

第3章 关系数据库语言SQL

2012.03



目录 Contents

- **3.1 数据库的用户接口**
- **3.2 SQL语言概况**
- **3.3 SQL数据定义语言**
- **3.4 SQL数据查询语言**
- **3.5 SQL数据操纵语言**
- **3.6 SQL中的视图**
- **3.7 嵌入式SQL**



3.1 数据库的用户接口

■ 一、数据库操作与数据库语言

■ 数据库操作包括：

■ 数据定义

- 在数据库中创建、撤销、修改数据模式。

■ 数据查询

- 在数据库中查询检索所需的数据。

■ 数据操纵—也称更新数据

- 在数据库中增加、删除、修改数据。

■ 数据控制

- 控制用户对数据库中数据的访问权限。



3.1 数据库的用户接口

一、数据库操作与数据库语言

- **数据库语言**：DBMS提供的语言，以支持用户进行数据库操作。包括：
 - **数据定义语言(Data Definition Language, DDL)**
 - 定义、撤销和修改数据模式—基表/视图，索引/簇集
 - **查询语言(Query Language, QL)**
 - 查询数据
 - **数据操纵语言(Data Manipulation Language, DML)**
 - 数据的插入/删除/修改，简单的数值计算与统计功能
 - **数据控制语言(Data Control Language, DCL)**
 - 数据的完整性、安全性、并发控制、故障恢复等



3.1 数据库的用户接口

■ 一、数据库操作与数据库语言

■ 数据库语言的特点

- 面向记录(record-oriented)与面向集合(set-oriented)
- 嵌入式(embedded)和交互式(interactive)
- 过程性(procedural)与说明性(declarative)



3.1 数据库的用户接口

■ 数据库语言的特点1

■ 面向记录(record-oriented)与面向集合(set-oriented)

- 早期的数据库操作(e.g. 层次、网状数据库)是使用物理指针的导航式访问(navigational access)，一次操作一个记录(one record one time)，操作过程繁琐，效率低下。
- 相应的数据库语言称面向记录的语言(record-oriented language)。
- 关系数据库使用联想式访问(associative access)，即按数据的内容访问数据（按属性值选取数据），一次操作得到一个记录的集合。
- 相应的数据库语言称面向集合的语言(set-oriented language)。



3.1 数据库的用户接口

■ 数据库语言的特点2

- 嵌入式(embedded)和交互式(interactive)
 - 数据库语言往往不是计算完备(computing complete)的语言。
 - 要实现数据“管理”与“计算”的集成，可以将数据库语言嵌入(embedding)到高级语言(e.g. PASCAL, C, FORTRAN)，称宿主语言(host language)中。
 - 因此，数据库语言就有两种：嵌入式和交互式。



3.1 数据库的用户接口

■ 数据库语言的特点3

■ 过程性(procedural)与说明性(declarative)

- 早期的数据库语言(e.g. 层次、网状数据库)是过程性的，使用这样的语言进行数据库操作，程序员的负担很重；而且编写的应用程序与数据的独立性(independence)很差、可维护性很差。
- 关系数据库提供了非过程性(non-procedural)语言即说明性语言(只要告诉DBMS “What to do?”)。但某些过程性机制(e.g. 存储过程(stored procedure)、计算流程控制, etc.)对方便用户、提高数据库性能是很有用的。
- 因此，关系数据库语言(e.g. SQL)后来又进行了过程化扩充，作为可选项提供给用户。



3.1 数据库的用户接口

■ 二、用户接口与前端开发工具

■ 用户接口(user interface)

- 是DBMS提供给用户操作数据库的界面。
- 用户接口将用户对数据库的操作请求以数据库语言的形式提交给系统，并接受系统的处理结果、将结果返回给用户。
- 用户接口提供了两种操作数据库的方式：交互方式和批处理方式（即编写应用程序）。
- 用户接口风格可有：文本的和GUI。

■ 前端开发工具(front-end development tools)

- DBMS厂商或第三方提供的数据库应用集成化开发工具 (e.g. ORACLE Developer/2000, PowerBuilder)。



目录 Contents

- 3.1 数据库的用户接口
- **3.2 SQL语言概况**
- 3.3 SQL数据定义语言
- 3.4 SQL数据查询语言
- 3.5 SQL数据操纵语言
- 3.6 SQL中的视图
- 3.7 嵌入式SQL



3.2 SQL语言概况

一、SQL的历史

- SEQUEL (Structured English Query Language), (IBM System R, Boyce and Chamberlin proposed, 1974)
- SQL-86 (SQL1, Standard Query Language), (ANSI, 1986)
- SQL-89 (ANSI/ISO, 1989)
 - SQL-92 (SQL2), (ANSI/ISO, 1992)
- SQL-93 (ANSI/ISO, 1993)
 - SQL-99 (SQL3), (ANSI/ISO, 1999)
- SQL-2008
- S — 标准化：按照英语的结构要求设计，可以嵌套
- Q — 查询：以查询为主，包括增删改等功能
- L — 语言：第四代非过程式描述性语言



3.2 SQL语言概况

■ 二、SQL的特点

- 非过程化的说明性语言
- 嵌入式和交互式相同/相似语法的语言
- 一体化的统一语言
- 关系数据库的公共语言
- English-like、简单易学的语言



3.2 SQL语言概况

■ 三、SQL标准与实现

- ISO的SQL标准SQL-89、SQL2以及SQL:1999(即SQL3)是**国际标准**，而各个RDBMS厂商提供的是SQL**实现**。
- SQL标准文本可参见：
<http://speckle.ncsl.nist.gov/~ftp/isowg3/dbl/BASEdocs>
- SQL实现可参见：各厂商提供的技术文档资料。



目录 Contents

- 3.1 数据库的用户接口
- 3.2 SQL语言概况
- **3.3 SQL数据定义语言**
- 3.4 SQL数据查询语言
- 3.5 SQL数据操纵语言
- 3.6 SQL中的视图
- 3.7 嵌入式SQL



3.3 SQL数据定义语言

- **SQL数据类型**
- **基表模式的创建**
- **基表模式的修改与撤销**
- **其他模式的建立与撤销**



3.3 SQL数据定义语言

一、SQL基本数据类型

序号	符号	数据类型	备注
1	INT	整数	
2	SMALLINT	短整数	
3	DEC(m, n)	十进制数	m为位数,n为小数点后位数
4	FLOAT	浮点数	
5	CHAR(n)	定长字符串	n表示字符串位数
6	VARCHAR(n)	变长字符串	n表示最大变长数
7	BIT(n)	位串	n为位串长度
8	BIT VARYING(n)	变长位串	n为最大变长数
9	DATE	日期	
10	TIME	时间	
11	TIMESTAMP	时间戳	



3.3 SQL数据定义语言

- SQL数据类型
- 基表模式的创建
- 基表模式的修改与撤销
- 其他模式的建立与撤销



3.3 SQL数据定义语言

■ 二、基表的创建

```
CREATE TABLE <表名>(<列名> <数据类型>[ <列级完整性约束条件> ] [, <列名> <数据类型>[ <列级完整性约束条件>] ] ...[, <表级完整性约束条件> ] );
```

- **<表名>**：所要定义的基本表的名字
- **<列名>**：组成该表的各个属性（列）
- **<列级完整性约束条件>**：涉及相应属性列的完整性约束条件
- **<表级完整性约束条件>**：涉及一个或多个属性列的完整性约束条件



3.3 SQL数据定义语言

- **二、基表的创建 (cont.)**
 - 常用完整性约束
 - 实体完整性约束: PRIMARY KEY
 - 唯一性约束: UNIQUE
 - 非空值约束: NOT NULL
 - 引用完整性约束: FOREIGN KEY
 - PRIMARY KEY与 UNIQUE的区别?



