

Introduction

Purpose of DBS.

- ① Data redundancy (冗余) and inconsistency (不一致) : 数据重复或发生改变
- ② Data isolation (数据孤岛)
- ③ Difficulty in accessing data.
- ④ Integrity problems (完整性问题), 约束条件改变, 数据选择发生变化
- ⑤ Atomicity problems (原子性问题), 某一操作不可被分割 (在发生异常时 (Eg 断电), 将进行到一半的操作复原)
- ⑥ Concurrent access anomalies (并发访问异常)
- ⑦ Security problems

Characteristics of DB

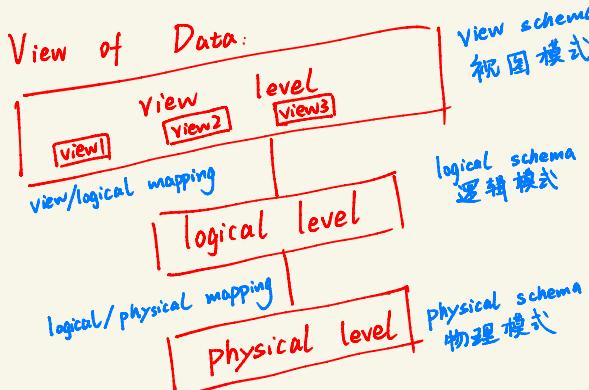
- ① data persistence
- ② convenience in accessing data
- ③ data integrity
- ④ concurrency control for multiple users
- ⑤ failure recovery
- ⑥ security control

Relational Model : 关系模型. 以表格形式组织的数据库.

其它还有:
object-oriented model models
object-relational model models
semistructured data models 半结构化
entity relationship model 实体联系模型

Relational Model.

- 术语变换:
 - table → relation (关系)
 - columns → attributes (属性)
 - rows → tuples (元组)



好处:

- ① Hide complexity
- ② Enhance the adaption to changes
 - physical data independence.
 - logical data independence

DDL, Data Definition Language
DML, Data Manipulation Language

metadata, 元数据, 用来定义数据的数据(也存放在DB中)

primary key: 主键

两种语言: { procedural language 过程式 (可编译的)

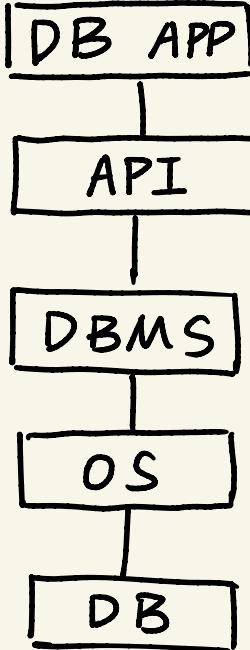
declarative language 陈述式 (其中很常用的一种是 SQL)

host language 宿主语言 (C, C++, Python ...)

调用 DB: ① 利用 API (Application Program Interface) Eg. ODBC, JDBC

两种方法 ② embedded SQL: 嵌入式

DBMS (Data Engine) { storage manager
query processor } file manager
transaction management transaction manager
authorization and integrity manager
事务管理



Structure of Relational Database

I. Concepts

A relation is a set of n-tuples $(a_1, a_2, a_3, \dots, a_n)$ where each $a_i \in D_i$.

A_1, A_2, \dots, A_n are attributes. (象)

$R = (A_1, A_2, \dots, A_n)$ is a relation schema.

$r(R)$ is a relation instance defined over schema R .

Structure of Relational Databases

1. Relation 的概念:

D_1, D_2, \dots, D_n 是 n 个集合, 对一个 n -tuple (a_1, a_2, \dots, a_n) , 其中 $a_i \in D_i$ 则称这个 n 元组是一个“关系”

A_1, A_2, \dots, A_n are attributes. (属性)

$R = (A_1, A_2, \dots, A_n)$ is a relation schema. (关系模式)

$r(R)$ is a relation instance (实例) r defined over R .

2. Keys

- ① super key (超键), 任意含有 candidate key 的集合
- ② candidate key (候选键). 能唯一确定是 which instance 的 attribute. (可空)
- ③ primary key (主键): 从 candidate key 中选一个.
- ④ foreign key (外键), 若关系 R 中的属性 K 是其它关系的主键值, 则在 K 模式中, R 被称为外键
- ⑤ Referential Integrity (参照完整性), 类似于外键限制, 但 R 不限于主键.

