Part 2: First Introduction to the Relational Model and SQL

Elmasri/Navathe:

- Section 7.1, “Relational Model Concepts”
- Section 8.2, “Basic Queries in SQL”

Objectives

- This part should give you enough information so that you can formulate simple SQL queries.
- Together with self study of appendices A and B you should be able to execute queries under Oracle SQL*Plus.

  If you have problems with this, please contact the GSA or the instructor.
- We will treat the relational model and SQL in much more depth later in this course.
- However, it might be good if you start practicing SQL and using Oracle early.

  Also, some students already know the relational model, and I want to bring you to the same level.

Overview

1. The Relational Model, Example Database
2. Using SQL*Plus: First Demonstration
3. Simple SQL Queries

The Relational Model (1)

- The relational model structures the data in form of tables.
- So a relational database schema defines:
  - The names of the tables in the database.
  - The columns of each table, i.e. the names of the columns and the data types of the column entries.
  - Plus integrity constraints (to be explained later).
- For instance, Oracle comes with an example database containing information about employees (table `EMP`) and their departments (`DEPT`).

  There are three further tables: `SALGRADE`, `BONUS`, `DUMMY`.

The Relational Model (2)

- The Table DEPT has the columns DEPTNO (Department Number), DNAME (Department Name), LOC (Location):

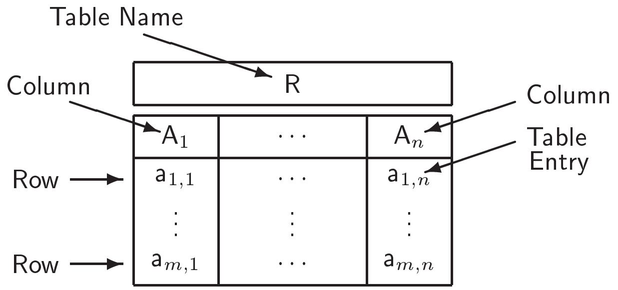
DEPT		
DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

- DEPTNO has the data type NUMBER(2) (two-digit integer), DNAME/LOC have string types (of max. 14/13 characters).

The Relational Model (3)

- A relational database state (instance of a given schema) defines for each table a set of rows.
- In the example, the table “DEPT” has currently four rows.
- The relational model does not define any particular order of the rows (e.g. which is the first one, the second, etc.).
  Of course, you can order the rows when you output them.
- Each row specifies values for each column of the table.
  E.g. one row above has the value 10 for the column DEPTNO, “ACCOUNTING” for DNAME, and “NEW YORK” for LOC.
- Of course these values must be compatible with the data types declared in the database schema for this column.

Summary



Synonyms: Relation and table.
Tuple, row, and record.
Attribute, column, field.
Attribute value, column value, table entry.

More on the Relational Model (1)

The second table of the example database, EMP, contains information about employees. It has the following columns:

- EMPNO: A unique number for every employee.
- ENAME: Name of the employee.
- JOB: Position of the employee (e.g. CLERK).
- MGR: Number of the direct supervisor of this employee.
- HIREDATE: Date when the employee started to work.
- SAL: Salary of the employee.
- COMM: Commission (only for salespeople).
- DEPTNO: Number of the department this employee works for.

More on the Relational Model (2)

EMP							
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

More on the Relational Model (3)

References (Foreign Keys):

- The relational model has no physical pointers.
- But it allows to compare column values of different rows.
- For instance, the column `DEPTNO` in the `EMP` table refers to the table `DEPT` (`DEPTNO` uniquely identifies rows in `DEPT`).

  `MGR` even refers to another row in the same table.

Null Values:

- The relational model in general allows that table entries remain empty (contain a “null value”).
- In the example table, this happens only for the commission and the manager of the president.

Overview

1. The Relational Model, Example Database
2. Using SQL*Plus: First Demonstration
3. Simple SQL Queries

SQL, Oracle, and SQL*Plus (1)

- SQL is the standard database language for relational database management systems (RDBMS).
- Oracle supports this language, but you can use the same language to work with all other modern RDBMS (e.g. DB2).

  The systems differ in many small details, see below.
- SQL*Plus is Oracle’s basic interface to the database.

  The database itself (DB server) runs as a set of background processes, maybe on another machine in the network.
- However, if you write a C program with embedded SQL, you also communicate with the Oracle server in SQL, but you do not use SQL*Plus (many interfaces to the same server).

