Principles of Database Systems (CS307)
Lecture 2: Introduction to SQL

Yuxin Ma

Department of Computer Science and Engineering
Southern University of Science and Technology

- Most contents are from slides made by Stéphane Faroult and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.

How Do We Manage Data in a Database?

- Usually, a special language is needed
  - Be able to use a language to query a database
    - Either interactively or from within a program

- Query language
  - Query data
  - Modify data

Some History

- ALPHA
  - Codd's querying language

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

$$(r_1[2], r_1[3]): P_1r_1 \land \exists P_2r_2(r_2[2] = 15 \land r_2[1] = r_1[1]).$$

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

$$(r_1[3], r_2[2]): P_1r_1 \land P_2r_2 \land (r_1[1] = r_2[1]).$$

Codd, E.F., "Data Base Sublanguage Founded on the Relational Calculus", Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access, and Control, San Diego.

Some History

- ALPHA
  - Codd's querying language

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

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

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

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

Codd, E.F., "Data Base Sublanguage Founded on the Relational Calculus", Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access, and Control, San Diego.
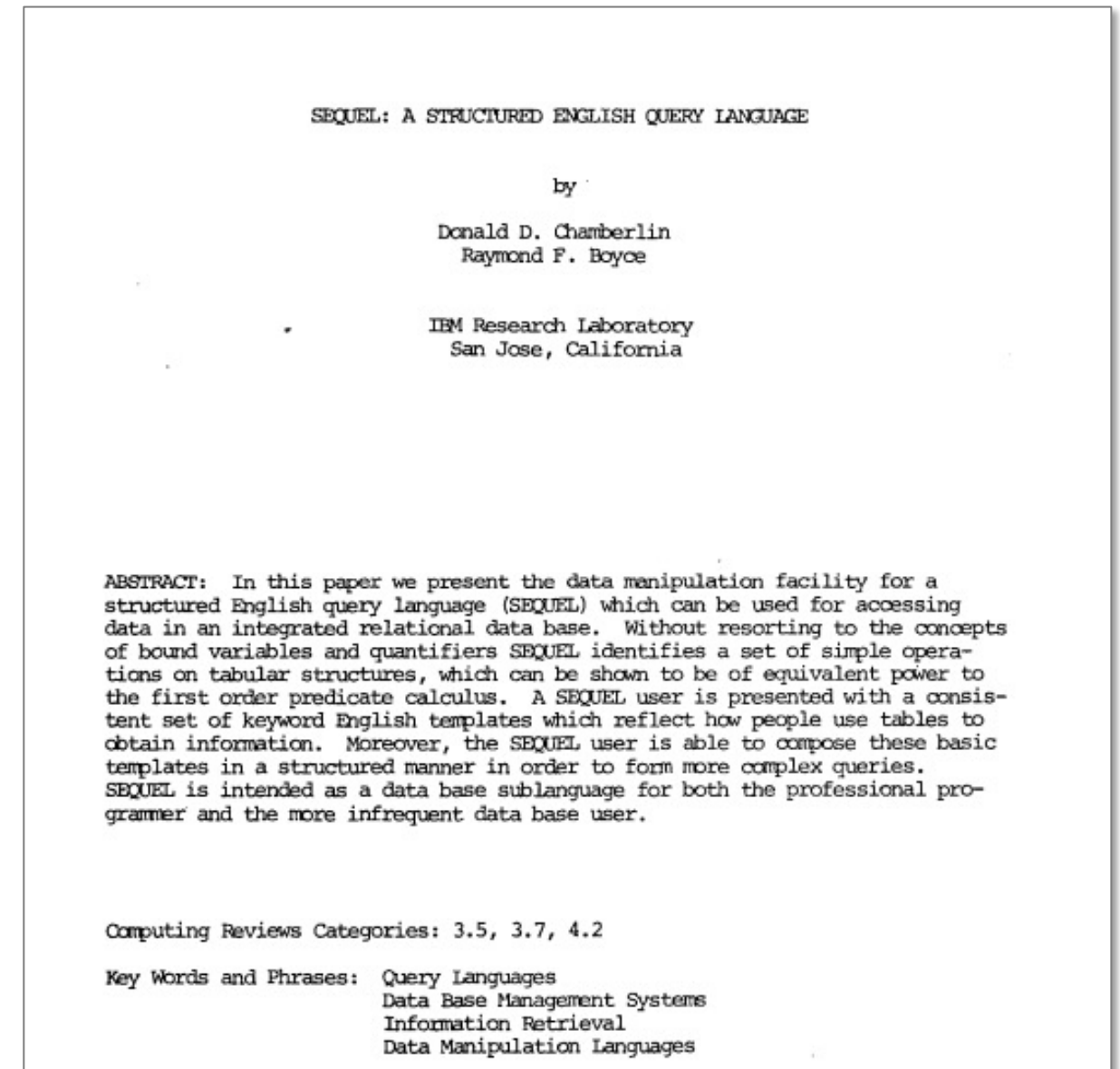
However, it didn't excite enthusiasm at IBM