3. Relational Algebra

- ① select 6: 筛选行

$\delta_p(r) = \{t \mid t \in r \text{ and } p(t)\}$, p is called selection predicate.

- ② project Π : 筛选列

$\Pi_{A_1, A_2, \dots, A_k}(r)$, 仅保留 r 的 A_1, A_2, \dots, A_k 属性, 并去重.

③ Union: \cup

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

前提: r 和 s 需拥有相同列数, 相同(类似属性)

④ Set Difference: $-$

$$r - s = \{t \mid t \in r \text{ and } t \notin s\}$$

⑤ Cartesian Product: \times

$$r \times s = \{tq \mid t \in r \text{ and } q \in s\}$$

⑥ Rename: $P_i(r)$, 将 r 重命名为 i .

Additional Operations

① Set intersection: \cap 前提: 有相同的列

$$r \cap s = \{t \mid t \in r \text{ and } t \in s\} = r - (r - s)$$

② Natural-Join \bowtie
相同属性要有相同值. 先进行 \times , 再进行 δ 、 π

③ Theta-Join: \bowtie_θ : $r \bowtie_\theta s = \delta_\theta(r \times s)$ 条件连接

④ Outer-Join:

$$\text{Left } r \bowtie s = (r \bowtie s) \cup (r - \pi_r(r \bowtie s) \times \{\text{null}, \text{null}, \dots, \text{null}\})$$

$$\text{Right } r \bowtie s = \dots \quad r \bowtie s = \dots$$

哪边开口就需要包括那边的全部元组.

④ Semijoin : Δ_θ

$$\Gamma \Delta_\theta S = \Pi_R (\Gamma \Delta_\theta S) \text{ 保留 } \Gamma \text{ 中能以 } \theta \text{ 条件与 } S \text{ 相连的列}$$

⑤ Devision: $\Gamma \div S$ is the largest relation $\tau(R-S)$ such that $\tau \times S \sqsubseteq \Gamma$

Aggregate Function 聚合函数: $y = f(x_1, x_2, \dots, x_n)$ (多元)

Eg: max, min, avg, sum, count

结果:

Aggregate Operation: $G_1 G_2 \dots G_k \quad F_{F_1(A_1)} F_{F_2(A_2)} \dots E$

G_i is the attribute on which to group.

F_i is aggregation function. A_i is an attribute name.

G	F(A)

4. Multiset 多重集

- **selection:** has as many duplicates of a tuple as in the input, if the tuple satisfies the selection
- **projection:** one tuple per input tuple, even if it is a duplicate
- **cross product:** If there are m copies of $t1$ in r , and n copies of $t2$ in s , there are $m \times n$ copies of $t1.t2$ in $r \times s$
- set operators
 - **union:** $m + n$ copies
 - **intersection:** $\min(m, n)$ copies
 - **difference:** $\min(0, m - n)$ copies

Data Definition

1. Domain Types in SQL

- ① char(n)
- ② varchar(n)
- ③ int
- ④ smallint
- ⑤ numeric(p,d) : p位有效数字, d位小数
- ⑥ real 浮点数
- ⑦ double precision 双精度浮点数
- ⑧ float(n) 浮点数, 至少n位精度

2. Built-in Data Types in SQL

- ① date (年.月.日)
- ② time (时分秒)
- ③ timestamp (date + time)
- ④ interval

2'. Time Functions :

current_date(), current_month(), year(x), month(x), day(x),
hour(x), minute(x), second(x)

建表 : create table r (

A ₁	D ₁ ,
A ₂	D ₂ ,
⋮	
A _n	D _n , 也可以在这里添加 not null 限制
	default
	primary key (A _i),
	foreign key (A _j) reference s
);

对 foreign key：可以规定，

on delete/update restrict / cascade / set null / set default
不可更改 一起更改 置空 特别设置

修改表本身：

drop table r 把r表删了

{ alter table r add A D 从表中插入/删除属性
} alter table r drop A

修改表内容：

delete from r 把r表的具体数据都删了

delete from r where P 把r表中满足条件的数据删了

insert into r

values (~, ~, ~, ...); 往r表中插入新数据

或

insert into r

select ----- 从其它表中选部分数据插入r表

update r
set A = ~
where P

→ 可以是数学表达式，也可以是 select ...
把r表中符合条件P的元组中的A属性数据进行更新。

update r
set A = case
when P the x
else y
end

更新r表中的A属性值，若满足P条件，
则更新为x；不满足则更新为y。

select (distinct / all) A_i / * from M where P.

