1.10 User-Defined Datatypes (Section may be skipped on first reading.)

As a Transact-SQL enhancement to SQL, you may name and design your own datatypes to supplement the system datatypes. The user-defined datatypes can serve as domains for attributes of relations. A user-defined datatype is defined in terms of system datatypes. You may give one name to a frequently used type definition. This makes it easy for you to custom fit datatypes and give semantics to columns.

For example, consider the employee table

employee (eno, ename, salary, manager, birthday, startday)

You can define a user-defined datatype 'id_no' for eno and manager in table employee instead of using 'int'. The advantage of user-defined datatypes is that you can bind rules and defaults to them, for use in several tables. (Refer to *Transact-SQL User's Guide* for rules and defaults).

The stored procedure "sp_addtype" is used to create user-defined datatypes. It takes as parameter the name of the user datatype being created, the SQL system datatype from which it is being built, a length (if one is required), and an optional NULL or NOT NULL specification.

The syntax for "sp_addtype" is:

sp_addtype *typename, phystype [(length)]* [, **NULL** | **NOT NULL**]

You need to specify a length after the datatype only for char and varchar. Double or single quotes are required around a stored procedure parameter when:

- the string includes a blank or some form of punctuation or
- the parameter is a keyword.

To create a new datatype 'id_no' for eno and manager of the employee table:

```
ACTION: 1>  sp_addtype id_no, int, NOT NULL
        2>  go
```

When you create the table 'employee', you may specify 'id_no' as the datatype of eno and manager instead of using 'int'.

1.9.4 Parts of a date or time using the *datename* and *datepart* functions

Individual parts of a date may be displayed using the *datename* and *datepart* functions. The function *datename* returns the name as a character string, e. g. "Monday". The function *datepart* returns the integer value, e. g. "2". An example showing the weekday, "dw" is shown below:

```
1> select datename(dw,shipdate), datepart(dw,shipdate) from supply
2> go

 ------------------------------ -----------
 Monday                                   2
 Friday                                   6
 Monday                                   2
   ...
```

The possible date and time parts are given in *Table 2-6* of the *SQL Server Reference Manual*.

1.9.5 Date/Time Data Entry Format

Sybase provides many formats for entering dates and time. One format will be described in this section. This format allows the entry of a date, e. g. March 20, 1995, or a date and time with hours (0 - 24) and minutes, e. g. March 20, 1995 13:30. For other formats and to enter seconds and milliseconds or to use AM and PM refer to the *SQL Server Reference Manual*.

The easiest format for entering a date is to use a *numeric format* "[m]m/dd/[yy]yy" for a date alone or "[m]m/dd/[yy]yy [h]h:[m]m" for a date and time. Note that if only two digits are used for the year, then values less than 50 are interpreted as 20yy and values of 50 or greater are interpreted as 19yy. Since the data being entered is treated as a character string, it must be enclosed in single or double quotes.

The following examples show some rows being entered into the "supply" relation:

```
insert supply values(22,22,34, '3/20/95', 5);
insert supply values(22,22,35, '3/20/95 14:30', 5);
insert supply values(22,22,36, '3/20/15', 5);
insert supply values(22,22,37, '3/20/2095 14:30',5);
insert supply values(22,22,38, '3/2/1995 1:3', 5);
```

The output for those dates is shown below:

```
select shipdate from supply where sno = 22
go

shipdate
--------------------
Mar 20 1995 12:00AM
Mar 20 1995 2:30PM
Mar 20 2015 12:00AM
Mar 20 2095 2:30PM
Mar 2 1995 1:03AM

(5 rows affected)
```

Note that the first insert translates year "95" to the year "1995" and generates the default time of "12:00AM", i. e. midnight. The second insert with a time of "14:30" is printed as "2:30PM", since that is the default output format. The third insert translates the year "15" to "2015". The fourth insert shows the use of a four digit year, "2095". The last insert shows leading zeros are not needed in the day, hours or minutes.

1.9 Date and Time in Sybase (Section may be skipped on first reading.)

Sybase provides two special data types for storing dates and times.

1.9.1 Date/Time Data Types

The date/time data types are called *datetime* and *smalldatetime*. These data types are different from those specified in the **SQL 92** standards and described in the *Elmasri and Navathe* text. In the Tutorial Database, the attribute "shipdate" of the relation "supply" is of type *datetime*. (Note that the attribute "bdate" of the relation "employee" is of type *char(9)*), **not** of type *datetime*.

