

西北工业大学

Northwestern Polytechnical University

数据库系统原理

Database System

第八章 数据库编程

赵晓南

2024.10

- 数据库安全性实现的方法
 - 用户标识与鉴别
 - 存取控制
 - 视图、审计、数据加密
- 自主存取控制 (DAC) 与强制存取控制 (MAC)
- 授权 (Authentication) 与回收 (GRANT & REVOKE)

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

```
REVOKE <权限>[, <权限>]...  
[ON <对象类型> <对象名>]  
FROM <用户>[, <用户>]...
```

8.1.概述 (扩展SQL)

8.2.过程化SQL

8.3 JDBC与PDBC

8.4 连接池

8.5 基于MVC框架的数据库应用开发



8.1.1 SQL表达能力的限制



【任务1】查询“数据库系统原理”课程的所有先修课信息

Course表

课程号Cno	课程名Cname	学分Ccredit	先修课程Cpno
81001	程序设计基础与C语言	4	NULL
81002	数据结构	4	81001
81003	数据库系统概论	4	81002
81004	信息系统概论	4	81003
81005	操作系统	4	81001
81006	Python语言	3	81002
81007	离散数学	4	NULL
81008	大数据技术概论	4	81003

求解思路：直接先修课 + 间接先修课

■ 任务1：查询“数据库系统原理”课程的所有先修课信息

■ 难点：课程可能同时存在直接先修课和间接先修课

情况一：如何查询直接先修课：自身连接查询

情况二：如何查询间接先修课：递归查询（无法表达）

情况一：查询直接先修课的自连接

① 步骤1：找出“数据库系统原理”课程的全部直接先修课：记为L[1]；
如果L[1]为空，则任务一结束。

直接先修课

```
select B.Cname
from Course A, Course B
where A.Cname = '数据库系统原理'
and A.Cpno=B.Cno;
```

■ 情况二：如何查询间接先修课 — 递归执行

- ② 步骤 i ($i \geq 2$): 找出集合 $L[i-1]$ 中每一门课程的全部直接先修课, 并计算它们的并集, 记为 $L[i]$
- ③ 迭代执行步骤 i , 直到并集 $L[i]$ 为空, 输出 $L[1] \cup \dots \cup L[i]$

```
L[2]: select B.Cname
      from Course A, Course B
      where A.Cname = '数据结构' and A.Cpno=B.Cno;

L[3]: select B.Cname
      from Course A, Course B
      where A.Cname = '程序设计基础与C语言' and A.Cpno=B.Cno;
```

■ 递归执行实例：查询间接先修课

1. 执行步骤1，找出“数据库系统原理”的直接先修课“数据结构”，即为L[1]。
2. 执行步骤2，找出“数据结构”的直接先修课“程序设计基础与C语言”，即为L[2]。
3. 执行步骤3，找出“数据结构”的先修课，得到 L[3]。
4. 执行步骤4，找出“程序设计基础与C语言”的先修课，得到 L[4]。
5. 发现L[3]和L[4]都为空，递归查询结束。根据计算结果L[1] U L[2]，任务1的输出如下所示。

Cpno	Cname
81002	数据结构
81001	程序设计基础与C语言

递归查询：引入**WITH RECURSIVE**子句，执行递归查询

WITH子句的一般格式:

```
WITH RS1 [(<目标列>,<目标列>)] AS /* RS1为临时结果集的命名*/  
  (SELECT 语句1) [, /* RS1对应SELECT 语句的执行结果*/  
    /*SELECT语句1中的目标列与RS1中的目标列必须保持一致*/  
  RS2 [(<目标列>,<目标列>)] AS /* RS2为临时结果集的命名*/  
  (SELECT 语句2) ,... /* RS2对应SELECT 语句的执行结果*/  
    /*SELECT语句2中的目标列与RS2中的目标列必须保持一致*/  
SQL语句; /* 执行与RS1, RS2,...,相关的查询*/
```


[例] 求81001-01和81001-02两个教学班之间学生选课平均成绩的差异。

```
WITH
RS1 (Grade)
    AS
    (SELECT AVG(Grade) FROM SC
     WHERE Teachingclass = '81001-01'),
RS2 (Grade)
    AS
    (SELECT AVG(Grade) FROM SC
     WHERE Teachingclass = '81001-02')

SELECT RS1.Grade-RS2.Grade from RS1,RS2;
```

■ WITH RECURSIVE子句的一般格式

WITH RECURSIVE RS AS

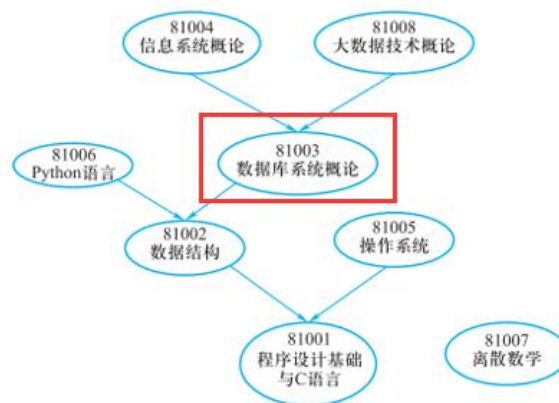
```
(  
    SEED QUERY          /*初始化查询的临时结果集，记为L[1]*/  
    UNION [ALL]         /*是否需要保留重复记录，加ALL为保留*/  
    RECURSIVE QUERY  
        /*执行递归查询，得到全部临时结果集，即L[2] U ... U L[i]*/  
)  
SQL语句          /*执行与RS相关的查询*/
```

8.1.2 扩展SQL的功能 — 递归查询



【例】查询“数据库系统原理”课程的所有先修课信息

课程号Cno	课程名Cname	学分Ccredit	先修课程Cpno
81001	程序设计基础与C语言	4	NULL
81002	数据结构	4	81001
81003	数据库系统概论	4	81002
81004	信息系统概论	4	81003
81005	操作系统	4	81001
81006	Python语言	3	81002
81007	离散数学	4	NULL
81008	大数据技术概论	4	81003



一门课有多
门先修课，
怎么处理？

WITH RECURSIVE RS AS (

/*初始化RS，假设结果集为L[1]，即“数据库系统原理”的所有直接先修课*/

SELECT Cpno FROM Course WHERE Cname = '数据库系统原理'

UNION

/*递归查询第i层(i>=1)的数据，即第i-1层数据的直接先修课课程号，并更新RS*/


SELECT Course.Cpno FROM Course,RS WHERE RS.Cpno = Course.Cno)

/*根据RS中记录的所有先修课程号，通过查找课程表，输出课程号与课程名*/

SELECT Cno, Cname FROM Course WHERE Cno IN (SELECT Cpno FROM RS);

斐波那契数列计算

```
WITH RECURSIVE fibonacci (n, fib_n, next_fib_n)
AS
(
    SELECT 1, 0, 1
    UNION ALL
    SELECT n + 1, next_fib_n, fib_n + next_fib_n
    FROM fibonacci WHERE n < 10
)
SELECT * FROM fibonacci;
```

Result Grid  Filter Rows: <input type="text"/>			
	n	fib_n	next_fib_n
▶	1	0	1
	2	1	1
	3	1	2
	4	2	3
	5	3	5
	6	5	8
	7	8	13
	8	13	21
	9	21	34
	10	34	55

8.1.2 扩展SQL的功能 — 内置函数



【任务2】打印一周内将过生日的学生信息

Student表

学号 Sno	姓名 Sname	性别 Sgender	生日 Sbirthdate	主修专业 Smajor
20180001	李勇	男	2000-3-8	信息安全
20180002	刘晨	女	1999-9-1	计算机科学与技术
20180003	王敏	女	2001-8-1	计算机科学与技术
20180004	张立	男	2000-1-8	计算机科学与技术
20180005	陈新奇	男	2001-11-1	信息管理与信息系统
20180006	赵明	男	2000-6-12	数据科学与大数据技术
20180007	王佳佳	女	2001-12-7	数据科学与大数据技术

求解思路：充分利用DBMS系统提供的日期函数



■SQL常用的内置函数可以分为：

- 数学函数（如绝对值函数等）
- 聚合函数（如求和、求平均函数等）
- 字符串函数（如求字符串长度、求子串函数等）
- 日期和时间函数（如返回当前日期函数等）
- 格式化函数（如字符串转IP地址函数等）
- 控制流函数（如逻辑判断函数等）
- 加密函数（如使用密钥对字符串加密函数等）
- 系统信息函数（如返回当前数据库名、服务器版本函数等）

【例】查询一周内将过生日的学生信息

```
SELECT Sno, Sname, Sgender, Sbirthdate, Smajor
FROM Student
WHERE to_date(to_char(current_date, 'yyyy') || '-' ||
to_char(Sbirthdate, 'mm-dd'))
BETWEEN CURRENT_DATE AND CURRENT_DATE + INTERVAL '7' DAY;
```

- ① 内置函数 `current_date` 返回当前的系统日期。
- ② 内置函数 `to_char(current_date, 'yyyy')` 返回当前系统日期的年份。
- ③ `to_date(to_char(current_date, 'yyyy') || '-' || to_char(Sbirthdate, 'mm-dd'))` 表示把当前年份与出生日期用 '-' 连在一起。符号 "||" 用于字符串拼接。
- ④ `current_date + interval '7' day`: 对当前的日期调整后的日期。参数 `interval` 是年(yyyy)、季度(q)、月(m)、日(d)、时(h)等粒度的时间单位。
- ⑤ 假设当前2021-6-9, 执行此WHERE语句, 判断学生表中每位学生转换后的出生日期是否在[2021-6-9, 2021-6-16]区间内, 如果是, 打印该学生的信息。

