Appendix 2. Contents Presidential Database

PRESIDENT

PRES_NAME	BIRTH_YR	YRS_SERV	DEATH_AGE	PARTY	STATE_BORN
Washington G	1732	7	67	Federalist	Virginia
Adams J	1735	4	90	Federalist	Massachusetts
Jefferson T	1743	8	83	Demo-Rep	Virginia
Madison J	1751	8	85	Demo-Rep	Virginia
Monroe J	1758	8	73	Demo-Rep	Virginia
Adams J Q	1767	4	80	Demo-Rep	Massachusetts
Jackson A	1767	8	78	Democratic	South Carolina
Van Buren M	1782	4	79	Democratic	New York
Harrison W H	1773	0	68	Whig	Virginia
Tyler J	1790	3	71	Whig	Virginia
Polk J K	1795	4	53	Democratic	North Carolina
Taylor Z	1784	1	65	Whig	Virginia
Fillmore M	1800	2	74	Whig	New York
Pierce F	1804	4	64	Democratic	New Hampshire
Buchanan J	1791	4	77	Democratic	Pennsylvania
Lincoln A	1809	4	56	Republican	Kentucky
Johnson A	1808	3	66	Democratic	North Carolina
Grant U S	1822	8	63	Republican	Ohio
Hayes R B	1822	4	70	Republican	Ohio
Garfield J A	1831	0	49	Republican	Ohio
Arthur C A	1830	3	56	Republican	Vermont
Cleveland G	1837	8	71	Democratic	New Jersey
Harrison B	1833	4	67	Republican	Ohio
McKinley W	1843	4	58	Republican	Ohio
Roosevelt T	1858	7	60	Republican	New York
Taft W H	1857	4	72	Republican	Ohio
Wilson W	1856	8	67	Democratic	Vermont
Harding W G	1865	2	57	Republican	Iowa
Coolidge C	1872	5	60	Republican	New York
Hoover H C	1874	4	90	Republican	Missouri
Roosevelt F D	1882	12	63	Democratic	Texas
Truman H S	1884	7	88	Democratic	Massachusetts
Eisenhower D D	1890	8	79	Republican	Texas
Kennedy J F	1917	2	46	Democratic	California
Johnson L B	1908	5	65	Democratic	Texas
Nixon R M	1913	5	-	Republican	California
Ford G R	1913	2	-	Republican	Nebraska
Carter J E	1924	4	-	Democratic	Georgia
Reagan R	1911	3	-	Republican	Illinois

PRES_MARRIAGE

PRES_NAME	SPOUSE_NAME	PR_AGE	SP_AGE	NR_CHILDREN	MAR_YEAR
Washington G	Custis M D	26	27	0	1759
Adams J	Smith A	28	19	5	1764
Jefferson T	Skelton M W	28	23	6	1762
Madison J	Todd D D P	43	26	3	1794
Monroe J	Kortright l	27	17	3	1786
Adams J Q	Johnson L C	30	22	4	1797
Jackson A	Robards R D	26	26	0	1794
Van Buren M	Hoes H	24	23	4	1807
Harrison W H	Symmes A T	22	20	10	1795
Tyler J	Christian L	23	22	8	1813
Tyler J	Gardiner J	54	24	7	1844
Polk J K	Childress S	28	20	0	1824
Taylor Z	Smith M M	25	21	6	1810
Fillmore M	Power A	26	27	2	1826
Fillmore M	McIntosh C C	58	44	0	1858
Pierce F	Appleton J M	29	28	3	1834
Lincoln A	Todd M	33	23	4	1842
Johnson A	McCardle E	18	16	5	1827
Grant U S	Dent J B	26	22	4	1848
Hayes R B	Webb L W	30	21	8	1852
Garfield J A	Rudolhp L	26	26	7	1858
Arthur C A	Herndon E L	29	22	3	1859
Cleveland G	Folson F	49	21	5	1886
Harrison B	Scott C L	20	21	2	1853
Harrison B	Dimmick M S L	62	37	1	1896
McKinley W	Saxton I	27	23	2	1871
Roosevelt T	Lee A H	22	19	1	1880
Roosevelt T	Carow E K	28	25	5	1886
Taft W H	Herron H	28	25	3	1886
Wilson W	Axson E L	28	25	3	1885
Wilson W	Galt E B	58	43	0	1915
Harding W G	De Wolfe F K	25	30	0	1891
Coolidge C	Goodhue G A	33	26	2	1905
Hoover H C	Henry L	24	23	2	1899
Roosevelt F D	Roosevelt A E	23	20	6	1905
Truman H S	Wallace E V	35	34	1	1919
Eisenhower D D	Doud G	25	19	2	1916
Kennedy J F	Bouvier J L	36	24	3	1953
Johnson L B	Taylor C A	26	21	2	1934
Nixon R M	Ryan T C	27	28	2	1940
Ford G R	Warren E B	35	30	4	1948
Carter J E	Smith R	21	18	4	1946
Reagan R	Wyman J	28	25	2	1940
Reagan R	Davis N	41	28	2	1952

PRES-HOBBY

PRES_NAME	HOBBY
Adams J Q	Billiards
	Swimming
	Walking
Arthur C A	Fishing
Cleveland G	Fishing
Coolidge C	Fishing
	Golf
	Indian Clubs
	Mechanical Horse
	Pitching Hay
Eisenhower D D	Bridge
	Golf
	Hunting
	Painting
	Fishing
Garfield J A	Billiards
Harding W G	Golf
	Poker
	Riding
Harrison B	Hunting
Hayes R B	Croquet
	Driving
	Shooting
Hoover H C	Fishing
	Medicine Ball
Jackson A	Riding
Jefferson T	Fishing
	Riding
Johnson L B	Riding
Kennedy J F	Sailing
	Swimming
	Touch Football
Lincoln A	Walking

PRES_NAME	HOBBY
McKinley W	Riding
	Swimming
	Walking
Nixon R M	Golf
Roosevelt F D	Fishing
	Sailing
	Swimming
Roosevelt T	Boxing
	Hunting
	Jujitsu
	Riding
	Shooting
	Tennis
	Wrestling
Taft W H	Golf
	Riding
Taylor Z	Riding
Truman H S	Fishing
	Poker
	Walking
Van Buren M	Riding
Washington G	Fishing
	Riding
Wilson W	Golf
	Riding
	Walking

ADMINISTRATION

ADMIN_NR.	PRES_NAME	YEAR_INAUGURATED
1	Washington G	1789
2	Washington G	1793
3	Adams J	1797
4	Jefferson T	1801
5	Jefferson T	1805
6	Madison J	1809
7	Madison J	1813
8	Monroe J	1817
9	Monroe J	1821
10	Adams J Q	1825
11	Jackson A	1829
12	Jackson A	1833
13	Van Buren M	1837
14	Harrison W H	1841
14	Tyler J	1841
15	Polk J K	1845
16	Taylor Z	1849
16	Fillmore M	1850
17	Pierce F	1853
18	Buchanan J	1857
19	Lincoln A	1861
20	Lincoln A	1865
20	Johnson A	1865
21	Grant U S	1869
22	Grant U S	1873
23	Hayes R B	1877
24	Garfield J A	1881
24	Arthur C A	1881
25	Cleveland G	1885
26	Harrison B	1889
27	Cleveland G	1893
28	McKinley W	1897
29	McKinley W	1901
29	Roosevelt T	1901
30	Roosevelt T	1905
31	Taft W H	1909
32	Wilson W	1913
33	Wilson W	1917
34	Harding W G	1921
34	Coolidge C	1923
35	Coolidge C	1925
36	Hoover H C	1929
37	Roosevelt F D	1933
38	Roosevelt F D	1937
39	Roosevelt F D	1941
40	Roosevelt F D	1945

ADMINISTRATION (contd.)

