

高级SQL

讲解人：李鸿岐

事务

高级SQL

❑ 事务(Transaction)

- A transaction is a sequence of database statements that needs to execute atomically.

❑ A database transaction consists of one of the following:

- DML statements which constitute one consistent change to the data.
- One DDL statement
- One DCL statement

❖ **事务(Transaction)**是用户定义的一个**数据库操作序列**，这些操作要么全做，要么全不做，是一个不可分割的工作单位。

❖ 事务是恢复和并发控制的基本单位


高级SQL

□ 事务(Transaction)

❖ 事务和程序是两个概念

- 在关系数据库中，一个事务可以是一条**SQL**语句，一组**SQL**语句或整个程序
- 一个程序通常包含多个事务

id	balance
01	100
02	100
...	..



```
UPDATE account
SET balance = balance - 10
WHERE id = '01';

UPDATE account
SET balance = balance + 10
WHERE id = '02';
```

高级SQL

- Beginning and end of transaction
 - Implicitly declare
 - Explicitly declare (begin transaction, commit / rollback)
- Beginning and end of transaction in PostgreSQL through interactive terminal

❖ 显式定义方式

BEGIN TRANSACTION

SQL 语句1

SQL 语句2

。 。 。 。 。

COMMIT

BEGIN TRANSACTION

SQL 语句1

SQL 语句2

。 。 。 。 。

ROLLBACK

❖ 隐式方式

当用户没有显式地定义事务时，
数据库管理系统按缺省规定自动划分事务

❖ 显式定义方式

BEGIN TRANSACTION

SQL 语句1

SQL 语句2

。 。 。 。 。

COMMIT



- 事务正常结束
- 提交事务的所有操作（读+更新）
- 事务中所有对数据库的更新写回到磁盘上的物理数据库中

BEGIN TRANSACTION

SQL 语句1

SQL 语句2

。 。 。 。 。

ROLLBACK



- 事务异常终止
- 事务运行的过程中发生了故障，不能继续执行
- 系统将事务中对数据库的所有已完成的操作全部撤销
- 事务滚回到开始时的状态

高级SQL

□ 案例

UPDATE account

SET balance = balance - 10

WHERE id = '01';

UPDATE account

SET balance = balance + 10

WHERE id = '02';



BEGIN TRANSACTION (implicit)

UPDATE account

SET balance = balance - 10

WHERE id = '01';

COMMIT (implicit)

BEGIN TRANSACTION (implicit)

UPDATE account

SET balance = balance + 10

WHERE id = '02';

COMMIT (implicit)

高级SQL

- Beginning and end of transaction in PostgreSQL through interactive terminal

BEGIN TRANSACTION (explicit)

```
UPDATE account  
SET balance = balance - 10  
WHERE id = '01';
```

```
UPDATE account  
SET balance = balance + 10  
WHERE id = '02';
```

COMMIT (explicit)

**事务的其他特性和多个事务执行时的相互影响参考
后续课程内容《事务模型》、《并发控制》**

完整性约束

❖ 数据库的完整性

■ 数据的正确性

- 是指数据是符合现实世界语义，反映了当前实际状况的

■ 数据的相容性

- 是指数据库同一对象在不同关系表中的数据是符合逻辑的

例如，

- 学生的学号必须唯一
- 性别只能是男或女
- 本科学生年龄的取值范围为14~50的整数
- 学生所选的课程必须是学校开设的课程，学生所在的院系必须是学校已成立的院系
- 等

高级SQL

□ 完整性约束(Integrity Constraints)

❖ 为维护数据库的完整性，DBMS必须：

1. 提供定义完整性约束条件的机制

- 完整性约束条件也称为完整性规则，是数据库中的数据必须满足的语义约束条件
- SQL标准使用了一系列概念来描述完整性，包括关系模型的**实体完整性**、**参照完整性**和**用户定义完整性**
- 这些完整性一般由SQL的数据定义语言语句来实现

2. 提供完整性检查的方法

- 数据库管理系统中检查数据是否满足完整性约束条件的机制称为完整性检查。
- 一般在INSERT、UPDATE、DELETE语句执行后开始检查，也可以在事务提交时检查

3. 违约处理

- 数据库管理系统若发现用户的操作违背了完整性约束条件，就采取一定的动作

拒绝 (NO ACTION)
执行该操作

级联 (CASCADE)
执行其他操作

高级SQL

□ 完整性约束(Integrity Constraints)

实体完整性约束

❖ 关系模型的实体完整性

- CREATE TABLE中用PRIMARY KEY定义

❖ 单属性构成的码有两种说明方法

- 定义为列级约束条件

- 定义为表级约束条件

❖ 对多个属性构成的码只有一种说明方法

- 定义为表级约束条件

[案例] 将Student表中的Sno属性定义为码

(1) 在列级定义主码

```
CREATE TABLE Student  
( Sno CHAR(9) PRIMARY KEY,  
  Sname CHAR(20) NOT NULL,  
  Ssex CHAR(2),  
  Sage SMALLINT,  
  Sdept CHAR(20) );
```

(2) 在表级定义主码

```
CREATE TABLE Student  
( Sno CHAR(9),  
  Sname CHAR(20) NOT NULL,  
  Ssex CHAR(2),  
  Sage SMALLINT,  
  Sdept CHAR(20),  
  PRIMARY KEY (Sno) );
```

高级SQL

□ 完整性约束(Integrity Constraints)

实体完整性约束

❖ 关系模型的实体完整性

- CREATE TABLE中用PRIMARY KEY定义

❖ 单属性构成的码有两种说明方法

- 定义为列级约束条件
- 定义为表级约束条件

❖ 对多个属性构成的码只有一种说明方法

- 定义为表级约束条件

[案例] 将Student表中的Sno属性定义为码
(1) 在列级定义主码

CREATE TABLE Student

[案例] 将SC表中的Sno, Cno属性组定义为码

CREATE TABLE SC

**(Sno CHAR(9) NOT NULL,
Cno CHAR(4) NOT NULL,
Grade SMALLINT,**

PRIMARY KEY (Sno,Cno) /*只能在表级定义主码*/

);

Sdept CHAR(20),

PRIMARY KEY (Sno));

❑ 实体完整性检查和违约处理

❖ 插入或对主码列进行更新操作时，关系数据库管理系统按照实体完整性规则自动进行检查。包括：

- 检查主码值**是否唯一**，如果不唯一则拒绝插入或修改
- 检查主码的各个属性**是否为空**，只要有一个为空就拒绝插入或修改

待插入记录

Keyi	F2i	F3i	F4i	F5i
------	-----	-----	-----	-----

基本表

Key1	F21	F31	F41	F51
Key2	F22	F32	F42	F52
Key3	F23	F33	F43	F53
⋮				

❖ 检查记录中主码值是否唯一的一种方法是进行**全表扫描**

- 依次判断表中每一条记录的主码值与将插入记录上的主码值（或者修改的新主码值）是否相同

❑ 实体完整性检查和违约处理

❖ 表扫描缺点

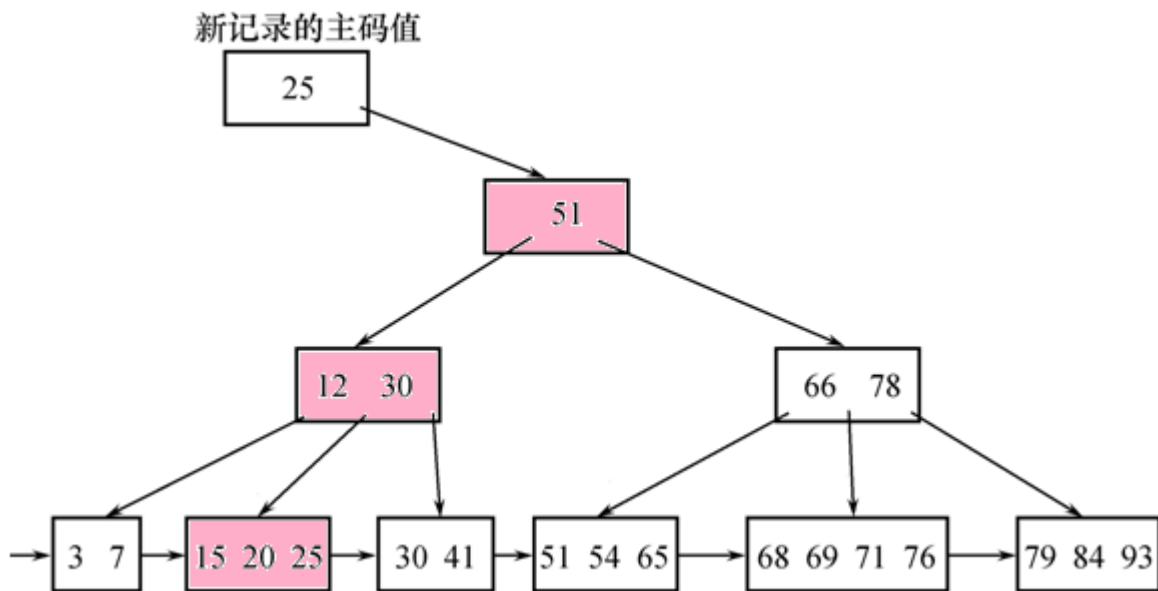
■ 十分耗时

❖ 为避免对基本表进行全表扫描，RDBMS核心一般都在主码上自动建立一个索引

❖ B+树索引

■ 新插入记录的主码值是25

- 通过主码索引，从B+树的根结点开始查找
- 读取3个结点：根结点（51）、中间结点（12 30）、叶结点（15 20 25）
- 该主码值已经存在，不能插入这条记录



高级SQL

□ 完整性约束(Integrity Constraints)

