Part 2: First Introduction to the Relational Model and SQL

References:

- Elmasri/Navathe:Fundamentals of Database Systems, 3rd Edition, 1999.
  Section 7.1, "Relational Model Concepts"
  Section 8.2, "Basic Queries in SQL"
- Kemper/Eickler: Datenbanksysteme (in German), 3rd Edition, 1999.
  Section 3.1, "Definition des relationalen Modells" ("Definition of the Relational Model")
  Section 4.6, "Einfache SQL-Anfragen" ("Simple SQL Queries")

Objectives

- Basic notions of the relational model.
- Simple SQL queries.
- Introduction to using Oracle SQL*Plus.
- Readiness for practical homework assignments.

Overview

The Relational Model (1)

- The relational model structures data in table form.
- So a relational database schema defines:
  - Names of tables in the database.
  - The columns of each table, i.e. the names of the columns and the data types of the column entries.
  - Integrity constraints (conditions which must be satisfied).
- 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 `NUMERIC(2)` (two-digit integer), `DNAME`/`LOC` have string types (of max. 14/13 characters).

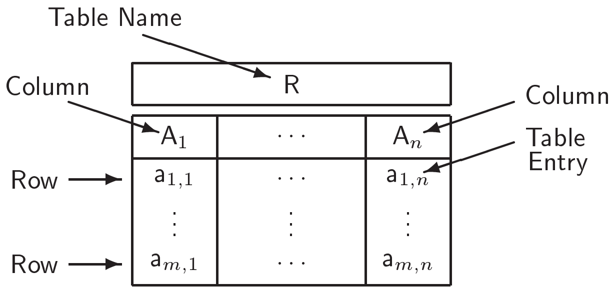
The Relational Model (3)

- 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 first, second, etc.).

  But rows can be sorted for output.
- 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`.
- 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)

`EMP` (information about employees) has the following columns:

- `EMPNO`: A unique number for every employee.
- `ENAME`: Employee name.
- `JOB`: Employee position (e.g. `CLERK`).
- `MGR`: Employee number (`EMPNO`) for this employee's direct supervisor.
- `HIREDATE`: Employee hire date.
- `SAL`: Employee salary.
- `COMM`: Commission (only for salespeople).
- `DEPTNO`: Number of the department where this employee works.

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 comparison of column values in 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 allows table entries to remain empty (contain a "null value").
- In the example table: Only salespeople have a commission, the company president has no manager (supervisor).

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 SQL, as do all other modern RDBMS.

  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, a C program with embedded SQL also communicates with the Oracle server in SQL, but does not use SQL*Plus.

  There are many interfaces to the same server.

SQL, Oracle, and SQL*Plus (2)

- The main task of SQL*Plus is
  - to allow the user to enter SQL statements,

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

    It provides some control over 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, enter the command "`sqlplus`".

  Environment variables must be set, see Appendix B.
- Under Windows, select "Start → Programs → Oracle-OraHome81 → Application Development → SQL Plus".
- SQL*Plus then asks for username and password.

  The "host string" field in the login box can be left empty.
- Many Oracle installations have a guest user "`scott`" with password "`tiger`" (with the example tables `EMP`, `DEPT`, etc.).

  To install the tables under another account, execute "`demobld`" under UNIX from the UNIX prompt or "`@C:\Oracle\Ora81\DBS\demobld.sql`" under Windows from the SQL*Plus prompt.

Basic Use of SQL*Plus (2)

- The SQL*Plus prompt is "`SQL>`".

  Or "2", "3", etc. for continuation lines.
- In SQL*Plus, SQL statements must end with a semicolon.

  Alternative: "/" as the only contents of the next line.
- SQL statements can extend over several lines, and SQL*Plus needs to know when the user is finished.

  The ";" is not part of SQL. In DB2/SQL server, the user must instead click on "execute" when the query is complete.
- Leave SQL*Plus with "`exit`" or "`quit`".

  In contrast to SQL statements, such control commands need no semicolon at the end. If they extend over multiple lines, each line (except the last) must end with a hyphen "-".

