Introduction to the Relational Model and SQL

- After completing this chapter, you should be able to:
  - ▷ explain **basic notions** of the **relational model**:
    - table/relation, row/tuple, column/attribute, column value/attribute value,
  - ▷ explain the meaning of **keys** and **foreign keys**,
  - ▷ write **simple SQL queries** (queries against one table).

Introduction to the Relational Model and SQL

Overview

The Relational Model (1)

- The **relational model** structures data in **table form**, *i.e.*, a relational DB is a set of named tables.

A relational database schema defines:

① **Names of tables** in the database,

② the **columns of each table**, *i.e.*, the **column name** and the **data types** of the column entries,

③ **integrity constraints**, *i.e.*, conditions that data entered into the tables is required to satisfy.

The Relational Model (2)

- **Example:** Assume a database maintaning information about a small real-world subset: a company's departments and employees.
- Two tables:
  - ▷ `EMP`: information about employees.
  - ▷ `DEPT`: information about departments.

Table `DEPT`

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

The Relational Model (3)

- The three **columns** of table `DEPT` have the following **data types**:
  - `DEPTNO` has data type `NUMERIC(2)`, *i.e.*, the column can hold two-digit integer entries $-99 \ldots 99$.

    An integrity constraint can be used to exclude negative department numbers.
  - `DNAME` has type `VARCHAR(14)`, *i.e.*, the entries are character strings of variable length of up to 14 characters.
  - `LOC` has type `VARCHAR(13)`.

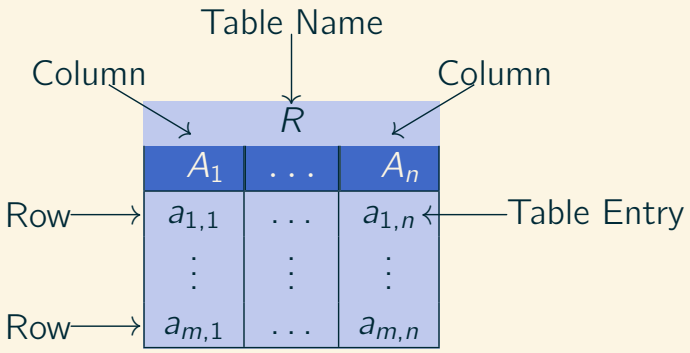
The Relational Model (4)

- A **relational database state** (instance of a given schema) defines **for each table a set of rows.**
- In the now current state, table `DEPT` has four rows.
- The relational model **does not define any particular order of the rows** (*e.g.*, first row, second row).

  Rows can be sorted for output, though.
- Each row specifies values for each column of the table.

Summary (1)

A relational table



Summary (2)

- A more theoretically inclined person would use the following equivalent terminology:
  - ▷ Table ≡ **relation**

    Formally, a table is a subset of the Cartesian product of the domains of the column data types, *i.e.*, a relation in the mathematical sense.
    Example: *Cartesian coordinates* are (X, Y)-tuples of real numbers, *i.e.*, elements of $\mathbb{R} \times \mathbb{R}$. The relation $<$ defines a subset of $\mathbb{R} \times \mathbb{R}$, *e.g.*, $(1, 2)$ is contained in $<$, while $(2, 1)$ is not.
    In an RDBMS, relations are always finite and may involve more than two columns.
  - ▷ Row ≡ **tuple**
  - ▷ Column ≡ **attribute**

Summary (3)

- "Old-style" practitioners might equivalently say
  - ▷ Row ≡ **record**

    A table row (or tuple) is basically the same as a `record` in Pascal or `struct` in C: it has several named components. However, the storage structure of a tuple in external memory (disk) does not necessarily match the layout of a record in main memory.
  - ▷ Column ≡ **field**
  - ▷ Table entry ≡ **field value**
  - ▷ Table ≡ **file**

Keys (1)

- The column `DEPTNO` is declared as the **key** of table `DEPT`.

Relational Keys

A key always **uniquely identifies a single row** in its associated table.

- Table `DEPT`, for example, already contains a row with key `DEPTNO` = `10`.
- If one tries to add another row with the same value `10` for `DEPTNO`, the DBMS responds with an error.

Keys (2)

- Keys are an example of **constraints:** conditions that the table contents (DB state) must satisfy *in addition* to the basic structure prescribed by the columns.
- Constraints are declared as part of the DB schema.
- More than one key can be declared for a table

More keys?

One could, for example, discuss whether `DNAME` should also be a key (in addition to `DEPTNO` already being a key). This would exclude the possibility that there can ever be two or more departments of the same name.

- Keys and other constraints are treated more fully in Chapter 2.

Another Example Table (1)

- Table `EMP` (data about employees) with the following columns:
  - `EMPNO`: A unique number for every employee.
  - `ENAME`: Employee name.
  - `JOB`: Employee position (*e.g.*, `ENGINEER`)
  - `MGR`: Direct supervisor of this employee.
  - `HIREDATE`: Employee hire date.
  - `SAL`: Employee salary.
  - `COMM`: Commission (only for salespeople).
  - `DEPTNO`: Department where this employee works.