参照完整性约束

❖ 关系模型的参照完整性

- 在CREATE TABLE中用FOREIGN KEY定义哪些列为外码
- 用REFERENCES指明这些外码参照哪些表的主码

- ❖ 一个参照完整性将两个表中的相应元组联系起来
- ❖ 对被参照表和参照表进行增删改操作时有可能破坏参照完整性，必须进行检查

- 关系SC中 (Sno, Cno) 是主码。Sno, Cno分别参照Student表的主码和Course表的主码

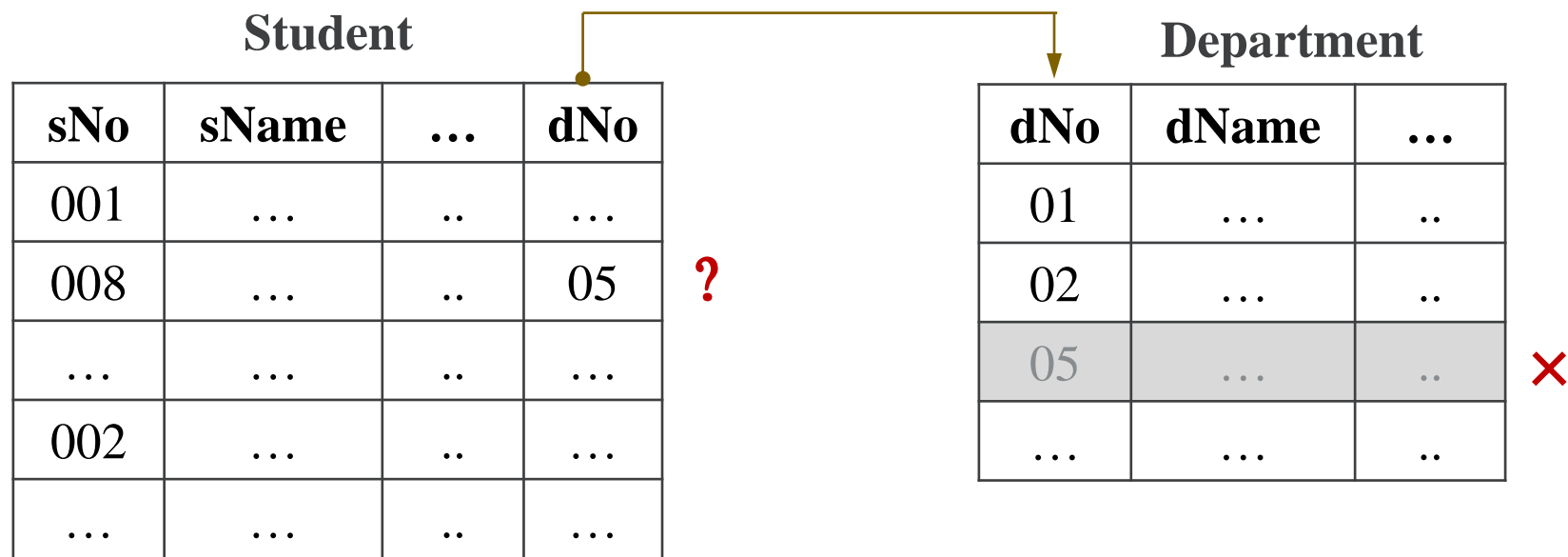
[案例]定义SC中的参照完整性

```
CREATE TABLE SC
( Sno   CHAR(9) NOT NULL,
  Cno   CHAR(4) NOT NULL,
  Grade SMALLINT,
  PRIMARY KEY (Sno, Cno), /*在表级
定义实体完整性*/
  FOREIGN KEY (Sno)
REFERENCES Student(Sno),
/*在表级定义参照完整性*/
  FOREIGN KEY (Cno)
REFERENCES Course(Cno)
/*在表级定义参照完整性*/
);
```


高级SQL

□ 完整性约束(Integrity Constraints)

插入、更新和删除操作可能导致的数据不一致性问题



**DELETE FROM department
WHERE dNo='05';**

ERROR: update or delete on table "department" violates foreign key constraint "student_dno_fkey" on table "student"
SQL 状态: 23503
详细:Key (dno)=(05) is still referenced from table "student".

□ 完整性约束(Integrity Constraints)

插入、更新和删除操作可能导致的数据不一致性问题

dNo	dName	...	dean
01	001
02
...
...

INSERT INTO department
VALUES('01', ..., '001');

?

tNo	tName	dID
001	...	01
002
003
...

INSERT INTO teacher
VALUES('001', ..., '01');



完整性约束(Integrity Constraints)

表1 可能破坏参照完整性的情况及违约处理

被参照表（例如Student）	参照表（例如SC）	违约处理
可能破坏参照完整性 ←	插入元组	拒绝
可能破坏参照完整性 ←	修改外码值	拒绝
删除元组 →	可能破坏参照完整性	拒绝/级联删除/设置为空值
修改主码值 →	可能破坏参照完整性	拒绝/级联修改/设置为空值

□ 完整性约束(Integrity Constraints)

❖ 参照完整性违约处理

(1) 拒绝 (**NO ACTION**) 执行

- 不允许该操作执行。该策略一般设置为默认策略

(2) 级联 (**CASCADE**) 操作

- 当删除或修改被参照表 (**Student**) 的一个元组造成了与参照表 (**SC**) 的不一致, 则删除或修改参照表中的所有造成不一致的元组

(3) 设置为空值 (**SET-NULL**)

- 当删除或修改被参照表的一个元组时造成了不一致, 则将参照表中的所有造成不一致的元组的对应属性设置为空值。

□ 完整性约束(Integrity Constraints)

[案例] 有下面2个关系



外码

学生 (学号, 姓名, 性别, 专业号, 年龄)

专业 (专业号, 专业名)

- 假设专业表中某个元组被删除, 专业号为**12**
- 按照设置为空值的策略, 就要把学生表中专业号=**12**的所有元组的专业号设置为空值
- 对应语义: 某个专业删除了, 该专业的所有学生专业未定, 等待重新分配专业

□ 完整性约束(Integrity Constraints)

[案例] 显式说明参照完整性的违约处理示例

```
CREATE TABLE SC
```

```
( Sno CHAR(9) NOT NULL,
```

```
  Cno CHAR(4) NOT NULL,
```

```
  Grade SMALLINT,
```

```
  PRIMARY KEY(Sno,Cno),
```

```
  FOREIGN KEY (Sno) REFERENCES Student(Sno)
```

```
    ON DELETE CASCADE      /*级联删除SC表中相应的元组*/
```

```
    ON UPDATE CASCADE,     /*级联更新SC表中相应的元组*/
```

```
  FOREIGN KEY (Cno) REFERENCES Course(Cno)
```

```
    ON DELETE NO ACTION
```

```
    /*当删除course 表中的元组造成了与SC表不一致时拒绝删除*/
```

```
    ON UPDATE CASCADE
```

```
    /*当更新course表中的cno时，级联更新SC表中相应的元组*/
```

```
);
```

□ 完整性约束(Integrity Constraints)

```
ALTER TABLE student  
  ALTER CONSTRAINT student_dno_fkey  
  DEFERRABLE INITIALLY DEFERRED;
```

```
BEGIN TRANSACTION;  
DELETE FROM department  
WHERE dno='05';
```

```
UPDATE student  
SET dno='02'  
WHERE sno='008';
```

```
commit;
```

```
ALTER TABLE XXX  
  ALTER CONSTRAINT XXX  
  DEFERRABLE  
  INITIALLY DEFERRED;
```

```
BEGIN TRANSACTION;  
INSERT INTO department  
VALUES('01', ..., '001');
```

```
INSERT INTO teacher  
VALUES('001', ..., '01');
```

```
commit;
```

高级SQL

□ 完整性约束(Integrity Constraints)

用户定义完整性约束

❖ 用户定义的完整性:

- 针对某一具体应用的数据必须满足的语义要求
- 关系数据库管理系统提供了定义和检验用户定义完整性的机制, 不必由应用程序承担

❖ 属性上的约束条件

- 列值非空 (**NOT NULL**) ; 列值唯一 (**UNIQUE**) ; 满足某条件 (**CHECK**)

❖ 元组上的约束条件

[案例] 当学生的性别是男时, 其名字不能以Ms.打头。

```
CREATE TABLE Student
( Sno CHAR(9),
  Sname CHAR(8) NOT NULL,
  Ssex CHAR(2),
  Sage SMALLINT,
  Sdept CHAR(20),
  PRIMARY KEY (Sno),
  CHECK (Ssex='女' OR Sname NOT
  LIKE 'Ms.%')
  /*定义了元组中Sname和 Ssex两个属性值之间的约束条件*/
);
```

