第4章 SQL

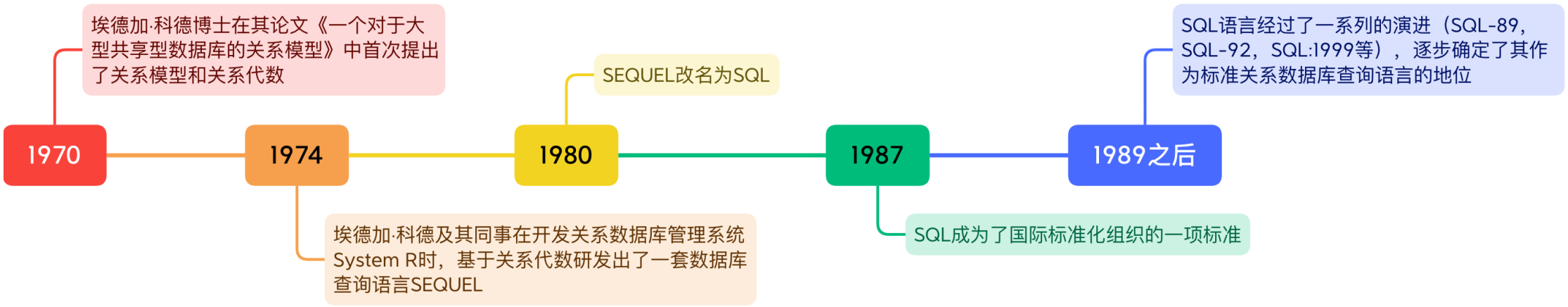
清华大学计算机系数据库组 李国良

liguoliang@tsinghua.edu.cn
http://dbgroup.cs.tsinghua.edu.cn/ligl

SQL查询语言概览

- SQL：Structured Query Language，结构化查询语言
- SQL是用于管理关系型数据库并对其中的数据进行一系列操作（包括数据插入、查询、修改、删除等）的一种语言

- SQL语言发展概览：



SQL查询语言概览

- SQL语言可以分为以下几类：



目录

1. SQL数据定义语言
2. SQL数据操纵语言
3. SQL数据控制语言
4. 存储过程和函数
5. 触发器
6. 使用程序设计语言访问数据库

目录

1. **SQL数据定义语言**
2. SQL数据操纵语言
3. SQL数据控制语言
4. 存储过程和函数
5. 触发器
6. 使用程序设计语言访问数据库

SQL数据定义语言

- SQL数据定义语言包含创建、修改、删除表的语句，创建、修改、删除索引的语句，创建、修改、删除视图的语句。

	创建	修改	删除
表	CREATE TABLE	ALTER TABLE	DROP TABLE
索引	CREATE INDEX	ALTER INDEX	DROP INDEX
视图	CREATE VIEW	ALTER VIEW	DROP VIEW

- 本章继续以第2章介绍的学生选课数据库为例，来介绍SQL。

学生选课数据库

- 包含三个Student、Course、SC三个表（关系）。

Student表

Sno (学号)	Sname (姓名)	Sgender (性别)	Sage (年龄)	Sdept (所在系)
2021310721	李博	男	17	CS
2021310722	赵宇	男	19	CS
2021310723	张敏	女	18	CS
2021310724	王勇	男	18	MA
2021310725	刘佳	女	17	MA

Course表

Cno (课程号)	Cname (课程名)	Cpno (先修课)	Ccredit (学分)
1	数据库	2	4
2	数据结构与算法	6	4
3	操作系统	2	3
4	高等数学		4
5	软件工程	6	2
6	程序设计		3
7	数值分析	4	2

SC表

Sno (学号)	Cno (课程号)	Grade (成绩)
2021310721	5	98
2021310722	1	87
2021310723	1	92
2021310723	5	76
2021310724	7	84
2021310725	4	95

创建表

➢ **在SQL中创建关系模式（表）的语法格式为：**

```
CREATE TABLE <表名> (
    <列名> <数据类型> [列级完整性约束] … [列级完整性约束]
    [,<列名> <数据类型> [列级完整性约束] … [列级完整性约束]]
    …
    [,<列名> <数据类型> [列级完整性约束] … [列级完整性约束]]
    [,表级完整性约束]
    …
    [,表级完整性约束]
);
```

- 中括号内的项为可选项
- 分号标志着一条SQL语句的结束
- SQL语言不区分大小写

➢ **【例4.1】创建学生表的SQL语句如下所示（#后面的内容为注释）：**

```
CREATE TABLE Student(
    Sno CHAR(10) PRIMARY KEY,                    # Sno 为主键
    Sname CHAR(10) NOT NULL,                     # Sname 非空
    Sgender CHAR(2),
    Sage INT,
    Sdept CHAR(20)
);
```

- 也可以将主键作为表级约束 PRIMARY KEY (Sno)
- CONSTRAINT C1 CHECK (Sgender IN ('男', '女'))

表的数据类型

	数据类型	描述
文本型	CHAR(n)	长度为 n 的定长字符串
	VARCHAR(n)	长度为 n 的变长字符串
数字型	INT	整数（4字节）
	SMALLINT	短整数（2字节）
	BIGINT	大整数（8字节）
	FLOAT	单精度浮点数
	DOUBLE	双精度浮点数
	DECIMAL(p, d)	定点数，由 p 位数字组成，小数点后有 d 位数字
	BOOLEAN	布尔型
时间型	DATE	日期，包含年、月、日，格式为YYYY-MM-DD
	TIME	时间，包含时、分、秒，格式为HH:MM:SS
	TIMESTAMP	时间戳，格式为YYYY-MM-DD HH:MM:SS

创建表

- 完整性约束：
  - 非空约束（NOT NULL）：指定某列不可为空
  - 唯一约束（UNIQUE）：指定某列或者几列不能重复
  - 主键约束（PRIMARY KEY）：指定某列或者几列为主键，主键属性必须是非空且唯一的
  - 外键约束（FOREIGN KEY…REFERENCES）：指定某列或者几列为外键，同时需指定外键引用的其他表的主键

- 【例4.2】创建课程表的SQL语句如下所示：

```
CREATE TABLE Course(
    Cno CHAR(4) PRIMARY KEY,                    # Cno 为主键
    Cname CHAR(20) NOT NULL,                    # Cname 非空
    Cpno CHAR(4),                               # Cpno 为先修课
    Ccredit INT,
    FOREIGN KEY (Cpno) REFERENCES Course(Cno)
                # Cpno 为外键，被参照表为 Course 表，被参照列为 Course 表的 Cno 列
);
```

修改表

- ALTER TABLE语句可以用于在现有的表中添加、删除、修改列。
- 使用ALTER TABLE语句添加列的语法格式为：

```
ALTER TABLE <表名> ADD [COLUMN] <列名> <数据类型>;
```

- 【例4.4】在学生表中增加一个电话列（Sphone）的SQL语句为：

```
ALTER TABLE Student ADD COLUMN Sphone CHAR(15);
```

修改表

- ALTER TABLE语句可以用于在现有的表中添加、删除、修改列。
- 使用ALTER TABLE语句删除列的语法格式为：

```
ALTER TABLE <表名> DROP [COLUMN] <列名> [RESTRICT | CASCADE];
```

  - RESTRICT：如果该列被其它列引用，则无法删除该列
  - CASCADE：引用该列的其它列会和该列同时被删除

- 【例4.5】在学生表中删除电话列（Sphone）的SQL语句为：

```
ALTER TABLE Student DROP COLUMN Sphone;
```

修改表

- ALTER TABLE语句可以用于在现有的表中添加、删除、修改列。
- 使用ALTER TABLE语句修改列的语法格式为：

```
ALTER TABLE <表名> ALTER COLUMN <列名> <数据类型>;
```

- 【例4.6】将学生表中的年龄列数据类型改为SMALLINT（原始数据类型为INT）的SQL语句为：

```
ALTER TABLE Student ALTER COLUMN Sage SMALLINT;
```

删除表

- 在SQL中删除一个（或多个）表的语法格式为：

```
DROP TABLE <表名> [,<表名>] … [,<表名>] [RESTRICT | CASCADE];
```

- DROP TABLE语句不仅删除了表中的所有元组，还删除了该表的关系模式。

- 【例4.7】删除学生表的SQL语句为：

```
DROP TABLE Student;
```

- 【例4.8】使用级联选项（CASCADE）删除学生表的SQL语句为：

```
DROP TABLE Student CASCADE;
```

当使用级联选项（CASCADE）删除学生表时，引用学生表的学生选课表也会被删除。

创建索引

- 考虑一个简单的SQL查询，该查询查找计算机系的所有学生信息：

```
SELECT * FROM Student WHERE Sdept="CS";
```

- 逐个扫描所有元组并检查其是否满足条件，效率很低
- 索引可以加速查询操作 → 在Sdept属性上建立索引，可以快速定位到在Sdept属性上取值为CS的元组

- 用户可以在一个表上建立一个或多个索引，以加速查询。
- 数据库中的索引有很多类型，包括B+树索引，散列索引，位图索引等。
- 索引的分类、实现机制等细节请参考第9章。

创建索引

- 在SQL中创建索引的语法格式为：

```
CREATE [UNIQUE] [CLUSTER] INDEX <索引名>
ON <表名> (<列名> [<次序>] [,<列名> [<次序>], …, <列名> [<次序>]]);
```

- **含义：**在ON子句中指定的一列或多列上建立索引，还可以指定某列索引值的排列次序
  - **表名、索引名**：要建索引的表的名字、要建索引的名字
  - **次序**：ASC（升序）或DESC（降序）
  - **UNIQUE**：此索引列的值必须唯一
  - **CLUSTER**：此索引是聚集/聚簇索引（有关索引的内容见第9章）

- 【例4.9】在学生表的Sno属性上建立聚集索引，命名为Sno_index，并说明Sno属性满足唯一约束，索引按照Sno的值递增排序。

```
CREATE UNIQUE CLUSTER INDEX Sno_index ON Student (Sno ASC);
```

- 【例4.10】在学生选课表的Sno和Cno属性上建立唯一索引

```
CREATE UNIQUE INDEX Sno_Cno_index ON SC (Sno, Cno);
```

修改、删除索引

- 可以使用ALTER INDEX对已经建立的索引进行重命名。语法格式为：

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

- 【例4.11】将学生表的Sno_index重命名为Sno_ind。

```
ALTER INDEX Sno_index RENAME TO Sno_ind;
```

