(sid, name, gpa) and S2(sid, dept, year)
    - Many "nulls" if done in relational model, very easy in XML
- NULL = A keyword to denote missing or unknown values

+ Flexible – easy to change the schema and data

- Makes query processing more difficult

Which one is easier?

- XML (semi-structured) to relational (structured)

or

- relational (structured) to XML (semi-structured)?

XML to Relational Model

- Problem 1: Repeated attributes

```
<book>
    <author>Ramakrishnan</author>
    <author>Gehrke</author>
    <title>Database Management Systems</title>
    <publisher> McGraw Hill
</book>
```

What is a good relational schema?

XML to Relational Model

- Problem 1: Repeated attributes

```
<book>
    <author>Ramakrishnan</author>
    <author>Gehrke</author>
    <title>Database Management Systems</title>
    <pubisher> McGraw Hill</publisher>
</book>
```

Title	Publisher	Author1	Author2

What if the paper has a single author?

XML to Relational Model

- Problem 1: Repeated attributes

```
<book>
    <author>Garcia-Molina</author>
    <author>Ullman</author>
    <author>Widom</author>
    <title>Database Systems – The Complete Book</title>
    <pubisher>Prentice Hall</publisher>
</book>
```

Does not work

Title	Publisher	Author1	Author2

XML to Relational Model

Book

BookId	Title	Publisher
b1	Database Management Systems	McGraw Hill
b2	Database Systems – The Complete Book	Prentice Hall

BookAuthoredBy

BookId	Author
b1	Ramakrishnan
b1	Gehrke
b2	Garcia-Molina
b2	Ullman
b2	Widom

XML to Relational Model

- Problem 2: Missing attributes

```
<book>
	<author>Ramakrishnan</author>
	<author>Gehrke</author>
	<title>Database Management Systems</title>
	<pubisher> McGraw Hill
	<edition>Third</edition>
</book>
<book>
	<author>Garcia-Molina</author>
	<author>Ullman</author>
	<author>Widom</author>
	<title>Database Systems – The Complete
Book</title>
	<pubisher>Prentice Hall</publisher>
</book>
```

BookId	Title	Publisher	Edition
b1	Database Management Systems	McGraw Hill	Third
b2	Database Systems – The Complete Book	Prentice Hall	null

Summary: Data Models

- Relational data model is the most standard for database managements
  - and is the main focus of this course
- Semi-structured model/XML is also used in practice – you will use them in hw assignments
- Unstructured data (text/photo/video) is unavoidable, but won’t be covered in this class

Back to SQL!

Expressions and Strings

```
SELECT  S.age,   age1=S.age-5,   2*S.age AS age2
FROM  Sailors S
WHERE  S.sname LIKE 'B_%B'
```

- Illustrates use of arithmetic expressions and string pattern matching
- *Find triples (of ages of sailors and two fields defined by expressions) for sailors*
  - *whose names begin and end with B and contain at least three characters*
- LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters
  - You will need these often

Find sid's of sailors who've reserved a red or a green boat

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
  AND (B.color='red' OR B.color='green')
```

- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples
  - can themselves be the result of SQL queries
- If we replace OR by AND in the first version, what do we get?
- Also available: EXCEPT (What do we get if we replace UNION by EXCEPT?)

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
            AND B.color='red'
UNION
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
            AND B.color='green'
```

Find sid's of sailors who've reserved a red <u>and</u> a green boat

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

Find sid's of sailors who've reserved a red and a green boat

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

```
SELECT  S.sid
FROM  Sailors S, Boats B1, Reserves R1,
          Boats B2, Reserves R2
WHERE  S.sid=R1.sid AND R1.bid=B1.bid
  AND  S.sid=R2.sid AND R2.bid=B2.bid
  AND (B1.color='red' AND B2.color='green')
```

- INTERSECT: Can be used to compute the intersection of any two union-compatible sets of tuples.
  - Included in the SQL/92 standard, but some systems don't support it

