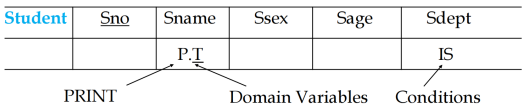
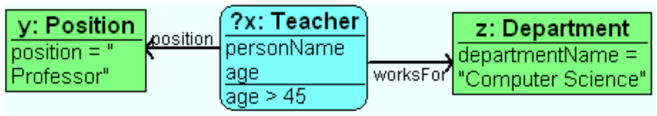
Chapter 3 User Interfaces and SQL Language

1. User interface of DBMS

- 数据库管理系统必须提供一些接口来支持用户访问数据库，包括：查询语言、接口和维护工具（GUI）、API、类库
- 查询语言
  - 正式查询语言Formal Query Language
  - 表格查询语言Tabular Query Language(p1)
  - 图形查询语言Graphic Query Language(p2)
  - 有限自然语言查询语言Limited Natural Language Query Language





1.1. 关系查询语言

- 查询语言：允许从数据库中操作和检索数据
- 关系模型支持简单而强大的QLs：
  - 基于逻辑的强大正式基础
  - 允许进行大量优化
- QLs与编程语言不同：
  - QLs不是“图灵完备”的
  - 不使用QLs进行复杂的计算
  - QLs支持对大型数据集进行简单而高效的访问

1.2. 正式关系查询语言 Formal Relational Query Languages

- 两种数学查询语言形成了真正的语言（如SQL）的基础和实现：
  - 关系代数：更注重操作，非常有助于表示执行计划
  - 关系演算：让用户描述他们想要什么，而不是如何计算它。（非操作性的，是声明性的。）
- 最成功的的关系数据库语言——SQL（结构化查询语言；标准查询语言（1986年）；现在SQL:2016）

2. SQL

执行顺序

> from->on->join->where->group by（从这一步开始，后面的语句中都可以使用select中别名）->agg function->having->select->distinct->order by->limit

2.1. 优点和限制

- 声明式（Declarative）：说明要什么，忽略怎么得到
- 广泛实现（Implement widely）：具有各种效率、完整性水平
- 受限（Constraints）：不针对Turing-complete任务进行优化
- 通用的、功能丰富的 General-purpose and feature-rich
  - 多年添加新功能
  - 可扩展性：可调用其他语言、数据源
- **Summary：现代SQL扩展了“纯粹”的关系模型（重复行、非原子类型），DBMS会找出一种快速执行查询的方法，无论查询如何编写**

2.2. SQL语言分类

- 数据定义语言（Data Definition Language）：用于定义、删除或修改数据架构
- 查询语言（Query Language）：用于检索数据
- 数据操纵语言（Data Manipulation Language）：用于插入、删除或更新数据
- 数据控制语言（Data Control Language）：用于控制用户对数据的访问权限
- RDBMS负责高效评估
  - 选择并运行声明性查询的算法，算法的选择不得影响查询答案。

2.3. DDL

```
--建表
CREATE TABLE Reserves (
  sid INTEGER, bid INTEGER, day DATE,
  PRIMARY KEY (sid, bid, day)
  FOREIGN KEY (sid) REFERENCES Sailors,
  FOREIGN KEY (bid) REFERENCES Boats);
```

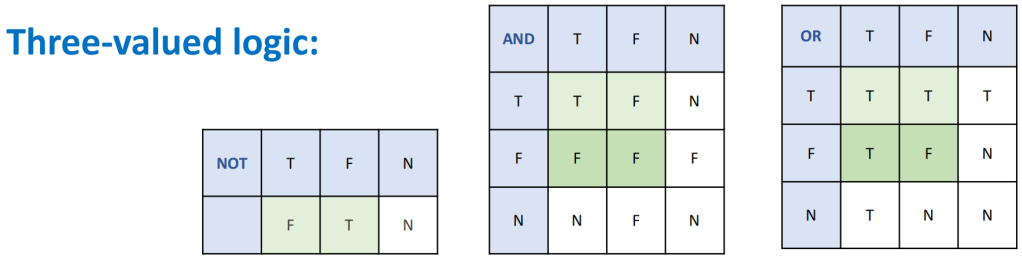
```sql
-- Insert
insert into instructor values ('10211','Smith', 'Biology', 66000);

-- Update
update teacher set salary=2000 where name='william'

-- Delete: Remove all tuples from the student relation
delete from student;

-- Drop Table
drop table r;

-- Alter
-- 为表r增加attribute A, A的域为D
alter table r add A D;
-- 删除表r的attribute A
alter table r drop A;
```

