# SQL

CPS 216
Advanced Database Systems

# SQL

- SQL: Structured Query Language
  - Pronounced “S-Q-L” or “sequel”
  - The query language of every commercial DBMS
- A brief history
  - System R
  - SQL89
  - SQL92 (SQL2)
  - SQL3 (still under construction)

# Table creation

- CREATE TABLE *table_name*
  (…, $column\_name_i$ $column\_type_i$, …);
- Example
  - create table Student (SID integer,
    name varchar(30), email varchar(30),
    age integer, GPA float);
  - create table Course (CID char(10),
    title varchar(100));
  - create table Enroll
    (SID integer, CID char(10));

SQL is case insensitive

# SFW queries

- 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

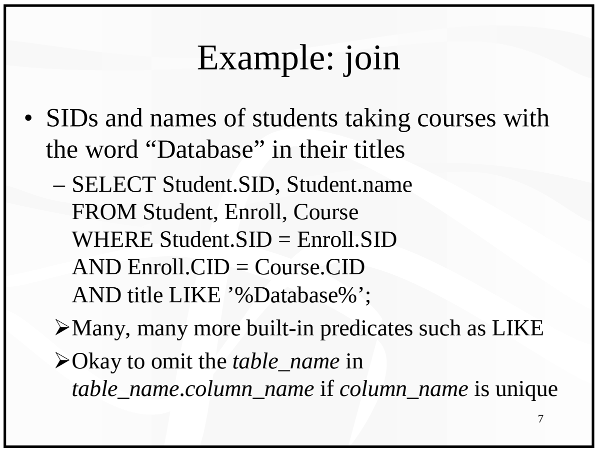
$$\pi_{A_1, A_2, \ldots, A_n} (\sigma_{condition} (R_1 \times R_2 \times \ldots \times R_m))$$

# Example: reading a table

- SELECT * FROM Student;
  - Single-table query; no cross product
  - WHERE clause is optional
  - “*” is a shorthand for “all columns”

# Example: selection and projection

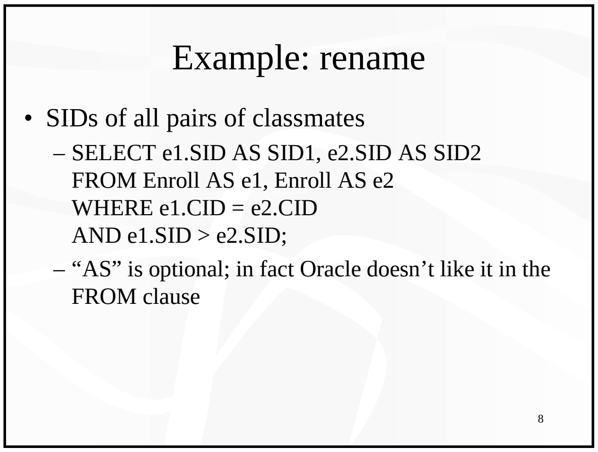
- Names of students under 18
  - SELECT name FROM Student WHERE age < 18;
- When was Lisa born?
  - SELECT 2001 – age
    FROM Student
    WHERE name = ’Lisa’;
  - ➢SELECT list can contain calculations
  - ➢String literals are enclosed in single quotes (case sensitive)



## Example: join

- SIDs and names of students taking courses with the word “Database” in their titles