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
          AND B.color='red'
INTERSECT
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
          AND B.color='green'
```

Nested Queries

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

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN  (SELECT  R.sid
                  FROM  Reserves R
                  WHERE  R.bid=103)
```

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

- A very powerful feature of SQL:
  - a WHERE/FROM/HAVING clause can itself contain an SQL query
- To find sailors who've not reserved #103, use NOT IN.
- To understand semantics of nested queries, think of a nested loops evaluation
  - For each Sailors tuple, check the qualification by computing the subquery

Nested Queries with Correlation

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

```
SELECT  S.sname
FROM  Sailors S
WHERE   EXISTS  (SELECT  *
                 FROM  Reserves R
                 WHERE  R.bid=103 AND S.sid=R.sid)
```

- EXISTS is another set comparison operator, like IN
- Illustrates why, in general, subquery must be re-computed for each Sailors tuple

Nested Queries with Correlation

Find names of sailors who've reserved boat #103 at most once:

```
SELECT  S.sname
FROM  Sailors S
WHERE   UNIQUE  (SELECT  R.bid
                 FROM  Reserves R
                 WHERE  R.bid=103 AND S.sid=R.sid)
```

- If UNIQUE is used, and * is replaced by *R.bid*, finds sailors with at most one reservation for boat #103
  - UNIQUE checks for duplicate tuples

More on Set-Comparison Operators

- We've already seen IN, EXISTS and UNIQUE
- Can also use NOT IN, NOT EXISTS and NOT UNIQUE.
- Also available: *op* ANY, *op* ALL, *op* IN
  - where op : >, <, =, <=, >=
- Find sailors whose rating is greater than that of some sailor called Horatio
  - similarly ALL

```
SELECT  *
FROM  Sailors S
WHERE  S.rating > ANY  (SELECT  S2.rating
                        FROM  Sailors S2
                        WHERE S2.sname='Horatio')
```

Recall: Aggregate Operators

Check yourself:
What do these queries compute?

COUNT (*)
COUNT ([DISTINCT] A)
SUM ([DISTINCT] A)
AVG ([DISTINCT] A)
MAX (A)
MIN (A)

single column

```
SELECT  COUNT (*)
FROM  Sailors S
```

```
SELECT  AVG (S.age)
FROM  Sailors S
WHERE  S.rating=10
```

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.rating= (SELECT  MAX(S2.rating)
                  FROM  Sailors S2)
```

```
SELECT  COUNT (DISTINCT S.rating)
FROM  Sailors S
WHERE S.sname='Bob'
```

```
SELECT  AVG ( DISTINCT S.age)
FROM  Sailors S
WHERE  S.rating=10
```

Motivation for Grouping

- So far, we've applied aggregate operators to all (qualifying) tuples
  - Sometimes, we want to apply them to each of several groups of tuples
- Consider: Find the age of the youngest sailor for each rating level
  - In general, we don't know how many rating levels exist, and what the rating values for these levels are!
  - Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this (need to replace i by num):

For *i* = 1, 2, ... , 10:

```
SELECT  MIN (S.age)
FROM  Sailors S
WHERE  S.rating = i
```

First go over the examples in the following slides
Then come back to this slide and study yourself

Queries With GROUP BY and HAVING

```
SELECT     [DISTINCT] target-list
FROM       relation-list
WHERE      qualification
GROUP BY   grouping-list
HAVING     group-qualification
```

- The target-list contains
  - (i) attribute names
  - (ii) terms with aggregate operations (e.g., MIN (S.age))
- The attribute list (i) must be a subset of grouping-list
  - Intuitively, each answer tuple corresponds to a group, and these attributes must have a single value per group
  - Here a group is a set of tuples that have the same value for all attributes in grouping-list

First go over the examples in the following slides
Then come back to this slide and study yourself

Conceptual Evaluation

- The cross-product of relation-list is computed
- Tuples that fail qualification are discarded
- `Unnecessary' fields are deleted
- The remaining tuples are partitioned into groups by the value of attributes in grouping-list
- The group-qualification is then applied to eliminate some groups
- Expressions in group-qualification must have a single value per group
  - In effect, an attribute in group-qualification that is not an argument of an aggregate op also appears in grouping-list
  - like "…GROUP BY bid, sid HAVING bid = 3"
- One answer tuple is generated per qualifying group

Find age of the youngest sailor with age >= 18, for each rating with at least 2 such sailors.

