# 数据库系统之二
# --数据库语言

# 第10讲 嵌入式SQL语言之动态SQL


战德臣

哈尔滨工业大学　教授.博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

**Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology**

# 本讲学习什么?

战德臣 教授

## 基本内容

1. 动态SQL的概念和作用
2. SQL语句的动态构造
3. 动态SQL语句的执行方式
4. 数据字典与SQLDA
5. ODBC/JDBC简介?

## 重点与难点

- 熟练掌握SQL语句的动态构造技巧
- 了解数据字典的作用，掌握其使用技巧
- 理解ODBC/JDBC的工作原理

# 动态SQL的概念和作用

战德臣

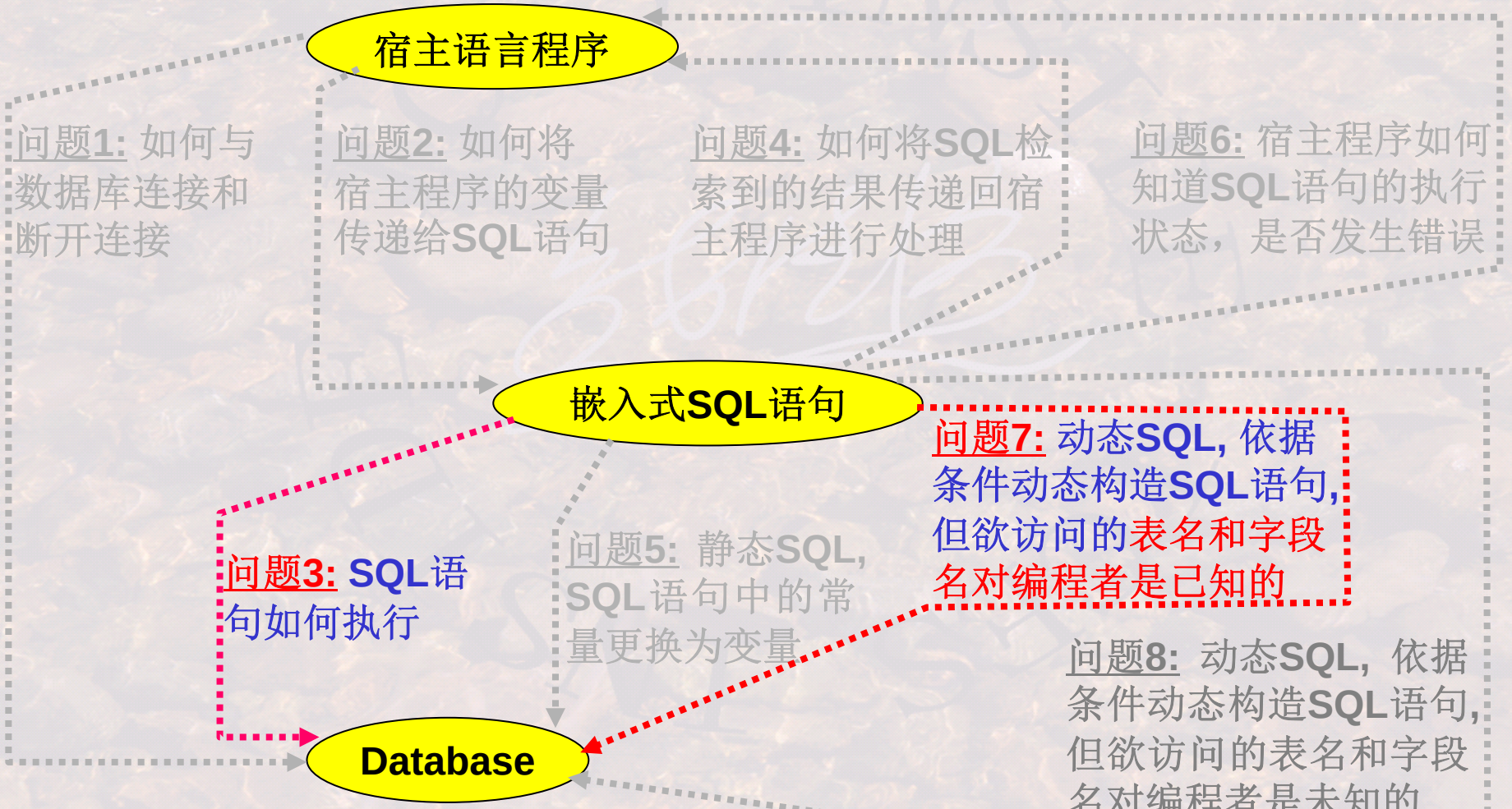
哈尔滨工业大学　教授.博士生导师
黑龙江省教学名师
教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent
Computing for Enterprises & Services,
Harbin Institute of Technology

# 动态SQL的概念和作用
## (1)拟解决的问题

战德臣 教授

**高级语言(语句)→嵌入式SQL语句→DBMS←→DB**



# 动态SQL的概念和作用
## (2)动态SQL的概念和作用

战德臣 教授

➢动态SQL是相对于静态SQL而言的

**静态SQL示例**

这里使用变量不带冒号

```
SpecName = '张三';
exec sql select Sno, Sname, Sclass into :vSno, :vSname, :vSclass from
Student where Sname= :SpecName ;
或
exec sql declare cur_student cursor for
        select Sno, Sname, Sclass from Student where Sclass= :vClass
        order by Sno for read only ; /*定义*/
exec sql open cur_student ; /*执行*/
```

这里使用变量要带冒号

➢静态SQL**特点**：SQL语句在程序中已经按要求写好，只需要把一些参数通过变量(**高级语言程序语句中不带冒号**) 传送给嵌入式SQL语句即可(**嵌入式SQL语句中带冒号**)

战德臣 教授

# 动态SQL示例

动态SQL与静态SQL的差异?

```
#include <stdio.h>
exec  sql  include  sqlca;

exec sql begin declare section;
   char  user_name[ ] = “Scott”;  char  user_pwd[ ] =“tiger”;
   char  sqltext[ ] = “ delete  from  customers  where  cid = \‘c006\’ ” ;
exec sql end declare section;

int main()
{
    exec sql whenever  sqlerror  goto report_error;
    exec sql connect  :user_name  identified by  :user_pwd;
    exec  sql   execute  immediate  :sqltext;
    exec sql commit  release;    return  0;
report_error:   print_dberror();     exec  sql  rollback  release;      return 1;
}
```

动态SQL语句的构造

动态SQL语句如何被执行

➢动态SQL**特点**：SQL语句可以在程序中动态构造，形成一个字符串，如上例sqltext，然后再交给DBMS执行，交给DBMS执行时仍旧可以传递变量

# 动态SQL的概念和作用
# (2)动态SQL的概念和作用

战德臣 教授

➢动态构造SQL语句是应用程序员必须掌握的重要手段

**示例：编写由用户确定检索条件的应用程序**

请输入 Y_N _Y_ 姓名 张三___
Y_N _N_ 年龄在 ____和____之间
Y_N _Y_ 班级 035103____
… …

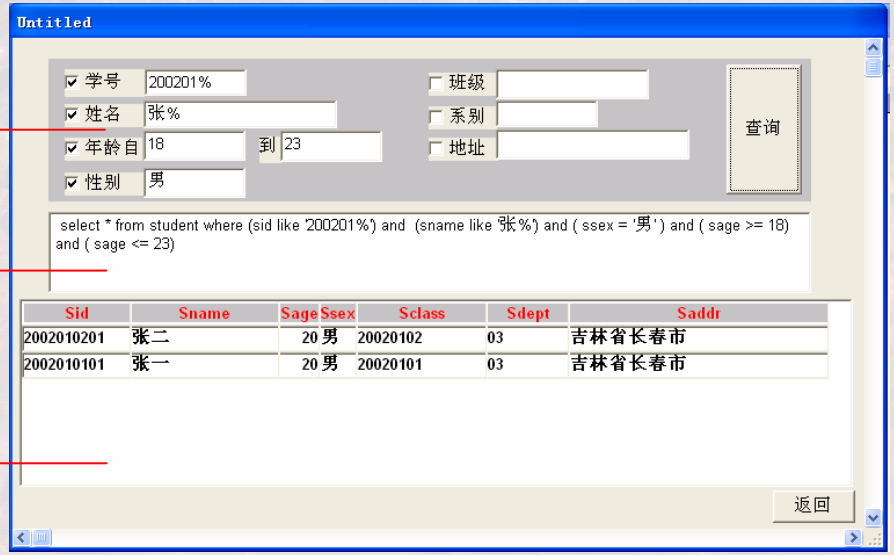
为什么需要动态SQL?

怎样按用户给定的条件构造SQL语句?

用户输入检索条件

显示动态SQL语句构造结果(字符串)

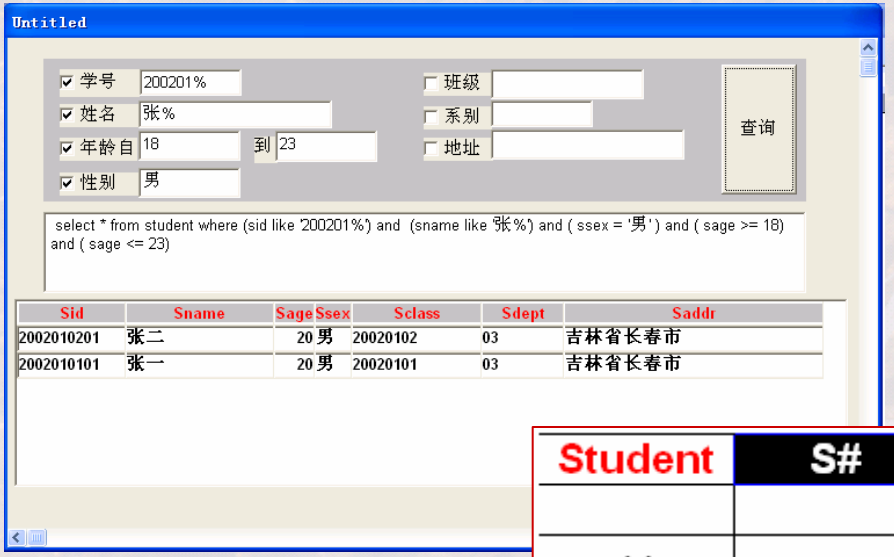
显示动态SQL语句执行结果



# 动态SQL的概念和作用
# (2)动态SQL的概念和作用

战德臣 教授

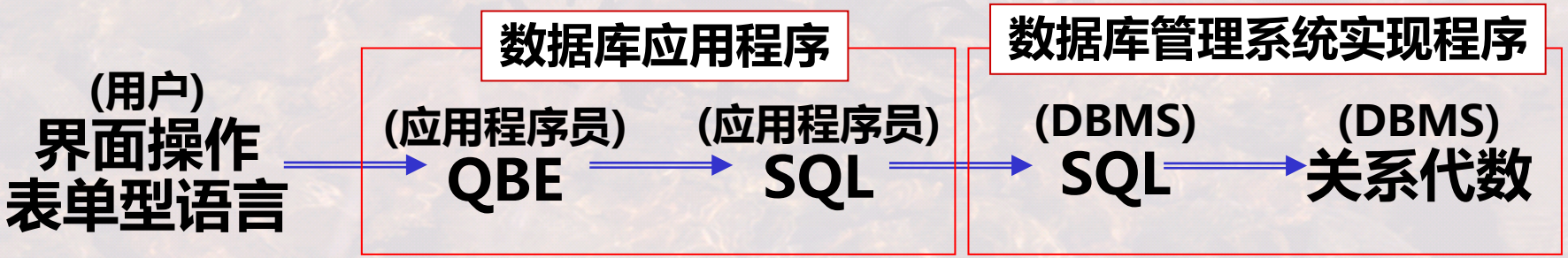
➢如何设计用户的操作界面?