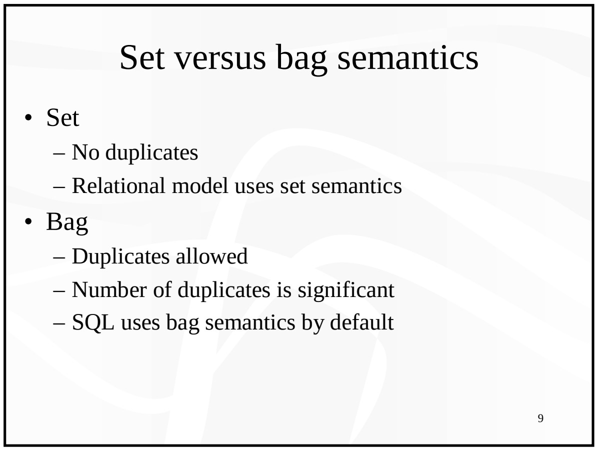
  -
    ```
    SELECT Student.SID, Student.name
    FROM Student, Enroll, Course
    WHERE Student.SID = Enroll.SID
    AND Enroll.CID = Course.CID
    AND title LIKE ’%Database%’;
    ```
  - ➢Many, many more built-in predicates such as LIKE
  - ➢Okay to omit the *table_name* in *table_name.column_name* if *column_name* is unique



## Example: rename

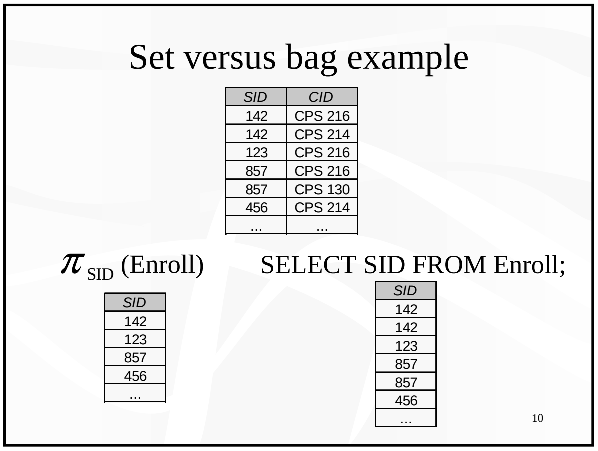
- SIDs of all pairs of classmates

  -
    ```
    SELECT e1.SID AS SID1, e2.SID AS SID2
    FROM Enroll AS e1, Enroll AS e2
    WHERE e1.CID = e2.CID
    AND e1.SID > e2.SID;
    ```
  - “AS” is optional; in fact Oracle doesn’t like it in the FROM clause



## Set versus bag semantics

- Set
  - No duplicates
  - Relational model uses set semantics
- Bag
  - Duplicates allowed
  - Number of duplicates is significant
  - SQL uses bag semantics by default



## Set versus bag example

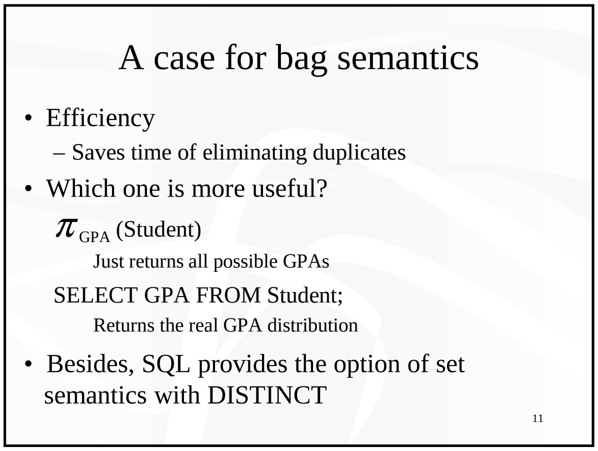
| *SID* | *CID* |
|---|---|
| 142 | CPS 216 |
| 142 | CPS 214 |
| 123 | CPS 216 |
| 857 | CPS 216 |
| 857 | CPS 130 |
| 456 | CPS 214 |
| ... | ... |

$\pi_{\text{SID}}$ (Enroll)

| *SID* |
|---|
| 142 |
| 123 |
| 857 |
| 456 |
| ... |

SELECT SID FROM Enroll;

| *SID* |
|---|
| 142 |
| 142 |
| 123 |
| 857 |
| 857 |
| 456 |
| ... |



## A case for bag semantics

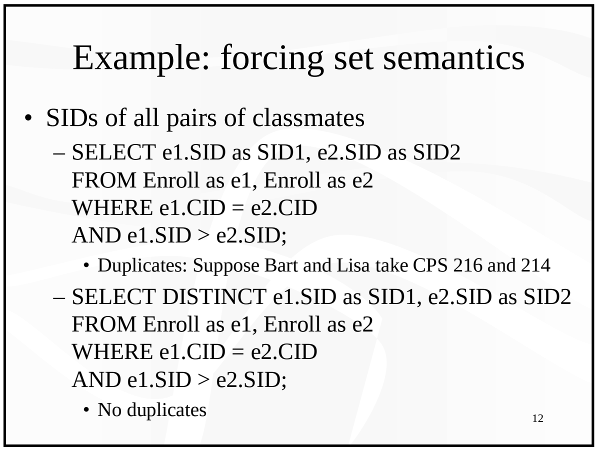
- Efficiency
  - Saves time of eliminating duplicates
- Which one is more useful?