3.3.2 SQL数据定义功能

- **[例1]** 建立一个“部门”表dept，它由部门号deptno、部门名deptname、位置loc等属性组成。其中部门号是主键，部门名取值不能为空，且唯一。

部门: dept (deptno, dname, loc)

```
CREATE TABLE dept
( deptno INT PRIMARY KEY,
  dname VARCHAR(10) NOT NULL UNIQUE,
  loc VARCHAR(10)
    CHECK ( loc IN ('Shanghai','Nanjing', 'Wuhan',
                    'Xian', 'Beijing'))
);
```



3.3.2 SQL数据定义功能

[例2] 建立一个“职员”表emp，它由工号、姓名、工种、主管经理、雇用时间、薪水、佣金、所在部门等属性组成。

职员： emp(empno, ename, job, mgr, hiredate, sal, comm, deptno)

CREATE TABLE emp

```
( empno  INT  PRIMARY KEY,  
  ename  VARCHAR(10) NOT NULL,  
  job    VARCHAR( 9) NOT NULL,  
  mgr    INT  REFERENCES emp(empno),  
  hiredate DATE DEFAULT SYSDATE,  
  sal    DEC(7,2) CHECK (sal>1000.0)  
  comm   DEC(7,2) DEFAULT NULL,  
  deptno INT NOT NULL REFERENCES dept(deptno)  
                                ON DELETE CASCADE
```

);



3.3 SQL数据定义语言

- SQL数据类型
- 基表模式的创建
- 基表模式的修改与撤销
- 其他模式的建立与撤销



3.3 SQL数据定义语言

三、基表模式的修改与撤消

■ 增加列

- **ALTER TABLE** <表名> [**ADD** <新列名> <数据类型> [完整性约束]];
- <表名> : 要修改的基本表
- **ADD子句**: 增加新列和新的完整性约束条件

■ **[例3]** 向emp表增加“性别”列，其数据类型为字符型。

- **ALTER TABLE emp ADD GENDER CHAR(2) NOT NULL;**
- 不论基本表中原来是否已有数据，新增加的列一律为空值。



3.3 SQL数据定义语言

■ 三、基表模式的修改与撤消(cont.)

■ 删除基表

- DROP TABLE <表名>;
- 基表删除，数据、表上的索引都删除
- 系统从数据字典中删去有关该基表及其索引的描述

■ [例4] 删除emp表

- DROP TABLE emp ;

■ 删除属性列-间接删除

- 把表中要保留的列及其内容复制到一个新表中
- 删除原表
- 再将新表重命名(RENAME)为原表名



3.3 SQL数据定义语言

三、基表模式的修改与撤消(cont.)

■ 修改属性列

- **ALTER TABLE** <表名> **MODIFY** <列名> <数据类型>;
- **[例5]** 将emp表中工号改为字符类型，串长度改为8位。
- **ALTER TABLE emp MODIFY empno CHAR(8);**
- **注：**修改原有属性定义有可能会破坏已有数据

■ 补充定义主键

- **ALTER TABLE** <表名> **ADD PRIMARY KEY**(<列名表>);
- 要求定义为主键的列名表**必须满足NOT NULL和唯一性条件**

■ 撤销主键定义

- **ALTER TABLE** <表名> **DROP PRIMARY KEY**;
- 暂时撤销主键定义，在插入新的元组时，可以提高系统的性能



3.3 SQL数据定义语言

三、基表模式的修改与撤消(cont.)

■ 补充定义外键

- ALTER TABLE <表名1>

ADD FOREIGN KEY [<外键名>](<列名表>)

REFERENCES <表名2>

[ON DELETE {RESTRICT | CASCADE | SET NULL}];

■ 撤销外键定义

- ALTER TABLE <表名1> DROP <外键名>;



3.3 SQL数据定义语言

- SQL数据类型
- 基表模式的创建
- 基表模式的修改与撤销
- 其他模式的建立与撤销



3.3 SQL数据定义语言

■ 四、其他模式对象的定义与撤消

■ 别名(alias)

- 用简单的别名代替全名，书写和输入都比较方便
- 由于各个用户对同一数据对象可能有不同的命名，若为不同用户定义不同的别名，则各用户可以保留自己习惯的命名。

■ 定义别名

- **CREATE SYNONYM** <标识符>
FOR {<基表名>|<视图名>};

■ 撤销别名

- **DROP SYNONYM** <标识符>;



3.3 SQL数据定义语言

■ 四、其他模式对象的定义与撤消(cont.)

■ 索引的建立与撤销

- 索引是物理存储路径，不属于逻辑数据模式。
- 建立索引是加快查询速度的有效手段。
- 建立索引
 - DBA或表的创建者根据需要建立
 - 有些DBMS自动建立以下列上的索引
 - PRIMARY KEY
 - UNIQUE
- 维护索引
 - DBMS自动完成
- 使用索引
 - DBMS自动选择是否使用索引，以及使用哪些索引



3.3 SQL数据定义语言

■ 四、其他模式对象的定义与撤消(cont.)

■ 建立索引的语句格式

CREATE [**UNIQUE**] [**CLUSTER**] **INDEX** <索引名>

ON <表名>(<列名>[<次序>][,<列名>[<次序>]]...);

- 用<表名>指定要建索引的基表名字
- 索引可以建立在该表的一列或多列上，各列名之间用逗号分隔
- 用<次序>指定索引值的排列次序，升序(ASC),降序(DESC)。
缺省值：**ASC**
- **UNIQUE**表明此索引的每一个索引值只对应唯一的数据记录
- **CLUSTER**表示要建立的索引是聚簇索引



3.3 SQL数据定义语言

四、其他模式对象的定义与撤消(cont.)

■ 建立索引

■ 唯一值索引(UNIQUE)

- 对于已含重复值的属性列不能建UNIQUE索引
- 对某个列建立UNIQUE索引后，插入新记录时DBMS会自动检查新记录在该列上是否取了重复值。这相当于增加了一个UNIQUE约束

■ 聚簇索引(CLUSTER)

- 建立聚簇索引后，基表中数据也需要按指定的聚簇属性值的升序或降序存放。也即聚簇索引的索引项顺序与表中记录的物理顺序一致
- 第五章详细介绍



3.3 SQL数据定义语言

■ 建立索引例子

- **[例6]** 为dept和emp两个表建立索引。其中dept表按deptno升序建唯一索引，emp表按deptno降序建唯一索引。
- **CREATE UNIQUE INDEX DNO ON dept(deptno);**
- **CREATE UNIQUE INDEX ENO ON emp (deptno DESC);**



3.3 SQL数据定义语言

■ 四、其他模式对象的定义与撤消(cont.)

■ 撤销索引

- 语句格式: **DROP INDEX** <索引名>;
- 撤销索引时, 系统会从数据字典中删去有关该索引的描述。
- **[例7]** 撤销emp表的ENO索引。

DROP INDEX ENO;



目录 Contents

- 3.1 数据库的用户接口
- 3.2 SQL语言概况
- 3.3 SQL数据定义语言
- **3.4 SQL数据查询语言**
- 3.5 SQL数据操纵语言
- 3.6 SQL中的视图
- 3.7 嵌入式SQL



3.4 SQL数据查询语言

- **3.4.1 SELECT语句的语法**
- **3.4.2 各种条件查询举例**
- **3.4.3 查询结果分组**
- **3.4.4 查询结果排序**
- **3.4.5 集合操作查询**



3.4.1 SELECT语句的语法

一、语句格式

SELECT [**ALL** | **DISTINCT**] <列表表达式> [, <列表表达式>...]