Some History

- "SEQUEL: A Structured English Query Language"
  - IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
  - An "easy" language, with an English-like syntax

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

SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

by

Donald D. Chamberlin
Raymond F. Boyce

IBM Research Laboratory
San Jose, California

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 keyword English templates which reflect how people use tables to obtain information. Moreover, the SEQUEL user is able to compose these basic templates in a structured manner in order to form more complex queries. SEQUEL is intended as a data base sublanguage for both the professional programmer and the more infrequent data base user.

Computing Reviews Categories: 3.5, 3.7, 4.2

Key Words and Phrases: Query Languages
Data Base Management Systems
Information Retrieval
Data Manipulation Languages

Some History

- SEQUEL was then renamed as Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999 (language name became Y2K compliant!)   A huge problem that appears everywhere and many times
  - SQL:2003
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features
  - Not all examples here may work on your particular system.

Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First formalized by ANSI
1989	SQL-89	FIPS 127-1	Minor revision that added integrity constraints, adopted as FIPS 127-1
1992	SQL-92	SQL2, FIPS 127-2	Major revision (ISO 9075), *Entry Level* SQL-92 adopted as FIPS 127-2
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries (e.g. transitive closure), triggers, support for procedural and control-of-flow statements, nonscalar types (arrays), and some object-oriented features (e.g. structured types), support for embedding SQL in Java (SQL/OLB) and vice versa (SQL/JRT)
2003	SQL:2003		Introduced XML-related features (SQL/XML), window functions, standardized sequences, and columns with autogenerated values (including identity columns)
2006	SQL:2006		ISO/IEC 9075-14:2006 defines ways that SQL can be used with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database, and publishing both XML and conventional SQL-data in XML form. In addition, it lets applications integrate queries into their SQL code with XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.[33]
2008	SQL:2008		Legalizes ORDER BY outside cursor definitions. Adds INSTEAD OF triggers, TRUNCATE statement,[34] FETCH clause
2011	SQL:2011		Adds temporal data (PERIOD FOR)[35] (more information at: Temporal database#History). Enhancements for window functions and FETCH clause.[36]
2016	SQL:2016		Adds row pattern matching, polymorphic table functions, JSON
2019	SQL:2019		Adds Part 15, multidimensional arrays (MDarray type and operators)

