# 数据库系统原理

## Database System Principle

邵蓥侠

Email：shaoyx@bupt.edu.cn

北京邮电大学计算机学院

计算机应用技术中心

# PART 2

# RELATIONAL DATABASES

# Chapter 5

# Advanced SQL

# Three Parts in Chapter 5

- Accessing SQL From a Programming Language
  - dynamic SQL, e.g. JDBC, ODBC, ADO, …
  - Embedded SQL
- Functions and Procedural（过程） Constructs
- Triggers
- (Recursive Queries, Advanced Aggregation Features, OLAP)

# 5.1 Accessing SQL From a Programming Language

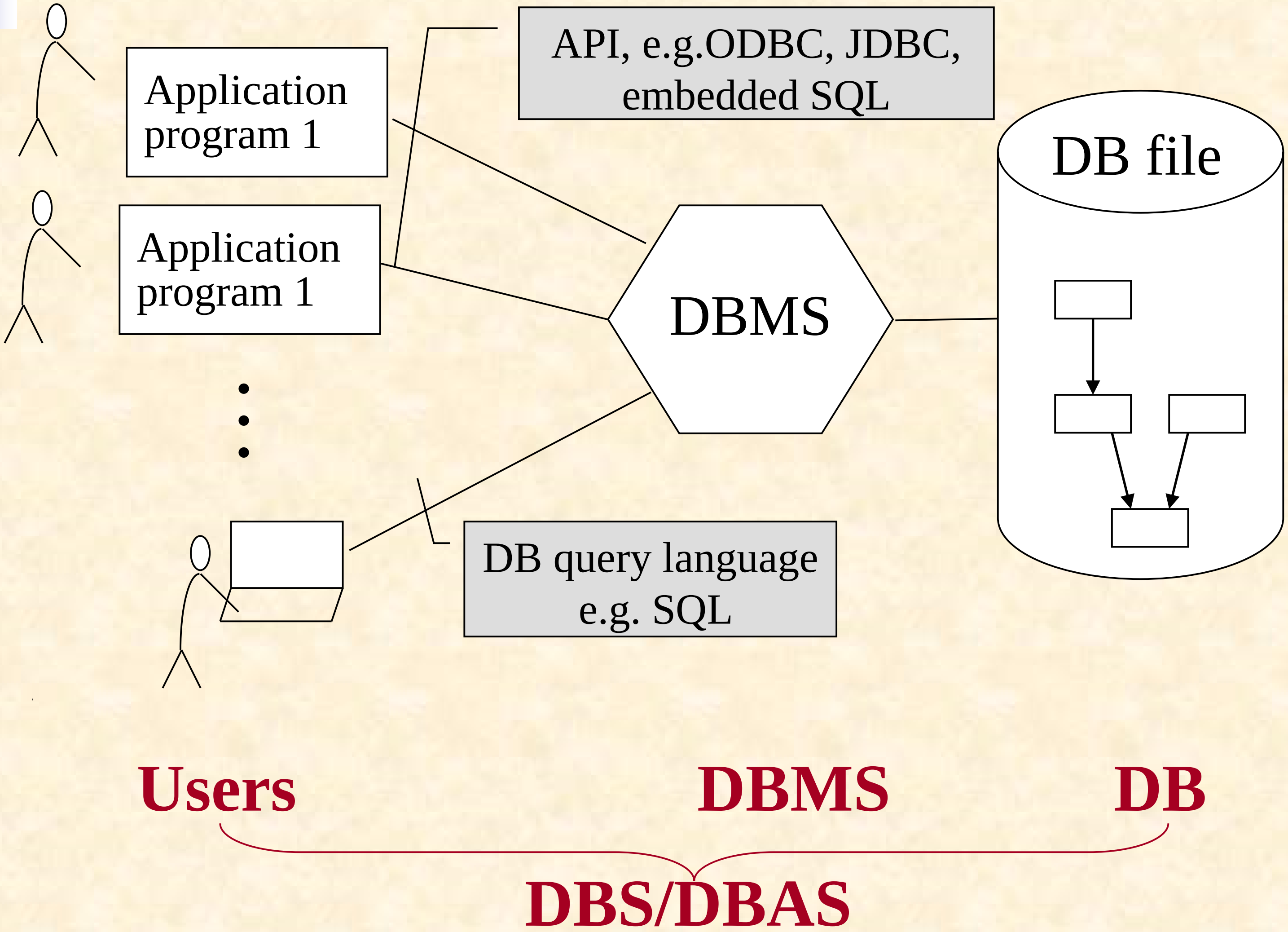


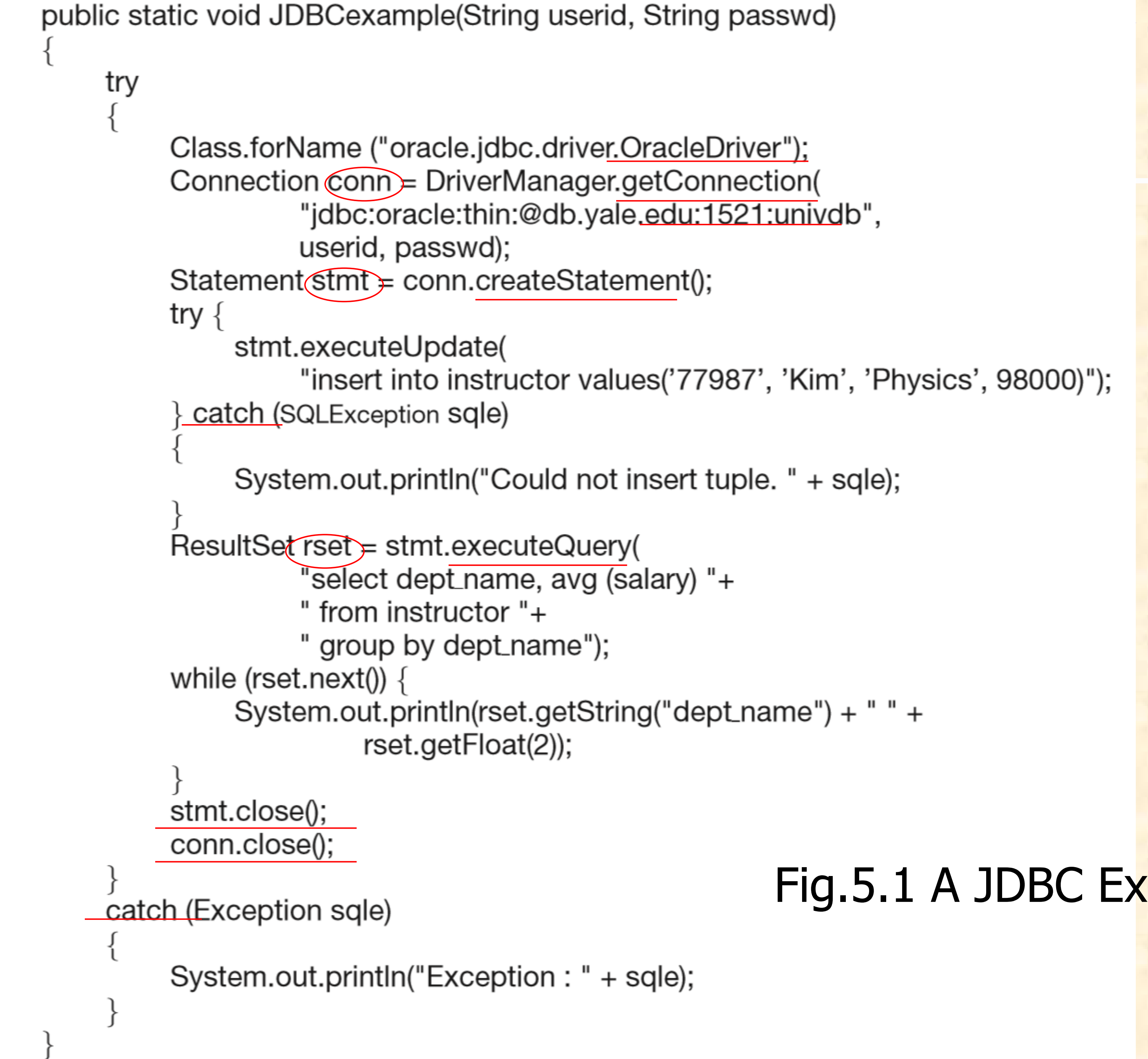
Fig.5.0.1 DBS and DBAS

# Accessing SQL From a Programming Language

- API (application-program interface) for a program to interact with a database server
- Application conducts complex data processing, and makes calls to
  - Connect with the database server
  - Send SQL commands to the database server
  - Fetch tuples of result one-by-one into program variables
- Various tools:
  - JDBC (Java Database Connectivity) works with Java
  - ODBC (Open Database Connectivity) works with C, C++, C#, and Visual Basic
  - Other API's such as ADO.NET sit on top of ODBC
  - Embedded SQL