- 在SQL中删除索引的语法格式为：

```
DROP INDEX <索引名>;
```

- 【例4.12】删除学生表的Sno_index索引。

```
DROP INDEX Sno_index;
```

创建视图

- 视图是数据库中的一个查询的查询结果构成的“虚关系”。
- 将重复执行的SQL查询存储在数据库中，并对其命名，在之后的查询中可通过名字进行查询，因此可以减少重复工作
- 一个视图就是一个命名的存储在数据库中的查询，视图中的数据即为该查询对应的查询结果

- 在SQL中创建视图的命令为：

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

  - 子查询：任何SELECT语句

创建视图

- 【例4.13】建立计算机系学生的视图，命名为Student_CS。

```
CREATE VIEW Student_CS
AS
SELECT *
FROM Student
WHERE Sdept="CS";
```

  - 在执行CREATE VIEW语句时并不执行其中的子查询语句，只是把视图的定义存入数据字典。
  - 在建立完视图后，就可以像对表一样对视图进行查询了。

- 【例4.14】查询计算机系所有年龄大于等于18的学生的信息。

```
SELECT * FROM Student_CS WHERE Sage>=18;
```

  - 只有在对视图进行查询时，才会按照视图的定义去查询数据。

修改视图

- 在SQL中修改视图的命令为：

```
ALTER VIEW <视图名> AS <子查询>;
```

- 【例4.15】修改Student_CS视图，使其不包含Sdept列。

```
ALTER VIEW Student_CS
AS
SELECT Sno, Sname, Sgender, Sage
FROM Student
WHERE Sdept="CS";
```

删除视图

- 在SQL中删除视图的命令为：

```
DROP VIEW <视图名> [CASCADE];
```

  - 该语句将把指定的视图定义从数据字典中删除。
  - CASCADE：该视图和由该视图导出的视图都会被删除

- 【例4.16】删除Student_CS视图。

```
DROP VIEW Student_CS;
```

物化视图

- 视图是一种“虚关系”，实际查询时需要根据定义查询底层关系，当存在大量这样的查询时会有较高的成本。
- 某些数据库支持物化视图，像存储表一样将创建的视图关系“物化”存储在数据库中。
- 物化视图的创建、修改与删除语法同视图类似，区别是多了关键字 MATERIALIZED。

- 例如，在SQL中创建物化视图的命令为：

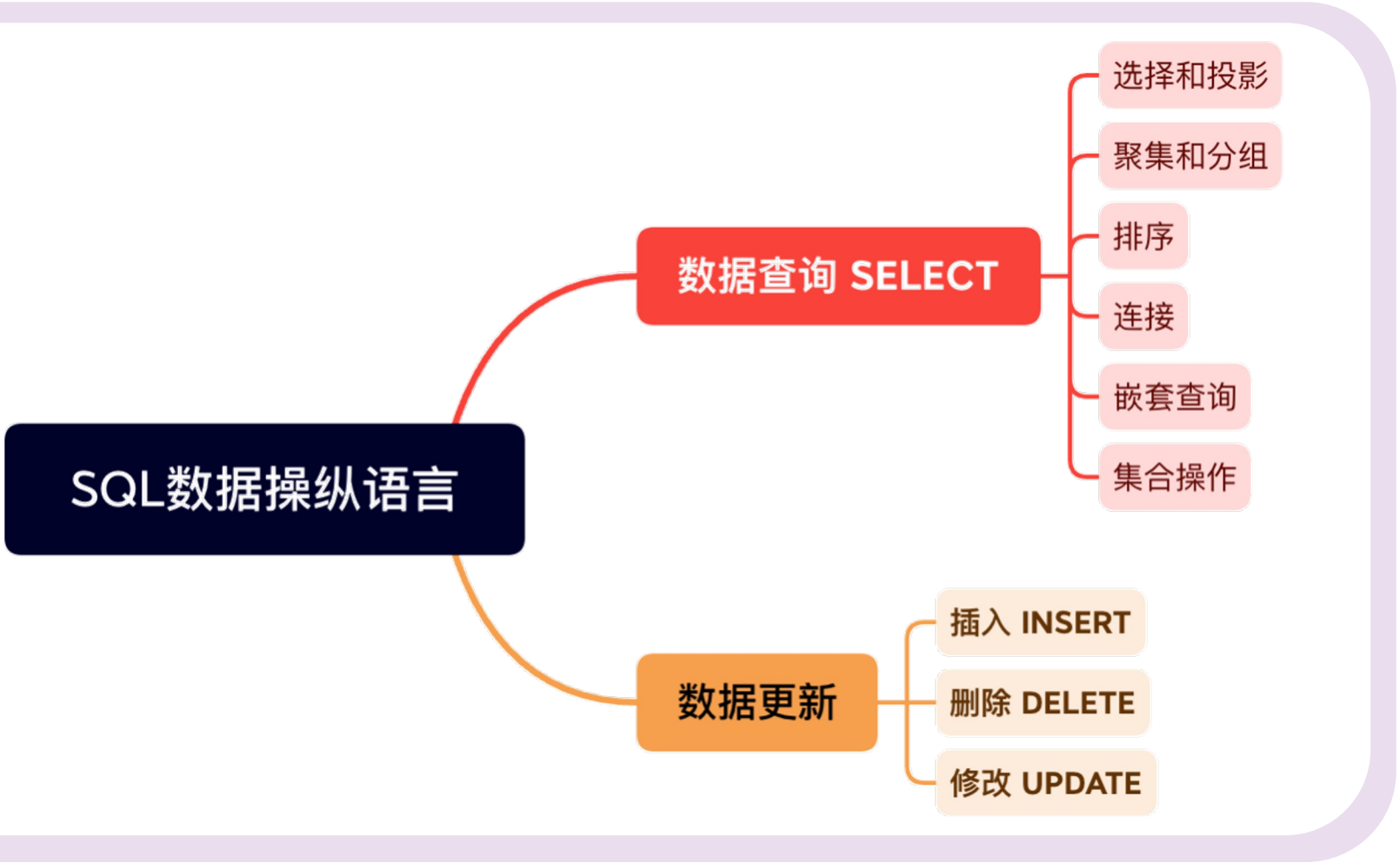
```
CREATE MATERIALIZED VIEW <物化视图名> [(<列名> [,<列名>, …, <列名>])] AS <子查询>;
```

目录

1. SQL数据定义语言
2. **SQL数据操纵语言**
3. SQL数据控制语言
4. 存储过程和函数
5. 触发器
6. 使用程序设计语言访问数据库

SQL数据操纵语言

- SQL数据操纵语言包括数据的查询、插入、删除、修改。



数据查询

- 数据查询是从数据库中获取满足一定条件的数据，是数据库的核心操作。利用SQL进行数据查询的基本语法格式为（其中每一行为一个子句）：

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

- SELECT子句：指定要显示的属性列
- FROM子句：指定从哪些表或者视图中查询数据
- WHERE子句：筛选满足条件的元组
- GROUP BY子句：指定按照哪些列对元组进行分组
- HAVING子句：对分组结果进行筛选
- ORDER BY子句：将查询结果按照指定顺序进行排序

选择和投影

- 选择操作可以选择表中的若干元组，通过WHERE子句实现
- 投影操作可以选择表中的若干列，通过SELECT子句实现
- 将选择操作和投影操作组合，可以组成一个简单的在单表上查询的SQL语句（查询表通过FROM子句指定）

- 【例4.17】查询计算机系学生的学号、姓名和年龄。

```
SELECT Sno, Sname, Sage
FROM Student
WHERE Sdept= “CS” ;
```

- 查询结果为：

Sno	Sname	Sage
2021310721	李博	17
2021310722	赵宇	19
2021310723	张敏	18

投影操作

- 投影操作可以选择表中的若干列，主要体现在SELECT子句后面的列表达式
- 列表达式：
  - 查询指定列：在SELECT子句中依次列出要查找的列的名字
  - 查询全部列：用符号*来代替所有列
  - 查询经过计算的值：对列进行计算后的表达式

- 【例4.18】查询所有学生的学号、姓名、所在系和年龄。

```
SELECT Sno, Sname, Sdept, Sage
FROM Student;
```

Sno	Sname	Sdept	Sage
2021310721	李博	CS	17
2021310722	赵宇	CS	19
2021310723	张敏	CS	18
2021310724	王勇	MA	18
2021310725	刘佳	MA	17

投影操作

- 【例4.19】查询所有学生的学号、姓名、所在系和年龄。

```
SELECT *
FROM Student;
```




```
SELECT Sno, Sname, Sgender, Sage, Sdept
FROM Student;
```

- 上述两条SQL均会输出所有学生信息（即原始学生表）。

- 【例4.20】查询所有学生的学号，姓名，出生年份。

```
SELECT Sno, Sname, 2021-Sage
FROM Student;
```

- 查询结果为：

Sno	Sname	2021-Sage
2021310721	李博	2004
2021310722	赵宇	2002
2021310723	张敏	2003
2021310724	王勇	2003
2021310725	刘佳	2004

投影操作

- 【例4.21】查询所有学生的姓名、出生年份和所在系，要求系名用小写字母表示。

```
SELECT Sname, 2021-Sage, LOWER(Sdept)
FROM Student;
```

- 其中LOWER(s)为一个函数，返回字符串s的小写表示。查询结果为：

Sname	2021-Sage	LOWER(Sdept)
李博	2004	cs
赵宇	2002	cs
张敏	2003	cs
王勇	2003	ma
刘佳	2004	ma

NAME	BIRTHDAY	DEPARTMENT
李博	2004	cs
赵宇	2002	cs
张敏	2003	cs
王勇	2003	ma
刘佳	2004	ma

- 用户可以通过AS指定列别名来改变查询结果的列标题。

```
SELECT Sname AS NAME, 2021-Sage AS BIRTHDAY, LOWER(Sdept) AS DEPARTMENT
FROM Student;
```

