**ITM 422 Review --Final**

1. **PL/SQL Summarization:**

**PL/SQL块的结构如下：**

**DECLARE    
    --声明部分: 在此声明PL/SQL用到的变量,类型及游标，以及局部的存储过程和函数  
  BEGIN  
    -- 执行部分:  过程及SQL 语句  , 即程序的主要部分  
  EXCEPTION  
    -- 执行异常部分: 错误处理  
  END;**

PL/SQL程序设计中的标识符定义与SQL 的标识符定义的要求相同。要求和限制有：

       标识符名不能超过30字符；

       第一个字符必须为字母；

       不分大小写；

       不能用’-‘(减号);

       不能是SQL保留字。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 子类 | 说     明 | 范   围 | ORACLE限制 |
| CHAR | Character  String  Rowid  Nchar | 定长字符串      民族语言字符集 | 032767  可选,确省=1 | 2000 |
| VARCHAR2 | Varchar, String  NVARCHAR2 | 可变字符串  民族语言字符集 | 032767  4000 | 4000 |
| BINARY\_INTEGER |  | 带符号整数,为整数计算优化性能 |  |  |
| NUMBER(p,s) | Dec    Double precision  Integer  Int  Numeric  Real  Small int | 小数, NUMBER 的子类型  高精度实数  整数, NUMBER 的子类型  整数, NUMBER 的子类型  与NUMBER等价  与NUMBER等价  整数, 比 integer 小 |  |  |
| LONG |  | 变长字符串 | 0->2147483647 | 32,767字节 |
| DATE |  | 日期型 | 公元前4712年1月1日至公元后4712年12月31日 |  |
| BOOLEAN |  | 布尔型 | TRUE, FALSE,NULL | 不使用 |
| ROWID |  | 存放数据库行号 |  |  |
| UROWID |  | 通用行标识符，字符类型 |  |  |

数据是具有相同数据类型的一组成员的集合。每个成员都有一个唯一的下标，它取决于成员在数组中的位置。在PL/SQL中，数组数据类型是VARRAY。

DECLARE  
   TYPE reg\_varray\_type IS VARRAY(**5**) OF VARCHAR(**25**);  
   v\_reg\_varray REG\_VARRAY\_TYPE;  
  
BEGIN

   v\_reg\_varray := reg\_varray\_type  
         ('中国', '美国', '英国', '日本', '法国');  
  
   DBMS\_OUTPUT.PUT\_LINE('地区名称：'||v\_reg\_varray(**1**)||'、'  
                                    ||v\_reg\_varray(**2**)||'、'  
                                    ||v\_reg\_varray(**3**)||'、'  
                                    ||v\_reg\_varray(**4**));  
   DBMS\_OUTPUT.PUT\_LINE('赋予初值NULL的第5个成员的值：'||v\_reg\_varray(**5**));  
  
   v\_reg\_varray(**5**) := '法国';  
   DBMS\_OUTPUT.PUT\_LINE('第5个成员的值：'||v\_reg\_varray(**5**));  
END;

|  |  |
| --- | --- |
| 方法 | 描述 |
| EXISTS(n) | 如果集合的第n个成员存在，则返回true |
| COUNT | 返回已经分配了存储空间即赋值了的成员数量 |
| FIRST  LAST | FIRST：返回成员的最低下标值  LAST： 返回成员的最高下标值 |
| PRIOR(n) | 返回下标为n的成员的前一个成员的下标。如果没有则返回NULL |
| NEXT(N) | 返回下标为n的成员的后一个成员的下标。如果没有则返回NULL |
| TRIM | TRIM：删除末尾一个成员  TRIM(n) ：删除末尾n个成员 |
| DELETE | DELETE：删除所有成员  DELETE(n) ：删除第n个成员  DELETE(m, n) ：删除从n到m的成员 |
| EXTEND | EXTEND：添加一个null成员  EXTEND(n)：添加n个null成员  EXTEND(n,i)：添加n个成员，其值与第i个成员相同 |
| LIMIT | 返回在varray类型变量中出现的最高下标值 |