  $\pi_{\text{GPA}}$ (Student)

  Just returns all possible GPAs

  SELECT GPA FROM Student;

  Returns the real GPA distribution
- Besides, SQL provides the option of set semantics with DISTINCT



## Example: forcing set semantics

- SIDs of all pairs of classmates

  -
    ```
    SELECT e1.SID as SID1, e2.SID as SID2
    FROM Enroll as e1, Enroll as e2
    WHERE e1.CID = e2.CID
    AND e1.SID > e2.SID;
    ```
    - Duplicates: Suppose Bart and Lisa take CPS 216 and 214

  -
    ```
    SELECT DISTINCT e1.SID as SID1, e2.SID as SID2
    FROM Enroll as e1, Enroll as e2
    WHERE e1.CID = e2.CID
    AND e1.SID > e2.SID;
    ```
    - No duplicates

## 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 $t_1$ 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 $E_1, E_2, …, E_n$
If DISTINCT is present
    Eliminate duplicates in output
```

## Set and bag operations

- UNION, EXCEPT, INTERSECT
  - Set semantics
  - Exactly like set ∪, −, ∩ in relational algebra
- UNION ALL, EXCEPT ALL, INTERSECT ALL
  - Bag semantics
  - Bag union: sum the two counts (the times an element appears in the two bags)
  - Bag difference: proper-subtract the two counts
  - Bag intersection: take the minimum of the two counts

## Examples of bag operations

*R*

| *A* |
|---|
| apple |
| apple |
| orange |

*S*

| *A* |
|---|
| apple |
| orange |
| orange |

*R* UNION ALL *S*

| *A* |
|---|
| apple |
| apple |
| apple |
| orange |
| orange |
| orange |

*R* EXCEPT ALL *S*

| *A* |
|---|
| apple |

*R* INTERSECT ALL *S*

| *A* |
|---|
| apple |
| orange |

## Example of set versus bag operations

Enroll(SID, CID), ClubMember(club, SID)

- (SELECT SID FROM ClubMember)
  EXCEPT
  (SELECT SID FROM Enroll)

  SIDs of students who are in clubs but not taking any classes
- (SELECT SID FROM ClubMember)
  EXCEPT ALL
  (SELECT SID FROM Enroll)

  SIDs of students who are in more clubs than classes

## Table expressions

- 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 class

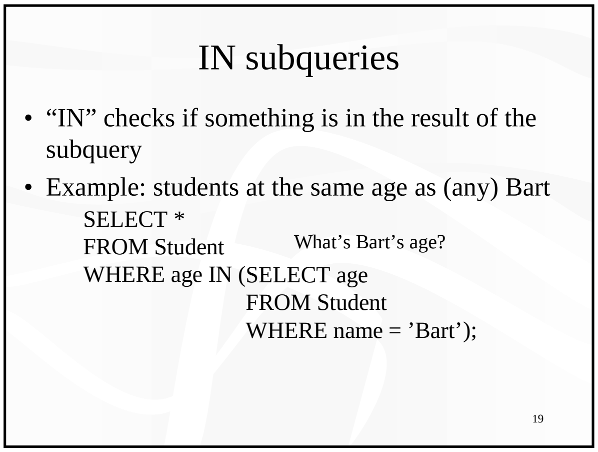
```
SELECT DISTINCT name
FROM Student,
     ( (SELECT SID FROM ClubMember)
       EXCEPT ALL
       (SELECT SID FROM Enroll) ) AS S
WHERE Student.SID = S.SID;
```

## 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          What's Bart's age?
WHERE age = (SELECT age
             FROM Student
             WHERE name = 'Bart');
```