选择操作

- 选择操作通过WHERE子句选择若干满足条件的元组
- WHERE子句后面跟着一个条件表达式，满足该条件表达式的元组会被返回
- WHERE子句常用查询条件：

查询条件	谓词
比较	=，>，<，>=，<=，!=，<>，!>，!<，NOT+上述比较符号
确定范围	BETWEEN AND，NOT BETWEEN AND
确定集合	IN，NOT IN
空值	IS NULL，IS NOT NULL
逻辑运算	AND，OR
字符串运算	LIKE，NOT LIKE，%，_，ESCAPE

like '%#%%' escape '#'
#字符后面第一个字符被认为是普通字符

选择操作

- 【例4.22】查询年龄在18岁及以上学生的学号、姓名和年龄。

```
SELECT Sno, Sname, Sage
FROM Student
WHERE Sage>=18;
```

- 查询结果为：

Sno	Sname	Sage
2021310722	赵宇	19
2021310723	张敏	18
2021310724	王勇	18

- 【例4.23】查询分数在80到90之间的学生学号、课程号以及分数。

```
SELECT Sno, Cno, Grade
FROM SC
WHERE Grade BETWEEN 80 and 90;
```

- 查询结果为：

Sno	Cno	Grade
2021310722	1	87
2021310724	7	84

选择操作

- 【例4.24】查询计算机系、数学系的学生学号、姓名以及所在系。

```
SELECT Sno, Sname, Sdept
FROM Student
WHERE Sdept in ("CS", "MA");
```

- 查询结果为：

Sno	Sname	Sdept
2021310721	李博	CS
2021310722	赵宇	CS
2021310723	张敏	CS
2021310724	王勇	MA
2021310725	刘佳	MA

- 【例4.26】查询不需要先修课的课程，输出课程号、课程名及其学分。

```
SELECT Cno, Cname, Ccredit
FROM Course
WHERE Cpno IS NULL;
```

- 查询结果为：

Cno	Cname	Ccredit
4	高等数学	4
6	程序设计	3

选择操作

- 【例4.27】查询计算机系18岁的学生学号、姓名、性别、年龄及所在系。

```
SELECT Sno, Sname, Sgender, Sage, Sdept
FROM Student
WHERE Sdept="CS" AND Sage=18;
```

- 查询结果为：

Sno	Sname	Sgender	Sage	Sdept
2021310723	张敏	女	18	CS

- 【例4.28】查询计算机系的学生或者18岁的学生学号、姓名、性别、年龄及所在系。

```
SELECT Sno, Sname, Sgender, Sage, Sdept
FROM Student
WHERE Sdept="CS" OR Sage=18;
```

- 查询结果为：

Sno	Sname	Sgender	Sage	Sdept
2021310721	李博	男	17	CS
2021310722	赵宇	男	19	CS
2021310723	张敏	女	18	CS
2021310724	王勇	男	18	MA

选择操作

- LIKE可以用来查询与匹配串匹配的字符串。其语法格式为：

```
[NOT] LIKE “<匹配串>” [ESCAPE “<换码字符>”]
```

  - 匹配串一般由字符和通配符组成，通配符包括_和%
  - _：匹配字符串中的任意一个字符
  - %：匹配字符串中的任意多个字符（包括 0 个和 1 个）
  - 当字符串的所有字符均可匹配成功时，该字符串的 LIKE 查询结果为真
  - 当匹配串中本身就包含通配符_或%时，需要在匹配串中包含的_或%前面加上转义字符来进行转义，同时需要使用 ESCAPE “<换码字符>” 来指定换码字符

- 【例4.29】查询所有姓王的且名字包含两个字的学生的学号、姓名、性别和所在系。

```
SELECT Sno, Sname, Sgender, Sdept
FROM Student
WHERE Sname LIKE “王_”;
```

- 查询结果为：

Sno	Sname	Sgender	Sdept
2021310724	王勇	男	MA

选择操作

- 【例4.30】查询课程名中包含“数据”的课程，输出其课程号、课程名及其学分。

```
SELECT Cno, Cname, Ccredit
FROM Course
WHERE Cname LIKE "%数据%";
```

- 查询结果为：

Cno	Cname	Ccredit
1	数据库	4
2	数据结构与算法	4

- 【例4.31】查询“数据库”课程的课程号和学分。

```
SELECT Cno, Ccredit
FROM Course
WHERE Cname LIKE "数据库";
```

查询结果：

Cno	Ccredit
1	4

- 【例4.32】查询以“数据”开头，且倒数第三个字为“与”的课程，输出课程名。

```
SELECT Cname
FROM Course
WHERE Cname LIKE "数据%与_ _";
```

查询结果：

Cname
数据结构与算法

选择操作

- DISTINCT：查询结果会消除取值重复的行

- 【例4.33】查询所有系的信息。

```
SELECT Sdept
FROM Student;
```

- 查询结果为：

Sdept
CS
CS
CS
MA
MA

- 【例4.34】查询所有系的信息（指定DISTINCT选项）。

```
SELECT DISTINCT Sdept
FROM Student;
```

- 查询结果为：

Sdept
CS
MA

聚集操作

- 为了查询一些数据聚集后的结果，比如查询一些统计值（平均值、最大值、最小值等），SQL提供了许多聚集函数。
- 常见的聚集函数包括：

聚集函数	含义
COUNT([DISTINCT | ALL *])	统计元组个数
COUNT([DISTINCT | ALL] <列名>)	统计一列值的个数
SUM([DISTINCT | ALL] <列名>)	统计一列值的总和
AVG([DISTINCT | ALL] <列名>)	统计一列值的平均值
MAX([DISTINCT | ALL] <列名>)	统计一列值的最大值
MIN([DISTINCT | ALL] <列名>)	统计一列值的最小值

- 当指定DISTINCT选项时，聚集函数要应用在消除重复取值的列上

聚集操作

- 【例4.35】查询学生总人数。

```
SELECT COUNT(*)
FROM Student;
```

查询结果：

COUNT(*)
5

- 【例4.36】查询选修了课程的学生人数。

```
SELECT COUNT(Sno)
FROM SC;
```

查询结果：

COUNT(Sno)
6

- 【例4.37】查询选修了课程的学生人数（指定DISTINCT选项）。

```
SELECT COUNT(DISTINCT Sno)
FROM SC;
```

查询结果：

COUNT(DISTINCT Sno)
5

- 【例4.38】查询选修1号课程的学生的平均成绩。查询结果为$(87+92)\div 2=89.5$。

```
SELECT AVG(Grade)
FROM SC
WHERE Cno="1";
```

查询结果：

AVG(Grade)
89.5

分组操作

- GROUP BY子句可以将查询到的满足条件的元组按照某一列或多列的值进行分组，值相等的为一组。
- 将学生选课表按照课程号列进行分组：

组	Sno	Cno	Grade
Cno=“1”的组	2021310722	1	87
	2021310723	1	92
Cno=“5”的组	2021310721	5	98
	2021310723	5	76
Cno=“7”的组	2021310724	7	84
Cno=“4”的组	2021310725	4	95

分组操作

- 未分组：聚集函数作用于整个查询结果
- 分组：聚集函数分别作用于每个组

- 【例4.39】查询各个课程的选课人数。

```
SELECT Cno, COUNT(*)
FROM SC
GROUP BY Cno;
```

- 查询结果为：

Cno	COUNT(*)
1	2
5	2
7	1
4	1

组	Sno	Cno	Grade
Cno=“1”的组	2021310722	1	87
	2021310723	1	92
Cno=“5”的组	2021310721	5	98
	2021310723	5	76
Cno=“7”的组	2021310724	7	84
Cno=“4”的组	2021310725	4	95

分组操作

➢ 【例4.40】查询所有选课学生的学号及平均分。

```
SELECT Sno, AVG(Grade)
FROM SC
GROUP BY Sno;
```

查询结果：

Sno	AVG(Grade)
2021310721	98
2021310722	87
2021310723	84
2021310724	84
2021310725	95

➢ 【例4.41】查询平均分超过90的学生的学号和平均成绩。

```
SELECT Sno, AVG(Grade)
FROM SC
GROUP BY Sno
HAVING AVG(Grade)>90;
```

查询结果：

Sno	AVG(Grade)
2021310721	98
2021310725	95

➢ HAVING子句用来对GROUP BY后得到的组进行筛选

分组操作

- 【例4.42】查询各个系的男生和女生人数。

```
SELECT Sdept, Sgender, COUNT(*)
FROM Student
GROUP BY Sdept, Sgender;
```

查询结果：

Sdept	Sgender	COUNT(*)
CS	男	2
CS	女	1
MA	男	1
MA	女	1

- 任何没有出现在GROUP BY子句中的列如果出现在SELECT子句中的话，那么它一定是出现在聚集函数内部。

- 【例4.43】下述SQL是错误的。因为Sname未出现在GROUP BY子句中，却出现在了SELECT子句中，且不是出现在聚集函数内部。

```
SELECT Sdept, Sgender, Sname, COUNT(*), AVG(Sage)
FROM Student
GROUP BY Sdept, Sgender;
```

排序操作

- ORDER BY子句将查询结果按照某一列（或者某多列）的值进行排序
  - DESC：降序排序；ASC：升序排序（默认选项）
- 【例4.44】查询选修1号课程的学生学号与成绩，并将查询结果按照成绩降序排序。

```
SELECT Sno, Grade
FROM SC
WHERE Cno="1"
ORDER BY Grade DESC;
```

查询结果：

Sno	Grade
2021310723	92
2021310722	87

- 【例4.45】查询所有选课学生的学号及平均分，并将查询结果按照平均分降序排序，平均分相同的学生按照学号升序排序。

```
SELECT Sno, AVG(Grade)
FROM SC
GROUP BY Sno
ORDER BY AVG(Grade) DESC, Sno ASC;
```

查询结果：

Sno	AVG(Grade)
2021310721	98
2021310725	95
2021310722	87
2021310723	84
2021310724	84

连接操作

- 同时涉及多个（两个以及上）表的查询称为**连接查询**。
- 在连接操作中，需在FROM子句中指定需要连接的表，在WHERE子句中指定**连接条件**。
- **连接条件**的常见格式为：
  - [<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>
  - [<表名1>.]<列名1> BETWEEN [<表名2>.]<列名2> AND [<表名2>.]<列名3>
- 【例4.46】查询所有选课学生的学号、姓名、课程号与成绩。

```
SELECT Student.*, SC.*
FROM Student, SC
WHERE Student.Sno=SC.Sno;
```

