关系数据库标准语言SQL

特点：综合统一，高度非过程化，面向集合的操作方式，以同一种语法结构提供多种使用方式（是独立语言又是嵌入式语言）

SQL功能	动词
数据查询	SELECT
数据定义	CREATE,DROP,ALTER
数据操纵	INSERT,UPDATE,DELETE
数据控制	GRANT,REVOKE

1. 数据定义

定义功能：模式定义、表定义、视图和索引的定义

表 3.3　SQL 的数据定义语句

操作对象	操作方式		
	创　建	删　除	修　改
模式	CREATE SCHEMA	DROP SCHEMA	openGauss 允许
表	CREATE TABLE	DROP TABLE	ALTER TABLE
视　图	CREATE VIEW	DROP VIEW	/
索　引	CREATE INDEX	DROP INDEX	ALTER INDEX

一个数据库可以建立多个SCHEMA，一个SCHEMA下通常包括多个表、视图和索引等数据库对象

1.1 模式的定义与删除

(1) 定义模式

实际上定义一个命名空间

```
CREATE SCHEMA <模式名> AUTHORIZATION<用户名>[<表定义子句><视图定义子句><授权定义子句>]
```

没有指定模式名时模式名默认为用户名

(2) 删除模式

`DROP SCHEMA <模式名> <CASCADE|RESTRICT>`

`CASCADE`：删除模式的同时删除该模式下的全部数据库对象

`RESTRICT`：只有当前模式下没有任何下属的对象时才能执行

1.2 基本表的定义、删除与修改

(1) 定义基本表

`CREATE TABLE <表名>`

`(<列名><数据类型>[<列级完整性约束条件>]，`

`<列名><数据类型>[<列级完整性约束条件>]...`

`[,<表级完整性约束条件>]);`

如果完整性约束条件涉及到该表的多个属性列，则必须定义在表级上，否则既可以定义在列级也可以定义在表级。

`UNIQUE` 和 `PRIMARY KEY`的区别：

- 作为`Primary Key`的属性(组)不能为null，而`Unique`属性可以为null。
- 在一个表中只能有一个`Primary Key`，而多个`Unique`属性可同时存在

```
CREATE TABLE SC
(Sno CHAR(9),
 Cno CHAR(4) UNIQUE, /*列级完整性约束*/
 Grade SMALLINT,
 PRIMARY KEY (Sno,  Cno),
 /* 主码由两个属性构成，必须作为表级完整性进行定义*/
 FOREIGN KEY (Sno) REFERENCES Student(Sno),
 /* 表级完整性约束条件，  Sno是外码，被参照表是Student */
 FOREIGN KEY (Cno) REFERENCES Course(Cno)
);
```

(2) 数据类型

数据类型	含义
CHAR(n)	长度为n的定长字符串（最长为n,如果不够使用空格补齐）
VARCHAR(n)	最大长度为n的变长字符串
CLOB, BLOB	字符串大对象，二进制大对象
INT	长整数（也可以写作INTEGER），4字节
SMALLINT, BIGINT	短整数（2字节），大整数（8字节）
NUMERIC(p,d)	定点数, 由p位数字(不包括符号、小数点)组成, 小数后面有d位数字
REAL	取决于机器精度的单精度浮点数
Double Precision	取决于机器精度的双精度浮点数
FLOAT(n)	可选精度的浮点数，精度至少为n位数字
BOOLEAN	逻辑布尔型（TRUE，FALSE）
DATE	日期，包含年、月、日，格式为YYYY-MM-DD
TIME	时间，包含一日的时、分、秒，格式为HH:MM:SS

(3) 模式与表

每一个基本表都属于某一个模式
一个模式包含多个基本表
定义基本表所属模式

- 方法一：在表名中明显地给出模式名
  `Create table “S-T”.Student（...）; /*模式名为 S-T*/`
- 方法二：在创建模式语句中同时创建表
- 方法三：设置所属的模式