The *datetime* data type can hold a date between January 1, 1753 and December 31, 9999. Values are accurate to 1/300th of a second. 8 bytes of storage are required, 4 bytes for the number of days since January 1,1753, and 4 bytes for the time of day.

The *smalldatetime* data type can hold a date between January 1, 1900 and June 6, 2079. 4 bytes of storage are required, 2 bytes for the number of days since January 1, 1900, and 2 for the number of midnights after midnight.

1.9.2 Display Format

The default display format for a date is "Mon dd yyyy hh:miAM" (or PM), e. g. "Mar 20, 1995 1:58PM". For example using the Tutorial database

```
1> select shipdate from supply
2> go
```

yields the partial result[6]:

```
shipdate
--------------------------
       Dec 31 1973 12:00AM
       May 31 1974 12:00AM
       Dec 31 1973 12:00AM
         ...
```

1.9.3 Other formats using the *convert* function

It is possible to display a date in many other formats (each is identified by a number) using the *convert* function. The year may be shown as 2 or 4 digits. The order of the day, month and year may be permuted. '/' or '-' may be used as separators. Most formats show the date alone. Some show the date and time.

An example using format "1" is shown below:

```
1> select convert(varchar(20), shipdate, 1 ) from supply
2> go

 --------------------
 12/31/73
 05/31/74
 12/31/73
   ...
```

All the possible formats are described in *Table 2-4* of the *SQL Server Reference Manual*.

[6] In this example and in all of the others in this section, the output has been truncated to save space.

1.5.2 Destroy view

```
ACTION:1>  drop view empview
       2>  go
```

1.6 How to create an index

We can use the following statement to create an index:

create [**unique**] **index** *indexname* **on** *tablename*
(*columnname* {, *columnname*})

The **create index** statement creates an index on an existing base table. The index contains the columns specified and is keyed on those columns, in the order they are specified.

1.6.1 An example of creating index

```
ACTION:1>  create index empidx on employee(eno)
       2>  go
```

1.6.2 Destroy index

```
ACTION:1>  drop index employee.empidx
       2>  go
```

To drop index, you must give both the table and the index name, in the form tablename.indexname.

1.7 How to give out privilege of access to your tables

The following grant statement grants privileges on a table, view, or procedure:

grant all [**privileges**] **on** *tablename* {, *tablename* }+
to public | *username* {,*username* }

grant ***priv*** {,***priv*** } **on** *tablename* {, *tablename* }
to public | *username* {,*username* }

The *priv* can be any of the following:

select
insert
delete
update [(*columnname* {, *columnname*})]
execute

1.8 Help Commands

The **help** command can be used to obtain information about the database. The legal forms are as follows:

OSUsp_help	Lists all user tables, views, and indexes that exist in the current database.
OSUsp_help *tablename*	Provides the name, owner, creation date and time of the table. And displays the name, data type, length, nullability, default-ability, and key sequence for each column in the table.

1.4.3 Deleting columns

To delete one or more columns from a table, proceed as in adding columns. The example deletes all columns from table student except id and name.
1. Create a temporary table containing all the columns to be included in the revised table.

```
ACTION: 1>  create table tmp
        2>      (id smallint, name varchar(20))
        3>  go
```

2. Populate new table by selecting columns from original table.

```
ACTION: 1> insert tmp
        2>    select id, name
        3>    from   student
        4> go
```

3. Destroy the original table.

```
ACTION: 1> drop table student;
        2> go
```

4. Rename the temporary table with the name of the original table.

```
ACTION: 1> sp_rename tmp, student
        2> go
```

1.5 How to create views

Views can be thought of as virtual tables. They do not store copies of the data, but refer to the base tables involved in the view. Primary uses for views include:

- Providing security by limiting access to specific columns in selected tables, without compromising database design
- Simplifying a commonly used query
- Defining reports

A view is created with the **create view** statement:

create view *viewname* [(*columnname* {,*columnname*})]
as *select-stmt*

A view is destroyed if the base table is destroyed.