ADMIN_NR.	PRES_NAME	YEAR_INAUGURATED
40	Truman H S	1945
41	Truman H S	1949
42	Eisenhower D D	1953
43	Eisenhower D D	1957
44	Kennedy J F	1961
44	Johnson L B	1963
45	Johnson L B	1965
46	Nixon R M	1969
47	Nixon R M	1973
47	Ford G R	1974
48	Carter J E	1977
49	Reagan R	1981

ADMIN-PR-VP

ADMIN_NR.	PRES_NAME	VICE_PRES_NAME
1	Washington G	Adams J
2	Washington G	Adams J
3	Adams J	Jefferson T
4	Jefferson T	Burr A
5	Jefferson T	Clinton G
6	Madison J	Clinton G
7	Madison J	Gerry E
8	Monroe J	Tompkins D
9	Monroe J	Tompkins D
10	Adams J Q	Calhoun J
11	Jackson A	Calhoun J
12	Jackson A	Van Buren M
13	Van Buren M	Johnson R M
14	Harrison W H	Taylor J
15	Polk J K	Dallas G M
16	Taylor Z	Fillmore M
17	Pierce F	De Vane King W R
18	Buchanan J	Breckinridge J C
19	Lincoln A	Hamlin H
20	Lincoln A	Johnson A
21	Grant U S	Colfax S
22	Grant U S	Wilson H
23	Hayes R B	Wheeler W
24	Garfield J A	Arthur C A
25	Cleveland G	Hendricks T A
26	Harrison B	Morton L P
27	Cleveland G	Stevenson A E
28	McKinley W	Hobart G A
29	McKinley W	Roosevelt T
30	Roosevelt T	Fairbanks C W

ADMIN-PR-VP {contd}

ADMIN_NR	PRES_NAME	VICE_PRES_NAME
31	Taft W H	Sherman J S
32	Wilson W	Marshall T R
33	Wilson W	Marshall T R
34	Harding W G	Coolidge C
35	Coolidge C	Dawes C G
36	Hoover H C	Curtis C
37	Roosevelt F D	Garner J N
38	Roosevelt F D	Garner J N
39	Roosevelt F D	Wallace H A
40	Roosevelt F D	Truman H S
41	Truman H S	Barkley A W
42	Eisenhower D D	Nixon R M
43	Eisenhower D D	Nixon R M
44	Kennedy J F	Johnson L B
45	Johnson L B	Humphrey H H
46	Nixon R M	Agnew S T
47	Nixon R M	Agnew S T
47	Nixon R M	Ford G R
47	Ford G R	Rockefeller N A
48	Carter J E	Mondale W F
49	Reagan R	Bush G

50 R B

50 X B

select

STATE

STATE_NAME	ADMIN_ENTERED	YEAR_ENTERED
Massachusetts	-	1776
Pennsylvania	-	1776
Virginia	-	1776
Connecticut	-	1776
South Carolina	-	1776
Maryland	-	1776
New Jersey	-	1776
Georgia	-	1776
New Hampshire	-	1776
Delaware	-	1776
New York	-	1776
North Carolina	-	1776
Rhode Island	-	1776
Vermont	1	1791
Kentucky	1	1792
Tennessee	2	1796
Ohio	4	1803
Louisianna	6	1812
Indiana	7	1816
Mississippi	8	1817
Illinois	8	1818
Alabama	8	1819
Maine	8	1820
Missouri	9	1821
Arkansas	12	1836
Michigan	12	1837
Florida	14	1845
Texas	15	1845
Iowa	15	1846
Wisconsin	15	1848
California	16	1850
Minnesota	18	1858
Oregon	18	1859
Kansas	18	1861
West Virginia	19	1863
Nevada	19	1864
Nebraska	20	1867
Colorado	22	1876
North Dakota	26	1889
South Dakota	26	1889
Montana	26	1889
Washington	26	1889
Idaho	26	1890
Wyoming	26	1890
Utah	27	1896
Oklahoma	30	1907
New Mexico	31	1912
Arizona	31	1912
Alaska	43	1959
Hawaii	43	1959

ELECTION

ELECTION_YEAR	CANDIDATE	VOTES	WINNER_LOSER_INDIC
1789	Washington G	69	W
	Adams J	34	L
	Jay J	9	L
	Harrison R H	6	L
	Rutledge J	6	L
	Hancock J	4	L
	Clinton G	3	L
	Huntington S	2	L
	Milton J	2	L
	Armstrong J	1	L
	Lincoln B	1	L
	Telfair E	1	L
1792	Washington G	132	W
	Adams J	77	L
	Clinton G	50	L
	Jefferson T	4	L
	Burr A	1	L
1796	Adams J	71	W
	Jefferson T	68	L
	Pinckney T	59	L
	Burr A	30	L
	Adams S	15	L
	Ellsworth O	11	L
	Clinton G	7	L
	Jay J	5	L
	Iredell J	3	L
	Henry J	2	L
	Johnson S	2	L
	Washington G	2	L
	Pinckney C C	1	L
1800	Jefferson T	73	W
	Burr A	73	L
	Adams J	65	L
	Pinckney C C	64	L
	Jay J	1	L
1804	Jefferson T	162	W
	Pinckney C C	14	L
1808	Madison J	122	W
	Pinckney C C	47	L
	Clinton G	6	L
1812	Madison J	128	W
	Clinton G	89	L
1816	Monroe J	183	W
	King R	34	L
1820	Monroe J	231	W
	Adams J Q	1	L
1824	Adams J Q	84	W
	Jackson A	99	L
	Crawford W H	41	L
	Clay H	37	L

ELECTION (contd)

ELECTION_YEAR	CANDIDATE	VOTES	WINNER_LOSER_INDIC
1828	Jackson A	178	W
	Adams J	83	L
1832	Jackson A	219	W
	Clay H	49	L
	Floyd J	11	L
	Wirt W	7	L
1836	Van Buren M	170	W
	Harrison W H	73	L
	White H L	26	L
	Webster D	14	L
	Mangum W P	11	L
1840	Harrison W H	234	W
	Van Buren M	60	L
1844	Polk J K	170	W
	Clay H	105	L
1848	Taylor Z	163	W
	Cass L	127	L
1852	Pierce F	254	W
	Scott W	42	L
1856	Buchanan J	174	W
	Fremont J C	114	L
	Fillmore M	8	L
1860	Lincoln A	180	W
	Breckinridge J	72	L
	Bell J	39	L
	Douglas S	12	L
1864	Lincoln A	212	W
	McClellan G B	21	L
1868	Grant U S	214	W
	Seymour H	80	L
1872	Grant U S	286	W
	Hendricks T A	42	L
	Brown B G	18	L
	Jenkins C J	2	L
	Davis D	1	L
1876	Hayes R B	185	W
	Tilden S J	184	L
1880	Garfield J A	214	W
	Hancock W S	155	L
1884	Cleveland G	219	W
	Blaine J G	182	L
1888	Harrison B	233	W
	Cleveland G	168	L
1892	Cleveland G	277	W
	Harrison B	145	L
	Weaver J B	22	L
1896	McKinley W	271	W
	Bryan W J	176	L
1900	McKinley W	292	W
	Bryan W J	155	L

ELECTION (contd)

ELECTION YEAR	CANDIDATE	VOTES	WINNER LOSER INDIC
1904	Roosevelt T	336	W
	Parker A B	140	L
1908	Taft W H	321	W
	Bryan W J	162	L
1912	Wilson W	435	W
	Roosevelt T	88	L
	Taft W H	8	L
1916	Wilson W	277	W
	Hughes C E	254	L
1920	Harding W G	404	W
	Cox W W	127	L
1924	Coolidge C	382	W
	Davis J W	136	L
	La Follette R M	13	L
1928	Hoover H C	444	W
	Smith A E	87	L
1932	Roosevelt F D	472	W
	Hoover H C	59	L
1936	Roosevelt F D	523	W
	Landon A M	8	L
1940	Roosevelt F D	449	W
	Wilkie W L	82	L
1944	Roosevelt F D	432	W
	Dewey T E	99	L
1948	Truman H S	303	W
	Dewey T E	189	L
	Thurmond J S	39	L
1952	Eisenhower D D	442	W
	Stevenson A	89	L
1956	Eisenhower D D	457	W
	Stevenson A	73	L
	Jones W B	1	L
1960	Kennedy J F	303	W
	Nixon R M	219	L
—	Byrd	15	L
1964	Johnson L B	486	W
	Goldwater B	52	L
1968	Nixon R M	301	W
	Humphrey H H	191	L
	Wallace G C	46	L
1972	Nixon R M	520	W
	McGovern G S	17	L
	Hospers J	1	L
1976	Carter J E	297	W
	Ford G R	240	L
1980	Reagan R	489	W
	Carter J E	49	L

Chapter 5. Selecting All or Particular Columns from One Table

The simplest case is to list a whole table. The format of this retrieval command is

```
SELECT   *
FROM     table-name
```

The asterisk * indicates all columns in the table.