  TYPE dept\_table\_type IS TABLE OF   
       dept%ROWTYPE INDEX BY BINARY\_INTEGER;  
  my\_dname\_table dept\_table\_type;  
  v\_count number(**2**) :=**4**;  
BEGIN  
  FOR int IN **1** .. v\_count LOOP  
    SELECT \* INTO my\_dname\_table(int) FROM dept WHERE deptno=int\***10**;  
  END LOOP;  
  FOR int IN my\_dname\_table.FIRST .. my\_dname\_table.LAST LOOP  
  DBMS\_OUTPUT.PUT\_LINE('Department number: '||my\_dname\_table(int).deptno);  
  DBMS\_OUTPUT.PUT\_LINE('Department name: '|| my\_dname\_table(int).dname);  
  END LOOP;  
END;

   variable  := expression ;

空值加数字仍是空值：NULL + < 数字> = NULL

空值加（连接）字符，结果为字符：NULL || <字符串> = < 字符串>

**DECLARE  
  emp\_id    emp.empno%TYPE :=7788;  
  emp\_name  emp.ename%TYPE;  
  wages     emp.sal%TYPE;  
BEGIN  
  SELECT ename, NVL(sal,0) + NVL(comm,0) INTO emp\_name, wages   
  FROM emp WHERE empno = emp\_id;  
  DBMS\_OUTPUT.PUT\_LINE(emp\_name||'----'||to\_char(wages));  
END;**

**PL/SQL流程控制语句**

**包括如下三类:**

控制语句: IF 语句

循环语句: LOOP语句, EXIT语句

顺序语句: GOTO语句, NULL语句

DECLARE  
    v\_empno  employees.employee\_id%TYPE :=&empno;  
    V\_salary employees.salary%TYPE;  
    V\_comment VARCHAR2(**35**);  
BEGIN  
   SELECT salary INTO v\_salary FROM employees   
   WHERE employee\_id = v\_empno;  
   IF v\_salary < **1500** THEN  
       V\_comment:= '太少了,加点吧~!';  
   ELSIF v\_salary <**3000** THEN  
      V\_comment:= '多了点,少点吧~!';  
   ELSE  
      V\_comment:= '没有薪水~!';  
   END IF;  
   DBMS\_OUTPUT.PUT\_LINE(V\_comment);  
   exception  
     when no\_data\_found then  
        DBMS\_OUTPUT.PUT\_LINE('没有数据~!');  
     when others then  
        DBMS\_OUTPUT.PUT\_LINE(sqlcode || '---' || sqlerrm);          
END;

**例4:**

DECLARE  
  V\_grade char(**1**) := UPPER('&p\_grade');  
  V\_appraisal VARCHAR2(**20**);  
BEGIN  
  V\_appraisal :=  
  CASE v\_grade  
    WHEN 'A' THEN 'Excellent'  
    WHEN 'B' THEN 'Very Good'  
    WHEN 'C' THEN 'Good'  
    ELSE 'No such grade'  
  END;  
  DBMS\_OUTPUT.PUT\_LINE('Grade:'||v\_grade||'  Appraisal: '|| v\_appraisal);  
END;

**例5:**

DECLARE  
   v\_first\_name employees.first\_name%TYPE;  
   v\_job\_id employees.job\_id%TYPE;  
   v\_salary employees.salary%TYPE;  
   v\_sal\_raise NUMBER(**3**,**2**);  
BEGIN  
   SELECT first\_name,   job\_id,   salary INTO  
          v\_first\_name, v\_job\_id, v\_salary  
   FROM employees WHERE employee\_id = &emp\_id;  
   CASE  
      WHEN v\_job\_id = 'PU\_CLERK' THEN  
         IF v\_salary < **3000** THEN v\_sal\_raise := .**08**;  
         ELSE v\_sal\_raise := .**07**;  
         END IF;  
      WHEN v\_job\_id = 'SH\_CLERK' THEN  
         IF v\_salary < **4000** THEN v\_sal\_raise := .**06**;  
         ELSE v\_sal\_raise := .**05**;  
         END IF;  
      WHEN v\_job\_id = 'ST\_CLERK' THEN  
         IF v\_salary < **3500** THEN v\_sal\_raise := .**04**;  
         ELSE v\_sal\_raise := .**03**;  
         END IF;  
      ELSE  
         DBMS\_OUTPUT.PUT\_LINE('该岗位不涨工资: '||v\_job\_id);  
   END CASE;  
   DBMS\_OUTPUT.PUT\_LINE(v\_first\_name||'的岗位是'||v\_job\_id  
                                    ||'、的工资是'||v\_salary  
                                    ||'、工资涨幅是'||v\_sal\_raise);  
END;

