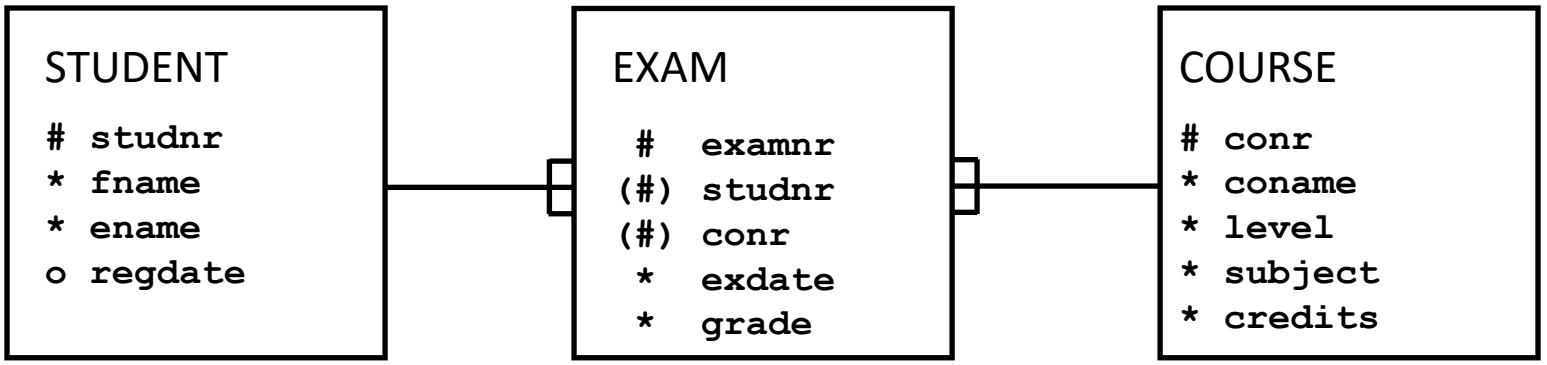
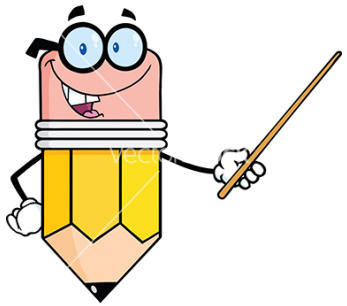
Lecture 09: Queries against several tables

1. ***Sub queries (nested queries)***
2. ***IN-operator***
3. ***Join***
4. ***Cartesian product***
5. ***Table alias***
6. ***Distinct***
7. ***Group by***
8. ***Having***
9. ***In Line View***
10. ***Order by***
11. ***The pseudo column: rownum***
12. ***ANSI Join syntax***

Pär Douhan, pdo@du.se

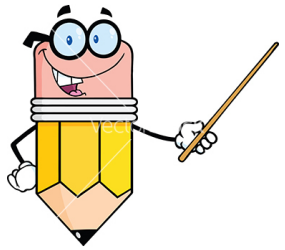
Exercises


STUDENT
studnr
* fname
* ename
o regdate
EXAM
examnr
(#) studnr
(#) conr
* exdate
* grade
COURSE
conr
* coname
* level
* subject
* credits



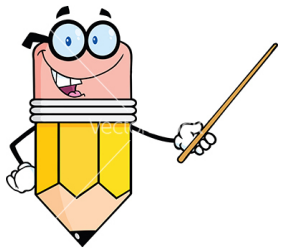
Nested search

- When columns are shown from only one table.
- The IN-operator is used when the nested query may return more than one row.
- The sub query must be within (parentheses).
- The nested queries are executed, in a given order, inside and out.
- The result is "distincted" i.e. no duplicate rows.
- No sorting (order by) in sub queries.



Join

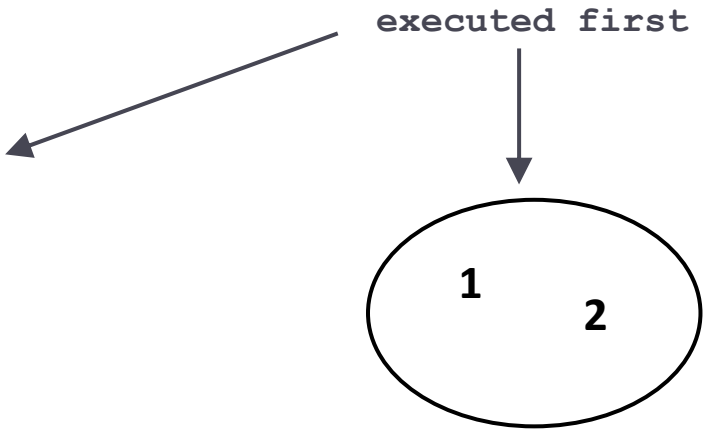
- Can show columns from more than one table.
- The result shows with duplicate rows, has to be "distincted" in order to remove duplicate rows.