```
SELECT  S.rating,  MIN (S.age) AS minage
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1
```

Sailors instance:

sid	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

Answer relation:

rating	minage
3	25.5
7	35.0
8	25.5

Find age of the youngest sailor with age >= 18, for each rating with at least 2 <u>such</u> sailors.

Step 1: Form the cross product: FROM clause
(some attributes are omitted for simplicity)

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

SELECT S.rating, MIN (S.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1

Find age of the youngest sailor with age >= 18, for each rating with at least 2 <u>such</u> sailors.

Step 2: Apply WHERE clause

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
~~10~~	~~16.0~~
9	35.0
3	25.5
3	63.5
3	25.5

SELECT S.rating, MIN (S.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1

Find age of the youngest sailor with age >= 18, for each rating with at least 2 such sailors.

```
SELECT  S.rating,  MIN (S.age) AS minage
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1
```

Step 3: Apply GROUP BY according to the listed attributes

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

→

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

→

rating	age
1	33.0
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0

Find age of the youngest sailor with age >= 18, for each rating with at least 2 <u>such</u> sailors.

Step 4: Apply HAVING clause

The *group-qualification* is applied to eliminate some groups

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

rating	age
~~1~~	~~33.0~~
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
~~9~~	~~35.0~~
~~10~~	~~35.0~~

```
SELECT  S.rating,  MIN (S.age) AS minage
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1
```

Find age of the youngest sailor with age >= 18, for each rating with at least 2 such sailors.

Step 5: Apply SELECT clause

Apply the aggregate operator

At the end, one tuple per group

```
SELECT S.rating, MIN (S.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1
```

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

rating	age
~~1~~	~~33.0~~
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
~~9~~	~~35.0~~
~~10~~	~~35.0~~

rating	minage
3	25.5
7	35.0
8	25.5

Nulls and Views in SQL

Null Values

- Field values in a tuple are sometimes
  - unknown, e.g., a rating has not been assigned, or
  - inapplicable, e.g., no spouse's name
  - SQL provides a special value null for such situations.

Standard Boolean 2-valued logic

- True = 1, False = 0
- Suppose X = 5
  - (X < 100) AND (X >= 1) is T ∧ T = T
  - (X > 100) OR (X >= 1) is F ∨ T = T
  - (X > 100) AND (X >= 1) is F ∧ T = F
  - NOT(X = 5) is ¬T = F

- Intuitively,
  - T = 1, F = 0
  - For V1, V2 ∈ {1, 0}
  - V1 ∧ V2 = MIN (V1, V2)
  - V1 ∨ V2 = MAX(V1, V2)
  - ¬(V1) = 1 – V1

2-valued logic does not work for nulls

- Suppose rating = null, X = 5
- Is rating>8 true or false?
- What about AND, OR and NOT connectives?
  – (rating > 8) AND (X = 5)?
- What if we have such a condition in the WHERE clause?

End of Lecture 3

3-Valued Logic For Null

- TRUE (= 1), FALSE (= 0), UNKNOWN (= 0.5)
  - unknown is treated as 0.5
- Now you can apply rules from 2-valued logic!
  - For V1, V2 ∈ {1, 0, 0.5}
  - V1 ∧ V2 = MIN (V1, V2)
  - V1 ∨ V2 = MAX(V1, V2)
  - ¬(V1) = 1 – V1
- Therefore,
  - NOT UNKNOWN = UNKNOWN
  - UNKNOWN OR TRUE = TRUE
  - UNKNOWN AND TRUE = UNKNOWN
  - UNKNOWN AND FALSE = FALSE
  - UNKNOWN OR FALSE = UNKNOWN