**QBE：Query by Example**

**(See also “关系模型之关系演算”)**

| Student | S# | Sname | Ssex | Sage | D# | Sclass |
|---|---|---|---|---|---|---|
| | | P.X | Male | <17 | | |
| V | | P.Y | Female | >20 | | |
| | | | | | | |



# SQL语句的动态构造-示例1

战德臣

哈尔滨工业大学　教授.博士生导师
黑龙江省教学名师
教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent
Computing for Enterprises & Services,
Harbin Institute of Technology

# SQL语句的动态构造-示例1
## (1)程序背景

战德臣 教授

- 已知关系：Customers(Cid, Cname, City, discnt)
- 从Customers表中删除满足条件的行

**Delete customer rows with ALL of the following properties:**

**Y_N_ Y _ Customer name is Mbale Jamson___**

**Y_N _N_ Customer is in city _____**

**Y_N _N_ Customer discount is in range from _____ to _____**

**...(and so on)**

- 假设：用户界面上的输入存在下面的变量中
  - char Vcname[ ];
  - char Vcity[ ];
  - double range_from, range_to;
  - int Cname_chose, City_chose, Discnt_chose;

请按用户输入构造相应的SQL语句

如何构造这个动态SQL语句？

大家编一下...

# SQL语句的动态构造-示例1
## (2)程序构造

战德臣 教授

```
#include  <stdio.h>
#include "prompt.h"
char  Vcname[ ];
char  Vcity[ ];
double range_from, range_to;
int Cname_chose, City_chose, Discnt_chose;
Cname_chose = 0;  City_chose = 0; Discnt_chose = 0;
int     sql_sign = 0;
char  continue_sign[ ];

exec  sql  include sqlca;
exec  sql  begin declare section;
        char user_name[20],user_pwd[20];
        char sqltext[ ]="delete from customers where ";
exec sql end declare section;
```

------程序变量声明

------SQLCA:  SQL Communication  Area

-----The Declare Section

```
•char  Vcname[ ];
•char  Vcity[ ];
•double range_from, range_to;
•int Cname_chose, City_chose, Discnt_chose;
```

# SQL语句的动态构造-示例1
## (2)程序构造

战德臣 教授

```
int main()
{
    exec sql whenever sqlerror goto report_error;   ------SQL错误捕获语句
    strcpy(user_name, “poneilsql”);
    strcpy(user_pwd, “XXXX”);
    exec sql connect   :user_name   identified      ------SQL Connect
                by   :user_pwd;

  while(1) {
      memset(Vcname, ‘\0’, 20);
      memset(Vcity, ‘\0’, 20);                      ------初始化
      if (GetCname(Vcname))                         ------获取Cname值
            Cname_chose = 1;
      if (GetCity(Vcity))                           ------获取City值
            City_chose = 1;
      if (GetDiscntRange(&range_from, &range_to))   ------获取Discnt区间值
            Discnt_chose = 1;
```

**Customers(Cid, Cname, City, discnt)**

- char Vcname[ ];
- char Vcity[ ];
- double range_from, range_to;
- int Cname_chose, City_chose, Discnt_chose;

# SQL语句的动态构造-示例1
## (2)程序构造

战德臣 教授

这里才开始构造SQL的字符串

```
if (Cname_chose){
      sql_sign = 1;
      strcat(sqltext, "Cname = \'");
      strcat(sqltext, Vcname);
      strcat(sqltext, "\'");
 }
```

-----如果选择了Cname，构造sqltext

Char型属性条件如何构造?

```
if (City_chose){
   sql_sign = 1;
   if(Cname_chose)
       strcat(sqltext, " and City = \'");
   else
        strcat(sqltext, " City = \'");
   strcat(sqltext, Vcity);
   strcat(sqltext, "\'");
}
```

-----如果选择了City，构造sqltext

**Customers(Cid, Cname, City, discnt)**

```
•char  Vcname[ ];
•char  Vcity[ ];
•double range_from, range_to;
•int Cname_chose, City_chose, Discnt_chose;
```

# SQL语句的动态构造-示例1
## (2)程序构造

战德臣 教授

数值型属性条件如何构造?

这里在继续构造SQL的字符串

```
if (Discnt_chose){
    sql_sign =1;
    if(Cname_chose=0 and City_chose = 0)
        strcat(sqltext, " discnt >");
    else
        strcat(sqltext, " and (discnt >");
    strcat(sqltext, dtoa(range_from));
    strcat(sqltext, " and discnt<");
    strcat(sqltext, dtoa(range_to));
    strcat(sqltext, ")");
}

if(sql_sign){
    exec sql execute immediate :sqltext;
    exec sql commit work;
}
```

------如果选择了Discnt区间值，构造sqltext

-----动态SQL语句执行

------SQL Commit Work

**Customers(Cid, Cname, City, discnt)**

```
•char Vcname[ ];
•char Vcity[ ];
•double range_from, range_to;
•int Cname_chose, City_chose, Discnt_chose;
```

# SQL语句的动态构造-示例1
## (2)程序构造

战德臣 教授

```
        scanf("continue (y/n) %1s", continue_sign)
        if(continue_sign = "n"){
            exec  sql   commit release;      ------SQL Commit Work and  Disconnect
            returen 0;
        }
      }      ------while 结束

report_error:                                ------SQL  Rollback  Work and  Disconnect
   print_dberror();
   exec sql rollback release;
   return 1;
}          ------main 结束
```

**Customers(Cid, Cname, City, discnt)**

```
•char  Vcname[ ];
•char  Vcity[ ];
•double range_from, range_to;
•int Cname_chose, City_chose, Discnt_chose;
```

# SQL语句的动态构造-示例1
# (3)动态SQL语句构造小结

战德臣 教授

```
char sqltext[]= “delete from customers where ” ;
…
If (Cname_chose) {   sql_sign = 1;   strcat(sqltext, “Cname = \’”);
          strcat(sqltext, Vcname);   strcat(sqltext, “\’”);   }
if (City_chose) {   sql_sign = 1;
          if (Cname_chose)
              strcat(sqltext, “ and City = \’”);
          else
            strcat(sqltext, “ City = \’”);
            strcat(sqltext, Vcity);
            strcat(sqltext, “\’”);     }
if (Discnt_chose)   {
            sql_sign =1;
            if (Cname_chose=0 and City_chose = 0)
                  strcat(sqltext, “ discnt >”);
            else
                   strcat(sqltext, “ and (discnt >”);
            strcat(sqltext, dtoa(range_from));
            strcat(sqltext, “ and discnt<”);
            strcat(sqltext, dtoa(range_to));
            strcat(sqltext, “)”);       }
```

**写好基本部分**

Sqltext=“delete from customers where ”

基本语句

Sqltext=Sqltext+ “Cname= ‘”+ Vcname + “’”

字符串需加引号

Sqltext=Sqltext + “City= ‘”+ Vcity + “ ’ ”

或者 Sqltext=Sqltext + “ and City= ‘”+ Vcity + “ ’ ”

添加逻辑运算符

Sqltext=Sqltext + “Discnt > ”+ dtoa(range_from)

数值变量转换成字符串

# SQL语句的动态构造-示例2

战德臣

哈尔滨工业大学　教授.博士生导师
黑龙江省教学名师
教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent
Computing for Enterprises & Services,
Harbin Institute of Technology

