本笔记大多代码是以MySQL为基准，关键字使用大写，对于“泛型关键字”以`_`开头，后接着大写字母`_CONSTRAINT`

基础

1. 一般来说，为了区分，关键字用大写，其他用小写
2. SQL并没有规定语句末尾要`;`，但是`MySQL`需要

MySQL

`sudo service mysql start`：启动mysql服务，实验楼默认mysql是没有启动的 `mysql -u root`：登录账户，实验楼的密码默认为空 `QUIT`或`exit`:退出mysql交互界面 `source filename.sql`执行文件中的sql代码

Database

1. 创建：`CREATE DATABASE database_name`
2. 显示所有的数据库：`SHOW DATABASES`
3. 使用某个数据库：`USE databases_name`
4. 删除：`DROP DATABASE database_name`

Table

1. 显示当前数据库所有表：`SHOW TABLES`
2. 创建新的表

```
CREATE TABLE table_name(
    col1_name DATA_TYPE,
    col2_name DATA_TYPE,
);
```

创建与t2相同形式的表t1

```
CREATE TABLE t1 LIKE t2;
```

将查询结果存储为新表：注，MySQL不用加最后的`WITH DATA`

```
CREATE TABLE t1 AS (query_statment) WITH DATA;
```

3. 显示表的主要描述：`DESCRIBE table_name`
4. 重命名：`RENAME TABLE ori_name TO new_name`或`ALTER TABLE ori_name TO new_name`或`ALTER TABLE ori_name TO new_name`
5. 查看：`SELECT * FROM table_name`
6. 删除：`DROP TABLE table_name`
7.

Record(Row)

1. 插入：`INSERT INTO table_name(col1_name,col2_name) VALUE(col1_value,col2_value)`
   注：如果插入的值和定义一样，可以省略table_name后面()的内容
2. 删除：`DELETE FROM table_name WHERE condition` 删除全部行`DELETE FROM table_name`或`DELETE * from table_name`
3. 修改(更新)

```
UPDATE table_name SET col1_name = val1, col2_name = val2
WHERE condition;
```

条件更新

```
UPDATE t1 SET c1 = CASE
    WHEN c1 > 100 THEN c1 - 0
    WHEN condition THEN val
    ELSE val
END;
```

Column

1. 增加：

```
ALTER TABLE table_name ADD COLUMN col_name DATA_TYPE CONTRAINT_TYPE; /*或*/
ALTER TABLE table_name ADD col_name DATA_TYPE CONTRAINT_TYPE;
```

可以发现：新增加的列，被默认放置在这张表的最右边。如果要把增加的列插入在指定位置，则需要在语句的最后使用`AFTER`关键词(`AFTER 列1`表示新增的列被放置在“列1”的后面)。

```
ALTER TABLE table_name
ADD new_col_name DATA_TYPE CONTRAINT_TYPE AFTER old_col_name
```

```
ALTER TABLE table_name
ADD new_col_name DATA_TYPE CONTRAINT_TYPE FIRST /*放在最前面*/
```

2. 删除一列

```
ALTER TABLE table_name DROP COLUMN col_name;
```

3. 重命名一列

```
ALTER TABLE table_name
CHANGE ori_col_name new_col_name _DATA_TYPE _CONTRAINT_TYPE;
```

注意：这条重命名语句后面的“数据类型”不能省略，否则重命名失败。 当原列名和新列名相同的时候，指定新的数据类型或约束，就可以用于修改数据类型或约束。需要注意的是，修改数据类型可能会导致数据丢失，所以要慎重使用。

