11: Data Manipulation (part two)

null values - page 531

Tx_isbn	Tx_title	Tx_year	Tx_publisher
000-66574998	Database Management	1999	Thomson
003-6679233	Linear Programming	1997	Prentice-Hall
001-55-435	Simulation Modeling	2001	Springer
118-99898-67	Systems Analysis	2000	Thomson
77898-8769	Principles of IS	2002	Prentice-Hall
0296748-99	Economics For Managers	2001	
0296437-1118	Programming in C++	2002	Thomson
012-54765-32	Fundamentals of SQL	2004	
111-11111111	Data Modeling	2006	

d NULL

th zero

⚠ 0 and blank spaces are not null values

```
INSERT INTO Textbook VALUES ('012-54765-32', 'Fundamentals of SQL', 2004, null);
INSERT INTO Textbook VALUES ('0296748-99', 'Economics for Managers', 2001, '');
INSERT INTO Textbook VALUES ('111-11111111', 'Data Modeling', 2006, ' ');
```

textbook relation

Tx_isbn	Tx_title	Tx_year	Tx_publisher
000-66574998	Database Management	1999	Thomson
003-6679233	Linear Programming	1997	Prentice-Hall
001-55-435	Simulation Modeling	2001	Springer
118-99898-67	Systems Analysis	2000	Thomson
77898-8769	Principles of IS	2002	Prentice-Hall
0296748-99	Economics For Managers	2001	
0296437-1118	Programming in C++	2002	Thomson
012-54765-32	Fundamentals of SQL	2004	
111-11111111	Data Modeling	2006	

null

blank space

null values look like blank spaces when viewing query results

textbook relation

```
SELECT TEXTBOOK.TX_PUBLISHER FROM TEXTBOOK;
```

```
Tx_publisher
-------------
Thomson
Prentice-Hall
Springer
Thomson
Prentice-Hall

Thomson



9 rows selected.
```

is null and not null

```
SELECT TEXTBOOK.TX_TITLE,
TEXTBOOK.TX_PUBLISHER FROM TEXTBOOK
WHERE TEXTBOOK.TX_PUBLISHER IS NOT NULL;
```

```
Tx_title                  Tx_publisher
------------------------- -------------
Database Management       Thomson
Linear Programming        Prentice-Hall
Simulation Modeling       Springer
Systems Analysis          Thomson
Principles of IS          Prentice-Hall
Programming in C++        Thomson
Data Modeling

7 rows selected.
```

\ blank space

is null and not null

```
SELECT * FROM TEXTBOOK
WHERE TEXTBOOK.TX_PUBLISHER IS NULL;
```

Tx_isbn	Tx_title	Tx_year	Tx_publisher
0296748-99	Economics For Managers	2001	
012-54765-32	Fundamentals of SQL	2004	

is null and not null

```
SELECT * FROM TEXTBOOK
WHERE TEXTBOOK.TX_PUBLISHER <> NULL;
```

```
no rows selected.
```

cannot use relational operators >, <, =, <>, etc. with null values

null values

```
SELECT DISTINCT TEXTBOOK.TX_PUBLISHER
    FROM TEXTBOOK;
```

```
Tx_publisher
-------------
Prentice-Hall
Springer
Thomson

5 rows selected.
```

null

blank space

COUNT function

```
SELECT COUNT(*) FROM TEXTBOOK;
```

```
  COUNT(*)
----------
         9

1 row selected.
```

count(*) includes null values

COUNT function

```
SELECT COUNT(TEXTBOOK.TX_PUBLISHER) FROM
TEXTBOOK;
```

```
COUNT(TEXTBOOK.Tx_publisher)
----------------------------
                           7

1 row selected.
```

count(*column_name*) ignores null values

COUNT function

```
SELECT COUNT(TEXTBOOK.TX_PUBLISHER) AS
"NUMBER OF PUBLISHERS" FROM TEXTBOOK;
```

```
Number of Publishers
--------------------
                   7
```