# 5.1.1 JDBC

- **JDBC** is a Java API for communicating with database systems supporting SQL.
- JDBC supports a variety of features for querying and updating data, and for retrieving query results.
- JDBC also supports metadata retrieval, such as querying about relations present in the database and the names and types of relation attributes.
- Model for communicating with the database:
  - Open a connection
  - Create a “SQL statement” object
  - Execute queries using the Statement object to send queries and fetch results
  - Exception mechanism to handle errors



```
public static void JDBCexample(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 (Exception sqle)
    {
        System.out.println("Exception : " + sqle);
    }
}
```

Fig.5.1 A JDBC Example

# JDBC

- Open a connection, named as *conn*
  - using the *getConnection* method of the *DriverManager* class
  - parameters

    URL/machine to be connected;

    user identifier, password

- Create a SQL “statement” object on the connection *conn*
- using the *createStatement* method
- its statement handle is named as *stmt*
- allows the Java program to invoke（调用）methods that ship an SQL statement given as an argument for execution by the DBMS

# JDBC

- Execute queries using the Statement object *stmt* to send queries and fetch results
- using *execute.query* or *execute.update such as insert/delete/update/createtable*
- parameters

  the SQL statement to be executed, represented as a string
- using the try{…}/*catch*{…}construct to catch any exceptions or error conditions

# JDBC

- Fetch the query result
- retrieve the set of tuples in the result into a ***ResultSet* object** *rset*, and fetch them one tuple at a time
- the *next*() method tests whether or not the result set has at least one tuple and if so, fetches it

- At the end of the query, the statement *stmt* and the connection *conn* are closed

# 5.1.2 ODBC

- Open DataBase Connectivity (ODBC) standard
  - standard for application program to communicate with a database server.
  - application program interface (API) to
    - open a connection with a database,
    - send queries and updates,
    - get back results.
- Applications such as GUI, spreadsheets, etc. can use ODBC
- More details can be found in Appendix A.

# § 5.4 Embedded SQL

- Approaches to take SQL as DB query tools
  - interactive SQL
    - SQL is used directly as DML and DDL through DBS human-machine interfaces
  - dynamic SQL, e.g JDBC, ODBC
  - embedded SQL
    - SQL is embedded in general-purpose programming languages, e.g. ***C*** language
    - executing of general-purpose programming language programs with SQL statement embedded results in DB access

# Embedded SQL

- The SQL standard defines embeddings of SQL in a variety of programming languages such as C, C++, Pascal, Fortran, and Cobol
- A language to which SQL queries are embedded is referred to as a *host language*(宿主语言), and the SQL structures permitted in the host language comprise *embedded* SQL
- Merits of embedded SQL
  - 交互式SQL只能进行DB的访问操作，不能对DB访问结果进行进一步的数据处理
  - Embedded SQL将SQL的数据库访问功能与C语言等宿主语言的数据处理能力相结合，提高了数据应用系统的能力

# Embedded SQL

- **EXEC SQL** statement is used to identify embedded SQL request to the preprocessor

  EXEC SQL <embedded SQL statement >;

- Note: this varies by language:
  - In some languages, like COBOL, the semicolon（分号）is replaced with END-EXEC
  - In Java embedding uses # SQL { …. };

# Embedded SQL (Cont.)

- Before executing any SQL statements, the program must first connect to the database. This is done using:

  EXEC-SQL **connect to** *server* **user** *user-name* **using** *password*;

  Here, *server* identifies the server to which a connection is to be established.

# Embedded SQL (Cont.)

- Variables of the host language can be used within embedded SQL statements. They are preceded by a colon (:) to distinguish from SQL variables
  - e.g., :*credit_amount*

- Variables used as above must be declared within DECLARE section, as illustrated below. The syntax（语法） for declaring the variables, however, follows the usual host language syntax.

```
EXEC-SQL BEGIN DECLARE SECTION;
        int  credit-amount ;
EXEC-SQL END DECLARE SECTION;
```

# Embedded SQL (Cont.)

- E.g.1 From within a host language, find the ID and name of students who have completed more than the number of credits stored in variable *credit_amount* in the host langue
    - *credit_amount* is the shared variable defined in the declaration part

# C program

```
   EXEC SQL BEGIN DECLARE SECTION
           int credit_amount;
  EXEC SQL   END   DECLARE SECTION
    credit_amount := e.g. input-from-screen-by-users
 EXEC SQL
    select ID, name
    from student
    where tot_cred > :credit_amount
 END-EXEC
```

shared variable defined in host language

# Cursor（游标） in Embedded SQL

- To write an embedded SQL query, we use the

  **declare** *c* **cursor for <SQL query>**

  statement.

