Mysql的SQL语言

- 基础操作
  - 数据删除
    - 关键字：**DELETE FROM** 和 **TRUNCATE**
      - 举例：**DELETE FROM** student **WHERE** ID = 2001；
  - 修改表结构
    - 关键字：**ALTER TABLE**
      - 添加列
        - 关键字：**ADD**
          - **ALTER TABLE** student **ADD** score **VARCHAR**(20);
      - 删除列
        - 关键字：**DROP**
          - **ALTER TABLE** student **DROP** score;
      - 修改列名和类型
        - 关键字：**CHANGE**
          - **ALTER TABLE** student **CHANGE** score stage **VARCHAR**(30);
            - (score是旧列名，stage是新列名)
      - 修改表名
        - 关键字：**TO**
          - **ALTER TABLE** student **TO** stu;
  - 数据插入
    - 关键字：**INSERT INTO**
      - 格式1：**INSERT INTO** student (ID,name,...) **VALUES** (2001,'Albert',...);
        - 需要插入的列名可以选择所有列名中的几个也可以交换顺序
      - 格式2：**INSERT INTO** student **VALUES** (2001,'Albert',...);
      - 注意：插入的数据一定要和列的数据类型相符
  - 数据修改
    - 关键字：**UPDATE SET**
      - 格式1：无条件
        - **UPDATE** student **SET** id = 2002;
          - **注意：没有条件的话就会把所有的数据都修改**
      - 格式2：有条件
        - **UPDATE** student **SET** id = 2002 **WHERE** id = 2004；
        - **UPDATE** student **SET** id = 2002, score = 100 **WHERE** id = 2001;

- 约束
  - 主键约束
    - 关键字：**PRIMARY KEY**
      - 主键的作用和性质：
        - 性质
          - 1. 有一个表中只能有一个主键（多列可以合并成一个）
          - 2. 主键这一列（或者多列）的数据不能重复且不为NULL
        - 作用
          - 在列上加标记，标识一行，提高效率
      - 主键的添加
        - 创建单列主键
          - 格式1：在创建表的时候
            - id INT **PRIMARY KEY**
          - 格式2：在表的最后（pk1是约束名, constraint pk1 可以省略）
            - id INT; **CONSTRAINT** pk1 **PRIMARY KEY** (id);
        - 创建多列主键
          - 创建多列主键的注意事项
            - 1. 联合主键所在的列对应的每一行的数据不完全相同即可 （比如 Albert 2001 和 Albert 2002有一个不相同即可）
            - 2. 任意一个数据都不能为NULL
            - 3. 一张表中只能有一个联合主键
          - **CREATE TABLE name01 (..... ..... PRIMARY KEY (字段1，字段2，字段3.....));**
        - 通过**修改表结构**来添加主键（在创建表语句的末尾）
          - 添加单列主键
            - **ALTER TABLE** emp **ADD PRIMARY KEY** (id);
          - 添加联合主键
            - **ALTER TABLE** emp **ADD PRIMARY KEY** (name, id);
      - 主键的删除 **DROP**
        - **ALTER TABLE** emp **DROP PRIMARY KEY**;
        - 无论是联合主键还是单列主键的语法都相同
  - 自增长约束
    - 关键字：**AUTO_INCREMENT**
      - 注意事项
        - 1. 自增长约束只运用在主键上

- 2. 自增长约束的默认初始化值是1，即从1开始自增长
- 3. 自动增长到该数据类型的上限
- 4. 自增长的字段只能是整型数据，有 NOT NULL 属性
- 5. 自增长的初始值可以自定义
- 6. delete 删除数据之后不会把自增长点删除，即 1, 2 ,delete操作, 3 , 4　而 truncate 清空数据之后，自增长会重新初始化，即 1,2, truncate,1, 2

