数据库系统

关系型数据库

关系代数语言

运算符		含义	运算符		含义
集合运算符	U	并(UNION)	比较运算符	>,<	大于（小于）
	-	差(EXCEPT)		⩾, ⩽	大于等于（小于等于）
	∩	交(INTERSECT)		=	等于
	×	笛卡尔积		<>	不等于
关系运算符	σ	选择	逻辑运算符	¬	非
	∏	投影		∧	与
	∞	连接		∨	或
	÷	除	查询IS系的全体学生	$\sigma Sdept =' IS'(Student)$	

- 笛卡尔积：将两个表中的数据进行相乘式的组合（相同的域使用**表名.域**的形式）
- 投影：结果仅显示投影的域
- 连接：将两个表的共有属性进行组合，其他未能组合的元组称为悬浮元组，而左、右连接就是将某一侧的悬浮元组直接在结果表中的显示，空白域的属性置为 $NULL$
- 除：保留父表（多个域）中完全满足子表（域为父表中的某个或几个）且去掉子表域的元组

SQL

基本数据类型

数据类型	含义
CHAR(n)	长度为n的字符串
VARCHAR(n)	最大长度为n的变长字符串
INT, SMALLINT, BIGINT	整数（4字节） 短整数（2字节） 大整数（8字节）
DECIMAL(p, d)	总共p位数字，其中小数的位数是d位
DATE, TIME, TIMESTAMP	日期，YYYY-MM-DD 时间，HH:mm:SS 时间戳

结构的创建

模式schema

创建数据库首先要定义模式，模式的定义可以包含字句

删除模式使用`DROP`关键字，可以选择两种删除选项`CASCADE`或`RESTRICT`

- cascade：级联删除，同时删除模式内的数据库对象
- restrict：限制删除，如果含有下属对象会拒绝删除

SQL

```
create schema <模式名> authorization <用户名>[<表的定义语句>|<视图定义子句>|<授权定义子句>];

create schema "learn" authorization root
create table user(
id int primary key
);

drop schema <模式名> <cascade|restrict>;
```

表table

表的创建与删除与模式类似，表的字段名还需指定数据类型和约束，其中删除时也要判断其下属是否含有外键和触发器，使用`ALTER`可以对表进行修改，比如添加删除列

SQL

```sql
create table test
(
    id   int primary key not null,
    name char(10),
    age  char(2)
);

alter table <表名> add Sdept char (10)

drop table test cascade
```

索引

创建索引可以有效的减少在大量数据中的查询时间，同样使用`ALTER`和`DROP`进行修改（改名）与删除，同时可以使用`CLUSTER`关键字进行聚簇索引（将数据存储与索引放到了一块，找到索引也就找到了数据）的标识

SQL

```sql
create unique index <索引名> on <表名>(<列名>[次序])
```

查询select

进行查询之前首先要进行数据的插入，插入需要注意约束逻辑关系，如表内或表外的域进行了外键连接，则被连接的数据要先存在，查询的基本格式为：

SQL

```sql
SELECT [ALL|DISTINCT] <目标列表达式>···
FROM <表名或视图名>···|AS 别名
WHERE <条件表达式>
GROUP BY <列名> HAVING <条件表达式>
ORDER  BY  <列名> ASC|DESC
```

查询条件	谓词

比较	>、⩽、!=、<>、!<、!>
确定范围	(NOT) BETWEEN AND
确定集合	(NOT) IN
字符匹配	(NOT) LIKE，%代表多个字符，_代表一个字符
空值	IS (NOT) NULL
多重条件	AND,OR,NOT

函数名	含义
COUNT(*)	统计元组的个数
COUNT ([DISTINCT | ALL]) <列名>	一列中值的个数
SUM ([DISTINCT | ALL]) <列名>	数值类型的列的数值总和
AVG ([DISTINCT | ALL]) <列名>	数值类型的列的平均值
MAX / MIN ([DISTINCT | ALL]) <列名>	最大最小值，字符串按字母序