【任务3】给定学生学号，计算学生的平均学分绩点GPA

表8.2学号为“20180001”的学生的选修课程

学号 Sno	课程号 Cno	成绩 Grade	选课学期 Semester	教学班 Teachingclass
20180001	81001	85	20192	81001-01
20180001	81002	96	20201	81002-01
20180001	81003	87	20202	81003-01

难点：需要用户自主设计业务处理逻辑。

求解思路：给定学生学号，找出该学生所有选修课程的学分、成绩；根据每门课程的成绩，参照“成绩和绩点对照表”，确定该成绩所处的范围，找出该门课程对应的绩点。

8.1.2 扩展SQL的功能 — PL/SQL



【例】给定学生学号，计算学生的平均学分绩点GPA

- 81001-81003三门课程的学分都是4。根据平均学分绩点GPA的计算公式=总学分绩/总学分=（每门课程的学分*对应课程的绩点）的总和/12

$$\text{GPA} = (3*4 + 4*4 + 3*4)/12 = 3.33$$

学号 Sno	课程号 Cno	成绩 Grade	选课学期 Semester	教学班 Teachingclass
20180001	81001	85	20192	81001-01
20180001	81002	96	20201	81002-01
20180001	81003	87	20202	81003-01

编码	成绩下限	成绩上限	绩点
1	0	59	0
2	60	69	1
3	70	79	2
4	80	89	3
5	90	100	4

解决方案：使用存储过程实现 **CREATE PROCEDURE**

8.1.2 扩展SQL的功能 — PL/SQL



BEGIN

totalGPA := 0;

totalCredit := 0;

OPEN mycursor; /*打开游标mycursor */

LOOP /*循环遍历游标*/

 FETCH mycursor INTO credit, grade; /*检索游标*/

 EXIT WHEN mycursor%NOTFOUND;

 IF grade BETWEEN 90 AND 100 THEN courseGPA := 4.0;

 ELSIF grade BETWEEN 80 AND 89 THEN courseGPA := 3.0;

 ELSIF grade BETWEEN 70 AND 72 THEN courseGPA := 2.0;

 ELSIF grade BETWEEN 60 AND 69 THEN courseGPA := 1.0;

 ELSE courseGPA := 0;

 END IF; /*参照表8.2, 根据成绩找出某门课程对应的学分绩点*/

 totalGPA := totalGPA + courseGPA * credit;

 totalCredit := totalCredit + credit;

END LOOP;

CLOSE mycursor; /*关闭游标mycursor */

outGPA:= 1.0 * totalGPA / totalCredit;

END

部分关键逻辑
(KingBase语法)

8.1.2 扩展SQL的功能 — 应用开发



【任务4】教学评价浏览与反馈：学生通过交互界面提交对某一位任课老师的教学评价意见，教师浏览这些评价意见并提供反馈信息。

教师教学评价表

学号 Sno	职工号 Tno	教学班号 TCno	意见内容 Assess	意见类型 CAtype	教师反馈 Feedback
20180001	19950018	81001-01	作业难度比较合适	正面	感谢肯定
20180003	19950018	81001-01	老师和助教也很耐心	正面	感谢肯定
20180002	19910101	81001-02	实验框架较为复杂	负面	根据同学们的建议， 简化框架

难点：需要建立交互功能，1) 学生需要找到指定的教学班和授课教师，建立交互界面并输入课程评价。2) 建立交互界面，教师浏览教学班学生的评价意见，并针对每条评价逐一做出回复。

解决思路： GUI界面的数据库应用程序设计与开发

8.1.2 扩展SQL的功能 —— 补充



扩展SQL (OLAP)

✓ 窗口分析函数



✓ 分页 (Limit, offset)

✓ 抽样函数(random)

序号函数	ROW_NUMBER()	当前行在分区中的序号 (行号)
	RANK()	当前行在分区中的序号 (排名), 跳过重复的序号
	DENSE_RANK()	当前行在分区中的序号 (排名), 不跳过重复的序号
分布函数	PERCENT_RANK()	等级值百分比
	CUME_DIST()	累积分布值
首尾函数	FIRST_VALUE()	window frame 的第一行参数的值
	LAST_VALUE()	window frame 的最后一行参数的值
前后函数	LAG	分区中当前行之前的参数的值
	LEAD()	分区中当前行之前的参数的值
其它函数	NTH_VALUE()	窗口框第n行参数的值
	NTILE()	分区内当前行的桶号

例：查询每门课程单课的前三名，以及相关同学信息。

```
select * from (  
    select name, subject, score,  
           dense_rank() over(partition by subject order by score desc) as 'rn'  
    from student  
) tmp where tmp.rn <= 3;
```

ROW_NUMBER(): 顺序排序——1、2、3

RANK(): 并列排序, 跳过重复序号——1、1、3

DENSE_RANK(): 并列排序, 不跳过重复序号——1、1、2

name	subject	score	rn
冯宝宝	数学	100	1
苏沐橙	数学	100	1
温宁	英语	98	1
冯宝宝	语文	99	1
温宁	语文	99	1
苏沐橙	语文	99	1

扩展SQL (OLAP)

- ✓ 窗口分析函数
- ✓ 分页 (Limit, offset)
- ✓ 抽样函数

例：学生表中随机抽样3个学生

```
SELECT * FROM student  
ORDER BY RAND()  
LIMIT 3;
```

例：查询第11行-20行数据

```
SELECT * FROM students  
LIMIT 10 OFFSET 10;
```

按照一定间隔数据抽样：

```
mysql> select id, avg(count) from t2 group by id div 1000;
```

id	avg(count)
1000	1.0000
2000	2.0000
3000	3.0000
4000	4.0000
5000	5.0000
6000	6.0000
7000	7.0000
8000	8.0000
9000	9.0000

9 rows in set (0.00 sec)

通过group by将id根据[1000, 2000), [2000, 3000), [3000, 4000), [4000, 5000), [5000, 6000), [6000, 7000), [7000, 8000), [9000, 10000)分组，分组后的数据进行求和/平均等操作

■ SQL语言表达能力的限制

1. 无法表达递归等复杂操作 — 例如间接先修课的查询
2. 无法对数据进行复杂操作 — 查询一周内将过生日的同学
3. 无法自主设计业务处理逻辑 — 计算学生平均学分绩点
4. 无法进行交互式操作 — 教学评价与反馈

DBMS内部：复杂SQL语法支持、系统函数支持、PL/SQL支持

DBMS外部：数据库应用程序开发

本次课后练习：在MySQL中的实现任务1-任务3

应用程序访问数据库的技术

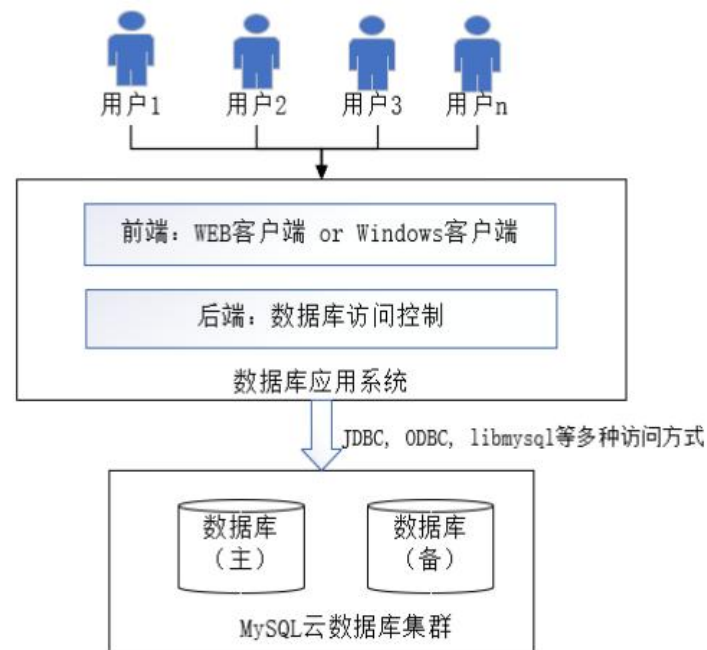
- 方法一：嵌入式SQL
- 方法二：直接利用DBMS提供的动态库
- 方法三：ODBC/JDBC等
- 方法四：各种框架技术

天府机场综合报警系统

2020年05月11日

0123456789123456

序号	发生时间	检测区域	报警类型	报警设备	报警编号	报警位置	报警详情	确认时间	操作
1	2020-05-06 18:00	航站楼	航站楼	航站楼	01178123456789	航站楼	航站楼	2020-05-06 18:00	清除
2	2020-05-06 18:00	航站楼	航站楼	航站楼	01178123456789	航站楼	航站楼	2020-05-06 18:00	清除
3	2020-05-06 18:00	航站楼	航站楼	航站楼	01178123456789	航站楼	航站楼	2020-05-06 18:00	清除
4	2020-05-06 18:00	航站楼	航站楼	航站楼	01178123456789	航站楼	航站楼	2020-05-06 18:00	清除
5	2020-05-06 18:00	航站楼	航站楼	航站楼	01178123456789	航站楼	航站楼	2020-05-06 18:00	清除
6	2020-05-06 18:00	航站楼	航站楼	航站楼	01178123456789	航站楼	航站楼	2020-05-06 18:00	清除
7	2020-05-06 18:00	航站楼	航站楼	航站楼	01178123456789	航站楼	航站楼	2020-05-06 18:00	清除
8	2020-05-06 18:00	航站楼	航站楼	航站楼	01178123456789	航站楼	航站楼	2020-05-06 18:00	清除
9	2020-05-06 18:00	航站楼	航站楼	航站楼	01178123456789	航站楼	航站楼	2020-05-06 18:00	清除