Basic Use of SQL*Plus (3)

```
UNIX> sqlplus
...  (Version and copyright information for SQL*Plus)

Enter user-name:  scott
Enter password:  tiger (not visible)
...  (Version information for the database server)

SQL> select * from dept;
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

Basic Use of SQL*Plus (4)

- 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 editor (which can be selected by the user, see Appendix B).
- The user can also write the SQL query into a file, e.g. "`x.sql`", and execute the file with `@x`.

Basic Use of SQL*Plus (5)

```
SQL> select *
  2  frm cat;
frm cat
*
ERROR at line 2:
ORA-00923: FROM keyword not found where expected

SQL> c/frm/from/
  2* from cat

SQL> r
  1  select *
  2* from cat
```

(shows the catalog, i.e. lists tables/views owned by current user)

Basic Use of SQL*Plus (6)

- If the first keyword is misspelled, the command is not stored in the buffer, and must be retyped.
  Only SQL statements (recognized by first keyword) are stored in the buffer, not SQL*Plus control commands.
- Oracle reports only one error in the SQL command.
  There might be more errors. The position which Oracle marks is not necessarily the first position where a parser could find an error (e.g. quotes are missing around strings):

```
SQL> insert into dept values(40, WWW, DALLAS);
insert into dept values(40, WWW, DALLAS)
                                 *
ERROR at line 1:
ORA-00984:  column not allowed here
```


Overview

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

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 list the table from which to extract data.
  More than one table can be listed, see below.
- After WHERE specify conditions for the rows to be selected.
  The WHERE-clause can be missing, then all rows are selected.
- After SELECT define which columns 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 not case-sensitive, except inside strings.
  All characters in an SQL command are turned into uppercase before they are processed (except inside quotes). To get a table or column name containing lower case letters, it must be enclosed in " (should be avoided).
- SQL is format-free (like Pascal, C, Java, etc):
  Line breaks, spaces, and tabulator characters can be inserted between the words (tokens) of an SQL command.
- Thus, the following query is the same as the previous one:

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


Using Conditions (1)

- To get all data about the department in "DALLAS":

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

DEPTNO	DNAME	LOC
20	RESEARCH	DALLAS

- String constants are marked by single quotes.
  Double quotes are only used for delimited identifiers (column names containing lowercase characters etc.).
- 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)

- It is not necessary to output columns which are used 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 "?".
- LIKE must be used for pattern matching. The equals sign only tests for literal equality, even if the strings contain "%".

Arithmetic Expressions

- Standard arithmetic expressions are allowed in SQL.
- E.g. print all employees who earn less than $15000 per year:

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

ENAME
SMITH
ADAMS
JAMES

- The condition "SAL * 12 < 15000" is equivalent.

Renaming Output Columns

- E.g. print the yearly salary of all managers:

```
SELECT ENAME, SAL * 12
FROM   EMP
WHERE  JOB = 'MANAGER'
```

ENAME	SAL * 12
JONES	35700
BLAKE	34200
CLARK	29400

- To print the column heading “YEARLY SALARY”:

```
   SELECT ENAME, SAL * 12 "YEARLY SALARY" ...
or SELECT ENAME, SAL * 12 AS "YEARLY SALARY" ...
```


Logical Connectives (1)

- AND, OR, NOT and parentheses “(”, “)” can be used 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. I.e. only the first three cases occur.
- Thus, “C1 AND C2” will be false.

Logical Connectives (3)

- Without parentheses, AND binds more strongly than OR (and NOT binds even more strongly than AND).
- E.g. consider this query:

Wrong!

```
SELECT ENAME, SAL
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)
```

Removing Duplicates

- 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, not single columns. Specify it only once, even if printing multiple columns.

Sorting Output Rows

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

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

- The "`ORDER BY`" clause is purely cosmetic:
  It does not change the query result in any way, it only prints the result in a more readable fashion.

  See the section on output formatting in Appendix B for more information on the nice presentation of query results.

Outlook: Joining Tables

- Data from different tables can be combined. 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.
- If a column appears in more than one table, the name must be made unique by adding the table name.
- It is a common error to forget the "join condition".

Exercises

Please formulate the following queries in SQL:

- Who has employee number 7839 (King) as direct supervisor?
- Who has a salary between \$1000 and \$2000 (inclusive)? Print name and salary and order the result by names.
- Which employee names consist of exactly four characters?
- Print name, salary, and department of all employees who work in department 10 or 30 and earn less than \$1500.

  Make sure that both conditions are really satisfied.
- Which jobs occur in which departments? I.e. print every combination of department number and job which occurs in the `EMP` table, and print each such combination only once.