Student.Sno	Sname	Sgender	Sage	Sdept	SC.Sno	Cno	Grade
2021310721	李博	男	17	CS	2021310721	5	98
2021310722	赵宇	男	19	CS	2021310722	1	87
2021310723	张敏	女	18	CS	2021310723	1	92
2021310723	张敏	女	18	CS	2021310723	5	76
2021310724	王勇	男	18	MA	2021310724	7	84
2021310725	刘佳	女	17	MA	2021310725	4	95

- 当连接符号为=时，称为**等值连接**。例4.46为等值连接。
- 在等值连接中把完全相同的列（比如Student.Sno和SC.Sno）去掉一列为**自然连接**。

连接操作

- 【例4.47】使用自然连接查询所有选课学生的学号、姓名、课程号与成绩。

```
SELECT Student.Sno, Sname, Sgender, Sage, Sdept, Cno, Grade
FROM Student, SC
WHERE Student.Sno=SC.Sno;
```

- 查询结果同例4.46（但是相同的列Sno只保存一份）

- 【例4.48】查询选修1号课程且成绩在90分以上的学生学号、姓名、所在系及成绩。

```
SELECT Student.Sno, Sname, Sdept, Grade
FROM Student, SC
WHERE Student.Sno=SC.Sno AND Cno="1" AND Grade>90;
```

Student.Sno	Sname	Sdept	Grade
2021310723	张敏	CS	92

  - Student.Sno=SC.Sno为连接条件
  - Cno=“1” AND Grade>90为选择条件

连接操作

- 【例4.49】查询所有课程及其先修课。输出课程名及先修课的课程名。

```
SELECT A.Cname AS Cname, B.Cname AS Cpno_name
FROM Course AS A, Course AS B
WHERE A.Cpno=B.Cno;
```

Cname	Cpno_name
数据库	数据结构与算法
数据结构与算法	程序设计
操作系统	数据结构与算法
软件工程	程序设计
数值分析	高等数学

- 连接操作的两个表是同一个表时，为了区分，需在FROM子句中使用AS重命名表。
- 【例4.51】查询每个学生选课的总学分，输出学生学号、姓名、总学分，将结果按照总学分降序排序，总学分相同的学生按照学号升序排序。

```
SELECT Student.Sno, Sname, SUM(Ccredit)
FROM Student, Course, SC
WHERE Student.Sno=SC.Sno AND Course.Cno=SC.Cno
GROUP BY Student.Sno, Sname
ORDER BY SUM(Ccredit) DESC, Student.Sno ASC;
```

Student.Sno	Sname	SUM(Ccredit)
2021310723	张敏	6
2021310722	赵宇	4
2021310725	刘佳	4
2021310721	李博	2
2021310724	王勇	2

连接操作

连接的三种写法

1. SELECT Student.Sno, Course.Cno, Grade
   FROM Student, Course, SC
   WHERE Student.Sno = SC.Sno AND Course. Cno = SC.Cno AND Sname= “李博”

2. SELECT Student.Sno, Course.Cno, Grade
   FROM ((Student
   INNER JOIN SC on Student.Sno = SC.Sno)
   INNER JOIN Course ON Course. Cno = SC.Cno)
   WHERE Sname= “李博”

3. SELECT Student.Sno, Course.Cno, Grade
   FROM ((Student
   INNER JOIN SC USING(Sno)) //同名字段
   INNER JOIN Course USING(Cno))
   WHERE Sname= “李博”

- 内连接：join 或 inner join
- 左连接：left join 或 left outer join
- 右连接：right join 或 right outer join
- 外连接：full join 或 full outer join
- 笛卡尔积：cross join

外连接操作

- 外连接是连接运算的扩展，用来处理缺失值。
  - 左外连接取出左表中所有与右表中的任一元组都不匹配的元组，用空值填充所有右表中的属性，再将所产生的元组加到左表和右表自然连接的结果中。
  - 右外连接取出右表中所有与左表中的任一元组都不匹配的元组，用空值填充所有左表中的属性，再将所产生的元组加到右表和左表自然连接的结果中。
  - 全外连接的查询结果是左外连接和右外连接查询结果的并集。
- 【例4.52】将课程表与学生选课表进行左外连接。

```
SELECT Course.Cno, Cname, Cpno, Ccredit, Sno, Grade
FROM Course LEFT OUTER JOIN SC ON (Course.Cno=SC.Cno);
```

Course.Cno	Cname	Cpno	Ccredit	Sno	Grade
1	数据库	2	4	2021310722	87
1	数据库	2	4	2021310723	92
2	数据结构与算法	6	4	NULL	NULL
3	操作系统	2	3	NULL	NULL
4	高等数学		4	2021310725	95
5	软件工程	6	2	2021310721	98
5	软件工程	6	2	2021310723	76
6	程序设计		3	NULL	NULL
7	数值分析	4	2	2021310724	84

嵌套查询

- 一个SELECT-FROM-WHERE语句是一个查询块
- 将一个查询块嵌套在另一个查询块的WHERE子句，FROM子句或者HAVING短语的条件中的查询称为嵌套查询
- 上层的查询称为外层查询或者父查询，下层的查询称为内层查询或者子查询
- 在求解嵌套查询时，先求解子查询，然后基于子查询的求解结果来求解父查询

【例4.53】查询选修了1号课程的学生学号、姓名及所在系。

```
SELECT Sno, Sname, Sdept
FROM Student
WHERE Sno IN
      (SELECT Sno
      FROM SC
      WHERE Cno="1");
```

Sno
2021310722
2021310723

内层查询的查询结果

Sno	Sname	Sdept
2021310722	赵宇	CS
2021310723	张敏	CS

外层查询的查询结果

嵌套查询

- 有些嵌套查询可以用连接操作实现

- 【例4.54】例4.51中的SQL语句可以重写为：

```
SELECT Student.Sno, Sname, Sdept
FROM Student, SC
WHERE Student.Sno=SC.Sno AND Cno="1";
```

- 【例4.55】查询选修数据库课程的学生的学号。

```
SELECT Sno
FROM SC
WHERE Cno=
      (SELECT Cno
      FROM Course
      WHERE Cname="数据库");
```

Sno
2021310722
2021310723

- IN 等价于 =ANY; NOT IN 等价于 != ALL
- > ANY 大于任一个； > ALL 大于所有

- 当内层查询的查询结果是一个值（不是集合）时，使用=,>,<,!=等比较运算符
- 当内层查询的查询结果是一个集合时，使用IN、NOT INT、ANY/SOME、ALL

嵌套查询

- 子查询的查询条件不依赖于父查询的查询为不相关子查询（例4.52，例4.54）
- 子查询的查询条件依赖于父查询的某个属性值的查询为相关子查询（例4.56）

- 【例4.57】查询各院系中年龄小于平均年龄的学生信息。

```
SELECT DINSTINCT Sdept, Sno, Sname, Sgender, Sage
FROM Student AS A
WHERE Sage<
    (SELECT AVG(Sage)
    FROM Student AS B
    WHERE B.Sdept=A.Sdept);
```

- 查询结果为：

Sdept	Sno	Sname	Sgender	Sage
CS	2021310721	李博	男	17
MA	2021310725	刘佳	女	17

集合操作

- SQL语句查询的结果是元组的集合，因此可以将查询结果进行集合操作
- SQL中支持的集合操作主要包括并（UNION）、交（INTERSECT）、差（EXCEPT）
- 参加集合操作的各查询结果列数必须相同，对应的数据类型也需要相同

- 并操作：【例4.60】查询计算机系学生或男生的信息。

```
(SELECT *
FROM Student
WHERE Sdept="CS")
UNION
(SELECT *
FROM Student
WHERE Sgender="男");
```

Sno	Sname	Sgender	Sage	Sdept
2021310721	李博	男	17	CS
2021310722	赵宇	男	19	CS
2021310723	张敏	女	18	CS
2021310724	王勇	男	18	MA

- Union/Intersect/Except：去重； Union/Intersect/Except ALL：不去重

集合操作

- 交操作：【例4.61】查询计算机系男生的信息。

```
(SELECT *
FROM Student
WHERE Sdept="CS")
INTERSECT
(SELECT *
FROM Student
WHERE Sgender="男");
```

Sno	Sname	Sgender	Sage	Sdept
2021310721	李博	男	17	CS
2021310722	赵宇	男	19	CS

- 差操作：【例4.62】查询计算机系女生的信息。

```
(SELECT *
FROM Student
WHERE Sdept="CS")
EXCEPT
(SELECT *
FROM Student
WHERE Sgender="男");
```

Sno	Sname	Sgender	Sage	Sdept
2021310723	张敏	女	18	CS

关系代数与SQL的转换

- SQL是一种结构化的、便于计算机所理解的查询语言。
- 关系代数是关系型数据库理论的一部分，是SQL的基础：SQL在执行时需要先转为等价的关系代数表达式。
- 关系代数和SQL之间可以进行相互转换：

关系代数运算	对应的SQL语句	关系代数运算	对应的SQL语句
选择运算（σ）	WHERE	连接运算（$\bowtie$）	JOIN
投影运算（Π）	SELECT	赋值运算（$\leftarrow$）	AS
并运算（$\cup$）	UNION	除运算（$\div$）	NOT EXISTS
差运算（$-$）	EXCEPT	去重运算（δ）	DISTINCT
笛卡尔积运算（$\times$）	FROM	广义投影运算（Π）	SELECT
重命名运算（ρ）	AS	聚集运算（$\mathcal{G}$）	聚集函数
交运算（$\cap$）	INTERSECT	分组运算（$\mathcal{G}$）	GROUP BY

数据更新

- 插入数据
  - 插入单个元组
  - 插入子查询结果
- 删除数据
- 修改数据

- 在执行插入、修改和删除语句时会检查所插入、修改和删除的元组是否破坏表中的完整性约束。如果不满足完整性约束，则可能会执行失败！

插入数据

➢ 插入单个元组的语法格式为：

```
INSERT
INTO <表名> [(<列名 1> [,<列名 2>, …, <列名 n>])]
VALUES (<常量 1> [,<常量 2>, …, <常量 n>]);
```

- 该语句的含义为将一个新元组插入到指定表中
  - ◆该元组的列名1，列名2，…，列名n这n个列的取值分别为常量1，常量2，…，常量n
  - ◆对于没有在INTO子句中出现的列，新元组在这些列上的取值为空值

➢ 【例4.64】将新元组（学号：2021310726，姓名：李丽，年龄：17，性别：女，所在系：MA）插入到Student表中。

```
INSERT
INTO Student (Sno, Sname, Sage, Sgender, Sdept
VALUES ("2021310726","李丽", 17,"女","MA");
```