Many of the queries that follow will operate on the following table:

RECENT_PRESIDENTS

PRES_NAME	BIRTH_YR	YRS_SERV	DEATH_AGE	PRATY	STATE_BORN
Roosevelt F D	1882	12	63	Democratic	New York
Truman H S	1884	7	88	Democratic	Missouri
Eisenhower D D	1890	8	79	Republican	Texas
Kennedy J F	1917	2	46	Democratic	Massachusetts
Johnson L B	1908	5	65	Democratic	Texas
Nixon R M	1913	5	?	Republican	California
Ford G R	1913	2	?	Republican	Nebraska
Carter J E	1924	4	?	Democratic	Georgia
Reagan R	1911	3	?	Republican	Illinois

Figure 5.1

Before formulating a query we need to know the table names and column names and their data type. It is useful to have a table template of relational schema diagram containing all the table names and column names. For our example of RECENT_PRESIDENTS this template is given in Figure 5.2

TEMPLATE

RECENT_PRESIDENTS ⇦ table name

column name ⇩

PRES_NAME	BIRTH_YR	YRS_SERV	DEATH_AGE	PRATY	STATE_BORN
CHAR(15)	SMALLINT	SMALLINT	SMALLINT	CHAR(12)	VARCHAR(14)

⇧ data type

Figure 5.2

Question 5.02

List the names, birth years, ages at death (if any), years served, birth states, and parties of recent presidents.

```
SELECT    PRES_NAME , BIRTH_YR , DEATH_AGE , YRS_SERV , -
          STATE_BORN , PARTY -
FROM      RECENT_PRESIDENTS
```

Result:

PRES_NAME	BIRTH_YR	DEATH_AGE	YRS_SERV	STATE_BORN	PRATY
Roosevelt F D	1882	63	12	New York	Democratic
Truman H S	1884	88	7	Missouri	Democratic
Eisenhower D D	1890	79	8	Texas	Republican
Kennedy J F	1917	46	2	Massachusetts	Democratic
Johnson L B	1908	65	5	Texas	Democratic
Nixon R M	1913	?	5	California	Republican
Ford G R	1913	?	2	Nebraska	Republican
Carter J E	1924	?	4	Georgia	Democratic
Reagan R	1911	?	3	Illinois	Republican

* END OF RESULT ***** 9 ROWS DISPLAYED **********

Ordering rows:

We might also wish to have the rows in a specific order independent of the order in the tables of the database. To achieve that, we have to append an ORDER BY clause to the query. There can be one or several ordering criteria in an ORDER BY clause. An ordering criterion is an attribute of the table(s) to be retrieved together with an indicator whether the table should be ordered in an ascending or descending lexicographical order of that column. The format of an ORDER BY clause is:

```
ORDER BY  column-specification[ASC | DESC ]
          [,column-specification[ASC | DESC ] ...]
```

If DESC is present, the order is defined ads descending, otherwise ascending.

Question 5.04

List the table named RECENT_PRESIDENTS, ordered by years served in descending order, and within the same years served, ordered by birth state in ascending order.

```
SELECT    * –
FROM      RECENT_PRESIDENTS –
ORDER BY  YRS_SERV DESC , STATE_BORN
```

The first ordering criterion is YRS_SERV. However, since there are presidents with the same years-served figure, we decided to have a second ordering criterion, birth state.

Result:

PRES_NAME	BIRTH_YR	YRS_SERV	DEATH_AGE	PRATY	STATE_BORN
Roosevelt F D	1882	12	63	Democratic	New York
Eisenhower D D	1890	8	79	Republican	Texas
Truman H S	1884	7	88	Democratic	Missouri
Nixon R M	1913	5	?	Republican	California
Johnson L B	1908	5	65	Democratic	Texas
Carter J E	1924	4	?	Democratic	Georgia
Reagan R	1911	3	?	Republican	Illinois
Kennedy J F	1917	2	46	Democratic	Massachusetts
Ford G R	1913	2	?	Republican	Nebraska

* END OF RESULT ***** 9 ROWS DISPLAYED **********

first ordering criterion — second ordering criterion

The combination of years and state born is unique in this case, but only by chance. If it were not unique, the order within a years-served/state-born bracket would be undefined, since we did not specify a third ordering criterion.

Note:

In the ORDER BY clause, one can use the column name of the position of the column in the SELECT list counting from left to right.

Thus, the question 5.04 could also be formulated as follows:

```
SELECT    * –
FROM      RECENT_PRESIDENTS –
ORDER BY  3 DESC , 6
```

Question 5.06

List all states where a recent president was born.
Order by state (ascending order)

```
SELECT      STATE_BORN –
FROM        RECENT_PRESIDENTS –
ORDER BY    STATE_BORN
```

Result:

```
STATE_BORN
--------------------
California
Georgia
Illinois
Massachusetts
Missouri
Nebraska
New York
Texas
Texas
* END OF RESULT ****** 9 ROWS DISPLAYED **************************
```

We find in this result that there are duplicate rows, namely, the last two rows are both Texas. The same result would have been achieved by using

```
SELECT      ALL STATE_BORN –
FROM        RECENT_PRESIDENT –
ORDER BY    STATE_BORN
```

If we wish, however, to avoid these duplicates, we have to use the DISTINCT operator, as in the following Question 5.07

Question 5.07

List all states where a recent president was born, and eliminate supplicates, Order by state.

```
SELECT    DISTINCT STATE_BORN –
FROM      RECENT_PRESIDENTS
ORDER BY  STATE_BORN
```

Result:

```
STATE_BORN
--------------------
California
Georgia
Illinois
Massachusetts
Missouri
Nebraska
New York
Texas
* END OF RESULT ****** 8 ROWS DISPLAYED **************************
```

Chapter 6. Selecting Specified Rows of One Table

For selecting specific rows of a table, that is for horizontal subsetting, we need to specify a WHERE clause after the table name. The condition in the WHERE clause is a comparison of fields of a column of the table with a constant or with constants or with fields of another column or other columns. To start with we now will consider only the simple case that these other columns are in the same table.

A further distinction is whether the second part of the comparison is a constant or a single field of a column, or whether it is a set of constants or a set of fields of a column. In the first and simpler case, we can apply the following comparison operators:

=	**equal to**
^ =	**not equal to**
>	**greater than**
>=	**greater than or equal to**
<	**less than**
<=	**less than or equal to**

These symbols may vary slightly depending on the equipment used.

Question 6.01

List all facts available in the table named RECENT_PRESIDENT about President Carter J E.

```
SELECT    * –
FROM      RECENT_PRESIDENTS –
WHERE     PRES_NAME = 'Carter J E'
```

Note 1:

Upper and lower case characters are determined by the SET command.

SET CASE { UPper | STRing }

If UPPER is specified then all characters entered from the keyboard or a routine that are enclosed in quotes will not be converted to uppercase characters.

If STRING is specified then all characters entered from the keyboard of a routine that are enclosed in quotes will not be converted to uppercase characters.

When ISQL is started then the system is SET to UPPER. To keep the examples in this book as natural as possible we have used the natural way of upper and lower case letters. Therefore the user needs to SET the CASE option to STRING.

Note 2:

A constant containing characters (e.g. 'Carter J E') must be enclosed in single quotes. If the constant is a numeric, no quotes are required.