COUNT function

```
SELECT DISTINCT COUNT(TEXTBOOK.TX_PUBLISHER)
  AS "NUMBER OF DISTINCT PUBLISHERS" FROM
                    TEXTBOOK;
```

```
Number of Distinct Publishers
-----------------------------
                            4

1 row selected.
```

COUNT function

```
SELECT TEXTBOOK.TX_PUBLISHER, COUNT(*)
    FROM TEXTBOOK
GROUP BY TEXTBOOK.TX_PUBLISHER;
```

```
Tx_publisher    COUNT(*)
------------- ----------
                         1
Prentice-Hall            2
Springer                 1
Thomson                  3
                         2

5 rows selected.
```

blank space

null

COUNT function

```
SELECT TEXTBOOK.TX_PUBLISHER,
COUNT(TEXTBOOK.TX_PUBLISHER) FROM TEXTBOOK
GROUP BY TEXTBOOK.TX_PUBLISHER;
```

```
Tx_publisher  COUNT(TEXTBOOK.Tx_publisher)
------------- ---------------------------
                                         1
Prentice-Hall                            2
Springer                                 1
Thomson                                  3
                                         0

5 rows selected.
```

blank space

null

COUNT function

```
SELECT TEXTBOOK.TX_PUBLISHER, COUNT(*) FROM
TEXTBOOK WHERE TEXTBOOK.TX_PUBLISHER IS NOT
    NULL GROUP BY TEXTBOOK.TX_PUBLISHER;
```

Tx_publisher	COUNT(*)
	1
Prentice-Hall	2
Springer	1
Thomson	3

blank space

4 rows selected.

COUNT function

```
SELECT TEXTBOOK.TX_PUBLISHER, COUNT(*) FROM
TEXTBOOK WHERE TEXTBOOK.TX_PUBLISHER = ' '
      GROUP BY TEXTBOOK.TX_PUBLISHER;

Tx_publisher    COUNT(*)
------------- ----------
                        1

1 row selected.
```

blank space

COUNT function

```
SELECT TEXTBOOK.TX_PUBLISHER, COUNT(*) FROM
TEXTBOOK WHERE TEXTBOOK.TX_PUBLISHER <> ' '
      GROUP BY TEXTBOOK.TX_PUBLISHER;
```

Tx_publisher	COUNT(*)
Prentice-Hall	2
Springer	1
Thomson	3

3 rows selected.

cannot use relational operators >, <, =, <>, etc. with null values

pattern matching

%	one or more characters
_	one character

pattern matching

```
SELECT TEXTBOOK.TX_TITLE FROM TEXTBOOK
  WHERE TEXTBOOK.TX_TITLE LIKE '_i%';
```

```
Tx_title
--------------------
Linear Programming
Simulation Modeling

2 rows selected.
```

pattern matching

```
SELECT TEXTBOOK.TX_TITLE FROM TEXTBOOK
  WHERE TEXTBOOK.TX_TITLE LIKE 'P%';
```

```
Tx_title
--------------------
Principles of IS
Programming in C++

2 rows selected.
```

pattern matching

```
SELECT TEXTBOOK.TX_TITLE FROM TEXTBOOK
  WHERE TEXTBOOK.TX_TITLE LIKE 'p%';
```

```
no rows selected.
```

pattern matching

```
SELECT TEXTBOOK.TX_TITLE FROM TEXTBOOK
  WHERE TEXTBOOK.TX_TITLE LIKE '%e%';
```

```
Tx_title
--------------------
Database Management
Linear Programming
Simulation Modeling
Systems Analysis
Principles of IS
Economics For Managers
Fundamentals of SQL
Data Modeling

8 rows selected.
```

pattern matching

```
SELECT TEXTBOOK.TX_TITLE FROM TEXTBOOK
  WHERE TEXTBOOK.TX_TITLE LIKE '%';
```

Tx_isbn	Tx_title	Tx_year	Tx_publisher
000-66574998	Database Management	1999	Thomson
003-6679233	Linear Programming	1997	Prentice-Hall
001-55-435	Simulation Modeling	2001	Springer
118-99898-67	Systems Analysis	2000	Thomson
77898-8769	Principles of IS	2002	Prentice-Hall
0296437-1118	Programming in C++	2002	Thomson
111-11111111	Data Modeling	2006	

