

# CS213

# Principles of Database Systems(H)

## Chapter 2

---

Shiqi YU 于仕琪

yusq@sustech.edu.cn

Most contents are from Stéphane Faroult's slides

# 2.1 Introduction to SQL

---

Shiqi Yu 于仕琪

yusq@sustech.edu.cn

Most contents are from Stéphane Faroult's slides

# How to manage data in a database?

**Special language needed!**

We must be able to use a language to query a database, either interactively or from within a program: a query language.

- Find the supplier names and locations of those suppliers who supply part 15.

$$(r_1[2], r_1[3]): P_1 r_1 \wedge \exists P_2 r_2 (r_2[2] = 15 \wedge r_2[1] = r_1[1]).$$

- Find the locations of suppliers and the parts being supplied by them (omitting those suppliers who are supplying no parts at this time).

$$(r_1[3], r_2[2]): P_1 r_1 \wedge P_2 r_2 \wedge (r_1[1] = r_2[1]).$$


# ALPHA

Codd worked on a language. It didn't excite enthusiasm at IBM.

## SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

by

Donald D. Chamberlin  
Raymond F. Boyce

IBM Research Laboratory  
San Jose, California



What the IBM management wanted was an "easy" language, with an English-like syntax. Here comes SQL (born SEQUEL)

**Don Chamberlin  
with Ray Boyce (+ 1974)**

**ABSTRACT:** In this paper we present the data manipulation facility for a structured English query language (SEQUEL) which can be used for accessing data in an integrated relational data base. Without resorting to the concepts of bound variables and quantifiers SEQUEL identifies a set of simple operations on tabular structures, which can be shown to be of equivalent power to the first order predicate calculus. A SEQUEL user is presented with a consistent set of bounded English templates which reflect how people use tables to

---

**select ...**

**from ...**

**where ...**

The basic syntax of SQL is very simple. SELECT is followed by the names of the columns you want to return, FROM by the name of the tables that you query, and WHERE by filtering conditions.

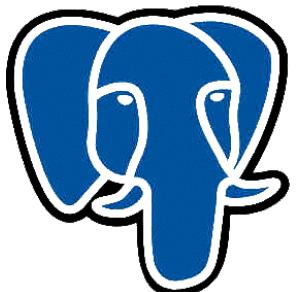
range of e is employee  
retrieve (comp = e.salary / (e.age - 18))  
where e.name = "Jones"

# QUEL

INGRES



PostgreSQL



SQL was just one of many competing languages. QUEL, born at Berkeley, and associated with INGRES, was highly regarded.



Michael  
Stonebraker

Movies		Movieid		Title		Country		Year Released
P.						IN		> 1985

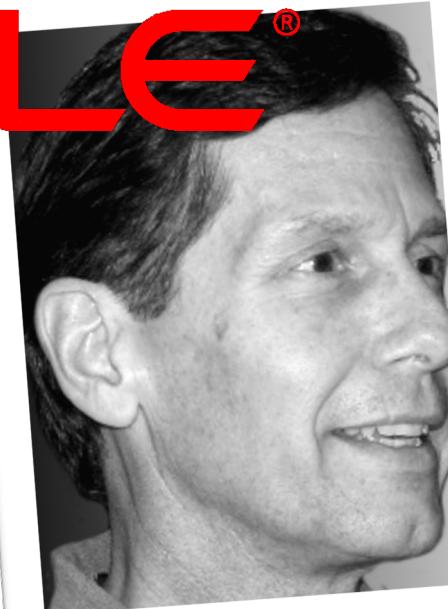
# QBE

Query By Example (QBE) was a visual querying tool (visual but with characters) also created at IBM. Some descendants of it still exist today.



Moshe Zloof

# ORACLE®



Oates



Ellison



Miner

Those three guys (the founders of Oracle) took SQL and created an SQL-compatible system on other systems than the IBM systems, thus helping turn SQL into a standard.

# TWO main components

Codd had required in his seminal paper that a good database language should allow to deal as easily with contents (data) as containers (tables), something that SQL does reasonably well.

# CREATE ALTER DROP



## Data Definition Language

The data definition language (usually called DDL) deals with tables (as well as other database "objects" that we'll see later).