2.4. QL

```sql
-- Like
/*
AS and = are two ways to name fields in result.
LIKE is used for string matching.
‘_’: any one character; ‘%’: 0 or more characters.
*/
SELECT S.age, age1=S.age-5, 2*S.age AS age2
FROM Sailors S
WHERE S.sname LIKE ‘B_%B’;

-- Order by
-- Desc 降序 Asc 升序

-- General Single-Table Queries
SELECT [DISTINCT] <column expression list>
FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list>
[HAVING <predicate>] ]
[ORDER BY <column list>]
[LIMIT <integer>];
```

```sql
-- UNION[ALL](or) EXCEPT[ALL](but not) INTERSECT[ALL](and)
-- 加All按列表进行计算（个数+，-，min），不加按集合计算
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
AND B.color='red'
[INTERSECT/UNION/EXCEPT[ALL]]
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
AND B.color='green'

-- ⭐ EXISTS
SELECT S.sname FROM Sailors S
WHERE EXISTS
          (SELECT *
          FROM Reserves R
          WHERE R.bid=103 AND S.sid=R.sid)

-- We've already seen IN, EXISTS. Can also use NOT IN, NOT EXISTS.
-- Also available: op ANY, op ALL, op IN <,>,=,≤,≥,≠
SELECT * FROM Sailors S
WHERE S.rating > ANY (SELECT S2.rating
                      FROM Sailors S2
                      WHERE S2.sname='Horatio')

-- Find sailors who've reserved all boats.
-- with EXCEPT
SELECT S.sname FROM Sailors S
WHERE NOT EXISTS (
    (SELECT B.bid
    FROM Boats B)
    EXCEPT
    (SELECT R.bid
    FROM Reserves R
    WHERE R.sid=S.sid))

-- without except
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (SELECT B.bid
                  FROM Boats B
                  WHERE NOT EXISTS (SELECT R.bid
                                    FROM Reserves R
                                    WHERE R.bid=B.bid
                                    AND R.sid=S.sid))
```

2.4.1. 关于NULL：



where NULL 的结果不输出

General rule: **NULL column values are ignored** by aggregate functions

COUNT、SUM、AVG、MIN、MAX（带或不带DISTINCT）

- 丢弃null值然后应用聚合函数
- 除了count(*)
- 如果只对 null 值应用，结果为null：(sum(values)而values全为null)

2.4.2. CAST Expression

- 改变表达式为目标数据类型

```
-- 匹配函数参数
SELECT substr(string1, CAST(x AS Integer), CAST(y AS Integer))
FROM Test

-- 计算时改变精度
SELECT name, CAST(grade as Demical(5,0))
FROM Students

-- 将NULL值分配给数据类型
-- 貌似不能将NULL改为INTEGER
CREATE VIEW prospects (name, school, service) AS
  SELECT name, school, CAST(NULL AS Varchar(20))
  FROM Students
UNION
  SELECT name, CAST(NULL AS Varchar(20)), service
  FROM Soldiers ;
```

2.4.3. CASE

```
-- Demo for CASE
SELECT name, CASE status
                    WHEN 1 THEN 'Active Duty'
                    WHEN 2 THEN 'Reserve'
                    WHEN 3 THEN 'Special Assignment'
                    WHEN 4 THEN 'Retired'
                    ELSE 'Unknown'
               END AS status
FROM Officers;

SELECT type, CASE
                   WHEN sum(hours_used)>0 THEN
                       sum(accidents)/sum(hours_used)
                   ELSE NULL
               END AS accident_rate
FROM Machines
GROUP BY type;
```