**例 6.**

DECLARE  
    int NUMBER(**2**) :=**0**;  
BEGIN  
   LOOP  
      int := int + **1**;  
      DBMS\_OUTPUT.PUT\_LINE('int 的当前值为:'||int);  
      EXIT WHEN int =**10**;  
   END LOOP;  
END;

**例7.**

DECLARE   
  x NUMBER :=**1**;  
BEGIN  
   WHILE x<=**10** LOOP  
      DBMS\_OUTPUT.PUT\_LINE('X的当前值为:'||x);  
       x:= x+**1**;  
   END LOOP;  
END;

**例8.**

BEGIN  
   FOR int  in **1**..**10** LOOP  
       DBMS\_OUTPUT.PUT\_LINE('int 的当前值为: '||int);  
   END LOOP;  
END;

PL/SQL中GOTO语句是无条件跳转到指定的标号去的意思。语法如下：

GOTO label;  
......  
<<label>>　/\*标号是用<<　>>括起来的标识符 \*/

**注意**，在以下地方使用是不合法的，编译时会出错误。

跳转到非执行语句前面。

跳转到子块中。

跳转到循环语句中。

跳转到条件语句中。

从异常处理部分跳转到执行。

从条件语句的一部分跳转到另一部分。

**例12:**

DECLARE  
   V\_counter NUMBER := **1**;  
BEGIN  
   LOOP   
     DBMS\_OUTPUT.PUT\_LINE('V\_counter的当前值为:'||V\_counter);  
     V\_counter := v\_counter + **1**;  
   IF v\_counter > **10** THEN  
       GOTO labelOffLOOP;  
   END IF;  
   END LOOP;  
   <<labelOffLOOP>>  
     DBMS\_OUTPUT.PUT\_LINE('V\_counter的当前值为:'||V\_counter);  
END;

在PL/SQL 程序中，NULL语句是一个可执行语句，可以用 null 语句来说明“不用做任何事情”的意思，相当于一个占位符或不执行任何操作的空语句，可以使某些语句变得有意义，提高程序的可读性，保证其他语句结构的完整性和正确性。

**游标的使用**

|  |  |
| --- | --- |
| SQL语句 | 游标 |
| 非查询语句 | 隐式的 |
| 结果是单行的查询语句 | 隐式的或显示的 |
| 结果是多行的查询语句 | 显示的 |

PL/SQL 程序不能用OPEN 语句重复打开一个游标。

**1. 显式游标处理**

**定义/声明游标：**

    CURSOR cursor\_name[(parameter[, parameter]…)]   
           [RETURN datatype]  
    IS   
        select\_statement;

**打开游标：**

OPEN cursor\_name[([parameter =>] value[, [parameter =>] value]…)];

**提取游标数据：**就是检索结果集合中的数据行，放入指定的输出变量中。

格式：

FETCH cursor\_name INTO {variable\_list | record\_variable };

**关闭游标：**

CLOSE cursor\_name;

**例1.**查询前10名员工的信息。

DECLARE  
   CURSOR c\_cursor   
   IS SELECT first\_name || last\_name, Salary   
   FROM EMPLOYEES   
   WHERE rownum<**11**;     
   v\_ename  EMPLOYEES.first\_name%TYPE;  
   v\_sal    EMPLOYEES.Salary%TYPE;     
BEGIN  
  OPEN c\_cursor;  
  FETCH c\_cursor INTO v\_ename, v\_sal;  
  WHILE c\_cursor%FOUND LOOP  
     DBMS\_OUTPUT.PUT\_LINE(v\_ename||'---'||to\_char(v\_sal) );  
     FETCH c\_cursor INTO v\_ename, v\_sal;  
  END LOOP;  
  CLOSE c\_cursor;  
END;