或

```
INSERT
INTO Student
VALUES ("2021310726","李丽","女", 17,"MA");
```

➢ INTO子句中的列名可以省略，那么VALUES子句中则要按照表中列的顺序依次列出所有列的取值。

插入数据

- 插入子查询结果的语法格式为：

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

  - 该语句的含义为将子查询的查询结果插入到指定表的指定列中

- 【例4.65】对每一个系，求学生总人数，并将结果存入数据库。
  - 首先，建立一个系（Department）表，该表包含2列：系名（Dname）列，数据类型为CHAR(20)，和人数（Dnum）列，数据类型为INT：

```
CREATE TABLE Department(
    Dname CHAR(20),
    Dnum INT
);
```

  - 然后，从学生表中查询每个系的学生总人数，并将结果存在系表中。

```
INSERT
INTO Department (Dname, Dnum)
SELECT Sdept, COUNT(*)
FROM Student
GROUP BY Sdept;
```

修改数据

- 在数据库中修改数据的语法格式为：

```
UPDATE <表名>
SET <列名 1>=<表达式 1> [,<列名 2>=<表达式 2>, …, <列名 n>=<表达式 n>]
[WHERE <条件>];
```

  - 该语句的含义为更新指定表中满足WHERE子句中条件的元组
    - ◆将这些元组的列名1，列名2，…，列名n这n个列的取值分别设置为表达式1，表达式2，…，表达式n的取值
    - ◆如果不包含WHERE子句，则表中的所有元组都将被修改

- 【例4.66】将学号为2021310721的学生的年龄加一。

```
UPDATE Student
SET Sage=Sage+1
WHERE Sno="2021310721";
```

- 【例4.67】将所有学生的年龄加一。

```
UPDATE Student
SET Sage=Sage+1;
```

删除数据

- 在数据库中删除数据的语法格式为：

```
DELETE
FROM <表名>
[WHERE <条件>];
```

  - 该语句的含义为删除指定表中满足WHERE子句中条件的元组
    - ◆如果不包含WHERE子句，则表中的所有元组都将被删除

- 【例4.68】删除学号为2021310721的学生记录。

```
DELETE
FROM Student
WHERE Sno="2021310721";
```

- 【例4.69】删除学生选课表中的所有选课信息。

```
DELETE
FROM Student;
```

目录

1. SQL数据定义语言
2. SQL数据操纵语言
3. **SQL数据控制语言**
4. 存储过程和函数
5. 触发器
6. 使用程序设计语言访问数据库

SQL数据控制语言

- SQL数据控制语言：
  - 权限授予：用于向用户授予数据库操作权限
  - 权限收回：用于向用户收回数据库操作权限

- 权限授予（GRANT）语句的语法格式为：

```
GRANT <权限>[,<权限>,…,<权限>]
ON <对象类型><对象名>[,<对象类型><对象名>,…, <对象类型><对象名>]
TO <用户名>[,<用户名>,…,<用户名>]
[WITH GRANT OPTION];
```

- 该语句的含义为将某些对象（由ON子句指定）的某些操作权限（由GRANT子句指定）授予给某些用户（由TO子句指定）
  - ◆WITH GRANT OPTION：被授权用户可以把该GRANT语句授予他的权限再授予给其他用户
  - ◆权限：查询数据权限（SELECT），插入新数据权限（INSERT），更新数据权限（UPDATE），删除数据权限（DELETE）
  - ◆可以指定GRANT语句的用户：数据库管理员，数据库的创建者，拥有GRANT子句中指定权限的用户。

SQL数据控制语言

- 【例4.70】把查询学生表和在学生表上插入数据的权限授予给用户Alice。

```
GRANT SELECT, INSERT
ON TABLE Student
TO Alice;
```

- 【例4.71】把学生表和课程表的所有权限（包括SELECT，INSERT，UPDATE和DELETE的权限）授予给用户Alice。

```
GRANT ALL PRIVILEGES
ON TABLE Student, TABLE Course
TO Alice;
```

- 【例4.72】把查询学生表的权限授予给所有用户。

```
GRANT SELECT
ON TABLE Student
TO PUBLIC;
```

SQL数据控制语言

- 权限收回（REVOKE）语句的语法格式为：

```
REVOKE <权限>[,<权限>,…,<权限>]
ON <对象类型><对象名>[,<对象类型><对象名>,…, <对象类型><对象名>]
FROM <用户名>[,<用户名>,…,<用户名>]
[CASCADE|RESTRICT];
```

  - 该语句的含义为收回某些用户（由FROM子句指定）对某些对象（由ON子句指定）的某些操作权限（由REVOKE子句指定）
    - ◆CASCADE：支持级联收回，即由这些用户授予了以上权限的用户的这些权限也会被收回（默认选项）
    - ◆RESTRICT：不支持级联收回

- 【例4.73】收回用户Alice查询学生表和在学生表上插入数据的权限。

```
REVOKE SELECT, INSERT
ON TABLE Student
FROM Alice;
```

目录

1. SQL数据定义语言
2. SQL数据操纵语言
3. SQL数据控制语言
4. **存储过程和函数**
5. 触发器
6. 使用程序设计语言访问数据库

存储过程和函数

- 存储过程和函数是事先经过编译并存储在数据库中的一段SQL语句的集合
- 存储过程和函数可以对一段代码进行封装，以便日后调用
- 数据库中创建存储过程的语句为CREATE PROCEDURE，并通过CALL语句加存储过程名来调用存储过程
- 数据库中创建函数的语句为CREATE FUNCTION，并通过函数名来调用函数

- 存储过程优势：
  - 降低网络开销（存储到DB中，不频繁网络交互）
  - 安全性高
  - 性能高，编译一次多次使用

- 存储过程劣势：
  - 开发调试难
  - 可移植性差
  - 难以支持分布式数据库

创建和调用存储过程

➢ 创建存储过程的语法格式为：

```
CREATE PROCEDURE <存储过程名>([参数,…,参数])
BEGIN
<SQL 语句块>
END <终止符>
```

- 括号内指定参数列表（也可以没有参数，但括号不可省略）
- BEGIN…END之间是该存储过程要执行的SQL语句
- 终止符标志着存储过程的结束
- 参数的格式为： [IN|OUT|INOUT] 参数名 参数数据类型
  - ◆IN表示该参数为输入参数，OUT表示该参数为输出参数，INOUT表示该参数既可以输入也可以输出
  - ◆参数名指定该参数的名称
  - ◆参数数据类型指定该参数的数据类型，比如INT，FLOAT，VARCHAR(n)，DATE等

➢ 调用存储过程的SQL语句为：

```
CALL <存储过程名()>;
```

创建和调用存储过程

- 【例4.75】创建名为PROC_COURSE_GRADE的存储过程，输出某门课程的学生成绩。

```
DELIMITER //
CREATE PROCEDURE PROC_COURSE_GRADE(IN course_id CHAR(4))
BEGIN
SELECT * FROM SC WHERE Cno=course_id;
END //
```

  - 第一条会将数据库的终止符修改为//，因为数据库的默认终止符为分号，为了避免与存储过程中SQL语句块中的终止符（即分号）冲突，需要先使用DELIMITER语句来修改终止符，并以“END //”结束存储过程

- 在调用已创建的存储过程时，需要先将数据库的终止符恢复为默认的分号，然后使用CALL语句调用存储过程。查询1号课程的学生成绩：

```
DELIMITER ;
CALL PROC_COURSE_GRADE(“1”);
```

- 输出结果为：

Sno	Cno	Grade
2021310722	1	87
2021310723	1	92

创建和调用存储过程

- 【例4.76】创建名为PROC_COURSE_AVG_GRADE的存储过程，计算某门课程的平均学生成绩，并将结果存储在某个变量中。

```
DELIMITER //
CREATE PROCEDURE PROC_COURSE_AVG_GRADE(IN course_id CHAR(4), OUT avg_grade FLOAT)
BEGIN
SELECT AVG(GRADE) INTO avg_grade FROM SC WHERE Cno=course_id;
END //
```

- 调用PROC_COURSE_AVG_GRADE存储过程，查询1号课程的平均学生成绩，并将结果存储在avg_grade变量中：

```
DELIMITER ;
CALL PROC_COURSE_AVG_GRADE("1",@avg_grade);
```

- 查询avg_grade变量的值：

```
SELECT @avg_grade;
```

@avg_grade
89.5

存储过程和函数

- 存储过程和函数都用于提高数据库性能，减少频繁访问数据库和减少网络延迟等方式加速执行效率。
- 函数：简单的计算型任务，例如字符串或日期拼接、返回单个值等。
- 存储过程：复杂的业务逻辑、更新和删除相关操作。存储过程需要显式地被调用，并且可以包含各种复杂的控制结构和代码块。

创建和调用函数

- 创建函数的语法格式为：Returns返回数据类型；Return返回数据值

```
CREATE FUNCTION<函数名>([参数,…,参数])
RETURNS <数据类型>
BEGIN
<SQL 语句块>
END <终止符>
```

  - 参数的格式与存储过程相似，但是只能是IN参数
  - RETURNS语句指定函数返回数据的类型

- 【例4.77】创建函数，返回某名学生选修的某门课程的成绩。

```
DELIMITER //
CREATE FUNCTION FUNC_STU_COU_GRADE(student_id CHAR(10), course_id CHAR(4))
RETURNS INT
RETURN (SELECT Grade FROM SC WHERE Sno=student_id AND Cno=course_id);
//
```

- 调用FUNC_STU_COU_GRADE函数，查询学号2021310722的学生的1号课程成绩。

```
DELIMITER ;
SELECT FUNC_STU_COU_GRADE("2021310722", "1");
```