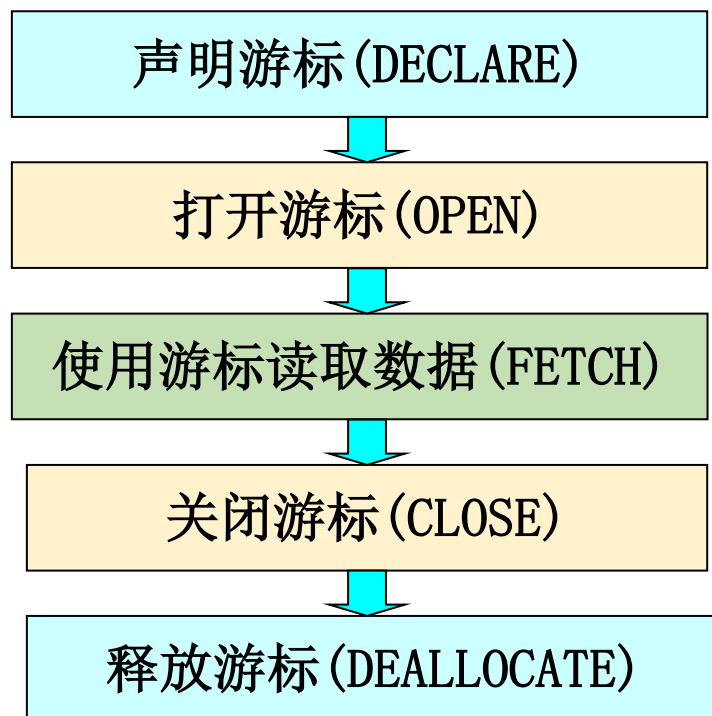




- 什么是嵌入式SQL (Embedded SQL)?
SQL语句嵌入到其他高级语言中，这时高级语言被称为(宿)主语言，这种方式下使用的SQL称为嵌入式SQL。
(静态SQL/动态SQL)
- 如何区分主语言与SQL语句?
 - 前缀: EXEC SQL
 - 结束标志: 随主语言的不同而不同
如: C语言中使用SQL: EXEC SQL <SQL语句> ;
例: EXEC SQL DROP TABLE Student ;

- 常见的几个数据库系统预编译器 (命令)
 - PostgreSQL: ecpg (.pgc -> .c)
 - SQL Server: nsqlprep.exe (.sqc -> .c)
 - Oracle: proc/c c++

- 游标：系统为用户开设的一个数据缓冲区，存放SQL语句执行的结果（多条记录）



应用场景：
需要逐条处理
集合中的记录

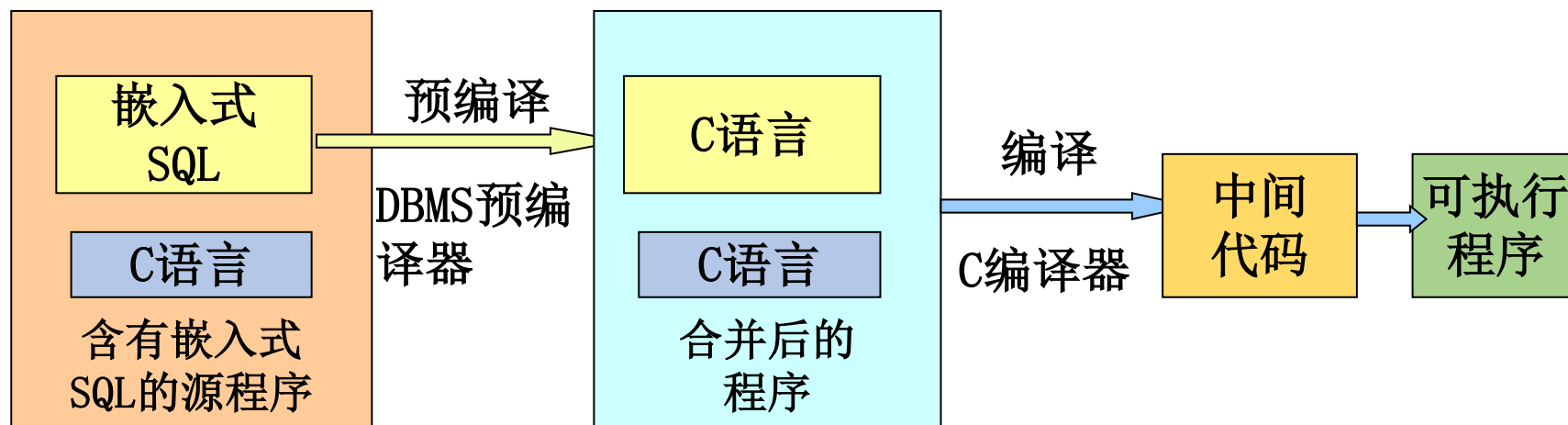
8.1.3 高级语言数据库编程 — 嵌入式SQL



例：带有嵌入式SQL的一小段C程序：查询并打印学生成绩。

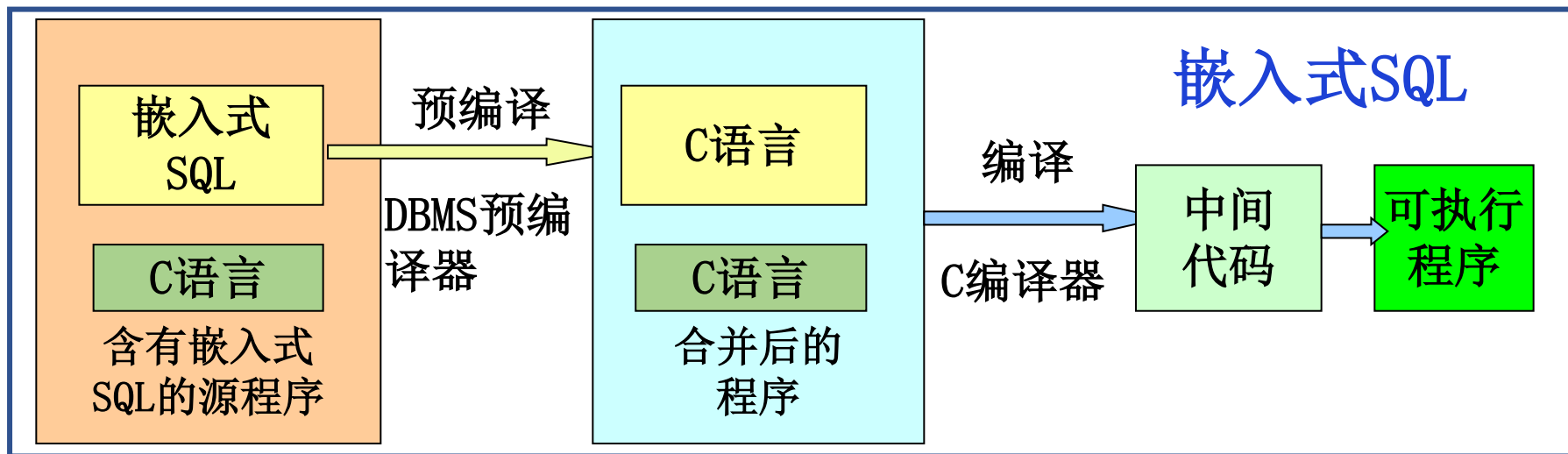
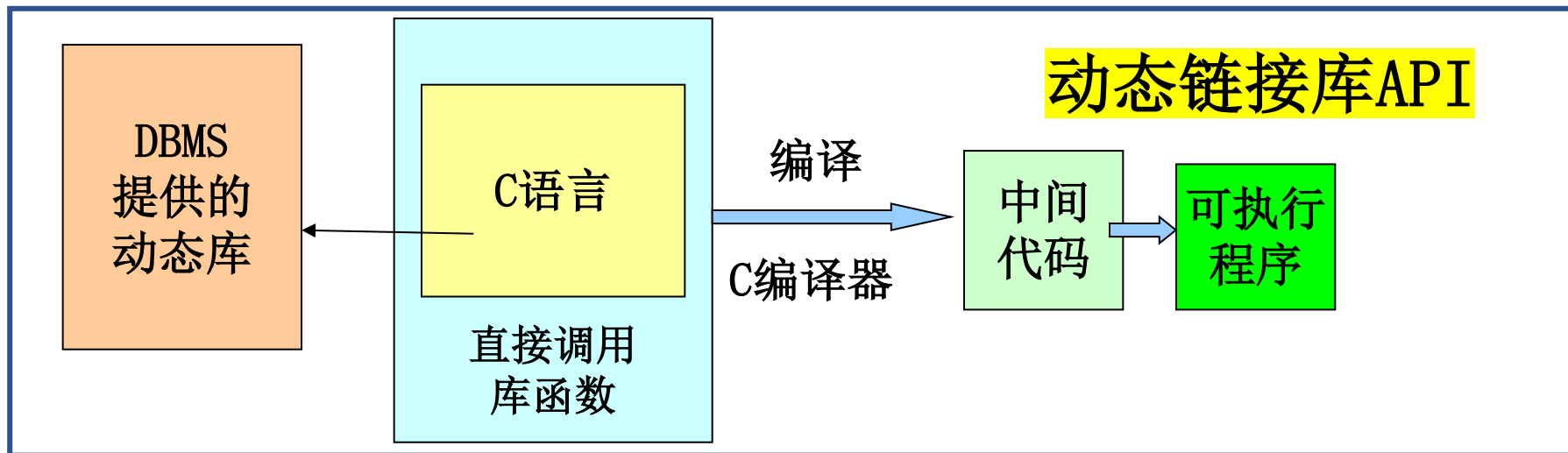
```
EXEC SQL INCLUDE SQLCA;          /* (1) 定义SQL通信区 */
EXEC SQL BEGIN DECLARE SECTION;   /* (2) 说明主变量 */
    CHAR  Sno(5);
    CHAR  Cno(3);
    INT    Grade;
EXEC SQL END DECLARE SECTION;
main()
{
    EXEC SQL DECLARE C1 CURSOR FOR /* (3) 游标操作(定义游标)*/
        SELECT  Sno, Cno, grade FROM SC; /* 从SC表查询Sno,Cno,Grade */
    EXEC SQL OPEN  C1;              /* (4) 游标操作（打开游标）*/
    for(;;)
    {
        EXEC SQL FETCH C1 INTO :Sno, :Cno, :Grade;
        /* (5) 游标操作（将当前数据放入主变量并推进游标指针）*/
        if (sqlca.sqlcode <> SUCCESS)
            /* (6) 利用SQLCA中的状态信息决定何时退出循环 */
            break;
        printf(" Sno: %s, Cno: %s, Grade:%d", Sno, Cno, Grade); /*打印查询结果*/
    }
    EXEC SQL CLOSE C1;              /* (7) 游标操作（关闭游标）*/
    EXEC SQL DEALLOCATE C1; /* (8) 游标操作（释放游标）*/
}
```