去重 不去重 默认 all attributes

M can be several tables

r_1, r_2 means Cartesian Product.

r_1 natural join r_2 自然连接 (同理可用其它连接)

r_1 join r_2 using (A_i) Theta连接, 要求 A_i 值相等

P: several conditions, using:

and / or / not / between ~ and / = / < > 不等

like, 字符串对比符号.

{ % 百分号匹配任意子字符串.

 } _ 下划线匹配任意字符

Eg: select name

from instructor

where name like '%dar_'

如果要匹配 % 或 _ 本身, 则有3种方法.

① like '\%'

② like '\%' escape '\'

③ like '#%' escape '#'

select ~ as T. 把 ~ 成为新表 T

按顺序显示查询得到的元组：

order by A_i desc/asc 根据 A_i 降/升序排列

限制返回元组的条数：

limit offset, row_count 返回从第 offset 条开始的 row_count 条
(limit rowcount = limit 0, row-count)

Set Operation

union、intersect、except 并、交、减且去重

union all、intersect all、except all 不去重

带有 null 的运算结果一定是 null.

Aggregate Functions (会自动忽略 null 值)

avg、min、max、sum、count (如果是 count(*) 则统计返回条数)

group by A_i 按 A_i 分类

select A_i, AggregateF(A_j)

from r where p

group by A_i

① 除聚合函数外被选择的属性一定要出现在 group list 中

② 最终结果形式：

A _i	函数值
--	--

group by 本身只是改变元组顺序，将一类的元组放在一起。
(Eg: select * from group by A; 效果是将r表重新排序)
只有当配合聚合函数后，才会有其它效果
对 group by 后的结果进行筛选。

having -----

Eg. 选出薪水起1000员工数不小于10的部门名以及对应员工数量。 ✓

```
select dept_name, count(*) as cnt          筛选顺序(步骤).
      from instructor where salary >= 1000 ① 仅剩下薪水>1000的数据
      group by dept_name ② 先将原表按 dept_name 分类重排
      having count(*) > 10 ③ 统计余下数据, 不同部门的元组数, 若大
      order by cnt          于10则可以留下。
```

嵌套查询

in / not in (set membership)
select ~ from ~ where Ai in (select ~ ...)

Some / all (set comparison)

找出比生物部其中一人工资高的球员名字。

① select distinct T.name
 from instructor as T, instructor as S
 where T.salary > S.salary and S.dept_name = 'Biology'
② select name from instructor
 where salary > some (select salary from instruct
 where dept_name = 'Biology')

空关系检测 exists/not exists

Eg. 找到上了所有生物系课程的学生.



```
select distinct S.ID, S.name from student as S  
where not exists (
```

```
(select course_id from course where dept-name = 'Biology')  
except (select T.course_id from takes as T where S.ID = T.ID))
```

思路：不存在该学生没选的生物课。

⇒ 选出生物系的所有课A，选出该学生选的课B，

如果 A-B 是空，就意味着该学生的课覆盖了前者。

多重集检测 unique : at most once (0或1)

where unique (select ~--)

unique + exist ⇒ 怪好 VR

构造临时表 with

With 新表名 (新表属性列表) as (select -----)

中级SQL

Join types
inner join
left outer join
right outer join
full outer join

Join Conditions
natural
on <predicate>
using (A_1, A_2, \dots, A_n)

自然连接
条件连接
等值连接

用户定义数据类型

create type M as numeric(12,2) final (将M定义为M)

Domains can have constraints!

大型对象在SQL里被存储为 Binary Large Object (BLOB)

TinyBlob 0~255B

Blob 0~64 KB

MediumBlob 0~16 MB

LargeBlob 0~4 GB

Integrity Constraints 约束

① not null ② primary key ③ unique ④ check

unique (A_1, A_2, \dots, A_m)

A_1, A_2, \dots, A_m 即形成 super key

也就是说除了 primary key 之外也要保持不同。

check (P) 检查条件 P 是否成立

一般来说，检查在某个事务结束后满足约束条件即可

create assertion M check (not exists (A))

如果 A 成立，则中断 \langle assertion-name \rangle

也就是说且保证 A 不存在 / 不成立

View

create view V as <query expression>

好处：①简化用户视野 ②方便查询书写
③有利于权限控制 ④有独立性

Update view

通过 view 修改，相当于通过这个 view 对原表修改。

例如对 view 进行插入，原表中也都会有，没有的数据栏填 null.