SQL

```sql
# 查询中可以包含计算表达式
select Sno, 2021 - Sage as birth
from student;
# 对查询结果某一列进行小写转换，插入文字标识列
select Sname, '出生年份', 2021 - student.Sage as birth, lower(Sdept)
from student;
# 查询结果中本身含有通配符需要使用转义字符，ESCAPE '\'声明'\'为转义字符
select *
from student
where Sname like '$\_' ESCAPE '\'
# 左外连接，将左侧表中的悬浮数据也保留在结果中
select *
from student
         left outer join sc s on student.Sno = s.Sno;
# 查询最年长学生的成绩
select *
from sc
where Sno in (select Sno
              from student
              where Sage in (select MAX(distinct Sage)
                             from student));
# 带ANY或ALL的子查询
select *
from sc
where Sno < all (select Sno from student where Sage = 19);
# 带 (NOT) EXISTS的子查询：查找没有一门课不选修的学生
select Sname
from student
where not exists(
        select * from course where not exists(select * from sc where sc.Sno = st
udent.Sno and sc.Cno = course.Cno));
```

数据操作

SQL

```sql
# 插入数据
insert into student
values ('12222', 'AAAA', '男', 18, 'MA');
# 更新数据
update student
set Sname='Aidan'
where Sage = 18;
# 删除数据
delete
from student
where Sname = 'Aidan';
```

视图

视图也称作虚表，对应着一条SELECT语句，保存查询的结果，本身不包含数据，是一个语句到表的映射，当基表的数据发生变化时，视图也会相应改变

SQL

```sql
create view <视图名>[<列名>···]
as <子查询>
[with check option]
# with check option表示对增删改的数据保证更新，避免基表改变后，视图中增加脏数据
```

因为视图不实际存在所以在更新视图时最终还是落实到相应的表上，在表UPDATE时就要加上视图的限制条件

SQL

```sql
# 更新视图（视图为IS系的学生）
update viewName
set Sname='Aidan'
where Sno='20211111';
# 转换后的表更新语句
update tableName
set Sname='Aidan'
where Sno='20211111' and Sdept='IS';
```

安全性

自主存取控制

用户可以自定义其他用户对自有数据库的权限，主要元素为数据库对象和操作权限

权限主要为模式创建（包括模式、基本表、视图、索引），数据的增删改查（表、视图、属性列）赋予所有权限为`ALL PRIVILEGES`，通过grant、revoke来进行控制

SQL

```sql
grant <权限> on databaseObj to userName
[with grant option]
# with grant option：赋予某用户权限后，该用户可以将得到的权限给其他人

revoke<权限> on databaseObj from userName cascade
# cascade：当前用户将自己得到的权限赋予其他人也全部收回
```

实际应用中用户可能经常更换但是角色不会，可以创建一个角色，将该角色头衔赋予不同的用户

SQL

```sql
create role roleName
grant <权限> on databaseObj to roleName
grant roleName to userName [with grant option]
revoke<权限> on databaseObj from roleName
```

视图机制

创建不同的视图，将视图的权限给某位用户，将基表数据隐藏起来，防止误操作

完整性

- 实体完整性：主码唯一而且不能为空
- 参照完整性：外码要么没有，要么只有一个
- 用户定义完整性
  - 非空
  - 列值唯一
  - check条件表达式

SQL

```sql
create table tableName
(
    id        int primary key,
    course_id int,
    foreign key (course_id) references course (id),
    # 末尾定义主键 primary key(id)
    age       int unique,
    grade     int not null,
    sex       char(2) check ( sex in ('男', '女') )
);
```

断言

使用断言对数据表进行一个限制（MySql中不支持断言可以使用触发器实现类似操作）

SQL

```sql
# 限制每一门课程最多60名学生选修
create assertion Ass_stu_count
    check(60>=all)(select count(*)
                   from sc
                   group by Cno)
                   );
# 删除断言
drop assertion <断言名>
```

触发器trigger

触发器理解为事件触发某条件产生的规则

SQL

```sql
create trigger <触发器名>
[before|after]<触发事件>on<表名>

referencing new|old row|table as <变量>
for each row|statement
[when<触发条件>]<触发动作体>

# 当修改sc表中grade字段的值大于原来的10%，将其复制到sc_u
create trigger copyOver
    after update of Grade on sc

    referencing
        oldrow as oldTuple,
        newrow as newTuple,
    for each row
    when (NewTuple.Grade >= 1.1*oldTuple.Grade)
        insert into sc_u
        values (oldTuple.Sno, oldTuple.Cno, oldTuple.Grade, newTuple.Grade)

# 删除触发器
drop trigger <触发器名> on <表名>
```