(4) 修改基本表

增加列、删除列

```
ALTER TABLE <表名>
[ADD [COLUMN]<新列名> <数据类型> [完整性约束]]
[ADD [表级完整性约束]]
[DROP [COLUMN]<列名> [CASCADE|RESTRICT]]
[DROP CONSTRAINT<完整性约束名>[CASCADE|RESTRICT]]
[ALTER COLUMN<列名> <数据类型>];
```

(5) 删除基本表

```
DROP TABLE <表名>[RESTRICT| CASCADE]
```

`RESTRICT`：删除表是有限制的。

- 欲删除的基本表不能被其他表的约束所引用
- 如果存在依赖该表的对象，则此表不能 被删除

`CASCADE`：删除该表没有限制。

- 在删除基本表的同时，相关的依赖对象一起删除

缺省时是 `RESTRICT`

1.3 索引的建立与删除

建立索引：加快查询速度(采用顺序文件上的索引、B+树、Hash、位图索引来实现，缺省时为B+树，内模式)

建立索引：DBA 或表的属主（即建立表的人），DBMS一般会自动建立`PRIMARY KEY、UNIQUE`的索引

维护索引：DBMS自动完成， 会增加数据库的负担

使用索引：DBMS自动选择是否使用索引以及使用哪些索引
用户不能显式地选择索引

(1) 建立索引

```
CREATE [UNIQUE] [CLUSTER] INDEX <索引名> ON <表名>
```

```
(<列名>[<次序>]
```

```
[,<列名>[<次序>] ]…
```

```
);
```

UNIQUE：索引的每一个索引值只对应唯一的记录

(2) 修改索引

```
ALTER INDEX <旧索引名> RENAME TO <新索引名> ;
```

(3) 删除索引

```
DROP INDEX <索引名>;
```

删除索引时，系统会从数据字典中删去有关该索引的描述

1.4 数据字典

关系数据库管理系统内的一组系统表， 它记录了数据库中所有的定义信息 :关系模式定义、视图定义、索引定义、完整性约束定义、操作权限、统计信息

对数据的增删改都可能会**影响**到数据字典
对数据的查询也会**用到**数据字典

2.数据查询

```
SELECT [ALL|DISTINCT] <目标列表达式>
[,<目标列表达式>] …
FROM <表名或视图名>[,<表名或视图名> ] …
[WHERE <条件表达式> ]
[GROUP BY <列名1> [HAVING<条件表达式>]]
[ORDER BY <列名2> [ASC|DESC]];
```

2.1 单表查询

一、选择表中的若干列

```
SELECT [ALL|DISTINCT] <目标列表达式>
```

目标表达式可以为：算术表达式、字符串常量、函数、列别名、*

二、选择表中的若干元组

消除取值重复的行：如果没有指定`DISTINCT`关键词，则缺省为`ALL`

(`DISTINCT`不是只对紧跟它的属性取值起作用，而是对它之后所有属性可能取值的组合起作用。因而`DISTINCT`必须紧跟`SELECT`)

查 询 条 件	谓　词
比　较	=，>，<，>=，<=，!=，<>，!>，!<；NOT+上述比较运算符
确定范围	BETWEEN AND，NOT BETWEEN AND
确定集合	IN，NOT IN
字符匹配	LIKE，NOT LIKE
空　值	IS NULL，IS NOT NULL
多重条件（逻辑运算）	AND，OR，NOT

确定范围：`BETWEEN...AND`,`NOT BETWEEN...AND`

谓词:`IN`,`NOT IN`

(1) 字符串匹配

谓词：`[NOT] LIKE ‘<匹配串>’ [ESCAPE ‘ <换码字符>’]`

1. 匹配串为固定字符串：`Sno='12'`或者`Sno LIKE '12'`
2. 匹配串为含通配符的字符串(必须用LIKE):