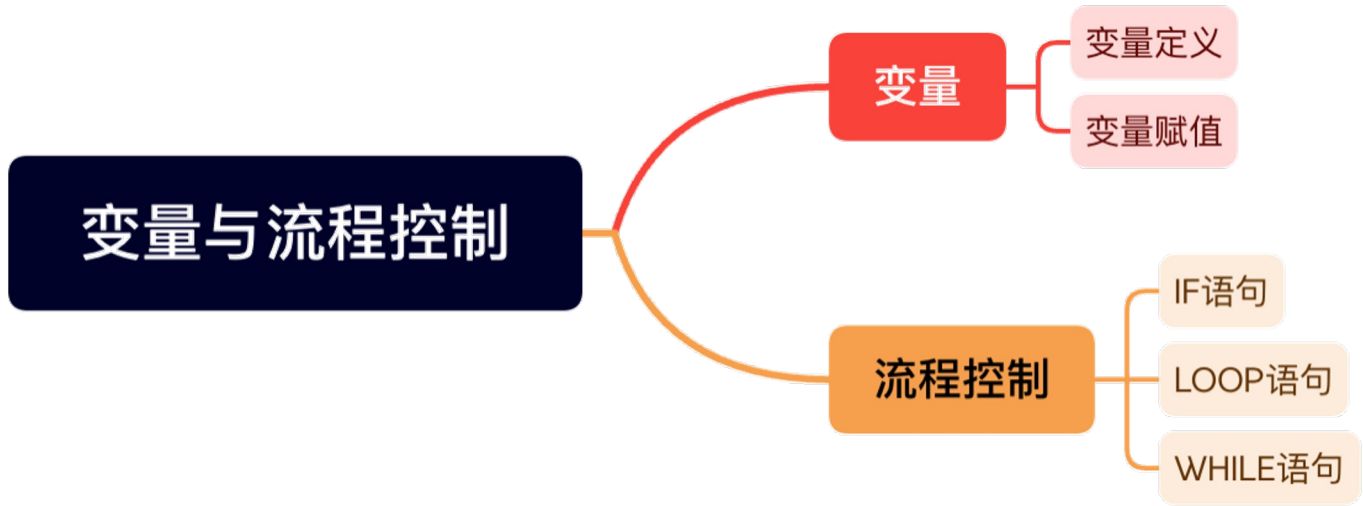
FUNC_STU_COU_GRADE(“2021310722” , “1”)
87

存储过程和函数的区别

- 存储过程可以通过OUT或INOUT参数返回多个值，而函数只能返回RETURNS子句中指定的某一类型的单值或表对象。
- 存储过程的参数可以为IN、OUT或INOUT，而函数的参数只能是IN类型的。
- 存储过程可以通过CALL语句作为一个独立的部分来调用和执行，而函数可以作为查询语句的一部分来调用。另外，由于函数可以返回表对象，因此函数的返回结果也可以用在查询语句的FROM子句中。
- 创建函数时必须指定返回值数据类型，且函数体内必须有一个RETURNS语句。
- 存储过程中可以执行更新表的数据库操作，而函数则不可以。

变量和流程控制

- 变量和流程控制是高级程序设计语言中十分重要的两个概念
- 在SQL存储过程和函数中也存在变量和流程控制的概念，从而实现一些复杂的功能
- 变量：可以在存储过程和函数中声明并使用变量
- 流程控制：可以用来改变存储过程和函数内部语句的执行顺序。包括IF语句、LOOP语句、WHILE语句、REPEAT语句、LEAVE语句等。



变量

- 可以在存储过程和函数中声明并使用变量
- 变量的作用范围是在BEGIN...END语句块中

- 变量定义：

```
DECLARE <变量名>[,<变量名>,…,<变量名>] <数据类型> [DEFAULT <默认值>];
```

  - 变量名（列表）指定了定义的变量的名称
  - 数据类型定义了这些变量的数据类型
  - 当指定DEFAULT选项时，默认值为这些变量提供了一个默认值。如果未指定DEFAULT选项，变量的初始值为NULL

- 变量赋值：

```
SET <变量名>=<表达式>[,<变量名>=<表达式>,…, <变量名>=<表达式>];
```

IF语句

- IF语句包含多个判断条件，根据判断结果是否为真来选择执行哪个分支：

```
IF <表达式> THEN <SQL 语句块>;
[ELSEIF <表达式> THEN <SQL 语句块>;
…
ELSEIF <表达式> THEN <SQL 语句块>;]
ELSE <SQL 语句块>;
END IF;
```

- 程序执行时，会自上而下地判断各子句中的表达式是否为真，如果为真，则执行该子句对应的语句块，并跳出该语句；如果为假，则继续判断下面的子句中的表达式是否为真。
- 如果IF子句和ELSEIF子句中对应的条件都为假，则执行ELSE子句中的语句块。

IF语句

- 【例4.78】创建一个名为FUNC_GRADE_LEVEL的成绩等级评定函数，参数为学生学号与课程号，返回值为学生该门课程的成绩等级，包括：A[90-100），B[80-90），C[70-80），D[60-70），E（60以下）。

```
DELIMITER //
CREATE FUNCTION FUNC_GRADE_LEVEL(student_id CHAR(10), course_id CHAR(4))
RETURNS VARCHAR(5)
BEGIN
DECLARE level VARCHAR(5) DEFAULT "E";
DECLARE grade INT;
SET grade=(SELECT Grade FROM SC WHERE Sno=student_id AND Cno=course_id);
IF grade>=90 THEN SET level="A";
ELSEIF grade<90 AND grade>=80 THEN SET level="B";
ELSEIF grade<80 AND grade>=70 THEN SET level="C";
ELSEIF grade<70 AND grade>=60 THEN SET level="D";
ELSE SET level="E";
END IF;
RETURN level;
END //
```

- 调用FUNC_GRADE_LEVEL函数，查询学号2021310722的学生的1号课程成绩。

```
DELIMITER ;
SELECT FUNC_GRADE_LEVEL("2021310722", "1");
```

FUNC_GRADE_LEVEL(“2021310722”，“1”)
B

LOOP语句

- LOOP语句是循环语句，用来重复执行一些语句：

```
[<标签>:] LOOP <SQL 语句块> END LOOP [<标签>];
```

- 程序执行时，会重复执行LOOP后的语句块，直到循环被退出。
- 在LOOP语句中，使用LEAVE子句可跳出循环。
- LOOP语句中必须包含LEAVE子句，否则会陷入死循环。
- 标签可以用来标志一个LOOP语句，为可选项。

LOOP语句

- 【例4.79】创建一个名为FUNC_SUM的函数，参数为num，计算1+2+...+(num-1)+num的结果。

```
DELIMITER //
CREATE FUNCTION FUNC_SUM(num INT)
RETURNS INT
BEGIN
DECLARE total INT DEFAULT 0;
sum_loop: LOOP
SET total=total+num;
SET num=num-1;
IF num<=0 THEN LEAVE sum_loop;
END IF;
END LOOP sum_loop;
RETURN total;
END //
```

- 调用FUNC_SUM函数，参数为10。

```
DELIMITER ;
SELECT FUNC_SUM(10);
```

FUNC_SUM(10)
55

WHILE语句

- WHILE语句也是循环语句，当满足WHILE语句的循环条件时，循环会一直执行；当不满足WHILE语句的循环条件时，会跳出循环：

```
[<标签>:] WHILE <表达式>
DO <SQL 语句块>
END WHILE [<标签>];
```

- 程序执行时，会先判断WHILE子句后的表达式是否为真，如果为真，则执行DO子句指定的语句块；否则，跳出循环。
- 标签可以用来标志一个WHILE语句，为可选项。

WHILE语句

- 【例4.80】创建一个名为FUNC_SUM2的函数，参数为num，计算1+2+...+(num-1)+num的结果。

```
DELIMITER //
CREATE FUNCTION FUNC_SUM2(num INT)
RETURNS INT
BEGIN
DECLARE total INT DEFAULT 0;
WHILE num>0
DO SET total=total+num, num=num-1;
END WHILE;
RETURN total;
END //
```

- 调用FUNC_SUM2函数，参数为10。

```
DELIMITER ;
SELECT FUNC_SUM2(10);
```

FUNC_SUM2(10)
55

删除存储过程和函数

- 删除存储过程的语句为：

```
DROP PROCEDURE <存储过程名>;
```

- 删除函数的语句为：

```
DROP FUNCTION <函数名>;
```

目录

1. SQL数据定义语言
2. SQL数据操纵语言
3. SQL数据控制语言
4. 存储过程和函数
5. **触发器**
6. 使用程序设计语言访问数据库

触发器

- 触发器是与表相关的特殊的存储过程，在满足特定条件时，它会被触发执行
- 触发器是定义在基本表上的，当基本表被修改（比如插入、删除、更新数据）时，会激活定义在其上的触发器，该基本表称为触发器的目标表
- 触发器可以用来保证数据库的完整性

创建触发器

- SQL中创建触发器的命令是CREATE TRIGGER：

```
CREATE TRIGGER <触发器名>
<触发时机> <触发事件> ON <表名>
FOR EACH ROW
<触发动作体>
```

- 触发器名：指定要创建的触发器的名字
- 触发时机：指定了触发执行的时间，可以为BEFORE（触发事件之前）或AFTER（触发事件之后）
- 触发事件：指定了当发生何种事件时，触发器会被激活。包括INSERT、DELETE、UPDATE
- 表名：指定了触发器是在哪张表上创建的
- 触发动作体：触发事件发生后要执行的SQL语句块
  - 对于触发事件作用的每一行（FOR EACH ROW），会执行触发动作体
  - 对于触发事件作用的每一行，在触发事件发生之前该行称之为OLD，在触发事件发生之后该行称之为NEW
  - 可以使用OLD和NEW来访问触发事件发生前后的元组的值

创建触发器

- 【例4.81】创建一个名为TRI_PRO_STU的触发器，在每次对学生成绩进行更新时，判断该学生成绩是否提升，如果提升，将该学生学号存入表Progressive_Student(Sno)中；否则，将该学生学号存入表Regressive_Student(Sno)中。

```
DELIMITER //
CREATE TRIGGER TRI_PRO_STU
AFTER UPDATE OF Grade ON SC
FOR EACH ROW
BEGIN
IF NEW.Grade>OLD.Grade THEN INSERT INTO Progressive_Student VALUES (NEW.Sno);
ELSE INSERT INTO Regressive_Student VALUES(NEW.Sno);
END IF;
END //
DELIMITER ;
```