- 性别是女性的元组都能通过该项检查, 因为Ssex='女' 成立;
- 当性别是男性时, 要通过检查则名字一定不能以Ms.打头

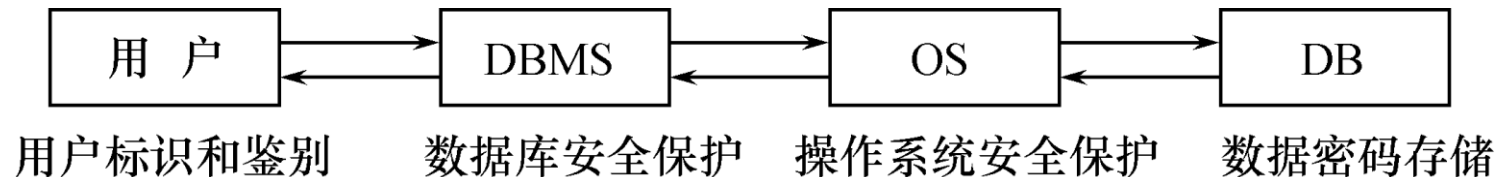
访问控制

❖ 问题的提出

- 数据库的一大特点是数据可以共享
- 数据共享必然带来数据库的安全性问题
- 数据库系统中的数据共享不能是无条件的共享

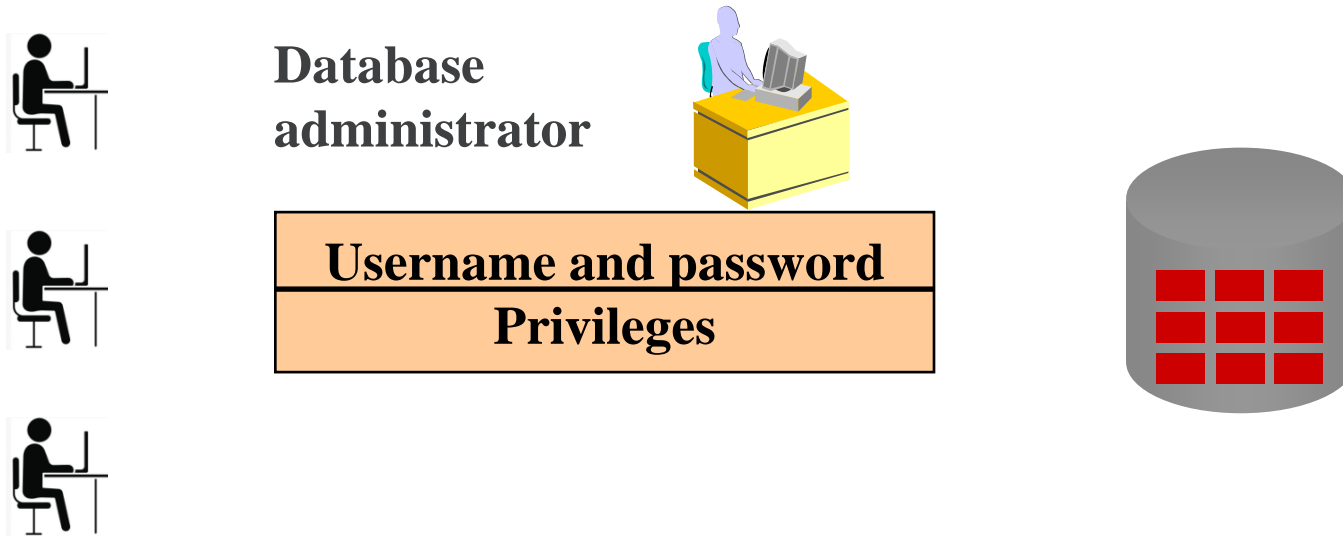
例： 军事秘密、国家机密、新产品实验数据、
市场需求分析、市场营销策略、销售计划、
客户档案、医疗档案、银行储蓄数据

 数据库安全性



高级SQL

□ 访问控制(Access control)



□ Access privileges: Gaining access to the database

- **System privileges**: Gaining access to the database
- **Object privileges**: Manipulating the content of the database objects

高级SQL

□ Grant and revoke system privileges

GRANT {system_privilege | role}
[, {system_privilege | role}]...

TO {user | role | | PUBLIC}

[, {user | role | | PUBLIC}]

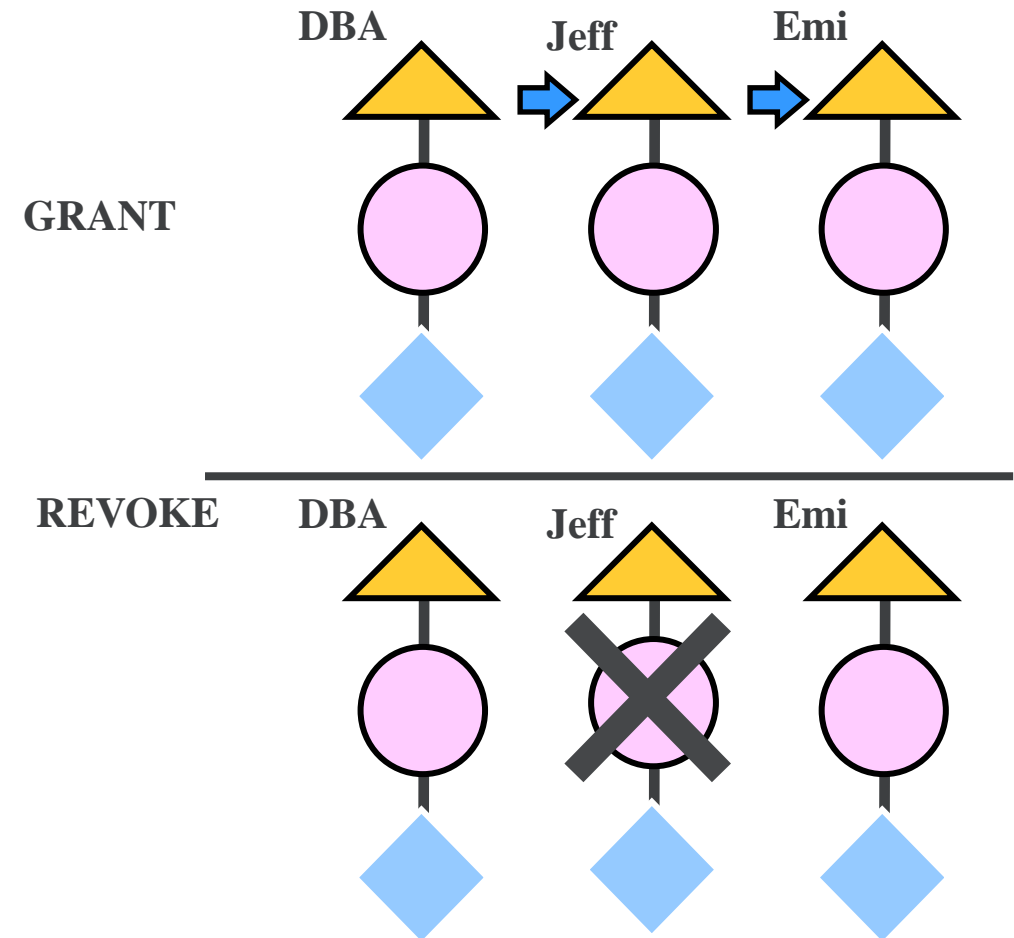
[WITH ADMIN OPTION];

REVOKE {system_privilege | role}

[, {system_privilege | role}]...

FROM {user | role | | PUBLIC}

[, {user | role | | PUBLIC}]...;





高级SQL

□ Grant and revoke system privileges

CREATE USER语句格式

CREATE USER <username>

[WITH][DBA | RESOURCE | CONNECT];

❖ CREATE USER不是SQL标准，各个系统的实现相差甚远

拥有的权限	可否执行的操作			
	CREATE USER	CREATE SCHEMA	CREATE TABLE	登录数据库，执行数据查询和操纵
DBA	可以	可以	可以	可以
RESOURCE	不可以	不可以	不可以	不可以
CONNECT	不可以	不可以	不可以	可以，但必须拥有相应权限

高级SQL

□ Grant and revoke system privileges

- ❖ 数据库角色：被命名的一组与数据库操作相关的权限
- ❖ 角色是权限的集合
- ❖ 可为一组具有相同权限的用户创建一个角色
- ❖ 简化授权的过程

CREATE ROLE 语句格式

```
CREATE ROLE <rolename>
```

给角色授权：

```
GRANT <privileges> [, <privileges>]...  
ON [schema.] object  
TO <ROLE >[,<ROLE>]...
```

将一个角色授予其他角色或用户：

```
GRANT <ROLE1> [, <ROLE2>]...  
TO <ROLE3 >[,<USER1>]...  
[WITH ADMIN OPTION]
```

高级SQL

□ Grant and revoke object privileges

```
GRANT {object_privilege[(column_list)]  
      [, object_privilege[(column_list)] ...  
      | ALL [PRIVILEGES]}
```

```
ON [schema.]object
```

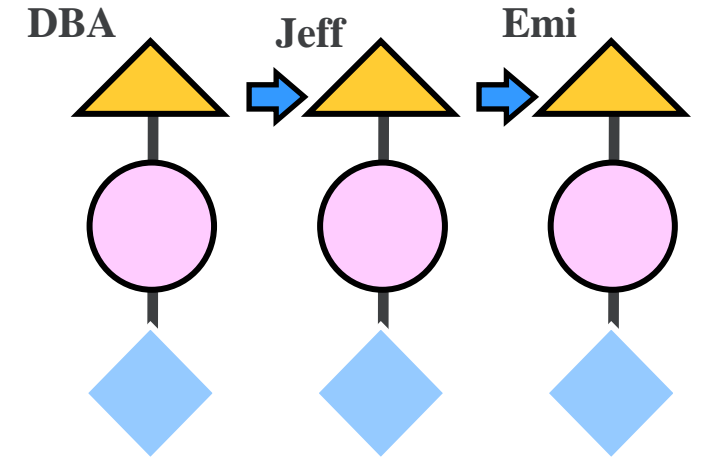
```
TO {user | role | PUBLIC}[, {user | role | PUBLIC}]  
[WITH GRANT OPTION];
```

```
REVOKE {object_privilege  
       [, object_privilege]... | ALL [PRIVILEGES]}
```

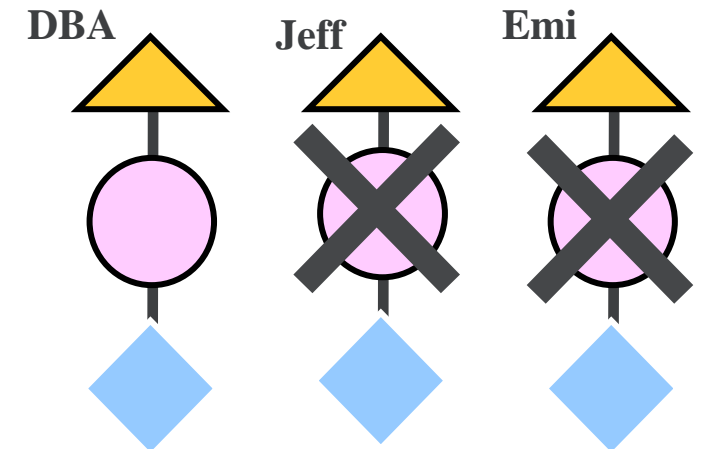
```
ON [schema.] object
```

```
FROM {user | role | | PUBLIC}  
     [, {user | role | | PUBLIC}]...  
[CASCADE CONSTRAINTS];
```