Result:

PRES_NAME	BIRTH_YR	YRS_SERV	DEATH_AGE	PARTY	STATE_BORN
Carter J E	1924	4	7	Democratic	Georgia

* ENE OF RESULT ***** 1 ROWS DISPLAYS **********

In the above example, we used a key column of the table for comparison using =. In such a case, the result is either one row, or empty. If we use a non-key column for comparison applying =, the situation would be different. The result could be empty, one or several rows.

Question 6.02

List all facts available in the table named RECENT_PRESIDENTS about all presidents born in Texas.

```
SELECT    * –
FROM      RECENT_PRESIDENTS –
WHERE     STATE_BORN = 'Texas'
```

Result:

PRES_NAME	BIRTH_YR	YRS_SERV	DEATH_AGE	PARTY	STATE_BORN
Eisenhower D D	1890	8	79	Republican	Texas
Johnson L B	1908	5	65	Democratic	Texas

* ENE OF RESULT ***** 2 ROWS DISPLAYS **********

In the cases of the other kinds of comparisons (^ = , > , > = , < , < =), the result could be empty, one or several rows, independent of whether we use a key column of a non-key column for comparison.

SQL logical Execution Sequence ลำดับ:
1) From
2) Where

Question 6.07

List all facts available in the table named RECENT_PRESIDENTS about all presidents who are Republican and not born in Texas.

```
SELECT    *-
FROM      RECENT_PRESIDENTS -
WHERE     PARTY = 'Republican' -
          AND NOT STATE_BORN = 'Texas'
```

Equivalent formulation with ^= :

```
SELECT    *-
FROM      RECENT_PRESIDENTS -
WHERE     PARTY = 'Republican' -
          AND STATE_BORN ^= 'Texas'
```

Result:

PRES_NAME	BIRTH_YR	YRS_SERV	DEATH_AGE	PARTY	STATE_BORN
Nixon R M	1913	5	?	Republican	California
Ford G R	1913	2	?	Republican	Nebraska
Reagan R	1911	3	?	Republican	Illinois

* ENE OF RESULT ***** 3 ROWS DISPLAYS **********

from PRESIDENT
select * where PARTY='Republican' Or Party='Democrat'

SP
S# P# QTY
list S# ที่ supply ทั้ง P1 และ P2 (มีมากกว่าก็ได้) ต้องใช้ function ด้วย
ต้อง Group by กับ having ด้วย.

```
Slct S#
from SP
where P# = 'P1' or P# = 'P2'
group by S#
having count(*) > 1
```

Question 6.09

List all facts available in the table named RECENT_PRESIDENTS about presidents born in Texas, California, Georgia or New York.

First possibility, without set comparison operator:

```
SELECT  * –
FORM    RECENT_PRESIDENTS –
WHERE   STATE_BORN = 'Texas' –
        OR STATE_BORN = 'California' –
        OR STATE_BORN = 'Georgia' –
        OR STATE_BORN = 'New York' –
```

Second possibility, with set comparison operator IN;

```
SELECT  * –
FORM    RECENT_PRESIDENTS –
WHERE   STATE_BORN IN ('Texas' , 'California' , 'Georgia' , 'New York')
```

Note:

If we use a set comparison operator like IN (or = ANY) in connection with a list of constants, this list must contain at least two elements.

Result:

PRES_NAME	BIRTY_YR	YRS_SERV	DEATH_AGE	PARTY	STATE_BONRN
Roosevelt F D	1882	12	63	Democratic	New York
Eisenhower D D	1890	8	97	Republican	Texas
Johnson L B	1908	5	65	Democratic	Texas
Nixon R M	1913	5	?	Republican	California
Cater J E	1924	4	?	Democratic	Georgia

* END OF RESULT ****** 5 ROWS DISPLAYED **********

The issue of set comparison needs more elaboration which will be given in chapter 11.

A few special operators are available for convenient formulation of special comparisons. For example, there is the BETWEEN operator, testing whether a value is within a specified range of values (constants), or not.

```
.....column-name BETWEEN { constant  } AND { constant  }
                         { expression }     { expression }
```

Question 6.10

List all facts available in the table named RECENT_PRESIDENTS about all presidents who died at an age between 60 and 70 years.

Formulation without BETWEEN operator:

```
SELECT    * –
FROM      RECENT_PRESIDENTS –
WHERE     DEATH_AGE > = 60 AND DEATH_AGE < = 70
```

Formulation without BETWEEN operator:

```
SELECT    * –
FROM      RECENT_PRESIDENTS –
WHERE     DEATH_AGE BETWEEN 60 AND 70
```

Note:

The BETWEEN A AND B function is equal to
.... > = A AND ....< = B
in other words the values A and B are included.

Result:

PRES_NAME	BIRTH_YR	YRS_SERV	DEATH_AGE	PARTY	STATE_BORN
Roosevelt F D	1882	12	63	Democratic	New York
Johnson L B	1908	5	65	Democratic	Texas

* END OF RESULT ***** 2 ROWS DISPLAYED **********

Another comparison facility which can be useful in some cases is the facility to compare a field not with a whole constant, but with a specific part of a constant. For that purpose we need the Like function:

...column-name [NOT] LIKE quoted-string.

A quoted string used here may contain any string of characters. However special meanings are reserved for the characters _ and %. The character _ represents any single character, while the character % represents any string of zero, one or more characters. The two special characters may be used together with ordinary characters in the quoted string in any combination.

LIKE NOT LIKE

Question 6.11

List all facts available in the table named RECENT_PRESIDENTS about all presidents whose name has the letter e in the second position.

```
SELECT    * –
FROM      RECENT_PRESIDENTS –
WHERE     PRES_NAME LIKE '_e%'
```

Result:

PRES_NAME	BIRTH_YR	YRS_SERV	DEATH_AGE	PARTY	STATE_BORN
Kennedy J F	1917	2	46	Democratic	Massachusetts
Reagan R	1911	3	?	Republican	Illinois

* END OF RESULT ****** 2 ROWS DISPLAYED **********

Question 6.12

List all facts available in the table named RECENT_PRESIDENTS about all presidents whose name has the letter e in the second position, and not the letter R in the first position.

```
SELECT    * –
FROM      RECENT_PRESIDENTS –
WHERE     PRES_NAME LIKE '_e%' –
          AND PRES_NAME NOT LIKE 'R%'
```

Result:

PRES_NAME	BIRTH_YR	YRS_SERV	DEATH_AGE	PARTY	STATE_BORN
Kennedy J F	1917	2	46	Democratic	Massachusetts

* END OF RESULT ****** 1 ROWS DISPLAYED **********

Chapter 7. Built-in Functions

There are five built-in functions:

AVG can be applied to any numeric columns or sets of numbers, and calculates the average of the elements of that column or set. Null values are ignored in that calculation. If all the elements of the column or set are null, the result of the calculation is also null.
The AVG function can be used with or without the keyword DISTINCT.
If AVG (column-name) is used, then the average is computed for all values in the column, including the duplicates.
If AVG (DISTINCT column-name) is used, then the average is computed for all different values in the column, i.e. excluding duplicates.
Example:

A
10
6
6
6

AVG (A) = 7 (10+6+6+6)/4 = 7
AVG (DISTINCT A) = 8 (10+6)/2 = 8

SUM can be applied to any numeric columns or sets of numbers, and calculates the sum of the elements of that column or set. Null values are ignored in that calculation. If all the elements of the column or set are null, the result of the calculation is also null.
The SUM function can be used with or without the keyword DISTINCT.

MIN can be applied to columns or sets of any type, and determines the smallest value of that column or set. If applied to a non-numeric column or set, ASCII (American Standard Code for Information Interchange) ordering is assumed.

MAX can be applied to columns or sets of any type, and determines the largest value of that column or set. If applied to a non-numeric column or set, ASCII (American Standard Code for Information Interchange) ordering is assumed.

COUNT can be applied to a table or a column, and determines the number of rows of the table or fields of the column. There are two cases:
COUNT (*) determines the number of rows of a table regardless of whether it contains null values or duplicates.
COUNT (DISTINCT column-name) determines the number of non-duplicate fields of a column, excluding null values.