1.5.1 An example of creating view

```
ACTION: 1>  create view empview(
        2>      col1,
        3>      col2,
        4>      col3)
        5>    as select eno, ename, salary
        6>      from   employee
        7>  go
```

```
ACTION: 1>  create table student_tmp(
        2>       id   smallint,
        3>       name varchar(20))
        4>  go
```

Then, we insert into the table by selecting data from "student" table.

```
ACTION: 1>  insert student_tmp
        2>      select id, name
        3>      from    student
        4>  go
```

To insert only the students with id < 3, type:

```
ACTION: 1> insert student_tmp
        2>      select id, name
        3>      from   student
        4>      where  id < 3
        5>  go
```

1.3.4 Destroy a table

```
ACTION: 1>  drop table student_tmp
        2>  go
```

COMMENT: removes table "student_tmp" from the database.

1.4 Adding, Changing, and Deleting columns

1.4.1 Adding columns

You can add new columns to a table at any time with the **ALTER TABLE** command. Here is its syntax:

alter table *tablename*
add *columnname datatype* **null**

For example, to add a column "birthday" to the student table, type:

```
ACTION:1>  alter table student
       2>  add birthday datetime null
       3>  go
```

Columns added with the ALTER TABLE statement must allow null values. This is because when the new column is added to the existing rows, there must be some value for it.

1.4.2 Changing columns

You can use sp_rename to rename columns of a table. The syntax is:

sp_rename "table_name.column_name", new_column_name

For example, you want to change "birthday" of student table to "birthdate", type:

```
ACTION:1>  sp_rename ''student.birthday'', birthdate
       2>  go
```

The following are some of the most frequently used data types in SYBASE/SQL:

Data Type	Meaning	Length	Example
char	fixed length strings		'ABC'
varchar	variable length strings		'columbus'
smallint	small integers	2	999
int	bigger integers	4	1,100,341,645
real	floating point numbers	4	12.34(7 digit precision)
float	floating point numbers	8	1234.56(16 digit precision)
datetime	dates	12	'03-21-91'
money	money data types	8	$45.99

1.3.2 Insert data into table

After creating a table, you can use the **insert into** statement to input data. SYBASE does not limit the number of tables in a database; you are limited only by the available disk space. If you are dealing with very large tables or data configurations, beware that:

- The maximum number of columns is 250.

To enter the following data into table "student",

id	name	street	city	zipcode	phone
1	Jones,Tim	32 high st	columbus	43201	614-9948767
2	Smith, Paul	2890 dennison ave	columbus	43206	614-3376812
3	Evans, Michael	2050 woodruff ave	columbus	43201	614-1029221
4	Thomas, Peter	2345 high st	columbus	43203	614-2341345
5	Collins, Joanne	345A neil ave	columbus	43221	614-7896045

You have to use the **insert** command we described in 1.2.2.

ACTION:

```
1>  insert  student
2>  values (1, 'Jones, Tim', '32 high st', 'columbus', '43210', '614-9948767')
3>  insert  student
4>  values ( 2, 'Smith, Paul', '2890 dennison ave ', 'columbus', '43210', '614-3376812')
5>  insert  student
6>  values ( 3, 'Evans, Michael', '2050 woodruff ave', 'columbus', '43210', '614-1029221')
7>  insert  student
8>  values ( 4, 'Thomas, Peter', '2345 high st ', 'columbus', '43210', '614-2341345')
7>  insert  student
10>  values ( 5, 'Collins, Joanne', '345A neil ave', 'columbus', '43210', '614-7896045')
11>  go
```

Note that we can type in several insert commands and execute them all together.

1.3.3 Insert with Selection

To pull values into a table from one or more other tables, use a **select** in the **insert** statement. As an exercise, let's create a temporary table "student_tmp" which consists of only the "id" and "name" and then insert the corresponding data from the "student" table.

We first create the table:

```
2>  values(22,'chair','grey',20,20)
3>  go
```

COMMENT: The above two tuples were appended to the table.

Update operation

```
ACTION:1>  update parts
       2>  set quantity=100
       3>  where pno > 20
       4>  go
```

COMMENT: You can also update a field by using an expression, and you can change more than one field in a single update query.

Delete operation

```
ACTION:1>  delete from  parts
       2>  where pno > 20
       3>  go
```

COMMENT: This command deletes all the tuples added.

1.3 Database Creation

This part of the tutorial is intended to provide you with a quick overview of database creation. There are two components in creating a new database. They are:

- specifying the relations and associated attributes
- entering data into the database.

