SQL query

Query elements:

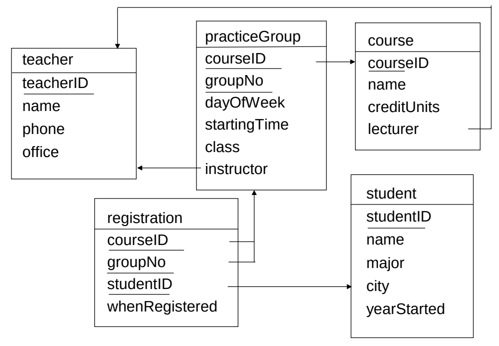
select result_specification
from tables
[where select_conditions]
[group by grouping_criteria]
[having group_restrictions]
[order by ordering_criteria]

may be missing

SQL-query

- A query produces an anonymous result table.
- The values for the elements in the result_specification are computed for each row combination that satisfies the selection criteria listed after the keyword where.

Building a query



Sub-queries

- Sub-queries are queries included inside other queries. They may be used in the where-part of a query and also in the from-part of a query
- A sub-query produces a result table as normal queries
- There are predicates to compare values with the result table (in, not in, Θ some, Θ all, exists, not exists)

Sub-queries

- Find out teachers that give lectures

```
select name from teacher
where teacherID in
  (select lecturer from course)
order by name;
```

- Find out teachers that do not give lectures

```
select name from teacher
where teacherID not in
  (select lecturer from course)
order by name;
```

  – this is one way of expressing difference in SQL

Sub-queries

- Find out teachers that give lectures

```
select name from teacher
where teacherID in
  (select lecturer from course)
order by name;
```

- Find out teachers that do not give lectures

```
select name from teacher
where teacherID not in
  (select lecturer from course)
order by name;
```

  – this is one way of expressing difference in SQL

Sub-queries (connected)

- Teachers that give lectures

```
select name from teacher
where
exists  (select 'yes' from course
              where lecturer= teacher.teacherID)
order by name;
```

there is a condition that connects the sub-query to the external query

the sub-query must be (logically) executed once for each row of the table in the external query

Sub-queries

- Sub-queries may also be used in the from-part of the query. Their results may be temporarily renamed as well as their columns
- from …, (sub-query) [[as] alias] [(column list)], …
- sub-queries in the from-part are useful when combining aggregate data collected using different criteria

Aggregate queries

- SQL provides a collection of aggregate functions

– AVG	average
– MIN	minimum
– MAX	maximum
– SUM	sum
– COUNT	count

  – When aggregate functions are used in the query there will be only one result row, unless grouping is used.

Aggregate queries

- Find the number of students:
  – select count(*) from student;
  – Counts the number of rows
- A constant may be used as the argument of count to get the same result as above
  – select count(1) from student;
  – For each student row we get a "1'" and the number of them is counted
- If a column is given as the argument we get the number of non-null values in the column
  – select count(studentID) from student;

Aggregate queries

- If keyword distinct precedes the argument, then only distinct non-null values are counted
- Find the number of cities the students live in
  select count(distinct city) from student
- What is the longest time anybody living in Helsinki has studied
  select 2004-min(startingYear) from student
  where city='Helsinki';
- When computing average, sum, minimum, and maximum, null values are omitted
- The average credit units for courses
  – select avg(creditUnit) from course;

Aggregate queries

- It's not possible to include in the answer both detail data and aggregate data from the same set of rows
- Which course gives the biggest amount of credit units and how many?
- This cannot be solved as:

```
select name, max(creditUnits)
from course;
```

detail

aggregate value

Aggregate queries

- Which course gives the biggest amount of credit units and how many? Queries that work OK:

```
select name, creditUnits from course
where creditUnits >=
ALL (select creditUnits from course);

Select name, creditUnits from course
where creditUnits=
(select max(creditUnits) from course);

select name, maxUnits
from course,
      (select max(creditUnits) maxUnits from course) as m
where course.creditUnits =m.maxUnits;
```

logically 'different' sets

Aggregate queries with groups

- If grouping is used the result will contain one row for each group.
- grouping is specified by listing the columns (or expression) the values of which determine the groups
- each distinct value combination determines a group
- groups are formed after the conditions of the where part have been first evaluated

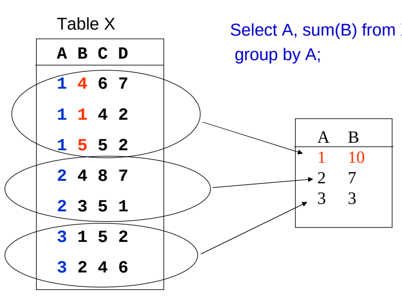
Aggregate queries with groups

Table X

A	B	C	D
1	4	6	7
1	1	4	2
1	5	5	2
2	4	8	7
2	3	5	1
3	1	5	2
3	2	4	6

```
Select A, sum(B) from X
 group by A;
```