Three commands are enough for creating a new table, changing its structure (for instance adding a new column) or deleting it.

# INSERT

<b>id</b>	<b>title</b>	<b>country</b>	<b>year_released</b>
1	Casablanca	us	1942
2	Blade Runner	us	1982
3	On The Waterfront	us	1954
4	Lawrence Of Arabia	uk	1962
5	Annie Hall	us	1977
6	Goodfellas	us	1990
7	The Third Man	uk	1949
8	Citizen Kane	us	1941
9	Bicycle Thieves	it	1948
10	The Battleship Potemkin	ru	1925
11	Sholay	in	1975
12	A Better Tomorrow	hk	1986

## Data Manipulation Language

# INSERTE

<b>id</b>	<b>title</b>	<b>country</b>	<b>year_released</b>
1	Casablanca	us	1942
2	Blade Runner	us	1982
3	On The Waterfront	us	1954
4	Lawrence Of Arabia	gb	1962
5	Annie Hall	us	1977
6	Goodfellas	us	1990
7	The Third Man	gb	1949
8	Citizen Kane	us	1941
9	Bicycle Thieves	it	1948
10	The Battleship Potemkin	ru	1925
11	Sholay	in	1975
12	A Better Tomorrow	hk	1986

## Data Manipulation Language

# DE

# DATE

<b>id</b>	<b>title</b>	<b>country</b>	<b>year_released</b>
1	Casablanca	us	1942
2	Blade Runner	us	1982
3	On The Waterfront	us	1954
4	Lawrence Of Arabia	gb	1962
6	Goodfellas	us	1990
7	The Third Man	gb	1949
8	Citizen Kane	us	1941
9	Bicycle Thieves	it	1948
10	The Battleship Potemkin	ru	1925
11	Sholay	in	1975
12	A Better Tomorrow	hk	1986

## Data Manipulation Language

# SELECT

<b>id</b>	<b>title</b>	<b>country</b>	<b>year_released</b>
1	Casablanca	us	1942
2	Blade Runner	us	1982
3	On The Waterfront	us	1954
4	Lawrence Of Arabia	gb	1962
5			
6	Goodfellas	us	1990
7	The Third Man	gb	1949
8	Citizen Kane	us	1941
9	Bicycle Thieves	it	1948
10	The Battleship Potemkin	ru	1925
11	Sholay	in	1975
12	A Better Tomorrow	hk	1986

## Data Manipulation Language

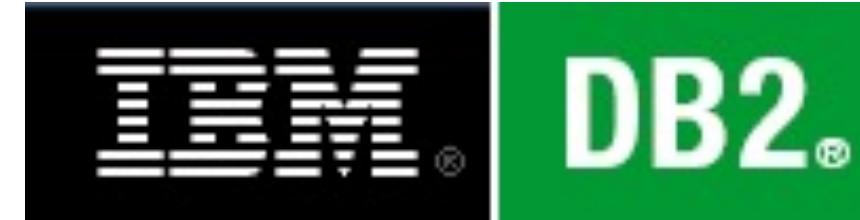
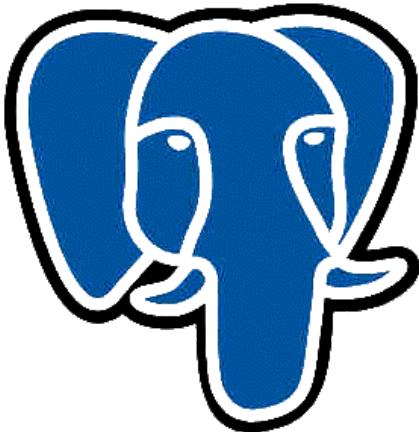
# SIMPLE or NOT SIMPLE?

As you can see it's simple. Like chess, it becomes complicated when you COMBINE operations . SQL is one of a few languages where you spend more time thinking about how you are going to do things than actually coding them.

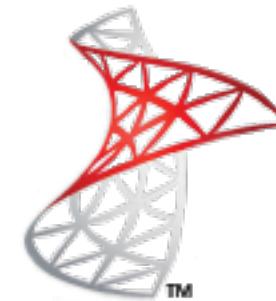


PostgreSQL

ORACLE®



There is an official SQL standard that no product fully implements, and many subtly and sometimes irritatingly different dialects. We'll see the main variants of SQL.