Another Example Table (2)

Table EMP

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	20
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	20
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		20
7782	CLARK	MANAGER	7839	09-JUN-81	2450		20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		20
7844	TURNER	SALESMAN	7698	09-SEP-81	1500	0	20
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		20
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		20

Foreign Keys (1)

- **Physical pointers are unknown** to the relational model.
- However, the relational model provides a kind of "**logical pointer.**"
  - ▷ For example, the column DEPTNO in table EMP **refers to** column DEPTNO in table DEPT.
- Since DEPTNO is declared as a key in table DEPT, a department number uniquely identifies a single row of DEPT.
- A value for DEPTNO (in table EMP) can be seen as a "logical address" of a row in table DEPT.

Foreign Keys (2)

- By including a department number in table `EMP`, each row in `EMP` "points to" a single row in table `DEPT`.
- It is important for the integrity of the database, that the department numbers in `EMP` in fact occur in `DEPT`.

"Dangling pointers"

If a row in table `EMP` contains a `DEPTNO` value of `70`, the reference "dangles."

A DBMS, however, does *not* crash as would be the case with real physical pointers (memory addresses). In SQL, references are followed by comparing column values. Nevertheless, such a column entry is a kind of error and should be avoided. This is the purpose of **foreign keys.**

Foreign Keys (3)

- The relational model permits to declare column `DEPTNO` as a **foreign key** that references table `DEPT`.

Foreign Keys

▷ Then the DBMS will refuse

- an insertion into table `EMP` with a value for `DEPTNO` that **does not appear** in `DEPT`,
- a deletion of a row in `DEPT` that is **still referenced** by a row in `EMP`,[a]
- corresponding updates (changes) of `DEPTNO` values.

[a] It might be reasonable to recursively delete all employees of that department, too (cascading delete).

Foreign Keys (4)

- Table `EMP` also contains a second foreign key:

  Column `MGR` contains the employee number of the employee's direct supervisor.

- This shows that
  - ▷ it is possible that a foreign key refer to another row in the **same table** (or even the **same row**),
  - ▷ the foreign key column and referenced key may have **different names.**

Null Values

- The relational model allows column entries to remain empty (*i.e.*, the column/row contains a **null value**).
- For example, in table `EMP`:
  1. only salespeople have a commission,
  2. the company president has no supervisor.
- When a table schema is declared, one can specify for each column whether null values are accepted or not.
- Null values are treated specially in comparisons.

Introduction to the Relational Model and SQL

Overview

1. The Relational Model, Example Database
2. Simple SQL Queries
3. Historical Remarks

Simple SQL Queries (1)

- Simple SQL queries have the following syntactic form:

  `SELECT ··· FROM ··· WHERE ···`

  ▷ After `FROM`: comma-separated list of **table names** from which to extract data.

  ▷ After `WHERE`: specify predicate (Boolean expression) which selected **rows** are required to satisfy.

  A missing `WHERE`-clause is equivalent to `WHERE TRUE`.

  ▷ After `SELECT`: define which **columns** appear in the result table.

  Note: `SELECT *` selects all columns

Simple SQL Queries (2)

Show the entire department table (all rows, all columns)

```
SELECT * FROM DEPT
```

- Result:

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

- Equivalent formulation:

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


Simple SQL Queries (3)

- SQL is *not* case-sensitive, except inside string constants.

  The SQL parser converts all characters into upper case (except inside quotes) before further processing continues. To refer to a table/column name containing lower case characters, enclose the name in double quotes (").

- The syntax of SQL is format-free (newlines, whitespaces, *etc.* may be arbitrarily used).

Commonly found layout of `SELECT-FROM-WHERE` blocks

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

Using Conditions (1)

To extract all data about the department in DALLAS

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

- Result:

DEPTNO	DNAME	LOC
20	RESEARCH	DALLAS

- String constants are enclosed in single quotes (').
- Inside string constants, SQL is case-sensitive. 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 earning at least $2500

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

- Result:

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

Using Conditions (3)

- SQL evaluates `WHERE` **before** `SELECT`:
  The `WHERE`-clause may refer to columns which do not appear in the result.

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 (1)

- SQL also provides an operator for text **pattern matching** allowing the use of wildcards:

Print the employee number and name of all managers

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

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

- Cf. UNIX shell globbing via $*$ and ?.

Pattern Matching (2)

- The SQL keyword `LIKE` needs to be used for pattern matching. The equals sign (=) tests for literal equality.
- The following is legal SQL but will return the empty result:

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


Arithmetic Expressions

- SQL provides the standard arithmetic operators.

Print employee names earning less than $15,000 per year

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

ENAME
SMITH
ADAMS
JAMES

- The predicate `SAL * 12 < 15000` is equivalent (column names act like variable identifiers in programming languages).

Renaming Output Columns

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 rename the second result column:

```
SELECT ENAME, SAL * 12 [AS] YEARLY_SALARY
```


Logical Connectives (1)

- The Boolean operators `AND`, `OR`, `NOT` and parentheses `()` may be used to construct more complex conditions.