Data Type

数据类型的种类取决于数据库软件本身的实现，下面的类型以MySQL8.0为基础

Numeric

key word	uint size(byte)	allocated length	range(signed)	range(unsigned)
TINYINT	1		$[-2^7, 2^7-1]$	$[0, 2^8-1]$
SMALLINT	2		$[-2^{15}, 2^{15}-1]$	$[0, 2^{16}-1]$
MEDIUMINT	3		$[-2^{23}, 2^{23}-1]$	$[0, 2^{24}-1]$
INT(INTEGER)	4		$[-2^{31}, 2^{31}-1]$	$[0, 2^{32}-1]$
BIGINT	8		$[-2^{63}, 2^{63}-1]$	$[0, 2^{64}-1]$
FLOAT	4			
DOUBLE	8			
DECIMAL				

Date and Time

String

Spatial Data

JSON Data

null

SQL将涉及空值比较的结果设置为`unknown`，下面几个除外

1. `false AND unknown`为false
2. `true OR unknown`为true

`IS NOT NULL`：该谓语表示不是空值

```
SELECT * FROM t1
WHERE c1 IS NOT NULL;
```

`IS NULL`与上面相反

自定义（MySQL暂不支持）

创建：

```
CREATE TYPE mytype AS NUMERIC(12,2) FINAL;
```

删除和修改`drop type`和`alter type`添加限制：中文课本P79

Constraint

在通用代码中用`_CONSTRAINT`代替constraint各种类型

How To Use

在此部分给出的示例都是针对一个列而言

1. 在创建表时加入Constraint

```
CREATE TABLE table_name(
    col1_name _DATA_TYPE _CONSTRAINT,
);
```

2. 为存在的列添加Constraint

```
ALTER TABLE table_name
ADD _CONSTRAINT (col_name); /*或*/
ALTER TABLE table_name
ADD CONSTRAINT constraint_name _CONSTRAINT (col_name);
```

2. 修改已有的Constraint：同"重命名一列"

```
ALTER TABLE table_name
CHANGE ori_col_name new_col_name DATA_TYPE CONTRAINT_TYPE;
```

3. 删除

```
ALTER TABLE table_name
DROP _CONSTRAINT; /*或*/
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

Constraint Type

NOT NULL

约束强制列不接受 NULL 值，示例同`how to use`

DEFAULT

默认值常用于一些可有可无的字段，比如用户的个性签名，如果用户没有设置，系统应该给他设定一个默认的文本，例如“这个人太懒了，没有留下任何信息” 使用同`How To Use`

AUTO_INCREMENT

会在新记录插入表中时生成一个唯一的数字，默认在该列上一个插入的值增加1，若没有，则从0开始。要求：

1. 一个表中只能有一个自增值；
2. 该自增值必须为主键

```
CREATE TABLE t(
    id int AUTO_INCREMENT
);
```

如果需要修改自增值起始值

```
ALTER TABLE table_name AUTO_INCREMENT = val;
```

PRIMARY KEY

- 约束唯一标识数据库表中的每条记录
- 主键必须包含唯一的值（primary key具有unique属性）
- 主键列不能包含 NULL 值，
- 每个表只能有一个主键。 示例可以是`How TO Use`所呈现的，也可以是

```
CREATE TABLE table_name(
    col1_name _DATA_TYPE,
    col2_name _DATA_TYPE,
    PRIMARY KEY (col1_name)
);
```

若为多个列设置主键，需要自行起一个名字，下面起名为constraint_name

1. 创建

```
CREATE TABLE table_name(
    col1_name _DATA_TYPE,
    col2_name _DATA_TYPE,
    col3_name _DATA_TYPE,
    CONSTRAINT  constraint_name PRIMARY KEY (col1_name,col2_name);
);
```

2. 为存在的添加

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name PRIMARY KEY (col1_name,col2_name);
```

3. 删除

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

UNIQUE

它规定一张表中指定的一列的值必须不能有重复值，即这一列每个值都是唯一的。 使用类似`PRIMARY KEY`

CHECK

CHECK 约束用于限制列中的值的范围。 使用类似`PRIMARY KEY`，但是需要在`CHECK`后面加上`(condition)`。

```
CREATEE TABLE t(
    id INT CHECK(id > 0)
);
```