2.4.4. 公用表表达式 Common Table Expression

- 在某些复杂的查询中，一个表对应的表达式可能需要出现在同一个SQL语句中多次。
    - 虽然这是允许的，但是效率低，且可能存在不一致的问题
- WITH子句可以用于定义公用表表达式：实际上，它定义了一个临时视图

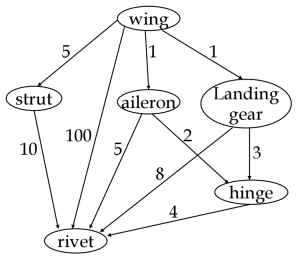
```
WITH payroll (deptno, totalpay) AS
    (SELECT deptno, sum(salary)+sum(bonus)
    FROM emp
    GROUP BY deptno)
SELECT deptno
FROM payroll
WHERE totalpay = (SELECT max(totalpay)
                    FROM payroll);
```

2.4.5. Recursion 递归查询

如果一个通用表表达式在它的定义中使用了自身，这就被称为递归。

Components

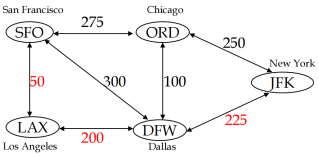
Part	Subpart	QTY
wing	strut	5
wing	aileron	1
wing	landing gear	1
wing	rivet	100
strut	rivet	10
aileron	hinge	2
aileron	rivet	5
landing gear	hinge	3
landing gear	rivet	8
hinge	rivet	4



Directed acyclic graph, which assures the recursion can be stopped

```
1   WITH wingpart (subpart, qty) AS
2         ((SELECT subpart, qty              ---initial query
3           FROM components
4           WHERE part='wing')
5         UNION ALL
6         (SELECT c.subpart, w.qty*c.qty     ---recursive qry
7           FROM wingpart w, components c
8           WHERE w.subpart=c.part))
9   SELECT subpart, sum(qty) AS qty FROM wingpart
10  Group BY subpart;
```

- Typical airline route searching problem
- ***Find the lowest total cost route from SFO to JFK***



Flights

FlightNo	Origin	Destination	Cost
HY 120	DFW	JFK	225
HY 130	DFW	LAX	200
HY 140	DFW	ORD	100
HY 150	DFW	SFO	300
HY 210	JFK	DFW	225
HY 240	JFK	ORD	250
HY 310	LAX	DFW	200
HY 350	LAX	SFO	50
HY 410	ORD	DFW	100
HY 420	ORD	JFK	250
HY 450	ORD	SFO	275
HY 510	SFO	DFW	300
HY 530	SFO	LAX	50
HY 540	SFO	ORD	275

```
1   WITH trips (destination, route, nsegs, totalcost) AS
2       ((SELECT destination, CAST(destination AS varchar(20)), 1, cost
3         FROM flights                                    --- initial query
4         WHERE origin='SFO')
5       UNION ALL
6         (SELECT f.destination,                          --- recursive query
7         CAST(t.route||','||f.destination AS varchar(20)),
8         t.nsegs+1, t.totalcost+f.cost
9         FROM trips t, flights f
10        WHERE t.destination=f.origin
11                          AND f.destination<>'SFO'      --- stopping rule 1
12                          AND f.origin<>'JFK'           --- stopping rule 2
13                          AND t.nsegs<=3))              --- stopping rule 3
14  SELECT route, totalcost FROM trips                    --- final query
15  WHERE destination='JFK' AND totalcost=                --- lowest cost rule
16                                  (SELECT min(totalcost) FROM trips
17                                  WHERE destination='JFK') ;
```