   `Sname LIKE ‘刘%’` 表示若干个字符，可以为空串

   `Sname LIKE '欧阳_'`表示一个字符
3. 使用换码字符将**通配符**转义为普通字符:

```
--查询DB_Design课程的课程号和学分。
SELECT Cno,  Ccredit
FROM Course
WHERE Cname LIKE 'DB\_Design' ESCAPE '\';
```

```
--查询以"DB_"开头，且倒数第3个字符为 i的课程的详细情况
SELECT *
FROM Course
WHERE Cname LIKE 'DB\_%i_ _' ESCAPE '\';
```

`ESCAPE '\'` 表示“ \” 为换码字符

(2) 涉及空值的查询

- 谓词： IS NULL 或 IS NOT NULL
- **“IS”** 不能用 **“=”** 代替

三、ORDER BY子句

- 可以按一个或多个属性列排序
- 升序： ASC；降序： DESC； 缺省值为升序

当排序列含空值时(+∞)

- ASC：排序列为空值的元组最后显示
- DESC：排序列为空值的元组最先显示

四、聚集函数

计数

`COUNT（[DISTINCT|ALL] *）`

`COUNT（[DISTINCT|ALL] <列名>）`

计算总和`(必须针对数值型)`

`SUM（[DISTINCT|ALL] <列名>）`

计算平均值`(必须针对数值型)`

`AVG（[DISTINCT|ALL] <列名>）`

最大最小值`(可以是数值、字符串或是日期时间)`

`MAX（[DISTINCT|ALL] <列名>）`

`MIN（[DISTINCT|ALL] <列名>）`

除`COUNT(*)`外，其他都跳过空值而只处理非空值

五、GROUP BY子句

细化聚集函数的作用对象：

- 未对查询结果分组， 聚集函数将作用于整个查询结果
- 对查询结果分组后，**聚集函数将分别作用于每个组**

  作用对象是查询的中间结果表
  按指定的**一列或多列值**分组，值相等的为一组

```
--聚集函数 COUNT 将会对组内进行统计
SELECT Cno,COUNT(Sno)
FROM SC
GROUP BY Cno;

--统计“分组内的所有元组”数
SELECT Sno
FROM SC
GROUP BY Sno
HAVING COUNT(*) >3;
```

`HAVING`短语与`WHERE`子句的区别：作用对象不同

- `WHERE`子句作用于基表或视图， 从中选择满足条件的元组
- `HAVING`短语作用于组， 从中选择满足条件的组

重要原则：使用`group by`时， `select`后面的所有属性， 要么出现`group by` 中，要么使用了聚合函数

2.2 连接查询

一、等值与非等值连接查询

等值连接：连接运算符为=

自然连接：**一种特殊的等值连接**

- 两个关系中进行连接的分量必须是**同名的属性组**
- 在结果中把重复的属性列删除

```
SELECT *
FROM Student Natural JOIN SC;
```

二、自身连接

需要给表起别名来以示区分

三、外连接

外连接操作以指定表为**连接主体**，将主体表中不满足连接条件的元组一**并输出**

```
SELECT *
FROM STUDENT LEFT OUTER JOIN SC ON
(STUDENT.Sno=SC.Sno);
```

- 左外连接`LEFT OUTER JOIN`:列出左边关系中所有的元组
- 右外连接`RIGHT OUTER JOIN`:列出右边关系中所有的元组
- 全外连接`FULL OUTER JOIN`:完整外部联接返回左表和右表中的所有

四、复合条件连接

复合条件连接： WHERE子句中含多个连接条件

2.3 嵌套查询

嵌套查询概述

- 一个SELECT-FROM-WHERE语句称为一个查询块
- 将一个查询块嵌套在另一个查询块的`WHERE`子句或`HAVING`短语的条件中的查询称为嵌套查询
- 子查询不能使用`OREDER BY`

有些嵌套查询可以用连接运算替代

一、带有IN谓词的子查询

相关子查询:从外层查询表选出一个元组传给内层查询用于条件判断

不相关子查询

二、带有比较运算符的子查询

当能确切知道内层查询返回单值时，可用比较运算符（>，<，=，>=，<=，!=或<>）

三、带有ANY（SOME）或ALL谓词的子查询

子查询返回多值时，不能直接用比较运算符连接父查询与子查询

```
/*查询其他系中比计算机科学某一学生年龄小的学生姓名和年龄*/
SELECT Sname,  Sage
FROM Student
WHERE Sage < ANY (SELECT Sage
                  FROM Student
                  WHERE Sdept= 'CS')
AND Sdept <> 'CS' ;
```