E1

E1: Show STUDNR, FNAME, ENAME for those students who have studied course number 1. Use nested search.

```
select studnr,fname,ename
from student
where studnr in(select studnr
                from exam
                where conr = 1);
```




```
STUDNR FNAME                          ENAME
------ ------------------------------ ---------
     1 niklas                         strömberg
     2 arman                          dobitnik
```

E2

E2: Show STUDNR, FNAME, ENAME for those student who have studied a course with the name 'beginning java'. Use sub queries.

```
select studnr,fname,ename
from student
where studnr in(select studnr
                from exam
                where conr    in (select conr
                                  from course
                                  where lower(coname) = 'beginning java'));
```

1

1 2

```
STUDNR FNAME                          ENAME
------ ------------------------------ ---------
     1 niklas                         strömberg
     2 arman                          dobitnik
```

E3

E3: Same as E2 but use Equi-Join without table alias.

```
select student.studnr,student.fname,student.ename
from student,exam,course
where lower(course.coname) = 'beginning java'
and student.studnr = exam.studnr
and exam.conr = course.conr;
```

-- Join condition

If we forget the join condition, we get cartesian product

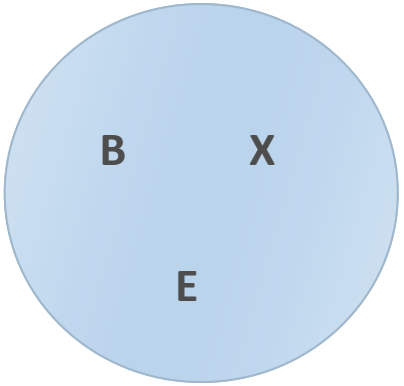
```
STUDNR FNAME                          ENAME
------ ------------------------------ ---------
     1 niklas                         strömberg
     2 arman                          dobitnik
```

Cartesian product

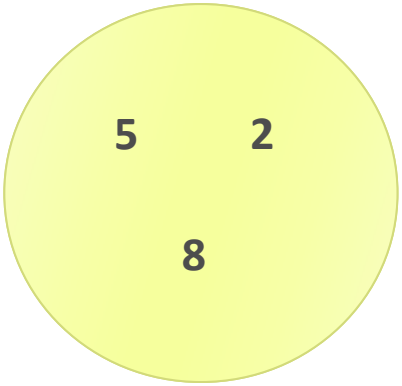
A collection is an unordered set of unique data

A database table = a collection.

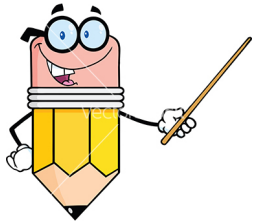
Collection A



Collection B



The cartesian product of A x B is the set of all pairs (a,b)



```
A x B = {B,X,E} x {5,2,8}

= {(B,5),(B,2),(B,8),(X,5),(X,2),(X,8),(E,5),(E,2),(E,8)}
```

E4

E4: Same as E3 but use table alias.

Does not work with as

```
select s.studnr,s.fname,s.ename
from student s,exam e,course c
where lower(c.coname) = 'beginning java'
and s.studnr = e.studnr
and e.conr = c.conr;
```

```
STUDNR FNAME                          ENAME
------ ------------------------------ ---------
     1 niklas                         strömberg
     2 arman                          dobitnik
```

E5

E5: Show FNAME, ENAME for those students who haven't participated in any course.

```
select initcap(fname||' '||ename) NO_STUDY_RESULT
from student
where studnr not in (select studnr
                     from exam);
```

```
NO_STUDY_RESULT
-------------------------
Hans Dalros
```

E6

E6: Show FNAME, ENAME for those students who have participated in a course, order by ename descending. Use join!

```
select s.fname,s.ename
from student s, exam e
where s.studnr = e.studnr  -- Join condition
order by ename desc;
```

```
FNAME                    ENAME
------------------------ ---------
niklas                   strömberg
niklas                   strömberg
lena                     nordin
anna                     larsson
jörgen                   killing
arman                    dobitnik
arman                    dobitnik
arman                    dobitnik
```

E7

E7: Same as E6, without duplicates.

```
select distinct s.fname,s.ename
from student s, exam e
where s.studnr = e.studnr
order by enamn desc;
```

```
FNAME                          ENAME
------------------------------ ---------
niklas                         strömberg
lena                           nordin
anna                           larsson
jörgen                         killing
arman                          dobitnik
```