Microsoft®  
SQL Server®

---

It must be stressed that although for most people SQL is synonym with "relational database", SQL wasn't designed as a "relationally correct" language, but as an "easy language" that anybody could use. It implements features that are heretical for a database purist. More importantly, it's, in some respects, very lax. As a result, it's very easy to misuse, using it well is difficult, and there have been performance issues with SQL since circa 1974.

SQL  
a BIG  
misunderstanding

```
#include <stdio.h>

int main() {
    float a = 7.0;
    float b = 3.0;
    int c;
    c = (a / b) % 2;
    printf("c = %d\n", c);
    return 0;
}
```

To give an analogy, if in a C program you try to apply an operator (here the modulo operator, `%`) to variable types that don't support it (floats), the compiler won't let you go scot-free.

```
yushiqi@Shiqi-Yu-SurfacePro:~/Documents$ gcc hello.c -o hello
hello.c: In function 'main':
hello.c:6:16: error: invalid operands to binary % (have 'float' and 'int')
  c = (a / b) % 2;
          ~~~~~~ ^
```

Codd's operations, like the modulo, are only valid with certain types of "table variables". Contrary to the C compiler, an SQL engine won't complain when your tables don't fit the requirements of the theory. As a result, if you aren't rigorous enough, you may get wrong results without any warning.

---

Key property of relations (Codd's original paper):

**ALL ROWS ARE DISTINCT**

**CAN BE enforced for tables in SQL**

But you have to create your tables well.

**NOT enforced for query results in SQL**

You have to be extra-careful if the result of a query is the starting point for another query, which happens often.

# **2.2 Create Tables**

---

Shiqi YU 于仕琪

yusq@sustech.edu.cn

---

**CREATE TABLE**

*table\_name*

```
(  
    ,  
    ,  
    ...  
)
```

This is the syntax (simplified, some products can take a lot of additional options) for creating a table. The weird capitalization here is only to explain that SQL keywords (words that have a special meaning in SQL) are NOT case sensitive and can be typed in any case you want. I mostly use lowercase but some people have different habits.

---

# tablename



# TABLENAME



# tABLENAMe

Same story with identifiers, the names you give to tables or, as you will soon see, columns, aren't case-sensitive.

Some classic rules apply though: table (and column) names must start with a letter (PostgreSQL tolerates an underscore) and only contain letters, digits, or underscores. The \$ sign is also accepted, and some products allow #.



Because names are not case-sensitive, CamelCaps aren't often used and underscores are preferred to separate words. Note that names can sometimes be quoted between double quotes or square brackets, in which case spaces are allowed AND names become case-sensitive. Better to avoid it.

```
create table table_name  
  (column_name, datatype  
   ,  
   ...  
)
```

A comma-separated list of a column-name followed by spaces and a datatype specifies the columns in the table.

---

**Text**

That's basically what you find in a database; nothing fancy. Some products allow user defined types, but they aren't much used. Dates are quite important in databases.

**Number**

**Date**

**Binary**

plus some specific  
types or variants

# Text datatypes

char(*length*)

char    char(1)

varchar(*max length*)

varchar2(*max length*)

ORACLE®

clob

Datatype names vary with the DBMS. char() is for fixed-size columns (data is padded with spaces if shorter). Used for codes. Oracle understands varchar() and transforms it into its own varchar2(), but it's slightly different (an empty varchar2() is the same as nothing, not an empty varchar()). varchars don't pad. They are limited in length (a few thousand bytes). CLOB (called TEXT in MySQL) allows to store much bigger text (Gb).

# Number datatypes

**int**

**float**

8

2

**numeric (*precision*) scale)**

999999999

Oracle knows mostly one number datatype, NUMBER, which can optionally take a precision (number of digits) and a scale (number of these digits after the decimal point). Something equivalent is called NUMERIC (sometimes DECIMAL) with other products, but other products also use INT and FLOAT far more.