7 rows selected.

null values not included

set theoretic operations - page 546

RELATION R

Se_section#	Se_qtr	Se_year	Se_time	Se_maxst	Se_room	Se_co_course#	Se_pr_profid
902	A	2006	H1700	25	Lindner 108	22IS270	SK85977
901	A	2006	W1800	35	Rhodes 611	22IS270	SK85977
101	A	2007	H1700	29	Lindner 108	22IS330	SK85977
101	A	2007	T1015	25		22QA375	HT54347
101	A	2007	W1800		Baldwin 437	20ECES212	RR79345

RELATION S

Se_section#	Se_qtr	Se_year	Se_time	Se_maxst	Se_room	Se_co_course#	Se_pr_profid
902	A	2006	H1700	25	Lindner 108	22IS270	SK85977
101	S	2006	T1045	29	Lindner 110	22IS330	SK85977
102	S	2006	H1045	29	Lindner 110	22IS330	CC49234
101	A	2007	H1700	29	Lindner 108	22IS330	SK85977

union - page 546

```
SELECT * FROM R
UNION
SELECT * FROM S;
```

Se_section#	Se_qtr	Se_year	Se_time	Se_maxst	Se_room	Se_co_course#	Se_pr_profid
101	A	2007	H1700	29	Lindner 108	22IS330	SK85977
101	A	2007	T1015	25		22QA375	HT54347
101	A	2007	W1800		Baldwin 437	20ECES212	RR79345
101	S	2006	T1045	29	Lindner 110	22IS330	SK85977
102	S	2006	H1045	29	Lindner 110	22IS330	CC49234
901	A	2006	W1800	35	Rhodes 611	22IS270	SK85977
902	A	2006	H1700	25	Lindner 108	22IS270	SK85977

duplicate rows not included (unless UNION ALL is used)

intersect - page 547

```
SELECT * FROM R
INTERSECT
SELECT * FROM S;
```

Se_section#	Se_qtr	Se_year	Se_time	Se_maxst	Se_room	Se_co_course#	Se_pr_profid
101	A	2007	H1700	29	Lindner 108	22IS330	SK85977
902	A	2006	H1700	25	Lindner 108	22IS270	SK85977

minus - page 547

```
SELECT * FROM R
MINUS
SELECT * FROM S;
```

Se_section#	Se_qtr	Se_year	Se_time	Se_maxst	Se_room	Se_co_course#	Se_pr_profid
101	A	2007	T1015	25		22QA375	HT54347
101	A	2007	W1800		Baldwin 437	20ECES212	RR79345
901	A	2006	W1800	35	Rhodes 611	22IS270	SK85977

```
SELECT * FROM S
MINUS
SELECT * FROM R;
```

Se_section#	Se_qtr	Se_year	Se_time	Se_maxst	Se_room	Se_co_course#	Se_pr_profid
101	S	2006	T1045	29	Lindner 110	22IS330	SK85977
102	S	2006	H1045	29	Lindner 110	22IS330	CC49234

inner joins

```
SELECT
FROM <TABLE>
JOIN <TABLE> ON
<JOIN CRITERIA>
```

inner joins

```
SELECT
FROM <TABLE>,
<TABLE> WHERE
<JOIN CRITERIA>
```

inner joins

what are the student IDs of those students who took a section of a course during the year 2006?

```
SELECT TAKES.TK_ST_SID FROM TAKES INNER JOIN
    SECTION ON SECTION.SE_YEAR = 2006 AND
        TAKES.TK_SE_YEAR = 2006 AND
SECTION.SE_SECTION# = TAKES.TK_SE_SECTION#;
```

```
Tk_st_sid
---------
BE76598

1 row selected.
```