  The variable *c* is used to identify the query

```
EXEC SQL
    declare c cursor for
    select ID, name
    from student
    where tot_cred > :credit_amount
 END_EXEC
```

# why using Cursor in Embedded SQL

- *利用Embedded SQL进行查询时，查询结果有可能包括多个元组，此时无法直接将多个元组通过共享变量赋值传递给宿主程序

- *系统开辟专门working区域存放SQL查询的结果关系（也称作中间关系），并利用查询游标c指向此区域。宿主程序根据c指向的查询结果关系集合，使用**open, fetch, close**依次获取结果关系中的各元组

# Example 2

```
EXEC SQL BEGIN DECLARE SECTION
      int  credit_amount;
      char  si, sn;
EXEC SQL   END   DECLARE SECTION
 credit_amount:= input-from-screen-by-users
EXEC SQL
 declare c cursor for
           select ID, name
            from student
            where tot_cred > :credit_amount
 END-EXEC
 EXEC SQL open c END-EXEC
 EXEC SQL fetch c into  :si,  :sn  END-EXEC
 EXEC SQL close c END-EXEC
```

可以用循环语
句依次
取走全部查询
结果

shared variable

# Cursor in Embedded SQL (cont.)

- Usage of cursor in embedded SQL

**declare cursor – open – fetch - close**

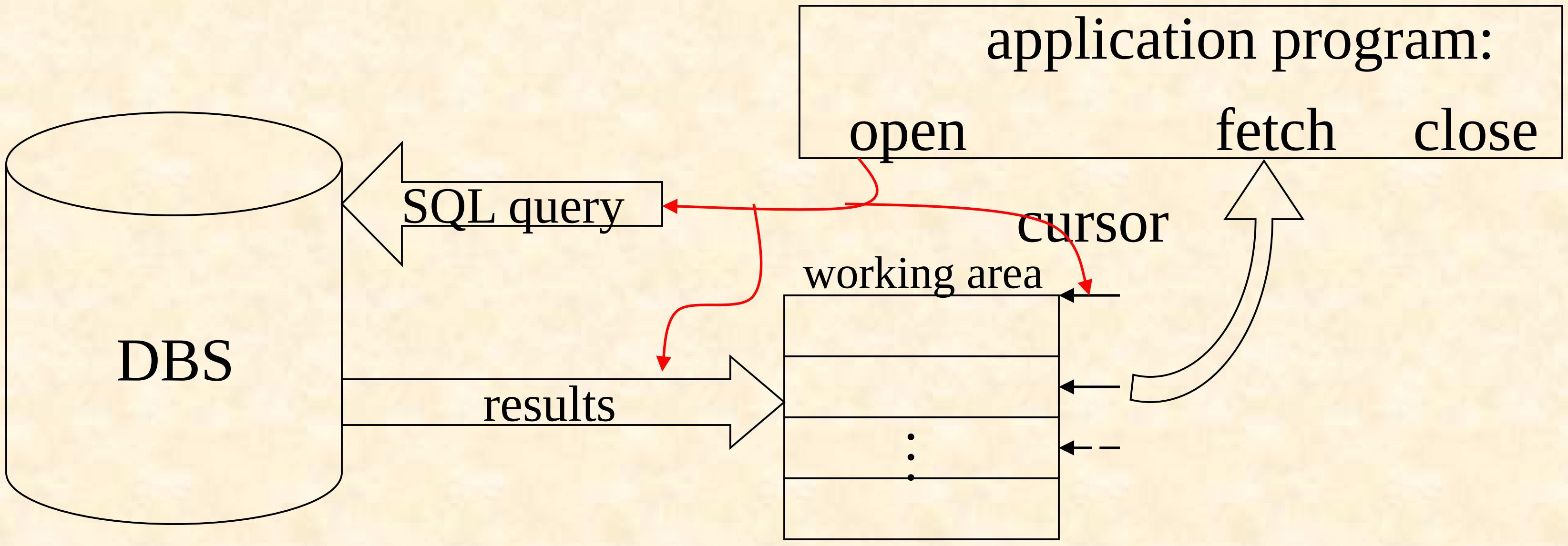


Fig. 5.0.3 Cursor in Embedded SQL

# Cursor in Embedded SQL (cont.)

- The **open** statement for our example is as follows:

  EXEC SQL **open** *c* ;

  This statement causes the database system to execute the query and to save the results within a temporary relation. The query uses the value of the host-language variable *credit-amount* at the time the **open** statement is executed.