1.3.1 How to create a table

The following **create table** statement creates a new table with the specified column formats:

create table *tablename* (*columnname format* [**no null** | **null**]
{, *columnname format* [**no null** | **null**] })

Note that **no null** means the column specified doesn't allow null to be stored as a value; **null** means the column specified allow null value.

Create a table

```
ACTION: 1>  create table student(
        2>          id       smallint,
        3>          name     varchar(20),
        4>          street   varchar(30),
        5>          city     varchar(15),
        6>          zipcode  char(5),
        7>          phone    char(10))
        8>  go
```

COMMENT: This query modifies the previous example by printing out the names of those managers having more than two employees.

Use of "Union"

The complete select statement syntax is:

subselect { **union** [**all**] *subselect*}
[**order by** *result-column* [*asc*|*desc*] {, *result-column* [*asc*|*desc*] }]

where the syntax for "subselect" is:

select[**all**|**distinct**] *expression* {, *expression* }
[**from** *table*[*corr-name*] {, *table*[*corr-name*]}
[**where** *search-condition*]
[**group by** *column* {, *column*}]
[**having** *search-condition*]

Note that duplicate rows are always eliminated if **union** is specified. But if you say **union all**, duplicates are not removed. If you say **union all** once, you must say it for all unions within one statement. Also all subselects in a **select** statement with **union** must have the same number of columns in their result tables. Additionally, columns of numeric type cannot be matched with columns of character type.

```
ACTION: 1>  select pno from parts where color='black'
        2>        union
        3>  select pno from supply where shipdate = '12-31-73'
        4>  go
```

COMMENT: This query selects all part numbers (pno) that are either black or are shipped on December 31, 1973. Duplicates are eliminated.

1.2.2 How to modify the database

In addition to its query facilities, SQL also provides the capability to modify the database. This means that you can interactively insert, modify and delete tuples in your database. Access privileges for insert, modify and delete operations will be discussed in 1.7. Here you are allowed to experiment with these operations on the tutorial database. The command formats are as follows:

insert into *tablename* [(*column* {, *column*})]
values (*expr* {, *expr*}) | *subselect*

update *tablename* [*corr-name*]
from *tablename* [*corr-name*] {, *tablename* [corr-name]}
set *columnname* = *expression* {, *columnname* = *expression*}
where *search-condition*

delete from *tablename* [*corr-name*] [**where** *search-condition*]

Insert from the screen

```
ACTION:1>  insert into parts
       2>  values(21,'desk','yellow',40,20)
       3>  go
       1>  insert into parts
```

select[**all**|**distinct**] *expression* {, *expression* }
[**from** *table*[*corr-name*] {, *table*[*corr-name*]}]
[**where** *search-condition*]
[**group by** *column* {, *column*}] [**having** *search-condition*]
[**order by** *result-column* [**asc** | **desc**]]

Use of "order by" clause

.

```
ACTION: 1>  select ename, salary, startday
        2>  from   employee
        3>  where  startday between '1960' and '1986'
        4>  order by startday desc
        5>  go
```

COMMENT: The use of "between" and "and" keywords in the where clause is equivalent to combining the two inequalities <= and >=. The "order by" clause specifies the sequence of the rows that result from a query. The default sort sequence is in ascending order. "desc" represents "descending".

Use of "group by" and "having clause"

```
ACTION: 1>  select manager, count(*)
        2>  from   employee
        3>  group by manager having count(*) > 2
        4>  go
```

COMMENT: This query finds out all manager numbers of those having more than two employees. Note that we have an empty header for the second column of the output rather than "count" or "count(*)". SYBASE provides the following aggregation functions:

Name	Result Data Type	Description
count	integer	count of occurrences
sum	integer, float, money	column total
avg	float, money	average
max	same as argument	maximum value
min	same as argument	minimum value

The built-in functions are only valid when used in "select" or "having" clauses.

Nested query

We have seen a simple nested query in the previous example. Nesting allows you to use the results of one query as input to another, so you can use the results of one question to answer another one. Let's try another nested query in the next example.

```
ACTION: 1>  select distinct  eno, ename
        2>  from   employee
        3>  where  eno =
        4>              (select manager
        5>               from   employee
        6>               group by manager having count(*) > 2)
        7>  go
```

```
Feb  1 1975 12:00AM
Jun  1 1975 12:00AM
Jul  1 1975 12:00AM
Jan  1 1976 12:00AM

(12 rows affected)
```

From now on, for most examples we will show only the queries but not the results. You should run the queries to see the results.