- 应用语法
  - 基础语法
    - CREATE TABLE product(pid **INT PRIMARY KEY AUTO_INCREMENT**, pname VARCHAR(20) NOT NULL);
  - 自定义语法
    - 方式1：
      - CREATE TABLE product(pid **INT PRIMARY KEY AUTO_INCREMENT**) **AUTO_INCREMENT = 100** ;
    - 方式2: 通过更改表结构的方式
      - CREATE TABLE product(pid **INT PRIMARY KEY AUTO_INCREMENT**); **ALTER TABLE** product **AUTO_INCREMENT = 100**;

- 非空约束
  - 关键字：**NOT NULL**
    - 创建非空约束
      - 方式1：在创建表的时候
        - id INT **NOT NULL**; name VARCHAR(20) **NOT NULL**;
      - 方式2：在创建表之后 关键字：**MODIFY**
        - **ALTER TABLE** 表名 **MODIFY** 字段 类型 **NOT NULL**;
        - CREATE TABLE student(name VARCHAR(20));
          ALTER TABLE student **MODIFY** name VARCHAR(20) **NOT NULL**;　注意数据类型不能漏掉
    - 删除非空约束（只需要把NOT NULL 删掉即可）
      - 关键字：**MODIFY**
        - ALTER TABLE student **MODIFY** name VARCHAR(20);
- 唯一性约束
  - 关键字：**UNIQUE**
    - 作用

        - 对应字段的值不能重复
        - NULL就可以重复，因为在Mysql中NULL与任何值包括其自身都不相同
    - 创建唯一性约束
        - 方式1：在创建表的时候

          name VARCHAR(20) UNIQUE;
        - 方式2：在创建表通过修改表结构 (**ALTER TABLE + ADD CONSTRAINT**)

          CREATE TABLE student(
          name VARCHAR(20)
          );
          ALTER TABLE student **ADD CONSTRAINT** uni01 **UNIQUE** (name);
          (其中uni01是唯一约束名)
    - 删除唯一性约束 **(ALTER TABLE + DROP INDEX)**
        - 如果有约束名

          ALTER TABLE student DROP INDEX uni01;
        - 如果没有约束名，就把唯一约束作用的列名作为约束的名字

          ALTER TABLE student DROP INDEX name;
- 默认约束
    - 关键字：**DEFAULT**
        - 作用
            - 约束某列的默认值
        - 创建方法
            - 方式1：在创建表的时候

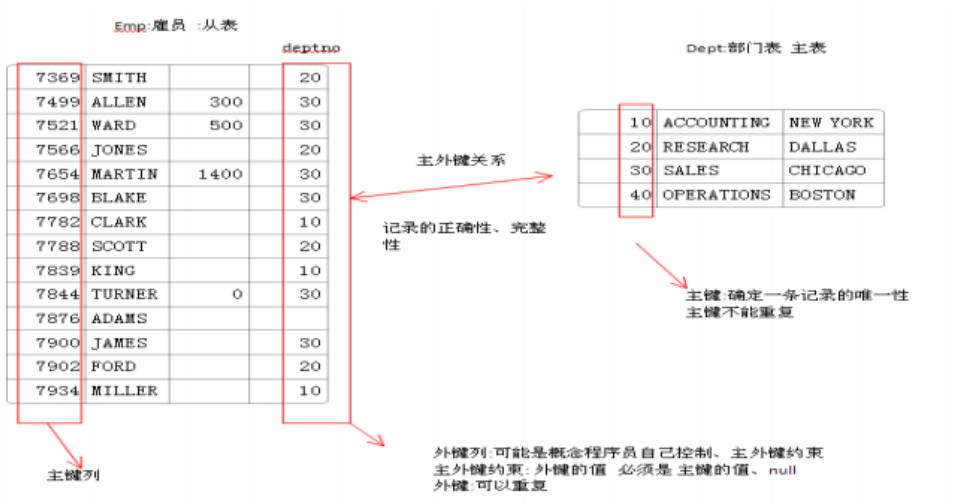
              name VARCHAR(20) **DEFAULT** 'Albert'
            - 方式2：在创建表之后，通过修改表结构 (**ALTER TABLE + MODIFY + DEFAULT**)