表3.7 ANY（SOME），ALL谓词与聚集函数、IN谓词的等价转换关系

	=	<>或!=	<	<=	>	>=
ANY	IN	--	<MAX	<=MAX	>MIN	>= MIN
ALL	--	NOT IN	<MIN	<= MIN	>MAX	>= MAX

四、带有EXISTS谓词的子查询

1.EXISTS谓词

存在量词∃

带有EXISTS谓词的子查询不返回任何数据，只产生逻辑真值“true”或逻辑假值“false”。

- 若内层查询结果非空，则外层的WHERE子句返回真值
- 若内层查询结果为空，则外层的WHERE子句返回假值

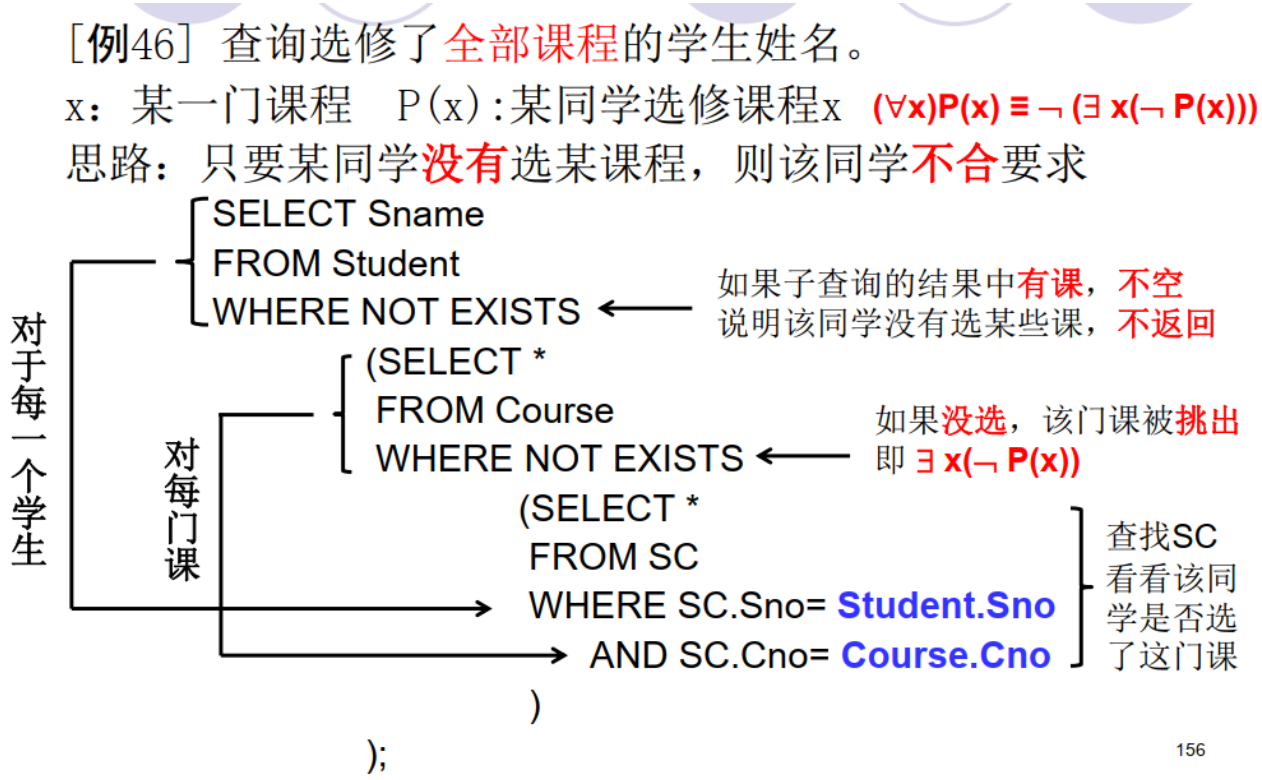
由`EXISTS`引出的子查询，其目标列表达式通常都用*，因为带EXISTS的子查询只返回真值或假值，给出列名无实际意义

