Lecture 2 & Lab 2

SQL: Struture query language

1. expalnation: SQL是一种声明性语言，用户只需要指定需要的数据，而不需要指定如何获取数据。SQL的语法规则和语句可以用于对关系型数据库进行增删改查等操作，包括创建表、插入数据、更新数据、删除数据、查询数据等。(by chatgpt)
2. basic syntax in SQL

```
select ...; --followed by the names of columns you want to select(列筛选)
from ...; --followed by the name of tables(表的名字)
where ..; --filtering condition(一个单条件或组合条件语句，允许对行进行过滤)
```

DDL: Data Definition Language

1. expalnation: a main component for a query language(是用于定义数据库对象的SQL语言部分)
2. basic syntax:

```
create ...; --创建表、视图、索引等。
alter ...; --修改表结构、视图、索引、约束等
drop ...; --删除表、视图、索引等
truncate ...; --清空表的数据
rename ...; --重命名表，图标等
```

(by chatgpt)

DML：Data Manipulation Language

1. explanation: 操纵表中的数据
2. basic syntax:

```
select ...; --检索数据
insert ...; --插入数据
update ...; --更新数据
delete ...; --删除数据
```

Create Tables:

1. characteristics:
    1. case-insensitive: 关键字，标识符等都是大小写不敏感的
    2. make identifier case-sensitive: double quotes, not recommended

```
-- two tables are created in the following
CREATE TABLE "myTable";
create table "mytable";
```

3. 命名惯例：使用小写

2. basic syntax:

```
-- if no exist 可选择不写，表示如果这个表不存在就进行创建
create table if no exist {table_name1}(
    {attribute_name1} {data_type1},
    {attribute_name2} {data_type2},
    ...
    (
        {integrity_constraint1},
        {integrity_constraint2},
        ...
    )
)
```

Data Types

1. text data types

```
/*
当使用CHAR存储字符串时，如果字符串的长度小于length，则在字符串后面补充空格，使其达到指定长度。CHAR类型的字段始终会占用指定长度的存储空间
*/
char(length) --fix-length string

/*
使用 VARCHAR 存储字符串时，它只会占用存储实际字符串所需的空间
*/
varchar(max length) --non-fix-length text
varchar2(max length) --non-fix-length text(Oracles's)

clob -- very very long text(GB level)
text -- very very long text(GB level)
```

2. numerical type

```
int -- a finite subset of the intergers, machine-dependent
float(n) -- Floating point number, precision: at least n digits
real -- 单精度浮点数(4 bytes)
double -- 双精度浮点数(8 bytes)
double precision -- 双精度浮点数(8 bytes)
numeric(p, d) -- 数字总长为p, 小数部分长度为d
```

3. date types

```
date -- format: YYYY-MM-DD
datetime -- format: YYYY-MM-DD HH:mm:SS
timestamp -- format: YYYY-MM-DD HH:mm:SS, in UNIX system, have 2038 problem
```

4. binary data type

```
raw(max length) -- 定长二进制数据，超过会报错，但是一定占用{max length}个字节的空间
varbinary(max length) -- 变长二进制数据，超过也会报错，但是所需存储空间会变
blob -- 存储较大的二进制数据，变长
bytea -- postgresSql中使用，可存储任意类型的的二进制数据，变长，
```

Constraints

1. DBMS(Database Management System) will check the constraints or declarative rules every time when data is added, changed, deleted.
2. NOT-NULL constraints

```
/*
    we don't want some colomns with no element in some cells
*/
create table notnull_example(
    peopleid int not null -- 这一列一定不能为空
)
```

better:

```
create table notnull_example(
    peopleid int constraint nn not null -- 这一列一定不能为空
)
```

由于not null是两个关键字，所以不能有如下操作：

```
create table notnull_example(
    peopleid int,
    ...
    constraint nn not null(peopleid)
)
```

3. Primary-Key constraints
    1. the value is mandatory(该字段必须)
    2. the value is unique
    3. Only 1 column can be set as primary key
    4. format:

```
create table prime_example(
    peopleid int primary key
)
```