- 如何实现SQL语句的执行？
 - 扩充主语言使之能处理SQL
 - 预编译，将SQL语句转为主语言可执行的目标代码



1. 由DBMS的预处理程序对源程序进行扫描，识别出SQL语句
2. 把它们转换成主语言调用语句，以使主语言编译程序能识别它
3. 最后由主语言的编译程序将整个源程序编译成目标码。

■ 方法二：直接利用DBMS提供动态链接库



8.1.3 高级语言数据库编程 — MySQL驱动



← → ↻ dev.mysql.com/downloads/

MySQL Community Downloads

- MySQL Yum Repository
- MySQL APT Repository
- MySQL SUSE Repository
- MySQL Community Server
- MySQL Cluster
- MySQL Router
- MySQL Shell
- MySQL Operator
- MySQL Workbench
- MySQL Installer for Windows
- MySQL for Visual Studio
- C API (libmysqlclient)
- Connector/C++
- Connector/J
- Connector/NET
- Connector/Node.js
- Connector/ODBC
- Connector/Python
- MySQL Native Driver for PHP
- MySQL Benchmark Tool
- Time zone description tables
- Download Archives

ORACLE © 2022 Oracle

[Privacy](#) / [Do Not Sell My Info](#) | [Terms of Use](#) | [Trademark Policy](#) | [Cookie 喜好设置](#)

■ 方法二：直接利用DBMS提供的动态链接库

例如： mysql提供lib: libmysql.dll

```
mysql_init()           //初始化数据结构
mysql_library_init()    //初始化数据库
mysql_real_connect()    //连接数据库
mysql_query()           //查询
mysql_store_result()    // 存储结果
mysql_fetch_row()       //逐行处理
mysql_free_result()     //释放资源
mysql_close()           //关闭连接
```

例子： C_MYSQL.cpp



VS2019中C调用MySQL.docx

8.1.3 C/C++语言调用libmysql示例



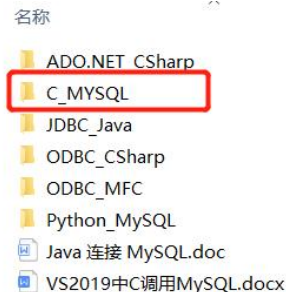
```
int main()
{
    //必备数据结构
    MYSQL myconn;

    //初始化数据结构
    if (mysql_init(&myconn) != NULL) {
        cout << "mysql_init() succeed" << endl;
    }

    //设置编码方式
    mysql_options(&myconn, MYSQL_SET_CHARSET_NAME, "gbk");

    //初始化数据库
    if (mysql_library_init(0, NULL, NULL) == 0) {
        cout << "mysql_library_init() succeed" << endl;
    }

    //连接数据库
    if (mysql_real_connect(&myconn, "127.0.0.1", "root",
        "123456", "student", 3306, NULL, 0) != NULL) {
        cout << "mysql_real_connect() succeed" << endl;
    }
}
```



8.1.3 C/C++语言调用libmysql示例



```
MYSQL_RES* res; //查询结果集
MYSQL_ROW row;  //存放一条数据记录，二维数组

const char* sql = "select * from s";
cout << sql << endl;

mysql_query(&myconn, sql);
res = mysql_store_result(&myconn);
while (row = mysql_fetch_row(res))
{
    cout << row[0] << "|" << row[1] << endl;
}
mysql_free_result(res);
mysql_close(&myconn);

return 0;
}
```

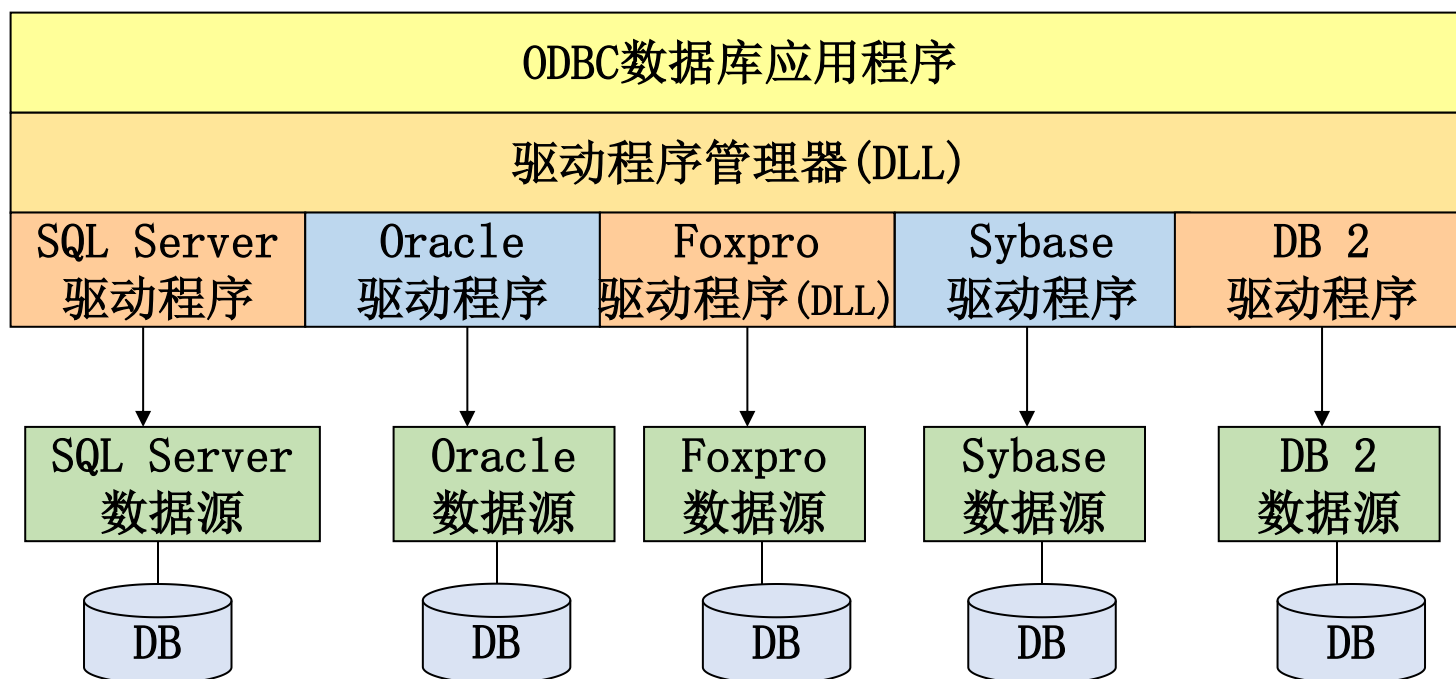
名称

- ADO.NET_CSharp
- C_MYSQL**
- JDBC_Java
- ODBC_CSharp
- ODBC_MFC
- Python_MySQL
- Java 连接 MySQL.doc
- VS2019中C调用MySQL.docx