Cursor\_name%FOUND     布尔型属性，当最近一次提取游标操作FETCH成功则为 TRUE,否则为FALSE；

 Cursor\_name%NOTFOUND   布尔型属性，与%FOUND相反；

 Cursor\_name%ISOPEN     布尔型属性，当游标已打开时返回 TRUE；

 Cursor\_name%ROWCOUNT   数字型属性，返回已从游标中读取的记录数。

显式游标主要是用于对查询语句的处理，尤其是在查询结果为多条记录的情况下；而对于非查询语句，如修改、删除操作，则由ORACLE 系统自动地为这些操作设置游标并创建其工作区，这些由系统隐含创建的游标称为隐式游标，隐式游标的名字为SQL，这是由ORACLE 系统定义的。对于隐式游标的操作，如定义、打开、取值及关闭操作，都由ORACLE 系统自动地完成，无需用户进行处理。用户只能通过隐式游标的相关属性，来完成相应的操作。在隐式游标的工作区中，所存放的数据是与用户自定义的显示游标无关的、最新处理的一条SQL 语句所包含的数据。

格式调用为： SQL%

**注：INSERT, UPDATE, DELETE, SELECT 语句中不必明确定义游标。**

**隐式游标属性**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **属性** | **值** | **SELECT** | **INSERT** | **UPDATE** | **DELETE** |
| SQL%ISOPEN |  | FALSE | FALSE | FALSE | FALSE |
| SQL%FOUND | TRUE | 有结果 |  | 成功 | 成功 |
| SQL%FOUND | FALSE | 没结果 |  | 失败 | 失败 |
| SQL%NOTFUOND | TRUE | 没结果 |  | 失败 | 失败 |
| SQL%NOTFOUND | FALSE | 有结果 |  | 成功 | 失败 |
| SQL%ROWCOUNT |  | 返回行数，只为1 | 插入的行数 | 修改的行数 | 删除的行数 |

**例12:**通过隐式游标SQL的%ROWCOUNT属性来了解修改了多少行。

DECLARE  
   v\_rows NUMBER;  
BEGIN  
   UPDATE employees SET salary = **30000**  
   WHERE department\_id = **90** AND job\_id = 'AD\_VP';  
   v\_rows := SQL%ROWCOUNT;  
   DBMS\_OUTPUT.PUT\_LINE('更新了'||v\_rows||'个雇员的工资');  
   ROLLBACK;  
END;

SELECT … INTO 语句触发 NO\_DATA\_FOUND；

当一个显式游标的WHERE子句未找到时触发%NOTFOUND；

当UPDATE或DELETE 语句的WHERE 子句未找到时触发 SQL%NOTFOUND；在提取循环中要用 %NOTFOUND 或%FOUND 来确定循环的退出条件，不要用 NO\_DATA\_FOUND.

为了对正在处理(查询)的行不被另外的用户改动，ORACLE 提供一个 FOR UPDATE 子句来对所选择的行进行锁住。该需求迫使ORACLE锁定游标结果集合的行，可以防止其他事务处理更新或删除相同的行，直到您的事务处理提交或回退为止。

**语法：**

 SELECT column\_list FROM table\_list FOR UPDATE [OF column[, column]…] [NOWAIT]

    如果另一个会话已对活动集中的行加了锁，那么SELECT FOR UPDATE操作一直等待到其它的会话释放这些锁后才继续自己的操作，对于这种情况，当加上**NOWAIT**子句时，如果这些行真的被另一个会话锁定，则OPEN立即返回并给出：

ORA-0054 ：resource busy  and  acquire with nowait specified.

如果使用 FOR UPDATE 声明游标，则可在DELETE和UPDATE 语句中使用

WHERE CURRENT OF cursor\_name子句，修改或删除游标结果集合当前行对应的数据库表中的数据行.