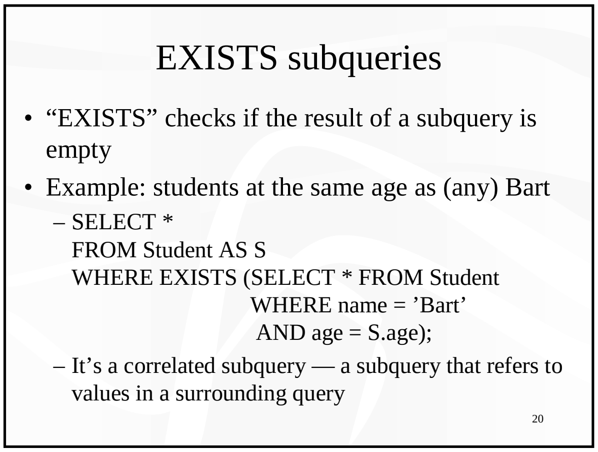
- Runtime error if subquery returns more than one row



## IN subqueries

- “IN” checks if something is in the result of the subquery
- Example: students at the same age as (any) Bart

```
SELECT *
FROM Student          What’s Bart’s age?
WHERE age IN (SELECT age
              FROM Student
              WHERE name = ’Bart’);
```

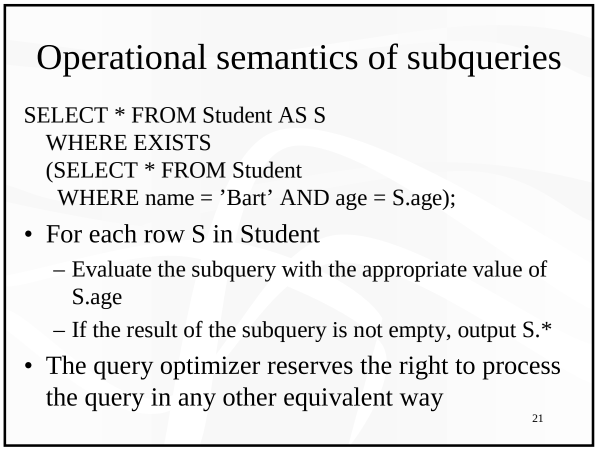


## EXISTS subqueries

- “EXISTS” checks if the result of a subquery is empty
- Example: students at the same age as (any) Bart
  - SELECT *

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

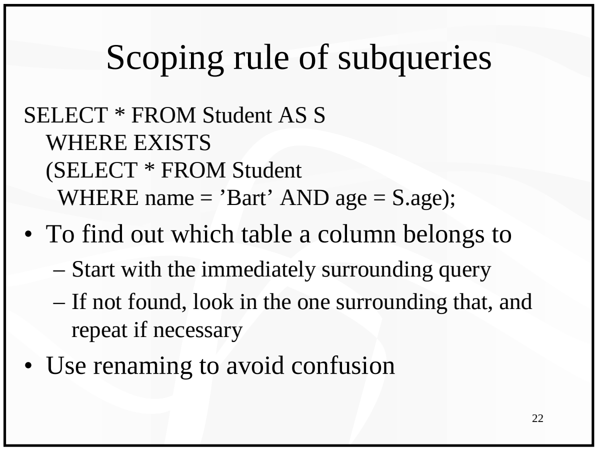
  - It’s a correlated subquery — a subquery that refers to values in a surrounding query



## Operational semantics of subqueries

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

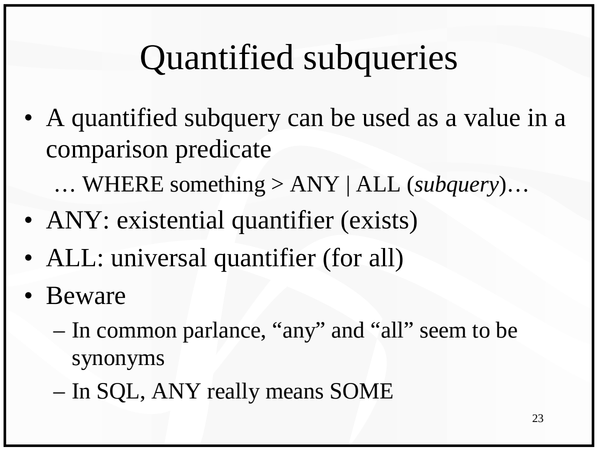
- For each row S in Student
  - Evaluate the subquery with the appropriate value of S.age
  - If the result of the subquery is not empty, output S.*
- The query optimizer reserves the right to process the query in any other equivalent way