self join - page 550

list the student IDs of those students recorded as having taken more than one course

```
SELECT X.TK_ST_SID
FROM TAKES X JOIN TAKES Y
ON X.TK_ST_SID = Y.TK_ST_SID
AND X.TK_SE_CO_COURSE# <> Y.TK_SE_CO_COURSE#;
```

```
Tk_st_sid
---------
BG66765
BG66765
KP78924
KP78924
KS39874
KS39874
KS39874
KS39874
KS39874
KS39874
```

self join - page 550

list the student IDs of those students recorded as having taken more than one course

```
SELECT DISTINCT X.TK_ST_SID
  FROM TAKES X JOIN TAKES Y
 ON X.TK_ST_SID = Y.TK_ST_SID
AND X.TK_SE_CO_COURSE# <> Y.TK_SE_CO_COURSE#;
```

```
Tk_st_sid
---------
BG66765
KP78924
KS39874
```

joins

list the names of those students having taken more than one course

```
SELECT DISTINCT STUDENT.ST_NAME
    FROM (TAKES X JOIN TAKES Y
    ON X.TK_ST_SID = Y.TK_ST_SID
AND X.TK_SE_CO_COURSE# <> Y.TK_SE_CO_COURSE#)
JOIN STUDENT ON X.TK_ST_SID = STUDENT.ST_SID;
```

```
St_name
--------------
Gladis Bale
Poppy Kramer
Sweety Kramer
```

joins - page 551

for each student taking a course in the fall quarter of 2007, list the student's name, classroom where the course is offered, and course number

```
SELECT STUDENT.ST_NAME, SECTION.SE_ROOM,
TAKES.TK_SE_CO_COURSE# FROM (STUDENT JOIN
TAKES ON STUDENT.ST_SID = TAKES.TK_ST_SID)
JOIN SECTION ON TAKES.TK_SE_CO_COURSE# =
SECTION.SE_CO_COURSE# AND SECTION.SE_QTR =
TAKES.TK_SE_QTR AND
SECTION.SE_YEAR = TAKES.TK_SE_YEAR AND
TAKES.TK_SE_QTR = 'A' AND TAKES.TK_SE_YEAR =
2007;
```

joins - page 551-552

St_name	Se_room	Tk_se_co_course
Poppy Kramer		22QA375
Sweety Kramer		22QA375
Gladis Bale		22QA375
Joumana Kidd	Lindner 108	22IS330
Poppy Kramer	Lindner 108	22IS330
Sweety Kramer	Lindner 108	22IS330
Sweety Kramer	Lindner 108	22IS330
Elijah Baley	Lindner 108	22IS330
Shweta Gupta	Lindner 108	22IS330

left outer join - page 553

for each section taken by a student, display the section number, quarter, year, course number, grade, student ID, and student name. include the names of all students (i.e., even those who have never taken a course)

```
SELECT TAKES.*, ST_NAME
FROM STUDENT LEFT OUTER JOIN TAKES
ON STUDENT.ST_SID = TAKES.TK_ST_SID;
```

left outer join - page 553-554

Tk_se_section#	Tk_se_qtr	Tk_se_year	Tk_se_co_course#	Tk_grade	Tk_st_sid	St_name
101	A	2007	22QA375	A	KP78924	Poppy Kramer
101	A	2007	22QA375	A	KS39874	Sweety Kramer
101	A	2007	22QA375	B	BG66765	Gladis Bale
101	S	2006	22IS330	C	BE76598	Elijah Baley
101	A	2007	22IS330	B	KJ56656	Joumana Kidd
101	A	2007	22IS330	A	KP78924	Poppy Kramer
101	A	2007	22IS330	A	KS39874	Sweety Kramer
701	W	2007	22IS832	A	KS39874	Sweety Kramer
101	A	2007	22IS330	A	BE76598	Elijah Baley
701	W	2007	22IS832	B	BG66765	Gladis Bale
101	A	2007	22IS330	C	GS76775	Shweta Gupta
						Tim Duncan
						Troy Hudson
						Rick Fox
						Jenny Aniston
						Daniel Olive
						Diana Jackson
						Jenna Hopp
						Wanda Seldon
						David Sane
						Vanessa Fox