FROM <表标识> [<别名>] [, <表标识> [<别名>] ...]

[**WHERE** <查询条件>]

[**GROUP BY** <列标识> [, <列标识>...] [**HAVING** <分组条件>]]

[**ORDER BY** <列标识> | <序号> [**ASC** | **DESC**]

[, <列标识> | <序号> [**ASC** | **DESC**]...]];

- **SELECT子句**：指定要显示的属性列
- **FROM子句**：指定查询对象(基本表或视图)
- **WHERE子句**：指定查询条件
- **GROUP BY子句**：对查询结果按指定列的值分组，该属性列值相等的元组为一个组。通常会在每组中使用聚集函数。
- **HAVING短语**：筛选出只有满足指定条件的组
- **ORDER BY子句**：对查询结果表按指定列值的升序或降序排序



3.4.1 SELECT语句的语法

■ 查询语句的组成

1) 目标子句:

SELECT {ALL | [DISTINCT]} <列表达式>[, <列表达式> ...]

- 定义结果关系所需要的属性

■ 目标子句的构造方式

- **DISTINCT**: 对查询结果中的重复行只返回其中一行, 消除结果关系中的重复元组。 (查询结果为set)
- **ALL**: (缺省)。返回查询结果的所有行, 不去掉重复行。
(查询结果为bag)
- **<列标识>**: 用以标识一个表中的一个列。
形式: [**<表名>** | **<别名>**.] **<列名>**。



3.4.1 SELECT语句的语法

■ 查询语句的组成(cont.)

1) 目标子句(cont.) :

SELECT { ALL | [DISTINCT] } <列表表达式>[, <列表表达式> ...]

■ 目标子句的构造方式(cont.)

- <列表表达式>: 算术表达式。用于投影表中的列，并对列值进行计算。定义如下：
 - 列标识是一个<列表表达式>;
 - 列标识的SQL函数是一个<列表表达式>;
 - <列表表达式>、常量、算术运算符（+, -, *, /）及括号所组成的算术表达式是一个<列表表达式>;
 - * 表示一个表中的所有列, 是一个特殊的<列表表达式>。
 - 结果属性的重命名
 - <column_expression> **AS** <column_name>



3.4.1 SELECT语句的语法

■ 查询语句的组成 (cont.)

2) 范围子句:

- **FROM** <表标识> [<别名>] [, <表标识> [<别名>] ...]
- 指定操作对象（被访问的关系）
- <表标识>与<列标识>
- <表标识>: 用以标识一个表（基表、视图或快照）。
 - 形式: [<模式名>.] <表名>。
- <列标识>: 用以标识一个表中的一个列。
 - 形式: [<表名>|<别名>.] <列名>。
- 可以将FROM子句中对一个关系重新命名（即定义一个别名）
 - <table_name> <alias_name>
 - 主要用于关系自身的联接运算

SELECT子句和**FROM子句**是查询语句中必不可少的两个部分



3.4.1 SELECT语句的语法

■ 查询语句的组成 (cont.)

3) 条件子句：WHERE <查询条件>

- 是查询语句中的可选部分，用于定义查询条件（即结果关系中的元组必须满足的条件）
- 包括“单个关系中的元组选择条件”以及“关系与关系之间的连接条件”都需要在WHERE子句中通过一定的逻辑表达式显式地表示出来。



3.4.1 SELECT语句的语法

查询语句的组成 (cont.)

3) 条件子句: **WHERE** <查询条件> (cont.)

■ <查询条件>

- 逻辑表达式, 用于选择表中的行。定义如下:

■ 简单条件:

- **比较条件**——用于比较大小:

<列标识> IS [NOT] NULL | Θ <列表表达式> | Θ [ALL | ANY | SOME](<子查询>)

其中, Θ 为关系运算符。

- **BETWEEN条件**——用于确定范围:

<列标识> [NOT] BETWEEN <列表表达式1> AND <列表表达式2>

- **LIKE条件**——用于字符匹配:

<列标识> [NOT] LIKE 'xx...x'

其中, x可为字符(精确匹配)、_ (单字符匹配)、% (任意多个字符匹配)。



3.4.1 SELECT语句的语法

3) 条件子句: **WHERE** <查询条件> (cont.)

- <查询条件> (cont.)

- 简单条件:

- **IN**条件——用于属于判断:

- <列标识> [**NOT**] **IN** (常量1, 常量2, ..., 常量n) | (<子查询>)

- **EXISTS**条件——用于存在判断:

- [**NOT**] **EXISTS** (<子查询>)

- 复合条件:

- 简单条件、逻辑运算符(**NOT**, **AND**, **OR**)及括号所组成的逻辑表达式。



3.4 SQL数据查询语言

- 3.4.1 SELECT语句的语法
- 3.4.2 各种条件查询举例
- 3.4.3 查询结果分组
- 3.4.4 查询结果排序
- 3.4.5 集合操作查询



3.4.2 各种条件查询举例

■ 背景

- dept (deptno, dname, loc)
- emp (empno, ename, job, mgr, hiredate, sal, comm, deptno)

■ 无条件查询

- 1) 查询全部职员的工号与姓名。

`SELECT empno, ename FROM emp;`

- 2) 查询全部部门的号码、名称、所在地。

`SELECT deptno, dname, loc FROM dept;`

或 `SELECT * FROM dept;`

- 3) 查询职员的所有工种。

`SELECT DISTINCT job FROM emp;`



3.4.2 各种条件查询举例

- 4) 查询每个职员提薪20%后的薪水。

```
SELECT empno, ename, sal*1.2 FROM emp;
```

- 5) 查询单位的薪水种类。

```
SELECT [ALL] sal FROM emp;
```

```
SELECT DISTINCT sal FROM emp;
```

- 注意 **DISTINCT** 短语的作用范围是所有目标列
- 例：查询所有主管经理的工号

- 错误的写法

```
SELECT DISTINCT ename, DISTINCT mgr FROM emp;
```

- 正确的写法

```
SELECT DISTINCT ename, mgr FROM emp;
```

这与 `SELECT ename, mgr FROM emp;` 查询结果不同



3.4.2 各种条件查询举例

■ 比较条件查询

- 6) 查询所有销售人员的姓名、所在部门号。

```
SELECT ename, deptno FROM emp WHERE job='salesman';
```

- 7) 查询薪水超过5000的职员。

```
SELECT empno, ename, sal FROM emp WHERE sal>5000;
```

- 8) 查询没有佣金的职员。

```
SELECT empno, ename FROM emp WHERE comm IS NULL;
```

错误:

```
SELECT empno, ename FROM emp WHERE comm = NULL;
```

■ 范围查询

- 9) 查询薪水在3000与5000之间的职员。

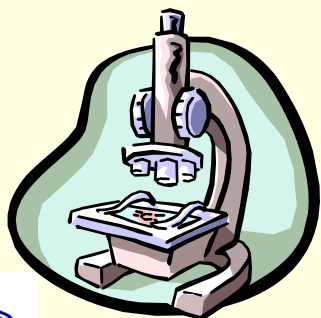
```
SELECT empno, ename FROM emp  
WHERE sal BETWEEN 3000 AND 5000;
```