         IN,OUT,IN OUT是形参的模式。若省略，则为IN模式。IN模式的形参只能将实参传递给形参，进入函数内部，但只能读不能写，函数返回时实参的值不变。OUT模式的形参会忽略调用时的实参值（或说该形参的初始值总是NULL），但在函数内部可以被读或写，函数返回时形参的值会赋予给实参。IN OUT具有前两种模式的特性，即调用时，实参的值总是传递给形参，结束时，形参的值传递给实参。调用时，对于IN模式的实参可以是常量或变量，但对于OUT和IN OUT模式的实参必须是变量。

**例7．**删除指定员工记录；

CREATE OR REPLACE  
PROCEDURE DelEmp  
(v\_empno IN employees.employee\_id%TYPE)   
AS  
No\_result EXCEPTION;  
BEGIN  
   DELETE FROM employees WHERE employee\_id = v\_empno;  
   IF SQL%NOTFOUND THEN  
      RAISE no\_result;  
   END IF;  
   DBMS\_OUTPUT.PUT\_LINE('编码为'||v\_empno||'的员工已被删除!');  
EXCEPTION  
   WHEN no\_result THEN   
      DBMS\_OUTPUT.PUT\_LINE('温馨提示:你需要的数据不存在!');  
   WHEN OTHERS THEN  
      DBMS\_OUTPUT.PUT\_LINE(SQLCODE||'---'||SQLERRM);  
END DelEmp;

## 1 触发器类型

触发器组成:

        触发事件：引起触发器被触发的事件。 例如：DML语句(INSERT, UPDATE, DELETE语句对表或视图执行数据处理操作)、DDL语句（如CREATE、ALTER、DROP语句在数据库中创建、修改、删除模式对象）、数据库系统事件（如系统启动或退出、异常错误）、用户事件（如登录或退出数据库）。

        触发时间：即该TRIGGER 是在触发事件发生之前（BEFORE）还是之后(AFTER)触发，也就是触发事件和该TRIGGER的操作顺序。

        触发操作：即该TRIGGER 被触发之后的目的和意图，正是触发器本身要做的事情。 例如：PL/SQL块。

        触发对象：包括表、视图、模式、数据库。只有在这些对象上发生了符合触发条件的触发事件，才会执行触发操作。

        触发条件：由WHEN子句指定一个逻辑表达式。只有当该表达式的值为TRUE时，遇到触发事件才会自动执行触发器，使其执行触发操作。

        触发频率：说明触发器内定义的动作被执行的次数。即语句级(STATEMENT)触发器和行级(ROW)触发器。

语句级(STATEMENT)触发器：是指当某触发事件发生时，该触发器只执行一次；

行级(ROW)触发器：是指当某触发事件发生时，对受到该操作影响的每一行数据，触发器都单独执行一次。

**编写触发器时，需要注意以下几点：**

        触发器不接受参数。

        一个表上最多可有12个触发器，但同一时间、同一事件、同一类型的触发器只能有一个。并各触发器之间不能有矛盾。

        在一个表上的触发器越多，对在该表上的DML操作的性能影响就越大。

        触发器最大为32KB。若确实需要，可以先建立过程，然后在触发器中用CALL语句进行调用。

        **在触发器的执行部分只能用DML语句（SELECT、INSERT、UPDATE、DELETE），不能使用DDL语句（CREATE、ALTER、DROP）**。

        触发器中不能包含事务控制语句(COMMIT，ROLLBACK，SAVEPOINT)。因为触发器是触发语句的一部分，触发语句被提交、回退时，触发器也被提交、回退了。

        在触发器主体中调用的任何过程、函数，都不能使用事务控制语句。

        在触发器主体中不能申明任何Long和blob变量。新值new和旧值old也不能向表中的任何long和blob列。

**2 创建触发器**

**创建触发器的一般语法是:**

CREATE [OR REPLACE] TRIGGER trigger\_name  
{BEFORE | AFTER }  
{INSERT | DELETE | UPDATE [OF column [, column …]]}  
[OR {INSERT | DELETE | UPDATE [OF column [, column …]]}...]  
ON [schema.]table\_name | [schema.]view\_name   
[REFERENCING {OLD [AS] old | NEW [AS] new| PARENT as parent}]  
[FOR EACH ROW ]  
[WHEN condition]  
PL/SQL\_BLOCK | CALL procedure\_name;