2.4.6. 视图：Views

- 任何不是概念模型，但被用户显示为“虚拟关系”的关系，称为视图。

2.4.6.1. 创建与使用视图

  - 使开发更简单
  - 通常用于安全目的
  - 不是“物理化的（marterialized）”
- 一旦定义了视图，就可以使用视图名称来引用视图生成的虚拟关系
- 视图定义与通过评估查询表达式创建新关系不同
  - 视图定义会导致保存一个表达式；在查询中使用视图时将表达式替换为视图。

```
-- create
CREATE VIEW Redcount
AS SELECT B.bid, COUNT(*) AS scount
    FROM Boats2 B, Reserves2 R
    WHERE R.bid=B.bid AND B.color='red'
    GROUP BY B.bid;
-- use
SELECT bname, scount
FROM Redcount R, Boats2 B
WHERE R.bid=B.bid
AND scount < 10;
```

- 一个视图可以在定义另一个视图的表达式中使用
  - 视图关系v1被称为直接依赖于（depend directly）视图关系v2，如果v2在定义v1的表达式中被使用
  - 视图关系v1被称为依赖于（depend on）视图关系v2，如果v1直接依赖于v2，或者存在从v1到v2的依赖路径
  - 视图关系v被称为递归（rescursive）的，如果它依赖于自身

2.4.6.2. 材料视图：Materialized Views（物理化）

- 某些数据库系统允许将视图关系物理化存储
  - 物理副本在定义视图时创建，这样的视图被称为材料视图
- 如果查询中使用的关系被更新，材料视图的结果就会过时
  - 需要维护视图，每当底层关系被更新时，都需要更新视图

2.4.6.3. 更新视图：Update a View

- 向之前定义的教员视图添加一个新的元组

```
insert into faculty
        values ('30765', 'Green', 'Music');
```

- 这个插入必须通过向教师关系中插入来表示。（会插入原表）
  - 必须有一个薪水值。
  - 将元组('30765', 'Green', 'Music', null)插入到教师关系中。
- 一些更新无法唯一翻译

```
--  创建视图instructor_info的SQL语句：
select ID, name, building
from instructor, department
where instructor.dept_name = department.dept_name;
-- 向instructor_info插入数据的SQL语句：
insert into instructor_info values ('69987', 'White', 'Taylor');
```

  - 问题：如果Taylor有多个部门，应该选择哪个部门｜如果Taylor没有部门，应该怎么办?
- 大多数SQL实现只允许对简单视图进行更新。
  - FROM子句只有一个数据库关系
  - SELECT子句只包含关系的属性名，不包含任何表达式、聚合或DISTINCT
  - 在SELECT子句中未列出的任何属性都可以设置为NULL
  - 查询不包含GROUP BY或HAVING子句

2.5. DML

```sql
--Insert: Insert a tuple into a table
INSERT INTO EMPLOYEES
VALUES ('Smith', 'John', '1980-06-10', 'Los Angles', 16, 45000);

--Delete: Delete tuples fulfill qualifications
DELETE FROM Person WHERE LastName = 'Rasmussen' ;

--Update: Update the attributes' value of tuples fulfill qualifications
UPDATE Person SET Address = 'Zhongshan 23', City = 'Nanjing'
WHERE LastName = 'Wilson';

-- DDL?
DROP TABLE r
ALTER TABLE r ADD A D
ALTER TABLE r DROP A
```

2.6. 嵌入式SQL Embedd SQL

- 为了在程序中访问数据库，并对查询结果进行进一步处理，需要结合SQL和编程语言（如C/C++等）
- 嵌入式SQL：通过预编译，将嵌入式的SQL语句传输到内部库函数调用中以访问数据库
- 编程API：向程序员提供一套库函数或DLL，可以直接链接到应用程序中进行编译
- 类库 Class Library：面向对象编程之后出现，封装了访问数据库的库函数，作为一组类提供。