3.4.2各种条件查询举例

■ 空值注意事项

- 除is [not] null之外，空值不满足任何查找条件
- 如果null参与算术运算，则该算术表达式的值为null
- 如果null参与比较运算，则结果可视为false。在SQL—92中视为unknown。
- 如果null参与聚集运算，则除count(*)之外其它聚集函数都忽略null



count (*) 与count (列名) 的差别



空值

select sum(G) **360**
from SC

select count(G) **4**
from SC

select count(*) **6**
from SC

S#	C#	G
s1	c1	80
s1	c2	90
s1	c3	95
s2	c1	85
s2	c2	null
s3	c2	null



3.4.2 各种条件查询举例

■ 字符匹配查询

- % (百分号) 代表任意长度（长度可以为0）的字符串

- 例：a%b表示以a开头，以b结尾的任意长度的字符串。如acb, addgb, ab 等都满足该匹配串

- _ (下横线) 代表任意单个字符

- 例：a_b表示以a开头，以b结尾的长度为3的任意字符串。如acb, afb等都满足该匹配串

- ESCAPE 短语：

- 当用户要查询的字符串本身就含有 % 或 _ 时，要使用 ESCAPE ‘<换码字符>’ 短语对通配符进行转义。
- 用\%去匹配%，用_去匹配_



3.4.2各种条件查询举例

■ 字符匹配查询

- 10) 找出姓名以M打头的所有职员。

```
SELECT deptno, ename FROM emp WHERE ename LIKE 'M%';
```

- 11) 找出姓名第三个字母为r的所有职员。

```
SELECT deptno, ename FROM emp  
WHERE ename LIKE '__R%';
```

- 12) 查询DB_Design课程的课程号和学分。

```
SELECT CNO, CREDIT FROM Course  
WHERE CNAME LIKE 'DB\_Design' ESCAPE '\';
```

■ 属于判断查询

- 13) 找出不是经理、销售人员的所有职员的薪水。

```
SELECT ename, job, sal FROM emp  
WHERE job NOT IN ( 'manager', 'salesman' );
```



3.4.2各种条件查询举例

■ 连接查询

- 14) (多表连接) 查询职员Allen的工作所在地。

```
SELECT ename, loc FROM emp, dept  
WHERE ename='Allen' AND emp.deptno = dept.deptno;
```

- 15) (单表连接) 查询薪水超过其部门经理的职员、及其经理姓名。

```
SELECT worker.ename, manager.ename  
FROM emp worker, emp manager  
WHERE manager.empno = worker.mgr AND worker.sal >  
manager.sal;
```

- 16) (单表连接) 查询薪水比Jones高的职员。

```
SELECT x.empno, x.ename  
FROM emp x, emp y  
WHERE y.ename = 'Jones' AND x.sal > y.sal;
```




3.4.2各种条件查询举例

■ 存在查询

- 17)查询所有已雇用职员部门。

```
SELECT deptno, dname FROM dept
WHERE EXISTS ( SELECT * FROM emp
                WHERE emp.deptno = dept.deptno);
```

子查询 (subquery) /
嵌套查询 (nested query)



■ 子查询 / 嵌套查询

- 子查询 的SELECT语句不能有ORDER BY之句；子查询可嵌套。
- 18)查询与Jones相同工种的所有职员。

```
SELECT empno, ename FROM emp
WHERE job = (SELECT job FROM emp
              WHERE ename = 'Jones') ;
```

↑
用IN更安全!



3.4.2 各种条件查询举例

子查询 / 嵌套查询

- 19) 查询比部门30中所有人薪水高的职员。 谓词!

```
SELECT empno, ename, deptno FROM emp
WHERE deptno <> 30 AND sal > ALL ( SELECT sal FROM emp
                                   WHERE deptno = 30 );
```

- 20) (相关子查询(correlated subquery))

查询薪水超过其所在部门平均薪水的所有职员。

```
SELECT empno, ename, sal
FROM emp x
WHERE sal > ( SELECT AVG (sal)
              FROM emp y
              WHERE y.deptno = x.deptno );
```

相关

SQL函数

SQL函数包括：
— 单行函数 e.g. ABS
— 组函数 e.g. AVG



3.4 SQL数据查询语言

- 3.4.1 SELECT语句的语法
- 3.4.2 各种条件查询举例
- **3.4.3 查询结果分组**
- 3.4.4 查询结果排序
- 3.4.5 集合操作查询



3.4.3 查询结果分组

- SELECT语句中，可使用**GROUP BY子句**对已选择的行进行分组，**HAVING子句**用于进一步选择已分的组，对每个已选中的组在查询结果中只返回其单行总计信息。
- **GROUP BY**将表中的元组按指定列上值相等的原则分组，然后在每一分组上使用聚集函数，得到单一值。
- **HAVING**则对分组进行选择，只将聚集函数作用到满足条件的分组上。



3.4.3 查询结果分组

- **GROUP BY**子句将表按列的值分组，列值相同的分在一组。
- 若**GROUP BY**子句后有多个列名，则从第一列名分组，再第二列，一直分下去。
- 使用**GROUP BY**子句分组，细化SQL函数的作用对象
 - 未对查询结果分组，SQL函数将作用于整个查询结果
 - 对查询结果分组后，SQL函数将分别作用于每个分组
- 使用**HAVING**子句筛选最终输出结果
 - 只有满足**HAVING**子句指定条件的组才输出
 - **HAVING**子句与**WHERE**子句的区别：作用对象不同
 - **WHERE**子句作用于基表或视图，从中选择满足条件的元组。
 - **HAVING**子句作用于分组，从中选择满足条件的元组。



3.4.3 查询结果分组

- 21) 查询每个部门的薪水最大值、最小值和平均值。

```
SELECT deptno, MAX(sal), MIN(sal), AVG(sal)  
FROM emp GROUP BY deptno;
```

- 22) 查询整个公司的薪水最大值、最小值和平均值。

```
SELECT deptno, MAX(sal), MIN(sal), AVG(sal) FROM emp;
```

- 23) 查询每个部门中clerk人员的人数、平均薪水。

```
SELECT deptno部门号, COUNT(*), AVG(sal) FROM emp  
WHERE job = 'clerk' GROUP BY deptno;
```

- 24) 查询每个部门中salesman人员最高薪水、最低薪水，要求最高薪水、最低薪水相差超过1000。

```
SELECT deptno, MAX(sal), MIN(sal) FROM emp  
WHERE job = 'salesman'  
GROUP BY deptno  
HAVING MAX(sal)-MIN(sal) > 1000;
```



3.4.3 查询结果分组

- 25) 查询每个部门中每个工种有多少职员。

```
SELECT deptno, job, COUNT(job)  
FROM emp  
GROUP BY deptno, job;
```

可能的结果是：

DEPTNO	JOB	COUNT (JOB)
11	clerk	2
11	manager	1
13	manager	2
13	analyst	4
13	clerk	3
12	manager	1
12	clerk	1
12	salesman	5



3.4 SQL数据查询语言

- 3.4.1 SELECT语句的语法
- 3.4.2 各种条件查询举例
- 3.4.3 查询结果分组
- **3.4.4 查询结果排序**
- 3.4.5 集合操作查询



3.4.4 查询结果排序

■ 对查询结果排序

■ 使用 **ORDER BY** 子句

- 可以按一个或多个属性列排序
- 升序: ASC; 降序: DESC;
- 缺省值为升序
- 排序时遵循 “**NULL值最大**” 原则
 - ASC: 排序列为空值的元组最后显示
 - DESC: 排序列为空值的元组最先显示

ORDER BY <列标识> | <序号> [ASC | DESC] [, <列标识> | <序号> [ASC | DESC]...]