- The **fetch** statement causes the values of one tuple in the query result to be placed on host language variables.

  EXEC SQL **fetch** *c* **into** :*si*, :*sn* END_EXEC

  Repeated calls to fetch get successive tuples in the query result

# Cursor in Embedded SQL (cont.)

- A variable called SQLSTATE in the SQL communication area (SQLCA) gets set to ‘02000’ to indicate no more data is available
- The **close** statement causes the database system to delete the temporary relation that holds the result of the query.

  EXEC SQL **close** *c* ;

- Note
  - above details vary with language
  - for example, the Java embedding defines Java iterators to step through result tuples.

# SQL Procedures

- The *dept_count* function could instead be written as procedure:

```
create procedure dept_count_proc (in dept_name varchar(20),
                                  out d_count integer)
    begin
        select count(*) into d_count
        from instructor
        where instructor.dept_name = dept_count_proc.dept_name
    end
```

# SQL Procedures

- Procedures can be invoked either from an SQL procedure or from embedded SQL, using the **call** statement.

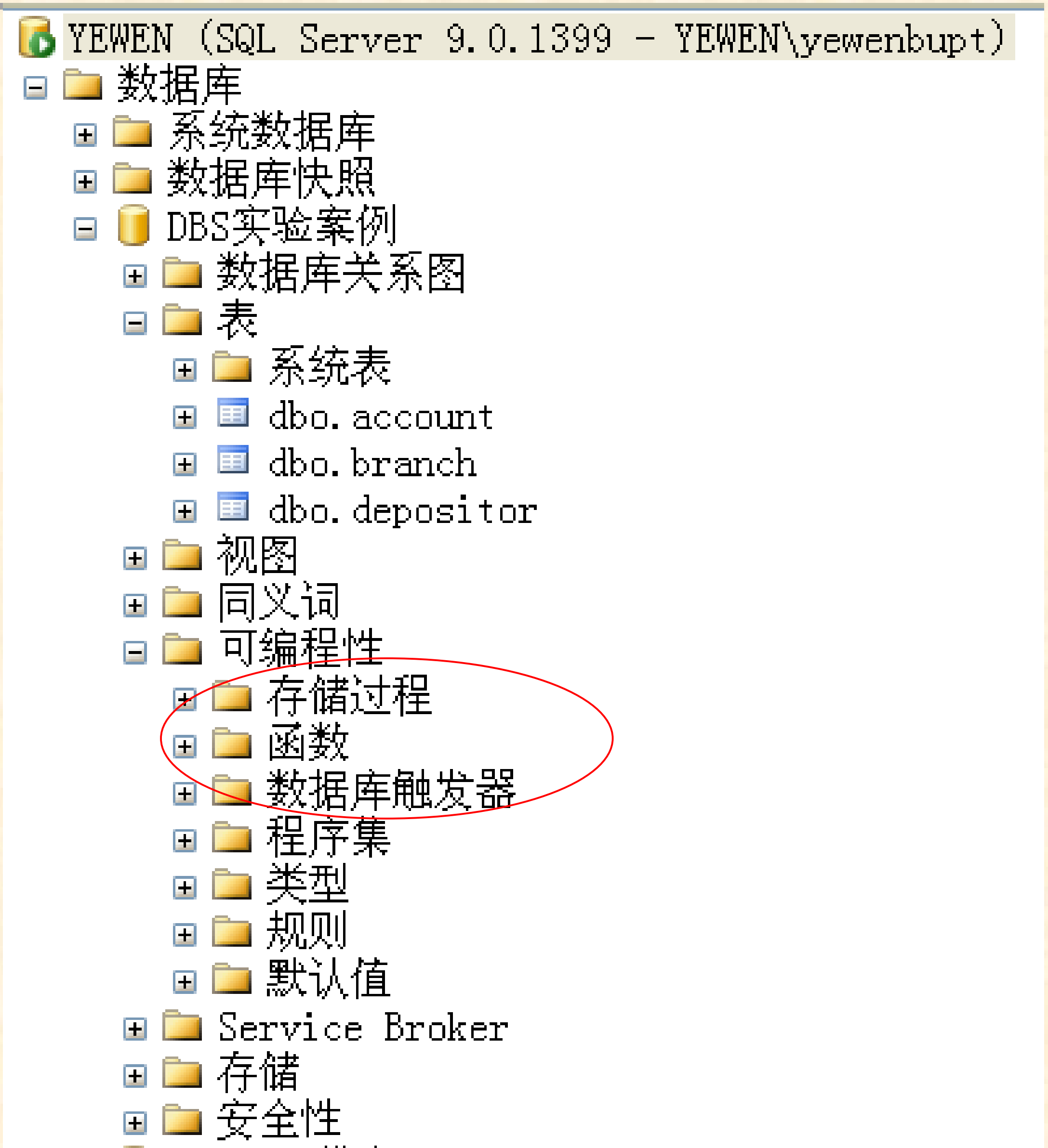
  **declare** *d_count* **integer**;
  **call** *dept_count_proc*( ‘Physics’, *d_count*);