SQL logical execution Sequence

1. From
2. where
3. group by
4. Built in fn.
5. having
6. order by

Note:

The object of a function must be enclosed in parenthesis.

From here on in this text we will make use of the Presidential database which consists of a number of tables of which the template is given in appendix 1 and the contents in appendix 2.

Question 7.01

Show the average age at death of the deceased presidents.

```
SELECT    AVG(DEATH_AGE) —
FROM      PRESIDENT
```

Note:

Since null values are ignored, only the deceased presidents are considered.

When nulls are ignored in computation the system will give the user the following message before returning the end result:

```
AR15021  FOLLOWING SQL WARNING CONDITION ENCOUNTERED:
         NULLIGNORED
         PRESS ENTER TO CLEAR THE SCREEN AND CONTINUE
```

After the user has pressed the ENTER key, the system will display the following result

```
AVG (DEATH_AGE)
---------------
          68.85
* END OF RESULT ***** 1 ROWS DISPLAYED ********************************
```

These built-in functions can only be applied in the SELECT clause (or in the HAVING clause as we will see in chapter 9). If used in the SELECT clause, there must be a built-in function applied to all items of the SELECT list , unless a specific feature (Grouping feature, see chapter 9) is used.

Example:

Look at the following query

```
SELECT    PARTY , COUNT (*) –
FROM      PRESIDENT –
WHERE     PARTY = 'Republican'
```

Syntax error.

This is incorrect syntax, because PARTY does not have a function applied to it. This can be corrected by applying a MIN or MAX function to PARTY. Whether we choose MIN of MAX is in this case irrelevant, because only Republican presidents are selected, so both the maximum and the minimum party will be Republican.

Question 7.02

```
SELECT    MIN (PARTY) , COUNT (*) –
FROM      PRESIDENT –
WHERE     PARTY = 'Republican'
```

Result:

```
MIN (PARTY)    COUNT (*)
---------------    ----------
Republican              16
* END OF RESULT ****** 1 ROWS DISPLAYED **********
```

Question 7.03

Show the total of the number of children of all presidents.

```
SELECT    SUM ( NR_CHILDREN) –
FROM      PRES_MARRIAGE
```

Result:

```
SUM ( NR_CHILDREN)
------------------------
                   143
* END OF RESULT ****** 1 ROWS DISPLAYED **********
```

Chapter 8. Calculation

The following arithmetic operators may be used to connect numeric data types:

+	Addition
-	Subtraction
*	Multiplication
/	Division

The may be used in the SELECT items list as well as within a WHERE clause. The following examples shall illustrate the use of arithmetic operations.

Calculation of the average, not using the AVG function, but using the SUM and COUNT functions:

Question 8.01

Show the average age at death of the deceased presidents.
If we write:

```
SELECT    SUM (DEATH_AGE) / COUNT (*) –
FROM      PRESIDENT
```

The SQL system will give the following messages:

```
ARI5021   FOLLOWING SQL WARNING CONDITION ENCOUNTERED: TRUNCATION

ARI5021   FOLLOWING SQL WARNING CONDITION ENCOUNTERED: NULLIGNORED

          PRESS ENTER TO CLEAR THE SCREEN AND CONTINUE
```

Thereafter we get the result:

```
SUM (DEATH_AGE)/COUNT (*)
-------------------------
                      61
* END OF RESULT ****** 1 ROWS DISPLAYED **************************
```

If we compare this result with the result of Q7.01 (68.85) we see that there is a substantial difference.

The query above did not give us the right result, since the SUM function is applied only to the non-null values of DEATH_AGE (deceased presidents), while the COUNT function is applied to all rows of the table PRESIDENT (deceased or alive presidents). To obtain the right result, we have to eliminate the rows where DEATH_AGE is null:

Question 8.02

```
SELECT    SUM (DEATH_AGE) / COUNT (*) -
FROM      PRESIDENT -
WHERE     DEATH_AGE IS NOT NULL
```

The SQL system gives the following message:

```
ARI5021  FOLLOWING SQL WARNING CONDITION ENCOUNTERED: TRUNCATION

         PRESS ENTER TO CLEAR THE SCREEN AND CONTINUE
```

Thereafter we get the following result:

Result:

```
SUM (DEATH_AGE)/COUNT (*)
-------------------------
                      68
* END OF RESULT ****** 1 ROWS DISPLAYED **************************
```

Since the division is an integer division, the result is truncated. Thus, we obtain a slightly different result than in the case where we used the AVG function (see Question 7.01). How to avoid truncation will be discussed under Question 8.05

```
select  1.0*sum(deathage)/count(*)
from    president
where   deathage is not null
```

```
select  prename
from    president
where   X (PRES_serv/death_age) * 100 > 10  AND death_age is not null
        YRS_SERV > 0.1 * Death_age
```

Chapter 9. The Grouping Feature

Suppose we want to know the number of presidents for each party. With the SQL features discussed so far this would require the following repetition:

```
SELECT    COUNT (*) -                    Result:   COUNT (*)
FROM      PRESIDENT -                              ----------
WHERE     PARTY = 'Demo-Rep'                       4

SELECT    COUNT (*) -                    Result:   COUNT (*)
FROM      PRESIDENT -                              ----------
WHERE     PARTY = 'Democratic'                     13

SELECT    COUNT (*) -                    Result:   COUNT (*)
FROM      PRESIDENT -                              ----------
WHERE     PARTY = 'Federalist'                     2

SELECT    COUNT (*) -                    Result:   COUNT (*)
FROM      PRESIDENT -                              ----------
WHERE     PARTY = 'Republican'                     16

SELECT    COUNT (*) -                    Result:   COUNT (*)
FROM      PRESIDENT -                              ----------
WHERE     PARTY = 'Whig'                           4
```

It is obvious that is a tedious procedure. Furthermore one has to know all the party name. This list could be retrieved with the command SELECT DISTINCT PARTY FROM PRESIDENT, but then the user still has to type a separate SELECT command for each party to find out the number of presidents of each party.

What we want to do in such cased is to apply a built-in-function (AVG, SUM, MIN, MAX, COUNT) or a calculation to each one of a number of specific subcategories of that column or table. The features we have introduced so far are not well suited to handle this kind of query. Such queries can be handled by using the grouping feature. In its simplest form, there is a GROUP BY clause to be appended after the ordinary retrieval command:

```
SELECT    ...
FROM      ...
[WHERE    ...]
[GROUP BY column-name , or list-of-column-names]
```

The effect of this GROUP BY clause on the query is that any calculation or function in the SELECT clause is applied only to each individual group of elements specified in the GROUP BY clause.

The GROUP BY feature is very powerful and deserves additional explanation.

Let us look at the result of the following query:

```
SELECT    * –
FROM      PRESIDENT –
ORDER BY  PARTY
```

↑ Sort เรียงลำดับ o/p

The result is presented in figure 9.1

Group by Sort เรียงลำดับ.

```
select  count(*)
from    President
group by party
```