3.4.4 查询结果排序

- 26) 查询每个部门中每个工种有多少职员，要求查询结果按deptno升序、job降序输出。

```
SELECT deptno, job, COUNT(job) FROM emp  
GROUP BY deptno, job;  
ORDER BY deptno, 2 DESC ;
```

可能的结果是：

DEPTNO	JOB	COUNT (JOB)
11	manager	1
11	clerk	2
12	salesman	5
12	manager	1
12	clerk	1
13	manager	2
13	clerk	3
13	analyst	4



3.4 SQL数据查询语言

- 3.4.1 SELECT语句的语法
- 3.4.2 各种条件查询举例
- 3.4.3 查询结果分组
- 3.4.4 查询结果排序
- 3.4.5 集合操作查询



3.4.5集合操作查询

- 标准SQL直接支持的集合操作种类
 - 并操作(UNION)
- 一般商用数据库支持的集合操作种类
 - 并操作(UNION)
 - 交操作(INTERSECT)
 - 差操作(MINUS)
- 这些运算符可连接多个SELECT，**括号**可改变缺省的运算次序。



3.4.5 集合操作查询

■ 并操作(UNION)

- 形式: <查询块> UNION <查询块>
- 参加UNION操作的各结果表的属性数必须相同; 对应的属性域也必须相同

- 27)列出所有老销售人员及刚刚雇用的新销售人员名单。

```
SELECT empno, ename FROM emp WHERE job = 'salesman'  
UNION
```

```
SELECT empno, ename FROM new-emp;
```

并兼容!



3.4 SQL数据查询语言

■ 回顾一下查询语句的格式

SELECT [**ALL** | **DISTINCT**] <列表达式> [, <列表达式>...]
FROM <表标识> [<别名>] [, <表标识> [<别名>] ...]
[**WHERE** <查询条件>]
[**GROUP BY** <列标识> [, <列标识>...] [**HAVING** <分组条件>]]
[**ORDER BY** <列标识> | <序号> [**ASC** | **DESC**]
[, <列标识> | <序号> [**ASC** | **DESC**]...]];



目录 Contents

- 3.1 数据库的用户接口
- 3.2 SQL语言概况
- 3.3 SQL数据定义语言
- 3.4 SQL数据查询语言
- **3.5 SQL数据操纵语言**
- 3.6 SQL中的视图
- 3.7 嵌入式SQL



3.5 SQL数据操纵语言

- **3.5.1 插入数据**
- **3.5.2 修改数据**
- **3.5.3 删除数据**



3.5.1 插入数据

- 两种插入数据方式

- 单行直接插入

- 多行间接插入



3.5.1 插入数据

■ 插入单个元组

■ 语句格式

INSERT INTO <表名> [(<属性列1>[, <属性列2 >...])]

VALUES (<常量1> [, <常量2>] ...)

- 属性列的顺序可与表定义中的顺序不一致。
- 属性列表可以被省略。在此情况下，表示要插入的是一条完整的元组，属性的排列顺序采用基表定义中的顺序。
- 常量表表示被插入的常量元组值。其中属性值的数量及其排列顺序必须与INTO子句的属性名列表一致。
- 每个属性值可以为空，用NULL表示空值。

■ 功能

- 将新元组插入指定表中。



3.5.1 插入数据

- 28)在dept表中增加一个新部门。

```
INSERT INTO dept  
VALUES (14, 'production', 'Wuhan');
```

- 29)在emp表中增加一个新的销售员工，部门为14，部门经理为158。

```
INSERT INTO emp (empno, ename, job, mgr, hiredate, deptno)  
VALUES (298, 'Julian', 'salesman', 158, 2006-09-01, 14);
```

新插入的记录在sal和comm列上取空值。

等价于：

```
INSERT INTO emp  
VALUES (298, 'Julian', 'salesman', 158, 2006-09-01, NULL, NULL, 14);
```



3.5.1 插入数据

- 多行间接插入

- 语句格式

INSERT INTO <表名> [(<属性列1> [, <属性列2>...)]
子查询;

- 功能:

- 将子查询结果插入指定表中



3.5.1 插入数据

- 30) (多行间接插入)将emp表中manager员工或佣金超过其工资50%的员工之相关数据拷贝到bonus表中。

INSERT INTO bonus (e-name, work, salary, comm)

SELECT ename, job, sal, comm

FROM emp

WHERE job='manager' OR comm > 0.5*sal ;

- **注：**大型DBMS厂商一般还提供非SQL手段的批量数据插入工具。
- **e.g.** Oracle SQL*Loader 可将Text数据、Excel数据批量插入基表。



3.5 SQL数据操纵语言

- 3.5.1 插入数据
- 3.5.2 修改数据
- 3.5.3 删除数据



3.5.2 修改数据

■ 语句格式

UPDATE <表名>

SET <列名> = <表达式>[, <列名> = <表达式>]...

[**WHERE** <条件>];

■ 功能

- 修改指定表中满足WHERE子句条件的元组。即：用SET子句中的赋值语句修改相关元组上的属性值。

■ 三种修改方式

- 修改某一个元组的值
- 修改多个元组的值
- 带子查询的修改语句



3.5.2 修改数据

- 31) (全部更新) 将emp表中所有员工的佣金置为NULL。

UPDATE emp SET comm = NULL ;

- 32) (条件更新) 将Jones提升为14部门的经理，其薪水增加2000。

UPDATE emp

SET job='manager', sal=sal+2000.0, deptno=14

WHERE ename='Jones' ;

- 33) (复杂更新) 在西安或武汉的部门要撤消，其员工全部调入公司总部，职务不变，薪水和佣金分别是原部门平均值的1.1倍和1.5倍。

UPDATE emp x

**SET deptno = (SELECT deptno FROM dept WHERE
dname=' headquarters'),**

**(sal, comm) = (SELECT 1.1*AVG(sal), 1.5*AVG(comm) FROM emp y
WHERE y.deptno = x.deptno)**

WHERE deptno IN (SELECT deptno FROM dept

WHERE loc=' Xian' OR loc=' Wuhan');



3.5 SQL数据操纵语言

- 3.5.1 插入数据
- 3.5.2 修改数据
- 3.5.3 删除数据



3.5.3 删除数据

■ 语句格式

- **DELETE FROM** <表名> [**WHERE** <条件>];

■ 功能

- 删除指定表中满足WHERE子句条件的**元组**
- WHERE子句
 - 指定要删除的元组
 - 缺省表示要修改表中的所有元组，但表仍作为一个空表存在。

■ 三种删除方式

- 删除某一个元组的值
- 删除多个元组的值
- 带子查询的删除语句



3.5.3 删除数据

- 34) (全部删除) 将emp表中全部行删除。

DELETE FROM emp ;

- [注]: 区别DROP TABLE语句。

- 35) (条件删除) 将emp表无佣金及佣金低于500的salesman数据。

DELETE FROM emp

WHERE job='salesman' AND (comm IS NULL OR comm < 500.0) ;

- DBMS在执行插入语句时会检查所插元组是否破坏表上已定义的完整性规则
 - 引用完整性
 - 不允许删除
 - 级联删除



目录 Contents

- 3.1 数据库的用户接口
- 3.2 SQL语言概况
- 3.3 SQL数据定义语言
- 3.4 SQL数据查询语言
- 3.5 SQL数据操纵语言
- **3.6 SQL中的视图**
- 3.7 嵌入式SQL