# SQL语句的动态构造-示例2
## (1)程序背景

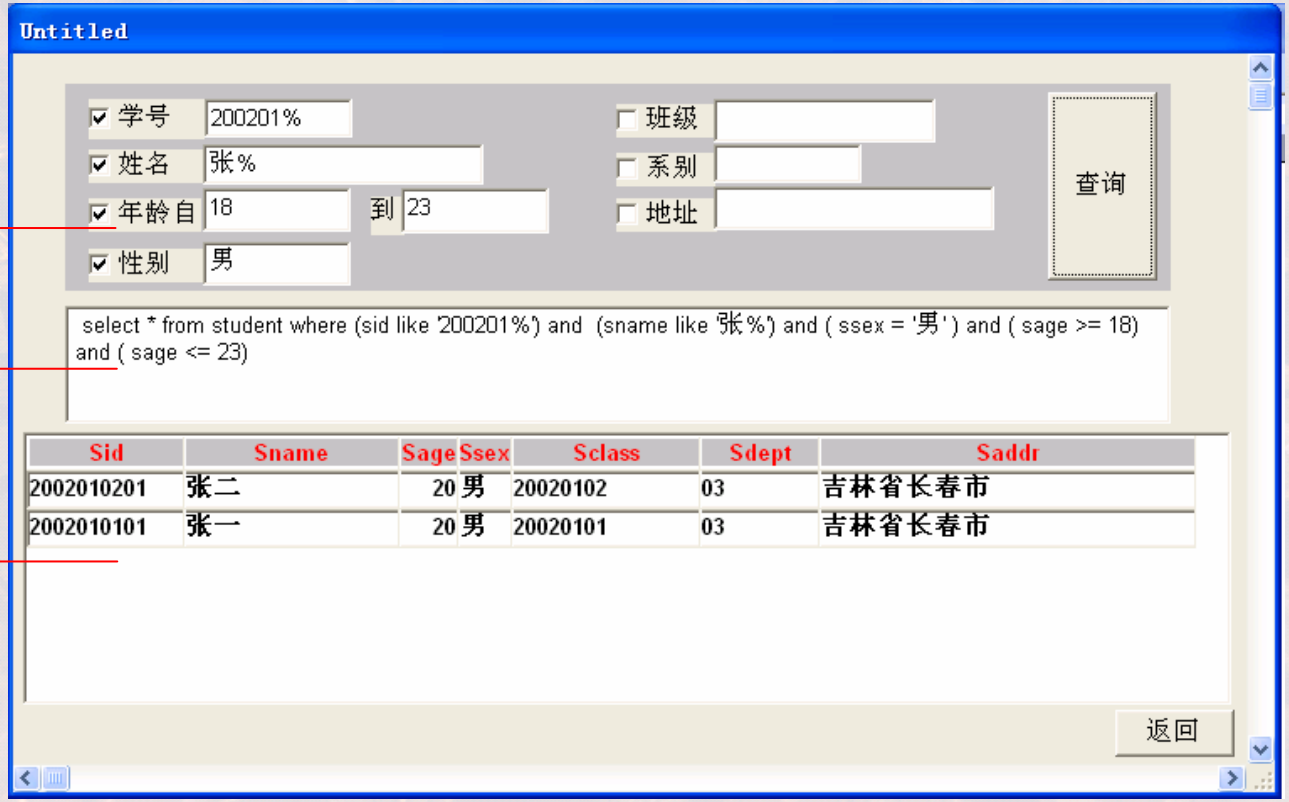
战德臣 教授

**示例: 编写程序，依据用户输入条件构造SQL语句并执行**

**用户输入检索条件**

**显示动态SQL语句构造结果(字符串)**

**显示动态SQL语句执行结果**



# SQL语句的动态构造-示例2
## (1)程序背景

战德臣 教授

## 界面要素及其背后的变量简介

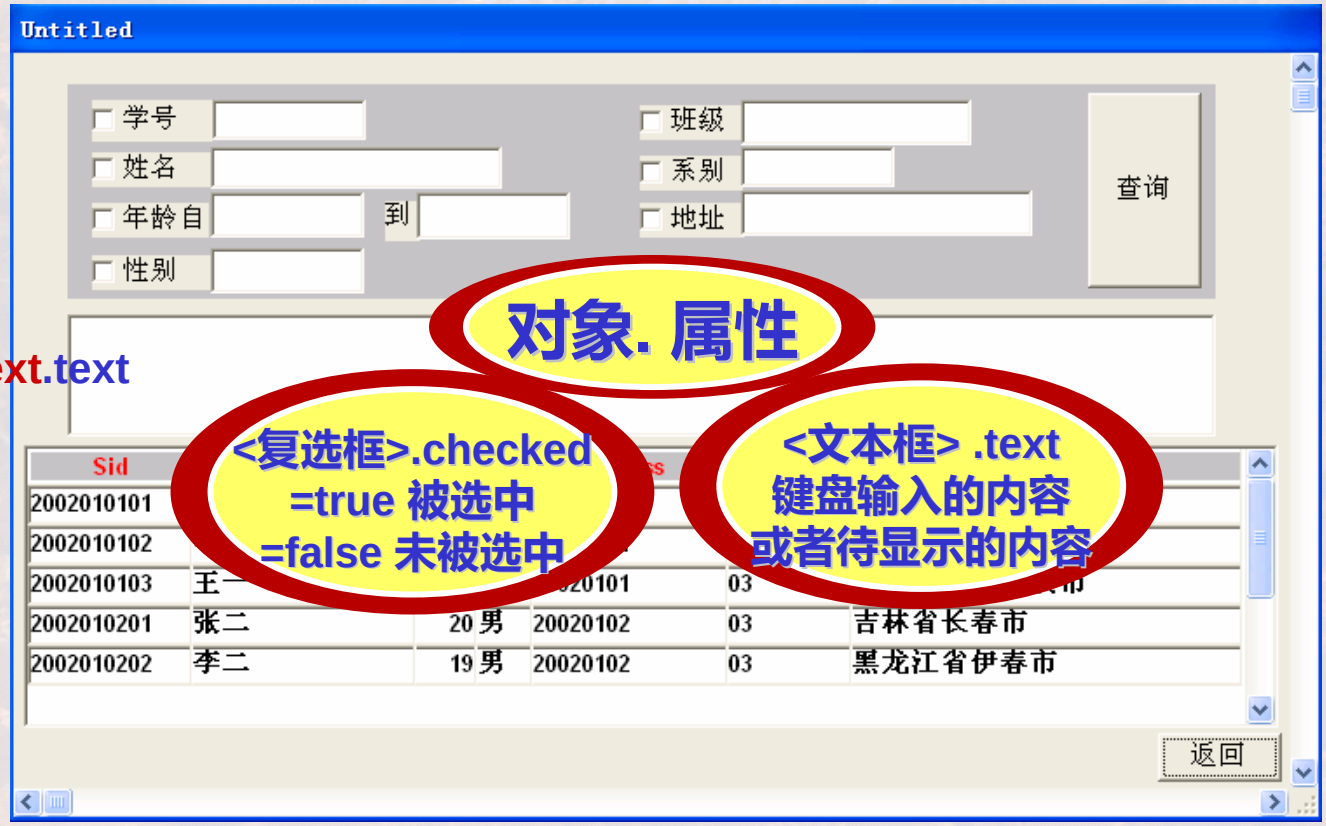
cbx_id.checked
cbx_name.checked
cbx_agest.checked
cbx_sex.checked

sle_id.text
sle_name.text
sle_agest.text
sle_ageed.text
sle_sex.text

cbx_class.checked
cbx_dept.checked
cbx_addr.checked

sle_class.text
sle_dept.text
sle_addr.text

sle_sqltext.text

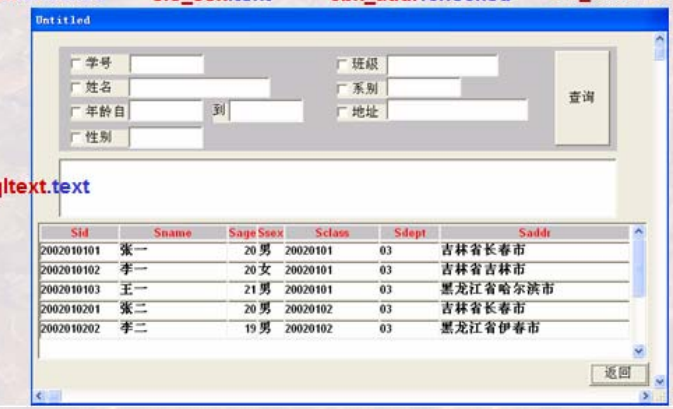


# SQL语句的动态构造-示例2
## (2)动态SQL语句的构造—SQL字符串

战德臣 教授

## SQL字符串的构造

sle_id.text
cbx_id.checked　sle_name.text
cbx_name.checked　sle_agest.text　cbx_class.checked　sle_class.text
cbx_agest.checked　sle_ageed.text　cbx_dept.checked　sle_dept.text
cbx_sex.checked　sle_sex.text　cbx_addr.checked　sle_addr.text




怎样将输入和SQL语句合并在一起?

Char型属性条件如何构造?

写好基本部分

```
string str_temp
int   firstflag = 0
str_temp = ""

// dw_2.setfilter("")
Str_temp = " select * from student where "
if (cbx_id.checked = true and len(trim(sle_id.text))>0) then
   str_temp = str_temp + "(sid like '" + trim(sle_id.text)+ "')"
   firstflag = 1
end if
if (cbx_name.checked = true and len(trim(sle_name.text))>0) then
   if (firstflag = 1) then
      str_temp = str_temp + " and  (sname like '" + trim(sle_name.text)+ "')"
   else
      str_temp = str_temp + " ( sname like '" + trim(sle_name.text)+ "' )"
      firstflag = 1
   end if
end if
if (cbx_sex.checked = true and len(trim(sle_sex.text))>0) then
   if (firstflag = 1) then
      str_temp = str_temp + " and ( ssex = '" + trim(sle_sex.text)+ "' )"
   else
      str_temp = str_temp + " (ssex = '" + trim(sle_sex.text)+ "')"
      firstflag = 1
   end if
end if
```

基本语句

字符串需加引号

添加逻辑运算符

trim()去两边的空格，len()求长度

# SQL语句的动态构造-示例2
## (2)动态SQL语句的构造—SQL字符串

战德臣 教授


```
if (cbx_class.checked = true and len(trim(sle_class.text))>0) then
   if (firstflag = 1) then
      str_temp = str_temp + " and (sclass like '" + trim(sle_class.text)+ "')"
   else
      str_temp = str_temp + " ( sclass like '" + trim(sle_class.text)+ "')"
      firstflag = 1
   end if
end if
if (cbx_dept.checked = true and len(trim(sle_dept.text))>0) then
   if (firstflag = 1) then
      str_temp = str_temp + " and (sdept = '" + trim(sle_dept.text)+ "') "
   else
      str_temp = str_temp + " ( sdept like '" + trim(sle_dept.text)+ "')"
      firstflag = 1
   end if
end if
if (cbx_addr.checked = true and len(trim(sle_addr.text))>0) then
   if (firstflag = 1) then
      str_temp = str_temp + " and (saddr like '" + trim(sle_addr.text)+ "') "
   else
      str_temp = str_temp + " (saddr like '" + trim(sle_addr.text)+ "') "
      firstflag = 1
   end if
end if
```

# SQL语句的动态构造-示例2
## (2)动态SQL语句的构造—SQL字符串

战德臣 教授

```
if (cbx_addr.checked = true and len(trim(sle_addr.text))>0) then
   if (firstflag = 1) then
      str_temp = str_temp + " and (saddr like '" + trim(sle_addr.text)+ "') "
   else
      str_temp = str_temp + " (saddr like '" + trim(sle_addr.text)+ "') "
      firstflag = 1
   end if
end if
if (cbx_agest.checked = true and len(trim(sle_agest.text))>0 ) then
   if (firstflag = 1) then
    str_temp = str_temp + " and ( sage >= " + trim(sle_agest.text) +")"
    if (len(trim(sle_ageed.text))>0 and integer(trim(sle_ageed.text))> Integer(trim(sle_agest.text))) then
       str_temp = str_temp + " and ( sage <= " + trim(sle_ageed.text) +")"
    end if
  else
     str_temp = str_temp + " (sage >= " + trim(sle_agest.text) +")"
     if ( len(trim(sle_ageed.text))>0 and integer(trim(sle_ageed.text))>integer(trim(sle_agest.text))) then
        str_temp = str_temp + " and (sage <= " + trim(sle_ageed.text)+")"
     end if
        firstflag = 1
  end if
end if
//sle_sqltext.text   = str_temp
sqltext = str_temp
exec  sql   execute  immediate  :sqltext
exec sql commit  release

// dw_2.setfilter(str_temp)
// dw_2.filter()
```

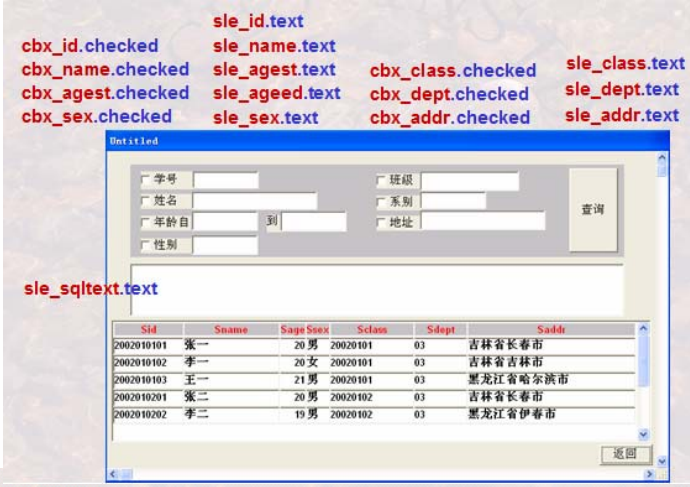
文本框中输入的是字符，比较时要转换成数值

Integer型属性条件如何构造?

数值变量转换成字符串

构造好的SQL字符串如何执行?