可以使用复合列作为主键

```
create table prime_example(
    peopleid int,
    name_ varchar(40),
    ...
    primary key(peopleid, name_)
)
```

better: declare a constraint name explicitly

```
create table prime_example(
    peopleid int constraint id_pm primary key
)
```

an equivalent format

```
create table prime_example(
    peopleid int,
    ...
    constraint id_pm primary key(peopleid)
)
```

4. Unique
    1. the **value** of a column or a **combination** of several column cannot be the same for 2 rows
    2. format:

```
create table unique_example(
    peopleid int unique,
    ...
    unique (first_name, second_name)
)
```

better: declare a constraint name explicitly

```
create table unique_example(
    peopleid int constraint id_uni unique,
    ...
    constraint name_uni unique (first_name, second_name)
)
```

    3. 区分Unique和primary key

    primary key只有一个且要求not-null

    unique可以有多个，不要求not-null

5. Check
    1. check(condition), 只有满足条件表达式时，数据才会被插入或更新
    2. 在check约束中，可以使用的条件表达式包括比较操作符、逻辑操作符、函数调用等。（by chatgpt)

3. format

```
create table check_example(
    firstname varchar(100),
    lastname varchar(100),
    age int,
    ...
    check(lastname = upper(lastname)), -- 保证大写
    check(firstname = lower(firstname)), -- 保证小写
    constraint age_check check(age >= 0)
    -- 前面constraint age_check部分可加可不加
)
```

6. foreign key（外键）

1. format

```
create table fk_example(
    {column1} {type1},
    ...
    constraint id_fk {column1}
        references {outer_table1}({outer_column1})
)
```

2. remark：

outer_column1一定要是unique或者primary key

column1的值一定出自outer_column1，但是column1，outer_column1不一定完全相同

Alter

1. set or drop not-null constraint

1. syntax

```
/*
    [...]表示可有可无
    {|} 表示任选其一
    {} 表示标识符
*/
alter table [if exist] [only] {table_name}
alter {column_name} {set | drop} not null -- 要保证不含null值
-- 不需要add constraint，因为有drop方法，不需要自定义约束的标识符
```

2. example

```
alter table customer
alter passw set not null,
add constraint nn check(passw is not null)
```

2. add/drop **unique, primary key, foreign key, check**

1. syntax

   adding:

```
-- unique
alter table {table_name}
add constraint cons_name unique (column1, column2, ...);

-- primary key
alter table {table_name}
add constraint cons_name primary key (column1, column2, ...);

-- foreign key
alter table {table_name}
add constraint cons_name foreign key({inner_col})
    references {outer_table} ({outer_table});

-- check
alter table {table_name}
add constraint cons_name check ({condition});
```

   droping:

```
alter table {table_name}
drop constraint cons_name
-- that's why declaring the constraint name explicitly matters
```

3. change data type

   1. syntax

```
alter table {ta_name}
alter column {col_name} type {new_type}
```

   2. example

```
alter table customer
alter column phone_number type varchar(2);
```

4. add/drop column

   1. syntax

```
-- adding
alter table {ta_name}
add column {col_name} {type_name};

-- droping
alter table {ta_name}
drop column {col_name};
```

2. example

```
alter table table2
add column age int;

alter table table2
drop column age;
```

5. rename

1. syntax

```
-- rename the whole table
alter table {ta_name}
rename to {new_table};

-- rename a column
alter table {ta_name}
rename column {col_name} to {new_col}
```

2. example

```
alter table table1
rename to table2

alter table table2
rename column age to ages
```

6. check constraint

```
select tc.constraint_name, tc.constraint_type, tc.table_name
from information_schema.table_constraints tc
where tc.constraint_schema="current_schema"();
```

7. drop table

```
drop table {ta_name};
```

如果存在外部键指向表内键，则无法删除，解决方法是alter其他表格，将外部键全部删除