GRANT



REVOKE



高级SQL

□ Grant and revoke object privileges

```
GRANT ALL PRIVILIGES  
ON TABLE Student, Course  
TO U2, U3;
```

把对Student表和Course表的全部权限授予用户U2和U3

```
GRANT UPDATE(Sno), SELECT  
ON TABLE Student  
TO U4;
```

把查询Student表和修改学生学号的权限授给用户U4

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

把对表SC的查询权限授予所有用户

```
REVOKE UPDATE(Sno)  
ON TABLE Student  
FROM U4;
```

把用户U4修改学生学号的权限收回

高级SQL

□ Grant and revoke object privileges

将U4拥有的表SC的INSERT权限，传播给U5，同时允许U5继续传播该权限

```
GRANT INSERT  
ON TABLE SC  
TO U5
```

```
WITH GRANT OPTION;
```

收回所有用户对表SC的查询权限

```
REVOKE SELECT  
ON TABLE SC  
FROM PUBLIC;
```

将U5拥有的表SC的INSERT权限，传播给U6，同时不允许U6继续传播该权限

```
GRANT INSERT  
ON TABLE SC  
TO U6;
```

把用户U5对SC表的INSERT权限收回

```
REVOKE INSERT  
ON TABLE SC  
FROM U5 CASCADE ;
```

存储过程和触发器

存储过程和触发器

□ 存储过程(Procedure)

- 存储过程 (Stored Procedure) 是一组为了完成特定功能的SQL语句集，它**存储在数据库**中，一次编译后永久有效，用户可以通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行或者调用它。

□ 存储过程的优点

- (1) 运行效率高
- (2) 降低了客户机和服务器之间的通信量

由**名字、参数**向服务器调用，最终**结果返回**客户端。

- (3) 方便实施企业规则

存储过程和触发器

□ 存储过程(Procedure)

(1) 创建存储过程

CREATE OR REPLACE PROCEDURE 过程名([参数1,参数2,...])

AS <过程化SQL块>;

- 过程名：数据库服务器合法的对象标识
- 参数列表：用名字来标识调用时给出的参数值，必须指定值的数据类型。参数也可以定义输入参数、输出参数或输入/输出参数，默认为输入参数
- 过程体：是一个<过程化SQL块>，包括声明部分和可执行语句部分

(2) 执行存储过程

CALL/PERFORM PROCEDURE 过程名([参数1,参数2,...]);

- 使用**CALL**或者**PERFORM**等方式激活存储过程的执行

存储过程和触发器

□ 存储过程(Procedure)

(1) 创建存储过程

CREATE OR REPLACE PROCEDURE 过程名([参数1,参数2,...])

AS <过程化SQL块>;

- 过程名：数据库服务器合法的对象标识
- 参数列表：用名字来标识调用时给出的参数值，必须指定值的数据类型。参数也可以定义输入参数、输出参数或输入/输出参数，默认为输入参数
- 过程体：是一个<过程化SQL块>，包括声明部分和可执行语句部分

(3) 修改存储过程

ALTER PROCEDURE 过程名1 **RENAME TO** 过程名2;

(4) 删除存储过程

DROP PROCEDURE 过程名();

存储过程和触发器

□ 函数(Function)

- 与存储过程 (Stored Procedure) 相同，都是持久性存储模块。
- 函数必须返回指定的类型

```
CREATE OR REPLACE FUNCTION  
weighted_AVG(IN sNoIn student.sNo%TYPE)  
RETURNS NUMERIC  
AS $weighted_AVG$  
...  
END;  
$weighted_AVG$  
LANGUAGE plpgsql;
```

```
select * from weighted_AVG('170101');
```

Function weighted_AVG(sNo)
returns numeric

.....

... ..

存储过程和触发器

□ 触发器(Trigger)

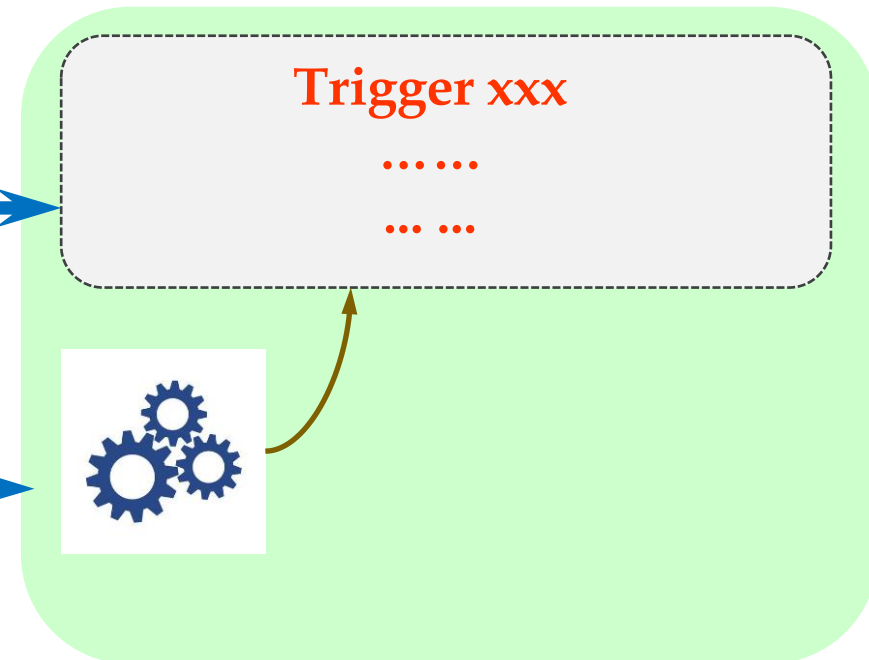
- Procedure that starts automatically if specified changes occur to the DBMS.

□ 触发器三个主要构成部分

- Event (activates the trigger)
- Condition (tests whether the triggers should run)
- Action (what happens if the trigger runs)

```
CREATE TRIGGER xxx  
  AFTER INSERT OR UPDATE OR DELETE ON ttt  
  FOR EACH ROW EXECUTE PROCEDURE ppp();
```

```
INSERT INTO tt VALUES('xxx', 'xxx', xx, default);
```



存储过程和触发器

❖ CREATE TRIGGER语法格式

```
CREATE TRIGGER <触发器名>
{BEFORE | AFTER} <触发事件> ON <表名>
REFERENCING NEW|OLD ROW AS<变量>
FOR EACH {ROW | STATEMENT}
[WHEN <触发条件>]<触发动作体>
```

- INSERT、DELETE或UPDATE
- 也可以是这几个事件的组合
- 还可以UPDATE OF<触发列, ...>, 即进一步指明修改哪些列时激活触发器

- 触发器又叫做事件-条件-动作（**event-condition-action**）规则。
- 当特定的系统事件发生时，对规则的条件进行检查，如果条件成立则执行规则中的动作，否则不执行该动作。
- 规则中的动作体可以很复杂，通常是一段**SQL存储过程**。

存储过程和触发器

□ 案例

sc

sNo	cNo	score
001	01	56
001	02	..
002	01	..
003	02	58
...

Student

sNo	sName	...
001	赵明	...
002
003	孙悦	...
...

Course

cNo	cName	credit	...
01	离散数学
02	数据结构
...
...

Failure_record

sNo	sName	cNo	cName	score
001	赵明	01	离散数学	56
003	孙悦	02	数据结构	58
...

Trigger xxx

.....

存储过程和触发器

□ [案例]

定义一个**BEFORE**行级触发器，为教师表**Teacher**定义完整性规则“教授的工资不得低于**8000**元，如果低于**8000**元，自动改为**8000**元”。

```
CREATE TRIGGER Insert_Or_Update_Sal
BEFORE INSERT OR UPDATE ON Teacher
            /*触发事件是插入或更新操作*/

FOR EACH ROW      /*行级触发器*/

BEGIN            /*定义触发动作体，是PL/SQL过程块*/
    IF (new.Job='教授') AND (new.Sal < 8000)
    THEN new.Sal :=8000;
    END IF;
END;
```

存储过程和触发器

□ 触发器(Trigger)的激活与删除

- ❖ 触发器的执行，是由**触发事件激活**的，并由数据库服务器自动执行
- ❖ 一个数据表上可能定义了**多个触发器**，遵循如下的执行顺序：
 - (1) 执行该表上的**BEFORE**触发器；
 - (2) 激活触发器的**SQL**语句；
 - (3) 执行该表上的**AFTER**触发器。

❖ 删除触发器的**SQL**语法：

DROP TRIGGER <触发器名> ON <表名>;

- ❖ 触发器必须是一个已经创建的触发器，并且只能由具有相应权限的用户删除。

SQL应用

数据库应用存在的问题思考

```
C:\Windows\system32\cmd.exe - C:\PostgreSQL\9.0\bin\psql -d mydb
-U, --password          force password prompt (should happen automatically)

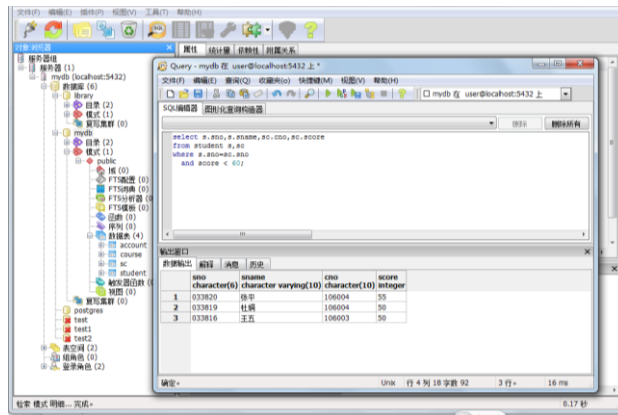
For more information, type "?" (for internal commands) or "help" (for SQL
commands) from within psql, or consult the psql section in the PostgreSQL
documentation.

Report bugs to <pgsql-bugs@postgresql.org>.

C:\PostgreSQL\9.0\bin\psql -d mydb
psql (9.0.1)
Type "help" for help.

mydb=# select s.sno,s.sname,sc.cno,sc.score
mydb=# from student s,sc
mydb=# where s.sno=sc.sno
mydb=# and score < 60;
 sno | sname | cno | score
-----+-----+----+-----
 833820 | 张平 | 106004 | 55
 833819 | 杜娟 | 106004 | 50
 833816 | 王五 | 106003 | 50
(3 rows)

mydb=#
```



SQL的特点?