FORIGEN KEY

一个表中的FOREIGN KEY指向另一个表中的UNIQUE KEY（具有该性质即可。 **暂时没有发现FORIGEN KEY可以像How To Use那样使用** 创建时添加

```
CREATE TABLE table1_name(
    col1_name DATA_TYPE,
    col2_name coll_type,
    FOREIGN KEY (col1_name) REFERENCES table2_name(col_name)
);
```

为存在的添加

```
ALTER TABLE table1_name
ADD FOREIGN KEY (table1_col_name) REFERENCES (table2_col_name)
```

删除 同`How To Use` 多于多个列的`FOREIGN KEY` 为存在的添加

```
ALTER TABLE table1_name
ADD CONSTRAINT constraint_name
FOREIGN KEY (col1_name)
REFERENCES table2_name(col_name);
```

删除

```
ALTER TABLE table_name
DROP FOREIGN KEY constraint_name;
```

级联操作`FOREIGN KEY (c1) REFERENCES t2(c1) ON DELETE CASCADE ON UPDATE CASADE;` `ON DELETE CASCADE`：外键本身被删除，跟着删除 `ON UPDATE CASCADE`：外键本身被更新，跟着更新 `ON DELETE SET NULL`：外键本身被删除，设置为空 `ON DELETE SET DEFAULT`：外键本身被删除，设置默认值

互相引用外键的插入：`INITIAL DEFERRED`来推迟事务的检查，未证实

ASSERTION

断言表示数据库总能满足的一个条件。域约束(check)和参照完整性约束(foreign key)就是特殊的断言。目前MySQL好像还没有支持

```
CREATE ASSERTION name CHECK condition
```

Query

String Match

1. `_`表示一个未指定的字符
2. `%`表示多个未指定的字符

Bacial Query

1. 查看一张表的所有内容

```
SELECT * FROM table_name;
```

2. 查看一张表某些列的内容

```
SELECT col1_name, col2_name FROM table_name;
```

3. Conditional Query Logical Operation: `=`, `<=`, `<`, `>=`, `>`

Op	
ALL	如果一组的比较都为 TRUE，那么就为 TRUE。
AND	如果两个布尔表达式都为 TRUE，那么就为 TRUE。
ANY	如果一组的比较中任何一个为 TRUE，那么就为 TRUE。
BETWEEN	如果操作数在某个范围之内，那么就为 TRUE。
EXISTS	如果子查询包含一些行，那么就为 TRUE。
IN	如果操作数等于表达式列表中的一个，那么就为 TRUE。
LIKE	如果操作数与一种模式相匹配，那么就为 TRUE。
NOT	对任何其他布尔运算符的值取反。
SOME	如果在一组比较中，有些为 TRUE，那么就为 TRUE。

选择包含字串`s`的`c1`

```
SELECT c1 FROM t1 WHERE c1 LIKE '%s%';
```

见中文课本P49

ESCAPE

该关键字用来定义转义字符 `LIKE 'ab\%cd%'ESCAPE '\'`：匹配以`%abcd`开头的字符串

DISINCT

该关键字使结果(被限制的列)不重复

```
SELECT DISTINCT col_name FROM table_name
```

ALL

该关键字与`DISTINCT`相反，使用方法也一样，但默认就是不去除重复，所以必要性不是很大

join

1. Cartesian product
2. NATURAL JOIN = NATURAL INNER JOIN 暂不支持：natural left outer join
3. JOIN .. USING ... 一种特殊的自然连接，只在指定特定的列相等

```
SELECT * FROM t1 JOIN t2 USING (c1,c2);
```

3. LEFT JOIN ... ON ... = LEFT OUTER JOIN ... ON ...
4. RIGHT JOIN ... ON ...
5. FULL JOIN ... ON...
6. JOIN .. ON ...

AS

该关键字提供了更名操作(别名)