E8

E8: Same as E7 but use sub queries instead.

```
select fname,ename
from student
where studnr in (select studnr
                 from exam)
order by ename desc;    -- No order by in the sub query!
```

```
FNAME                          ENAME
------------------------------ ---------
niklas                         strömberg
lena                           nordin
anna                           larsson
jörgen                         killing
arman                          dobitnik
```

E9

E9: Show STUDNR for those students who haven't participated in the course 'AI for Masters'. Join or nested search? Always nesting, with the negation in the outer query.

```
select studnr
from student
where studnr not in(select studnr
                    from exam
                    where conr in(select conr
                                  from course
                                  where upper(coname) = 'AI FOR MASTERS'));
```

```
   STUDNR
   ------
        1
        2
        5
        6
```

E10

E10: Show STUDNR, FNAME, ENAME and the number of courses that the students have participated in.

```
select s.studnr,s.fname,s.ename,count(e.examnr) number_of_courses
from student s,exam e
where s.studnr = e.studnr
group by s.studnr,s.fname,s.ename; /* the columns that count() do not work
                                      with! */
```

STUDNR	FNAME	ENAME	NUMBER_OF_COURSES
1	niklas	strömberg	2
2	arman	dobitnik	3
3	anna	larsson	1
4	lena	nordin	1
5	jörgen	killing	1

E11

E11: Same as E10, but also show the students that haven't read any courses. Outer Join shows NULL-values.

```
select s.studnr,s.fname,s.ename,nvl(count(e.examnr),0) number_of_courses
from student s,exam e
where s.studnr = e.studnr(+)
group by s.studnr,s.fname,s.ename;
```

STUDNR	FNAME	ENAME	NUMBER_OF_COURSES
1	niklas	strömberg	2
2	arman	dobitnik	3
3	anna	larsson	1
4	lena	nordin	1
5	jörgen	killing	1
6	hans	dalros	0

E12

E12: Show STUDNR, FNAME, ENAME and the total credits that each student has read.

```
select s.studnr,s.fname,s.ename,sum(c.credits) sum_credits
from student s,exam e,course c
where s.studnr = e.studnr
and e.conr = c.conr
group by s.studnr,s.fname,s.ename;
```

```
STUDNR FNAME      ENAME        SUM_CREDITS
------ ---------- ----------- -----------
     1 niklas     strömberg            10
     2 arman      dobitnik             30
     3 anna       larsson               5
     4 lena       nordin                5
```

E13

E13: Same as E12 but sort the result so the student with most credits is on top.

```
select s.studnr,s.fname,s.ename,sum(c.credits) sum_credits
from student s,exam e,course c
where s.studnr = e.studnr
and e.conr = c.conr
group by s.studnr,s.fname,s.ename
order by sum(c.credits) desc;
```

```
STUDNR FNAME      ENAME           SUM_CREDITS
------ ---------- ------------- -----------
     2 arman      dobitnik             30
     1 niklas     strömberg            10
     3 anna       larsson               5
     4 lena       nordin                5
     5 jörgen     killing               5
```

E14

E14: Same as E13 but show only the sudents with 10 credits or more.

```
select s.studnr,s.fname,s.ename,sum(c.credits) sum_credits
from student s,exam e,course c
where s.studnr = c.studnr
and e.conr = c.conr
group by s.studnr,s.fname,s.ename
having sum(c.credits) >= 10  -- condition for group functions
order by sum(c.credits) desc;
```

```
STUDNR FNAME      ENAME        SUM_CREDITS
------ ---------- ----------- -----------
     2 arman      dobitnik            30
     1 niklas     strömberg           10
```

E15

E15: Same as E14, but only the top row (with the highest value). Use an 'in line view' to solve the task:

```
select *
from
(  -- in line view start
select s.studnr,s.fname,s.ename,sum(c.credits) sum_credits
from student s,exam e,course c
where s.studnr = e.studnr
and e.conr = c.conr
group by s.studnr,s.fname,s.ename
order by sum(c.credit) desc
) -- in line view end
where rownum = 1;  -- using Oracle's pseudo column rownum
```

```
STUDNR FNAME           ENAME               SUM_CREDITS
------ --------------- ------------------- -----------
     2 arman           dobitnik            30
```

ANSI join syntax

Joins are used to **combine data** from **multiple tables** to form a single result set. Oracle provides **two approaches** to joining tables, the **non-ANSI join** syntax and the **ANSI join** syntax, which look quite different.