              CREATE TABLE student (
              name VARCHAR(20),
              id INT
              );
              ALTER TABLE student **MODIFY** name VARCHAR(20) **DEFAULT** 'Albert'; 数据类型不要忘了
        - 删除方法
            - 本质也是通过修改表结构,只需要将默认约束改成NULL即可 (**ALTER TABLE + MODIFY + DEFAULT**)

              ALTER TABLE student **MODIFY** name VARCHAR(20) **DEFAULT** NULL;
- 零填充约束
    - 关键字：**ZEROFILL**
- **外键约束**

主要限制数据的增删改

- 关键字：**FOREIGN KEY**
  - 注意事项
    - 与主键约束一起使用，具有关联性

      主键所在的表是主表（父表），外键所在的表是从表（子表）
    - 必须先给主表添加数据，给从表添加数据时，外键列的值不能随便写，必须依赖主表的主键列
  - 

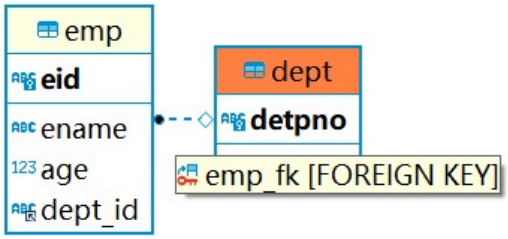
7369	SMITH		20
7499	ALLEN	300	30
7521	WARD	500	30
7566	JONES		20
7654	MARTIN	1400	30
7698	BLAKE		30
7782	CLARK		10
7788	SCOTT		20
7839	KING		10
7844	TURNER	0	30
7876	ADAMS		
7900	JAMES		30
7902	FORD		20
7934	MILLER		10

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

    - 创建**一对多**的外键约束
      - 方式1：在创建表的时候设置外键约束 **(FOREIGN KEY + REFERENCES)**

        可以用**CONSTRAINT**关键字给外键起名字
        - 格式：[CONSTRAINT <name>] **FOREIGN KEY** 字段1 [, 字段2,.....] **REFERENCES** <主表名> 主键列1 [, 主键列2,.....]
          - 举例

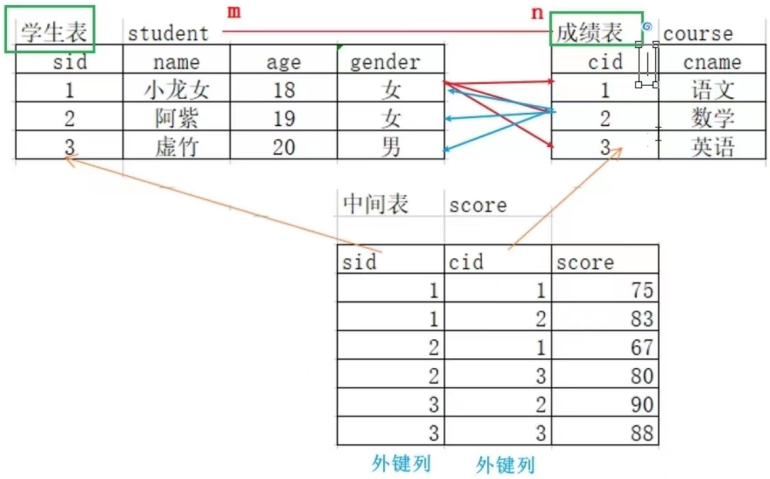
            ```
            -- 创建表
            CREATE TABLE IF NOT EXISTS dept(
            detpno VARCHAR(20) PRIMARY KEY,
            name VARCHAR(20)
            );