SQL

Lookup for all students which have failure records for any course.

For each student which score < 60
send a message to him about makeup
print name list for all students need makeup

Select sNo
From sc
Where score < 60



如何表达用户业务?

数据库应用存在的问题

□ SQL is a direct query language and it is very suit for data depositing and retrieving; as such, it has limitations.

- Complex computational processing of the data.
- Specialized user interfaces.
- Access to more than one database at a time.

□ 数据库应用中如何弥补SQL的不足，进而解决业务表达问题？

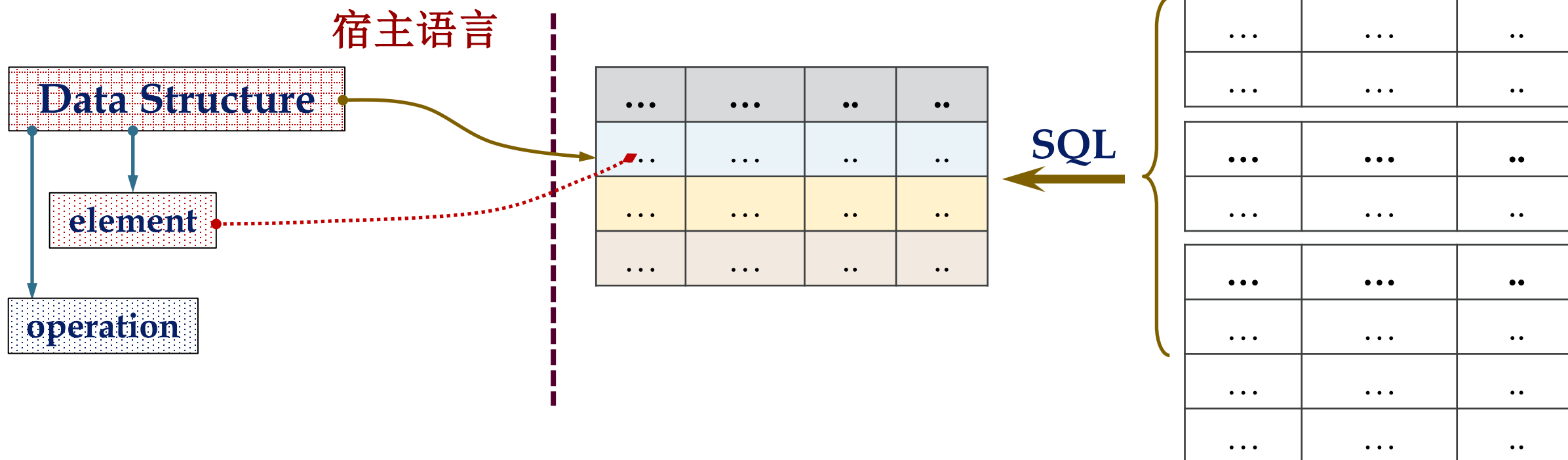
- **嵌入式**-将SQL直接嵌入到通用程序设计语言，各司其职。
- **应用编程接口(API)**-通过库函数(接口)调用，实现与数据库交互。
- **设计新语言(混合)**-支持SQL和通用程序设计语言特性。

} 程序式

程序式SQL需要思考的问题-1

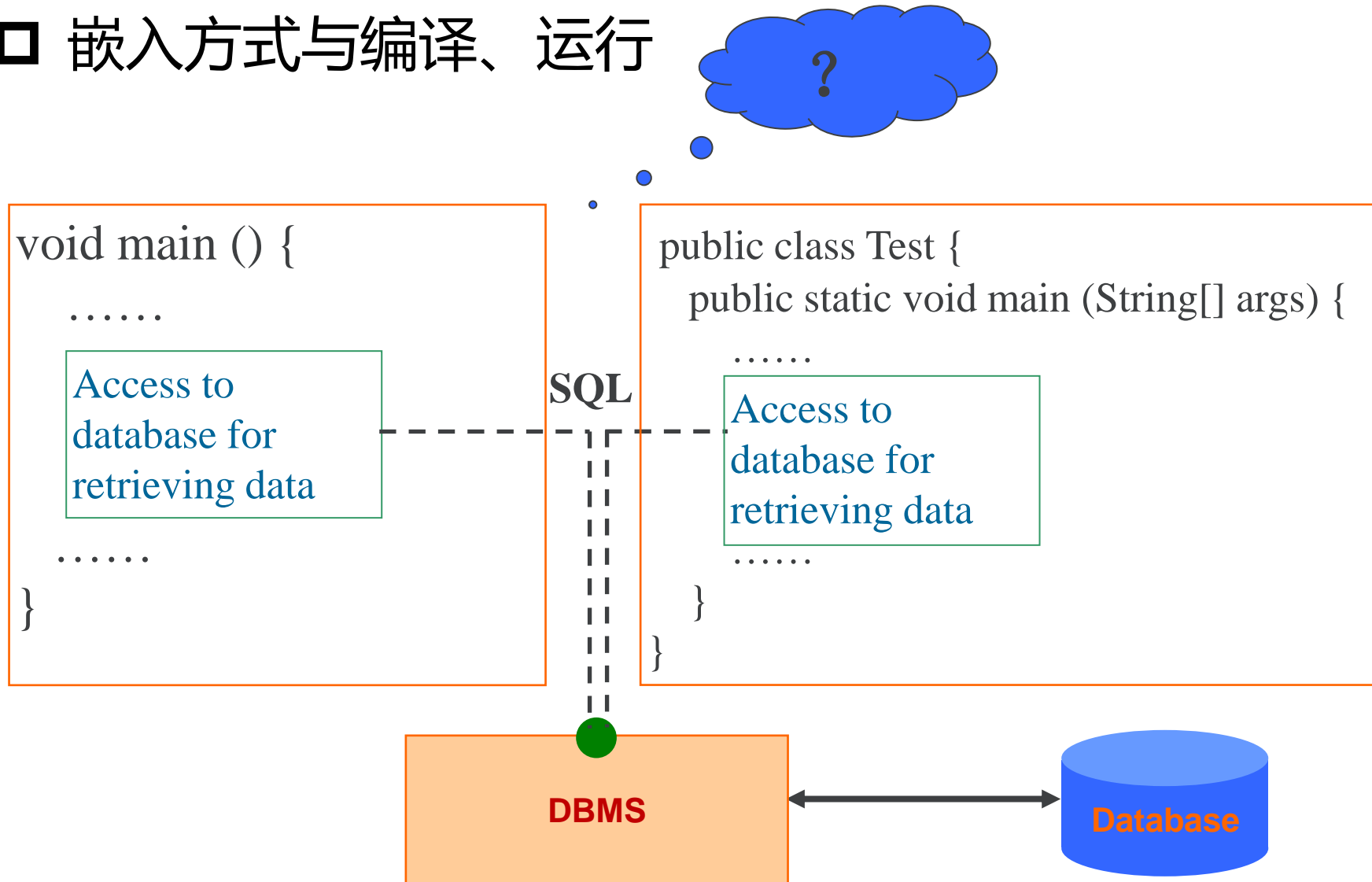
□ 阻抗失配(impedance mismatch): 数据库模型与程序设计语言模型之间存在的差异而导致的不匹配问题。