1. Column

```
SELECT col_name1 AS c1 FROM t1;
```

2. Table

```
SELECT t1.c1, t2.c2
FROM table_name1 AS t1, table_name2 AS t2;
```

ORDER BY

该关键字提供了结果的排序功能

```
SELECT * FROM t ORDER BY c1 DESC, c2 ASC;
```

其中，`DESC`表示降序，`ASC`表示升序

WHERE从句

SQL允许在使用元组

```
SELECT c1, c2
FROM t1, t2
WHERE (t1, t2) = (t1_val, t2_val);
```

注：(a1, a2) < (b1, b2)使用的是字典序

集合运算

UNION

```
(SELECT c1 FROM t1)
UNION
(SELECT c2 FROM t2);
```

`UNION`表示并集，默认自动去除重复

UNION ALL

类似`UNION`，但是不去除重复

others

INTERSECT, EXCEPT MySQL好像还没有实现

FROM子句

```
select s_no
from (
    select sno as s_no, jno as j_no, sum(qty) as sum_qty
    from spj
    where pno = 'P1'
    group by s_no, j_no
) as tb
where sum_qty > all(
    select avg(qty)
    from spj
    where pno = 'P1' and jno = j_no
    group by jno
);
```

WITH

上面的FROM子句等价于

```
with tb(
    select sno as s_no, jno as j_no, sum(qty) as sum_qty
    from spj
    where pno = 'P1'
    group by s_no, j_no
)
select s_no
from tb
where sum_qty > all(
    select avg(qty)
    from spj
    where pno = 'P1' and jno = j_no
    group by jno
);
```

Aggregation Function

```
SELECT dept_name, AVG(salary) AS avg_salary
FROM instructor
GROUP by dept_name
HAVING avg_salary > 2000;
```

注：`GROUP BY`后面接着的列必须在`SELECT`中出现。 上面的执行顺序：

1. `FROM`找出关系(表)
2. `WHERE`将条件应用到`FROM`上的关系
3. `GROUP BY`形成分组
4. `HAVING`对分组进行筛选
5. `SELECT`进行打印

空关系测试

`EXIST`在作为参数的子查询非空时返回true

```
SELECT course_id FROM section AS s
WHERE semester = 'Fall' AND year = 2019 AND
    EXISTS(
        SELECT * FROM section AS T
        WHERE semester = 'Spring' AND year = 2010 AND S.course_id = T.course_id
    );
```

关系A包含关系B：`not exists (B except A)`，except，MySQL暂时没有实现 重复元组测试`UNIQUE`，没有重复元组返回true，用法同`EXISTS`，MySQL暂时没有实现

```
SELECT t1.c1 FROM t1
WHERE UNIQUE(
        SELECT t2.c2 FROM t2
        WHERE t2.c1 = t1.c1
    );
```

上面等价于

```
SELECT t1.c1 FROM t1
WHERE 1 >= (
        SELECT count(t2.c2) FROM t2
        WHERE t2.c1 = t1.c1
    );
```

Index

索引是一种与表有关的结构，它的作用相当于书的目录，可以根据目录中的页码快速找到所需的内容。当表中有大量记录时，若要对表进行查询，没有索引的情况是全表搜索：将所有记录一一取出，和查询条件进行一一对比，然后返回满足条件的记录。这样做会消耗大量数据库系统时间，并造成大量磁盘 I/O 操作。 而如果在表中已建立索引，在索引中找到符合查询条件的索引值，通过索引值就可以快速找到表中的数据，可以大大加快查询速度。 在使用 SELECT 语句查询的时候，语句中 WHERE 里面的条件，会自动判断有没有可用的索引。 一些字段不适合创建索引，比如性别，这个字段存在大量的重复记录无法享受索引带来的速度加成，甚至会拖累数据库，导致数据冗余和额外的 CPU 开销。

1. 对一张表建立某个列的索引

```
ALTER TABLE table_name ADD INDEX index_name (col_name); /*或*/
CREATE INDEX index_name ON table_name (col_name);
```