■ 方法三：基于ODBC/JDBC等中间件

■ ODBC的概念

微软开发的用于连接各种关系型数据库的函数库，以动态链接库的形式提供给程序使用，其目的是给出统一的编程接口，从而简化数据库应用程序的编写。





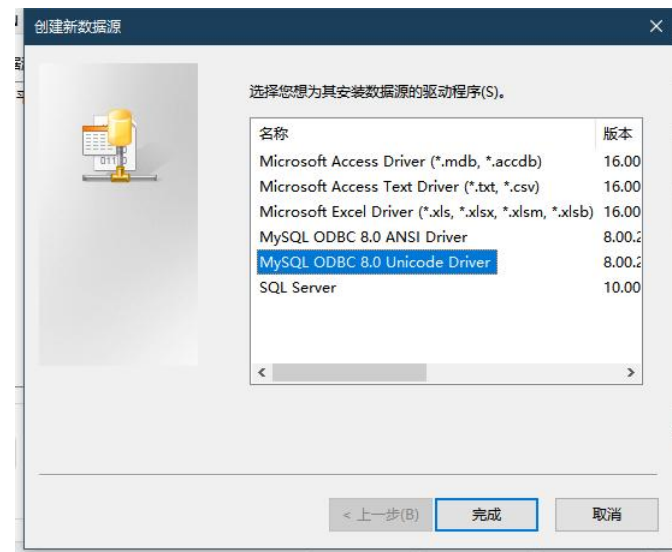
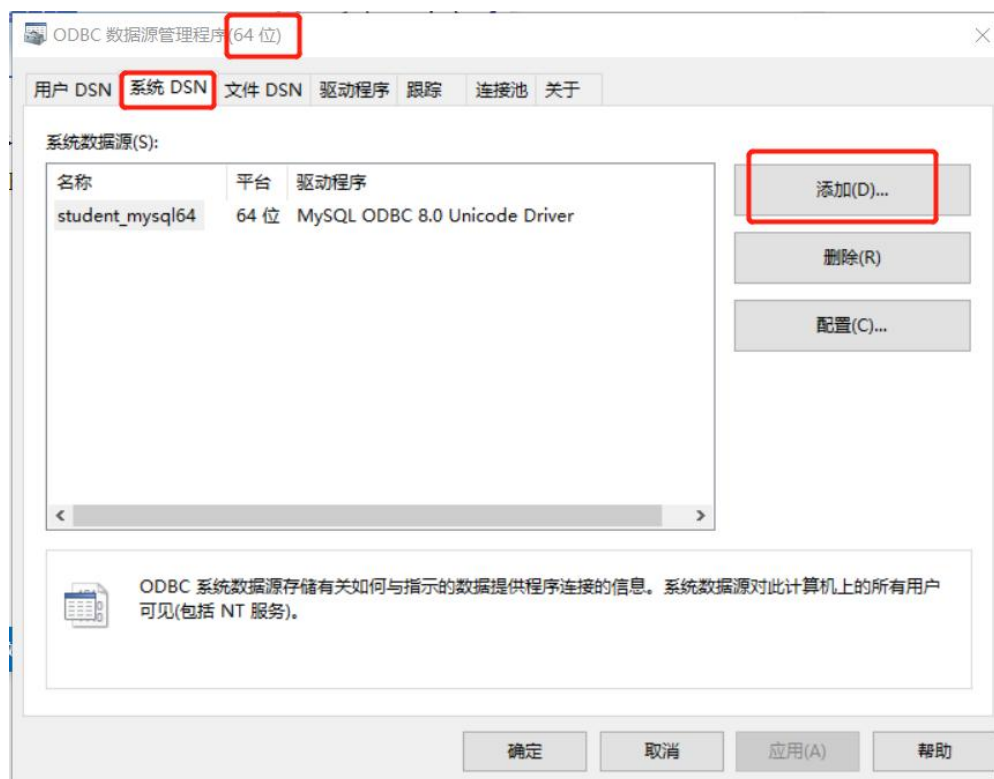
■ ODBC编程步骤

1. 环境准备：配置数据源DSN。（ODBC管理工具中完成）
2. 设置环境（利用编程接口设置准备好的DSN）。
3. 连接数据库（建立与数据库的连接）。
4. 操作数据库（增删改查等）。
5. 关闭数据库连接，释放占用资源。

ODBC驱动程序管理器

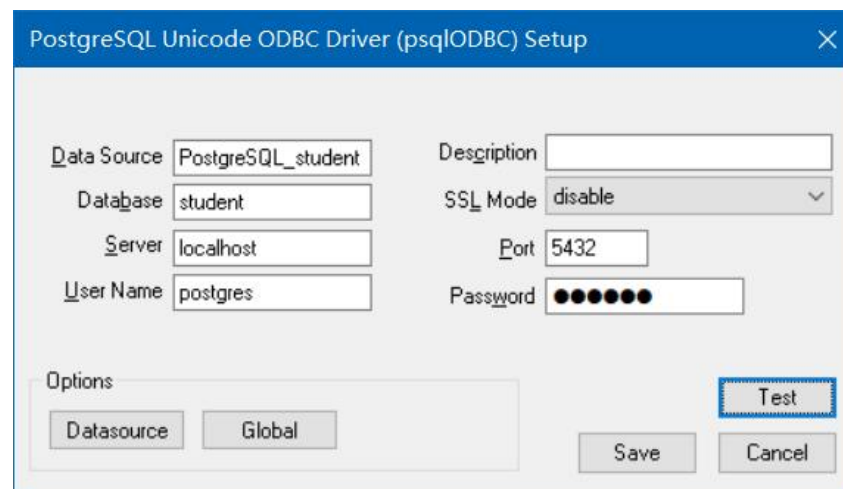
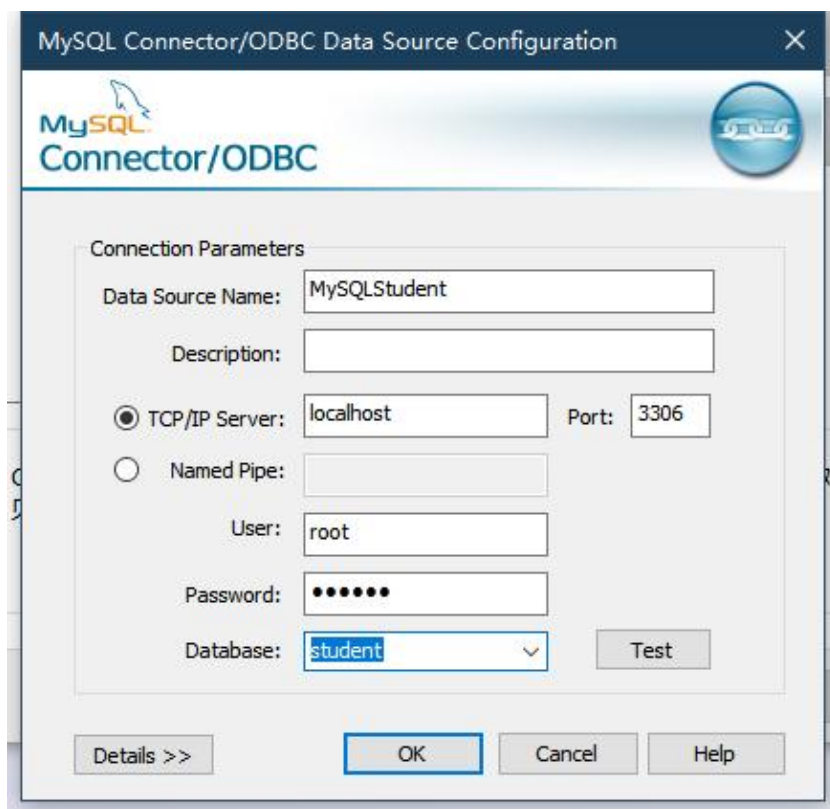
■ ODBC数据源的配置

⊗ 手工配置：控制面板 -> 管理工具 -> ODBC 数据源(64 位)或32位
(需要事先安装该MySQL的ODBC驱动)



■ ODBC数据源的配置

④ 手工配置：控制面板 -> 管理工具 -> 数据源(ODBC)



■ ODBC数据源的配置

⌚ 动态配置（通过程序中配置）

1) 通过修改注册表加载数据源：

用户数据源：HKEY_CURRENT_USER \ SOFTWARE \ ODBC \ ODBC. INI

系统数据源：HKEY_LOCAL_MACHINE \ SOFTWARE \ ODBC \ ODBC. INI

2) 通过ODBC API加载：SQLConfigDataSource函数



■ ODBC编程 - API的调用基本过程

④ 初始化ODBC

SQLAllocEnv

④ 与ODBC数据源建立连接

SQLAllocConnect , SQLConnect

④ 存取数据

SQLAllocStmt , SQLExecDirect

④ 检索结果集

SQLFetch , SQLGetData

④ 结束应用程序

SQLFreeStmt, SQLDisconnect, SQLFreeConnect, SQLFreeEnv



```
int main()
{
    HENV  henv;
    HDBC  hdbc;
    HSTMT hstmt;
    RETCODE retcode;
    SQLAllocEnv(&henv); /*分配环境句柄*/
    SQLAllocConnect(henv, &hdbc); /*分配连接句柄*/
    SQLConnect(hdbc, "student_mysql64", SQL_NTS, NULL, 0, NULL, 0); /*连接数据源*/
    SQLAllocStmt(hdbc, &hstmt); /*分配语句句柄*/
    retcode = SQLExecDirect(hstmt, "SELECT * FROM S", SQL_NTS);
    /*执行SQL语句*/
    /*结果集处理*/
    /*...*/
    SQLFreeStmt(hstmt, SQL_DROP); /*释放语句句柄*/
    SQLDisconnect(hdbc); /*断开数据源*/
    SQLFreeConnect(hdbc); /*释放连接句柄*/
    SQLFreeEnv(henv); /*当完成应用后, 释放环境句柄*/
    return 0;
}
```

如果不是特殊需求必须使用
ODBC, 推荐使用 **libmysql.dll**

8.1.3 ODBC编程 - C++调用ODBC示例



```
Cdatabase database;
Crecordset recordSet;
if(!database.IsOpen())
{
    database.Open(NULL, FALSE, FALSE,
        _T("ODBC;DSN=student_mysql64;UID=root;PWD=123456"));
    recordSet.m_pDatabase= &database;
}
CString strSQL;
strSQL.Format("select * from s");
recordSet.Open(CRecordset::forwardOnly, strSQL);
for(int i=0;i<recordSet.GetRecordCount();i++){
    CString temp;
    recordSet.GetFieldValue("score_id", temp);
    recordSet.GetFieldValue("student_name", temp);
    recordSet.MoveNext();
}
recordSet.Close();
database.Close();
```

```
#include <afxdb.h>
```

名称

- ADO.NET_CSharp
- C_MYSQL
- JDBC_Java
- ODBC_CSharp
- ODBC_MFC**
- Python_MySQL
- Java 连接 MySQL.doc
- VS2019中C调用MySQL.docx