3.6 SQL中的视图

- **3.6.1 视图的概念**
- **3.6.2 定义与撤销视图**
- **3.6.3 视图上查询数据**
- **3.6.4 视图上更新数据**



3.6.1 视图的概念

■ 视图 (view)

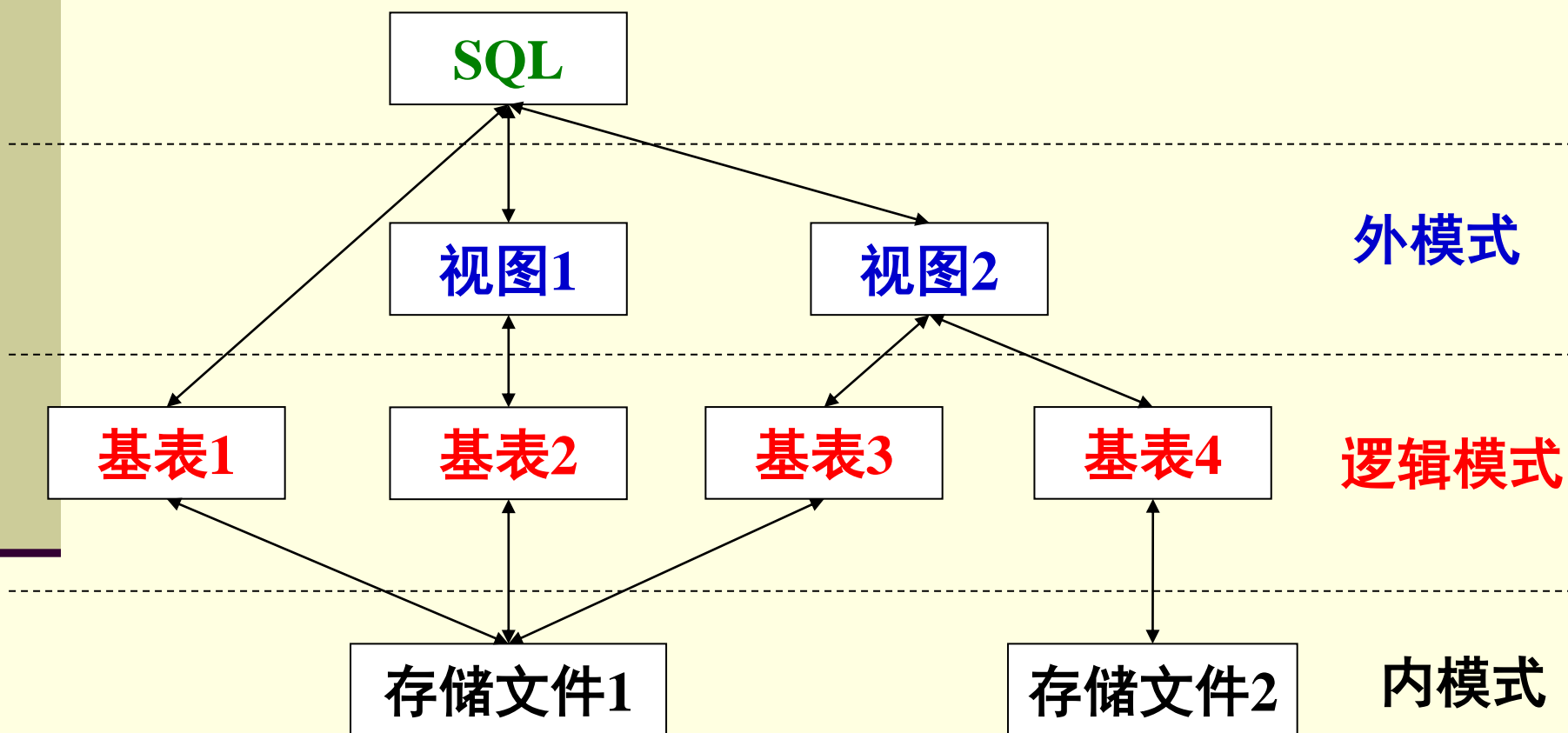
- 由其他表（基表/视图）导出的**虚表**，可用于定义外模式。
- 又称为：**导出表(derived table)**

■ 视图与基表(base table)的区别

- 视图，如同基表（base table）一样，它是由行、列组成的二维表，在其中可查询数据、**更新数据（有限制）**。
- 但**视图中不直接包含数据**（即不对应物理数据文件），其数据包含在导出它的基表中。视图**仅仅保留了其构造信息**(有关视图的定义信息，即逻辑定义)。因此，视图又被称为‘**虚表**’ (virtual table)
- 由于数据库中只存放视图的定义，不会出现数据冗余。
- 当用户执行视图上的访问操作时，DBMS将根据视图的定义命令将用户对于视图的访问操作转换称相应基表上的访问操作。
- 基表中的数据发生变化，从视图中查询出的数据也随之改变。



3.6.1 视图的概念



3.6.1 视图的概念

■ 视图的作用

■ 1. 视图能够简化用户的操作

- 当视图中数据不是直接来自基本表时，定义视图能够简化用户的操作

- 基于多张表连接形成的视图
- 基于复杂嵌套查询的视图
- 含导出属性的视图

■ 2. 视图使用户能以多种角度看待同一数据

- 视图机制能使不同用户以不同方式看待同一数据，适应数据库共享的需要

■ 3. 视图对重构数据库提供了一定程度的逻辑独立性

■ 4. 视图能够对机密数据提供安全保护

- 对不同用户定义不同视图，使每个用户只能看到他有权看到的数据



3.6 SQL中的视图

- 3.6.1 视图的概念
- 3.6.2 定义与撤销视图
- 3.6.3 视图上查询数据
- 3.6.4 视图上更新数据



3.6.2 定义与撤销视图

■ 建立视图的语句格式

CREATE VIEW <视图名> [(<列名> [, <列名>]...)]

AS <子查询> [**WITH CHECK OPTION**];

- 创建一个<视图名>为表名的视图，对应的数据查询语句是<子查询>语句。以<子查询>作查询所得到的查询结果即是该视图中的元组。
- 如果省略视图中的属性命名，则用<子查询>的SELECT子句中的属性名作为视图的属性名。否则视图中的属性必须与<子查询>的SELECT子句中的结果属性一一对应。
- 明确指定视图的所有列名：
 - (1) 某个目标列是SQL函数或列表表达式
 - (2) 需要在视图中为某个列启用新的更合适的名字



3.6.2 定义与撤销视图

- **WITH CHECK OPTION**用于约束视图上的修改操作：
 - 如果允许在该视图上执行更新操作，则其更新后的结果元组仍然必须满足视图的定义条件。即通过该视图插入或修改的新元组能够通过该视图上的查询操作。
- DBMS执行**CREATE VIEW**语句时只是把视图的定义存入数据字典，并不执行其中的SELECT语句。
- 在对视图查询时，按视图的定义从基本表中将数据查出。



3.6.2 定义与撤销视图

■ 定义视图例子

- 36) 创建全体员工工作所在地的视图。

```
CREATE VIEW emp-loc
```

```
AS SELECT empno, ename, loc
```

```
FROM emp, dept
```

```
WHERE emp.deptno = dept.deptno ;
```

- 37) 创建12号部门全体员工年薪视图。

```
CREATE VIEW emp-ann-sal (eno, ename, ann-sal)
```

```
AS SELECT empno, ename, 12*sal
```

```
FROM emp
```

```
WHERE deptno = 12 ;
```