Arithmetic Operation

ACTION:
```
1> select ename, salary + (salary-4000) * 0.15
2> from employee
3> where salary <= $10000
4> go
```

COMMENT: This query shows the result of giving a bonus to employees whose salary is not greater than $10,000 .

Logical operation

ACTION:
```
1> select *
2> from   employee
3> where  manager = 199
4>   and  (startday <= '1960' or
5>        salary <= $10000)
6> go
```

COMMENT: This query shows the names and salaries of the employee whose manager=199, if the employee was hired before 1961 or has a salary not greater than $10,000.

Simple Join Operation

Up until now, we haven't used another important relational operation: **join**. The next example illustrates a simple join with projections. Suppose we want to find out all part names and the corresponding supplier numbers for a given shipdate: 12-31-73. To perform this operation, two relations: parts and supply, are referenced.

ACTION:
```
1> select sno, pname
2> from   supply, parts
3> where  shipdate = '12-31-73'
4>   and  supply.pno = parts.pno
5> go
```

COMMENT: If you forgot to give the last qualification (that is, supply.pno = parts.pno), you will get a Cartesian product on selected "sno" and all "pname". From this, can you see what is the relation between Cartesian product and join?

The syntax for a simple "select" is:

```
1> select shipdate
2> from   supply
3> go
```

The result of the above query is the following:

```
shipdate
--------------------------
       Dec 31 1973 12:00AM
       May 31 1974 12:00AM
       Dec 31 1973 12:00AM
       May 31 1974 12:00AM
       Feb  1 1975 12:00AM
       Feb  1 1975 12:00AM
       Feb  1 1975 12:00AM
       Dec 31 1973 12:00AM
       Jul  1 1974 12:00AM
       Dec 31 1973 12:00AM
       Jun  1 1975 12:00AM
       Jun  1 1975 12:00AM
       Jun  1 1975 12:00AM
       Jul  1 1975 12:00AM
       Jul  1 1975 12:00AM
       Nov 15 1974 12:00AM
       Jan 22 1975 12:00AM
       Jan 10 1975 12:00AM
       Jan 10 1975 12:00AM
       Oct 10 1974 12:00AM
       Jun 18 1974 12:00AM
       Jul  1 1975 12:00AM
       Jan  1 1976 12:00AM

(23 rows affected)
```

To eliminate the duplicates, use the keyword “distinct”:

```
1> select distinct shipdate
2> from   supply
3> go
```

The result is the following:

```
shipdate
--------------------------
       Dec 31 1973 12:00AM
       May 31 1974 12:00AM
       Jun 18 1974 12:00AM
       Jul  1 1974 12:00AM
       Oct 10 1974 12:00AM
       Nov 15 1974 12:00AM
       Jan 10 1975 12:00AM
       Jan 22 1975 12:00AM
```

```
eno       ename                salary    manager birthday startday
------- -------------------- ---------- ------- -------- --------
     35 Evans, Michael          5,000.00      32 1952     1974
   1110 Smith, Pau              6,000.00      33 1952     1973
    215 Collins, Joanne         7,000.00      10 1950     1971
   2398 Wallace, Maggie J.      7,880.00      26 1949     1959
   4901 Bailey, Chas M.         8,377.00      32 1956     1975
   1330 Onstad, Richard         8,779.00      13 1952     1971
     13 Edwards, Peter          9,000.00     199 1928     1958
     98 Williams, Judy          9,000.00     199 1935     1969
     32 Smith, Carol            9,050.00     199 1929     1967
    129 Thomas, To             10,000.00     199 1941     1962
     33 Hayes, Evelyn          10,100.00     199 1931     1963
   1639 Choy, Wanda            11,160.00      55 1947     1970
    843 Schmidt, Herman        11,204.00      26 1936     1956
     37 Raven, Lemont          11,985.00      26 1959     1974
     15 Jones, Tim             12,000.00     199 1940     1960
     55 James, Mary            12,000.00     199 1920     1969
     11 Ross, Stuart           12,067.00       0 1931     1932
     26 Thompson, Bob          13,000.00     199 1930     1970
   5219 Williams, Bruce        13,374.00      33 1944     1959
   5119 Ferro, Tony            13,621.00      55 1939     1963
    994 Iwano, Masahiro        15,641.00     129 1944     1970
     10 Ross, Stanley          15,908.00     199 1927     1945
    430 Brunet, Paul C         17,674.00     129 1938     1959
   1523 Zugnoni, Arthur A.     19,868.00     129 1928     1949
    199 Bullock,J.D.           27,000.00       0 1920     1920

(25 rows affected)
```