例子：ODBC_MFC

8.1.概述

8.2.过程化SQL

8.3 JDBC与PDBC

8.4 连接池

8.5 基于MVC框架的数据库应用开发



- 定义部分：DECLARE 变量、常量、游标、异常等

注意：定义的变量、常量等只能在该基本块中使用，当基本块执行结束时，定义就不再存在。

- 执行部分

BEGIN

SQL语句、过程化SQL的流程控制语句

EXCEPTION /*异常处理部分*/

END

```
begin
  declare out_names varchar(100) default "";

  select * from course;
end
```

8.2.2 过程化SQL —— 变量与常量



- 全局变量（global，系统变量，不能自定义，可改值）
- 会话变量（session，系统变量，不能自定义，可改值）
- 用户变量（@var，用户可以自定义）
- 局部变量（参数，declare）

```
mysql> show global variables like '%timeout%';
```

Variable_name	Value
connect_timeout	10
delayed_insert_timeout	300
have_statement_timeout	YES
innodb_flush_log_at_timeout	1
innodb_lock_wait_timeout	50
innodb_rollback_on_timeout	OFF
interactive_timeout	28800
lock_wait_timeout	31536000
mysqlx_connect_timeout	30
mysqlx_idle_worker_thread_timeout	60
mysqlx_interactive_timeout	28800
mysqlx_port_open_timeout	0
mysqlx_read_timeout	30
mysqlx_wait_timeout	28800
mysqlx_write_timeout	60
net_read_timeout	30
net_write_timeout	60
replica_net_timeout	60
rpl_stop_replica_timeout	31536000
rpl_stop_slave_timeout	31536000
slave_net_timeout	60
wait_timeout	28800

22 rows in set, 1 warning (0.00 sec)

```
mysql> show session variables like '%timeout%';
```

Variable_name	Value
connect_timeout	10
delayed_insert_timeout	300
have_statement_timeout	YES
innodb_flush_log_at_timeout	1
innodb_lock_wait_timeout	50
innodb_rollback_on_timeout	OFF
interactive_timeout	28800
lock_wait_timeout	31536000
mysqlx_connect_timeout	30
mysqlx_idle_worker_thread_timeout	60
mysqlx_interactive_timeout	28800
mysqlx_port_open_timeout	0
mysqlx_read_timeout	30
mysqlx_wait_timeout	28800
mysqlx_write_timeout	60
net_read_timeout	30
net_write_timeout	60
replica_net_timeout	60
rpl_stop_replica_timeout	31536000
rpl_stop_slave_timeout	31536000
slave_net_timeout	60
wait_timeout	28800

22 rows in set, 1 warning (0.00 sec)

■ MySQL的用户变量

- @var, 以“@”开头, 可以作用于当前整个连接, 但是若当前连接断开后, 所定义的用户变量都会消失。可以在存储过程之间传递全局范围的变量。

赋值: `set @count=1;`
 `select count(id) into @count`
 `from items where price < 99;`

读取: `select @count;`

■ MySQL的局部变量

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
```

例：DECLARE语句部分声明变量，SET进行赋值。

```
declare names varchar(10) default '';
```

```
declare i int default 0;
```

```
set names = concat('test', '');
```

字符串拼接函数： concat

注意：

1) 支持SQL的数据类型

2) 给出DEFAULT，该变量进入BEGIN块时初始化为该值。

8.2.2 过程化SQL变量与常量 — 变量对比



西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

<https://blog.csdn.net/albertsh/article/details/103421646>

MySQL变量对比

操作类型	全局变量	会话变量	用户变量	局部变量 (参数)
文档常用名	global variables	session variables	user-defined variables	local variables
出现的位置	命令行、函数、存储过程	命令行、函数、存储过程	命令行、函数、存储过程	函数、存储过程
定义的方式	只能查看修改, 不能定义	只能查看修改, 不能定义	直接使用, @var 形式	declare count int(4);
有效生命周期	服务器重启时恢复默认值	断开连接时, 变量消失	断开连接时, 变量消失	出了函数或存储过程的作用域, 变量无效
查看所有变量	show global variables;	show session variables;	-	-
查看部分变量	show global variables like 'sql%';	show session variables like 'sql%';	-	-
查看指定变量	select @@global.sql_mode、 select @@max_connections;	select @@session.sql_mode、 select @@local.sql_mode、 select @@sql_mode;	select @var;	select count;
设置指定变量	set global sql_mode='';、 set @@global.sql_mode='';	set session sql_mode = ''、 set local sql_mode = ''、 set @@session.sql_mode = ''、	set @var=1;、 set @var:=101;、 select 100 into @var;	set count=1;、 set count:=101;、

■ MySQL流程控制：条件分支

➤ IF

```
IF search_condition THEN statement_list  
  [ELSEIF search_condition THEN statement_list] ...  
  [ELSE statement_list]  
END IF
```

➤ CASE WHEN

```
CASE case_value  
  WHEN when_value THEN  
statement_list  
  [WHEN when_value THEN  
statement_list] ...  
  [ELSE statement_list]  
END CASE
```

```
CASE  
  WHEN search_condition THEN  
statement_list  
  [WHEN search_condition  
THEN statement_list] ...  
  [ELSE statement_list]  
END CASE
```

8.2.3 过程化SQL流程控制 — IF示例



```
DELIMITER //
```

```
CREATE FUNCTION SimpleCompare(n INT, m INT)  
  RETURNS VARCHAR(20)
```

```
BEGIN
```

```
  DECLARE s VARCHAR(20);
```

```
  IF n > m THEN SET s = '>';
```

```
  ELSEIF n = m THEN SET s = '=';
```

```
  ELSE SET s = '<';
```

```
  END IF;
```

```
  SET s = CONCAT(n, ' ', s, ' ', m);
```

```
  RETURN s;
```

```
END //
```

```
DELIMITER ;
```


8.2.3 过程化SQL流程控制 — CASE示例



```
DELIMITER $$
CREATE PROCEDURE p()
BEGIN
    DECLARE v INT DEFAULT 1;

    CASE v
        WHEN 2 THEN SELECT v;
        WHEN 3 THEN SELECT 0;
        ELSE
            BEGIN
                END;
            END CASE;
    END;
    $$
```

```
select sno, cno, grade,
( case
  when grade <60 then '不及格'
  when grade <70 then '及格'
  when grade <85 then '良好'
  when grade >=85 then '优秀'
  else '未知' end)
as level
from sc;
```



MySQL流程控制：循环REPEAT

```
[begin_label:]  
REPEAT  
    statement_list  
UNTIL  
search_condition  
END REPEAT [end_label]
```

```
mysql> delimiter //  
mysql> CREATE PROCEDURE dorepeat(p1 INT)  
    BEGIN  
        SET @x = 0;  
        REPEAT  
            SET @x = @x + 1;  
        UNTIL @x > p1 END REPEAT;  
    END  
    //  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> CALL dorepeat(1000) //  
Query OK, 0 rows affected (0.00 sec)
```



■ MySQL流程控制：循环WHILE

```
[begin_label:]  
WHILE search_condition DO  
    statement_list  
END WHILE [end_label]
```

```
CREATE PROCEDURE dowhile()  
BEGIN  
    DECLARE v1 INT DEFAULT 5;  
  
    WHILE v1 > 0 DO  
        ...  
        SET v1 = v1 - 1;  
    END WHILE;  
END;
```

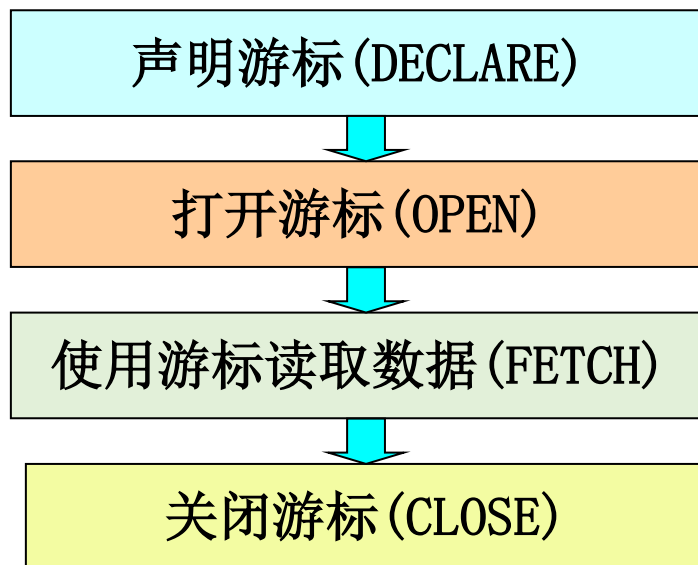
■ MySQL流程控制：循环LOOP