3.6.2 定义与撤销视图

■ 撤销视图的语句格式

DROP VIEW <视图名>

- 该语句从数据字典中删除指定的视图定义
- 由该视图导出的其他视图定义仍在数据字典中，但已不能使用，必须显式删除
- 删除基表时，由该基表导出的所有视图定义都必须显式删除
- 例：**DROP VIEW** emp-loc;



3.6 SQL中的视图

- 3.6.1 视图的概念
- 3.6.2 定义与撤销视图
- 3.6.3 视图上查询数据
- 3.6.4 视图上更新数据



3.6.3 视图上查询数据

- 对**用户**而言

- 如同在基表中查询数据一样，使用SELECT语句。

- 对**系统**而言

- 需进行视图消解(resolution):
 - 从数据字典中取视图定义；
 - 视图上查询转换成基表上查询；
 - 再执行基表上查询。

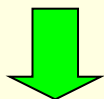


3.6.3 视图上查询数据

■ 查询视图例子

- 38) 查询在南京工作的全体员工名单。

```
SELECT ename  
FROM emp-loc  
WHERE loc='Nanjing';
```



```
SELECT ename  
FROM emp, dept
```

```
WHERE loc='Nanjing' AND emp.deptno =dept.deptno ;
```

```
CREATE VIEW emp-loc  
AS SELECT empno, ename, loc  
FROM emp, dept  
WHERE emp.deptno = dept.deptno ;
```

- 对于在定义中使用“列表表达式”或“GROUP BY/分组函数”而形成的视图，其消解过程很麻烦。系统会先将视图形成一个临时表，然后再在临时表上执行查询。总之，视图消解由DBMS完成，用户不必考虑。



3.6 SQL中的视图

- 3.6.1 视图的概念
- 3.6.2 定义与撤销视图
- 3.6.3 视图上查询数据
- 3.6.4 视图上更新数据



3.6.4 视图上更新数据

- 视图上更新就不象查询那样容易消解了。

- 39) 假如创建部门平均薪水的视图。

```
CREATE VIEW emp-avg-sal (deptno, avg-sal)
AS SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno ;
```

- 以下更新就无法消解：

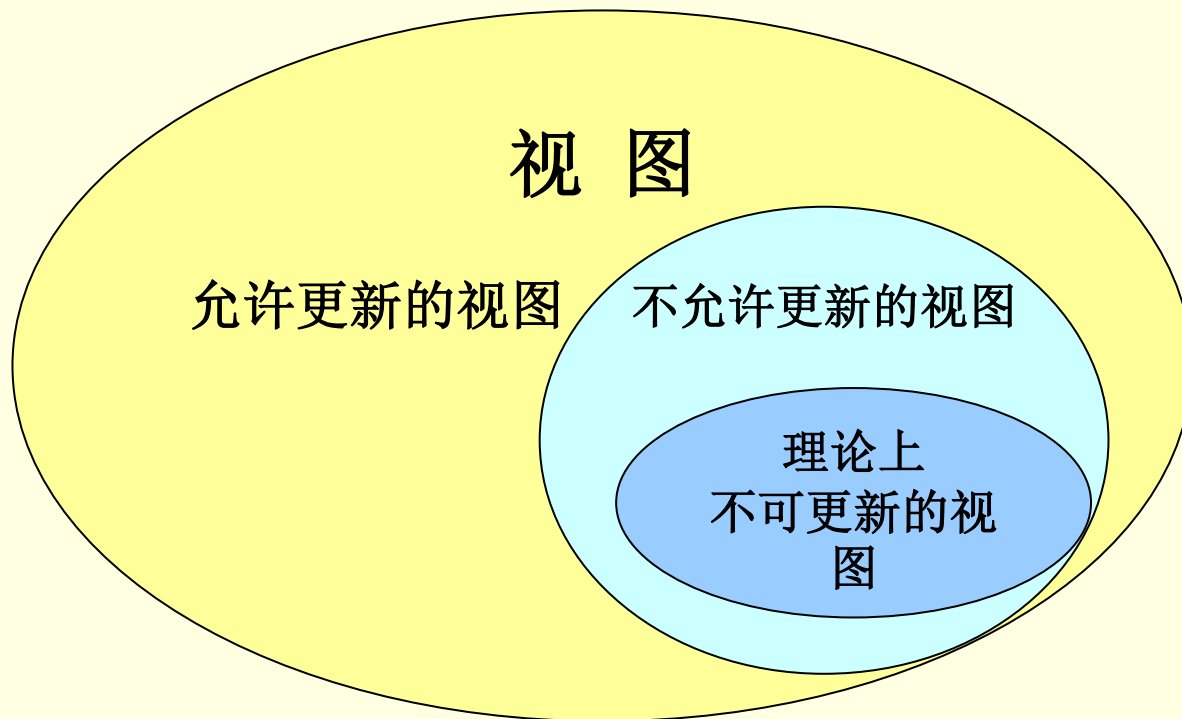
```
UPDATE emp-avg-sal
SET avg-sal = avg-sal + 1000
WHERE deptno = 12 ;
```

- 视图上更新就必须有所限制。



3.6.4 视图上更新数据

- 一般地，严格规定：含主键的单表行列子集视图可以更新。



- 注：SQL:1999扩大了可更新视图(updatable view)的范围(取决于函数依赖)。



目录 Contents

- 3.1 数据库的用户接口
- 3.2 SQL语言概况
- 3.3 SQL数据定义语言
- 3.4 SQL数据查询语言
- 3.5 SQL数据操纵语言
- 3.6 SQL中的视图
- **3.7 嵌入式SQL**



3.7 嵌入式SQL与SQL过程化扩充

- 交互式SQL (Interactive SQL)是计算不完备的、非过程化数据库语言。
- 但开发一个完整的数据库应用（程序）时，往往需要数据“管理”与“计算”的集成、数据库操作与其他处理的集成。
- 因此，就有**嵌入式SQL (embedded SQL)**和**SQL过程化扩充**。