其中：

BEFORE 和AFTER指出触发器的触发时序分别为前触发和后触发方式，前触发是在执行触发事件之前触发当前所创建的触发器，后触发是在执行触发事件之后触发当前所创建的触发器。

       FOR EACH ROW选项说明触发器为行触发器。行触发器和语句触发器的区别表现在：行触发器要求当一个DML语句操走影响数据库中的多行数据时，对于其中的每个数据行，只要它们符合触发约束条件，均激活一次触发器；而语句触发器将整个语句操作作为触发事件，当它符合约束条件时，激活一次触发器。当省略FOR EACH ROW 选项时，BEFORE 和AFTER 触发器为语句触发器，而INSTEAD OF 触发器则只能为行触发器。

            REFERENCING 子句说明相关名称，在行触发器的PL/SQL块和WHEN 子句中可以使用相关名称参照当前的新、旧列值，默认的相关名称分别为OLD和NEW。触发器的PL/SQL块中应用相关名称时，必须在它们之前加冒号(:)，但在WHEN子句中则不能加冒号。

WHEN 子句说明触发约束条件。Condition 为一个逻辑表达时，其中必须包含相关名称，而不能包含查询语句，也不能调用PL/SQL 函数。WHEN 子句指定的触发约束条件只能用在BEFORE 和AFTER 行触发器中，不能用在INSTEAD OF 行触发器和其它类型的触发器中。

    当一个基表被修改( INSERT, UPDATE, DELETE)时要执行的存储过程，执行时根据其所依附的基表改动而自动触发，因此与应用程序无关，用数据库触发器可以保证数据的一致性和完整性。

### 2.1 触发器触发次序

1.        执行 BEFORE语句级触发器;

2.        对与受语句影响的每一行：

         执行 BEFORE行级触发器

         执行 DML语句

         执行 AFTER行级触发器

3.        执行 AFTER语句级触发器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 特性 | | INSERT | UPDATE | DELETE |
| OLD | | NULL | 实际值 | 实际值 |
| NEW | | 实际值 | 实际值 | NULL |
| **事件** | | **允许的时机** | | **说明** | | |
| STARTUP | | AFTER | | 启动数据库实例之后触发 | | |
| SHUTDOWN | | BEFORE | | 关闭数据库实例之前触发（非正常关闭不触发） | | |
| SERVERERROR | | AFTER | | 数据库服务器发生错误之后触发 | | |
| LOGON | | AFTER | | 成功登录连接到数据库后触发 | | |
| LOGOFF | | BEFORE | | 开始断开数据库连接之前触发 | | |
| CREATE | | BEFORE，AFTER | | 在执行CREATE语句创建数据库对象之前、之后触发 | | |
| DROP | | BEFORE，AFTER | | 在执行DROP语句删除数据库对象之前、之后触发 | | |
| ALTER | | BEFORE，AFTER | | 在执行ALTER语句更新数据库对象之前、之后触发 | | |
| DDL | | BEFORE，AFTER | | 在执行大多数DDL语句之前、之后触发 | | |
| GRANT | | BEFORE，AFTER | | 执行GRANT语句授予权限之前、之后触发 | | |
| REVOKE | | BEFORE，AFTER | | 执行REVOKE语句收权限之前、之后触犯发 | | |
| RENAME | | BEFORE，AFTER | | 执行RENAME语句更改数据库对象名称之前、之后触犯发 | | |
| AUDIT **/**NOAUDIT | | BEFORE，AFTER | | 执行AUDIT**或**NOAUDIT进行审计或停止审计之前、之后触发 | | |