# Date datatypes

**date**

includes time, down to second with  
Oracle, not with other products

**datetime**

down to second (other than Oracle, DB2  
and PostgreSQL)

**timestamp**

down to 0.000001 second, Replace datetime  
in DB2/PostgreSQL

Same kind of mess with date and time datatypes. You also find a datetime2 type with SQL Server. Some subtle differences in ranges of acceptable values, precision, etc. Some products also implement a distinct TIME datatype, or datatypes that represent time intervals.

# Binary datatypes

**raw(*max length*)**

**varbinary(*max length*)**

**blob,bytea**

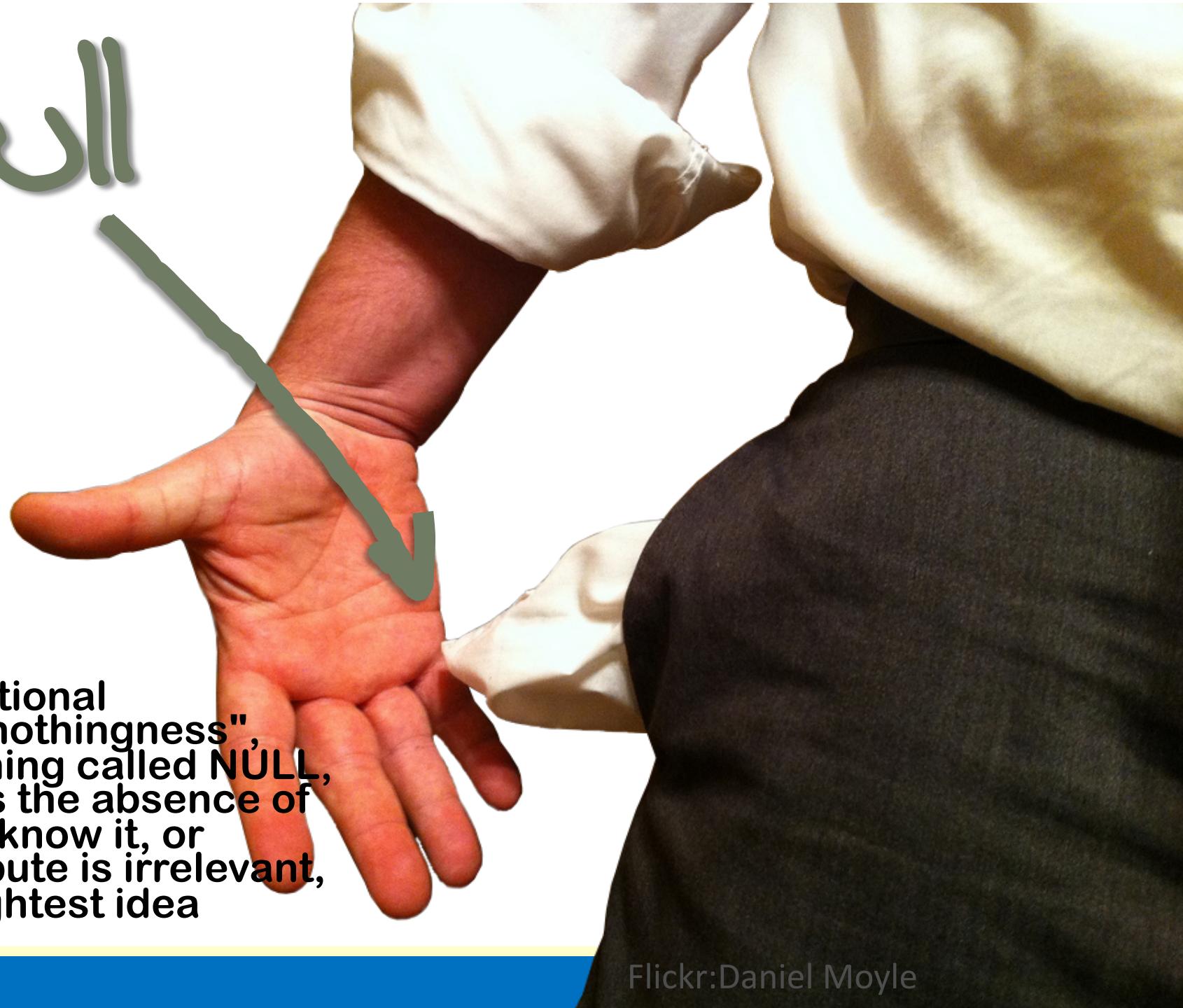
RAW in Oracle, and VARBINARY (SQL Server) are the binary equivalent of VARCHAR. BLOB is the binary equivalent of CLOB (BLOB means Binary Large Object). PostgreSQL calls the binary datatype BYTEA, don't ask me why.

```
create table people (          peopleid int,  
                           first_name varchar(30),  
                           surname  varchar(30),  
                           born      numeric(4),  
                           Died      numeric(4)  
)
```