            -- 创建从表
            CREATE TABLE emp(
            eid varchar(20) PRIMARY KEY,
            ename VARCHAR(20),
            age INT,
            dept_id VARCHAR(20),
             CONSTRAINT emp_fk  FOREIGN KEY (dept_id) REFERENCES dept
            (detpno)
            );
            ```



- 方式2：在创建表之后通过修改表结构（**ALTER TABLE + ADD CONSTRAINT + FOREIGN KEY + REFERENCES**）
  - 格式：**ALTER TABLE** <name> **ADD CONSTRAINT** <foreign key name> **FOREIGN KEY** (<列名>) **REFERENCES** <主表名> (<列名>);
    - 举例

      ALTER TABLE emp **ADD CONSTRAINT** emp_fk **FOREIGN KEY** (dept_id) **REFERENCES** dept (detpno);

-

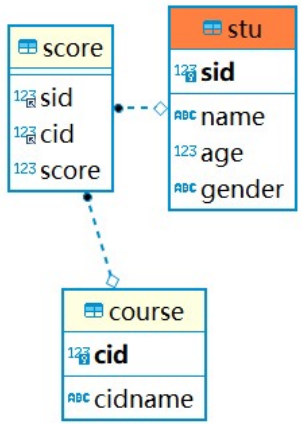


中间表有两个外键列，对应着两个主表

- 创建**多对多**的外键约束
  - 注意：分组字段可以是一个也可以是多个
    - 多个分组条件意味着：当这些条件全部相同的时候才会被分到一组
  - 多对多的**外键约束**
    - 举例

```
USE school;
-- lefthand side father table
CREATE TABLE IF NOT EXISTS stu(
sid INT PRIMARY KEY AUTO_INCREMENT,
name VARCHAR (20),
age INT,
gender VARCHAR(20)
);
```

```
-- righthand side father table
CREATE TABLE course(
cid INT PRIMARY KEY AUTO_INCREMENT,
cidname varchar(20)
);

-- middle son table
CREATE TABLE score(
sid int,
cid int,
score double
);

-- build the foreign key constraint (2 times)
ALTER TABLE score ADD FOREIGN KEY (sid) REFERENCES stu (sid);
ALTER TABLE score ADD FOREIGN KEY (cid) REFERENCES course(cid);
INSERT INTO stu VALUES (1,'Albert',16,'male'),(2,'Vivian',19,'female'),
(3,'Levon',20,'male');
INSERT INTO course VALUES (1,'Chinese'),(2,'Maths'),(3,'English');
INSERT INTO score VALUES (1,1,100),(1,2,100),(2,1,90),(2,3,99),(3,2,89),
(3,3,99);
```



- 注意事项
  - 1.对子表进行数据插入的时候必须要让其依赖于两个（甚至更多）父表
  - 2.数据删除时：父表被子表依赖的数据不能删除，否则可以删除；子表的数据可以任意删除
- 删除数据（**DELETE FROM + WHERE**）
  - 1.主表被从表依赖时，数据不能删除；否则可以删除
  - 2.从表的数据可以随便删除
- 删除外键约束：通过改变表结构的方式（**ALTER TABLE + DROP**）

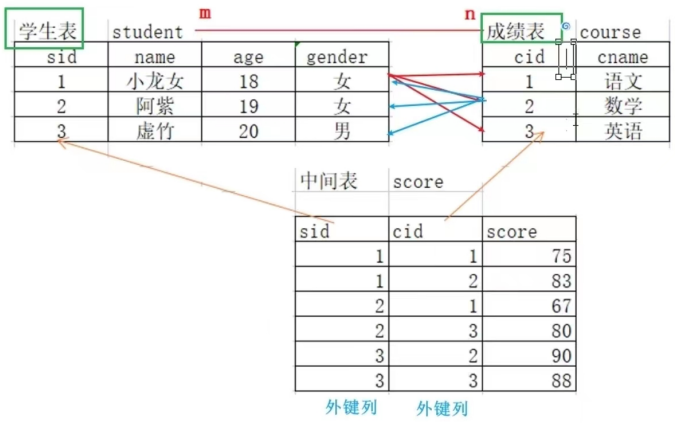
  体现了外键约束名的重要性
  - 格式：**ALTER TABLE** <从表名> **DROP FOREIGN KEY** emp_fk;