Procedures and functions can be invoked also from dynamic SQL

函数、存储过程预先生成对应的查询执行计划（类似于目标代码），存储在**DBMS**中，应用程序直接调用，不需再进行查询处理和优化（**i.e.**编译）

# Functions and Procedures (cont.)

- The functions and procedures defined are stored in DBS, and can be called by SQL statements or programs



- YEWEN (SQL Server 9.0.1399 - YEWEN\yewenbupt)
  - ⊟ 数据库
    - ⊞ 系统数据库
    - ⊞ 数据库快照
    - ⊟ DBS实验案例
      - ⊞ 数据库关系图
      - ⊟ 表
        - ⊞ 系统表
        - ⊞ dbo.account
        - ⊞ dbo.branch
        - ⊞ dbo.depositor
      - ⊞ 视图
      - ⊞ 同义词
      - ⊟ 可编程性
        - ⊞ 存储过程
        - ⊞ 函数
        - ⊞ 数据库触发器
        - ⊞ 程序集
        - ⊞ 类型
        - ⊞ 规则
        - ⊞ 默认值
      - ⊞ Service Broker
      - ⊞ 存储
      - ⊞ 安全性

# § 5.3 Trigger (触发器)

- **Trigger**
  - a statement that is executed automatically by DBMS as a side effect of a modification to the database

- As an integrity control mechanism, trigger is introduced to SQL standard in SQL:1999, but supported even earlier using non-standard syntax by most databases.
  - Syntax illustrated here may not work exactly on your database system;
  - check the system manuals

# 5.3 Trigger (cont.)

- Trigger is an *event-condition-action* model based integrity definition, checking, remedy（更正） mechanism
  - specify what **event**s cause the trigger to be executed (e.g. insert, delete, update), and under which **conditions** the trigger execution will proceed
    - integrity constraints checking
  - specify the **actions** to be taken when the trigger executes
    - if constraints is violated,  remedy actions are taken

# Triggering Events and Actions in SQL

- Triggers on update can be restricted to specific attributes
  - for example, **after update of** *takes* **on** *grade*
- Values of attributes before and after an update can be referenced
  - **referencing old row as :** for deletes and updates
  - **referencing new row as :** for inserts and updates

# Triggering Events and Actions in SQL

- Triggers can be activated before an event, which can serve as extra constraints.

- For example,  convert blank grades to null.

```
create trigger setnull_trigger before update of takes
referencing new row as nrow
for each row
when (nrow.grade = ‘ ‘)
begin atomic
        set nrow.grade = null;
end;
```

优点：
预先编译（查询处理
优化），存储在
DBMS中，直接调用

# Example: Trigger to Maintain credits_earned value

```
create trigger credits_earned after update of takes on (grade)
referencing new row as nrow
referencing old row as orow
for each row
when nrow.grade <> ’F’ and nrow.grade is not null
    and (orow.grade = ’F’ or orow.grade is null)
begin atomic
     update student
     set tot_cred= tot_cred +
           (select credits
            from course
            where course.course_id= nrow.course_id)
     where student.id = nrow.id;
end;
```

# Conclusion

- Accessing SQL From a Programming Language
  - dynamic SQL, e.g. JDBC, ODBC, ADO, …
  - Embedded SQL
- Procedure
- Triggers

# Appendix A: ODBC

- Architecture of ODBC
  - refer to Fig. 4.0.6
  - ODBC利用***driver***来适应不同厂家数据库的物理结构，如分块大小、访问方式

