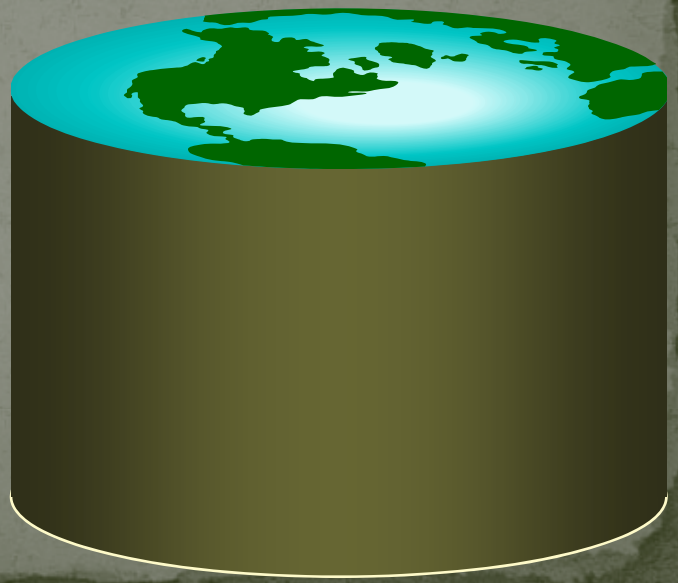
CS 505: Intermediate Topics to Database Systems

Instructor: Jinze Liu

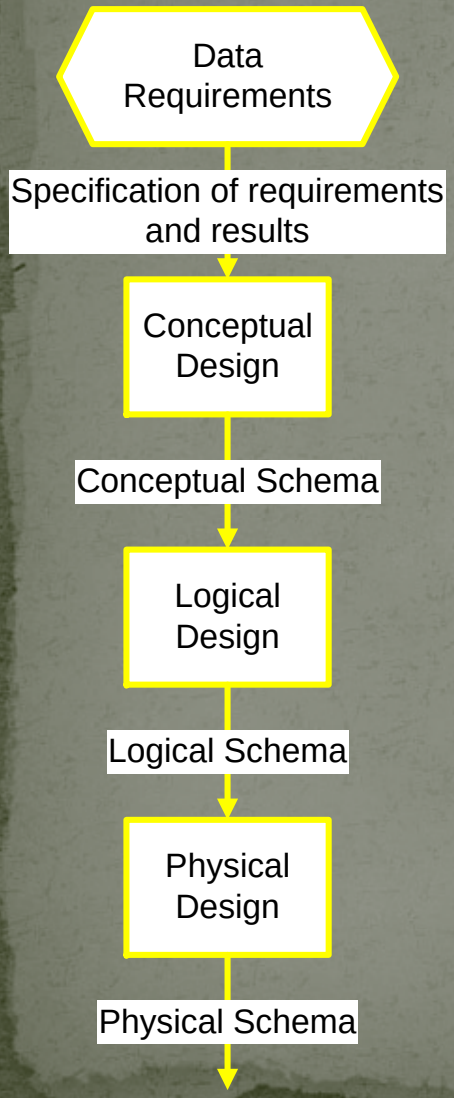
Fall 2008



Project (2 + 2 + 2)

- Basic Components (2)
  - Relational Database
  - Web-Interface
  - Done before mid-term
- Must-Have Components (2)
  - Security: access control
    - Done before mid-term
  - Transaction management
    - Optional
- Optional Components (2)
  - Data replication
  - Indexing
  - XML as the middle-ware between database and web interface
  - Informational retrieval
  - Mining method

Phases of Database Design



- Conceptual design begins with the collection of requirements and results needed from the database (ER Diag.)
- Logical schema is a description of the structure of the database (Relational, Network, etc.)
- Physical schema is a description of the implementation (programs, tables, dictionaries, catalogs

Why do we need core operator *X*?

- Cross product
  - The only operator that adds columns
- Difference
  - The only non-monotone operator
- Union
  - The only operator that allows you to add rows?
- Selection? Projection?

Why is "-" needed for highest GPA?

- Composition of monotone operators produces a monotone query
  - Old output rows remain "correct" when more rows are added to the input
- Highest-GPA query is non-monotone
  - Current highest GPA is 4.1
  - Add another GPA 4.2
  - Old answer is invalidated

☞ So it must use difference!