- 查询
  - 简单查询
  - 条件查询
  - 排序查询
  - 聚合查询
  - 分组查询
    - 关键字：GROUP BY
      - 格式：**SELECT** 字段1，字段2... **FROM** 表名 **GROUP BY** 分组字段 **HAVING** 分组条件

        SELECT id, count(*) FROM student GROUP BY id;

        - 进行group by 操作之后，将原先的大表按照分组字段分成 n 张临时表，之后SELECT FROM 操作就是针对这些临时表做的
        - Having 就是对分组之后的结果再进行条件判断
      - 注意：分组字段可以是一个也可以是多个
        - 多个分组条件意味着：当这些条件全部相同的时候才会被分到一组
  - 分页查询
  - 运算符操作
- 多表查询
  - 表与表之间的关系
    - 一对一
    - 多对一
    - **多对多**

      一般需要创建一个**中间表**来分开展示数据
      - 多对多的**外键约束**

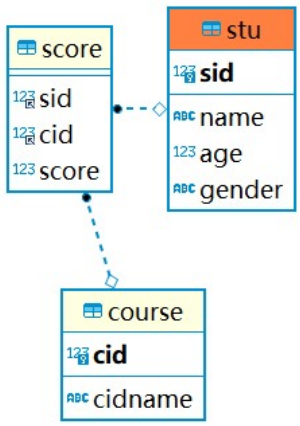
        中间表有两个外键列，对应着两个主表
        - 举例

          USE school;

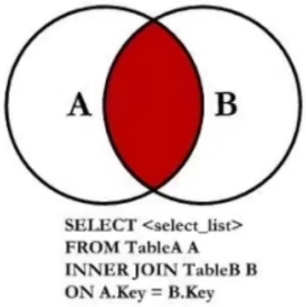
          -- lefthand side father table

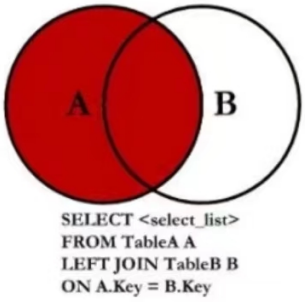
          CREATE TABLE IF NOT EXISTS stu(

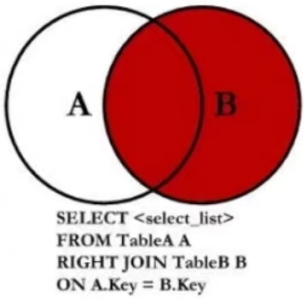
```
sid INT PRIMARY KEY AUTO_INCREMENT,
name VARCHAR (20),
age INT,
gender VARCHAR(20)
);

-- righthand side father table
CREATE TABLE course(
cid INT PRIMARY KEY AUTO_INCREMENT,
cidname varchar(20)
);

-- middle son table
CREATE TABLE score(
sid int,
cid int,
score double
);

-- build the foreign key constraint (2 times)
ALTER TABLE score ADD FOREIGN KEY (sid) REFERENCES stu (sid);
ALTER TABLE score ADD FOREIGN KEY (cid) REFERENCES course(cid);
INSERT INTO stu VALUES (1,'Albert',16,'male'),(2,'Vivian',19,'female'),
(3,'Levon',20,'male');
INSERT INTO course VALUES (1,'Chinese'),(2,'Maths'),(3,'English');
INSERT INTO score VALUES (1,1,100),(1,2,100),(2,1,90),(2,3,99),(3,2,89),(3,3,99);
```