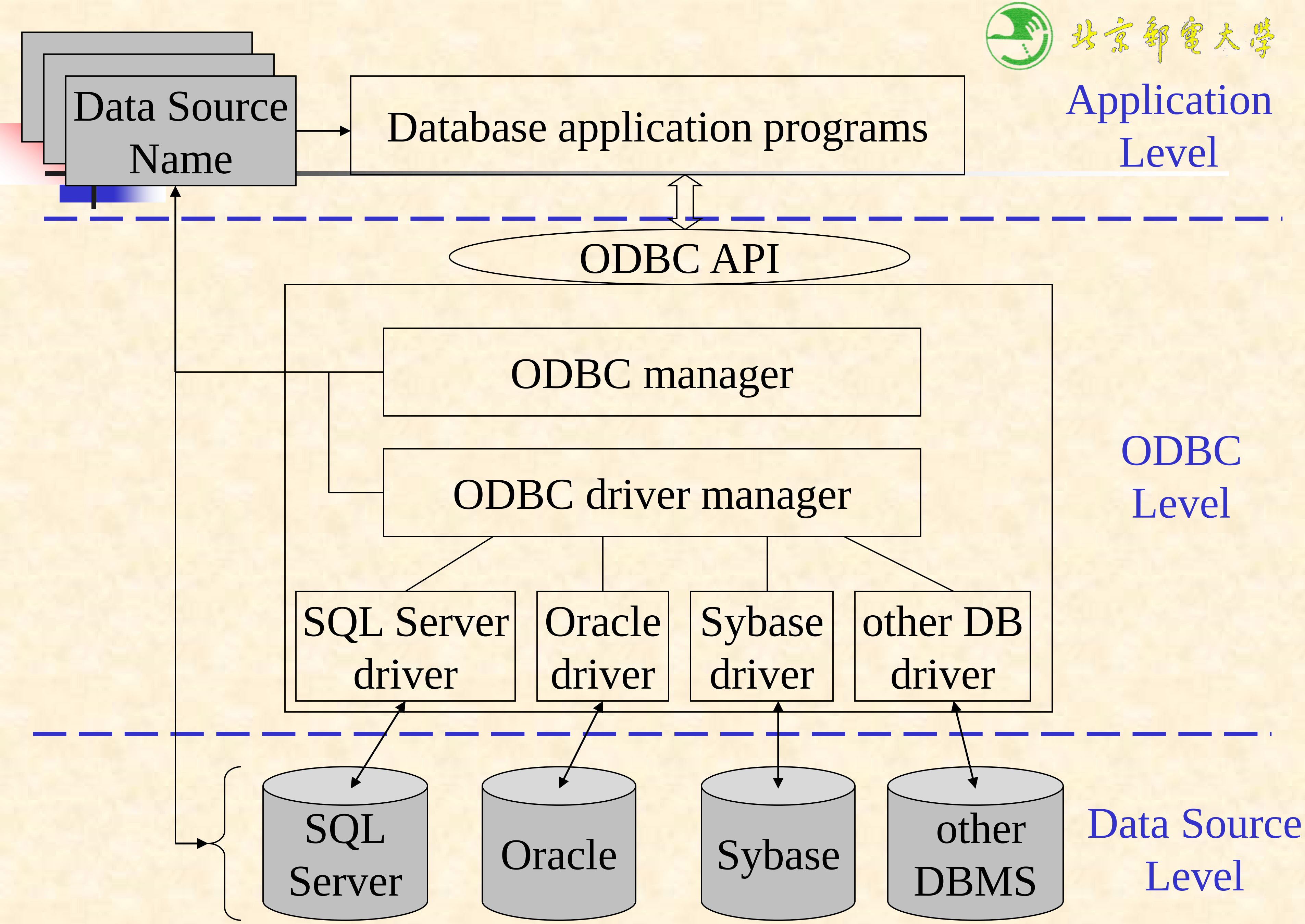
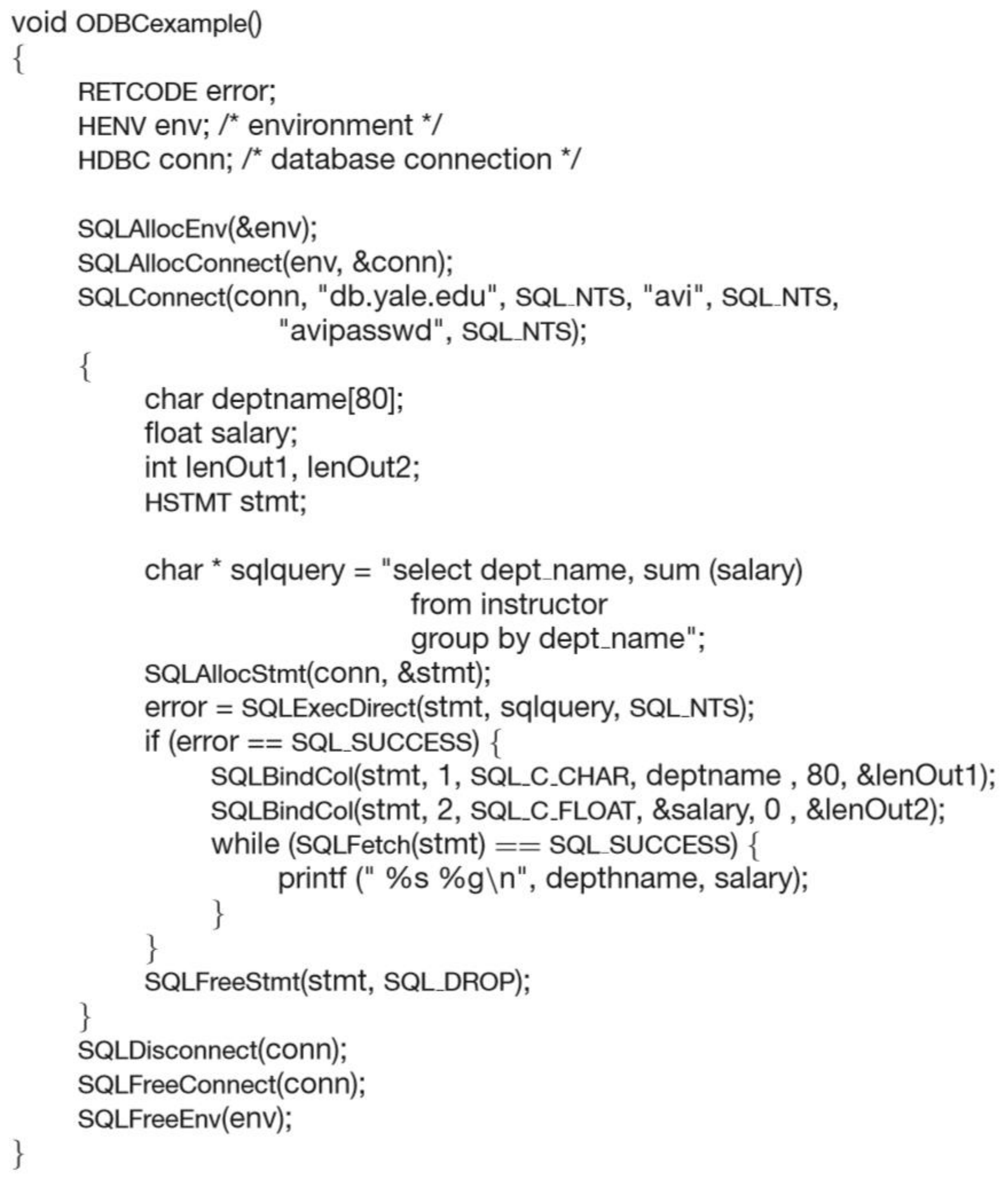