SQL, Oracle, and SQL*Plus (2)

- So the main task of SQL*Plus is
  - to allow you to enter SQL statements,

    It gives you some editing functions, e.g. to correct errors.
  - send the query to the server, fetch the result,
  - and print the output table.

    It gives you some control over the output formatting. Simpler reports can be developed entirely in SQL*Plus.
- SQL*Plus also can process batch files (with several commands). This is also supported by a simple variable replacement mechanism.
- So SQL*Plus has some additional commands to SQL.

Basic Use of SQL*Plus (1)

- Under UNIX, you enter the command "`sqlplus`" to get into the SQL interpreter.
- Under Windows, you select "SQL*Plus 8.0" under "Oracle for Windows 95" in the "Programs" menu.
- SQL*Plus then asks for username and password.
- When you are logged in, you see the prompt "`SQL>`".
- Oracle normally has a guest login: User "`scott`" with password "`tiger`". The example tables (`EMP`, `DEPT`, etc.) are installed under this account.

  Under UNIX, execute "`demobld`" (from the UNIX prompt) to install the demonstration tables under any account.

Basic Use of SQL*Plus (2)

- In SQL*Plus, SQL statements have to end with a semicolon.

```
SELECT * FROM DEPT;
```

- The reason is that SQL statements can extend over several lines, and SQL*Plus needs to know when you are finished.

  The semicolon is not part of SQL, other systems may have other rules. Alternative in SQL*Plus: "/" in the next line.
- SQL is not case sensitive (except in strings).
- You leave SQL*Plus with "`exit`" or "`quit`".

  Such control commands of SQL*Plus do not need a semicolon at the end. If you need to write formatting commands which extend over multiple lines, each line (except the last) needs to finish with a hyphen ("-").

Basic Use of SQL*Plus (3)

- The last SQL command is still stored in a buffer. You can change and re-execute it (e.g. in case of a syntax error).
- `c/old/new` replaces the first occurrence of "`old`" by "`new`".

  In the current line (line in which the error was detected).
- `l` (`list`) shows the contents of the buffer.

  E.g. "`l3`" shows line 3 and makes it the current one.
- `r` (`run`) executes the contents of the buffer.
- `edit` writes the contents of the buffer to a file and calls an operating system editor (which you can select → Appendix B).
- You can also write the SQL query into a file, e.g. "`x.sql`", and use `@x` to execute the file.

Creating a New User Account

- If you have Oracle on your own PC, you can log in as user “`system`” with password “`manager`”.
- This is the standard DBA account and gives you a lot of privileges which normal users do not have.

  Most powerful: “`sys`” with password “`change_on_install`”.
- You can then create a new user with

```
CREATE USER sbrass IDENTIFIED BY my_password
            DEFAULT TABLESPACE user_data
            TEMPORARY TABLESPACE temporary_data;
```

- Before the user can actually work, you must also execute

```
GRANT connect, resource TO sbrass;
```


Overview

Simple SQL Queries (1)

- SQL is the standard language for relational databases (for writing queries, updates, and schema definitions).
- Simple SQL queries have the structure

```
SELECT ...  FROM ...  WHERE ...
```

- After `FROM` you list the table (or the tables) from which you want to extract data.
- After `WHERE` you can specify conditions for the rows you want to select. The `WHERE`-clause can be missing.

  Without `WHERE` all rows are selected.
- After `SELECT` you define which columns you want to print.

  “*” prints all columns.

Simple SQL Queries (2)

- Show the entire department table:

```
SELECT * FROM DEPT
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

- Equivalent Solution:

```
SELECT DEPTNO, DNAME, LOC FROM DEPT
```

Simple SQL Queries (3)

- SQL is in general not case-sensitive.
  All characters in an SQL command are turned into uppercase before they are processed (except in strings).
  If you really want a table or column name containing lower case letters, you must enclose it in ".
- SQL is also format-free: You can put line breaks, spaces, and tabulator characters between the words of an SQL command.
- So the following query is equivalent, too:

```
select deptno, dname, loc
from   dept
```


Using Conditions (1)

- Give all data about the department in "DALLAS".