This CREATE TABLE statement would be accepted as perfectly valid by most DBMS product (Oracle would automatically convert INT into NUMBER(38), VARCHAR into VARCHAR2 and NUMERIC into NUMBER). However, this is a VERY BAD CREATE TABLE statement because it does nothing to enforce that we have a valid "relation" in Codd's sense.

A first question to ask ourselves is what should be mandatory? Do we really want rows about people we don't even know the name of? Obviously not.

# Null



One important concept in relational databases is the concept of "nothingness", represented in SQL by something called **NULL**, which isn't a value. It indicates the absence of a value, because we don't yet know it, or because in that case the attribute is irrelevant, or because we haven't the slightest idea about what this should be.

```
create table people (peopleid int  
    first_name varchar(30),  
    surname varchar(30)  
    , not null,  
    born numeric(4),  
    died numeric(4))
```

We indicate that a column is mandatory by saying that NULL isn't acceptable for this column, which is indicated by NOT NULL after the data type. The more NOT NULL columns, the better, because saying "I don't know" everywhere isn't very interesting. Surname and people identifier are columns that MUST have a value. Unless we only want dead people in our database, we should allow column DIED to take unknown values (we all know it will take a value one day, but we don't know it now). What about first\_name?



```
filmdb=# select * from people where surname like '%Gaga%';
peopleid | first_name | surname | born | died | gender
-----+-----+-----+-----+-----+-----+
      8105 |           | Lady Gaga | 1986 |       | F
(1 row)
```



Stored in the database

comment on column people.surname is 'Surname or stage name';

Not usually stored in the database

-- comments in an SQL statement start with a double dash

In that case the surname will be either the surname or a stage name. We can document it inside the database with the COMMENT statement available with SOME DBMS products, or we may document it in the CREATE STATEMENT itself (usually saved to a file) with a comment that extends from a double dash to the end of the line.

```
create table people (peopleid int not null,  
first_name varchar(30),  
surname varchar(30) not null,  
born numeric(4),  
died numeric(4))
```



For column BORN, we can either accept that a row is created for a person before we have the information when that person was born, or we may consider that people who enter data should do their homework and find the information before they create a row for a person. This is the option taken in the demo database.

```
create table people (peopleid int not null,  
                    first_name varchar(30),  
                    surname varchar(30) not null,  
                    born numeric(4) , not null,  
                    died numeric(4))
```

# 2.3 Constraints

---

Shiqi YU 于仕琪

yusq@sustech.edu.cn

```
create table people (peopleid int not null,  
                    first_name varchar(30),  
                    surname varchar(30) not null,  
                    born numeric(4)  
                    ,  
                    died numeric(4))
```