fn applied เหมือน Group

count(*)
4
13
2
16
4

PRES_NAME	BIRTH_YR	YRS_SERV	DEATH_AGE	PARTY	STATE_BORN	Group nr.
Adams J Q	1767	4	80	Demo-Rep	Massachusetts	
Jefferson T	1743	8	83	Demo-Rep	Virginia	
Madison J	1751	8	85	Demo-Rep	Virginia	1
Monroe J	1758	8	73	Demo-Rep	Virginia	
Buchanan J	1791	4	77	Democratic	Pennsylvania	
Carter J E	1924	4	?	Democratic	Georgia	
Cleveland G	1837	8	71	Democratic	New Jersey	
Jackson A	1767	8	78	Democratic	South Carolina	
Johnson A	1808	3	66	Democratic	North Carolina	
Johnson L B	1908	5	65	Democratic	Texas	2
Kennedy J F	1917	2	46	Democratic	Massachusetts	
Pierce F	1804	4	64	Democratic	New Hampshire	
Polk J K	1795	4	53	Democratic	North Carolina	
Roosevelt F D	1882	12	63	Democratic	New York	
Truman H S	1884	7	88	Democratic	Missouri	
Van Buren M	1782	4	79	Democratic	New York	
Wilson W	1856	8	67	Democratic	Virginia	
Adams J	1735	4	90	Federalist	Massachusetts	3
Washington G	1732	7	67	Federalist	Virginia	
Arthur C A	1830	3	56	Republican	Vermont	
Coolidge C	1872	5	60	Republican	Vermont	
Eisenhower D D	1890	8	79	Republican	Texas	
Ford G R	1913	2	?	Republican	Nebraska	
Garfield J A	1831	0	49	Republican	Ohio	
Grant U S	1822	8	63	Republican	Ohio	
Harding W G	1865	2	57	Republican	Ohio	
Harrison B	1833	4	67	Republican	Ohio	4
Hayes R B	1822	4	70	Republican	Ohio	
Hoover H C	1874	4	90	Republican	Iowa	
Lincoln A	1809	4	56	Republican	Kentucky	
McKinley W	1843	4	58	Republican	Ohio	
Nixon R M	1913	5	?	Republican	California	
Reagan R	1911	3	?	Republican	Illinois	
Roosevelt T	1858	7	60	Republican	New York	
Taft W H	1857	4	72	Republican	Ohio	
Fillmore M	1800	2	74	Whig	New York	
Harrison W H	1773	0	68	Whig	Virginia	
Taylor Z	1784	1	65	Whig	Virginia	5
Tyler J	1790	3	71	Whig	Virginia	

* END OF RESULT ***** 39 ROWS DISPLAYED **********

[The count to the right as well as its heading was done by the authors and is not displayed on the screen.]

Figure 9.1

We see that there are 5 groups of rows in the table PRESIDENT, and within each group each row has the same party.

```
SELECT      PARTY , COUNT (*) –
FROM        PRESIDENT –
GROUP BY    PARTY –
ORDER BY    PARTY
```

Result:

PRATY	COUNT (*)
Demo-Rep	4
Democratic	13
Federalist	2
Republican	16
Whig	4

* END OF RESULT ****** 5 ROWS DISPLAYEER **********

Question 9.02

For each state find the number of presidents who were born in that state. List the state name together with this number and present in sequence of ascending number and within the same number by ascending state name.

```
SELECT      STATE_BORN , COUNT (*) –
FROM        PRESIDENT –
GROUP BY    STATE_BORN –
ORDER BY    2, STATE_BORN
```

Note:

The number 2 in the ORDER BY clause refers to the second item in the SELECT list COUNT (*). Only columns can be referred to by their name.

The list of columns names in the GROUP BY clause is not restricted to one. Several column names mean that there are several criteria for grouping, or several dimensions of grouping. In the following example we show a two-dimensional grouping. Let us look at the result of the following query:

```
SELECT      *-
FROM        PRESIDENT-
ORDER BY    PARTY . STATE_BORN , PRES_NAME
```

The result is presented in figure 9.2

If we would apply to the PRESIDENT table the clause GROUP BY PARTY STATE_BORN then all functions specified in the SECLECT clause will be computed once for each of the 26 party / state-born combinations.

Note:

If the query contains a GROUP BY clause then every column in the SELECT list must be either contained in the GROUP BY clause or have a built in function applied to it.

Ex

```
select party , sum(YRS_SERV), count (*)
from president
where party = 'Democrat' or party = 'Republ'
group by party.
```

where party in ('Democrat', 'Republ')

having SUM(YRS_SERV) > 70

Ex

```
select frecname, count(*)
from fres_marriage
where
group by frecname
having count (*) > 1
```

Ex

```
select frecname, sum(NR_children)
from fres_marriage
group by frecname
having sum(NR_children) = 0
```

Republican	Nebraska	1
Republican	New York	1
Republican	Ohio	7
Republican	Texas	1
Republican	Vermont	2
Whig	New York	1
Whig	Virginia	3

* END OF RESULT ***** 26 ROWS DISPLAYED ***************************

*Ex select pres_name, birth_yr, party — ERROR! ເພາະ: ໃນ 1 group ນີ້ party ອາດເປັນ
from president ຫຼາຍ ຄ່າທີ່ແຕກຕ່າງ Value.
groupby party.

*Ex select pres_name, birth_yr — Error ຄືກັນ
:
groupby party

*Ex select * — Error
:
group by party

*Column ທີ່ບໍ່ໄດ້ Group by
ຈະຢູ່ໃນ Select ບໍ່ໄດ້ເດັດຂາດ ເພາະ:
ຈະເຮັດໃຫ້ Error
ຖ້າຕ້ອງໃຊ້ Select * ແລ້ວຕ້ອງ groupby
ຕ້ອງໃຊ້ Sub query ແທນ.

Question 9.08

For those party(ies) which had more than 8 presidents born after 1850 list the name(s) of the party(ies) and the corresponding number(s) of presidents born after 1850

A typical erroneous solution is as follows:

```
SELECT      PARTY , COUNT (*) –
FROM        PRESIDENT –
WHERE       BIRTH_YR > 1850 –
            AND COUNT (*) > 8 –
GROUP BY    PARTY
```

The SQL system will return the following message:

```
ARI503E  AN SQL PROCESSING ERROR HAS OCCURRED. SQLCODE = -120,
         ROW COUNT = 0, SQLCODE: A BUILT-IN FUNCTION SHOULD NOT
         OCCUR IN A WHERE-CLAUSE OR AS THE VALUE TO BE ASSIGNED
         TO A COLUMN IN A SET-CLAUSE OF AN UPDATE STATEMENT.
```

The problem is, that the condition that an individual president is born after 1850 is clearly a row condition and therefore belongs to the WHERE clause, However, COUNT (*) > 8 is a condition which cannot apply to a row because COUNT (*) deals with an entire table, column or group. Therefore COUNT (*) > 8 has to come in the HAVING clause.

Correct solution:

```
SELECT      PARTY , COUNT (*) –
FROM        PRESIDENT –
WHERE       BIRTH_YR > 1850 –
GROUP BY    PARTY –
HAVING      COUNT (*) > 8
```

Result:

```
PARTY                COUNT (*)
----------           ----------
Republican                    9
*END OF RESULT ****** 1 ROW DISPLAYED ******************************
```

functions (COUNT, MIN, MAX) to all marriages of a president, we have to group by president. That means that we have groups of rows in the table PRES_MARRIAGE where the field PRESIDENT is equal for each element of the group. But we do not consider all PRESIDENT groups, we consider only those fulfilling a specific condition, which has to be specified in a HAVING clause. Altogether the retrieval command is as follows:

```
SELECT    PRES_NAME , MAX (NR_CHILDREN) , MIN (NR-CHILDREN) –
FROM      PRES_MARRIAGE –
GROUP BY  PRES_NAME –
HAVING    COUNT (*) > = 2 –
          AND MAX (NR_CHILDREN)> = MIN (NR_CHILDREN) + 2
```

Result:

PRES_NAME	MAX(NR_CHILDERN)	MIN(NR _CHILDERN
Fillmore M	2	0
Roosevelt T	5	1
Wilson W	3	0

*END OF RESULT ***** 3 ROWS DISPLAYED **********

Chapter 10.
Selecting Columns and Rows From Several Tables: Joining 58

All SQL queries discussed in the preceding chapters used only one table.

One of the most useful and powerful functions in SQL is the ability to retrieve with one SQL command information from more than one table.

In the Relational Data Model literature this operation is know under the name JOIN. The practically unrestricted JOIN is a major difference between the powerful 4th generation database systems and the third generation database systems.

As in previous examples of retrieval we specify the column names requires from the various tables in the SELECT clause.

The names of the various tables have to be specified in the FROM clause.

Let us first look at a simple example dealing with two tables as presented in figures 10.1 and 10.2

P_TABLE

	PRES_NAME	BIRTH_YR
1	Buchanan J	1791
2	Harrison B	1833
3	Nixon R M	1913
4	Reagan R	1911

Figure 10.1

M_TABLE

	PRES_NAME	SPOUSE_NAME
1	Harrison B	Scott C L
2	Harrison B	Dimmick M S L
3	Nixon R M	Ryan T C
4	Reagan R	Wyman J
5	Reagan R	Davis N

Figure 10.2

We have selected to operate from the smaller tables of figures 10.1 and 10.2 because of the lengthy result of the unrestricted table combination.