How to compute the highest GPA

sid	name	age	gpa
1234	John Smith	21	3.5
1123	Mary Carter	22	3.8
1011	Bob Lee	22	2.6

Why is r.a. a good query language?

- Simple
  - A small set of core operators who semantics are easy to grasp
- Declarative?
  - Yes, compared with older languages like CODASYL
  - Though operators do look somewhat “procedural”
- Complete?
  - With respect to what?

Database Design

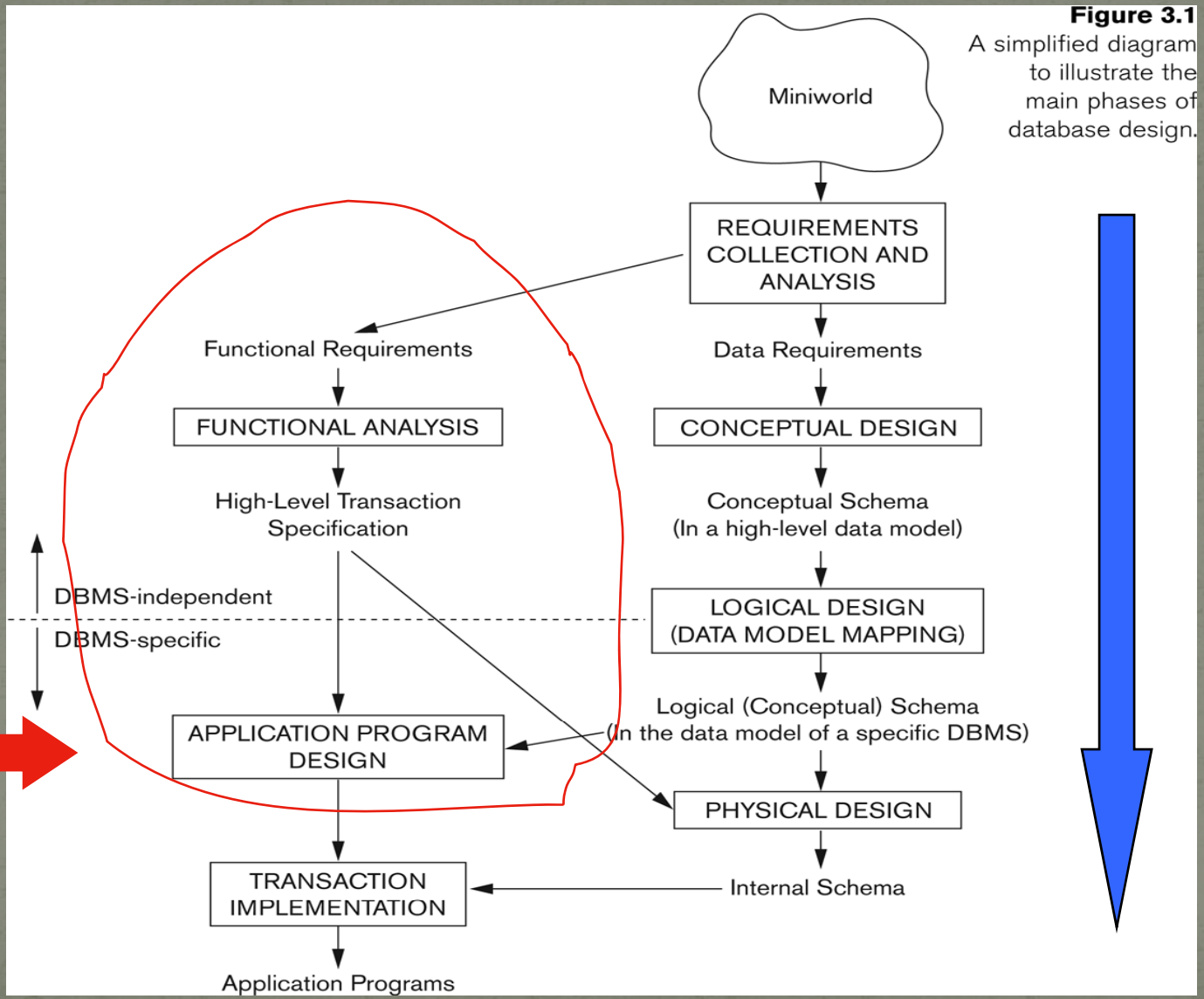


Figure 3.1
A simplified diagram to illustrate the main phases of database design.