2.6.1. Usage of Embedded SQL（in C）

- 使用EXEC SQL开始，以SQL语句结束，并以“；”结束
- 通过Host variabels在C和SQL之间传输信息。Host variables应该以EXEC SQL开头进行定义
- 在SQL语句中，应该在 host variables 前添加 “：”以区分SQL自身的变量或属性名称
- 在主机语言（如C）中，host variales用作一般变量，不能定义为结构体
- 一种特殊的host variable：SQLCA（SQL通信区域）
  - EXEC SQL INCLUDE SQLCA
  - 使用SQLCA.SQLCODE 来表明结果状态
- 使用指示器（短整型）来处理host variables中的NULL值。

```
// Example of host variables defining
EXEC SQL BEGIN DECLARE SECTION;
char SNO[7];
char GIVENSNO[7];
char CNO[6];
char GIVENCNO[6];
float GRADE;
short GRADEI;               /*indicator of GRADE*/
EXEC SQL END DECLARE SECTION;

//CONNECT
EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;

//Execute DDL or DML Statements
EXEC SQL INSERT INTO SC(SNO,CNO,GRADE) VALUES(:SNO, :CNO, :GRADE);

//Execute Query Statements 由于{SNO,CNO}主键确定唯一
EXEC SQL SELECT GRADE
INTO :GRADE :GRADEI
FROM SC
WHERE SNO=:GIVENSNO AND
CNO=:GIVENCNO;
```

2.6.2. Cursor（查询结果不仅一组tuple时）

- 定义Cursor
  - EXEC SQL DECLARE <cursor name> CURSOR FOR SELECT … FROM … WHERE …
- EXEC SQL OPEN <cursor name>： 类似打开文件
- 利用Cursor 获取数据
  - EXEC SQL FETCH <cursor name> INTO :hostvar1, :hostvar2, …;
- 当达到末端时，SQLCA.SQLCODE 为 100
- EXEC SQL CLOSE CURSOR <cursor name>

```
EXEC SQL DECLARE C1 CURSOR FOR
SELECT SNO, GRADE
FROM SC
WHERE CNO = :GIVENCNO;
EXEC SQL OPEN C1;
if (SQLCA.SQLCODE<0)
    exit(1); /* There is error in query*/
while (1) {
    EXEC SQL FETCH C1 INTO :SNO, :GRADE :GRADEI
    if (SQLCA.SQLCODE==100) break;
    /* treat data fetched from cursor, omitted*/
}
EXEC SQL CLOSE C1;
```

2.7. 动态SQL Dynamic SQL

- 在某些应用程序中，SQL语句在执行程序之前无法确定，它们需要在程序运行时动态构建

2.7.1. 直接执行动态SQL Dynamic SQL executed directly

```
// 仅用于执行非查询SQL语句
EXEC SQL BEGIN DECLARE SECTION;
char sqlstring[200];
EXEC SQL END DECLARE SECTION;
char cond[150];
strcpy( sqlstring, "DELETE FROM STUDENT WHERE");
printf("Enter search condition :");
scanf("%s", cond);
strcat(sqlstring, cond);
EXEC SQL EXECUTE IMMEDIATE :sqlstring;
```

2.7.2. 动态SQL语句中的动态参数 Dynamic SQL with dynamic parameters

- 仅在执行非查询SQL语句时使用。
- 使用占位符在SQL语句中实现动态参数，类似于C语言中的宏处理方法。

```
EXEC SQL BEGIN DECLARE SECTION;
char sqlstring[200];
int birth_year;
EXEC SQL END DECLARE SECTION;
strcpy(sqlstring, "DELETE FROM STUDENT WHERE YEAR(BDATE) <= :y;");
printf("Enter birth year to delete :");
scanf("%d", &birth_year);
EXEC SQL PREPARE purge FROM :sqlstring;
EXEC SQL EXECUTE purge USING :birth_year;
```