Print name and salary of all managers and the president

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

- The query would not work as expected with `AND` instead of `OR`.

Logical Connectives (2)

- Without parentheses, `AND` binds stronger than `OR` (and `NOT` binds stronger than `AND`).

Print name, salary of all well-paid managers and presidents

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

- The system will parse the condition as :

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


Removing Duplicate Rows

- In general, queries can produce duplicate rows.

List all jobs

```
SELECT JOB FROM EMP
```

- The DBMS processes this query by a loop over table `EMP` and extracts the job of every employee. The same job name will be extracted multiple times (and then printed by the SQL console).
- **Duplicate row elimination** may be requested by adding the keyword `DISTINCT` after `SELECT`:

List all distinct jobs

```
SELECT DISTINCT JOB FROM EMP
```

Sorting Output Rows (1)

- The **row ordering emitted by a SQL query is not predictable**, unless an explicit request for **sorting** is added.

Print employees names and salary, alphabetically ordered by employee name

```
SELECT    ENAME, SAL
FROM      EMP
ORDER BY  ENAME
```

- Most SQL implementations implement the `ORDER BY`-clause on the SQL console level only: the row printing order is changed.

Sorting Output Rows (2)

- The `ORDER BY`-clause permits multiple sorting criteria (useful only if one or more columns contain duplicate entries).

Print employees who earn $2000 at least, ordered by department number and descending salaries

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

- SQL orders lexicographically.
- Sort order is ascending (`ASC`) by default.

Outlook: Joining Tables

- SQL allows to **combine data from different tables.**

Print employees in the RESEARCH department

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

- Conceptually, SQL evaluates the WHERE-predicate for **every combination of rows from both tables** (Cartesian product).
- Since column name DEPTNO appears in both tables, disambiguate the column reference by prefixing the table name.

SQL Quiz (1)

Table EMP

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	20
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	20
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		20
7782	CLARK	MANAGER	7839	09-JUN-81	2450		20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		20
7844	TURNER	SALESMAN	7698	09-SEP-81	1500	0	20
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		20
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		20

SQL Quiz (2)

- Formulate the following queries in SQL:

Who has employee `KING` as direct supervisor?

Who has a salary between $1000 and $2000 (inclusive)? Print name and salary and order the result by names.

SQL Quiz (3)

Which employee names consist of exactly four characters?

Print name, salary, department of all employees who work in department `10` or `30` and earn less than $1500?

SQL Quiz (4)

Which jobs occur in which departments?

Introduction to the Relational Model and SQL

Overview

1. The Relational Model, Example Database
2. Simple SQL Queries
3. Historical Remarks

Relational Model: History

- The **relational model (RM)** was proposed by **Edgar F. Codd** (1970).

  It was the first data model that was theoretically defined prior to implementation. Codd got the Turing Prize in 1981.

- First implementations (1967):
  - ▷ **System R** (IBM)
  - ▷ **Ingres** (Mike Stonebraker, UC Berkeley)
- First commercial systems: **Oracle** (1979), **Ingres** (1980?), and **IBM SQL/DS** (1981).

Reasons for Success (1)

- **Much simpler** than earlier data models.

  One core concept: **finite relation** (set of tuples).

- Easily understandable, even for non-specialists:

  Relations correspond to tables.

- The theoretical model fits well with common implementation techniques:

  A relation is an abstraction of a **file of records**.

- The relational model features **set-oriented operation**. In earlier models, record-to-record navigation was explicit.

Reassons for Success (2)

- **Declarative query language.**

  A SQL query expresses the format and conditions that resulting tuples need to respect. The RDBMS contains a **query optimizer** that derives an efficient query evaluation plan (*i.e.*, an imperative program that evaluates the query) from this declarative specification. In earlier models, programmers had to explicitly think about the use of indexes and many more details.

- The relational model has a **solid theoretical foundation.** It is tightly connected to **first-order logic.**

Standards

- First standard 1986/87 (ANSI/ISO).

  This was late as there were already several SQL systems on the market. The standard was the "smallest common denominator', containing only the features common to existing implementations.

- Extension to foreign keys *etc.* in 1989 (SQL-89).

  This version is also called SQL-1. All commercial implementations today support SQL-89, but also feature significant extensions.

- Major extension: SQL-2 (SQL-92) (1992).

  Standard defines three levels, "entry", "intermediate", "full." Oracle 8.0, SQL Server 7.0 have entry level conformance.

Future

- Currently widely established standard: SQL-99.

  SQL-99 is a preliminary version of the SQL 2003 standard. Until 12/2000, the volumes 1–5 and 10 appeared (2355 pages; the SQL-2 standard, which is not completely implemented yet had 587 pages).

- Some features in SQL 2003 and SQL 2008 (most recent versions of the standard, not widely available as of today):
  - User-defined data type constructors:
    - `LIST`, `SET`, `ROW` to structure column values
  - OO features (*e.g.*, inheritance, subtables).
  - Recursive queries.