删除触发器

- 删除触发器的语句为：

```
DROP TRIGGER <触发器名>;
```

- 触发器是基于行激活的，对数据库中每行数据的修改都会调用触发器。因此可能会导致数据库性能的降低，所以要避免编写过多的触发器。

- 触发器的应用场景：
  - 数据完整性
  - 数据审计
  - 数据安全性
  - 数据备份和同步

目录

1. SQL数据定义语言
2. SQL数据操纵语言
3. SQL数据控制语言
4. 存储过程和函数
5. 触发器
6. **使用程序设计语言访问数据库**

高级语言访问数据库的方法

- 嵌入式SQL（Embedded SQL，ESQL）是将SQL语句嵌入到程序设计语言中
  - 被嵌入的程序设计语言称为宿主语言，如C、C++、Java等
- JDBC： Java Database Connectivity，Java数据库连接
  - JDBC提供Java程序访问数据库的标准接口，Java应用程序可以基于JDBC访问不同类型的数据库。
- ODBC： Open Database Connectivity，开放数据库连接
  - 有统一的访问数据库的接口，并为不同的数据库系统实现了相应的ODBC驱动程序。

嵌入式SQL

- 嵌入式SQL（Embedded SQL，ESQL）是将SQL语句嵌入到程序设计语言中
- 被嵌入的程序设计语言称为宿主语言，如C、C++、Java等
- 通过调用宿主语言中可以用来访问数据库的包或库，可以达到使用嵌入式SQL来访问数据库的目的

- 在使用嵌入式SQL访问数据库的步骤：
  - 建立与数据库的连接，以验证访问的合法性
  - 使用嵌入式SQL来访问数据库
  - 释放所占用的连接资源，关闭程序与数据库的连接

嵌入式SQL

- 嵌入式SQL连接数据库：EXEC SQL CONNECT TO 数据库名字AS连接名字 USER用户名;
- 嵌入式SQL断开连接：EXEC SQL DISCONNECT 连接名字或EXEC SQL DISCONNECT CURRENT;
- 嵌入式SQL提交与撤销：
  - 执行提交语法为 EXEC SQL COMMIT WORK;
  - 执行撤消语法为 EXEC SQL ROOLBACK WORK;
  - 直接提交或者撤销后断开连接，即EXEC SQL COMMIT RELEASE和 EXEC SQL ROOLBACK RELEASE。

- 嵌入式SQL查询：嵌入式SQL可以将SQL查询结果赋值给高级语言的变量。
  - EXEC SQL BEGIN DECLARE SECTION; //开始声明
  - char cSname[10], queryName[10]= “李博”; int cSage; //声明变量
  - EXEC SQL END DECLARE SECTION; //结束声明
  - EXEC SQL SELECT Sname, Sage INTO :cSname, :cSage FROM Student WHERE Sname = :queryName ；定义的变量可以输入给SQL（例如queryName），也可以从SQL获取值（例如cSName和cSage）。

嵌入式SQL

- 在通过嵌入式SQL访问数据库时，嵌入式SQL语句与宿主语言之间要进行数据传输。
- 高级语言一般一次处理一条元组。当SQL查询结果为多条元组时，需解决SQL和高级语言转换的问题。
- 嵌入式SQL的数据传输是通过游标来实现一次获得一个元组。
- 游标存储嵌入式SQL语句的查询结果集合，每个SQL语句会返回一个游标（即返回SQL一条结果）。游标指向某条记录集的指针，通过这个指针移动，每次读一行数据处理一行，直到处理完毕。

- 在程序中使用宿主语言访问游标，可以获取查询的结果。游标的使用过程主要包括以下四个步骤：
  - （1）定义游标：将游标与SQL相关联。EXEC SQL DECLARE curStudent CURSOR FOR SELECT Sno, Sname FROM Student WHERE Sno='1' ;
  - （2）打开游标：执行与游标相关联的SQL，查询结果集合会被存储到游标的数据缓冲区。例如EXEC SQL OPEN curStudent;
  - （3）使用游标：从游标中读取数据（即查询结果集合），每次读取一条记录，也可以一次性读取全部记录。EXEC SQL FETCH curStudent INTO :cSno, :cSname; 将读取的一条记录的属性值赋值给两个变量cSno和cSname。
  - （4）关闭游标：停止使用游标，释放游标的数据缓冲区。例如EXEC SQL CLOSE curStudent。

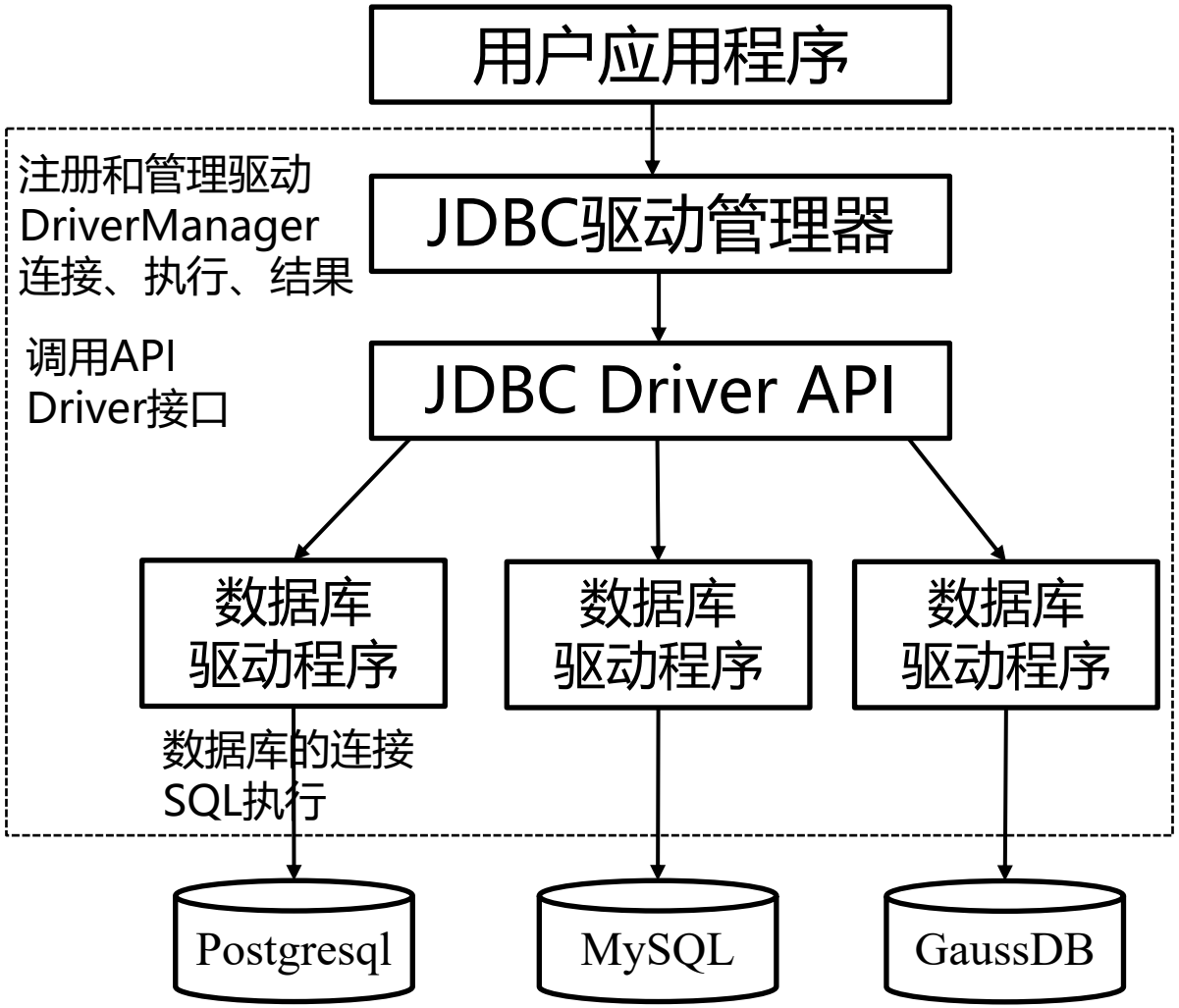
嵌入式SQL

- 嵌入式SQL语句与宿主语言之间的数据传输是通过游标实现的。
- 游标存储嵌入式SQL语句的查询结果集合，每个SQL查询语句会返回一个游标（即返回SQL查询的结果）。
- 在程序中使用宿主语言访问游标，可以获取查询的结果。

- 游标的使用过程主要包括以下四个步骤：
  1. 定义游标：将游标与SQL查询语句相关联。
  2. 打开游标：执行与游标相关联的SQL，查询结果集合会被存储到游标的数据缓冲区。
  3. 使用游标：从游标中读取数据，可以每次读取一条记录，也可以一次性读取全部记录。
  4. 关闭游标：停止使用游标，释放游标的数据缓冲区。

JDBC

- JDBC（Java Database Connectivity，Java数据库连接）是一种应用程序编程接口，是Java开发工具包（Java Development Kit，JDK）的一部分。
- JDBC提供Java程序访问数据库的标准接口，Java应用程序可以基于JDBC访问不同类型的数据库。
- JDBC依赖于两个软件组件：
  - java.sql包：JDBC访问数据库的标准接口
  - JDBC驱动程序：实现java.sql中定义的访问数据库的接口，从而从数据库中获取数据
- 通过编写调用java.sql中接口的代码，Java程序员可以实现对数据库的访问。