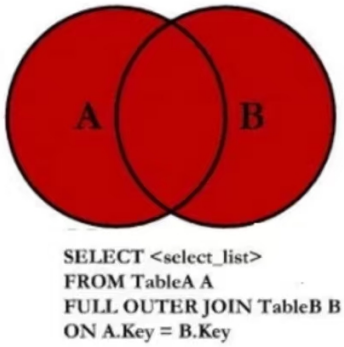


- 注意事项
  - 1.对子表进行数据插入的时候必须要让其依赖于两个（甚至更多）父表
  - 2.数据删除时：父表被子表依赖的数据不能删除，否则可以删除；子表的数据可以任意删除
- **外键约束**主要是对数据的增删改进行约束而对数据的查询没有约束
- 多表联合查询









- 交叉连接查询
  - 其结果类似于两张表相乘，会产生冗余数据

    但也可以通过条件查询进行有效选择

    SELECT * FROM dept4,emp4;

    SELECT * FROM dept4,emp4 WHERE dept4.deptno = emp4.dept_id ;
- 内连接查询
  - 其结果求出的是表的**交集**
    - 方式1：交叉连接查询加条件即为内连接查询

      SELECT * FROM dept4,emp4 WHERE dept4.deptno = emp4.dept_id ;
    - 方式2：显示内连接 **（SELECT FROM + INNER JOIN + ON）**
      - 语法：**SELECT** 字段1，... **FROM** 表1 **INNER JOIN** 表2 **ON** 条件

        *SELECT * FROM* dept4 IN**NER JOIN e**mp4 ON **d**ept4.deptno = emp4.dept_id;

        INNER 可以省略
        - 还可以通过增加命令语句来实现更为复杂的效果

          ```
          SELECT
           name, deptno,
           count(dept_id) AS total_cnt
          FROM dept4
          INNER JOIN emp4
          ON dept4.deptno = emp4.dept_id
          GROUP BY dept_id
          HAVING total_cnt >= 3
          ORDER BY total_cnt DESC;
          ```
- 外连接查询
  - 左外连接（**LEFT OUTER JOIN**）

会把左表的数据全部输出，右表不包含的数据用NULL代替

- 语法：**SELECT** 字段1，... **FROM** 表1 **LEFT OUTER JOIN** 表2 **ON** 条件
  - 注意可以连接多个表

```
SELECT * FROM
    LEFT JOIN B on 条件1
    LEFT JOIN C on 条件2
    ......
```

- 右外连接(**RIGHT OUTER JOIN**)
- 满外连接（**UNION**）

UNION默认去重

- 其结果求出的是表的**并集**

Mysql对FULL JOIN无法执行，因此用UNION命令代替
UNION是上下拼接

- UNION是上下拼接；JOIN是左右拼接

```
SELECT * FROM dept4 LEFT OUTER JOIN emp4 ON dept4.deptno = emp4.dept_id
UNION
SELECT * FROM DEPT4 RIGHT OUTER JOIN emp4 ON dept4.deptno =
emp4.dept_id;
```

- **UNION ALL** 没有去重的上下拼接

- 子查询
  - SELECT查询的嵌套

引入：
在一张表中，如果想找到年龄最大的一个或对个人
单单用 SELECT ename,max(age) FROM emp4;
是得不到正确结果的，其结果是表中的第一个人名
还有表中的最大年龄，与要求和事实不符，这个
时候就需要对最大年龄和人名进行匹配（分步执行）

- 单行单列
  - 让第一次查询为一个值

```
SELECT max(age) FROM emp4;
SELECT * FROM emp4 WHERE age = ( SELECT max(age) FROM emp4 );
```

- 单行多列

多行单列

- 用关键字：**IN**

```
SELECT deptno FROM dept4 WHERE name IN ('研发','销售'); # 结果是两个
SELECT * FROM emp4 WHERE dept_id IN (SELECT deptno FROM dept4 WHERE
name IN ('研发','销售'));
```

- 表自关联

以上内容整理于 幕布文档