- 数据类型问题
- 数据查询返回结果与程序设计语言数据类型匹配问题
- 数据查询结果遍历问题



程序式SQL需要思考的问题-2

□ 嵌入方式与编译、运行

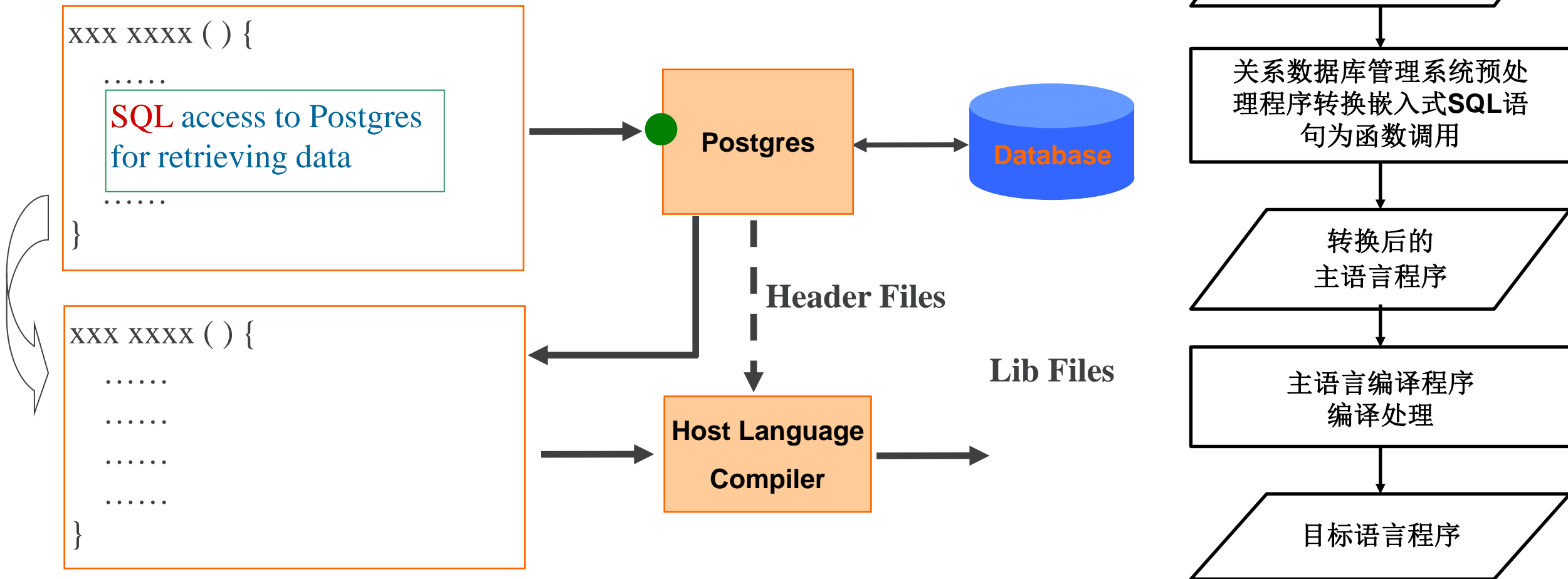


2001 年，IBM 高级副总裁 Paul Horn 在哈佛大学的报告中提出“自主计算（[Autonomic Computing^{\[1\]}](#)）”概念，以应对计算复杂性危机的问题。“自主计算”也译作“自治计算”，其基本思想是参照生物领域自主神经系统的自我调节机制，以现有的理论和技术为基础构建计算系统，使得计算系统具有自我感知与管理的能力。他在报告中指出“*It's time to design and build computing systems capable of running themselves, adjusting to varying circumstances, and preparing their resources to handle most effectively the workloads we put upon them. These autonomic systems must anticipate needs and allow users to concentrate on what they want to accomplish rather than figuring how to rig the computing systems to get them there.*”

.....

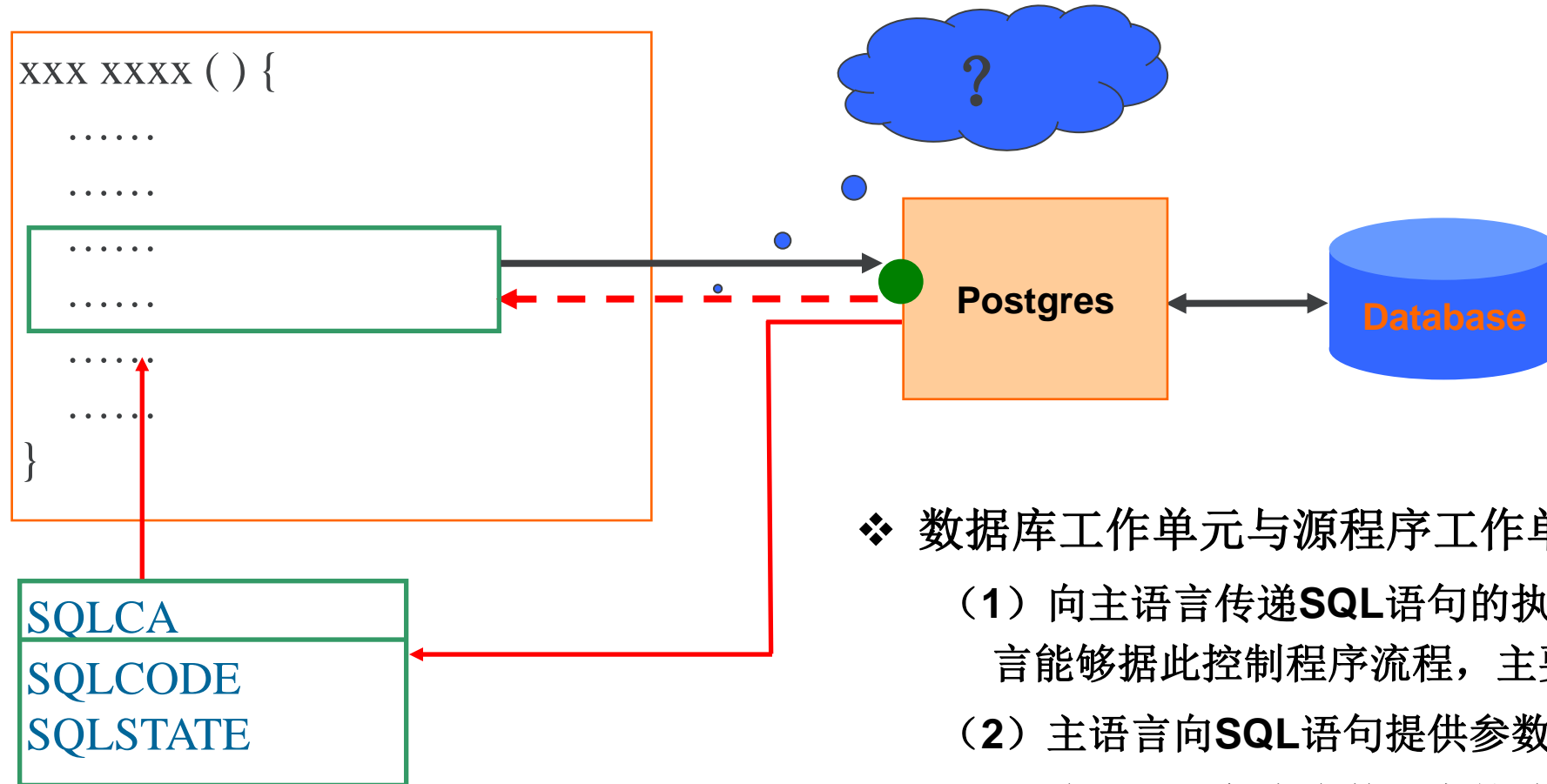
嵌入式SQL(Embedded SQL)

- ❑ SQL statements are embedded directly into the program source code and mixed with the host language statements.
- ❑ Need to be precompiled.



嵌入式SQL(Embedded SQL)

□ 返回结果的处理(通信区Communication Area)



- ❖ 数据库工作单元与源程序工作单元之间的通信
 - (1) 向主语言传递**SQL**语句的执行状态信息，使主语言能够据此控制程序流程，主要用**SQL**通信区实现
 - (2) 主语言向**SQL**语句提供参数，主要用主变量实现
 - (3) 将**SQL**语句查询数据库的结果交主语言处理，主要用主变量和游标实现

Example: embedding SQL in C program

嵌入式SQL(Embedded SQL)

□ 静态嵌入SQL

- In static embedded SQL statement, the pattern of database access is **fixed** and can be 'hard-coded' into the program.
- Static SQL does not allow host variables to be used in place of table names or column names.
- In static Embedded SQL, the follow elements must be fixed:
 - Reserved words (SELECT,UPDATE,DELETE...)
 - The number of host variables
 - The data type of each host variable
 - The database object will be accessed in SQL(table,column,view,index,...)

嵌入式SQL(Embedded SQL)

□ 动态嵌入SQL

- In many situations where the pattern of database access is not fixed and **is known only at runtime**. This requires more flexibility than static SQL.
- The basic idea of dynamic SQL is to place the complete SQL statement to be executed in a host variable. The **host variable** is then passed to the DBMS to be executed.

嵌入式SQL(Embedded SQL)

□ 静态嵌入 VS 动态嵌入SQL

```
EXEC SQL BEGIN DECLARE SECTION;  
    float increment;  
EXEC SQL END DECLARE SECTION;  
EXEC SQL UPDATE Staff SET  
    salary=salary+:increment WHERE  
    staffNo='SL21';
```

```
EXEC SQL BEGIN DECLARE SECTION;  
    char buffer[100];  
EXEC SQL END DECLARE SECTION;  
sprintf(buffer,"UPDATE Staff SET salary=salary+%/f WHERE  
    staffNo='SL21'",increment);
```

```
EXEC SQL EXECUTE IMMEDIATE :buffer;
```

❖ 静态嵌入式SQL

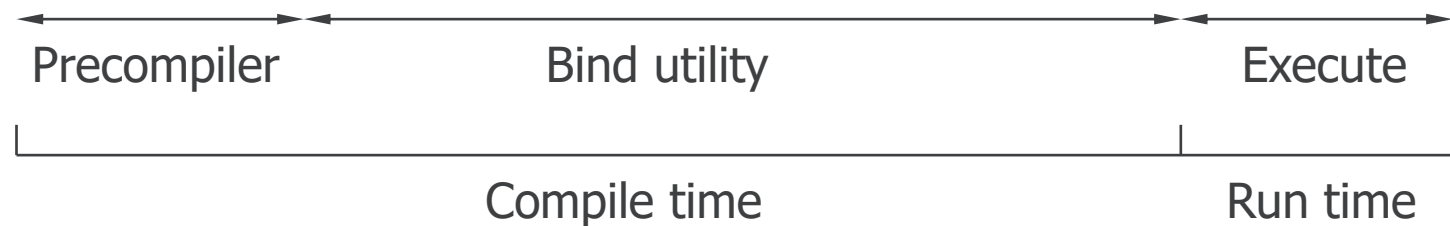
- 静态嵌入式SQL语句能够满足一般要求
- 无法满足要到执行时才能够确定要提交的SQL语句、查询的条件