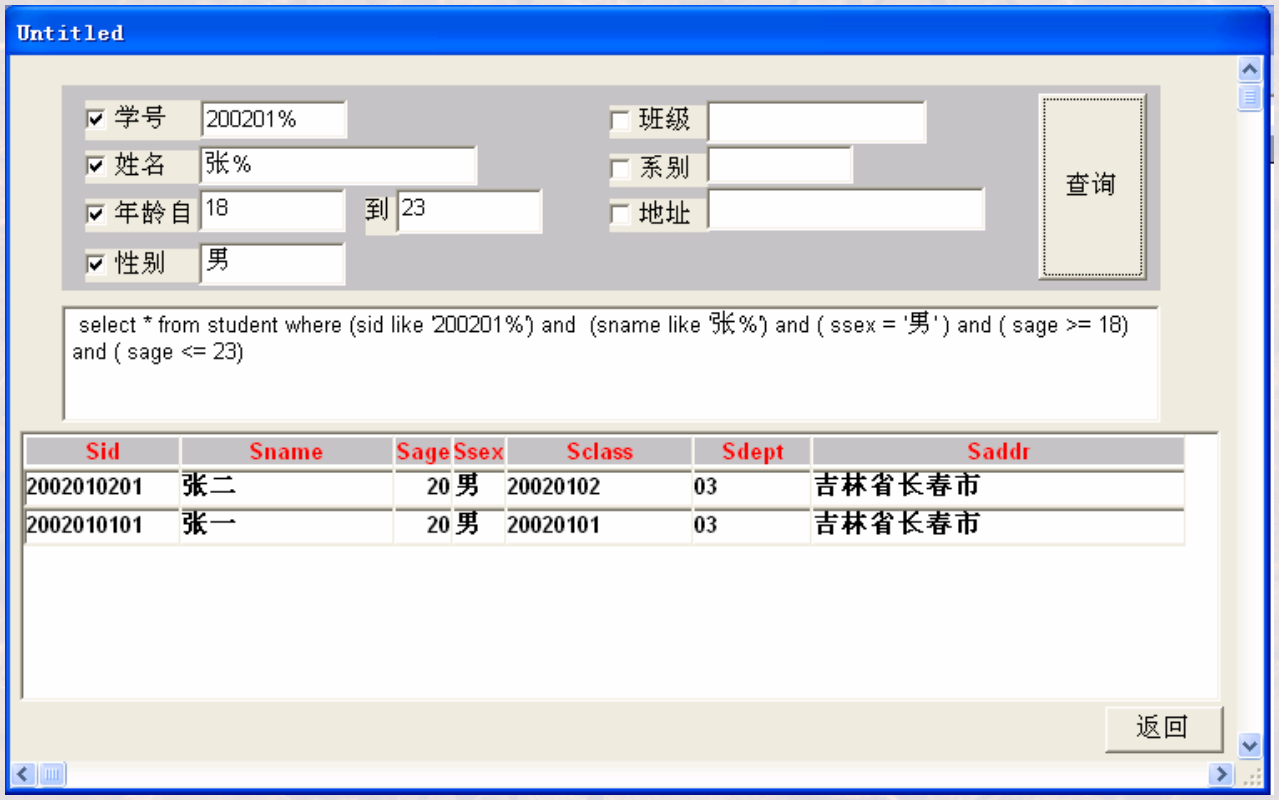
执行构造好的SQL字符串



# SQL语句的动态构造-示例2
# (3)动态SQL语句的执行结果

战德臣 教授

## 示例结果1:



# SQL语句的动态构造-示例2
# (3)动态SQL语句的执行结果

战德臣 教授

## 示例结果2:


(用户)
界面操作
表单型语言

数据库应用程序

(应用程序员)
QBE

(应用程序员)
SQL

数据库管理系统实现程序

(DBMS)
SQL

(DBMS)
关系代数

# 动态SQL语句的执行方式

战德臣

哈尔滨工业大学　教授.博士生导师
黑龙江省教学名师
教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent
Computing for Enterprises & Services,
Harbin Institute of Technology

## 动态SQL的两种执行方式

怎样执行字符串变量存储的SQL语句?

➢如SQL语句已经被构造在host-variable字符串变量中, 则：

❑**立即执行语句**: 运行时编译并执行

```
EXEC SQL EXECUTE IMMEDIATE :host-variable;
```

❑**Prepare-Execute-Using语句**: PREPARE语句先编译，编译后的SQL语句允许动态参数，EXECUTE语句执行，用USING语句将动态参数值传送给编译好的SQL语句

```
EXEC SQL PREPARE sql_temp FROM :host-variable;
... ...
EXEC SQL EXECUTE sql_temp USING :cond-variable
```

为什么需要Prepare-Execute-Using执行方式?

战德臣 教授

## Prepare-Execute-Using的例子

```
#include <stdio.h>
#include "prompt.h"
exec sql include sqlca;
exec sql begin declare section;
   char cust_id[5], sqltext[256], user_name[20], user_pwd[10];
exec sql end declare section;
char cid_prompt[ ] = "Name customer cid to be deleted:  ";
int main()
{
  strcpy(sqltext,   "delete from customers where cid = :dcid");
   ... ...
  while (prompt(cid_prompt, 1, cust_id, 4) >= 0) {
     exec sql whenever not found goto no_such_cid;
     exec sql prepare delcust from  :sqltext;    /* prepare statement */
     exec sql execute delcust using :cust_id;   /* using clause ... replaces ":n" above */
    exec sql commit work;      continue;
 no_such_cid: printf("No cust %s in table\n",cust_id);
    continue;
  }
  ...
}
```

# 示例

```
... ...
strcpy(sqltext,   "delete from customers where cid = 'C01'");
... ...
exec  sql   execute  immediate  :sqltext;
exec  sql   commit work;
```

构造的字符串SQL内部没有“变量”参数

```
... ...
strcpy(sqltext,   "delete from customers where cid = :dcid");
... ...
exec sql prepare delcust from  :sqltext;    /* prepare statement */
exec sql execute delcust using :cust_id;    /* using clause ... replaces ":n" above */
exec  sql   commit work;
```

构造的字符串SQL内部有“变量”参数

# 数据字典与SQLDA


战德臣

哈尔滨工业大学　教授.博士生导师

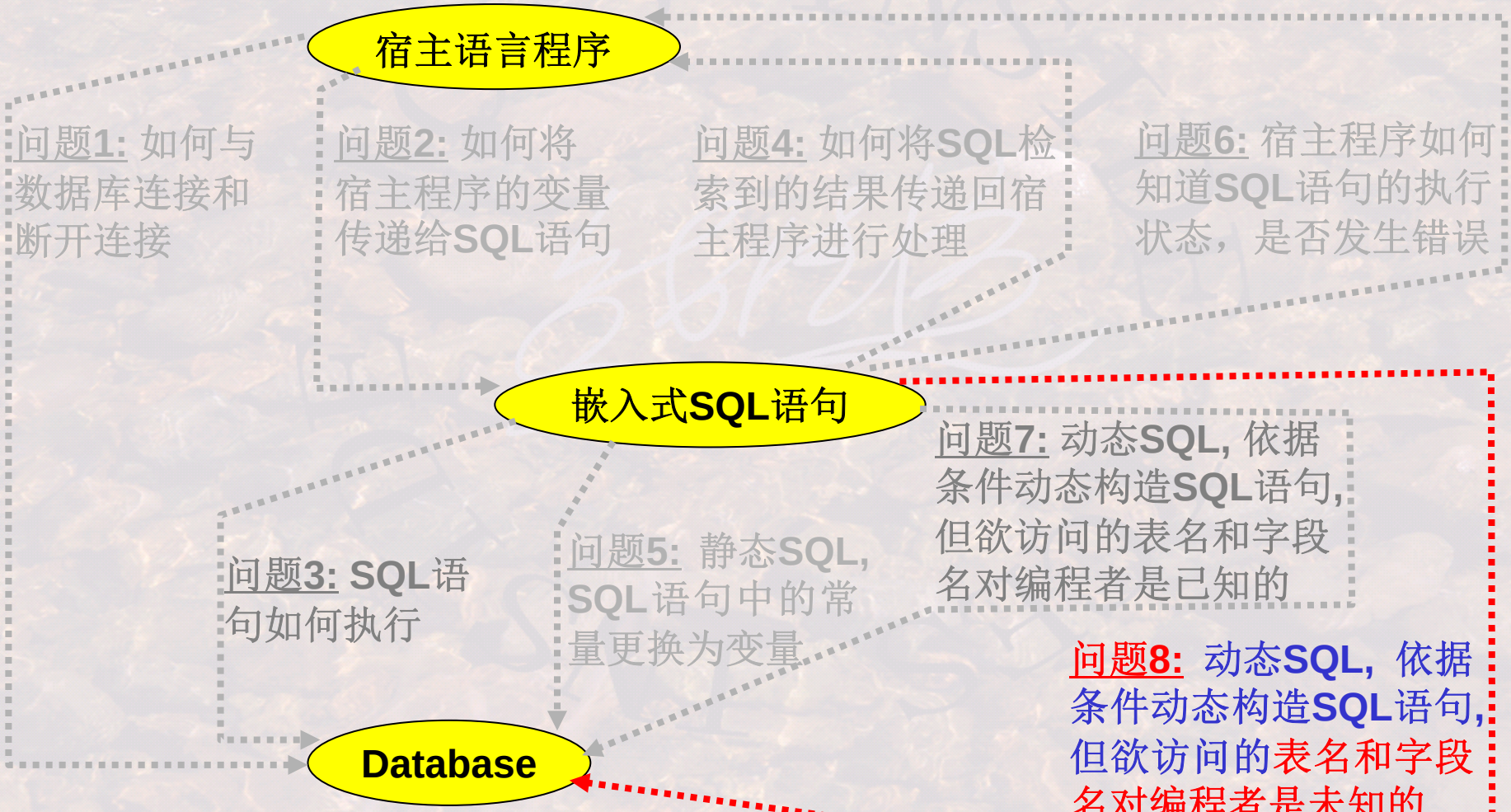
黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

**Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology**

战德臣 教授

# 高级语言(语句)→嵌入式SQL语句→DBMS←→DB



战德臣 教授

# 数据字典(Data dictionary)，又称为系统目录(System Catalogs)

- 是系统维护的一些表或视图的集合，这些表或视图存储了数据库中各类对象的定义信息，这些对象包括用Create语句定义的表、列、索引、视图、权限、约束等, 这些信息又称数据库的元数据--关于数据的数据。
- 不同DBMS术语不一样：数据字典(Data Dictionary(Oracle))、目录表(DB2 UDB)、系统目录(INFORMIX)、系统视图(X/Open)
- 不同DBMS中系统目录存储方式可能是不同的, 但会有一些信息对DBA公开。这些公开的信息, DBA可以使用一些特殊的SQL命令来检索。

战德臣 教授

## 数据字典的内容构成

➢数据字典通常存储的是数据库和表的元数据，即模式本身的信息：

- **❒与关系相关的信息**
  - ✓关系名字
  - ✓每一个关系的属性名及其类型
  - ✓视图的名字及其定义
  - ✓完整性约束
- **❒用户与账户信息，包括密码**
- **❒统计与描述性数据**：如每个关系中元组的数目
- **❒物理文件组织信息**：
  - ✓关系是如何存储的(顺序/无序/散列等)
  - ✓关系的物理位置
- **❒索引相关的信息**