https://en.wikipedia.org/wiki/SQL

Standardization of SQL

Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First formalized by ANSI
1989	SQL-89	FIPS 127-1	Minor revision that added integrity constraints, adopted as FIPS 127-1
1992	SQL-92	SQL2, FIPS 127-2	Major revision (ISO 9075), *Entry Level* SQL-92 adopted as FIPS 127-2
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries (e.g. transitive closure), triggers, support for procedural and control-of-flow statements, nonscalar types (arrays), and some object-oriented features (e.g. structured types), support for embedding SQL in Java (SQL/OLB) and vice versa (SQL/JRT)
2003	SQL:2003		Introduced XML-related features (SQL/XML), window functions, standardized sequences, and columns with autogenerated values (including identity columns)
2006	SQL:2006		ISO/IEC 9075-14:2006 defines ways that SQL can be used with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database, and publishing both XML and conventional SQL-data in XML form. In addition, it lets applications integrate queries into their SQL code with XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.[33]
2008	SQL:2008		Legalizes ORDER BY outside cursor definitions. Adds INSTEAD OF triggers, TRUNCATE statement,[34] FETCH clause
2011	SQL:2011		Adds temporal data (PERIOD FOR)[35] (more information at: Temporal database#History). Enhancements for window functions and FETCH clause.[36]
2016	SQL:2016		Adds row pattern matching, polymorphic table functions, JSON
2019	SQL:2019		Adds Part 15, multidimensional arrays (MDarray type and operators)

https://en.wikipedia.org/wiki/SQL

Standardization of SQL (any other examples of standardization?)

Basic Syntax of SQL

```
select * from lab where time = '3-34';
```

select …
- followed by the names of the columns you want to return

from …
- followed by the name of the tables that you want to query

where …
- followed by filtering conditions

Competing Languages of SQL

- QUEL, born at Berkeley, and associated with INGRES, was highly regarded
  - Ingres Database

QUEL ------ Ingres ⟶ SEQUEL (SQL) ------ Postgres

QUEL	SQL
| ```
create student(name = c10, age = i4, sex = c1, state = c2)

range of s is student
append to s (name = "philip", age = 17, sex = "m", state = "FL")

retrieve (s.all) where s.state = "FL"

replace s (age=s.age+1)

retrieve (s.all)

delete s where s.name="philip"
``` | ```
create table student(name char(10), age int, sex char(1), state char(2));

insert into student (name, age, sex, state) values ('philip', 17, 'm', 'FL');

select * from student where state = 'FL';

update student set age=age+1;

select * from student;

delete from student where name='philip';
``` |

Michael Stonebraker (1943-)
Turing Award 2014

Competing Languages of SQL

- QBE (Query by Example)
  - A visual querying tool (visual but with characters)
  - Created by IBM as well


```
id:
address:
time: 3-34
capacity:
teacher:
```

```
select * from lab where time = '3-34';
```

Moshé M. Zloof

Two Main Components for a Query Language

- From Codd's seminal paper:
  - A good database language should allow to deal as easily with contents (data) as containers (tables)
  - … Something that SQL does reasonably well

Two Main Components for a Query Language

- Data Definition Language (DDL)
  - The SQL data-definition language (DDL) allows the specification of information about relations, including:
    - The schema for each relation.
    - The type of values associated with each attribute.
    - The Integrity constraints
    - The set of indices to be maintained for each relation.
    - Security and authorization information for each relation.
    - The physical storage structure of each relation on disk.

Two Main Components for a Query Language

- Data Definition Language (DDL)
  - The SQL data-definition language (DDL) allows the specification of information about relations, including:
    - The schema for each relation
    - The type of values associated with each attribute
    - The Integrity constraints
    - The set of indices to be maintained for each relation
    - Security and authorization information for each relation
    - The physical storage structure of each relation on disk

```
create
alter
drop
```

Two Main Components for a Query Language

- Data Manipulation Language (DML)
  - Provides the ability to:
    - Query information from the database
    - Insert tuples into, delete tuples from, and modify tuples in the database.

```
select
insert
delete
update
```

Something about SQL

- Simple?
  - As you can see, it seems to be simple
  - But it becomes difficult when you combine operations
    - We will talk about it later

- Standard?
  - We have mentioned standardization of SQL before
  - However, no product fully implements it
    - Different product implements SQL differently
    - … and introduces dialects

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.