```
[begin_label:]  
LOOP  
    statement_list  
END LOOP [end_label]
```

```
CREATE PROCEDURE doiterate(p1 INT)  
BEGIN  
    label1: LOOP  
        SET p1 = p1 + 1;  
        IF p1 < 10 THEN  
            ITERATE label1;  
        END IF;  
        LEAVE label1;  
    END LOOP label1;  
    SET @x = p1;  
END;
```

■ 游标

若查询SQL只返回一条记录可以放入一个变量，当**返回多条记录**时，需使用游标逐行处理结果集。



8.2.4 过程化SQL —— 游标示例

delimiter \$\$

create procedure get_cs_s()

begin

declare out_names varchar(100) default '';

declare tmp varchar(100) default '';

declare p_sno varchar(5) default '';

declare p_sname varchar(10) default '';

declare done boolean default 0;

declare cursor_name cursor for select sno, sname from s where sdept='CS';

declare continue handler for sqlstate '02000' set done = 1; -- ER_SP_FETCH_NO_DATA

open cursor_name;

fetch cursor_name into p_sno, p_sname;

repeat

set tmp = concat(p_sno, '_', p_sname);

set out_names = concat(out_names, tmp, ' ');

fetch cursor_name into p_sno, p_sname;

until done

end repeat;

close cursor_name;

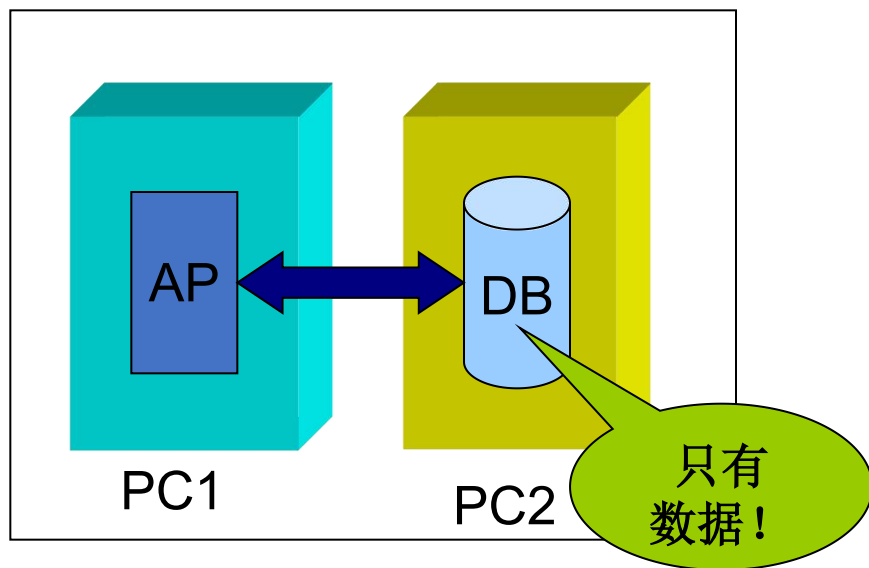
select out_names;

end \$\$

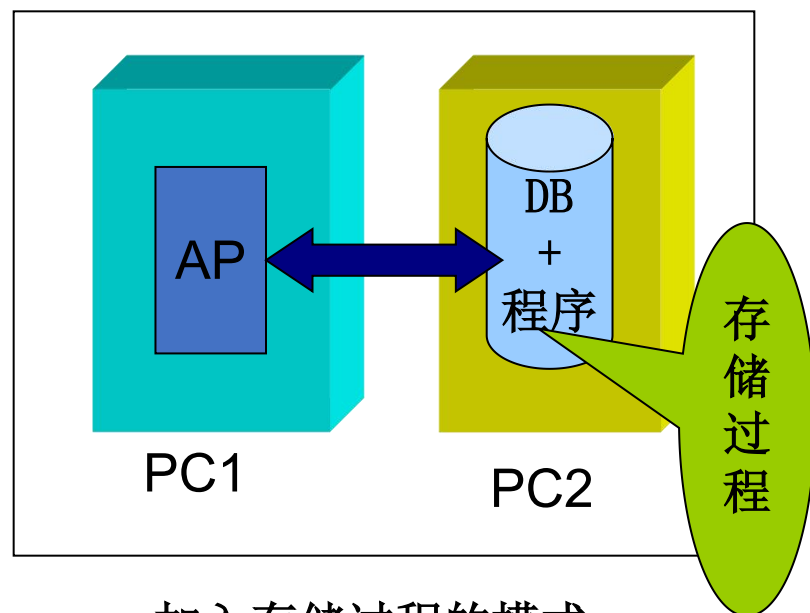
delimiter ;

1. 基本概念

存储过程/存储函数是一段在数据库服务器上执行的程序（被命名、编译和保存在数据库中）。它在服务器端对数据库记录进行处理，再把结果返回给客户端。



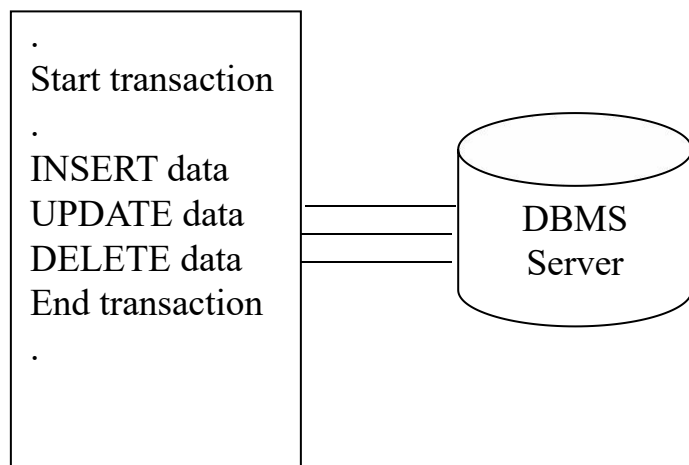
传统模式



加入存储过程的模式

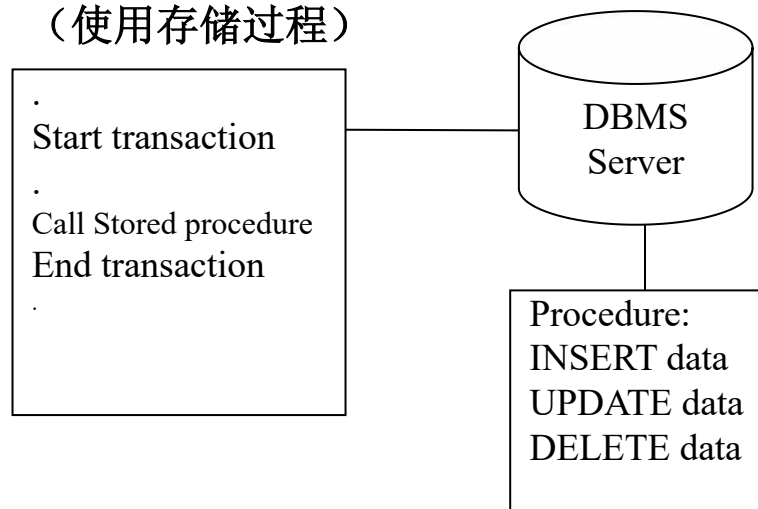
1. 优点

客户端应用
(不使用存储过程)



(a)

客户端应用
(使用存储过程)



(b)

1. 减少网络流量，充分利用服务端的强大计算能力。
2. 封装业务逻辑，数据结构变化时对应用程序的影响减至最小。
3. 增强代码的共享性和重用性，安全性。

2. 存储过程/存储函数 – 基本语法

```
delimiter $$  
create procedure 存储过程名(参数)  
Begin  
    方法体  
end  
$$  
delimiter;
```

```
call 存储过程名
```

```
delimiter $$  
create function 函数名(参数类型)  
returns 返回类型  
Begin  
    方法体  
    return 数据类型;  
end  
$$  
delimiter;
```

```
select 函数名
```

2. MYSQL存储过程/存储函数完整语法

```
1 CREATE
2   [DEFINER = user]
3   PROCEDURE sp_name ([proc_parameter[,...]])
4   [characteristic ...] routine_body
5
6 CREATE
7   [DEFINER = user]
8   FUNCTION sp_name ([func_parameter[,...]])
9   RETURNS type
10  [characteristic ...] routine_body
11
12 proc_parameter:
13   [ IN | OUT | INOUT ] param_name type
14
15 func_parameter:
16   param_name type
17
18 type:
19   Any valid MySQL data type
20
21 characteristic: {
22   COMMENT 'string'
23   | LANGUAGE SQL
24   | [NOT] DETERMINISTIC
25   | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
26   | SQL SECURITY { DEFINER | INVOKER }
27 }
28
29 routine_body:
30   Valid SQL routine statement
```

<https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html>



■ 2. 存储过程示例

```
1  mysql> delimiter //
```

```
2
```

```
3  mysql> CREATE PROCEDURE citycount (IN country CHAR(3), OUT cities INT)
```

```
4      BEGIN
```

```
5          SELECT COUNT(*) INTO cities FROM world.city
```

```
6          WHERE CountryCode = country;
```

```
7      END//
```

```
8  Query OK, 0 rows affected (0.01 sec)
```

```
9
```

```
10 mysql> delimiter ;
```

```
11
```

```
12 mysql> CALL citycount('JPN', @cities); -- cities in Japan
```

```
13 Query OK, 1 row affected (0.00 sec)
```

```
14
```

```
15 mysql> SELECT @cities;
```

```
16 +-----+
```

```
17 | @cities |
```

```
18 +-----+
```

```
19 |      248 |
```

```
20 +-----+
```

```
21 1 row in set (0.00 sec)
```

```
22
```

■ 2. 存储函数示例

```
mysql> CREATE FUNCTION hello (s CHAR(20))  
mysql> RETURNS CHAR(50)  
        RETURN CONCAT('Hello, ', s, ' !');  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT hello('world');
```

```
+-----+  
| hello('world') |  
+-----+  
| Hello, world!  |  
+-----+
```

```
1 row in set (0.00 sec)
```

<https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html>

3. MySQL的参数传递

- 存储过程：IN，OUT，INOUT类型
- 函数：所有参数为IN类型

```
1 • use student;
2
3 delimiter $
4 • create procedure p1(in n int)
5 begin
6     declare total int default 0;
7     declare num int default 0;
8     while num < n do
9         set num:=num+1;
10        set total:=total+num;
11    end while;
12    select total;
13 end$
```

```
call p1(10);
```

```
1 • use student;
2
3 delimiter $
4 • create procedure p2(in n int, out total int)
5 begin
6     declare num int default 0;
7     set total:=0;
8     while num < n do
9         set num:=num+1;
10        set total:=total+num;
11    end while;
12 end$
```