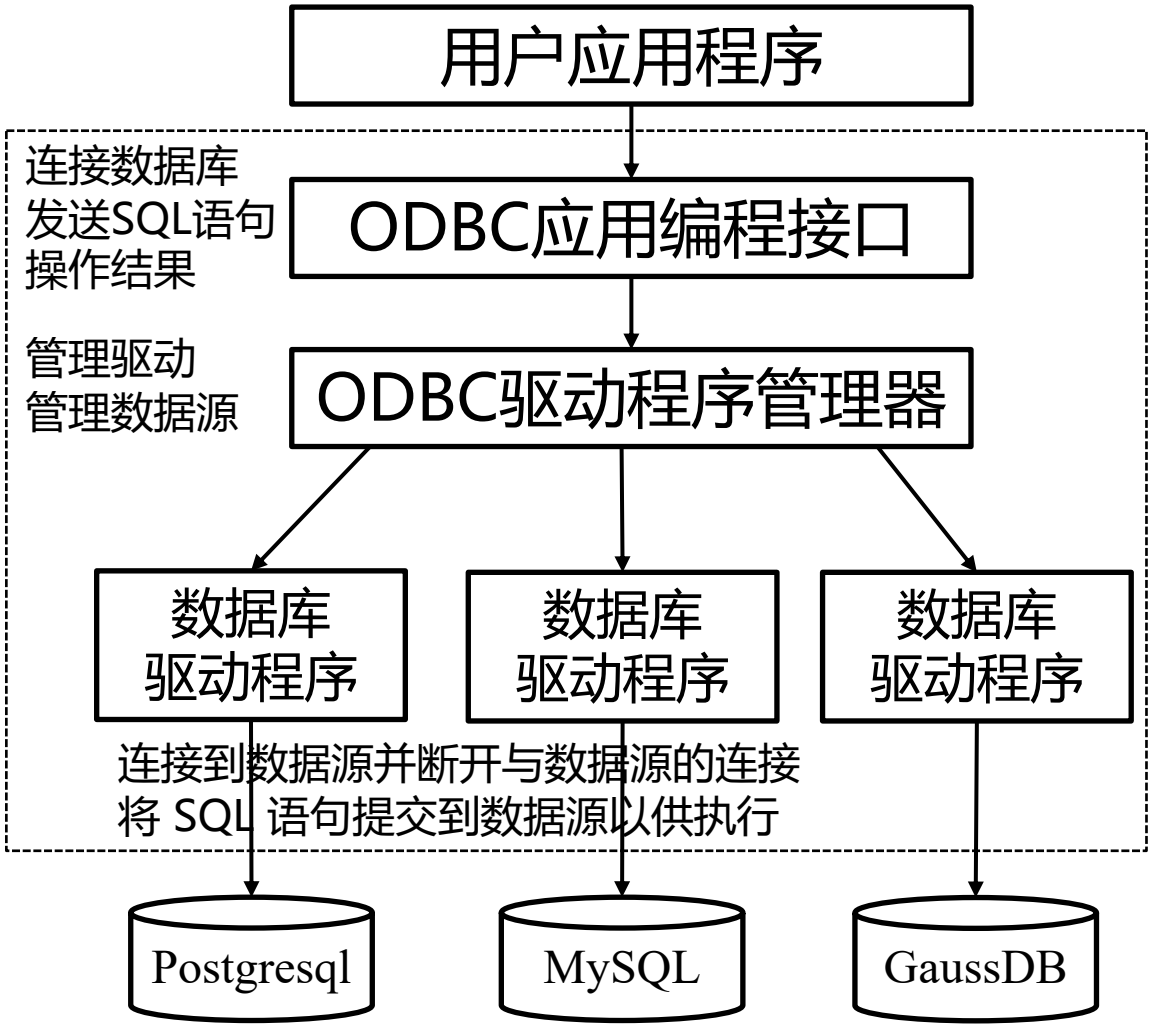


JDBC

```
Connection conn=null;//初始化连接
Statement stmt=null;//初始化执行对象，提交SQL
ResultSet rs=null;//SQL结果集
Class.forName(com.DB.jdbc.Driver);  //加载JDBC的数据库驱动
conn=DriverManager.getConnection("localhost", "root", "123456");  //连接数据库
stmt=conn.createStatement();  //实例化Statement对象
ResultSet rs=stmt.executeQuery("SELECT * FROM Student WHERE Sage>=18"); //执行SQL查询
while (rs.next()) {  //获取SQL查询结果中的一条记录
  String sno=rs.getString("Sno"); //获取SQL查询结果中的一条记录的各个字段
  String sname=rs.getString("Sname");
}
rs.close();   //关闭查询结果
stmt.close();   //关闭Statement对象
conn.close();    //关闭连接
```

ODBC

- ODBC（Open Database Connectivity，开放数据库连接）是微软公司建立的一组访问数据库的标准应用程序接口。
- ODBC可以用于任何程序设计语言
- 对于不同的数据库系统，ODBC有统一的访问数据库的接口，并为不同的数据库系统实现了相应的ODBC驱动程序。
- 当ODBC应用程序访问数据库时，其访问请求会被驱动程序管理器提交给相应的数据库系统的ODBC驱动程序，然后ODBC驱动程序会返回对数据库的结果。



ODBC

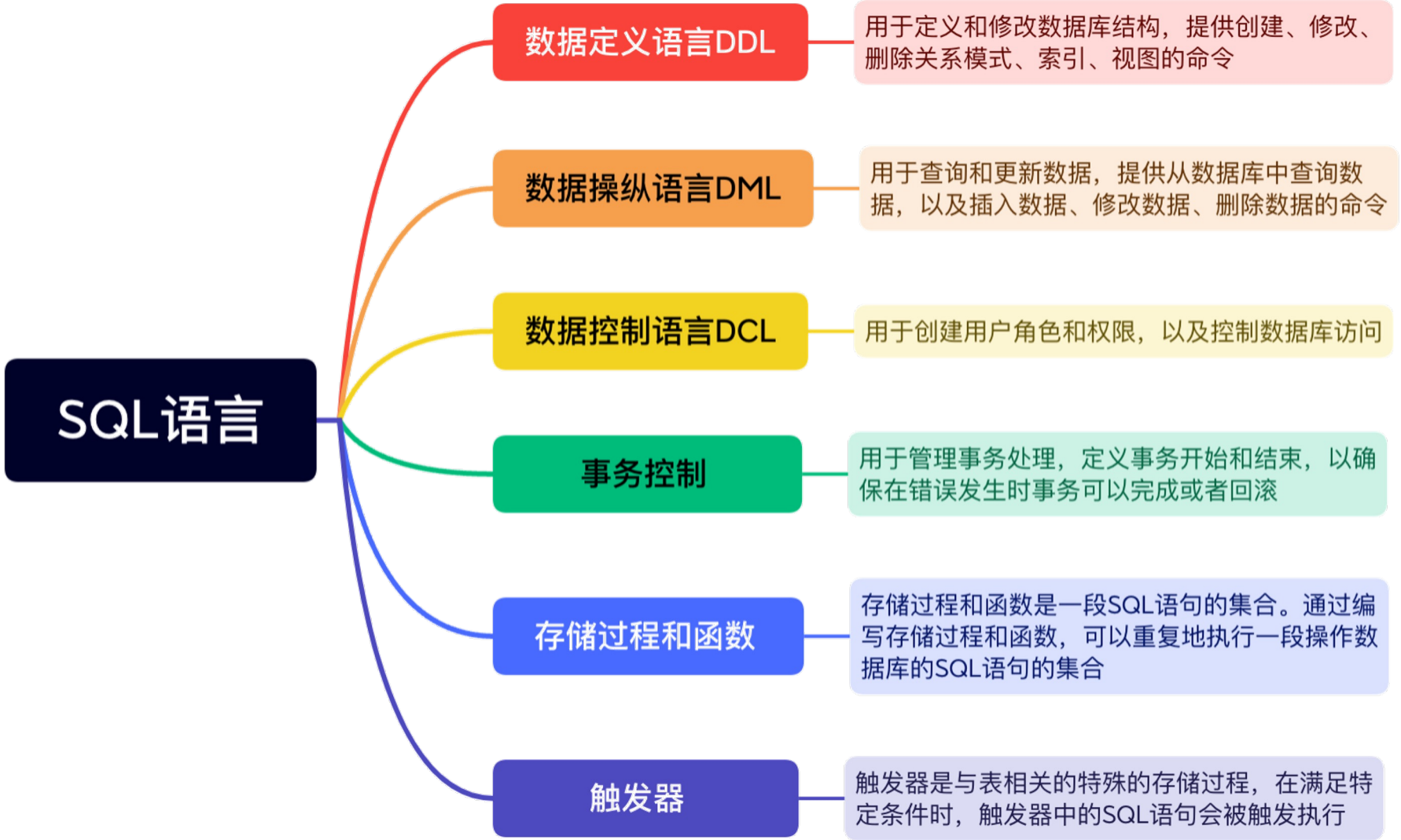
```
SQLHENV henv; //定义环境句柄：设置当前程序的整体属性，如ODBC的版本，最大连接数等
SQLHDBC hdbc; //定义连接句柄：控制当前连接的属性，如事务自动提交、连接类型、隔离级别等
SQLHSTMT hstmt; //定义语句句柄：完成数据库对象及数据的操作
SQLRETURN ret=SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv); //分配环境句柄
ret=SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);//设置ODBC环境属性
ret=SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc); //分配连接句柄
ret=SQLConnect(hdbc, "DB", SQL_NTS, "root", SQL_NTS, "123456", SQL_NTS);
if(SQL_SUCCEEDED(ret)){
  ret=SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt); //分配语句句柄
  ret=SQLExecDirect(hstmt, "SELECT * FROM Student WHERE Sage>=18", SQL_NTS);//执行SQL查询
  SQLCHAR sno[10]; SQLCHAR sname[10];
  while(SQL_SUCCEEDED(ret=SQLFetch(hstmt))) //获取SQL查询结果中的一条记录{
        //获取SQL查询结果中的一条记录的各个字段
        ret=SQLGetData(hstmt, 1, SQL_C_CHAR, sno, 10, NULL);
        ret=SQLGetData(hstmt, 2, SQL_C_CHAR, sname, 10, NULL); }}
SQLFreeHandle(SQL_HANDLE_STMT, hstmt);   //释放语句句柄
SQLDisconnect(hdbc);                     //关闭连接
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);     //释放连接句柄
SQLFreeHandle(SQL_HANDLE_ENV, henv);     //释放环境句柄
```

事务控制

- 开始事务：START TRANSACTION
- 提交事务：COMMIT
- 回滚事务：ROLLBACK/ABORT
- 在事务内部设置回滚标记点: SAVEPOINT sp_name
- 删除回滚标记点: RELEASE SAVEPOINT
- 将事务回滚到标记点: ROLLBACK TO sp_name

第六章将会介绍事务

小结


SQL语言
数据定义语言DDL
用于定义和修改数据库结构，提供创建、修改、删除关系模式、索引、视图的命令
数据操纵语言DML
用于查询和更新数据，提供从数据库中查询数据，以及插入数据、修改数据、删除数据的命令
数据控制语言DCL
用于创建用户角色和权限，以及控制数据库访问
事务控制
用于管理事务处理，定义事务开始和结束，以确保在错误发生时事务可以完成或者回滚
存储过程和函数
存储过程和函数是一段SQL语句的集合。通过编写存储过程和函数，可以重复地执行一段操作数据库的SQL语句的集合
触发器
触发器是与表相关的特殊的存储过程，在满足特定条件时，触发器中的SQL语句会被触发执行