3.7 嵌入式SQL与SQL过程化扩充

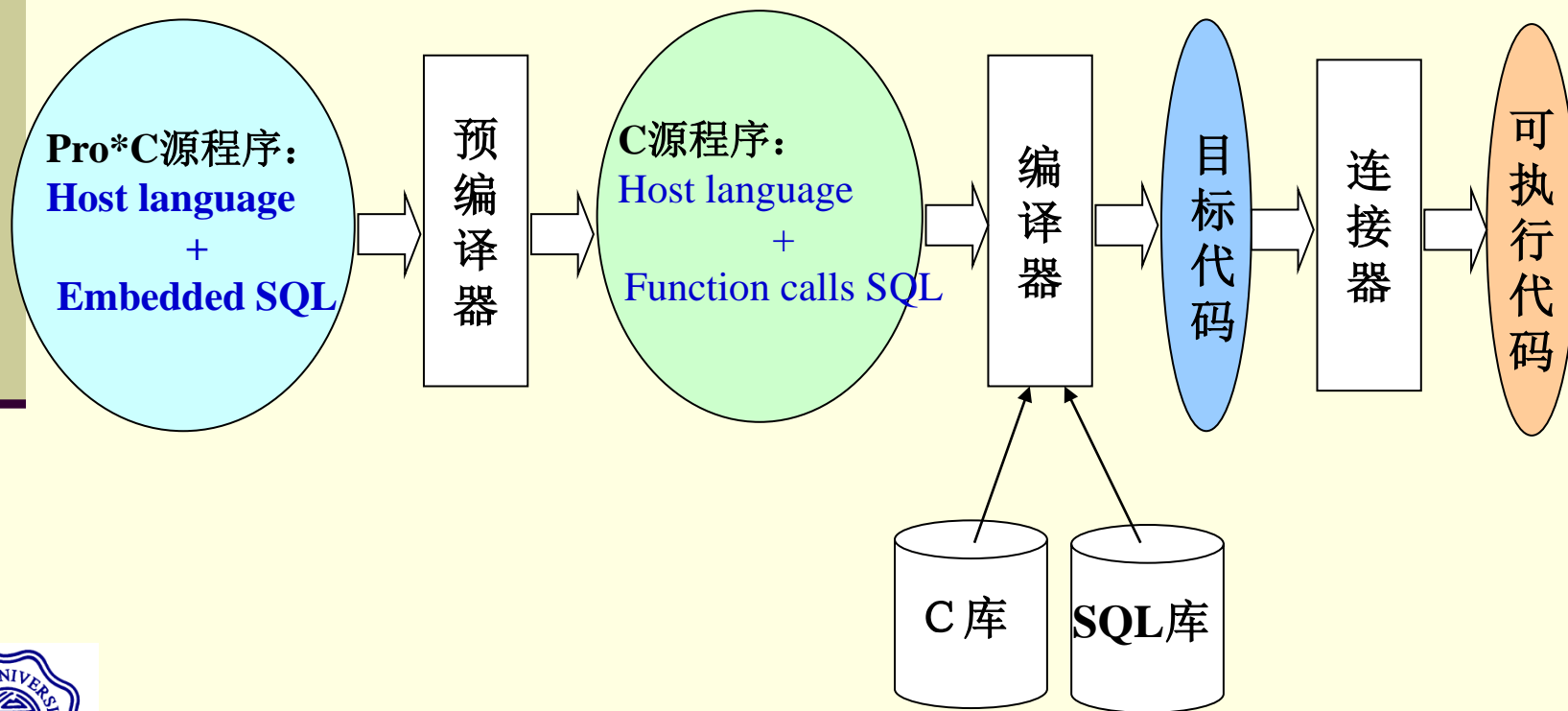
一、嵌入式SQL

- 将SQL语言嵌入到宿主语言(e.g. PASCAL, C, FORTRAN)中, 混合编成。SQL负责“数据库操作”, 宿主语言负责“其他处理”。e.g. Oracle Pro*C, Pro*FORTRAN etc.
- 然而, 需解决以下几个问题:
 - 如何将包含外语 (SQL) 的宿主语言源程序编译、连接成可执行代码?
 - 预编译器如何识别SQL语句?
 - DBMS如何识别SQL语句中的宿主语言变量?
 - 程序如何访问数据库?
 - 程序工作单元与数据库工作单元之间如何通讯?
 - 程序中如何逐行处理SELECT返回的多行结果?



3.7 嵌入式SQL与SQL过程化扩充

- 1、如何将包含外语(SQL)的宿主语言源程序编译、连接成可执行代码？
 - 提供预编译器(precompiler)



3.7 嵌入式SQL与SQL过程化扩充

■ 2、预编译器如何识别SQL语句？

■ SQL语句前加标志：

- 前缀：EXEC SQL <SQL语句>
- 结束：随宿主语言的不同而异，如END_EXEC 或以C为主语言的嵌入式SQL语句的一般形式
EXEC SQL <SQL语句>;

例：EXEC SQL DROP TABLE emp;



3.7 嵌入式SQL与SQL过程化扩充

3、DBMS如何识别SQL语句中的宿主语言变量？

■ 宿主语言变量前加标志：

e.g. eno=100;

EXEC SQL SELECT ename, comm

INTO :en, :commission :commission-id

FROM emp

WHERE empno = :eno ;

IF (commission-id == -1)

PRINTF (“Commission is NULL\n”);

输出主变量

指示变量

输入主变量



3.7 嵌入式SQL与SQL过程化扩充

■ 4、程序如何访问数据库？

- 以合法的数据库用户身份，通过CONNECT语句连接到数据库：

EXEC SQL CONNECT :username IDENTIFIED BY :password ;

■ 5、程序工作单元与数据库工作单元之间如何通讯？

- 通过SQL通讯区SQLCA：
- The SQLCA contains run-time information about the execution of SQL statements, such as:
 - Oracle error codes
 - warning flags
 - event information
 - rows-processed count
 - diagnostics

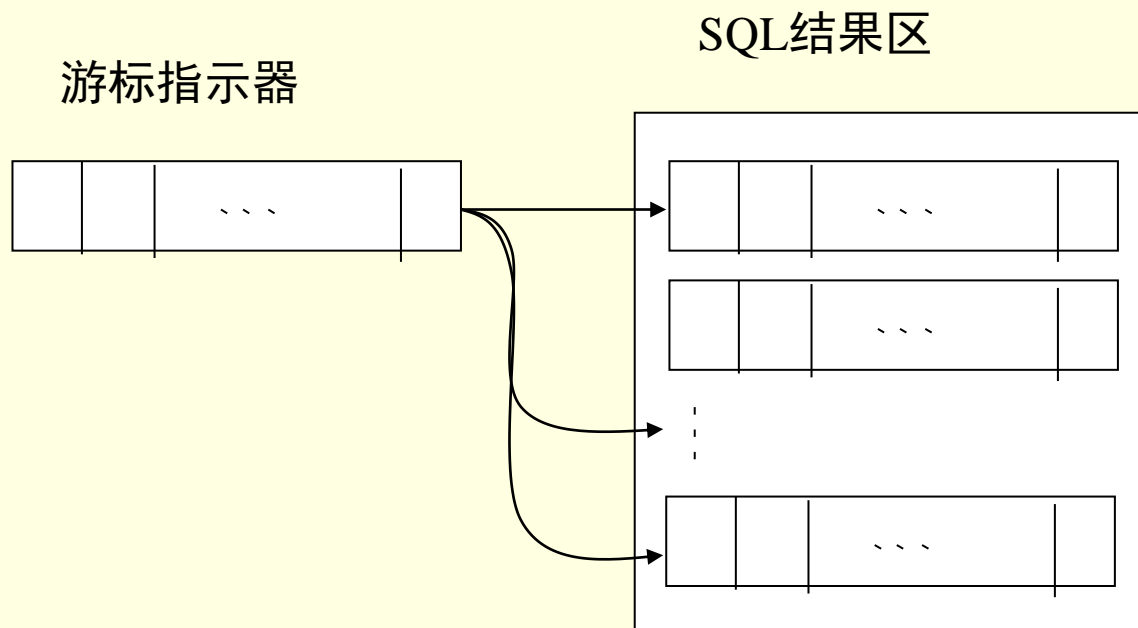


3.7 嵌入式SQL与SQL过程化扩充

6、程序中如何逐行处理SELECT返回的多行结果？

■ 通过游标(Cursor)机制 / 扩充的SQL语句：

- 定义游标：DECLARE <游标名> CURSOR FOR <SELECT语句>；
- 打开游标：OPEN <游标名>；
- 逐行取数：FETCH <游标名> INTO <主变量列表>;
- 关闭游标：CLOSE <游标名>;



3.7 嵌入式SQL与SQL过程化扩充

■ 二、SQL过程化扩充

- 1996年，SQL-92增加了一个关于存储过程的补充标准SQL-92/PSM(Persistent Stored Modules)。各种RDBMS厂商的SQL实现中也包含了诸多过程化扩充。e.g. Oracle PL/SQL
 - 存储过程与触发器
 - 流程控制语句（分支、循环等）
 - 增强的计算子句
 - 自定义数据类型、规则与缺省值
 - 错误处理
 - 系统管理