"Problems" in SQL

- SQL ≠ Relational Database
  - SQL wasn't designed as a "relationally correct" language
    - In some respects, it is very lax
    - But easy to use, however
    - And, easy to misuse
      - So, using it well is difficult
- Sometimes, you will get results even if the SQL is wrong
  - SQL is not as strict as C or Java; it will not give you error messages if your table cannot fit the requirement of the theory
    - "Wrong results" without warnings

"Problems" in SQL

- SQL ≠ Relational Database
  - SQL wasn't designed as a "relationally correct" language
    - In some respects, it is very lax
    - But easy to use, however
    - And, easy to misuse
      - So, using it well is difficult
- Sometimes, you will get results even if the SQL is wrong
  - SQL is not as strict as C or Java; it will not give you error messages if your table cannot fit the requirement of the theory
    - "Wrong results" without warnings

Be careful when designing your SQL queries

"Problems" in SQL

- Key propriety of relations (in Codd's original paper)

ALL ROWS ARE DISTINCT

- This can be enforced for tables in SQL
    - (But you have to create your tables well)

- But it is not enforced for query results in SQL
    - You must be extra-careful if the result of a query is the starting point for another query, which happens often. (i.e., combined queries)

Create Tables

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

Syntax

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
    ....
);
```

Example

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

Create Tables

- Table names
  - Case-insensitive (Usually, by default)
    - But, in some database systems, the case sensitivity is quite different
    - Try, or find the reference for the specific database system

For example, MariaDB is system-dependent with respect to case sensitivity

`https://mariadb.com/kb/en/identifier-case-sensitivity/`

```
CREATE TABLE tablename
CREATE TABLE TABLENAME
CREATE TABLE tABleNamE

/* Same table names */
```

Create Tables

- Table names
  - Case-insensitive (Usually, by default)
    - But, in some database systems, the case sensitivity is quite different
    - Try, or find the reference for the specific database system

For example, MariaDB is system-dependent with respect to case sensitivity

`https://mariadb.com/kb/en/identifier-case-sensitivity/`

```
CREATE TABLE tablename
CREATE TABLE TABLENAME
CREATE TABLE tABleNamE

/* Same table names */
```

Actually, keywords are case-insensitive as well

```
create table tablename
CREATE table TABLENAME
CReaTE tABle tABleNamE

/* Same keywords */
```

Create Tables

- Table names
  - Case-insensitive (Usually, by default)
    - But, in some database systems, the case sensitivity is quite different
    - Try, or find the reference for the specific database system
- Naming Convention
  - Underscores as word separators (instead of CamelCase in Java)

```
CREATE TABLE tablename
CREATE TABLE TABLENAME
CREATE TABLE tABleNamE

/* Same table names */
```

Actually, keywords are case-insensitive as well

```
create table tablename
CREATE table TABLENAME
CReaTE tABle tABleNamE

/* Same keywords */
```

Create Tables

- Table names
  - Case-insensitive (Usually, by default)
    - But, in some database systems, the case sensitivity is quite different
    - Try, or find the reference for the specific database system
- Naming Convention
  - Underscores as word separators (instead of CamelCase in Java)
- Be careful with double quotes
  - ... which represents a "case-sensitive" name

```
CREATE TABLE tablename
CREATE TABLE TABLENAME
CREATE TABLE tABleNamE

/* Same table names */
```

Actually, keywords are case-insensitive as well

```
create table tablename
CREATE table TABLENAME
CReaTE tABle tABleNamE

/* Same keywords */
```

Create Tables

- An SQL relation is defined using the create table command:

```
create table r (
    A1 D1, A2 D2, ..., An Dn,
    (integrity-constraint1),
    ...,
    (integrity-constraintk)
)
```

- `r` is the name of the relation
- each `Ai` is an attribute name in the schema of relation `r`
- `Di` is the data type of values in the domain of attribute `Ai`

Data Types

- Text data types
  - `char`(length) -- fixed-length strings
  - `varchar`(max length) -- non-fixed-length text
  - `varchar2`(max length) ORACLE -- Oracle's transformation of varchar
  - `clob` -- very long text (like GB-level text)
    - Or, `text` MySQL

Data Types

- Numerical types
  - `int` -- Integer (a finite subset of the integers that is machine-dependent)
  - `float(n)` -- Floating point number, with user-specified precision of at least *n* digits
  - `real` -- Floating point and double-precision floating point numbers, with machine-dependent precision
  - `numeric(p, d)`
    - Fixed point number, with user-specified precision of p digits, with d digits to the right of decimal point
    - E.g., `numeric(3,1)`, allows 44.5 to be stores exactly, but not 444.5 or 0.32
    - In SQL Server, it is also called `decimal`

Data Types

- Date types
  - `date` -- YYYY-MM-DD
  - `datetime` -- YYYY-MM-DD HH:mm:SS
  - `timestamp` -- YYYY-MM-DD HH:mm:SS
    - But it is in the UNIX timestamp
    - Value range: 1970-01-01 00:00:01 UTC - 2038-01-19 03:14:07 UTC
    - More reading about the "Year 2038 Problem" of the `timestamp` data type: `https://en.wikipedia.org/wiki/Year_2038_problem`

Data Types

- Binary data types
  - `raw`(max length)
  - `varbinary`(max length)
  - `blob` -- binary large object
  - `bytea` -- used in PostgreSQL

Constraints

- Can you find any problem in this statement?

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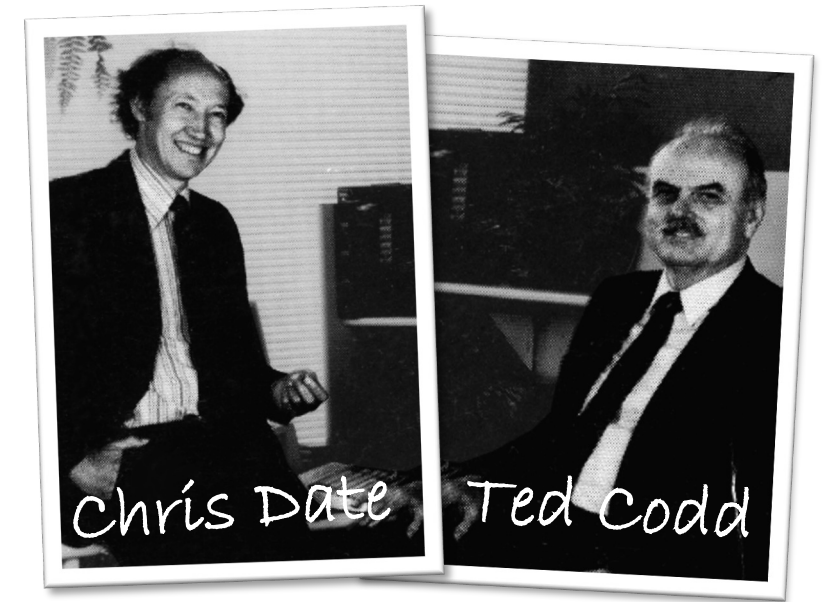
Constraints

- Can you find any problem in this statement?
  - It is valid and can be accepted by most DBMS
  - But it <u>does nothing</u> to enforce that we have a valid "relation" in Codd's sense

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

Constraints

Chris Date   Ted Codd

- Ted Codd and Chris Date
  - Worked on improving the relational theory
- Chris Date's work
  - Ensuring that only correct data that fits the theory can enter the database
  - Data inside the database remains correct
    - No need to double check for application programs

Constraints are **declarative rules** that the DBMS **will check every time** new data will be <u>added</u>, when data is <u>changed</u>, or even when data is <u>deleted</u>, in order to **prevent any inconsistency**.

* Any operation that violates a constraint fails and returns an error.

Constraints: Not NULL

- NULL values

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

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

- We don't want someone with no ID and name

- Use `not null` to indicate that these columns are mandatory

Constraints: Not NULL

- NULL values

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

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

- We don't want someone with no ID and name

- Use `not null` to indicate that these columns are mandatory

> We can still have rows that with `NULL` values in the columns of born, died, and `first_name`

Constraints: Not NULL

- NULL values

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

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

Why is only surname mandatory?

- It depends on the requirement. In this movie database case, some actors may be known as their stage names instead of real names. (Lady Gaga vs. Stefani Joanne Angelina Germanotta)

Takeaway: design your table according to the requirements

Constraints: Not NULL

- NULL values

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

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

Similar to the column born

- We can either accept that
  - A row is created before we have information of that person's birth date
  - Or we require that the information should be found before entering the data

Comments