物化视图

create materialized view ----- 产生一张临时表。

Index

create index <index-name> on T(A_i)

在 T 表的 A_i 属性上建立索引。（物理层面的实现）（B+ 树）（否则是顺序查找）

Transactions 事务 atomic 原子性

一般自动开始，通过 rollback 或 commit 结束

关闭方法：SET AUTOCOMMIT = 0；

一旦关闭自动提交后，每完成一项事务，都要操作指令：COMMIT.

ACID:

Atomicity 原子性

Consistency - 一致性

Isolation 独立性

Durability 持久性

Authorization 授权

① 数据层面：修改数据

(1) select (2) insert (3) update (4) delete

② 结构层面：

(1) resources /create : 创建新关系

(2) alteration : 新建 / 删除某属性

(3) drop : 删除关系 (4) index , 新增 / 删除索引 (5) create view

把权限给用户

就是数据层面的4次内容
或 all

grant <privilege list> on

<relation name or view name> to <user list>

■ grant select on instructor to U_1, U_2, U_3

■ grant select on department to public

■ grant update (budget) on department to U_1, U_2

■ grant all privileges on department to U_1

a user-id /
public (所有合法用户) /
A role

收回用户权限

revoke <privilege list> on

<relation name or view name> from <user list>

role：角色

user：用户

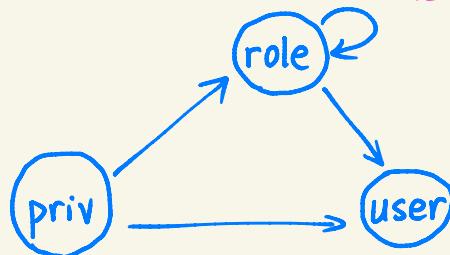
可以简单理解为：“角色”类似于“类”？

Eg. 一个 3000 人的公司，每个员工都是一个 user。
这 3000 人中有 50 人是领导，拥有管理权限，“领导”就是一个 role。
在修改权限时，可以针对某个类修改权限。

create role <role-name> 创建 role

grant <role-name> to <user-name> 把某 user 归为 role

grant <role-name> to <role-name> role 可以被归为其它 role



grant reference <attribute> on <schema/view> to <user/role>
允许用户在创建关系时，引用该表的该属性作为 foreign key