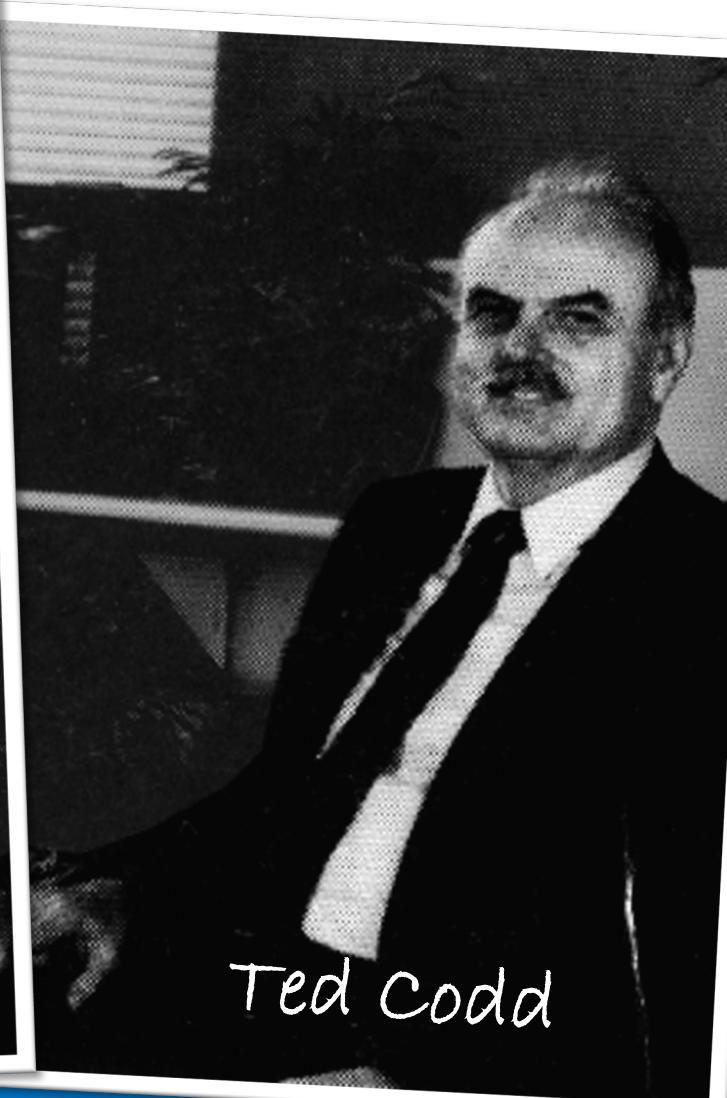
Our CREATE TABLE statement is better, but still doesn't prevent us from entering two strictly identical rows.

Surname	Firstname	Birthdate	Picture
Hepburn	Audrey	4-May-1929	
Hepburn	Audrey	4-May-1929	

Ted Codd was soon joined by another Briton, Chris Date, about 20 years his junior, who became his close friend, evangelist, and also worked on improving the relational theory.



Chris Date



Ted Codd

---

Chris Date's work was mostly about ensuring first that only correct data that fits the theory can enter the database, and that data inside the database remains correct. Whenever programs retrieve data, they shouldn't have to double-check it and should be able to rely on the database management system. This is ensured through

# CONSTRAINTS

Constraints are declarative rules that the DBMS will check every time new data will be added, when data is changed, or even when data is deleted, in order to prevent any inconsistency. Any operation that violates a constraint fails and returns an error.

```
create table people (peopleid int not null,  
                    first_name varchar(30) key,  
                    surname varchar(30) not null,  
                    born numeric(4) not null,  
                    died numeric(4))
```

NOT NULL is a constraint. But there are many others. For instance, PRIMARY KEY tells which is the main key for the table, and indicates two things:

- 1) that the value is mandatory (the additional NOT NULL doesn't hurt but is redundant), and
- 2) that the values are unique (no duplicates allowed in the column)

```
create table people (peopleid int not null  
                     primary key,  
                     first_name varchar(30),  
                     surname  varchar(30) not null,  
                     born    numeric(4) not null,  
                     died    numeric(4) ,  
                     )  
                     unique (first_name, surname))
```

So far, nothing would prevent us from entering two Audrey Hepburns with different ids. To ensure we only have one, we must say that the combination (first\_name, surname) is unique (for actors ...). Constraints on several columns at once (same story with primary keys) are specified at the end of the list of columns with a comma-delimited list of column names.

---

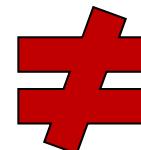
For Oracle, PostgreSQL and DB2 ...

Beware that with many products data IS case sensitive, and different capitalization mean different values that wouldn't violate a uniqueness constraint. You MUST standardize case.