Fig.5.0.2 ODBC Architecture



```
void ODBCexample()
{
        RETCODE error;
        HENV env; /* environment */
        HDBC conn; /* database connection */

        SQLAllocEnv(&env);
        SQLAllocConnect(env, &conn);
        SQLConnect(conn, "db.yale.edu", SQL_NTS, "avi", SQL_NTS,
                        "avipasswd", SQL_NTS);
        {
                char deptname[80];
                float salary;
                int lenOut1, lenOut2;
                HSTMT stmt;

                char * sqlquery = "select dept_name, sum (salary)
                                         from instructor
                                         group by dept_name";
                SQLAllocStmt(conn, &stmt);
                error = SQLExecDirect(stmt, sqlquery, SQL_NTS);
                if (error == SQL_SUCCESS) {
                        SQLBindCol(stmt, 1, SQL_C_CHAR, deptname , 80, &lenOut1);
                        SQLBindCol(stmt, 2, SQL_C_FLOAT, &salary, 0 , &lenOut2);
                        while (SQLFetch(stmt) == SQL_SUCCESS) {
                                printf (" %s %g\n", depthname, salary);
                        }
                }
                SQLFreeStmt(stmt, SQL_DROP);
        }
        SQLDisconnect(conn);
        SQLFreeConnect(conn);
        SQLFreeEnv(env);
}
```

# Appendix B: Updates Through Embedded SQL

Embedded SQL expressions for database modification (**update**, **insert**, and **delete**)

Can update tuples fetched by cursor by declaring that the cursor is for update

```
EXEC SQL
        declare c cursor for
        select *
        from instructor
        where dept_name = ‘Music’
        for update
```

# Appendix B: Updates Through Embedded SQL

We then iterate through the tuples by performing **fetch** operations on the cursor (as illustrated earlier), and after fetching each tuple we execute the following code:

```
update instructor
set salary = salary + 1000
where current of c
```