2.`NOT EXISTS`谓词

- 若内层查询结果非空，则外层的`WHERE`子句返回假值
- 若内层查询结果为空，则外层的`WHERE`子句返回真值

全称量词、蕴含表达式

$\forall x P(x) \Longleftrightarrow \neg \exists x(\neg P(x))$(结合字面意思)



2.4 集合查询

集合操作：交 `UNIO`、并`INTERSECR`、补`EXCEPT`

- `UNION`：将多个查询结果合并起来时，系统自动去掉重复元组
- `UNION ALL`：将多个查询结果合并起来时，保留重复元组

2.5 基于派生表的查询

子查询还可以出现在`FROM`子句中， 子查询生成临时的派生表(derived table)
派生表成为主查询的查询对象

```
--找出每个学生超过他选修课程平均成绩的课程号。
SELECT SNO, CNO
FROM SC, (SELECT SNO, AVG(GRADE)
          FROM SC GROUP BY SNO)
          AS AVG_SC(AVG_SNO, AVG_GRADE)
WHERE SC.SNO=AVG_SC.AVG_SNO
    AND SC.GRADE>= AVG_SC.AVG_GRADE;
```

如果子查询中没有聚集函数，派生表可以不指定属性列

2.6 Select语句的一般形式

```
SELECT [ALL|DISTINCT]<目标列表达式>[别名] [,<目标列表达式>[别名]] …
FROM <表名或视图名> [别名][,<表名或视图名>[别名]] …
[WHERE<条件表达式>]
[GROUP BY<列名1>
[HAVING<条件表达式>]]
[ORDER BY<列名2>[ASC|DESC]];l
```

3.数据更新

3.1 插入数据

（1）插入一个元组

`INSERT`
`INTO <表名> [(<属性列1>[,<属性列2 >]…)]`
`VALUES (<常量1>[,<常量2>] … )`

`INTO`子句

- 属性列的顺序可与表定义中的顺序不一致
- 没有指定属性列名，则新元组必须在每个列上均有值，且与表定义时属性顺序一致

- 指定部分属性列

`VALUES`子句:提供的值(个数、类型)必须与INTO子句匹配

（2）插入子查询结果（多个元组）

```
INSERT
INTO <表名> [(<属性列1> [,<属性列2>]… )]
子查询;
```

`SELECT`子句目标列(值的个数、类型)必须与`INTO`子句匹配

```
INSERT
INTO Dept_age(Sdept,  Avg_age)
    SELECT Sdept,  AVG(Sage)
    FROM Student
    GROUP BY Sdept;
```

RDBMS在执行插入语句时会检查所插入的数据是否会破坏表上已经定义的完整性规则

3.2 修改数据

```
UPDATE <表名>
SET <列名>=<表达式>[，<列名>=<表达式>]…
[WHERE <条件>];
```

只能修改一张表的数据，但可以同时修改多列的值

`where`缺省时表示要修改表中的所有元组

带子查询的更新语句

```
UPDATE SC
SET Grade=0
WHERE Sno IN
        (SELECT Sno
         FROM Student
         WHERE Sdept='CS');
```

RDBMS在执行修改语句时会检查所修改的数据是否会破坏表上已经定义的完整性规则

3.3删除数据

```
/*删除指定表中满足WHERE子句条件的元组*/
DELETE
FROM <表名>
[WHERE <条件>];
```

`where`缺省表示要删除表中的全部元组， **表的定义仍在字典中**

带子查询的删除语句

```
DELETE
FROM SC
WHERE Sno IN
      (SELECT Sno
       FROM Student
       WHERE Sdept='CS');
```