'audrey'



'AUDREY'



'Audrey'

... not for SQL Server, MySQL or SQLite ...

```
create table people (peopleid int not null  
                     primary key,  
                     first_name varchar(30),  
                     surname varchar(30) not null,  
                     born numeric(4) not null,  
                     died numeric(4),  
                     unique (first_name, surname))
```

```
create table people (peopleid int not null
                     primary key,
                     first_name varchar(30),
                     check (first_name = upper(first_name)),
                     surname varchar(30) not null,
                     check (surname = upper(surname)),
                     born numeric(4) not null,
                     died numeric(4),
                     unique (first_name, surname))
```

One way to guarantee that case is, for instance, uppercase, would be to use the CHECK constraint, which isn't much used and that's a pity. CHECK is also useful for checking discrete values (e.g. only Y and N are accepted), ranges (percentage is a number between 0 and 100) or date validity (not born after being dead). MySQL accepts CHECK but doesn't enforce it.

Chris Date



Picture: Douglas Robertson



But there is far better than CHECK,  
which is a fairly static control:  
Referential Integrity

```
create table movies (movieid      int not null primary key,  
                    title        varchar(60) not null,  
                    country      char(2) not null,  
                    year_released numeric(4) not null  
                               check(year_released >= 1895),  
                    unique (title, country, year_released))
```

In the MOVIES table, COUNTRY is a column that can take a LOT of different values, and listing all of them in a CHECK constraint would be clumsy. Besides, countries disappear (USSR, Yugoslavia) and new countries appear (South Sudan, Croatia, Slovakia). If we have no control, any typo would allow non-existing country codes to slip into the database.

# COUNTRIES

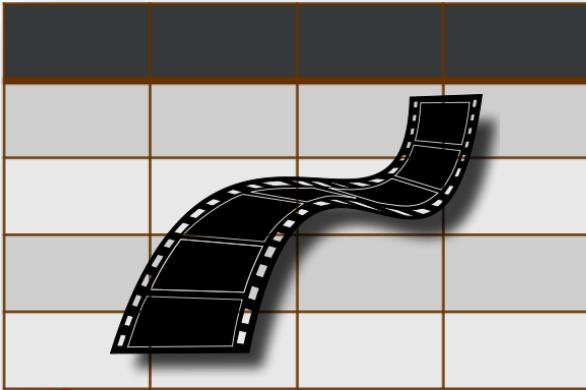
country_code	country_name	continent
us	United States	AMERICA
cn	China	ASIA
in	India	ASIA
br	Brazil	AMERICA
uk	United Kingdom	EUROPE
ru	Russia	EUROPE
mx	Mexico	AMERICA
it	Italy	EUROPE
de	Germany	EUROPE

The solution is referential integrity, and what is known as a reference table: a country that stores all country codes, and corresponding country names (all codes don't immediately ring a bell), with the code as primary key (and the country name declared as unique). We'll only accept a country code if we find it in this table (which can be modified, with new codes added)

```
create table movies (movieid      int not null primary key,  
                    title       varchar(60) not null,  
                    country     char(2) not null,  
                    year_released numeric(4) not null  
                                check(year_released >= 1895),  
                    unique (title, country, year_released) )  
                    foreign key(country)  
                    references countries(country_code)
```

We'll declare in MOVIES that column COUNTRY is a FOREIGN KEY, which means that we must be able to find it as a key of the table that is referenced. Only primary keys and columns declared as UNIQUE can be referenced. A foreign key can be composed of a combination of columns (rare). Note that the constraint works both ways: we won't be able to delete a country if movies reference it, because we would get "orphaned rows" and the database would become inconsistent.

## movies



## people



## credits

Two gold key icons are placed on the second and third columns of the credits table.

*foreign key*

*foreign key*

Primary and foreign keys are fairly independent notions. In table CREDITS, the movie id and the person id are foreign keys, because we cannot give credits for a non-existing film or person. Both foreign keys are part of the primary key of credits, which is composed of all three columns for this table.

# Reminder

Creating tables requires:

- Proper modelling
- Defining keys
- Determining correct data types
- Defining constraints



When all these are done, you can be sure that data that enters the database will be correct, and will remain so.

The preparatory work may seem (and sometimes is) a bit boring, but the rewards are huge: when programs no longer have to thoroughly check data but only check return codes, they become leaner and are far easier (and cheaper) to maintain. It also ensures, when several applications access the database, that controls are centralized and that one sloppily written small application won't corrupt data for other well written programs.