verifying the results - page 554

```
SELECT STUDENT.ST_SID FROM STUDENT
MINUS SELECT TAKES.TK_ST_SID FROM TAKES;
```

```
St_sid
-------
AJ76998
DT87656
FR45545
FV67733
HJ45633
HT67657
JD35477
OD76578
SD23556
SW56547
```

right outer join - page 555

for each textbook, display all available information about the book and its usage in courses. include textbooks that have never been used

```
SELECT * FROM USES RIGHT OUTER JOIN TEXTBOOK
    ON USES.US_TX_ISBN = TEXTBOOK.TX_ISBN;
```

right outer join - page 555

Us_co_course#	Us_tx_isbn	Us_pr_empid	Tx_isbn	Tx_title	Tx_year	Tx_publisher
22IS832	000-66574998	S43278	000-66574998	Database Management	1999	Thomson
22IS270	000-66574998	SS43278	000-66574998	Database Management	1999	Thomson
22IS270	77898-8769	SS43278	77898-8769	Principles of IS	2002	Prentice-Hall
22IS270	77898-8769	SK85977	77898-8769	Principles of IS	2002	Prentice-Hall
22IS270	77898-8769	CC49234	77898-8769	Principles of IS	2002	Prentice-Hall
20ECES212	0296437-1118	CC49234	0296437-1118	Programming in C++	2002	Thomson
22QA375	0296437-1118	SJ65436	0296437-1118	Programming in C++	2002	Thomson
22IS330	003-6679233	BC65437	003-6679233	Linear Programming	1997	Prentice-Hall
18ECON123	0296748-99	CM65436	0296748-99	Economics For Managers	2001	
22IS330	118-99898-67	SS43278	118-99898-67	Systems Analysis	2000	Thomson
22IS832	118-99898-67	SK85977	118-99898-67	Systems Analysis	2000	Thomson
22QA888	001-55-435	HT54347	001-55-435	Simulation Modeling	2001	Springer
			111-11111111	Data Modeling	2006	
			012-54765-32	Fundamentals of SQL	2004	

full outer join - page 556

join the GRAD_STUDENT and TAKES tables making sure that each row from each table appears in the result

```
SELECT * FROM GRAD_STUDENT FULL OUTER JOIN
    TAKES ON GRAD_STUDENT.GS_ST_SID =
        TAKES.TK_ST_SID;
```

full outer join - page 556

Gs_st_sid	Gs_thesis	Gs_ugmajor	Tk_se_section#	Tk_se_qtr	Tk_se_year	Tk_se_co_course#	Tk_grade	Tk_st_sid
BG66765	N	Archeology	101	A	2007	22QA375	B	BG66765
BE76598	Y	Marketing	101	S	2006	22IS330	C	BE76598
KJ56656	Y	History	101	A	2007	22IS330	B	KJ56656
BE76598	Y	Marketing	101	A	2007	22IS330	A	BE76598
BG66765	N	Archeology	701	W	2007	22IS832	B	BG66765
GS76775	N	Archeology	101	A	2007	22IS330	C	GS76775
DT87656	N	Physics						
SW56547	Y	Finance						
AJ76998	Y	Child Care						
JD35477	N	Mathematics						
HJ45633	Y	History						
			101	A	2007	22QA375	A	KP78924
			101	A	2007	22QA375	A	KS39874
			101	A	2007	22IS330	A	KP78924
			101	A	2007	22IS330	A	KS39874
			701	W	2007	22IS832	A	KS39874

subqueries

uncorrelated: subquery is executed first and passes one or more values to the outer query (IN, NOT IN, ANY, ALL, etc.)

correlated: subquery is executed once for every row in the outer query (EXISTS, NOT EXISTS)

subqueries

display the course number, course name, and college of those courses for which sections have been offered

```
SELECT COURSE.CO_COURSE#, COURSE.CO_NAME,
COURSE.CO_COLLEGE FROM COURSE
WHERE COURSE.CO_COURSE# IN
(SELECT SECTION.SE_CO_COURSE# FROM SECTION);
```