**应用程序员构造动态SQL可能需要知道这些信息**

**数据库管理系统实现算法需要用到这些信息**

战德臣 教授

## 数据字典的结构

- ✓也是存储在磁盘上的关系
- ✓专为内存高效访问设计的特定的数据结构

### ❒可能的字典数据结构

***Relation_metadata = (relation_name, number_of_attributes, storage_organization, location)***

***Attribute_metadata = (attribute_name, relation_name, domain_type, position, length)***

***User_metadata = (user_name, encrypted_password, group)***

***Index_metadata = (index_name, relation_name, index_type, index_attributes)***

***View_metadata = (view_name, definition)***

# 数据字典与SQLDA
## (5)X/Open标准的系统目录



➢X/Open标准中有一个目录表Info_Schem.Tables, 该表中的一行是一个已经定义的表的有关信息

| 列名 | 描述 |
|---|---|
| Table_Schem | 表的模式名(通常是表所有者的用户名) |
| Table_Name | 表名 |
| Table_Type | 'Base_Table'或 'View' |

➢可以使用SQL语句来访问这个表中的信息，比如了解已经定义了哪些表，可如下进行：

```
Select  Table_Name  From  Tables;
```

➢**模式**的含义是指某一用户所设计和使用的表、索引及其他与数据库有关的对象的集合，因此表的完整名应是：模式名.表名。这样做可允许不同用户使用相同的表名，而不混淆。

➢ 一般而言，一个用户有一个模式。可以使用Create Schema语句来创建模式(用法略，参见相关文献)，在Create Table等语句可以使用所定义的模式名称。

战德臣 教授

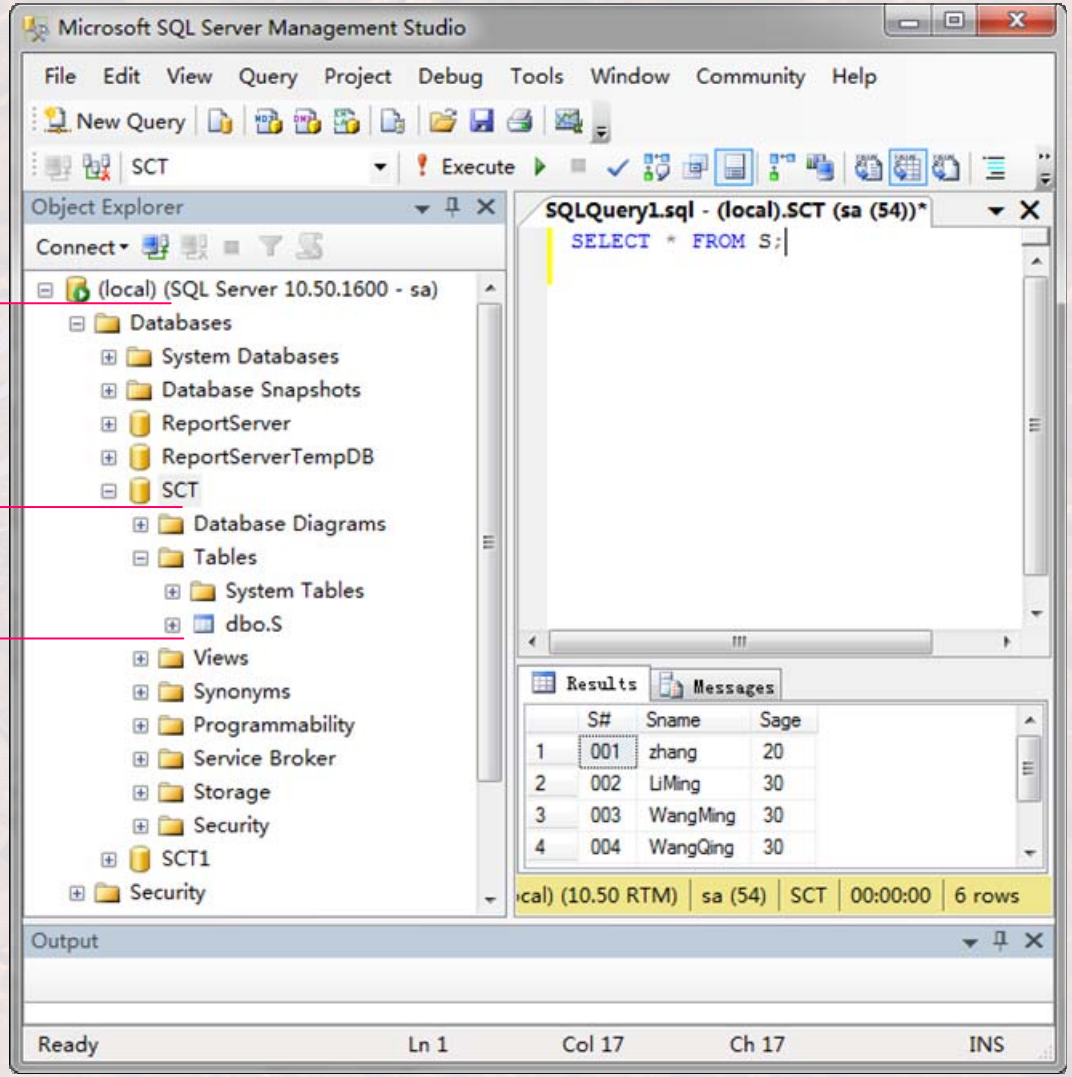
## 目录-模式-对象(如具体的表等) 示例

服务器—用户

数据库名

模式名.表名

Dbo为数据库拥有者,
为默认的模式名



# 数据字典与SQLDA
## (6)Oracle的数据字典

战德臣 教授

- **Oracle数据字典由视图组成，分为三种不同形式，由不同的前缀标识**
  - **USER_ :用户视图，用户所拥有的对象，在用户模式中**
  - **ALL_ :扩展的用户视图，用户可访问的对象**
  - **DBA_ :DBA视图(所有用户都可访问的DBA对象的子集)**
- Oracle数据字典中定义了三个视图USER_Tables, ALL_Tables, 和DBA_Tables供DBA和用户使用数据字典中关于表的信息

ALL_Tables

DBA_Tables

User_Tables:没有 Owner 列

| 列名 | 描述 |
|---|---|
| Owner | 表的所有者 |
| Table_Name | 表名 |
| (其他列...) | 磁盘存取和更新事务信息 |

# 数据字典与SQLDA
## (6)Oracle的数据字典

战德臣 教授

➢同样, Oracle数据字典中也定义了三个视图USER_TAB_Columns, ALL_TAB_Columns(Accessible_Columns), 和DBA_TAB_Columns供DBA和用户使用数据字典中关于表的列的信息

ALL_TAB_Columns
DBA_TAB_Columns
User_TAB_Columns:没有 Owner 列

| 列名 | 描述 |
|---|---|
| Owner | 表的所有者 |
| Table_Name | 表名 |
| Column_Name | 列名 |
| Data_Type | 列的数据类型 |
| Data_Length | 列的数据长度，以字节为单位 |
| (其他列...) | 其他属性，空值、缺省值等 |

➢可以使用SQL语句来访问这些表中的信息：

```
Select  Column_Name  From  ALL_TAB_Columns
Where  Table_Name  = 'STUDENT' ;
```

# 数据字典与SQLDA
## (6)Oracle的数据字典

战德臣 教授

- **Oracle数据字典中还定义了其他视图**
  - **TABLE_PRIVILEDGE(或ALL_TAB_GRANTS)**
  - **COLUMN_PRIVILEDGE(或ALL_COL_GRANTS)**

    可访问表的权限，列的权限
  - **CONSTRAINT_DEFS(或ALL_CONSTRAINTS)**

    可访问表的各种约束
  - 还有其他视图… …
- **可以使用下述命令获取Oracle定义的所有视图信息**

```
Select view_name from all_views where owner = 'SYS' and
view_name like 'ALL_%' or view_name like 'USER_%';
```

- 如果用户使用Oracle,可使用其提供的SQL*PLUS进行交互式访问

战德臣 教授

**示例：针对几个表，编写由用户确定检索条件的查询程序**

请输入 Y_N _Y_ 姓名张三____
Y_N _N_ 年龄在_____和_____之间
Y_N _Y_ 班级035103_____
... ...

动态SQL: 表和列都已知，动态构造检索条件

➢已知被检索的Table名, 以及Table中各列名；动态性体现在检索条件的可构造方面。

**示例：针对一组表，编写由用户确定检索表和检索条件的查询程序**

➢只知道表的集合，但具体检索哪个Table，以及检索该Table中的哪些列都是可构造的，检索条件也是用户临时设置的，可构造的。

动态SQL:检索条件动态构造，表和列也需要动态构造

# 数据字典与SQLDA
## (8)SQLDA

战德臣 教授

- 构造复杂的动态SQL需要了解数据字典及SQLDA，已获知关系模式信息

**SQLDA:** SQL Descriptor Area, SQL描述符区域。

- SQLDA是一个内存数据结构，内可装载关系模式的定义信息，如列的数目，每一列的名字和类型等等
- 通过读取SQLDA信息可以进行更为复杂的动态SQL的处理
- 不同DBMS提供的SQLDA格式并不是一致的。