删除不会破坏**实体完整性和用户定义的完整性**,RDBMS只会检查已定义的**参照完整性**

4.空值的处理

空值：目前不知道、某属性不应该有的值、不便填写的属性的值、不存在或无意义的值，外连接操作也会产生空值

判断是否为空值： `IS NULL, IS NOT NULL`

属性定义约束

- NOT NULL：不能取空值
- UNIQUE： 允许取空值
- PRIMARY KEY： UNIQUE + NOT NULL

空值的运算：

1. 空值与另一个值(包括另一个空值)的算术运算结果为空值

2. 空值与另一个值(包括另一个空值)的比较运算结果为UNKNOWN

3. 增加UNKNOWN后，传统的二值逻辑变成了三值逻辑

4. 在查询中， 只有使WHERE或HAVING子句中逻辑判断为T的元组或分组被选出

x	y	x AND y	x OR y	NOT x
T	T	T	T	F
T	U	U	T	F
T	F	F	T	F
U	T	U	T	U
U	U	U	U	U
U	F	F	U	U
F	T	F	T	T
F	U	F	U	T
F	F	F	F	T

5.视图

视图的特点

- 虚表，是从一个或几个基本表（或视图）导出的表
- **只存放视图的定义，不存放视图对应的数据**
- 基表中的数据发生变化，从视图中查询出的数据也随之改变
- 属于**外模式**的范畴

5.1定义视图

一、建立视图

```
CREATE VIEW
<视图名> [(<列名> [,<列名>]…)]
AS <子查询>
[WITH CHECK OPTION];
```

组成视图的属性列名:全部省略或全部指定
`with check option`: 更新操作时， RDBMS会检查视图定义中的条件，即`where`子句中所列的条件,如果更新操作不满足条件，拒绝执行

以下三种情况下必须指定视图的列名：

1. 由聚集函数或者表达式构成的列
2. 由多表连接选出的几个同名列
3. 需要在视图中启用新名字的列

`RDBMS`执行`CREATE VIEW`语句时只是把视图定义存入数据字典，并不执行其中的`SELECT`语句,在对视图查询时，按视图的定义从基本表中将数据查出

行列子集视图：

- 从单个基本表导出
- 去掉基本表的某些行、列，但保留了主码

二、删除视图

```
DROP VIEW <视图名>[CASCADE];
```

1. 该语句从数据字典中删除指定的视图定义
2. 如果该视图上还导出了其他视图，使用`CASCADE`级联删除语句，把该视图和由它导出的所有视图一起删除
3. 删除基表时，基于此的视图失效。但视图的定义并没有被删除。

5.2 查询视图

用户角度：查询视图与查询基本表相同
RDBMS实现视图查询的方法：视图消解法（View Resolution）

- 进行有效性检查，检查表、视图是否存在
- 转换成等价的对基本表的查询
- 执行修正后的查询

使用视图查询与基于派生表查询有区别

1. 视图一旦定义，永久保存于数据字典；派生表只是临时定义，执行完立即删除
2. 视图的定义可以被所有查询引用；派生表只有所属的语句可以使用

5.3更新视图

通过视图来插入、删除和修改数据
对视图的更新最终要转换为对基本表的更新

- 使用视图消解进行转换
- `with check option`可以起到约束操作作用

```
CREATE VIEW IS_Student
AS
SELECT Sno，  Sname，  Sage
FROM Student
WHERE Sdept= 'IS';
```

对`IS_Student`视图的更新操作为例：

- 修改操作：自动加上`Sdept= ‘IS’`的条件，并且不允许给`Sdept`赋新值
- 删除操作：只能对属于视图的记录删除
- 插入操作：自动检查`Sdept`属性值是否为`'IS'`:如果不是，则拒绝该插入操作;如果没有提供`Sdept`属性值，则自动定义`Sdept`为`'IS'`

更新视图的限制：一些视图是**不可更新**的，因为对这些视图的更新不能唯一地有意义地转换成对相应基本表的更新

```
UPDATE S_G
SET Gavg=90
WHERE Sno= ‘201215121’; /*无法直接修改平均分*/
```