Co_course#	Co_name	Co_college
20ECES212	Programming in C++	Engineering
22IS270	Principles of IS	Business
22IS330	Database Concepts	Business
22IS832	Database Principles	Business
22QA375	Operations Research	Business

subquery executed first

subqueries

display the section number and course number for which at least one grade of "A" has been assigned

```
SELECT DISTINCT SECTION.SE_SECTION#,
SECTION.SE_CO_COURSE# FROM SECTION WHERE
(SECTION.SE_SECTION#, SECTION.SE_CO_COURSE#)
IN (SELECT TAKES.TK_SE_SECTION#,
TAKES.TK_SE_CO_COURSE#
FROM TAKES WHERE TAKES.TK_GRADE = 'A');
```

```
Se_section# Se_co_course#
----------- -------------
        101 22IS330
        101 22QA375
        701 22IS832
```

subqueries

```
SELECT TAKES.TK_ST_SID, STUDENT.ST_NAME,
COUNT(*) AS "Sections Taken" FROM STUDENT JOIN
TAKES ON STUDENT.ST_SID = TAKES.TK_ST_SID
GROUP BY TAKES.TK_ST_SID, STUDENT.ST_NAME
UNION
SELECT STUDENT.ST_SID, STUDENT.ST_NAME, 0 FROM
STUDENT WHERE STUDENT.ST_SID NOT IN
(SELECT TAKES.TK_ST_SID FROM TAKES)
ORDER BY "Sections Taken" DESC;
```

what is the purpose of the 0?

subqueries - use of ANY and ALL

Operator	Description
> ALL	Greater than the highest value returned by the subquery
>= ALL	Greater than or equal to the highest value returned by the subquery
< ALL	Less than the lowest value returned by the subquery
<= ALL	Less than or equal to the lowest value returned by the subquery
> ANY	Greater than the lowest value returned by the subquery

Operator	Description
>= ANY	Greater than or equal to the lowest value returned by the subquery
< ANY	Less than the highest value returned by the subquery
<= ANY	Less than or equal to the highest value returned by the subquery
= ANY	Equal to any value returned by the subquery (same as the IN operator)

subqueries - use of ANY and ALL

display the names and salaries of those professors who earn more than all professors in department number 3

```
SELECT PROFESSOR.PR_NAME, PROFESSOR.PR_SALARY
FROM PROFESSOR WHERE PROFESSOR.PR_SALARY > ALL
  (SELECT PROFESSOR.PR_SALARY FROM PROFESSOR
    WHERE PROFESSOR.PR_DPT_DCODE = 3);
```

Pr_name	Pr_salary
Mike Faraday	92000
Marie Curie	99000
John Nicholson	99000

subqueries - use of ANY and ALL

display the names and salaries of those professors who earn as much or more than the highest paid professor in department number 3

```
SELECT PROFESSOR.PR_NAME, PROFESSOR.PR_SALARY
 FROM PROFESSOR WHERE PROFESSOR.PR_SALARY >=
ALL (SELECT PROFESSOR.PR_SALARY FROM PROFESSOR
     WHERE PROFESSOR.PR_DPT_DCODE = 3);
```

Pr_name	Pr_salary
Mike Faraday	92000
Chelsea Bush	77000
Tony Hopkins	77000
Marie Curie	99000
John Nicholson	99000

subqueries - use of ANY and ALL

display the names and salaries of those professors who earn less than all professors in department number 7

```
SELECT PROFESSOR.PR_NAME, PROFESSOR.PR_SALARY
FROM PROFESSOR WHERE PROFESSOR.PR_SALARY < ALL
   (SELECT PROFESSOR.PR_SALARY FROM PROFESSOR
      WHERE PROFESSOR.PR_DPT_DCODE = 7);
```