## Scoping rule of subqueries

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

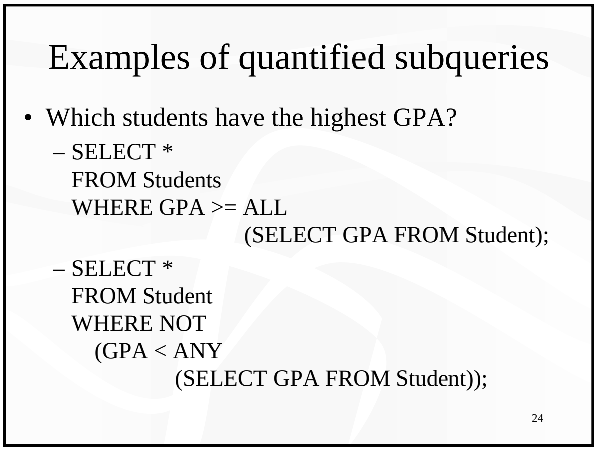
- To find out which table a column belongs to
  - Start with the immediately surrounding query
  - If not found, look in the one surrounding that, and repeat if necessary
- Use renaming to avoid confusion



## Quantified subqueries

- A quantified subquery can be used as a value in a comparison predicate

  … WHERE something > ANY | ALL (*subquery*)…
- ANY: existential quantifier (exists)
- ALL: universal quantifier (for all)
- Beware
  - In common parlance, “any” and “all” seem to be synonyms
  - In SQL, ANY really means SOME



## Examples of quantified subqueries

- Which students have the highest GPA?
  - SELECT *

```
FROM Students
WHERE GPA >= ALL
               (SELECT GPA FROM Student);
```

  - SELECT *

```
FROM Student
WHERE NOT
  (GPA < ANY
         (SELECT GPA FROM Student));
```

## Summary

- Bag semantics
  - Richer semantics, greater efficiency, but just not “relational”
- SELECT-FROM-WHERE
  - A canonical form for queries with any nesting of selection, projection, and join
  - Most queries are in this form
- Subqueries
  - More declarative (recall the highest GPA query)
  - But no more expressive
    - Try translating other forms of subqueries into (NOT) EXISTS, which in turn can be translated into join (and difference)

## Aggregates

- 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(DISTINCT SID)
    FROM Enroll;
  - SELECT COUNT(*)
    FROM (SELECT DISTINCT SID,
    FROM Enroll);

## GROUP BY

- SELECT … FROM … WHERE …
  GROUP BY *list_of_columns*;
- Operational semantics
  - Compute FROM ($\times$)
  - Compute WHERE ($\sigma$)
  - Compute GROUP BY: group results according to the values of GROUP BY columns
  - Compute SELECT for each group ($\pi$)
  - ➢Number of groups = number of rows in the output

## GROUP BY example

- Find the average GPA for each age group
  - SELECT age, AVG(GPA)
    FROM Student
    GROUP BY age;

## GROUP BY example with data

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

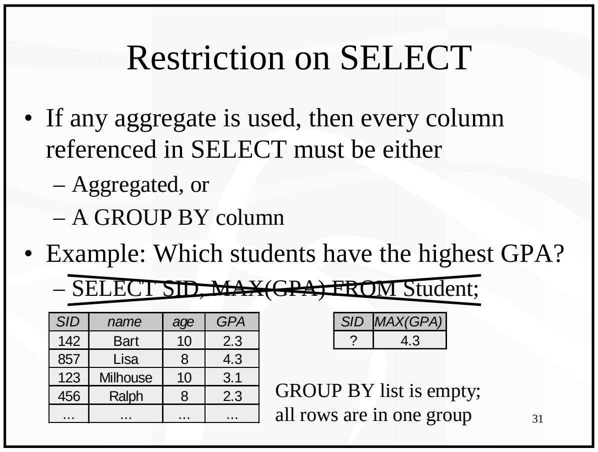
| *SID* | *name* | *age* | *GPA* |
|---|---|---|---|
| 142 | Bart | 10 | 2.3 |
| 857 | Lisa | 8 | 4.3 |
| 123 | Milhouse | 10 | 3.1 |
| 456 | Ralph | 8 | 2.3 |
| … | … | … | … |