❖ 动态嵌入式SQL

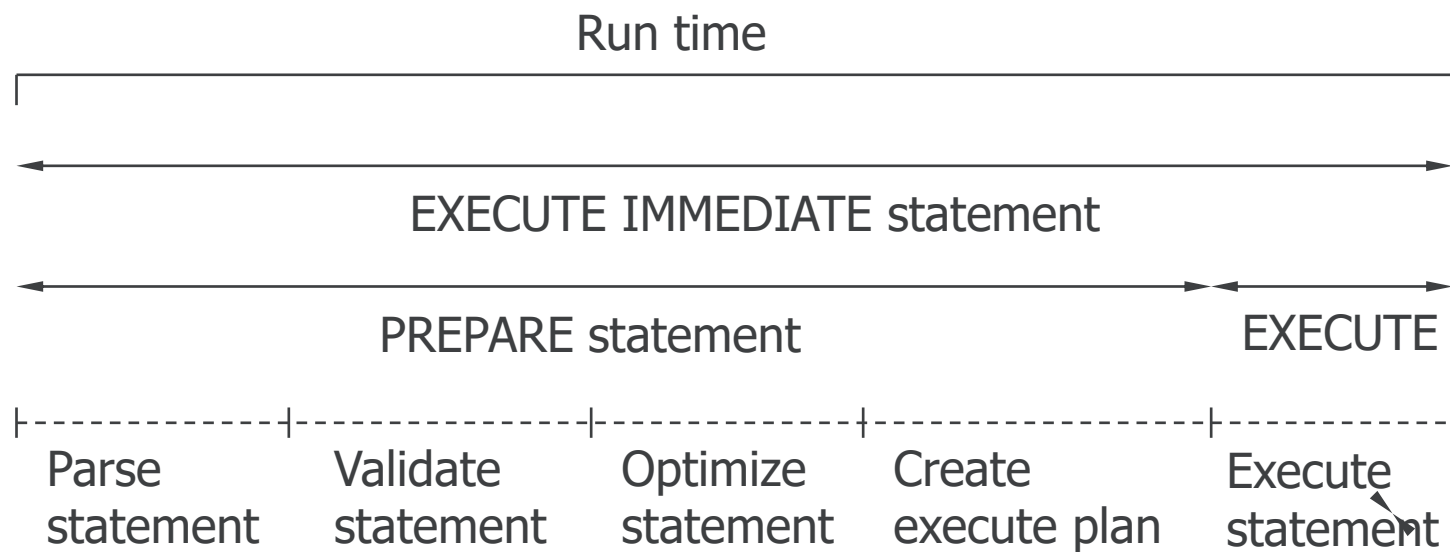
- 允许在程序运行过程中临时“组装”SQL语句
- 支持动态组装SQL语句和动态参数两种形式

嵌入式SQL(Embedded SQL)

□ 静态嵌入 VS 动态嵌入SQL



(a) static SQL



(b) Dynamic SQL

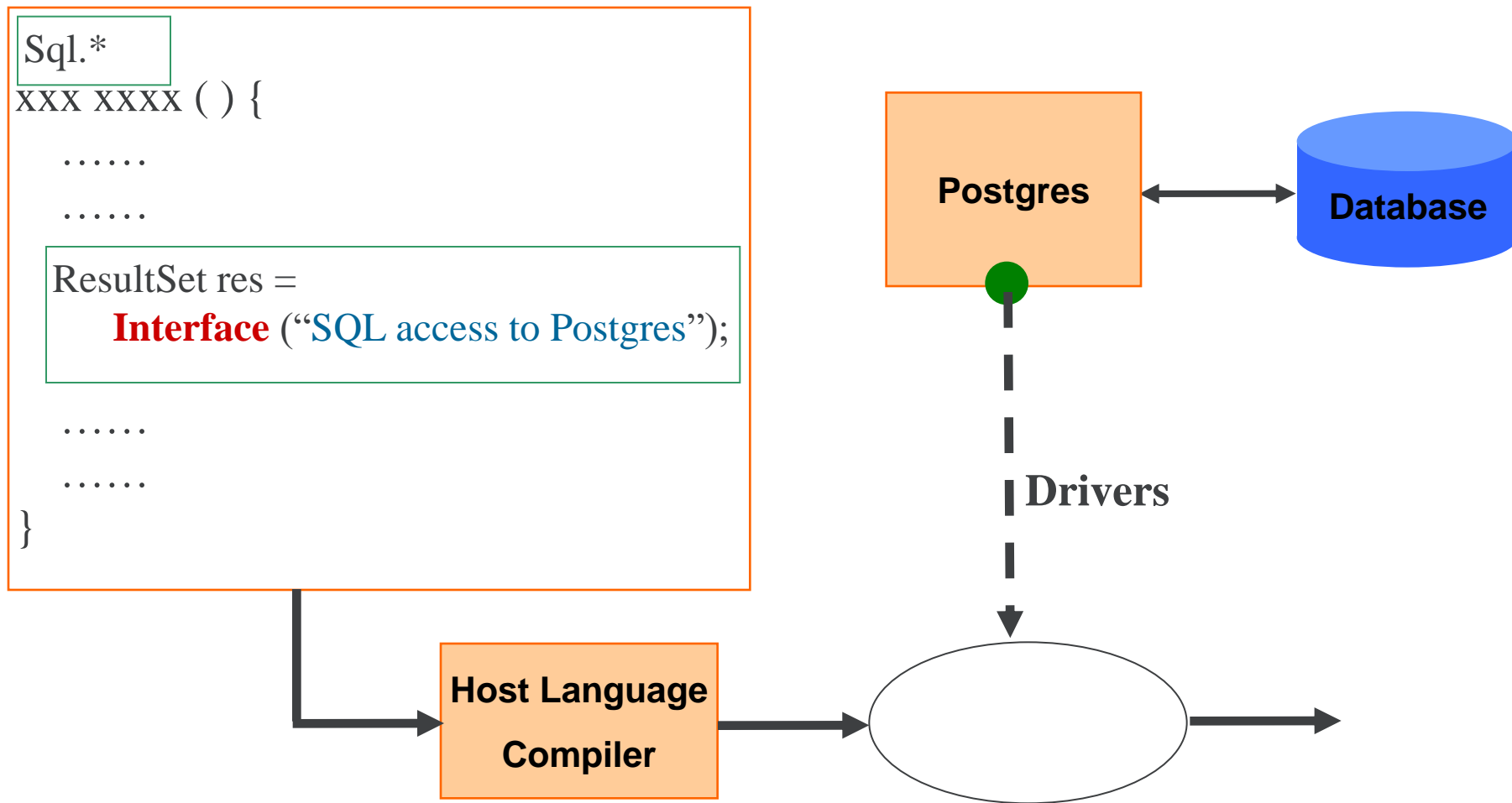
立即执行语句：运行时编译并执行

延迟执行语句：prepare-execute-using语句

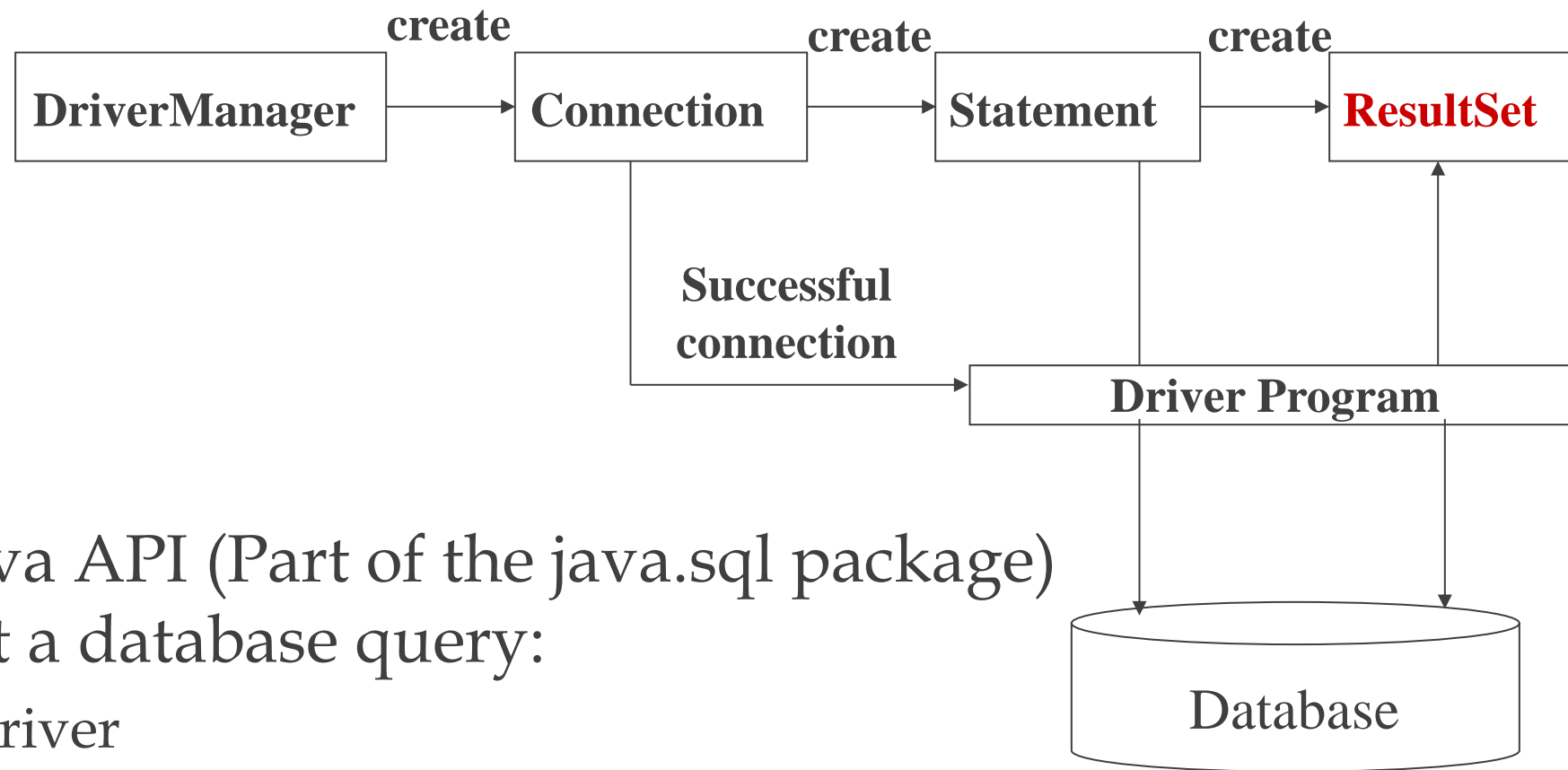
应用编程接口(API)

- ❑ Provide the programmer with a standard set of functions that can be invoked from the software.
- ❑ Need not to be precompiled.
- ❑ Open Database Connectivity (ODBC) standard.