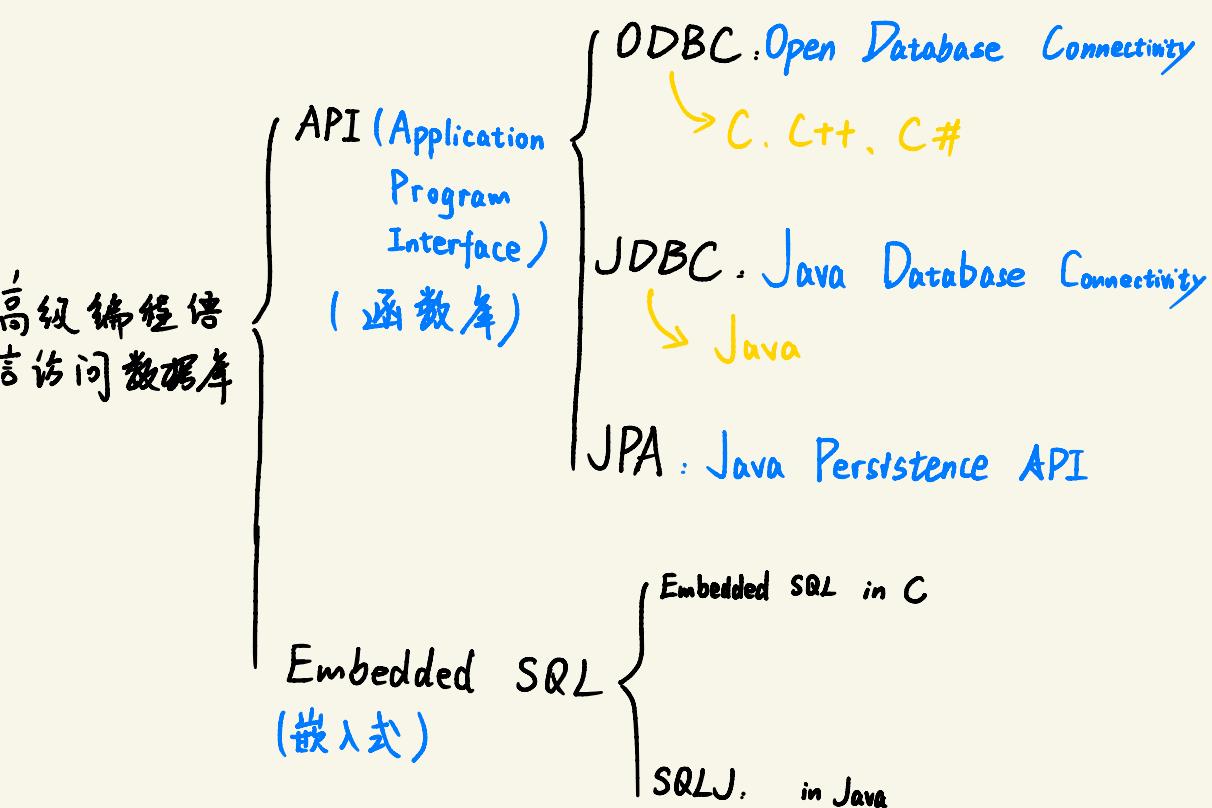
权限的传递

若在 grant 语句后加 with grant option，则

被赋予权限的对象有权力将此权限传下去

若在 revoke 语句后加 {
 cascade 级联反应，收回所有权限
 restrict 仅收回该用户的权限

Revoke grant option for <privilege list> on <relation/view name> from <user/role>
收回用户被授的权力



API 的步骤：

- ① 建立连接 Connect with DB server.
- ② 发送指令集 Send SQL commands to DB server.
- ③ 拿回结果 Fetch tuples of result into program variables.

ODBC

```
int ODBCexample() {  
    RETCODE error;  
    HENV env;  
    HDBC conn;
```

定义变量，储存 ODBC 函数的返回状态
环境句柄，用于管理环境
连接句柄，用于连接数据库

SQLAllocEnv (&env); 分配环境句柄

SQLAllocConnect (env, &conn); 分配连接句柄，并与环境句柄关联。
连接句柄 数据库地址 字符串长度 用户名 密码

SQLConnect (conn, ".....", SQL_NTS, ".....", SQL_NTS, ".....", SQL_NTS);

Body

//SQL_NTS 的作用：
忽略字符串类型，自动计算其长度

SQLDisconnect (conn); 断开连接

SQLFreeConnect (conn); 释放原原因申请的句柄
SQLFreeEnv (env);

}

ODBC 的特点：

- ① 搜索库中的全部 relation
- ② 截取 列名以及其类型
- ③ 每条 SQL 语句都被默认为一条事务，关闭方法。

SQLSetConnectOption (conn, SQL_AUTOCOMMIT, 0);

SQLTransact (conn, SQL_COMMIT);

SQLTransact (conn, SQL_ROLLBACK);

ODBC - BODY

```
char deptname [80],          // 定义存放 DB 输出结果的变量
float salary;
int lenOut1, lenOut2;         // 定义储存结果集列长度的变量
HSTMT stmt;                 // 定义句柄用于执行 SQL 语句
char * sqlquery = "select dept-name, sum(salary) -----";  
                         // 具体的 SQL 语句
SQLAllocStmt (conn, &stmt);   // 分配语句句柄并与 DB 相连
error = SQLExecDirect (stmt, sqlquery, SQL_NTS);  
                         // 具体 SQL 语句通过 SQLExecDirect 执行
if (error == SQL_SUCCESS) {
    SQLBindCol (stmt, 1, SQL_C_CHAR, deptname, 80, &lenOut1);
    SQLBindCol (stmt, 2, SQL_C_FLOAT, &salary, 0, &lenOut2);
    // 用 SQLBindCol 将查询结果分配给 C 语言变量
    while (SQLFetch (stmt) == SQL_SUCCESS) {
        printf ("%s %g\n", deptname, salary);
    } // 遍历结果集中的每一行
}
SQLFreeStmt (stmt, SQL_DROP); // 释放句柄
```

ODBC Prepared Statement

1. 准备 : SQLPrepare (stmt, < SQL_String >)

2. 绑定参数 : SQLBindParameter (stmt, < parameter#>, -----)

3. 执行 : SQLExecute (stmt);

区别在于：1. 先绑参数 后执行

2. SQL 语句首先在 DB 中被编译，然后后端被执行一次或多次。

3. 可以有占位符！ insert into account values (?, ?, ?)