```
call p2(10, @sum);
select @sum;
```



■ 3. MySQL参数传递: INOUT类型

```
delimiter $$  
create procedure p3(inout total int)  
begin  
    set total:=total + 10;  
end  
$$
```

```
set @currentCnt=10;  
call p3(@currentCnt);  
select @currentCnt;
```



■ 3. MySQL – 返回值（函数）

```
DELIMITER $  
CREATE FUNCTION myfun() RETURNS INT  
BEGIN  
    DECLARE C INT DEFAULT 0;  
    SELECT COUNT(*) INTO C FROM userinfo;  
    RETURN C;  
END $  
  
SELECT myfun()  
$
```



■ 存储过程实际用途举例

- 学生数据库：统计不同分数段人数
（指定课程号和开课学期）
- 人力资源数据库：为不同级别的人按一定规则涨工资
- 云计算收费管理数据库：
按照使用的资源情况，针对每个用户动态计算费用。

某气象站有一张表 `temperature`，每天在2点，8点，14点，20点自动采集温度

	列名	数据类型	允许 Null 值
▶	Year	smallint	<input type="checkbox"/>
	Month	smallint	<input type="checkbox"/>
	Day	smallint	<input type="checkbox"/>
	T02	float	<input checked="" type="checkbox"/>
	T08	float	<input checked="" type="checkbox"/>
	T14	float	<input checked="" type="checkbox"/>
	T20	float	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

	Year	Month	Day	T02	T08	T14	T20
1	2008	12	1	16	15	14	13
2	2008	12	2	17	15	14	13
3	2008	12	3	18	15	14	13
4	2008	12	4	19	15	14	13
5	2008	12	5	20	15	14	13
6	2008	12	6	NULL	15	14	13
7	2008	12	7	NULL	15	14	13

要求：指定年和月时，编写一个自定义函数：求该年该月的平均气温。



某气象站有一张表`temperature`，每天在2点，8点，14点，20点自动采集温度

```
DELIMITER $
CREATE FUNCTION temp_avg(myyear int, mymonth int)
RETURNS real
BEGIN
    DECLARE temp real DEFAULT 0;
    SELECT (SUM(T02)+SUM(T08)+SUM(T14)+SUM(T20)) /
           (Count(T02)+Count(T08)+Count(T14)+Count(T20)) INTO temp
    FROM Temperature WHERE Year = myyear AND Month = mymonth;
    return temp;
END $
```

■ 1. 触发器 (*trigger*) 概念

触发器： 一种特殊的存储过程，它在满足某个特定条件时 **自动触发** 执行。它是依附于表的数据库对象。



触发器用途：

- 可以完成比CHECK约束更复杂的限制，但效率可能不及CHECK约束
- 对数据库进行级联修改
- 检测改变后，触发一些自定义的功能，如回滚、审计等

■ 1. 触发器定义 - 普适

```
CREATE TRIGGER trigger_name {BEFORE|AFTER|INSTEAD OF}  
{ INSERT | UPDATE | DELETE | TRUNCATE}  
ON table|view  
[ FOR [ EACH ] { ROW | STATEMENT } ]  
EXECUTE PROCEDURE function_name ( arguments )
```

- *trigger_name*: 触发器的名称;
- BEFORE|AFTER|INSTEAD OF: 触发器类型;
- {INSERT | UPDATE | DELETE | TRUNCATE}:
激活触发器的数据操作语句;
- *table|view*: 说明了定义触发器的表或视图;
- FOR [EACH] { ROW | STATEMENT }: 语句级别或者行级别
- *function_name*: 事先定义好的该动作触发是需要执行的函数。

1. 触发器定义 - MySQL

```
1  CREATE
2      [DEFINER = user]
3      TRIGGER trigger_name
4      trigger_time trigger_event
5      ON tbl_name FOR EACH ROW
6      [trigger_order]
7      trigger_body
8
9      trigger_time: { BEFORE | AFTER }
10
11     trigger_event: { INSERT | UPDATE | DELETE }
12
13     trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

<https://dev.mysql.com/doc/refman/8.0/en/create-trigger.html>

2. 触发器的分类 (DML)

INSERT触发器

UPDATE触发器

DELETE触发器

TRUNCATE触发器

BEFORE触发器

AFTER触发器

INSTEAD OF触发器

在定义了触发器的表上发生修改操作时，会自动为触发器的运行而派生两个记录：

- 1) Old — 存放旧记录
(for: delete, update)
- 2) New — 存放新记录
(for: insert, update)

REFERENCING子句：将对应记录声明为一个表格使用

执行 INSTEAD OF 触发器代替通常的增删改等触发动作。
PG: 仅支持视图上的该触发器。

■ 2. 触发器的分类（DML）

When	Event	Row-level	Statement-level (mysql 不支持)
BEFORE/AFTER	INSERT/UPDATE/ DELETE	Table	Table and View
BEFORE/AFTER	TRUNCATE	-	Table
INSTEAD OF (mysql不支持)	INSERT/UPDATE/ DELETE	View	

TRUNCATE: 清空表中的数据，但是不删除表结构

Trigger可以暂停或者启用

■ 3. 触发器修改/删除

■ 修改：不支持

查看触发器：SHOW TRIGGERS

■ 删除

DROP TRIGGER database.trigger_name (mysql)

DROP TRIGGER trigger_name ON tablename (其他多数DBMS)

PG: 支持Event trigger

(针对DDL: CREATE, ALTER, DROP, SECURITY LABEL, COMMENT, GRANT or REVOKE)

ddl_command_star

ddl_command_end

table_rewrite

■ 4. MySQL中的Event (不是一种通常意义的触发器)

MySQL: create event 时间定时触发执行的SQL

```
1  CREATE EVENT e_hourly
2      ON SCHEDULE
3      EVERY 1 HOUR
4      COMMENT 'Clears out sessions table each hour.'
5      DO
6      DELETE FROM site_activity.sessions;
```

<https://dev.mysql.com/doc/refman/8.0/en/create-event.html>

■ 5. 触发器示例1 - MySQL

```
CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
```

例1：创建一个触发器，当每插入一条记录时累加金额

```
CREATE TRIGGER ins_sum  
BEFORE INSERT ON account  
FOR EACH ROW  
SET @sum = @sum + NEW.amount;
```

```
1 • SET @sum = 0;  
2 • INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);  
3 • SELECT @sum AS 'Total amount inserted';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	Total amount inserted
▶	1852.48


```
DROP TRIGGER ins_sum;
```

■ 5. 触发器示例2 - MySQL

例2：创建一个触发器，当删除学生表的一个记录时，从选课表中自动删除该学生的选课信息。

```
use student;
drop trigger if exists del_s;

delimiter //
create trigger del_s
after delete on s
for each row
begin
    set @sno = OLD.sno;
    delete from sc where sno =@sno;
end
//
```



一个删除触发器

8.2.6 触发器 —— 同类型多个触发器



同一个表上可以存在两个同类型同事件的触发器（如两个before update），**顺序是按照创建顺序被触发**；如果希望显示指定触发器的顺序，可以通过：precedes 或者 follows关键字指定

```
1  mysql> CREATE TRIGGER ins_transaction BEFORE INSERT ON account
2          FOR EACH ROW PRECEDES ins_sum
3          SET
4          @deposits = @deposits + IF(NEW.amount>0,NEW.amount,0),
5          @withdrawals = @withdrawals + IF(NEW.amount<0,-NEW.amount,0);
6  Query OK, 0 rows affected (0.01 sec)
```

<https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html>

■ 5. 触发器示例2 - PostgreSQL:

```
CREATE TRIGGER s_d_trigger AFTER DELETE ON s
FOR EACH ROW
EXECUTE PROCEDURE s_delete_tfun();
```

```
CREATE OR REPLACE FUNCTION s_delete_tfun()
RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM sc WHERE sno = OLD.SNO;
    raise notice '删除了一个学生元组以及他的选课记录';
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```



■ 5. 触发器示例3 - SQL Server . . .

一个instead
of 触发器

例3：创建一个触发器，对表tb插入前将该表数据清空。

SQL Server 触发器例子

```
create trigger test
on tb
instead of insert
as
begin
    delete from tb                /* 清除原始数据*/
    insert into tb select * from inserted /*把新数据加入表中*/
end
```

SQL Server虚拟表：
deleted/inserted

MySQL不支持 instead of 触发器

➤ 触发器实际用途举例

- 学生数据库：插入学生时，自动更新学生人数的统计信息表的人数字段。
- 图书数据库：毕业离校(删除学生记录)时，检查并删除其相关的借还书记录。
- 银行数据库：账户信息表和交易信息表。

```
create table bank  --账户信息表
(
    customerName char(8) not null, --顾客姓名
    cardID char(10) not null,  --卡号
    currnetMoney Money not null --当前余额
)
go
create table transInfo  --交易信息表
(
    cardID char(10) not null,  --卡号
    transType char(4) not null, --交易类型(存入/支取)
    transMoney money not null, --交易金额
    transDate datetime not null --交易日期
)
```

1. 交易信息表中插入一条交易信息时，应该自动更新账户信息表对应账余额以及流水表信息。
2. 跟踪用户的交易，交易金额超过10000元，则取消交易，并给出错误提示。

- 扩展SQL
- 高级语言访问数据库技术
 - 嵌入式SQL
 - 调用DBMS的动态链接库
 - ODBC
- 过程化SQL（变量与流程控制）
- 存储过程/存储函数/触发器

理论课实践作业：

1) 8.1.2小节中任务1-任务3的问题在MYSQL中的具体实现， 三选一

2) 8.2小节例子程序 P64
三选一

提交方式（10/29日前）：

zhaoxn@nwpu.edu.cn

预习作业：

实验五中的存储过程、触发器和存储函数