A	B
1	10
2	7
3	3



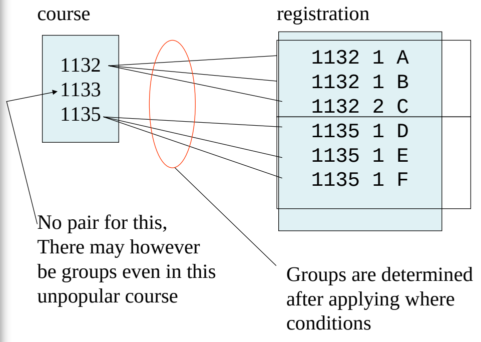
Aggregate queries with groups

- When grouping is used the result may contain only the columns listed in the group by specification, constants and aggregate function results
- All columns listed in the group by specification need not be included in the result (but usually they are)

```
select course.courseID, name, groupNo, count(*)
from course, registration
where registration.courseID=course.courseID
group by course.courseID, name, groupNo;
```

- Name is needed in the above group by specification because we want to include it in the result. It does not affect on how the groups are determined
- The above query does not list all the groups!

Aggregate queries with groups


course
registration
1132
1133
1135
1132 1 A
1132 1 B
1132 2 C
1135 1 D
1135 1 E
1135 1 F
No pair for this, There may however be groups even in this unpopular course
Groups are determined after applying where conditions

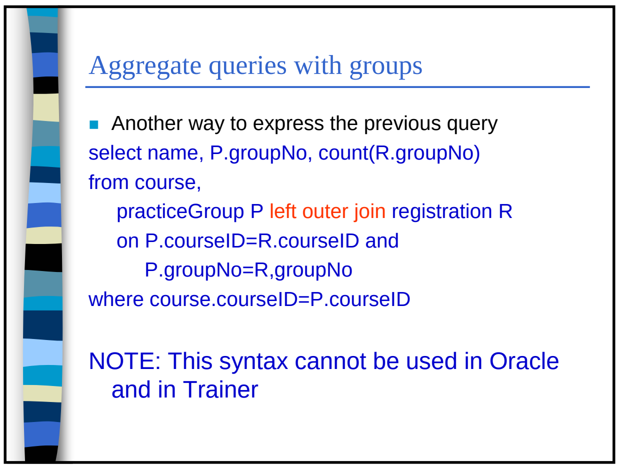
Aggregate queries with groups

```
select name, groupNo, count(*)
from course, registration
where registration.courseID=course.courseID
group by name, groupNo
```
non-empty

union

empty

```
select name, groupNo, 0
from course, practiceGroup P,
where course.courseID=P.courseID and
     (course.courseID, P.GroupNo) not in
     (select courseID, groupNo from registration)
```

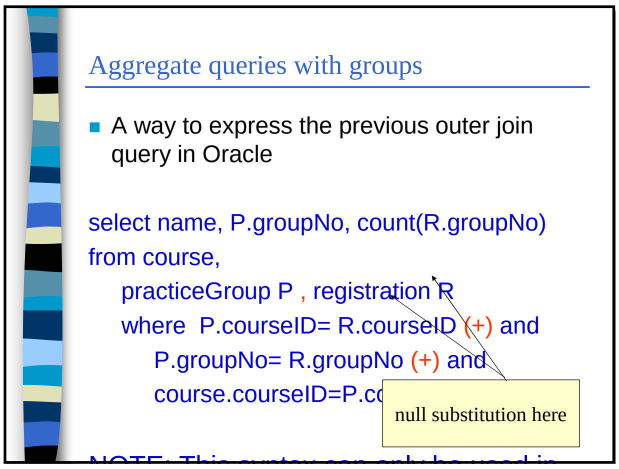


Aggregate queries with groups

- Another way to express the previous query

```
select name, P.groupNo, count(R.groupNo)
from course,
    practiceGroup P left outer join registration R
    on P.courseID=R.courseID and
        P.groupNo=R,groupNo
where course.courseID=P.courseID
```

NOTE: This syntax cannot be used in Oracle and in Trainer



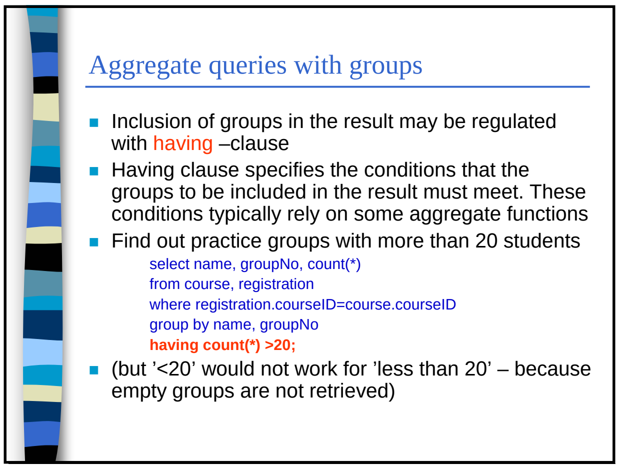
Aggregate queries with groups

- A way to express the previous outer join query in Oracle

```
select name, P.groupNo, count(R.groupNo)
from course,
    practiceGroup P , registration R
    where  P.courseID= R.courseID (+) and
        P.groupNo= R.groupNo (+) and
        course.courseID=P.co
```