### 2.4 系统触发器事件属性

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 事件属性\事件 | Startup/Shutdown | Servererror | Logon/Logoff | DDL | DML |
| 事件名称 | **⎫＊** | **⎫＊** | **⎫＊** | **⎫＊** | **＊** |
| 数据库名称 | **⎫＊** |  |  |  |  |
| 数据库实例号 | **⎫＊** |  |  |  |  |
| 错误号 |  | **⎫＊** |  |  |  |
| 用户名 |  |  | **⎫＊** | **＊** |  |
| 模式对象类型 |  |  |  | **⎫＊** | **＊** |
| 模式对象名称 |  |  |  | **⎫＊** | **＊** |
| 列 |  |  |  |  | **⎫＊** |

 除DML语句的列属性外，其余事件属性值可通过调用ORACLE定义的事件属性函数来读取。

|  |  |  |
| --- | --- | --- |
| 函数名称 | 数据类型 | 说    明 |
| Ora\_sysevent | VARCHAR2（20） | 激活触发器的事件名称 |
| Instance\_num | NUMBER | 数据库实例名 |
| Ora\_database\_name | VARCHAR2（50） | 数据库名称 |
| Server\_error(posi) | NUMBER | 错误信息栈中posi指定位置中的错误号 |
| Is\_servererror(err\_number) | BOOLEAN | 检查err\_number指定的错误号是否在错误信息栈中，如果在则返回TRUE，否则返回FALSE。在触发器内调用此函数可以判断是否发生指定的错误。 |
| Login\_user | VARCHAR2(30) | 登陆或注销的用户名称 |
| Dictionary\_obj\_type | VARCHAR2(20) | DDL语句所操作的数据库对象类型 |
| Dictionary\_obj\_name | VARCHAR2(30) | DDL语句所操作的数据库对象名称 |
| Dictionary\_obj\_owner | VARCHAR2(30) | DDL语句所操作的数据库对象所有者名称 |
| Des\_encrypted\_password | VARCHAR2(2) | 正在创建或修改的经过DES算法加密的用户口令 |

1. Normalization

1NF 的要求: 1. 各行没有顺序关系; 2. 各列也没有顺序关系; 3. 没有重复的行; 4. 没有NULL 的列. （实际要求：1. 有unique key; 2. 没有null）注: 不可分割的原子性, 没有重复,不得为空.

Unique key 🡪 super key 🡪 minimal super key: candidate key 🡪 primary key

2NF的要求: 每一个non-prime attribute 依赖于candidate key 的整个集合, 而非一部分. 注: 全都 与主键有关系

3NF 的要求: 所有的non-prime attributes 不能传递地依赖于candidate key. 注: 消除传递依赖, 不直接依赖于主键

1. *PL\SQL introduction*

Block is the basic unit

SQL statements only use data manipulation statement and some transaction control. (DML and TCL )

Assign a value, using “:=”

Identifier: start a letter, then can use anything.

%Rowtype inherits all columns’ datatype

%type inherits one column’s datatype

* **IF** <condition> **THEN** <statement\_list>
* **ELSIF**<condition> **THEN** <statement\_list>
* **ELSE** <statement\_list> **END IF**;

possible modes are IN (read-only), OUT (write-only), and INOUT (read and write).

Exception use when clause

1. *Trigger*

A trigger is either **a stored PL/SQL block** or a PL/SQL, **C**, or **Java** procedure associated with a table, view, schema, or the database itself

:old.column\_name; :new.column.name

CREATE OR REPLACE TRIGGER Print\_salary\_changes

BEFORE DELETE OR INSERT OR UPDATE ON Emp\_tab

FOR EACH ROW

WHEN (new.Empno > 0)

DECLARE

sal\_diff number;

BEGIN

sal\_diff := :new.sal - :old.sal;

dbms\_output.put('Old salary: ' || :old.sal);

dbms\_output.put(' New salary: ' || :new.sal);

dbms\_output.put\_line(' Difference ' || sal\_diff);

END;

1. View

**CREATE** VIEW **sales\_staff**

AS **SELECT** empno, ename, deptno

FROM emp WHERE deptno = 10   
WITH CHECK OPTION CONSTRAINT **sales\_staff\_cnst**;

* the **CHECK OPTION** creates the view with the constraint (named **sales\_staff\_cnst**) that INSERT and UPDATE statements issued against the view cannot result in rows that the view cannot select. FOR EXAMPLE
  + INSERT INTO **sales\_staff** VALUES (7584, 'OSTER', 10); **OKAY**
  + INSERT INTO **sales\_staff** VALUES (7591, 'WILLIAMS', 30); **ERROR**

你可以drop and replace view, 不能alter view.

