Aggregates

- Functions that operate on sets:
  - COUNT, SUM, AVG, MAX, MIN
- Produce numbers (not tables)
- Not part of relational algebra

```
SELECT COUNT(*)
FROM  Professor P
```

```
SELECT MAX (Salary)
FROM  Employee E
```

42

Aggregates

Count the number of courses taught in S2000

```
SELECT COUNT (T.CrsCode)
FROM Teaching T
WHERE  T.Semester = 'S2000'
```

But if multiple sections of same course are taught, use:

```
SELECT COUNT (DISTINCT T.CrsCode)
FROM Teaching T
WHERE  T.Semester = 'S2000'
```

43

Grouping

- But how do we compute the number of courses taught in S2000 *per professor*?
  - Strategy 1: Fire off a separate query for each professor:

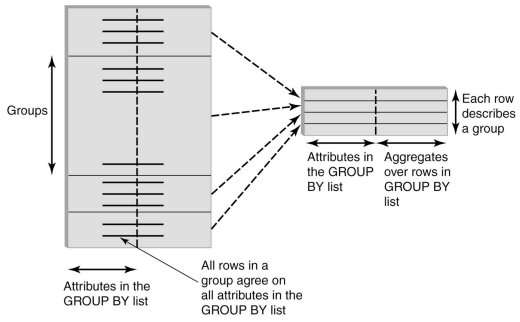
```
SELECT   COUNT(T.CrsCode)
FROM     Teaching T
WHERE  T.Semester = 'S2000' AND T.ProfId = 123456789
```

    - Cumbersome
    - What if the number of professors changes? Add another query?
  - Strategy 2: define a special *grouping operator*:

```
SELECT     T.ProfId,  COUNT(T.CrsCode)
FROM       Teaching  T
WHERE      T.Semester = 'S2000'
GROUP BY  T.ProfId
```

44

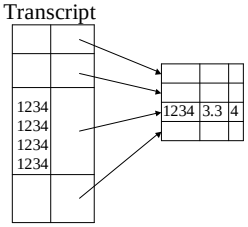
GROUP BY



45

GROUP BY - Example

Transcript



Attributes:
-student's *Id*
-avg grade
-number of courses

```
SELECT T.StudId, AVG(T.Grade), COUNT (*)
FROM Transcript T
GROUP BY T.StudId
```

46

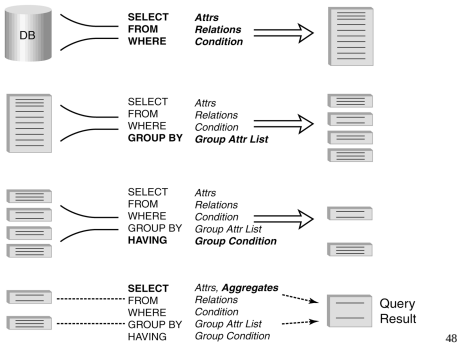
HAVING Clause

- Eliminates unwanted groups (analogous to WHERE clause)
- HAVING condition constructed from attributes of GROUP BY list and aggregates of attributes not in list

```
SELECT  T.StudId, AVG(T.Grade) AS CumGpa,
            COUNT (*) AS NumCrs
FROM   Transcript T
WHERE  T.CrsCode  LIKE  'CS%'
GROUP BY  T.StudId
HAVING  AVG (T.Grade) > 3.5
```

47

Evaluation of GroupBy with Having



Example

- Output the name and address of all seniors on the Dean's List

SELECT S.*Name*, S.*Address*
FROM Student S, Transcript T
WHERE S.*StudId* = T.*StudId* AND S.*Status* = 'senior'

GROUP BY < S.*StudId* -- *wrong*
S.*Name*, S.*Address* -- *right*

HAVING AVG (T.*Grade*) > 3.5 AND SUM (T.*Credit*) > 90

Aggregates: Proper and Improper Usage

SELECT COUNT (T.*CrsCode*), T. *ProfId*
– *makes no sense (in the absence of* GROUP BY *clause)*

SELECT COUNT (*), AVG (T.*Grade*)
– *but this is OK*

WHERE T.*Grade* > COUNT (SELECT ….
– *aggregate cannot be applied to result of* SELECT *statement*

ORDER BY Clause

- Causes rows to be output in a specified order

SELECT T.*StudId*, COUNT (*) AS *NumCrs*,
AVG(T.*Grade*) AS *CumGpa*
FROM Transcript T
WHERE T.*CrsCode* LIKE 'CS%'
GROUP BY T.*StudId*
HAVING AVG (T.*Grade*) > 3.5
ORDER BY DESC *CumGpa*, ASC *StudId*

Query Evaluation Strategy

1 Evaluate FROM: produces Cartesian product, A, of tables in FROM list
2 Evaluate WHERE: produces table, B, consisting of rows of A that satisfy WHERE condition
3 Evaluate GROUP BY: partitions B into groups that agree on attribute values in GROUP BY list
4 Evaluate HAVING: eliminates groups in B that do not satisfy HAVING condition
5 Evaluate SELECT: produces table C containing a row for each group. Attributes in SELECT list limited to those in GROUP BY list and aggregates over group
6 Evaluate ORDER BY: orders rows of C