Exercises of R. A.

Reserves

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

Boats

bid	bname	color
101	Interlake	Blue
102	Interlake	Red
103	Clipper	Green
104	Marine	Red

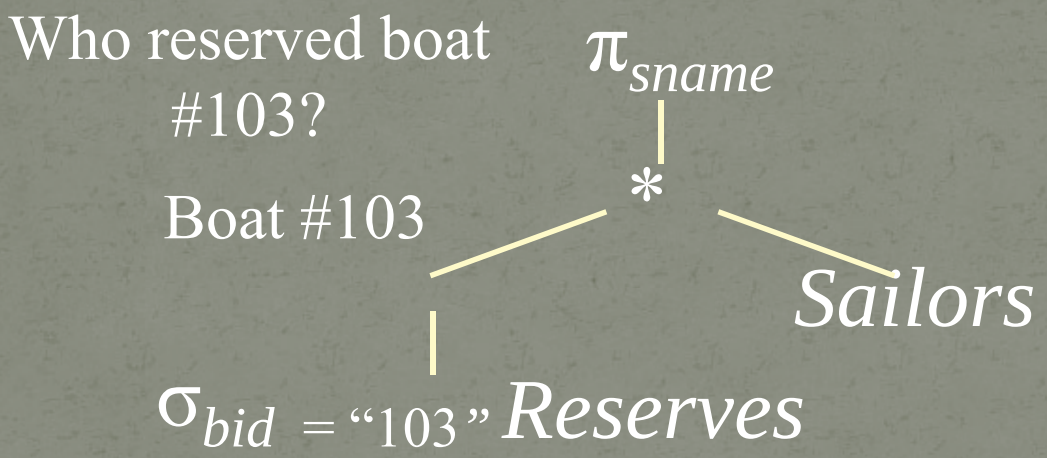
Sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Problem 1: Find names of sailors who've reserved boat #103

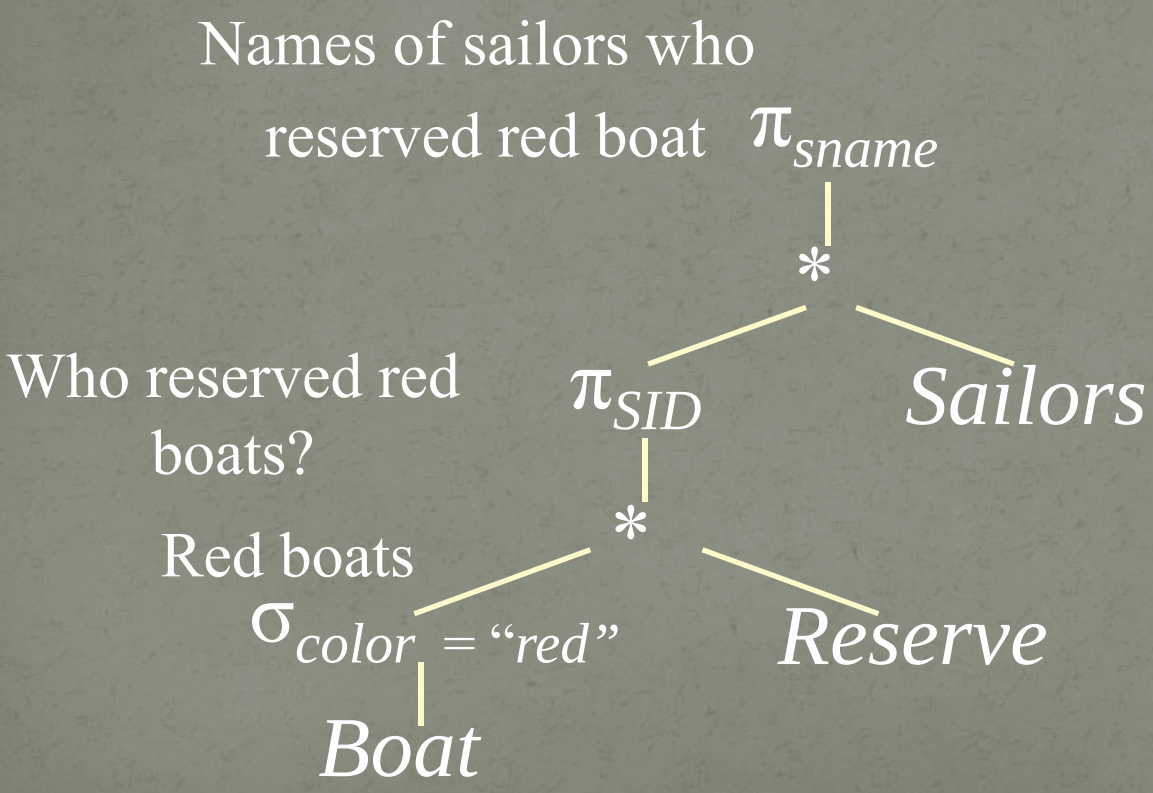
- Solution:

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



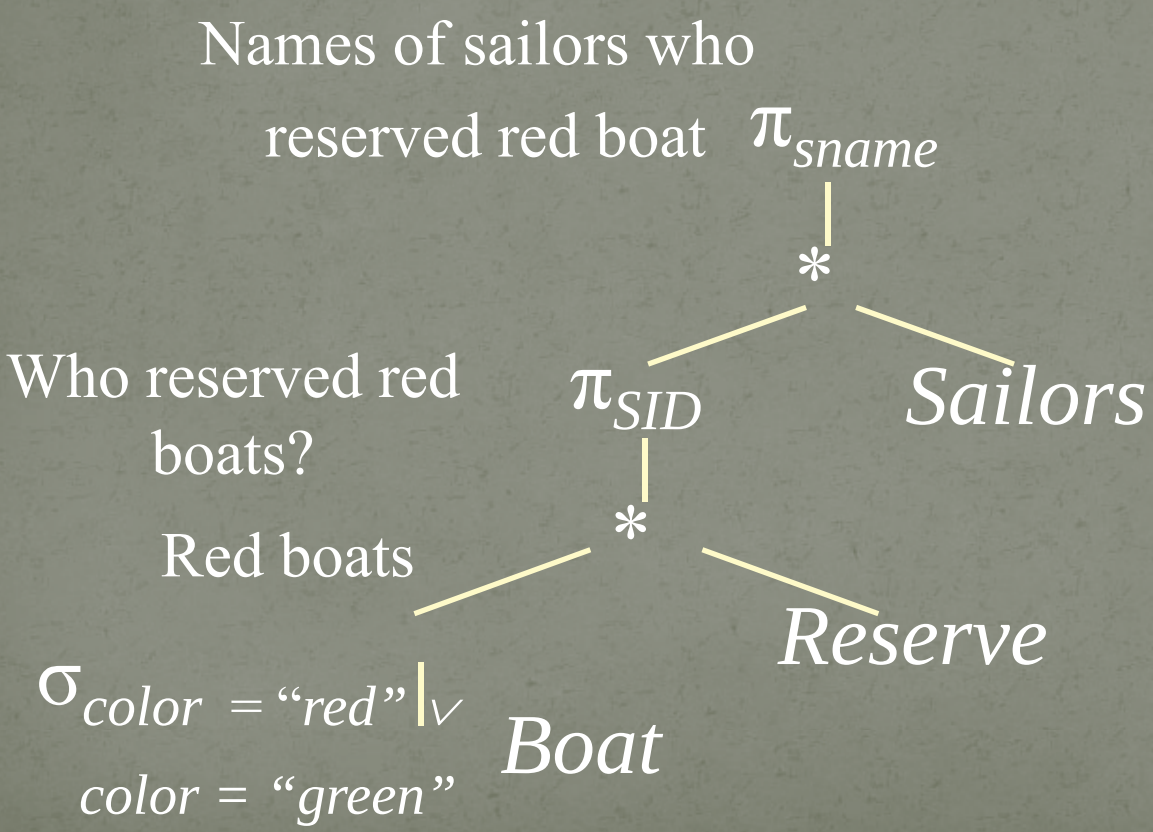
Problem 2: Find names of sailors who've reserved a red boat

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



Problem 3: Find names of sailors who've reserved a red boat or a green boat

- Can identify all red or green boats, then find sailors who've reserved one of these boats:



Problem 4: Find names of sailors who've reserved only one boat

Summary of SQL features

- `SELECT-FROM-WHERE` statements
- Ordering
- Set and bag operations
- Aggregation and grouping
- Table expressions, subqueries

More: `NULL`'s, outerjoins, data modification, constraints, …

Relational Database

```
Student (SID integer, name varchar(30),
  age integer, GPA float);
Course (CID char(10), title varchar(100));
Enroll (SID integer, CID char(10));

Create table Student (SID integer, name
  varchar(30), age integer, GPA float);
Create table Course (CID char(10), title
  varchar(100));
Create table Enroll (SID integer, CID
  char(10));
```

S-F-W

- `SELECT` *a list of attributes*
  `FROM` *a list of relations*
  `WHERE` *condition*`;`
- Condition may have logical operators AND, OR, NOT
- Condition may have comparison operators: <. <=, <>, >=, >
- String comparison may use "=" (exactly match) or "LIKE" (matching with regular expressions)
  - %, _, \
- (Arithmetic) expressions of attributes are allowed

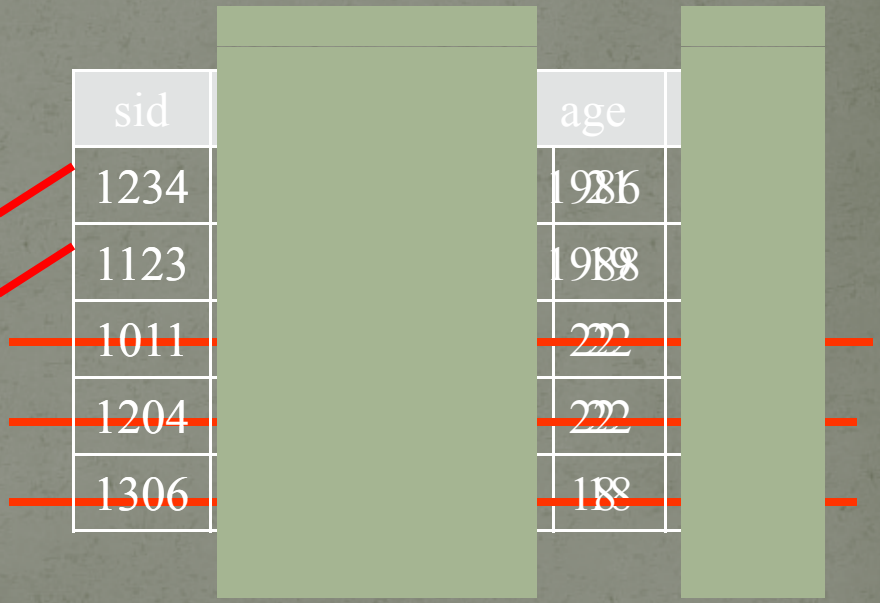
ORDER BY

- `SELECT …`
  `FROM … WHERE ORDER BY` *output_column* `[ASC | DESC], …;`
- `ASC` = ascending, `DESC` = descending
- Operational semantics
  - After `SELECT` list has been computed, sort the output according to `ORDER BY` specification

Exercise

- `SELECT` *sid, 2007 – age,*
  `FROM` *STUDENT*
  `WHERE` *name* LIKE *'%John%' OR GPA > 3.6;*

sid	name	age	gpa
1234	John Smith	21	3.5
1123	Mary Carter	19	3.8
1011	Bob Lee	22	2.6
1204	Susan Wong	22	3.4
1306	Kevin Kim	18	2.9



sid	age
1234	1986
1123	1988
1011	2007
1204	2007
1306	1989

asdfsadf

Next

- Set and Bag operation
  - UNION,
  - INTERSECTION,
  - EXCEPT
  - DISTINCT
- Aggregation
  - HAVING
- Nested queries

```
SELECT age, AVG(GPA)
FROM Student S ENROLL E
WHERE S.SID = E.SID
      AND E.CID = 'EECS108'
GROUP BY age
HAVING age > 20;
--Compute the average GPA for
Students who are at least 20 years
old and are enrolled in 108 with the
same age
```

Forcing set semantics

- SQL provides the option of set semantics with `DISTINCT` keyword
- SID's of all enrolled students

```
SELECT SID,
FROM Enroll,
```