Table 1.2: Employee

```
sno           int              4     1  (* supply number          *)
pno           int              4     1  (* part number            *)
jno           int              4     1  (* job number             *)
shipdate      datetime         8     1  (* ship date              *)
quantity      int              4     1  (* quantity               *)
```

This information may be obtained using the help facilities of SQL. For example "OSUsp_help" lists all tables, "OSUsp_help `parts`" list the description of the table "parts". However, note that the column "meaning" won't be provided by the help facility. It appeared above only as our remarks. Also note that an "1" in the null column means "null" is allowed in the corresponding attribute; a "0" means "null" is not allowed.

We have seen table "parts". Tables 1.1 and 1.2 in the following show the contents of tables "supply" and "employee" respectively.

```
sno         pno         jno         shipdate                   quantity
----------- ----------- ----------- -------------------------- -----------
        475           1        1001        Dec 31 1973 12:00AM           1
        475           2        1002        May 31 1974 12:00AM          32
        475           3        1001        Dec 31 1973 12:00AM           2
        475           4        1002        May 31 1974 12:00AM           1
        122           7        1003        Feb  1 1975 12:00AM          48
        122           7        1004        Feb  1 1975 12:00AM         144
        122           9        1004        Feb  1 1975 12:00AM           2
        475           2        1001        Dec 31 1973 12:00AM          32
        475           1        1002        Jul  1 1974 12:00AM           1
        241           4        1001        Dec 31 1973 12:00AM           1
        241           1        1005        Jun  1 1975 12:00AM           1
        241           2        1005        Jun  1 1975 12:00AM          32
        241           3        1005        Jun  1 1975 12:00AM           1
        241           8        1005        Jul  1 1975 12:00AM           1
        241           9        1005        Jul  1 1975 12:00AM         144
          5           4        1003        Nov 15 1974 12:00AM           3
          5           4        1004        Jan 22 1975 12:00AM           6
         10           5        1001        Jan 10 1975 12:00AM          20
         10           5        1002        Jan 10 1975 12:00AM          75
        440           6        1001        Oct 10 1974 12:00AM           2
         62           3        1002        Jun 18 1974 12:00AM           3
         67           4        1005        Jul  1 1975 12:00AM           1
        999          10        1006        Jan  1 1976 12:00AM         144

(23 rows affected)
```

Table 1.1: Supply

1.2.1 Some Query Examples

In the following, we use some examples to illustrate the "select" command of SQL.

Simple Projection

Type the following commands:

1.1.8 Exiting SYBASE

To leave the SYBASE, type:

```
1> exit
```

1.1.9 Summary of Commands Related to Query Buffer

The following are a list of the commands related to the query buffer:

`reset`	Erase the entire query
`emacs` or `vi`	Enter the named text editor.
`go`	Execute the current buffer.
`:r` *filename*	Read the named file into the query buffer.
`exit` or `quit`	Exit SYBASE.

1.2 The Tutorial Database

Thus far we have signed onto SYBASE, started SQL and viewed a table called "parts". The following is a description of all the tables used in this tutorial and their formats:

```
Name                 Type             When_created
-------------------- ---------------- --------------------
parts                user table       Mar  3 1993  6:49PM

Column_name   Type         Length Nulls (* Meaning                *)
------------- ------------ ------ ----- (* ---------------------- *)
pno           int               4     1 (* part number            *)
pname         varchar          20     1 (* part name              *)
color         varchar          10     1 (* color                  *)
weight        int               4     1 (* weight                 *)
quantity      int               4     1 (* quantity at hand       *)


Name                 Type             When_created
-------------------- ---------------- --------------------
employee             user table       Mar  3 1993  7:33PM

Column_name   Type         Length Nulls (* Meaning                *)
------------- ------------ ------ ----- (* ---------------------- *)
eno           int               4     1 (* employee number        *)
ename         varchar          20     1 (* employee name          *)
salary        money             8     1 (* employee salary        *)
manager       int               4     1 (* manager number         *)
birthday      char              4     1 (* birthday               *)
startday      char              4     1 (* start date             *)


Name                 Type             When_created
-------------------- ---------------- --------------------
supply               user table       Mar  3 1993  6:49PM

Column_name   Type         Length Nulls  (* Meaning                *)
------------- ------------ ------ -----  (* ---------------------- *)
```

```
go
```