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

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

Compute SELECT for each group

| *age* | *AVG(GPA)* |
|---|---|
| 10 | 2.7 |
| 8 | 3.3 |
| … | … |



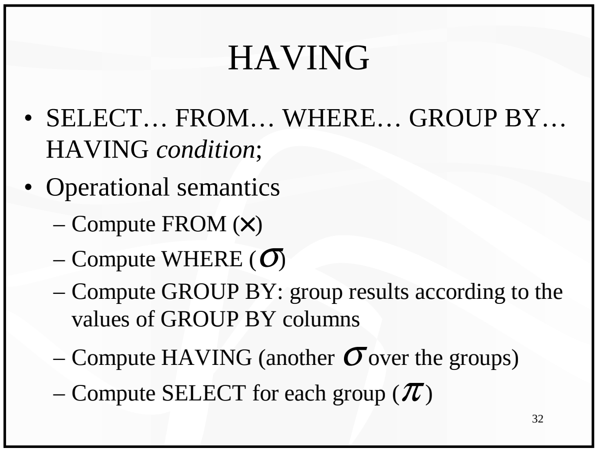
## Restriction on SELECT

- If any aggregate is used, then every column referenced in SELECT must be either
  - Aggregated, or
  - A GROUP BY column
- Example: Which students have the highest GPA?
  - ~~SELECT SID, MAX(GPA) FROM Student;~~

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

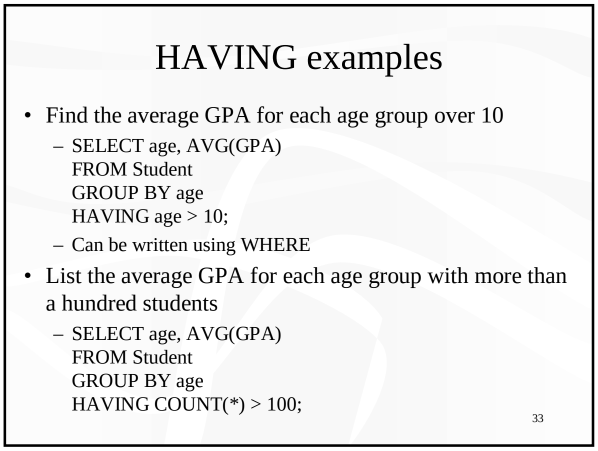
| SID | MAX(GPA) |
|---|---|
| ? | 4.3 |

GROUP BY list is empty;
all rows are in one group



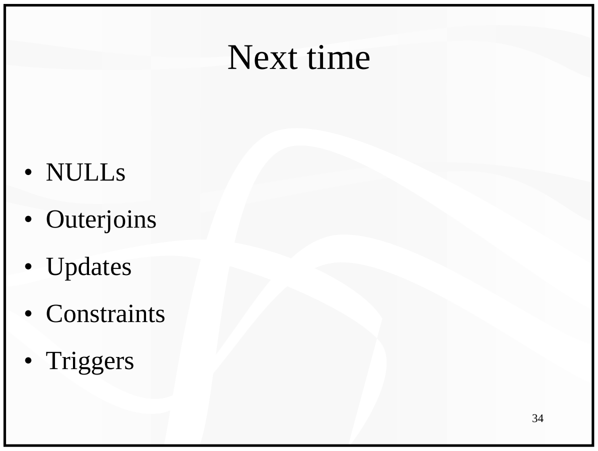
## HAVING

- SELECT… FROM… WHERE… GROUP BY… HAVING *condition*;
- Operational semantics
  - Compute FROM (×)
  - Compute WHERE ($\sigma$)
  - Compute GROUP BY: group results according to the values of GROUP BY columns
  - Compute HAVING (another $\sigma$ over the groups)
  - Compute SELECT for each group ($\pi$)



## 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
- 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;



## Next time

- NULLs
- Outerjoins
- Updates
- Constraints
- Triggers