- Say Bart takes CS700 and CS505

```
SELECT DISTINCT SID,
FROM Enroll,
```

  - With `DISTINCT`, all duplicate SIDs are removed from the output

Operational Semantics of SFW

- `SELECT [DISTINCT]` E_1, E_2, …, E_n
  `FROM` R_1, R_2, …, R_m
  `WHERE` *condition*;
- For each t_1 in R_1:
  For each t_2 in R_2: … …
  For each t_m in R_m:
  If *condition* is true over t_1, t_2, …, t_m:
  Compute and output E_1, E_2, …, E_n as a row
  If `DISTINCT` is present
  Eliminate duplicate rows in output
- t_1, t_2, …, t_m are often called tuple variables

SQL set and bag operations

- `UNION`, `EXCEPT`, `INTERSECT`
  - Set semantics
    - Duplicates in input tables, if any, are first eliminated
  - Exactly like set union, - and intersect in relational algebra
- `UNION ALL`, `EXCEPT ALL`, `INTERSECT ALL`
  - Bag semantics
  - Think of each row as having an implicit count (the number of times it appears in the table)
  - Bag union: sum up the counts from two tables
  - Bag difference: subtract the two counts (a row with negative count vanishes)
  - Bag intersection: take the minimum of the two counts

Examples of bag operations

Bag1

fruit
Apple
Apple
Orange

Bag2

fruit
Apple
Orange
Orange

Bag1 UNION ALL Bag2

fruit
Apple
Apple
Apple
Orange
Orange
Orange

Bag1 INTERSECT ALL Bag2

fruit
Apple
Orange

Bag1 EXCEPT ALL Bag2

fruit
Apple

Exercise

- *Enroll*(*SID*, *CID*), *ClubMember*(*club*, *SID*)
  - `(SELECT SID FROM ClubMember)`
    `EXCEPT`
    `(SELECT SID FROM Enroll);`
    - SID's of students who are in clubs but not taking any classes
  - `(SELECT SID FROM ClubMember)`
    `EXCEPT ALL`
    `(SELECT SID FROM Enroll);`
    - SID's of students who are in more clubs than classes

Aggregates

- Standard SQL aggregate functions: `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`
- Example: number of students under 18, and their average GPA
  - ```
    SELECT COUNT(*), AVG(GPA)
    FROM Student
    WHERE age < 18;
    ```
  - `COUNT(*)` counts the number of rows
    ```

Aggregates with `DISTINCT`

- Example: How many students are taking classes?
  - `SELECT COUNT (SID)`
    `FROM Enroll;`
  - `SELECT COUNT(DISTINCT SID)`
    `FROM Enroll;`

GROUP BY

- `SELECT … FROM … WHERE …`
  `GROUP BY` *list_of_columns*`;`

- Example: find the average GPA for each age group
  - ```
    SELECT age, AVG(GPA)
    FROM Student
    GROUP BY age;
    ```
    ```

Operational semantics of `GROUP BY`

```
SELECT … FROM … WHERE … GROUP BY …;
```

- Compute `FROM`
- Compute `WHERE`
- Compute `GROUP BY`: group rows according to the values of `GROUP BY` columns
- Compute `SELECT` for each group
  - For aggregation functions with `DISTINCT` inputs, first eliminate duplicates within the group
  - ☞ Number of groups = number of rows in the final output

Example of computing GROUP BY

```
SELECT age, AVG(GPA) FROM Student GROUP BY age;
```

sid	name	age	gpa
1234	John Smith	21	3.5
1123	Mary Carter	19	3.8
1011	Bob Lee	22	2.6
1204	Susan Wong	22	3.4
1306	Kevin Kim	19	2.9

Compute GROUP BY: group rows according to the values of GROUP BY columns

sid	name	age	gpa
1234	John Smith	21	3.5
1123	Mary Carter	19	3.8
1306	Kevin Kim	19	2.9
1011	Bob Lee	22	2.6
1204	Susan Wong	22	3.4

Compute SELECT for each group

age	gpa
21	3.5
19	3.35
22	3.0

Aggregates with no GROUP BY

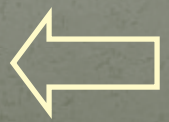
- An aggregate query with no GROUP BY clause represent a special case where all rows go into one group