Pr_name	Pr_salary
Ram Raj	44000
Prester John	44000
Laura Jackson	43000

subqueries - use of ANY and ALL

display the names and salaries of those professors who earn as much or more than the lowest-paid professor in department 3

```
SELECT PROFESSOR.PR_NAME, PROFESSOR.PR_SALARY
 FROM PROFESSOR WHERE PROFESSOR.PR_SALARY >=
ANY(SELECT PROFESSOR.PR_SALARY FROM PROFESSOR
    WHERE PROFESSOR.PR_DPT_DCODE = 3);
```

subqueries - use of ANY and ALL

Pr_name	Pr_salary
John Smith	45000
Mike Faraday	92000
Kobe Bryant	66000
Ram Raj	44000
Prester John	44000
Chelsea Bush	77000
Tony Hopkins	77000
Alan Brodie	76000
Jessica Simpson	67000
Laura Jackson	43000
Marie Curie	99000
Jack Nicklaus	67000
John Nicholson	99000
Sunil Shetty	64000
Katie Shef	65000
Cathy Cobal	45000
Jeanine Troy	45000
Mike Crick	69000

null salaries not included

subqueries - MIN and MAX functions

display the names and salaries of those professors who earn more than the highest paid professor in department number 3

```
SELECT PROFESSOR.PR_NAME, PROFESSOR.PR_SALARY
  FROM PROFESSOR WHERE PROFESSOR.PR_SALARY >
    (SELECT MAX(PROFESSOR.PR_SALARY) FROM
PROFESSOR WHERE PROFESSOR.PR_DPT_DCODE = 3);
```

Pr_name	Pr_salary
Mike Faraday	92000
Marie Curie	99000
John Nicholson	99000

subqueries - MIN and MAX functions

display the names and salaries of those professors who earn more than the lowest paid professor in department number 3

```
SELECT PROFESSOR.PR_NAME, PROFESSOR.PR_SALARY
  FROM PROFESSOR WHERE PROFESSOR.PR_SALARY >
    (SELECT MIN(PROFESSOR.PR_SALARY) FROM
PROFESSOR WHERE PROFESSOR.PR_DPT_DCODE = 3);
```

Pr_name	Pr_salary
John Smith	45000
Ram Raj	44000
Prester John	44000
Laura Jackson	43000
Cathy Cobal	45000
Jeanine Troy	45000

subqueries - HAVING clause

display all departments with an average salary that exceeds the average salary of all professors

```
SELECT DEPARTMENT.DPT_NAME,
AVG(PROFESSOR.PR_SALARY) FROM DEPARTMENT JOIN
PROFESSOR ON DEPARTMENT.DPT_DCODE =
PROFESSOR.PR_DPT_DCODE
GROUP BY DEPARTMENT.DPT_NAME
HAVING AVG(PROFESSOR.PR_SALARY) >
(SELECT AVG(PROFESSOR.PR_SALARY) FROM
PROFESSOR);
```

DPT_NAME	AVG(PROFESSOR.PR_SALARY)
Economics	78000
QA/QM	68000

aggregate functions and grouping

display the maximum, minimum, total, and average salary for the professors affiliated with each department. in addition, count the number of professors in each department as well as the number of professors in each department with a not null salary

aggregate functions and grouping

```
SELECT DEPARTMENT.DPT_NAME "Dept Name”,
    DEPARTMENT.DPT_DCODE "Dept Code",
  MAX(PROFESSOR.PR_SALARY) "Max Salary",
  MIN(PROFESSOR.PR_SALARY) "Min Salary”,
 SUM (PROFESSOR.PR_SALARY) "Total Salary”,
ROUND(AVG(PROFESSOR.PR_SALARY),0) “Avg Salary",
                COUNT(*) "Size",
      COUNT(PROFESSOR.PR_SALARY) "# Sals"
       FROM DEPARTMENT JOIN PROFESSOR ON
DEPARTMENT.DPT_DCODE = PROFESSOR.PR_DPT_DCODE
        GROUP BY DEPARTMENT.DPT_NAME,
            DEPARTMENT.DPT_DCODE
            ORDER BY 6 DESC;
```

what does the 6 mean?