null substitution here

NOTE: This syntax can only be used in

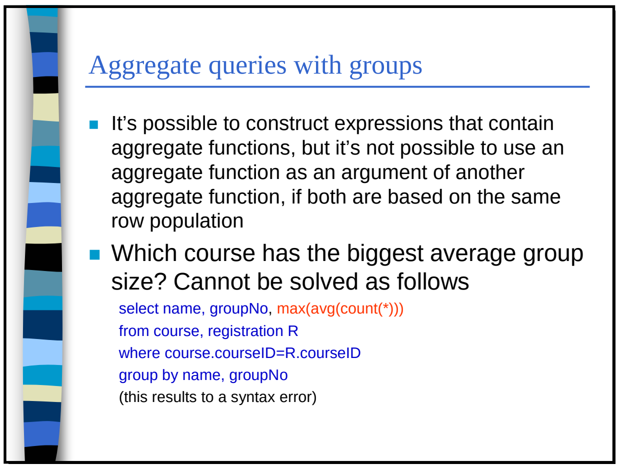


Aggregate queries with groups

- Inclusion of groups in the result may be regulated with having –clause
- Having clause specifies the conditions that the groups to be included in the result must meet. These conditions typically rely on some aggregate functions
- Find out practice groups with more than 20 students

```
select name, groupNo, count(*)
from course, registration
where registration.courseID=course.courseID
group by name, groupNo
having count(*) >20;
```

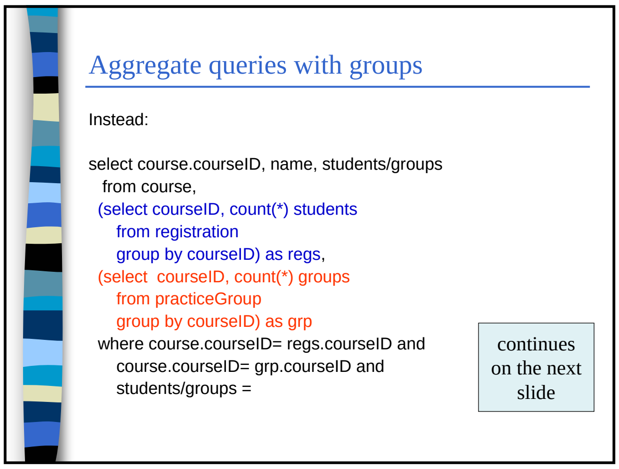
- (but '<20' would not work for 'less than 20' – because empty groups are not retrieved)



Aggregate queries with groups

- It's possible to construct expressions that contain aggregate functions, but it's not possible to use an aggregate function as an argument of another aggregate function, if both are based on the same row population
- Which course has the biggest average group size? Cannot be solved as follows

```
select name, groupNo, max(avg(count(*)))
from course, registration R
where course.courseID=R.courseID
group by name, groupNo
(this results to a syntax error)
```

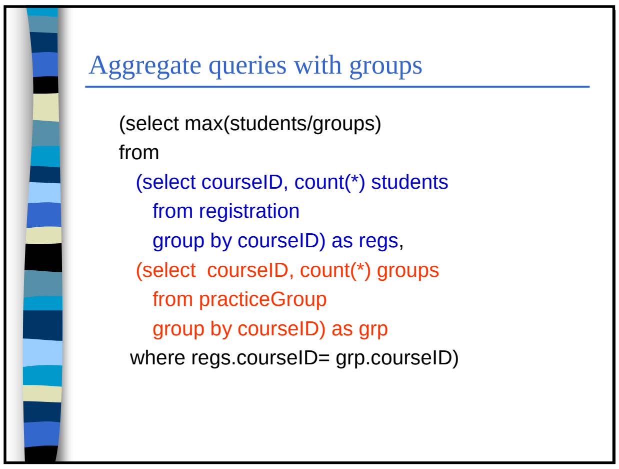


Aggregate queries with groups

Instead:

```
select course.courseID, name, students/groups
  from course,
 (select courseID, count(*) students
    from registration
    group by courseID) as regs,
 (select  courseID, count(*) groups
    from practiceGroup
    group by courseID) as grp
 where course.courseID= regs.courseID and
    course.courseID= grp.courseID and
    students/groups =
```

continues on the next slide



Aggregate queries with groups

```
(select max(students/groups)
from
  (select courseID, count(*) students
    from registration
    group by courseID) as regs,
  (select  courseID, count(*) groups
    from practiceGroup
    group by courseID) as grp
 where regs.courseID= grp.courseID)
```