Compute aggregate over the group

```
SELECT AVG(GPA) FROM Student;
```

sid	name	age	gpa
1234	John Smith	21	3.5
1123	Mary Carter	19	3.8
1011	Bob Lee	22	2.6
1204	Susan Wong	22	3.4
1306	Kevin Kim	19	2.9

sid	name	age	gpa
1234	John Smith	21	3.5
1123	Mary Carter	19	3.8
1011	Bob Lee	22	2.6
1204	Susan Wong	22	3.4
1306	Kevin Kim	19	2.9



gpa
3.24

Group all rows into one group

Restriction on SELECT

- If a query uses aggregation/group by, then every column referenced in SELECT must be either
  - Aggregated, or
  - A GROUP BY column

☞ This restriction ensures that any SELECT expression produces only *one* value for each group

Examples of invalid queries

- `SELECT SID, age FROM Student GROUP BY age;`
  - Recall there is one output row per group
  - There can be multiple SID values per group
- `SELECT SID, MAX(GPA) FROM Student;`
  - Recall there is only one group for an aggregate query with no `GROUP BY` clause
  - There can be multiple SID values
  - Wishful thinking (that the output SID value is the one associated with the highest GPA) does NOT work

HAVING

- Used to filter groups based on the group properties (e.g., aggregate values, `GROUP BY` column values)
- `SELECT … FROM … WHERE … GROUP BY … HAVING` *`condition`*`;`
  - Compute `FROM`
  - Compute `WHERE`
  - Compute `GROUP BY`: group rows according to the values of `GROUP BY` columns
  - Compute `HAVING` (another selection over the groups)
  - Compute `SELECT` for each group that passes `HAVING`

HAVING examples

- Find the average GPA for each age group over 10
  - ```
    SELECT age, AVG(GPA)
    FROM Student
    GROUP BY age
    HAVING age > 10;
    ```
  - Can be written using WHERE without table expressions
- List the average GPA for each age group with more than a hundred students
  - ```
    SELECT age, AVG(GPA)
    FROM Student
    GROUP BY age
    HAVING COUNT(*) > 100;
    ```
  - Can be written using WHERE and table expressions

Table expression

- Use query result as a table
  - In set and bag operations, `FROM` clauses, etc.
  - A way to "nest" queries
- Example: names of students who are in more clubs than classes

```
SELECT DISTINCT name
FROM Student,
     (SELECT SID FROM ClubMember)
     EXCEPT ALL
     (SELECT SID FROM Enroll) ) AS S
WHERE Student.SID = S.SID;
```

Scalar subqueries

- A query that returns a single row can be used as a value in `WHERE`, `SELECT`, etc.
- Example: students at the same age as Bart

```
SELECT *
FROM Student
WHERE age = (SELECT age
             FROM Student
             WHERE name = 'Bart');
```

What's Bart's age?

  - Runtime error if subquery returns more than one row
    - Under what condition will this runtime error never occur?
      - *name* is a key of *Student*
  - What if subquery returns no rows?
    - The value returned is a special `NULL` value, and the comparison fails

IN subqueries

- *x* `IN (`*subquery*`)` checks if *x* is in the result of *subquery*
- Example: students at the same age as (some) Bart

```
SELECT *
FROM Student
WHERE age IN (
```

What's Bart's age?

```
SELECT age
FROM Student
WHERE name = 'Bart'
                    );
```

EXISTS subqueries

- EXISTS (*subquery*) checks if the result of *subquery* is non-empty
- Example: students at the same age as (some) Bart
  - ```
    SELECT *
    FROM Student AS s
    WHERE EXISTS (SELECT * FROM Student
                  WHERE name = 'Bart'
                  AND age = s.age);
    ```
  - This happens to be a correlated subquery—a subquery that references tuple variables in surrounding queries
    ```

Operational semantics of subqueries

```
SELECT *
FROM Student AS s
WHERE EXISTS (SELECT * FROM Student
              WHERE name = 'Bart'
              AND age = s.age);
```

- For each row `s` in `Student`
  - Evaluate the subquery with the appropriate value of `s.age`
  - If the result of the subquery is not empty, output `s.*`
- The DBMS query optimizer may choose to process the query in an equivalent, but more efficient way (example?)