aggregate functions and grouping

Dept Name	Dept Code	Max Salary	Min Salary	Total Salary	Avg Salary	Size	# Sals
Economics	4	99000	67000	265000	88333	3	3
QA/QM	3	77000	43000	340000	68000	5	5
Economics	1	92000	45000	203000	67667	3	3
IS	7	65000	45000	174000	58000	3	3
Philosophy	9	69000	45000	114000	57000	3	2
Mathematics	6	44000	44000	88000	44000	3	2

ordered by
average salary

correlated subquery

correlated subqueries make use of the EXISTS operator which returns the value of true if a set is non-empty

execution stops and the condition of the main query is declared true for a given row if the condition in the subquery is true

correlated subquery

display the names of professors who have offered at least one section

```
SELECT PROFESSOR.PR_NAME FROM PROFESSOR WHERE
    EXISTS (SELECT * FROM SECTION WHERE
PROFESSOR.PR_EMPID = SECTION.SE_PR_PROFID);
```

```
Pr_name
---------------
Ram Raj
Tony Hopkins
Katie Shef
Cathy Cobal
```

can also use NOT EXISTS

PROFESSOR Relation

Pr_name	Pr_empid	Pr_phone	Pr_office	Pr_birthdate	Pr_datehired	Pr_dpt_dcode	Pr_salary
John Smith	SJ89324	5235567645	223 McMicken	1966-10-12	2001-06-23	1	45000
Mike Faraday	FM49276	5235568492	249 McMicken	1960-08-26	1996-05-01	1	92000
Kobe Bryant	BK68765	5235568522	322 McMicken	1968-03-02	1998-05-01	1	66000
Ram Raj	RR79345	5235567244	822 Old Chem	1970-02-06	2001-06-23	6	44000
John B Smith	SJ65436	5235567556	838 Old Chem			6	
Prester John	JP77869	5235567244	822 Old Chem	1955-08-25	1995-08-25	6	44000
Chelsea Bush	BC65437	5235567777	227 Lindner	1946-09-03	1993-05-01	3	77000
Tony Hopkins	HT54347	5235569977	324 Lindner	1949-11-24	1997-01-20	3	77000
Alan Brodie	BA54325	5235569876	238 Lindner	1944-01-14	2000-05-16	3	76000
Jessica Simpson	SJ67543	5235565567	324 Lindner	1955-08-25	1995-08-25	3	67000
Laura Jackson	JL65436	5235565436	336 Lindner	1973-10-16	2000-09-23	3	43000
Marie Curie	CM65436	5235569899	331 Dyer	1972-02-29	1999-10-22	4	99000
Jack Nicklaus	NJ33533	5235566767		1976-01-01	1999-12-31	4	67000
John Nicholson	NJ43728	5235569999	324 Dyer	1966-05-01	2003-06-22	4	99000
Sunil Shetty	SS43278	5235566764	526 Lindner		1993-06-28	7	64000
Katie Shef	SK85977	5235568765	572 Lindner	1948-08-08	1997-06-06	7	65000
Cathy Cobal	CC49234	5235565345	544 Lindner	1968-02-28	2001-01-23	7	45000
Jeanine Troy	TJ76546	5235565545	423 McMicken	1968-01-16		9	45000
Tiger Woods	WT65487	5235565563		1975-11-14	2003-11-14	9	
Mike Crick	CM87659	5235565569	444 McMicken	1970-05-31	2002-05-30	9	69000

SECTION Relation

Se_section#	Se_qtr	Se_year	Se_time	Se_maxst	Se_room	Se_co_course#	Se_pr_profid
101	A	2007	T1015	25		22QA375	HT54347
901	A	2006	W1800	35	Rhodes 611	22IS270	SK85977
902	A	2006	H1700	25	Lindner 108	22IS270	SK85977
101	S	2006	T1045	29	Lindner 110	22IS330	SK85977
102	S	2006	H1045	29	Lindner 110	22IS330	CC49234
701	W	2007	M1000	33	Braunstien 211	22IS832	CC49234
101	A	2007	W1800		Baldwin 437	20ECES212	RR79345
101	U	2007	T1015	33		22QA375	HT54347
101	A	2007	H1700	29	Lindner 108	22IS330	SK85977
101	S	2007	T1015	30		22QA375	HT54347
101	W	2007	T1015	20		22QA375	HT54347