```
/* Multi-line
comments */

-- Single line comments, similar to double back-slashes in Java and C++

// *Some DBMS also support double back-slashes, like SQL Server
```

Comments

```
create table people (
    peopleid int not null,
    first_name varchar(30),
    surname varchar(30) not null,  -- the actual surname or the stage name
    born numeric(4) not null,  -- the birth date is mandatorybefore entering the data
    died numeric(4)
)
```

- Add comments to the definition of tables

Constraints: Primary Key

- The main key for the table, and indicates two things:
  - the value is mandatory
  - 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)
)
```

Constraints: Primary Key

- The main key for the table, and indicates two things:
  - the value is mandatory
  - 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)
)
```

`primary key` implies `not null`, so `not null` here is redundant (but doesn't hurt)

Constraints: Unique

- So far, nothing would prevent us from entering two same rows with different IDs

peopleid	first_name	surname	born	died
1	Alfred	Hitchcock	1899	1980
2	Alfred	Hitchcock	1899	1980
3	....	...	...	...

```
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)
)
```

Constraints: Unique

- A unique constraint (on a combination of multiple columns)

peopleid	first_name	surname	born	died
1	Alfred	Hitchcock	1899	1980
2	Alfred	Hitchcock	1899	1980
3	....	...	...	...

```
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)
)
```

The combination of (`first_name, surname`) cannot be the same for any two rows
- But you still can have people with the same first name or surname, respectively

Constraints: Unique

- A unique constraint (on a single column)

peopleid	first_name	surname	born	died
1	Alfred	Hitchcock	1899	1980
2	Alfred	Hitchcock	1899	1980
3	....	...	...	...

No identical first names for any two people here
- But it is not what we want in this table

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

Constraints: Check

- A column must satisfy a certain boolean expression test
  - The most generic constriant type

You must ensure that the person died after born

A useful trick to standardize names
- Such that there won't be rows with the same name of "Alfred Hitchcock", "ALFRED HITCHCOCK", and "alfred hitchcock".
- `upper(string)` is a function in PostgreSQL

```
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),
    check (died - born >= 0),
    check (first_name = upper(first_name)),
    check (surname = upper(surname))
)
```

Named Constraints

- A name can be assigned to the constraints
  - … in order to refer to them easier in some other operations
    - PostgreSQL will give a name to the constraints if you don't assign a name explicitly

```
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),
    check (died - born >= 0),
    check (first_name = upper(first_name)),
    check (surname = upper(surname))
)
```

```
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),
    constraint validate_birthdate check (died - born >= 0),
    constraint standardize_first_name check (first_name = upper(first_name)),
    constraint standardize_surname check (surname = upper(surname))
)
```

Referential Integrity

- Check constaints are static
  - Once it is written into the table, the criteria cannot be updated automatically

```
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)
)
```

- It is very difficult to perform static checks
  - Too many countries; country names and codes may change

Referential Integrity

- Check constaints are static
  - Once it is written into the table, the criteria cannot be updated automatically

```
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)
)
```

country_code	country_name	continent
US	United States	AMERICA
CN	China	ASIA
RU	Russia	EUROPE

Referential Integrity

- The country column in `movies` should be linked with the `country_code` column in another table (called reference table)

Foreign Key

- Format:
  - `foreign key (Am, ..., An ) references r`