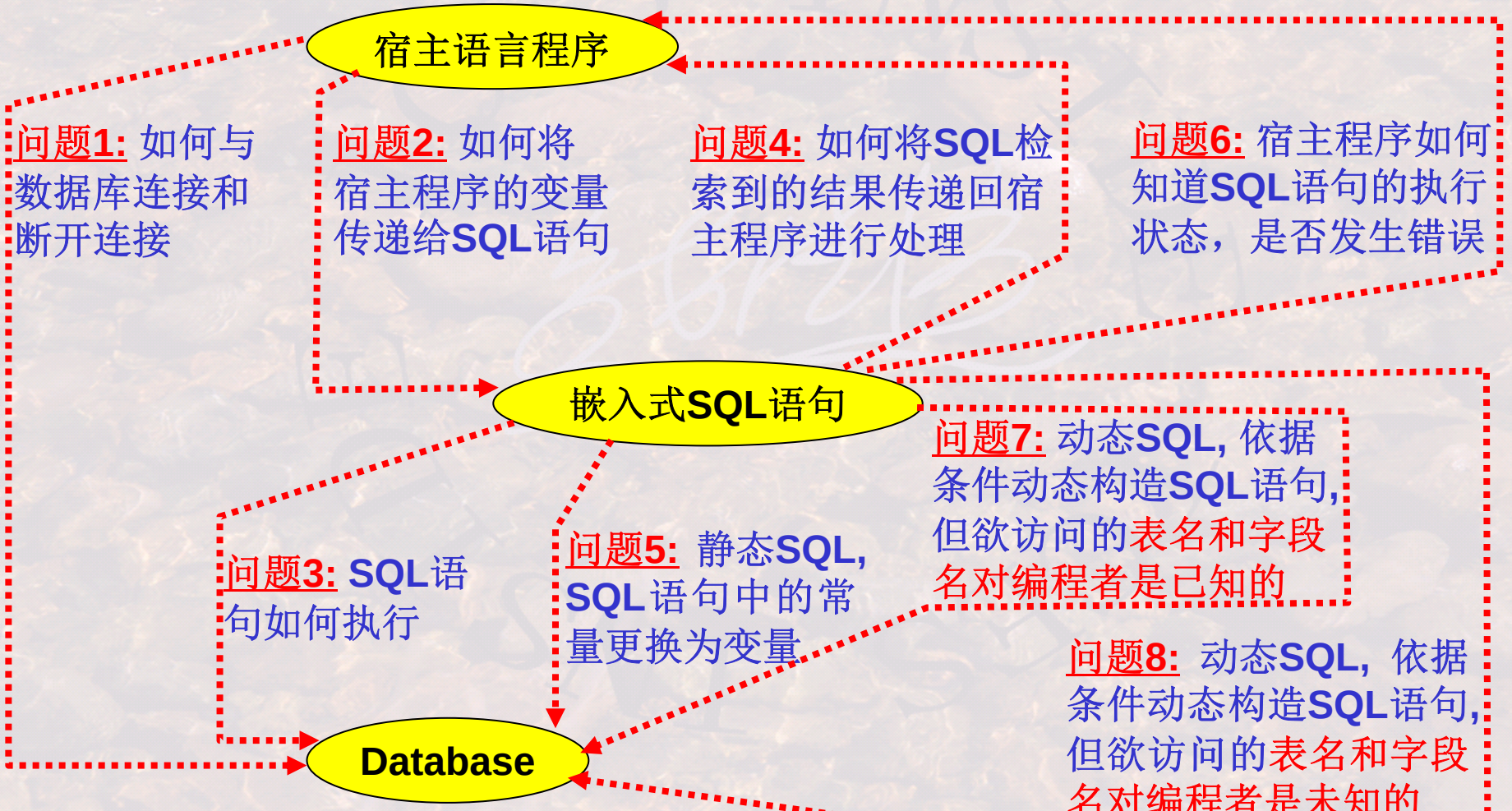
```
struct sqlvar_struct
{  short sqltype;  /* variable type*/
   int sqllen;   /* length in bytes*/
   char *sqldata;   /* pointer to data*/
   short *sqlind;   /* pointer to indicator*/
   char *sqlname;   /* variable name*/
   char *sqlformat;  /* reserved for future use */
   short sqlitype;      /* ind variable type*/
   short sqlilen;       /* ind length in bytes*/
   char *sqlidata;      /* ind data pointer*/
   int   sqlxid;       /* extended id type          */
   char *sqltypename;   /* extended type name        */
   short sqltypelen;    /* length of extended type name */
   short sqlownerlen;   /* length of owner name      */
   short sqlsourcetype;  /* source type for distinct of built-ins */
   char *sqlownername;  /* owner name                */
   int sqlsourceid;     /* extended id of source type  */
   /*   *sqlilongdata is new.  It supports data that exceeds the 32k
    * limit.  sqlilen and sqlidata are for backward compatibility
    * and they have maximum value of <32K.    */
   char *sqlilongdata;      /* for data field beyond 32K  */
   int4 sqlflags;           /* for internal use only      */
   void *sqlreserved;       /* reserved for future use    */
};
struct sqlda
{  short  sqld;
   struct sqlvar_struct *sqlvar;
   char   desc_name[19];   /* descriptor name */
   short  desc_occ;        /* size of sqlda structure */
   struct sqlda *desc_next; /* pointer to next sqlda struct */
   void *reserved;          /* reserved for future use */
}
```

这只是示意。其结构的解释请参看相应的手册

典型DBMS的SQLDA结构

# 数据字典与SQLDA
## (9)总结

战德臣 教授



# ODBC简介


战德臣

哈尔滨工业大学　教授.博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

**Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology**

# ODBC简介
## (1)什么是ODBC?

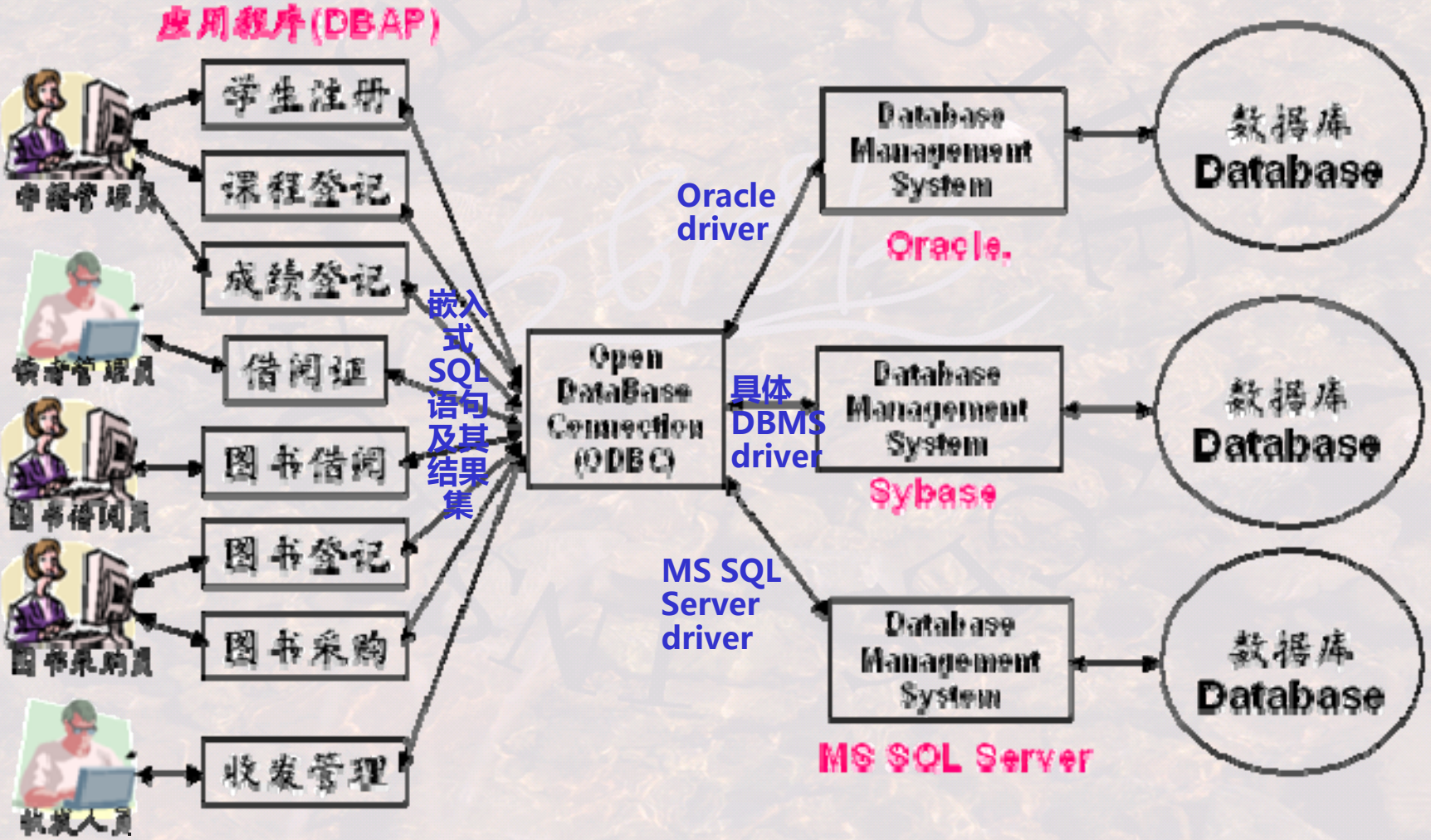
战德臣 教授

- ODBC：Open DataBase Connection

**ODBC**是一种标准---不同语言的应用程序与不同数据库服务器之间通讯的标准。

- 一组API(应用程序接口)，支持应用程序与数据库服务器的交互
- 应用程序通过调用ODBC API, 实现
  - ✓与数据服务器的连接
  - ✓向数据库服务器发送SQL命令
  - ✓一条一条的提取数据库检索结果中的元组传递给应用程序的变量
- 具体的DBMS提供一套驱动程序，即Driver库函数，供ODBC调用，以便实现数据库与应用程序的连接。
- ODBC可以配合很多高级语言来使用，如C,C++, C#, Visual Basic, Power-Builder等等；

# ODBC简介
## (1)什么是ODBC?



# ODBC简介

## (2)应用程序如何通过ODBC连接一个数据库服务器?

战德臣 教授

- ODBC应用前，需要确认具体DBMS Driver被安装到ODBC环境中
- 当应用程序调用ODBC API时，ODBC API会调用具体DBMS Driver库函数，DBMS Driver库函数则与数据库服务器通讯，执行相应的请求动作并返回检索结果
- ODBC应用程序首先要分配一个SQL环境，再产生一个数据库连接句柄
- 应用程序使用SQLConnect()，打开一个数据库连接，SQLConnect()的具体参数:
  - ✓connection handle,　连接句柄
  - ✓the server，要连接的数据库服务器
  - ✓the user identifier，用户
  - ✓password ，密码
  - ✓SQL_NTS 类型说明前面的参数是空终止的字符串

# ODBC简介

## (2)应用程序如何通过ODBC连接一个数据库服务器?

战德臣 教授

**示例**

```
int ODBCexample()
{
    RETCODE error;   /* 返回状态吗 */
    HENV    env;     /* 环境变量 */
    HDBC    conn;    /* 连接句柄 */
    SQLAllocEnv(&env);
    SQLAllocConnect(env, &conn);
    //分配数据库连接环境
    SQLConnect(conn, "aura.bell-labs.com", SQL_NTS,
    "avi", SQL_NTS, avipasswd", SQL_NTS);
     //打开一个数据库连接

      { …. Do actual work … }
     //与数据库通讯

    SQLDisconnect(conn);
    SQLFreeConnect(conn);
    SQLFreeEnv(env);
    //断开连接与释放环境
}
```

# ODBC简介

## (3)应用程序如何通过ODBC与数据库服务器进行通讯?

战德臣 教授

- ■应用程序使用**SQLExecDirect()**向数据库发送SQL命令；
- ■使用**SQLFetch()**获取产生的结果元组；
- ■使用**SQLBindCol()**绑定C语言变量与结果中的属性
  - ✓当获取一个元组时，属性值会自动地传送到相应的C语言变量中
  - ✓**SQLBindCol()**的参数：
    - □ODBC定义的stmt变量, 查询结果中的属性位置
    - □SQL到C的类型变换, 变量的地址.
    - □对于类似字符数组一样的可变长度类型，应给出
      - •变量的最大长度
      - •当获取到一个元组后，实际长度的存储位置.
      - •注: 当返回实际长度为负数，说明是一个空值。