2. 显示表的索引

```
SHOW INDEX FROM table_name
```

3. 删除

```
drop index index_name on table_name;
```

View

出发点：让用户看到整个逻辑模型是不合适的，需要向用户隐藏特定的数据 定义：视图是指计算机数据库中的视图，是一个虚拟表，其内容由查询定义。数据库系统存储与视图相关联的查询表达式，当视图关系被访问时，其中的元组是通过计算查询结果而被创建出来的。

1. 创建

```
CREATE VIEW view_name AS query_statement;/*或*/
CREATE VIEW view_name(c1,c2,...) AS query_statement;/*或*/
```

2. 物化视图(materialized view) 物化视图：用于定义视图的实际关系改变，视图也会跟着修改的视图物化视图维护(materialized view maintenance)(视图维护)：保持物化视图一直在最新状态的过程
3. 使用视图和使用表完全一样，只需要把视图当成一张表就OK了。视图是一张虚拟表。 用视图表达的数据库修改必须被翻译为对数据库逻辑模型中实际关系的修改。除了一些有限的情况之外，一般不允许对视图关系进行需改，不同的数据库指定了不同的条件以允许更新视图关系。
4. `WITH CHECK OPTION` 在结尾加上这条语句，可以限制向视图中插入一条不满足where自从句的语句。

事务(transaction)

事务由查询和（或）更新语句的序列组成

- commit work：提交当前事务，将该事物所做的更新在数据库中持久保存
- rollback work：回滚当前事务，即撤销该事务中所有SQL语句对数据库的更新

一个事务或者在完成所有步骤后提交其行为，或者在不能完成其所有动作的情况下回滚其所有动作。

授权(Authorization)

赋予权限

```
GRANT <权限列表> ON <表名/视图> TO <用户/角色>
```

权限包括：`SELECT`, `INSERT`, `UPDATE`, `DELETE`, `ALL PRIVILEGES`

```
GRANT UPDATE (c1,..) ON t1 TO u1;
```

收回权限：将`GRANT`换成`REOVKE`

角色： 创建：`CREATE ROLE r1;` 将角色赋予用户：`GRANT r1 TO u1;`

用户 创建：

```
create user 'user_name'@'hostIP' identified by 'password';
```

hostIP: 表示该用户可以在哪个主机IP访问数据库，默认为'%'，所有都可以访问 password：默认为空

权限的转移：`WITH GRANT OPTION`

```
GRANT SELECT ON t1 TO u1 WITH GRANT OPTION
```

上述命令表示u1可以将此权限转移给其他人

权限的收回

```
REVOKE SELECT ON t1 FROM u1;
```

注：级联收回(最后加`CASCADE`)是默认的，若不想要级联收回，则在最后加入`RESTRICT` 收回授权给他人的权限(mysql未测是否支持)

```
REVOKE GRANT OPTION FOR SELECT ON t1 FROM u1;
```

触发器(Trigger)

定义：触发器是一条语句，当对数据库作修改时，它自动被执行。 标准的触发器例子(MySQL不支持)

```
CREATE TRIGGER trigger1 ATTER INSERT ON table1
    REFERENCING NEW ROW AS nrow
    FOR EACH ROW
    WHEN(nrow.c1 NOT IN(
        ...
    ))
    BEGIN
        ROLLBACK
    END;
```

MySQL版本的

```
delimiter //;
create trigger tr1 before insert on db_spj.spj
for each row
begin
    if new.qty < 0
    then
      SIGNAL sqlstate '45001' set message_text = "Erroneous qty";
    end if;
end;
//
create trigger tr2 before update on db_spj.spj
for each row
begin
```

```
    if new.qty < 0
    then
      SIGNAL sqlstate '45001' set message_text = "Erroneous qty";
    end if;
end;
//
delimiter ; //
```

未完的待续

primary key, unique,check可以把关键字放在后面

not null, auto_increment

foreign key 不支持