```
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 country_list (country_code)
)
```

Meaning of this foreign key:
- The country column in this table (movies) refers to the country_code column in the table called country_list

Tip:
- country_code should be a key (primary key or unique) in the table country_list

Foreign Key

- Format:
  - `foreign key (Am, ..., An ) references r`

Movie Entities

Movie ID	Movie Title	Country	Year
0	Citizen Kane	US	1941
1	La règle du jeu	FR	1939
2	North By Northwest	US	1959
3	Singin' in the Rain	US	1952
4	Rear Window	US	1954

Directed By

Movie ID	Director ID
0	2
1	5
2	1

Foreign keys required in this table

Director Entities

Director ID	Director_Firstname	Director_Lastname	Born	Died
1	Alfred	Hitchcock	1899	1980
2	Orson	Welles	1915	1985
3	....	...	...	...

Foreign Key

- However, in some cases, foreign key can be a problem
  - Especially in big data processing applications
    - E.g., Alibaba Java Coding Guideline (阿里巴巴Java开发手册)

（三）SQL 语句

6. 【强制】不得使用外键与级联，一切外键概念必须在应用层解决。

说明：以学生和成绩的关系为例，学生表中的 student_id 是主键，那么成绩表中的 student_id 则为外键。如果更新学生表中的 student_id，同时触发成绩表中的 student_id 更新，即为级联更新。外键与级联更新适用于单机低并发，不适合分布式、高并发集群；级联更新是强阻塞，存在数据库更新风暴的风险；外键影响数据库的插入速度。

Summary: How to Create Tables

- Creating tables requires:
  - Proper modelling
  - Defining keys
  - Determining correct data types
  - Defining constraints

- Boring, but important
  - No further checks in the application programs; most things are ensured in the database layer

Updates to Tables

- Insert
  - `insert into instructor values ('10211', 'Smith', 'Biology', 66000);`
- Delete
  - Remove all tuples from the student relation
    - `delete from movies`

Updates to Tables

- Drop Table
  - `drop table r`
- Alter
  - `alter table r add A D`
    - where A is the name of the attribute (column) to be added to relation r and D is the data type of A.
    - All existing tuples in the relation are assigned null as the value for the new attribute.
  - `alter table r drop column A`
    - where A is the name of an attribute of relation r
    - Dropping of attributes not supported by many databases.
    - (There are some other things that can be "dropped", including checks, foreign keys, indexes, etc.)

Updates to Tables

- More about insert

```
create table lab (
    id serial primary key,
    address varchar(20) not null,
    time varchar(20) not null,
    capacity int,
    teacher varchar(20),
    unique (address,time)
);

                    Values must match column names one by one

insert into lab (address, time, capacity, teacher) values ('402','2-78',36,'yueming');

insert into lab (address, time, teacher) values ('402','2-78','yueming');

insert into lab (address, time, teacher) values ('408','2-78','o''relly');
```

Updates to Tables

- More about insert

```
create table lab (
    id serial primary key,
    address varchar(20) not null,
    time varchar(20) not null,
    capacity int,
    teacher varchar(20),
    unique (address,time)
);


insert into lab (address, time, capacity, teacher) values ('402','2-78',36,'yueming');

insert into lab (address, time, teacher) values ('402','2-78','yueming');

insert into lab (address, time, teacher) values ('408','2-78','o''relly');
```

Missing columns and values for "nullable" columns are allowed
- ... and a NULL willl be inserted

* But if you miss a mandatory column (such as address), an error will occur.

Updates to Tables

- More about insert

```sql
create table lab (
    id serial primary key,
    address varchar(20) not null,
    time varchar(20) not null,
    capacity int,
    teacher varchar(20),
    unique (address,time)
);


insert into lab (address, time, capacity, teacher) values ('402','2-78',36,'yueming');

insert into lab (address, time, teacher) values ('402','2-78','yueming');

insert into lab (address, time, teacher) values ('408','2-78','o''relly');
```

* Use two single quotes to represent a single quote in the content (i.e., escape charactor)

Lab Session (Week 2)

- We will have more examples for manipulating a table and inserting data into tables

Please remember to upload your "*Undergraduate Students Declaration Form*" to the correct place

New locations for the lab sessions this afternoon:
- *Group 1 (Tuesday 7-8, Yuxin Ma): Room 507, Lecture Hall 3*
- *Group 2 (Tuesday 7-8, Yueming Zhu): Room 510, Lecture Hall 3*