We want to have as result a table with 4 columns to combine information from these two tables in our case

P_TABLE		M_TABLE	
PRES_NAME	BIRTH_YR	PRES_NAME	SPOUSE_NAME

In order to distinguish between the two columns with name PRES_NAME we have to qualify each column name with a prefix which is the name of the table from which the column is to be retrieved.

table-name.column-name

The qualification is not necessary if the column names among the tables in the join are different. In our example this means there is no need to prefix the table names before the column names BIRTH_YR and SPOUSE_NAME. When prefixing, there must be a period and no other character of space between the table name and column name.

Question 10.01

If we want to have as result a table with 4 columns, where each row consists of the president name of the P_TABLE, and his birth year followed by the president name of the M_TABLE and spouse name, and this for all possible combinations, we have to write the following SQL query:

```
SELECT   P_TABLE.PRES_NAME , BIRTH_YR , -
         M_TABLE.PRES_NAM E , SPOUSE_NAME -
FROM     P_TABEL , M_TABLE
```

We get as result the table of figure 10.3 with 4 columns and 20 (!!) rows.

If there is more than one table listed in the FROM clause and there is on WHERE, GROUP BY , or HAVING clause, then the SQL system will give as result a table in which the number of rows is equal to the product of the number of rows of the tables in the FROM clause. In other words all possible combinations of rows from the tables listed in the FROM clause are included in the result. Each row of each table is combined with each row of each other table in the FROM clause. Of course, this is a useful result only in exceptional cases, and it does tend to use enormous quantities of computer resources.

Question 10.02

List the name, birth year spouse names of all married presidents of the database of figures 10.1 and 10.2

```
SELECT   P_TABLE.PRES_NAME , BIRTH_YR , SPOUSE_NAME -
FROM     P_TABLE , M_TABLE -
WHERE    P_TABLE.PRES_NAME = M_TABLE.PRES_NAME
```

ใช้ natural Join

Result:

PRES_NAME	BIRTH_YR	SPOUSE_NAME
Harrison B	1833	Scott C L
Harrison B	1833	Dimmick M S L
Nixon R M	1913	Ryan T C
Reagan R	1911	Wyman J
Reagan R	1911	Davis N

* END OF RESULT ****** 5 ROWS DISPLAYED ***************************

The WHERE clause operates on each of the 20 rows of figure 10.3. Only rows 6, 7, 13, 19 and 20 satisfy the WHERE clause.

An equality operator between two columns of different tables is the most common join condition, as well as the most efficient. When using other join conditions, keep in mind the potentially enormous volume of the product table that may have to be scanned.

Up to 16 tables may be listed in the FROM clause. For all kinds of practical application this has shown to be a very safe upper limit.

Select ...
from S inner-join SP on S.S# = SP.S# (อันนี้คือ Join Condition)

Join S# และ SP
จะ list SNAME
ของ supplier ที่อยู่ London ที่ supply P2

select SNAME
from S, SP
where S.S# = SP.S# and CITY = 'London' and P# = 'P2'
(เป็นตัวอย่าง Semijoin คือ O/P มาจากตารางเดียว แต่ต้องไปดึงมาจากอีกตาราง ใช้ Subquery ก็ได้)

Let us now return to the sample tables of the presidential database in appendix 2.

Question 10.03

List names, birth years, marriage years and spouses of all married presidents. Order by president name.

We find that the information we want to retrieve is contained in two tables, the PRESIDENT table and the PRES_MARRIAGE table. Thus we have to perform a join. Therefore we have to combine each row of the PRESIDENT table with the corresponding row(s) of the PRES_MARRIAGE table where the field(s) in the column PRES_NAME of the PRES_MARRIAGE table is (are) equal to the field in the column PRES_NAME of the PRESIDENT table. The SELECT clause specifies which columns will be selected from the join of the two tables.

```
SELECT    PRESIDENT.PRES_NAME , BIRTH_YR , MAR_YEAR , -
          SPOUSE_NAME -
FROM      PRESIDENT , PRES_MARRIAGE -
WHERE     PRESIDENT.PRES_NAME = PRES_MARRIAGE.PRES_NAME -
ORDER BY  PRESIDENT.PRES_NAME
```

Note:

Each reference to a non-unique column name (appearing in more than one table in the FROM clause) must be prefixed with the table name, hence the ORDER BY will (like in the SELECT clause) need to stipulate PRESIDENT.PRES_NAME

```
select presname, sum(nr_children)
from President, Pres_marriage
where President.presname = Pres_mar.presname
  and party = 'Republican'
group by president.pres_name
having sum(nr_children) > 3
  and count(*) > 1
```

```
select P1.presname
from PRESIDENT P1, PRESMAR P2
where P1.presname = P2.presname
```

```
select presname, sum(nr_children)
from pres_mar
where presname in (select presname from president where party
```

Chapter 11. Subqueries

The concept of a subquery is a very particular and powerful SQL concept. It permits the SQL user to phrase in one query a more complicated question which otherwise would have forced the SQL user to apply two or more queries, with the consequence for the user of putting the results of one query in the next query.

Let us first look at an example to introduce the concept.

Question 11.01

List all the facts of those presidential marriages which resulted in a number of children that is greater than the average number of children per presidential marriage.

With the SQL concepts so far described the user would formulate two queries:

1. List the average number of children per presidential marriage: the result is 3.25.

2. List all facts about those marriages which have more than 3.25 children.

The first query in SQL is:

```
SELECT    AVG (NR_CHILDREN) —
FROM      PRES_MARRIAGE
```

This query results in a simple number, 3.25 which can then be used in the second part

```
SELECT    * —
FROM      PRES_MARRIAGE —
WHERE     NR_CHILDREN > 3.25
```

The concept of a subquery permits the user to do this in one step:

```
SELECT    * —
FROM      PRES_MARRIAGE —
WHERE     NR_CHILDREN > —
(SELECT   AVG(NR_CHILDREN) —
FROM      PRES_MARRIAGE)
```

clause.

- Unlike the main query, a subquery may not have an ORDER BY clause.

- The result of a subquery must be of a type compatible with the first operand of the comparison.

A subquery may be used in the search condition of both the WHERE and the HAVING clause.

However, a subquery must be placed on the right side of the search condition.

Subqueries cannot be used with the LIKE or BETWEEN functions.

** Eg Presname, Deathage ที่ มีอายุมากกว่าคนที่อายุน้อยที่สุดอันดับ สอง (min ที่มีอายุมากที่สุด)

```
Select Presname, death_age
From President
where Death_age = ( Select min(death_age)
                    from President
                    where death_age > (Select min(death_age) from president) )
```

Semijoin =

It is, however quite meaningful to have queries with subqueries which return more than one value. In order to avoid an error indication, we have to use a set comparison operator. A set comparison operator is an ordinary comparison operator qualified by ANY or ALL:

comparison-operator **{ANY | ALL}**

e.g.: > ANY, = ALL, ...

The qualification by ANY means, that the condition id true, if at least one value in the set of values specified by the subquery fulfills the comparison.

The qualification by ALL means, that the condition is true, if all values in the set of values specified by the subquery fulfill the comparison.

Examples:

Roosevelt F D = ANY (ANY = IN)
(Jefferson T, Madison J, Monroe J, Jackson A, Grant US, Wilson W, Roosevelt F D, Eisenhower D D)
→ TRUE, since Roosevelt F D is contained in the set.

Note:

The operator = ANY can be written also as IN, and should be read is contained in.

Ford G R ^= ALL (^= ALL → NOT IN)
(Reagan R, Carter J E, Nixon R M, Johnson L B, Kennedy J F)
→ TRUE, since Ford G R is not contained in the set.

Note:

The operator ^= ALL can be written also as NOT IN and should be read is not contained in

BUG

A = ALL (A, B, C) false
→ FALSE, since A is not equal to some elements of the set in this case B or C.