Then the following appears:

```
pname                quantity
-------------------- -----------
central processor              1
memory                        32
disk drive                     2

(3 rows affected)
```

Loading Previously Prepared Query File into Query Buffer

First use emacs to create a file of the following:

```
select sno, pno
from supply
where pno in
          (select pno
          from parts
          where pname like 'memory%')
```

Note if you forget to type the right quote, when you execute the query later, nothing will happen. So please check if the quotes or parenthesis match. Notice also that the "%" used in the above query is one of the pattern matching constructs. In fact, there are three pattern matching constructs provided by SYBASE. All three can be used in any combination for character comparison. They are:

- % matches any length of character string
- _ matches any one (non-blank) character
- [. . .] can match any character listed in the brackets.

Save the above query in a file called "smallquery". Then type :

```
1> :r smallquery
2> go
```

Then the following appears:

```
sno         pno
----------- -----------
        475           2
        475           2
        241           2

(3 rows affected)
```

Erasing Query Buffer

The following command will allow you to clear the entire query buffer. The former contents of the buffer are lost and cannot be retrieved.

```
1> reset
```

1.1.7 The Query Buffer

Once you have entered `isql`, each query that you type is placed in a query buffer, rather than executed immediately. The queries are not executed until you type the command terminator (`go`). Recall that **go** must not be preceded by any blanks. The results, by default, appear on your terminal. For example, we have a table called "parts" in our database. Type the following command:

```
1> select * from parts
2> go
```

The line "select * from parts" selects all from the table "parts". The following then appears:

```
pno         pname                color      weight      quantity
----------- -------------------- ---------- ----------- -----------
          1 central processor    pink                10           1
          2 memory               gray                20          32
          3 disk drive           black              685           2
          4 tape drive           black              450           4
          5 tapes                gray                 1         250
          6 line printer         yellow             578           3
          7 l-p paper            whit                15          95
          8 terminals            blue                19          15
          9 terminal paper       white                2         350
         10 byte-soap            clear                0         143
         11 card reader          gray               327           0
         12 card punch           gray               427           0
         13 paper tape reader    black              107           0
         14 paper tape punch     black              147           0

(14 rows affected)
1>
```

What is printed on your terminal is the current content of the "parts" relation. In this case, the relation name is "parts". There are five columns (we call them attributes) named pno (part number), pname (part name), color, weight, quantity(quantity at hand). Each row of the relation (called a tuple) represents one entry, which in this case represents one part in a computer installation.

After a `go` command the query buffer is cleared if another query is typed in.

Editing a Query in the Query Buffer

It is possible to edit a command if you make an error. For example, enter the following query containing a typographical error.

```
1> select pname, quantity
2> from   paart
3> where  pno in (1,2,3)
4> go
```

You will get an error message. After a careful review, you will find that the relation name "paart" should be "parts". Then type:

```
1> emacs
```

An emacs editor will be invoked.[5] You can edit the query buffer and correct the error. After you have done, type `Ctrl-x-Ctrl-s` to save the correct query and `Ctrl-x-Ctrl-c` to return to SQL. After you have corrected the mistake and returned to SQL, rerun the query by typing:

[5] If you prefer vi, then type `vi` instead to invoke vi editor.

where "nnnn" is the last four digits of your student ID number and "aa" is your first and last initials in lower case. [2]

Type the following command at your operating system prompt:

```
isql
```

On your screen, you will then see the prompt:

```
Password:
```

Type in your password and press the return key. Then you will see:

```
1>
```

1.1.3 Changing Password

Once you have logged in, you can change your password at anytime [3]. Your new password must be at least 6 bytes long. Assuming your password is '1234xx', here is how you'd change the password '1234xx' to 'topsecret':

```
1> sp_password '1234xx', topsecret
2> go
```

Notice that the word go appears on a line by itself and must **not** be preceded by blanks. It is the command terminator, and lets SQL server know that you are done typing, and you are ready for your command to be executed. Also notice that the old password '1234xx' is enclosed in quotes. A password starting with a numeral must be enclosed in quotes.