2.7.3. 动态SQL查询语句 Dynamic SQL for query

- 用于动态地构造查询语句

```
EXEC SQL BEGIN DECLARE SECTION;
char sqlstring[200];
char SNO[7];
float GRADE;
short GRADEI;
char GIVENCNO[6];
EXEC SQL END DECLARE SECTION;

char orderby[150];
strcpy( sqlstring, ”SELECT SNO,GRADE FROM SC WHERE CNO= :c ”);
printf(“ Enter the ORDER BY clause :”);
scanf("%s", orderby);
strcat( sqlstring, orderby);
printf("Enter the course number :");
scanf("%s", GIVENCNO);
EXEC SQL PREPARE query FROM :sqlstring;
EXEC SQL DECLARE grade_cursor CURSOR FOR query; // 16行query

EXEC SQL OPEN grade_cursor USING :GIVENCNO;
if (SQLCA.SQLCODE<0) exit(1); /* There is error in query*/
while (1) {
    EXEC SQL FETCH grade_cursor INTO :SNO, :GRADE :GRADEI
    if (SQLCA.SQLCODE==100) break;
    /* treat data fetched from cursor, omitted*/
}
EXEC SQL CLOSE grade_cursor;
```

2.8. 存储过程 Stored Procedure

- 用于提高性能和方便用户使用

- 使用存储过程，用户可以将常用的数据库访问程序作为存储过程，编译后存储在数据库中，然后在需要时直接调用
  - 方便用户：用户可以直接调用它们而无需再次编写代码，是可重用的
  - 提高性能：存储过程已经编译过，因此在使用时不需要再次解析和查询优化
  - 扩展DBMS的功能（可以编写脚本）

```
EXEC SQL
    CREATE PROCEDURE drop_student
        (IN student_no CHAR(7),
        OUT message CHAR(30))
    BEGIN ATOMIC
        DELETE FROM STUDENT
            WHERE SNO=student_no;
        DELETE FROM SC
            WHERE SNO=student_no;
    SET message=student_no || 'droped';
END;
EXEC SQL
 /* call this stored procedure later*/
CALL drop_student(…);
```

2.9. 从编程语言访问SQL

- 数据库程序员至少有以下几个原因需要使用通用的编程语言：
  - 并非所有查询都可以用SQL表达，因为SQL不具备通用语言的全功能表达能力
  - 非声明性操作（如打印报告、与用户交互或将查询结果发送到图形用户界面）不能在SQL内部完成
- 有两种方法可以从通用的编程语言访问SQL
  - 通用的程序（general-purpose program）：可以使用一组函数连接到并和数据库服务器通信
  - 嵌入式SQL：提供了一种程序可以与数据库服务器交互的方式
    - SQL语句在编译时被翻译成函数调用
    - 在运行时，这些函数调用使用提供动态SQL功能的API连接到数据库

2.9.1. JDBC

```
public static void JDBCexample(String dbid, String userid, String passwd)
{
    try (Connection conn =
            DriverManager.getConnection(
            "jdbc:oracle:thin:@db.yale.edu:2000:univdb", userid, passwd);
        Statement stmt = conn.createStatement();)
    { //… Do Actual Work ….}
    catch (SQLException sqle) {
        System.out.println("SQLException : " + sqle);
    }
}
// Update to database
try{
    stmt.executeUpdate
        ("insert into instructor values('77987', 'Kim', 'Physics', 98
0)");
}
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));
}
```

2.9.2. SQLJ

- JDBC过于动态，编译器无法捕获错误
- SQLJ：Java中的嵌入式SQL

```
#sql iterator deptInfoIter( String dept name, int avgSal);
deptInfoIter iter = null;
#sql iter = { select dept_name, avg(salary) from instructor group by dept n
ame };
while (iter.next()) {
    String deptName = iter.dept_name();
    int avgSal = iter.avgSal();
    System.out.println(deptName + " " + avgSal);
}
iter.close();
```

2.9.3. ODBC 开放数据库连接标准

- 应用程序与数据库服务器通信的标准
- 应用程序编程接口（API）用于：
    - 与数据库建立连接
    - 发送查询和更新
    - 获取结果

GUI、电子表格等应用程序可以使用ODBC