The non-ANSI join syntax has historically been the way you perform joins in Oracle and it is still very popular today. The tables to be joined are listed in the FROM clause and the join conditions are defined as predicates in the WHERE clause. Even if you don't like it, you are going to have to get used to it as there is a lot of code out there that still uses it. If you are not familiar with the syntax you will struggle to bug fix any existing code and some of the examples on the internet will look rather mysterious to you.

Many Oracle developers still use the non-ANSI join syntax. Partly this is just because of habit. Partly this is because the Oracle optimizer transforms most ANSI join syntax into the non-ANSI join syntax equivalent before it is executed.

ANSI join syntax

Some join methods are more popular than others, so initially focus your attention on those you are most likely to see. The most common joins you are likely to see in code are the following.

- [INNER] JOIN ... ON
- LEFT [OUTER] JOIN
- RIGHT [OUTER] JOIN

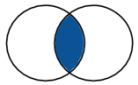
Let's have a look at some examples!

Create tables

```
CREATE TABLE departments (
  department_id   NUMBER(2) CONSTRAINT departments_pk PRIMARY KEY,
  department_name VARCHAR2(14),
  location        VARCHAR2(13));
```

```
CREATE TABLE employees (
  employee_id   NUMBER(4) CONSTRAINT employees_pk PRIMARY KEY,
  employee_name VARCHAR2(10),
  job           VARCHAR2(9),
  manager_id    NUMBER(4),
  hiredate      DATE,
  salary        NUMBER(7,2),
  commission    NUMBER(7,2),
  department_id NUMBER(2) CONSTRAINT emp_department_id_fk REFERENCES departments(department_id));
```

[INNER] JOIN … ON



An INNER JOIN combines data from two tables where there is a match on the joining column(s) in both tables.

Here is an example of an ANSI INNER JOIN:

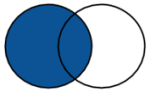
```
SELECT d.department_name,e.employee_name
FROM   departments d
       JOIN employees e ON d.department_id = e.department_id
WHERE  d.department_id >= 30
ORDER BY d.department_name;
```

```
DEPARTMENT_NAM EMPLOYEE_N
-------------- ----------
SALES          ALLEN
SALES          BLAKE
SALES          JAMES
SALES          MARTIN
SALES          TURNER
SALES          WARD
```

Here is the non-ANSI equivalent of the previous statement:

```
SELECT d.department_name,e.employee_name
FROM   departments d, employees e
WHERE  d.department_id = e.department_id
AND    d.department_id >= 30
ORDER BY d.department_name;
```

LEFT [OUTER] JOIN



Using the previous example but switching to a LEFT OUTER JOIN means we will see the OPERATIONS department, even though it has no employees.

```
DEPARTMENT_NAME EMPLOYEE_NAME
--------------- -------------
OPERATIONS
SALES           ALLEN
SALES           BLAKE
SALES           JAMES
SALES           MARTIN
SALES           TURNER
SALES           WARD
```

Here is an example of an ANSI LEFT OUTER JOIN:

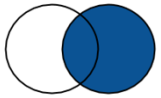
```
SELECT d.department_name,e.employee_name
FROM   departments d
       LEFT OUTER JOIN employees e ON d.department_id = e.department_id
WHERE  d.department_id >= 30
ORDER BY d.department_name;
```

Here is the non-ANSI equivalent of the previous statement:

```
SELECT d.department_name,e.employee_name
FROM   departments d, employees e
WHERE  d.department_id = e.department_id(+)
AND    d.department_id >= 30
ORDER BY d.department_name;
```

Notice the "(+)" is used to indicate the side of the join condition that may be missing. The non-ANSI join syntax is not affected by the order of the tables

RIGHT [OUTER] JOIN



The RIGHT [OUTER] JOIN is the opposite of the LEFT [OUTER] JOIN. It returns all valid rows from the table on the right side of the JOIN keyword, along with the values from the table on the left side, or NULLs if a matching row doesn't exist. All points raised in the previous section apply here also.

The following example has altered the order of the tables so a RIGHT [OUTER] JOIN is now required:

```
DEPARTMENT_NAME EMPLOYEE_NAME
--------------- -------------
OPERATIONS
SALES           ALLEN
SALES           BLAKE
SALES           JAMES
SALES           MARTIN
SALES           TURNER
SALES           WARD
```

```
SELECT d.department_name,
       e.employee_name
FROM   employees e
       RIGHT OUTER JOIN departments d ON e.department_id = d.department_id
WHERE  d.department_id >= 30
ORDER BY d.department_name, e.employee_name;
```

Remember, the non-ANSI outer join syntax is not dependent on table order, so there is no real concept of right or left outer joins, just outer joins.

The End