## (3)应用程序如何通过ODBC与数据库服务器进行通讯?

战德臣 教授

**示例**

```
char branchname[80];  float  balance;
int  lenOut1, lenOut2;
HSTMT   stmt;
SQLAllocStmt(conn, &stmt);
//分配一个与指定数据库连接的新的语句句柄
char * sqlquery = "select branch_name, sum (balance)
                   from account
                   group by branch_name";
error = SQLExecDirect(stmt, sqlquery, SQL_NTS);
//执行查询，stmt句柄指向结果集合
if (error == SQL_SUCCESS) {
SQLBindCol(stmt, 1, SQL_C_CHAR,   branchname , 80, &lenOut1);
SQLBindCol(stmt, 2, SQL_C_FLOAT, &balance,        0 , &lenOut2);
//绑定高级语言变量与stmt句柄中的属性
    while (SQLFetch(stmt) >= SQL_SUCCESS) {
//提取一条记录，结果数据被存入高级语言变量中
      printf (" %s  %g\n", branchname, balance);
     }
}
SQLFreeStmt(stmt, SQL_DROP);
//释放语句句柄
```

# ODBC简介
## (4)ODBC还有其他功能?

战德臣 教授

- 动态SQL语句的预编译-动态参数传递功能
- 获取元数据特性
  - 发现数据库中的所有关系的特性 以及
  - 发现每一个查询结果的列的名字和类型等;
- 默认, 每一条SQL语句都被作为一个独立的能够自动提交的事务来处理。
  - 应用程序可以关闭一个连接的自动提交特性
    - **SQLSetConnectOption(conn, SQL_AUTOCOMMIT, 0)}**
  - 此时事务要显式地给出提交和撤销的命令
    - **SQLTransact(conn, SQL_COMMIT)** or
    - **SQLTransact(conn, SQL_ROLLBACK)**

# JDBC简介


战德臣

哈尔滨工业大学　教授.博士生导师
黑龙江省教学名师
教育部大学计算机课程教学指导委员会委员

**Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology**

# JDBC简介
## (1)JDBC是什么？

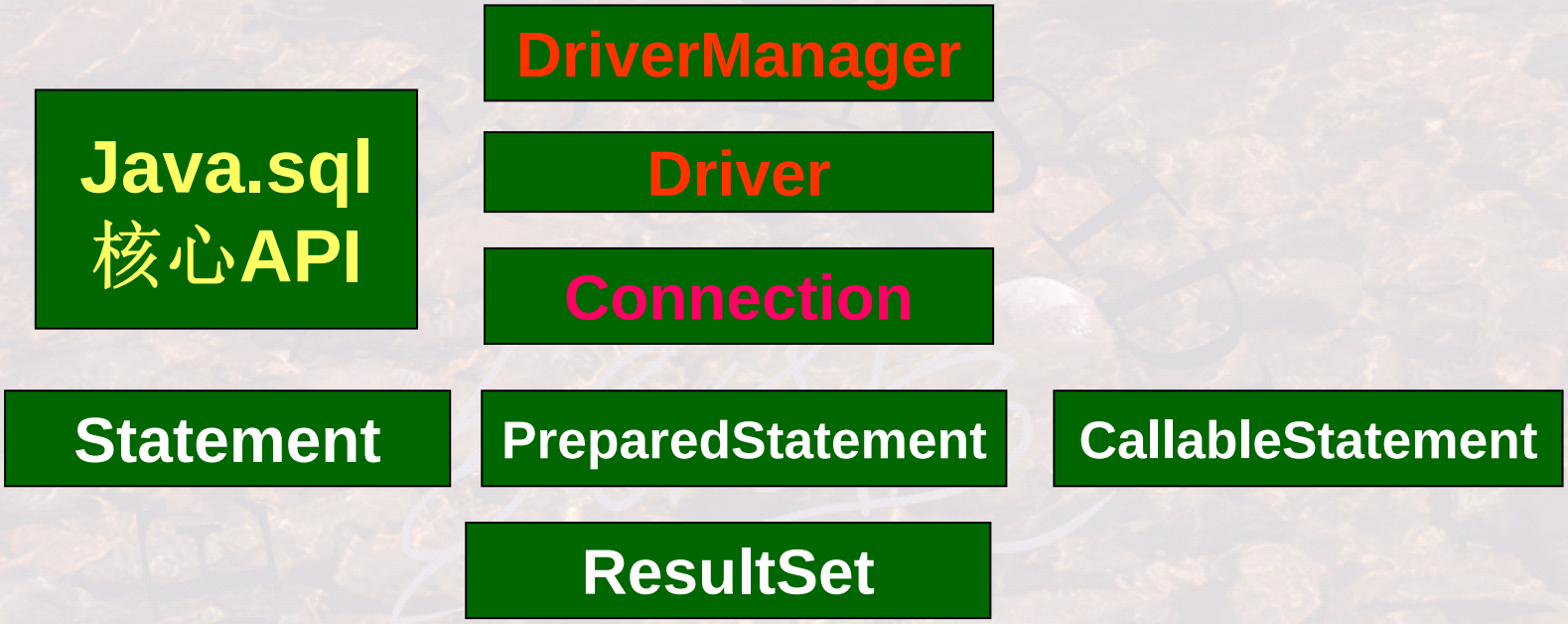


- **JDBC：Java DataBase Connection**
- JDBC是一组Java版的应用程序接口API，提供了Java应用程序与数据库服务器的连接和通讯能力。

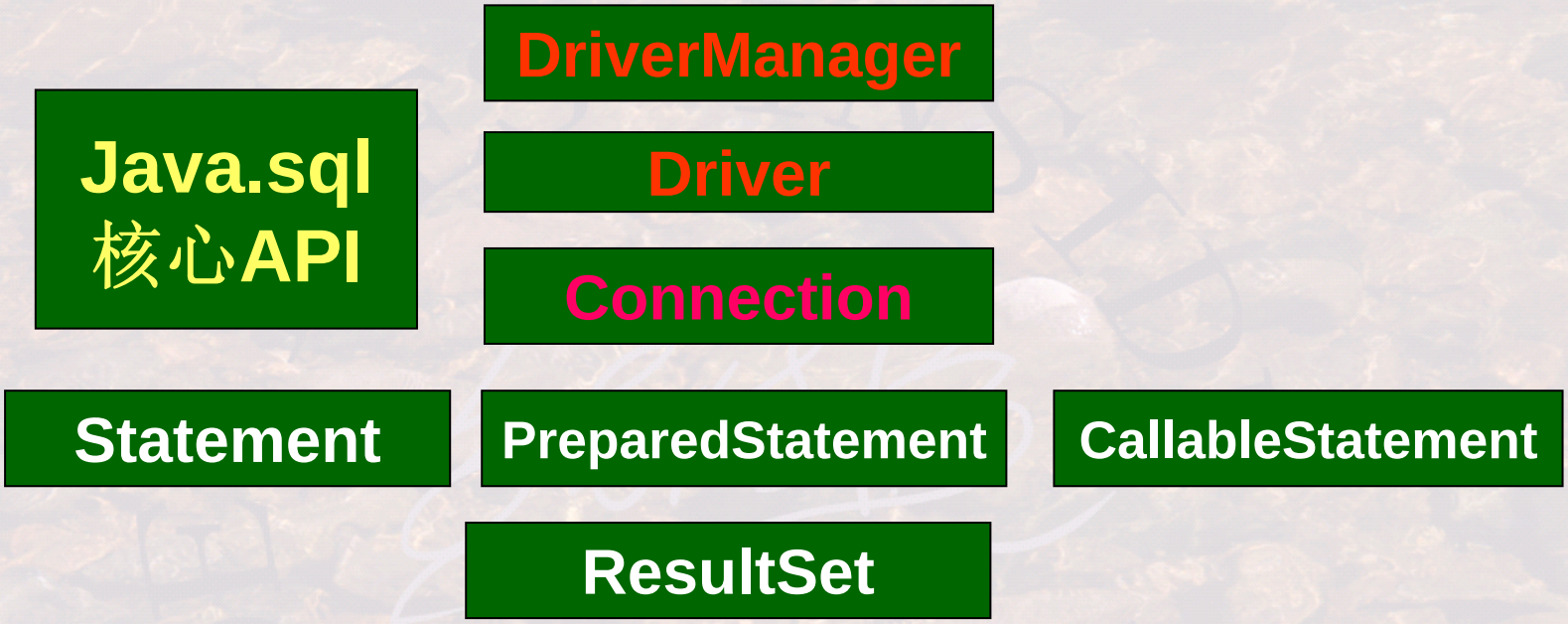
## JDBC API

JDBC API 分成两个程序包：

- **Java.sql 核心API** --J2SE(Java2标准版)的一部分。使用**java.sql.DriverManager**类、**java.sql.Driver**和**java.sql.Connection**接口连接到数据库
- **Javax.sql 可选扩展API**--J2EE(Java2企业版)的一部分。包含了基于JNDI(Java Naming and Directory Interface, Java命名和目录接口)的资源，以及管理连接池、分布式事务等，使用**DataSource**接口连接到数据库。




- java.sql.DriverManager——处理驱动的调入并且对产生新数据库连接提供支持
- Java.sql.Driver——通过驱动进行数据库访问，连接到数据库的应用程序必须具备该数据库的特定驱动。
- java.sql.Connection——代表对特定数据库的连接。
- Try {…}　Catch {…} ——异常捕获及其处理




- java.sql.Statement——对特定的数据库执行SQL语句
- java.sql.PreparedStatement —— 用于执行预编译的SQL语句
- java.sql.CallableStatement ——用于执行对数据库内嵌过程的调用。
- java.sql.ResultSet——从当前执行的SQL语句中返回结果数据。

## (3)应用程序使用JDBC API访问数据库的过程

战德臣 教授

### 概念性的基本过程

打开一个连接；创建“Statement”对象，并设置查询语句；使用Statement对象执行查询，发送查询给数据库服务器和返回结果给应用程序；处理错误的例外机制

### 具体实施过程

✓传递一个Driver给DriverManager，加载数据库驱动。

- Class.forName()

✓通过URL得到一个Connection对象, 建立数据库连接

- DriverManager.getConnection(sDBUrl)
- DriverManager.getConnection(sDBUrl,sDBUserID,sDBPassword)

✓接着创建一个Statement对象(PreparedStatement或CallableStatement)，用来查询或者修改数据库。

- Statement stmt=con.createStatement()

✓查询返回一个ResultSet。

- ResultSet rs=stmt.executeQuery(sSQL)

# 示例

```
public static void JDBCexample(String dbid, String userid, String passwd)
{  try {  //错误捕获
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        Connection conn = DriverManager.getConnection(
          "jdbc:oracle:thin:@db.yale.edu:1521:univdb", userid, passwd);
        //加载数据库驱动，建立数据库连接
         Statement stmt = conn.createStatement();
        //创建一个语句对象
            … Do Actual Work ….
        //进行SQL语句的执行与处理工作
         stmt.close();
         conn.close();
        //关闭语句对象，关闭连接
      }  catch (SQLException sqle) {
         System.out.println("SQLException : " + sqle);    }
}
```

## Update to database

```
try {   stmt.executeUpdate(  "insert into instructor values
                              ('77987', 'Kim', 'Physics',98000)");
        //插入一条记录
    } catch (SQLException sqle) {
        System.out.println("Could not insert tuple. " + sqle); }
```