Note:

Such an expression can never be true, unless the set contains only one element.

A ^= ANY (A, B, C) true
→ TRUE, since A is not equal to some elements of the set in this case B or C.

Ex list distinct presname, birth year
select presname, birth year
from president, pres mar
where president.presname = presmar.presname

select
from president
where presname IN
(select presname
from pres mar)

(semijoin) positive condition join subquery
semijoin negative condition

Note:

Such an expression is always true, unless the set contains only one element.

3 < ANY (1, 2, 3, 4)
→ TRUE, since there is one element in the set which is greater than 3, that is 4.

3 < ALL (1, 2, 3, 4)
→ FALSE, since there are elements in the set which is greater than 3, that is 1, 2.

3 > ALL (1, 2)
→ TRUE, since all elements in the set are less than 3.

Another kind of a frequently occurring query is the following one:

Question 11.07

Which state provided the largest number of presidents, and what is that number?

```
SELECT     STATE_BORN , COUNT (*) –
FROM       PRESIDENT –
GROUP BY   STATE_BORN –
HAVING     COUNT (*) > = ALL –          (Where the number
(SELECT    COUNT (*) –                  is greater or equal
FROM       PRESIDENT –                  to all number in the
GROUP BY   STATE_BORN)                  following list)
```

The qualification ALL means that the condition is true if ALL values in the set of values specified by the subquery satisfy the comparison. In this case only the max-imum count satisfies all, hence that is the only group which is retained in the main query.

Result:

```
STATE_BORN          COUNT (*)
------------------  ----------
Virginia                     8
* END OF RESULT ****** 1 ROWS DISPLAYED **************************
```

Note:

If we are sure that the subpuery is going to return only one value, we can use an ordinary comparison operator. However, if the subquery will then return more than one value, the main query will give an error indication.

* builtin fn ซ้อน builtin fn ไม่ได้

Question 11.08

Find those states which entered the union before President Washington was inaugurated

The subquery that finds in which year Washington was inaugurated is as follows:

```
SELECT          YEAR_INAUGURATED –
FROM            ADMINISTRATION –
WHERE           PRES_NAME = 'Washington G'
```

We find that this query returns two values:

```
YEAR_INAUGURATED
---------------------------------
                    1789
                    1793
* END OF RESULT ****** 2 ROWS DISPLAYED ****************************
```

Thus our query was not clear. Do we mean those states which entered the union before president Washington was inaugurated first (a), or last (b)?
The situation is clarified by suing the set comparison operators <ALL or <ANY respectively:

**Ex list รายชื่อ president พรรค Democratic ทุกคน ที่มีบุตรรวมน้อยกว่า 2 คนและมี hobby มากกว่า 4 hobby

```
select *
from president
where pres_name = ANY (select pres_name from pres_mar group by pres_name
                       having sum(nr_children) < 2)
and pres_name = ANY (select pres_name from hobby group by pres_name
                     having count(*) > 4)
and party = 'Democratic'
```

Ex list S# ของ supplier ที่ supply part ได้จำนวนรวม มากที่สุด

```
select S#
from SP
~~where S# = ANY~~
group by S#
having sum(QTY) >= ALL (select sum(QTY) from SP group by S#)
       = MAX( ... ) ไม่ได้
```

One may also solve this problem by having two tables say T1 and T2 which are both copies of the table PRESIDENT

New copies of a table are declared in the FROM clause by the placement of the new name after the old name separated by a blank.

E.g. to introduce two new name T1 and T2 for the table PRESIDENT our FROM clause should state:

```
FROM    PRESIDENT T1 , PRESIDENT T2 –
```

With the availability of two PRESIDENT tables we can compare a birth year of a given row in one table with a birth year of a row in the second table.

We can now write:

Question 12..01b

```
SELECT  T1.PRES_NAME , T1.BIRTH_YR , T2. PRES_NAME –
        T2.BIRTH_YR –
FROM    PRESIDENT T1 , PRESIDENT T2 –
WHERE   T1.BIRTH_YR = T2.BIRTH_YR
```

Q6 label

condition.

To avoid this kind of redundancy and irrelevancy we use the following SQL query:

Question 12.01c

List presidents born on same year. List each president only once. We can eliminate unwanted rows by making sure that the match is between different presidents and that this is the first time that this match has been found.

```
SELECT   T1.PRES_NAME , T1.BIRTH_YR , T2.PRES_NAME , –
         T2.BIRTH_YR –
FROM     PRESIDENT T1 , PRESIDENT T2 –
WHERE    T1.BIRTH_YR = T2.BIRTH_YR –
         AND T1.PRES_NAME < T2.PRES_NAME
```

Note:

The last condition in the WEHRE clause will only be satisfied the first time that a particular pair of presidents is matched.

Result:

Chapter 13. Correlated Subqueries

In the previous subquery cases, we were able to obtain the result of a subquery independently (without referring to anything outside the subquery) and in particular without referring to the main query. There are however, queries where we cannot obtain the result of the subquery without referring to rows in the main query, or where it is convenient to use such a construction.

A subquery which has a search condition which relates to the main query is called a correlated subquery.

Let us first look at an example that can be expressed in several ways in SQL.

Question 13.01

List the name and birth year of those presidents who were inaugurated at least once within 45 years of their birth year.

We deal in this case with two tables, PRESIDENT and ADMINISTRATION.

Without the aid of a computer we probably would have to take the two tables which were both sorted by president name and retain in the president table only two columns, namely PRES_NAME and BIRTH_YR and in the ADMINISTRATION table the columns PRES_NAME and YEAR_INAUGURATED for the first inauguration of each president.

These two tables are presented in figures 13.1 and 13.2

We would then go to the first row in the PRESIDENT table, take the birth year, go to the ADMINISTRATION table and take the row which has the same president name and compare whether the birth year plus 45 years is greater than the first inauguration year of that president. If yes; then list president name in the result of the query if not then take the next row in the PRESIDENT table and repeat the procedure. Continue until all rows of the PRESIDENT table are processed in this way.

It is important to repeat that we are comparing for each specific president, the birth year with 45 added to it with the first inauguration year of that president.

Chapter 14. Subqueries with Test for Existence

Instead of using a comparison operator with a subquery we may also use the EXISTS or NOT EXISTS operator within a WHERE clause.

... WHERE [NOT] EXISTS (subquery) ...

If the EXISTS operator is used the condition in the WHERE clause is satisfied if the subquery results in at least one row. If the NOT EXISTS operator is used the condition is satisfied if the subquery returns no rows.

Question 14.01

List the names and ages at death of all presidents who married.

The main query is on the table PRESIDENT, while the subquery is on the table PRES_MARRIAGE. However, it is not necessary in this case to investigate specific columns of the table PRES_MARRIAGE. All that is needed is to check whether a value appearing in the column PRES_NAME of the table PRESIDENT does appear in the column PRES_NAME of the table PRES_MARRAGE. This can be formulated using the EXISTS operator. It is also necessary in this case to correlate the column PRES_NAME in the table PRES_MARRIAGE with the column PRES_NAME in the able PRESIDENT.

```
SELECT    PRES_NAME , DEATH_AGE –
FROM      PRESIDENT –
WHERE     EXISTS –
(SELECT   * –
FROM      PRES_MARRIAGE –
WHERE     PRES_MARRIAGE.PRES_NAME = PRESIDENT.PRES_NAME)
```

Question 14.03

List the election year and name of the winning candidates who never became president.

```
SELECT    ELECTION_YEAR , CANDIDATE –
FROM      ELECTION –
WHERE     WINNER_LOSER_INDIC = 'W' AND NOT EXISTS –
(SELECT   * –
FROM      PRESIDENT –
WHERE     PRES_NAME = CANDIDATE)
```

Result:

The system displays the following:

ARI1501I THERE ARE NO ROW THAT SATISFY THE CONDITION

Ex list row ของ pres [illegible]

```
select *
from President P1
where not exist (select * from ~~president~~ election P2
    where ~~P2.presname~~ P2.candidate = P1.presname
    and win lose_indic = 'W')
```