# 2.4 Insert

---

Shiqi YU 于仕琪

yusq@sustech.edu.cn

# INSERT

SOME\_TABLE

column1

column2

INSERT enters data into a table; if you consider tables as a special kind of variables, INSERT is very much like an assignment (often more like '+=' in C and derived languages). It changes the contents of the tables.

---

```
insert into table_name  
  (list of columns)  
values (list of values)
```

"lists" are comma-separated lists.

Values must match column-names one by one. What happens if you omit a column name from the list? Nothing is entered into it. If the column is mandatory, the INSERT statement fails and nothing at all is done.

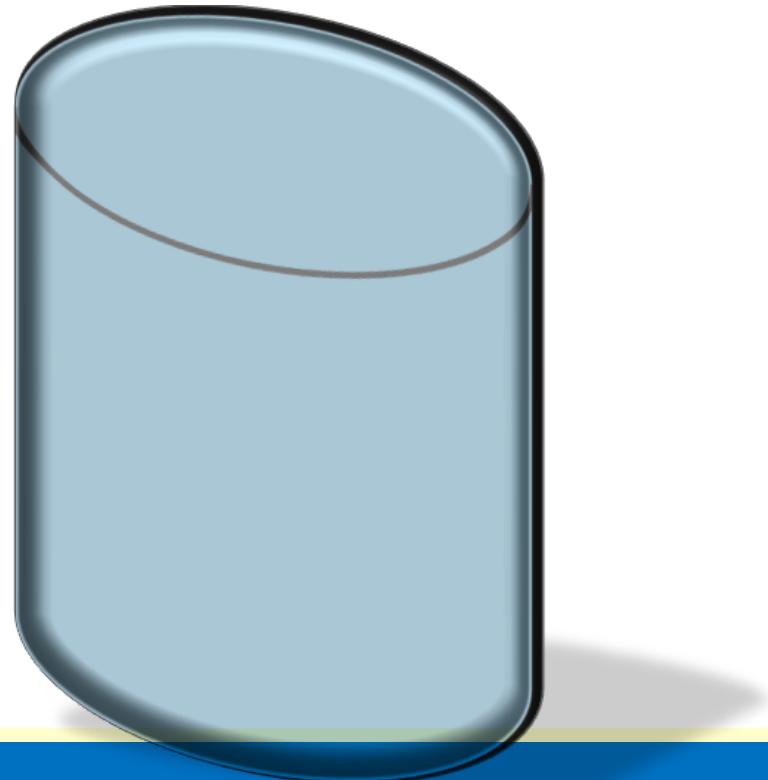
```
insert into countries(country_code, country_name,  
continent)  
values('us', 'United States', 'AMERICA')
```

Note that strings are given between SINGLE quotes. Some products (MySQL, SQLite) also accept double quotes but SINGLE quotes is the standard and what works everywhere. You should use them.

# Reminder



HELLO WORLD



I've stressed that case should be tightly controlled when inserting data. It's possible to use functions in INSERT statements to this effect.



# 1960

What happens when  
the data contains a  
quote?  
You must escape it.

```
insert into movies(movieid,  
title,  
country,  
year_released)  
values (123,  
'L"Avventura',  
'it',  
1960)
```

The standard SQL way to escape a quote is to double it. Only one is stored. MySQL also accepts a backslash as an escape character.



# Entering a date?

Entering a date in a table is a common task that demands some care, there are only two ways to enter a date, as a string or as the result of a function or computation.

---

Most products support a default date format, and are able to automatically translate text that is inserted into a date or datetime column if this text matches the default format. You shouldn't rely on this and always specify the format, as here with Oracle:

```
to_date( '07/20/1969' ,  
         'MM/DD/YYYY')
```

The reason is that firstly a database administrator CAN change the default format, and secondly that dates can be ambiguous: an American may read **11/05** as **November 5th**, for a European it will be **May 11th**.

---

# SYSDATE

ORACLE®



# CURDATE()

# GETDATE()



The current date is often used. CURRENT\_DATE (no time) and CURRENT\_TIMESTAMP (time included) are recognized by all products. For historical reasons, all products also have their, still frequently used, own functions.