## Execute query and fetch and print results

```
ResultSet rset = stmt.executeQuery( "select dept_name, avg(salary)"+
                 " from instructor  group by dept_name" );
    while ( rset.next() ) {
        System.out.println(rset.getString("dept_name") + "  "
            +  rset.getFloat(2));   }
```

//执行一条SQL语句。获取下一条记录。提取“dept_name”属性值，提取第2列即“平均工资”列的值

# 完整的示例程序

```
public static void JDBCexample(String dbid, String userid, String passwd)
{
    try {
            Class.forName ("oracle.jdbc.driver.OracleDriver");
            Connection conn = DriverManager.getConnection( "jdbc:oracle:thin:
                                @db.yale.edu:1521:univdb", userid, passwd);
            Statement stmt = conn.createStatement();
            try {
                    stmt.executeUpdate(  "insert into instructor values
                                        (‘77987', ‘Kim', ‘Physics’,98000)");
                } catch (SQLException sqle) {
                    System.out.println("Could not insert tuple. " + sqle);
                                                }
            ResultSet rset = stmt.executeQuery( "select dept_name, avg(salary)”+
                            “ from instructor  group by dept_name");
            while ( rset.next() ) {
                System.out.println(rset.getString(“dept_name") + "  " + rset.getFloat(2));  }
            stmt.close();
            conn.close();
          } catch (SQLException sqle) {
                System.out.println("SQLException : " + sqle);
                                                }
}
```

# 嵌入式语言-ODBC-JDBC比较


战德臣

哈尔滨工业大学　教授.博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

**Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology**

# 嵌入式语言-ODBC-JDBC比较
## (1)嵌入式SQL的思维模式

战德臣 教授

```
#define TRUE 1
#include  <stdio.h>
#include "prompt.h"
exec  sql  include sqlca;
exec  sql begin declare section;
        char  cust_id[5], agent_id[14];
        double dollar_sum;
exec sql end declare section;
int main()
{   char  cid_prompt[ ]="Please enter customer ID:";
     exec sql whenever sqlerror goto report_error;
     exec sql connect to testdb;

     { do actual work ... }

    exec sql  disconnect current;   return 0;
report_error:
    print_dberror();  exec sql rollback;
    exec sql disconnect current;  return 1;
}
```

语句形式执行

```
exec sql declare agent_dollars cursor  for  select  aid,sum(dollars)
          from orders  where cid = :cust_id  group by aid;
exec sql whenever not found goto finish;

while((prompt(cid_prompt,1,cust_id,4)) >=0) {
       exec sql open agent_dollars;
       while(TRUE) {
          exec sql fetch agent_dollars into :agent_id, :dollar_sum;
          printf("%s  %11.2f\n",agent_id, dollar_sum);
                    }
 finish: exec sql close agent_dollars;
          exec sql commit work;  }
```

声明时SQL语句与游标结合在一起

一条一条fetch时绑定变量

建立数据库连接 → 声明一个游标 → 打开游标 → 获取一条一条记录 → 关闭游标 → 断开数据库连接

**执行一条SQL语句，读取执行的结果集合**

# 嵌入式语言-ODBC-JDBC比较
## (2)ODBC的思维模式

战德臣 教授

```
int ODBCexample()
        {
     RETCODE error;   /* 返回状态吗 */
     HENV    env;     /* 环境变量 */
     HDBC    conn;    /* 连接句柄 */
    SQLAllocEnv(&env);
     SQLAllocConnect(env, &conn);
    //分配数据库连接环境
     SQLConnect(conn, "aura.bell-labs.com", SQL_NTS,
    "avi", SQL_NTS, avipasswd", SQL_NTS);
     //打开一个数据库连接

     { .... Do actual work ... }
     //与数据库通讯

     SQLDisconnect(conn);
     SQLFreeConnect(conn);
     SQLFreeEnv(env);
    //断开连接与释放环境
  }
```

```
char branchname[80];  float  balance;
int  lenOut1, lenOut2;
HSTMT   stmt;
SQLAllocStmt(conn, &stmt);
//分配一个与指定数据库连接的新的语句句柄
char * sqlquery = "select branch_name, sum (balance)
                   from account
                   group by branch_name";
error = SQLExecDirect(stmt, sqlquery, SQL_NTS);
//执行查询，stmt句柄指向结果集合
if (error == SQL_SUCCESS) {
SQLBindCol(stmt, 1, SQL_C_CHAR,   branchname , 80, &lenOut1);
SQLBindCol(stmt, 2, SQL_C_FLOAT, &balance,       0 , &lenOut2);
//绑定高级语言变量与stmt句柄中的属性
   while (SQLFetch(stmt) >= SQL_SUCCESS) {
//提取一条记录，结果数据被存入高级语言变量中
    printf (" %s  %g\n", branchname, balance);
   }
}
SQLFreeStmt(stmt, SQL_DROP);
//释放语句句柄
```

API函数形式执行

执行时SQL语句与句柄结合在一起

先绑定变量再一条一条的fetch

建立数据库连接 → 分配语句句柄 → 用句柄执行SQL → 建立高级语言变量与句柄属性的对应 → 获取一条一条记录 → 释放语句句柄 → 断开数据库连接

**执行一条SQL语句，读取执行的结果集合**

# 嵌入式语言-ODBC-JDBC比较
## (3)JDBC的思维模式

战德臣 教授

```
public static void JDBCexample(String dbid, String userid, String passwd)
{ try { //错误捕获
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        Connection conn = DriverManager.getConnection(
         "jdbc:oracle:thin:@db.yale.edu:1521:univdb", userid, passwd);
        //加载数据库驱动，建立数据库连接

           ... Do Actual Work ....
        //进行SQL语句的执行与处理工作

        conn.close();
        //关闭连接
      } catch (SQLException sqle) {
        System.out.println("SQLException : " + sqle);   }
}
```

```
Statement stmt = conn.createStatement();
        //创建一个语句对象
try { stmt.executeUpdate(  "insert into instructor values
                          ('77987', 'Kim', 'Physics',98000)");
      //插入一条记录
    } catch (SQLException sqle) {
        System.out.println("Could not insert tuple. " + sqle); }
ResultSet rset = stmt.executeQuery( "select dept_name, avg(salary)"+
          " from instructor  group by dept_name" );
    while (rset.next() ) {
        System.out.println(rset.getString("dept_name") + "  "
          + rset.getFloat(2));   }
//执行一条SQL语句。获取下一条记录。提取"dept_name"属性值，
列即"平均工资"列的值
stmt.close();  //关闭语句对象
```

创建对象

为对象设置查询语句

对象执行查询并返回结果对象

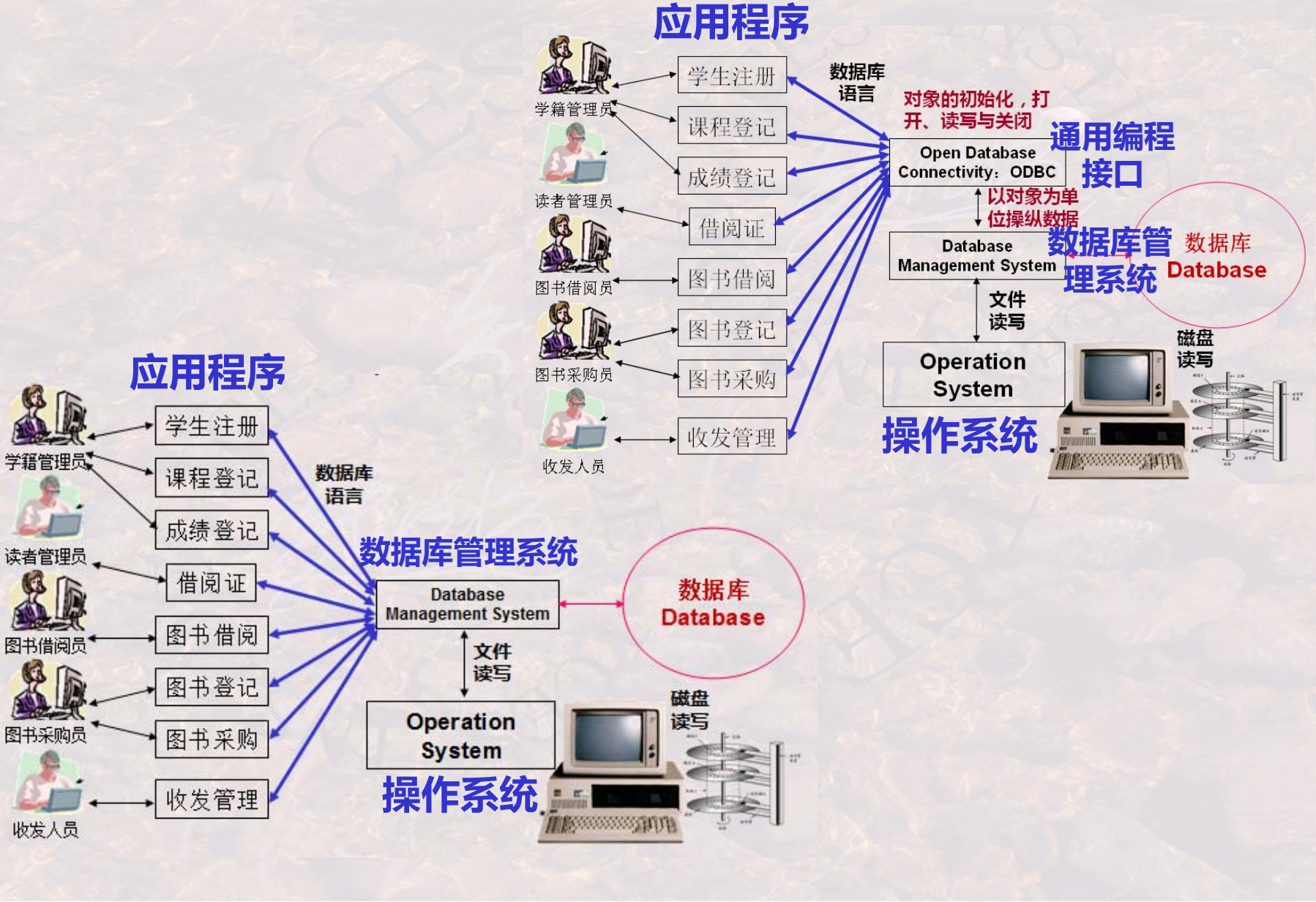
从返回结果对象中提取结果

建立数据库连接 → 创建语句对象 → 用语句对象执行SQL，并返回结果对象 → 从结果对象获取一条一条记录 → 提取对象的属性值传给高级语言变量 → 释放语句对象 → 断开数据库连接

**执行一条SQL语句，读取执行的结果集合**

# 嵌入式语言-ODBC-JDBC比较
## (4)基于ODBC/JDBC的数据库访问

战德臣 教授



# 回顾本讲学了什么?


战德臣

哈尔滨工业大学　教授.博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

**Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology**

# 回顾本讲学习了什么?

战德臣 教授