应用编程接口(API)



应用编程接口(API)



- ❑ Sun's JDBC: Java API (Part of the java.sql package)
- ❑ Steps to submit a database query:
 - Load the JDBC driver
 - Connect to the data source
 - Execute SQL statements

Example: 通过JDBC访问数据库

应用编程接口(API)

❖ 创建新Java Project

❖ 导入jar包

https://blog.csdn.net/qq_3537732

[3/article/details/112979532](https://blog.csdn.net/qq_3537732/article/details/112979532)

❖ 新建package

❖ Java版本、jar包等的适配

Test_LHQ - Backup_JDBC/src/com/example/backupjdbc/DBConnectionTest.java - Eclipse IDE

File Edit Source Refactor Navigate Search Run Window Help

Project Explorer

- Backup_JDBC
 - src
 - com.example.backupjdbc
 - App.java
 - DBConnectionTest.java
 - DBConnectionTest1.java
 - DBWrapper.java
 - JRE System Library [jre]
 - Referenced Libraries
 - postgresql-42.0.0.jre6.jar - E:\Des
 - JDBCDemo
 - JDBCDemo2
 - JDBCjava
 - ProjectJ

App.java DBConnectionTest.java

```
1 package com.example.backupjdbc;
2 import java.sql.*;
3
4 public class DBConnectionTest {
5     public static void main(String[] args) {
6         // if (args.length < 3) {
7         //     System.out.println("Syntax:DBConnection [url][username][password]");
8         //     // url = jdbc:postgresql:database or jdbc:postgresql://host:port/database
9         //     System.exit(-1);
10        // }
11
12        String url = "jdbc:postgresql://127.0.0.1:5433/LHQ_Exercise";
13        String username = "postgres";
14        String password = "8827";
15
16        String sql = "select * from sc";
17        String s;
18
19        try {
20            System.out.println("Step 01: Registering JDBC Driver");
21
22            /* There are three ways to registe driver. */
23
24            // The first way
25
26            try {
27                Class.forName("org.postgresql.Driver");
28            } catch (java.lang.ClassNotFoundException e) {
29                System.out.println(e.getMessage());
30            }
31
32            // The second way
33            // DriverManager.registerDriver(new org.postgresql.Driver());
34
35            // The third way
36            // System.setProperty("jdbc.drivers", "org.postgresql.Driver");
37
38        }
39    }
40
41}
```

❖ 过程化SQL

- SQL的扩展
- 增加了过程化语句功能
- 基本结构是块
 - 块之间可以互相嵌套
 - 每个块完成一个逻辑操作

❖ 过程化SQL块的基本结构

1. 定义部分

DECLARE 变量、常量、游标、异常等

- 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

2. 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;

过程语言/SQL

```
DECLARE
    weightedScore NUMERIC;
BEGIN
    SELECT CAST(SUM(CAST(score*credit AS
        NUMERIC)))/SUM(credit) AS NUMERIC) INTO
        weightedScore
    FROM sc, course c
    where sc.cNo=c.cNo
        and score is not null and credit is not null
        and sc.sNo=sNoIn
    group by sc.sNo;
    RETURN weightedScore;
END;
```

Example: 参照第10讲－存储过程和触发器

❖ 过程化SQL功能

1. 条件控制语句

2. 循环控制语句

3. 错误处理

1. 条件控制语句

IF-THEN, IF-THEN-ELSE和嵌套的**IF**语句

(1) **IF condition THEN**

Sequence_of_statements;

END IF;

(2) **IF condition THEN**

Sequence_of_statements1;

ELSE

Sequence_of_statements2;

END IF;

(3) 在**THEN**和**ELSE**子句中还可以再包含**IF**语句，即**IF**语句可以嵌套

❖ 过程化SQL功能

1. 条件控制语句

2. 循环控制语句

3. 错误处理

(3) FOR-LOOP

FOR count IN [REVERSE] bound1 ...
bound2 LOOP

Sequence_of_statements;

END LOOP;

2. 循环控制语句

LOOP, WHILE-LOOP和FOR-LOOP

(1) 简单的循环语句LOOP

LOOP

Sequence_of_statements;

END LOOP;

多数数据库服务器的过程化SQL都提供EXIT、BREAK或LEAVE等循环结束语句，保证LOOP语句块能够结束

(2) WHILE-LOOP

WHILE condition LOOP

Sequence_of_statements;

END LOOP;

- 每次执行循环体语句之前，首先对条件进行求值
- 如果条件为真，则执行循环体内的语句序列
- 如果条件为假，则跳过循环并把控制传递给下一个语句

❖ 过程化SQL功能

1. 条件控制语句
2. 循环控制语句
3. 错误处理

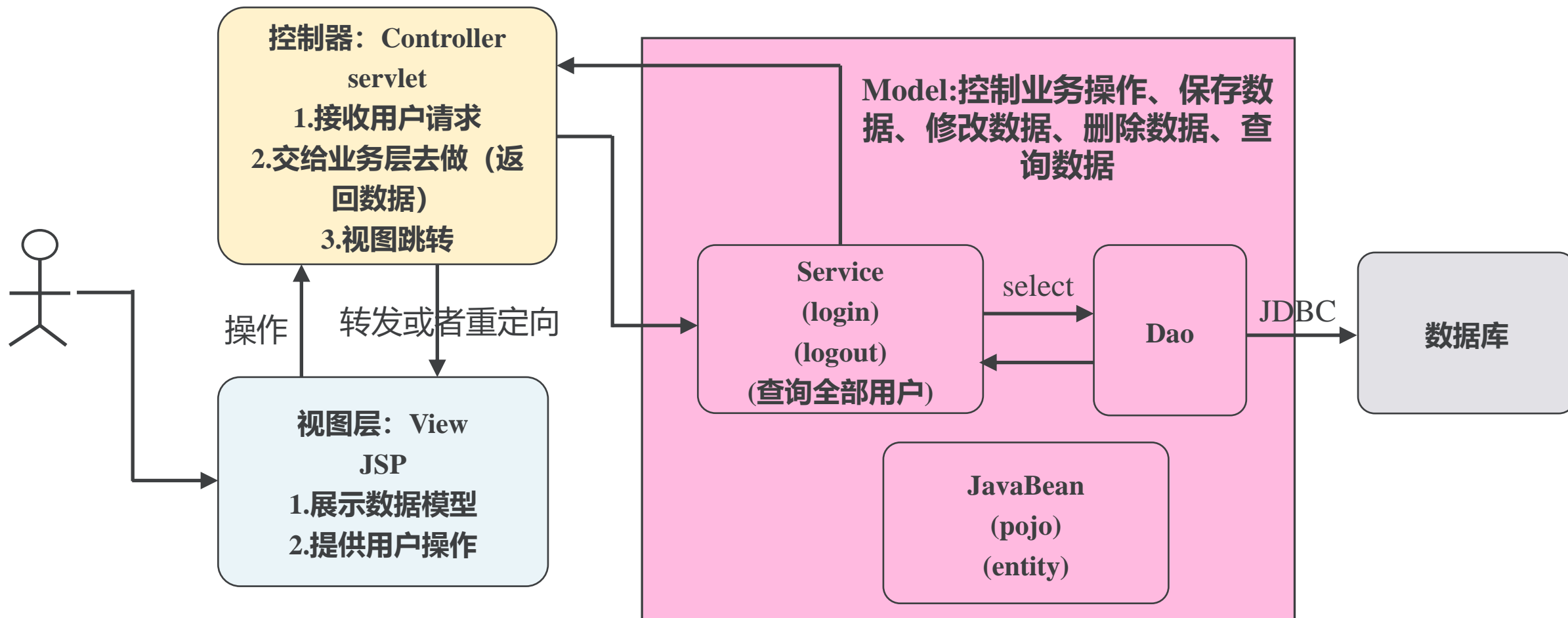
3. 错误处理

- 如果过程化SQL在执行时出现异常，则应该让程序在产生异常的语句处停下来，根据异常的类型去执行异常处理语句
- SQL标准对数据库服务器提供什么样的异常处理做出了建议，要求过程化SQL管理器提供完善的异常处理机制

模型-视图-控制器体系结构(MVC)

- ❑ The Model View Controller (MVC) architecture describes a way to organize and separate the tasks of an application into three distinct parts: Model, View, and Controller.
- ❑ The **View** manages the output of a user interface.
- ❑ The **Controller** processes the user's input.
- ❑ The **Model** represents the data and logic of the subset of the external world used in the program

模型-视图-控制器体系结构(MVC)



关于本讲内容



祝各位学习愉快!

感谢观看！

讲解人：李鸿岐