```
SELECT * FROM DEPT WHERE LOC = 'DALLAS'
```

DEPTNO	DNAME	LOC
20	RESEARCH	DALLAS

- Note that string constants are marked by single quotes.
  Double quotes are only used for delimited identifiers (column names containing lowercase characters etc.).
- Note that inside string constants, SQL is case-sensitive.
  So the following will select 0 rows (empty result):

```
SELECT * FROM DEPT WHERE LOC='Dallas'
```


Using Conditions (2)

- Print the name, job, and salary of all employees who earn at least $2500:

```
SELECT ENAME, JOB, SAL
FROM   EMP
WHERE  SAL >= 2500
```

ENAME	JOB	SAL
JONES	MANAGER	2975
BLAKE	MANAGER	2850
SCOTT	ANALYST	3000
KING	PRESIDENT	5000
FORD	ANALYST	3000

Using Conditions (3)

- Note that it is not necessary to output columns which you use in the conditions. WHERE is evaluated before SELECT.
- E.g. print the employee number and name of all managers:

```
SELECT EMPNO, ENAME
FROM   EMP
WHERE  JOB = 'MANAGER'
```

EMPNO	ENAME
7566	JONES
7698	BLAKE
7782	CLARK

Pattern Matching with LIKE

- SQL also has an operator for “pattern matching” of strings (allowing the use of “wildcards”).
- E.g. print number and name of all managers (again):

```
SELECT EMPNO, ENAME
FROM   EMP
WHERE  JOB LIKE 'MANA%'
```

- “%” matches any sequence of arbitrary characters, “_” matches any single character.

  So “%” corresponds to the “*” in the shell, and “_” to “?”.
- Note that you must use LIKE if you want to do pattern matching. The equals sign only tests for literal equality.

Arithmetic Expressions

- You can use standard arithmetic expressions in SQL.
- E.g. print all employees who earn less than $15000 per year (print also this yearly salary):

```
SELECT ENAME, 12 * SAL
FROM   EMP
WHERE  SAL < 15000 / 12
```

ENAME	12 * SAL
SMITH	9600
ADAMS	13200
JAMES	11400

Logical Connectives (1)

- You can use AND, OR, NOT and parentheses “(”, “)” to construct more complicated conditions.
- E.g. print name and salary of all managers and the president:

```
SELECT ENAME, SAL
FROM   EMP
WHERE  JOB = 'MANAGER' OR JOB = 'PRESIDENT'
```

- This query would not work with “AND” instead of “OR”.

  It would select 0 rows, since the JOB cannot be “MANAGER” and “PRESIDENT” at the same time. The WHERE-condition is conceptionally evaluated for every row of EMP. AND is true if both parts are true, OR is already true if one part is true.

Logical Connectives (2)

C1	C2	C1 AND C2	C1 OR C2	NOT C1
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

- E.g. C1 is `JOB = 'MANAGER'`, C2 is `JOB = 'PRESIDENT'`.
- For every given row, only one or the other can be true.

  Or both can be false.
- Thus, “C1 AND C2” will be false.

Logical Connectives (3)

- If you do not put explicit parentheses, then AND binds stronger than OR (and NOT binds even stronger than AND).
- E.g. consider this query:

```
SELECT ENAME, SAL            Wrong!
FROM   EMP
WHERE  JOB = 'MANAGER' OR JOB = 'PRESIDENT'
AND    SAL >= 3000
```

- The system will understand the condition as:

```
WHERE JOB = 'MANAGER'
OR    (JOB = 'PRESIDENT' AND SAL >= 3000)
```


Joining Tables

- You can combine data from different tables. E.g. print the names of employees from the "RESEARCH" department:

```
SELECT ENAME
FROM   EMP, DEPT
WHERE  EMP.DEPTNO = DEPT.DEPTNO
AND    DNAME = 'RESEARCH'
```

- Conceptually, the WHERE-condition will be evaluated for every combination of one row from each table.
- Note that if an attribute appears in more than one table, you must make the name unique by adding the table name.
- It is a common error to forget the "join condition".

Removing Duplicates

- Sometimes queries can produce duplicate rows.
- E.g. list all jobs:

```
SELECT JOB FROM EMP
```

- Since this query is processed by a loop over the employee table, in which the job of every employee is printed, it will output the same job multiple times.
- You can request duplicate elimination by adding the keyword DISTINCT after SELECT:

```
SELECT DISTINCT JOB FROM EMP
```

- DISTINCT works on output rows. So you specify it only once, even if you print multiple columns.

Sorting Output Rows

- Print the employees who earn at least 2000 dollars, ordered by department number. For equal department numbers, they should be ordered by descending salaries.

```
SELECT   DEPTNO, ENAME, SAL
FROM     EMP
WHERE    SAL >= 2000
ORDER BY DEPTNO, SAL DESC
```

- Note that the "ORDER BY" clause is pure cosmetics:
  It does not change the query result in any way, it only prints it somewhat nicer (→ output formatting).
  Good reports should be sorted in some understandable way.