JDBC

```
public static void JDBCexample ( String dbid, String userid, String passwd ) {  
    try {  
        Connection conn = DriverManager.getConnection ("jdbc:oracle:thin:", userid, passwd);  
        Statement stmt = conn.createStatement ();  
        BODY  
        stmt.close (); // 关闭 statement 对象  
        conn.close (); // 关闭数据库连接  
    }  
    catch (SQLException sqle) { // 异常检测  
        System.out.println ("SQLException : " + sqle);  
    }  
}
```

更新：

```
try { Stmt.executeUpdate ("insert -----"); }  
catch (SQLException sqle) { System.out.println ("Insertion fails: " + sqle); }
```

查询、打印

```
ResultSet rset = stmt.executeQuery ("select --- ");  
while (rset.next ()) {  
    System.out.println (rset.getString ("deptname") + " " + rset.getFloat (2));  
}
```

空值查询：

```
int a = rset.getInt ("a");  
if (rset.wasNull ()) System.out.println ("Got null value");
```

Prepared Statement

PreparedStatement pStmt

= conn.prepareStatement("insert into dep values (?, ?, ?, ?, ?)");

pStmt.setString(1, "876");

pStmt.setInt(2, 66);



插入的内容

插入的位置

pStmt.executeUpdate();

SQL Injection

Eg.: select * from dep where name = ' "+name+" '

若输入: x' or 'Y'=Y

⇒ where name = '

JDBC 的 Transaction Control

conn.setAutoCommit(false);

conn.commit(); conn.rollback();

conn.setAutoCommit(true);

Embedded SQL

SQLJ

```
#sql iterator deptInfoIter (String name, int avgSalary),  
deptInfoIter iter = null;  
# sql iter = { select dept-name, avg(salary) ----- };  
while (iter.next) {  
    String deptName = iter.dept-name();  
    int avgSal = iter.avgSal();  
    System.out.println(deptName + " " + avgSal);  
}  
iter.close();
```

Embedded SQL in C

```
EXEC SQL INCLUDE SQLCA;  
EXEC SQL BEGIN DECLARE SECTION;  
char account-no[10];  
char branch-name[10];  
int balance;  
C 变量声明  
EXEC SQL END DECLARE SECTION;  
EXEC SQL CONNECT TO Bank-db USER A Using Ee;  
scanf ("%s %s %d", account-no, branch-name, balance);  
EXEC SQL insert into account values  
(:account-no,:branch-name,:balance);  
If (SQLCA.sqlcode != 0) printf("error");  
else printf ("success!");
```

Procedural Constructs

1. SQL Function

```
create function function_name ( attribute_name type )
return integer ← return type
begin
    declare attribute_name_1 type - 1;
    select count(*) into attribute_name_1
    where ----- (根据输入做筛选)
    return attribute_name_1
end
```

也可以返回 Table !

```
create function instructors_of(dept_name char(20) )
    returns table ( ID varchar(5),
                    name varchar(20),
                    dept_name varchar(20),
                    salary numeric(8,2))

return table
    (select ID, name, dept_name, salary
     from instructor
     where instructor.dept_name =
instructors_of.dept_name)
```

2. SQL Procedures

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
```

- 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

3. 语句

① begin -----

end

④ for r as -----

do

② while ~ do

end while

end for

③ repeat -----

until ~

end repeat

⑤ if ~ then ---

elseif

then ---

else ---

endif

Triggers

```
create trigger grade_trigger after update of takes on grade
referencing new table as new_table
for each statement
when exists( select avg(grade)
              from new_table
              group by course_id, sec_id, semester, year
              having avg(grade)< 60 )
begin
    rollback
end
```

利用 triggers 防止对 keys 的修改。

```
create trigger grade_trigger after update of takes on grade
referencing new table as new_table
for each statement
when exists( select avg(grade)
              from new_table
              group by course_id, sec_id, semester, year
              having avg(grade)< 60 )
begin
    rollback
end
```