1.1.4 Copying Tutorial Tables

Before you go on, you must copy all of the tables in the tutorial database. Type the following command:[4]

```
1> OSUsp_CopyTutorial
2> go
```

The set of tutorial tables, including "parts", "supply" and "employee", will be used for every examples in this tutorial.

The OSUsp_CopyTutorial procedure will give you a fresh copy of the tutorial tables. However, you have to first make sure that no tables with the same names of the tutorial tables existing. (See the next section.)

1.1.5 Clearing Tutorial Tables

If you decide to delete all of the tutorial tables, type the following command:

```
1> OSUsp_ClearTutorial
2> go
```

1.1.6 Listing User Tables

To list all of the objects, including tables, indexes and views, created by you, type:

```
1> OSUsp_help
2> go
```

[2] Your existing Unix password cannot be used since only its encrypted version is stored.

[3] Since your SYBASE password is stored unencrypted in a SYBASE table readable by the database administrator, you should choose a different SYBASE password.

[4] The "sp" in the name stands for "stored procedure". Users may create stored procedures for later use. By convention, their names contain "sp".

Chapter 1

A Tutorial On SYBASE

Using the SYBASE[1] Relational Database System

Jing Xiang, Wang-chien Lee, Douglas S. Kerr, David Ebert
Department of Computer and Information Science
The Ohio State University
Version 2.3
August 1995

This tutorial has been designed to familiarize you with the SYBASE relational database system. You should be able to follow the examples given here and observe the same results.

1.1 Getting Started

The query language we are using is SQL. This tutorial provides an introduction to SQL on SYBASE. Complete information on SQL and SYBASE appears in the *SQL Server Quick Reference Guide*, *Transact-SQL User's Guide*, and *SQL Server Reference Manual Volume I and II*. These manuals are available on-line using 'sybooks'. They are in the *SQL Server Collection*.

Before you can use SYBASE, there are a few things that must be set up:

1.1.1 Setting up SYBASE environment

In order to access SYBASE, you must add "SYBASE" to your .subscriptions file. Do this by running

```
subscribe SYBASE
```

To have access to the SYBASE manuals you must also subscribe to SYBOOKS:

```
subscribe SYBOOKS
```

1.1.2 Entering SYBASE

Now you can start using SQL from the operating system. To use SQL, you must have an account for SYBASE's SQL server. SYBASE maintains its own password file distinct from the Unix password file. Your SYBASE account will be set up with your Unix user name and the standard default password "nnnnaa"

[1] SYBASE is a trademark of Sybase, Inc.

Contents

1 A Tutorial On SYBASE **3**

1.1 Getting Started . . . 3
1.1.1 Setting up SYBASE environment . . . 3
1.1.2 Entering SYBASE . . . 3
1.1.3 Changing Password . . . 4
1.1.4 Copying Tutorial Tables . . . 4
1.1.5 Clearing Tutorial Tables . . . 4
1.1.6 Listing User Tables . . . 4
1.1.7 The Query Buffer . . . 5
1.1.8 Exiting SYBASE . . . 7
1.1.9 Summary of Commands Related to Query Buffer . . . 7
1.2 The Tutorial Database . . . 7
1.2.1 Some Query Examples . . . 8
1.2.2 How to modify the database . . . 13
1.3 Database Creation . . . 14
1.3.1 How to create a table . . . 14
1.3.2 Insert data into table . . . 15
1.3.3 Insert with Selection . . . 15
1.3.4 Destroy a table . . . 16
1.4 Adding, Changing, and Deleting columns . . . 16
1.4.1 Adding columns . . . 16
1.4.2 Changing columns . . . 16
1.4.3 Deleting columns . . . 17
1.5 How to create views . . . 17
1.5.1 An example of creating view . . . 17
1.5.2 Destroy view . . . 18
1.6 How to create an index . . . 18
1.6.1 An example of creating index . . . 18
1.6.2 Destroy index . . . 18
1.7 How to give out privilege of access to your tables . . . 18
1.8 Help Commands . . . 18
1.9 Date and Time in Sybase (Section may be skipped on first reading.) . . . 19
1.9.1 Date/Time Data Types . . . 19
1.9.2 Display Format . . . 19
1.9.3 Other formats using the *convert* function . . . 19
1.9.4 Parts of a date or time using the *datename* and *datepart* functions . . . 20
1.9.5 Date/Time Data Entry Format . . . 20
1.10 User-Defined Datatypes (Section may be skipped on first reading.) . . . 21