* Before replacing a view, consider the following effects:
* Replacing a view 只是取代了data dictionary中的定义. 表不受这个影响.
* 之前有的 constraint在取代后的VIEW中自动取消.
* VIEW被取代之后, 老的旧VIEW被引用出来的别的VIEW自动没用了.
* Restrictions on DML operations for views:
  + If a view is defined by a query that contains SET or DISTINCT operators, a GROUP BY clause, or a group function, then rows cannot be inserted into, updated in, or deleted from the base tables using the view指这个VIEW用了任何功能键做的选择
  + If a view is defined with WITH CHECK OPTION, a row cannot be inserted into, or updated in, the base table (using the view), if the view cannot select the row from the base table.指加了CHECK条件的VIEW,如果插入的值因为CHECK原因就插入不了,那自然插入不了表中.
  + If a NOT NULL column that does not have a DEFAULT clause is omitted from the view, then a row cannot be inserted into the base table using the view.如果有NULL值却没有用DEFAULT值,插入不了VALUE到ＶＩＥＷ中．
* **A table is key-preserved** if every key of the table can also be a key of the result of the join. So, a key-preserved table has its keys preserved through a join.

ＪＯＩＮ VIEW中, 如果不是key-preserved table, 则无法更新这个表中的某些COLUMN, 也无法删除这个表中的某些行.

* + **ALTER VIEW** hr.test\_view
  + **ADD** **CONSTRAINT** pk\_test\_view
  + PRIMARY KEY (C1) DISABLE NOVALIDATE;

1. Sequence

CREATE SEQUENCE hr.employee\_identity START WITH 2055;

**START WITH**: Defines the first number that the sequence will generate. The default is MAXVALUE for descending sequences, which is −1, and MINVALUE for ascending sequences, which is 1.  
  
**INCREMENT BY**: Defines the increase or decrease amount for subsequently generated numbers. To specify a decreasing sequence, use a negative INCREMENT BY value. The default is 1.  
  
**MINVALUE**: Defines the lowest number the sequence will generate. This is the bounding value in a decreasing sequence. The default MINVALUE is NOMINVALUE, which evaluates to 1 for an increasing sequence and to −1026 for a decreasing sequence.  
  
**MAXVALUE**: Defines the largest number that the sequence will generate. This is the bounding value in the default, increasing sequence. The default MAXVALUE is the NOMAXVALUE, which evaluates to 1027 for an increasing sequence and to −1 for a decreasing sequence.  
  
**CYCLE**: Configures the sequence to repeat numbers after reaching the bounding value.  
  
**NOCYCLE**: Configures the sequence to not repeat numbers after reaching the bounding value. This is the default. When you try to generate MAXVALUE+1, an exception will be raised.  
  
**CACHE**: Defines the size of the block of sequence numbers held in memory. The default is 20.  
  
**NOCACHE**: Forces the data dictionary to be updated for each sequence number generated, guaranteeing no gaps in the generated numbers but decreasing the performance of the sequence.

* sequence\_name.**nextval**;
* sequence\_name.**currval**;
* Altering sequneces : A common problem with sequences is how to go about altering them to change the NEXTVAL. You cannot simply alter the sequence and set the NEXTVAL
* Maximum and Minimum Values: issues when NOCYCLE
* Missing Sequence Values: problem in the use of sequences involves “losing” sequence values when a rollback occurs

1. Synonyms

CREATE PUBLIC SYNONYM employees FOR hr.employees;

DROP PUBLIC SYNONYM employees;

CREATE PUBLIC DATABASE LINK **new\_jersey**

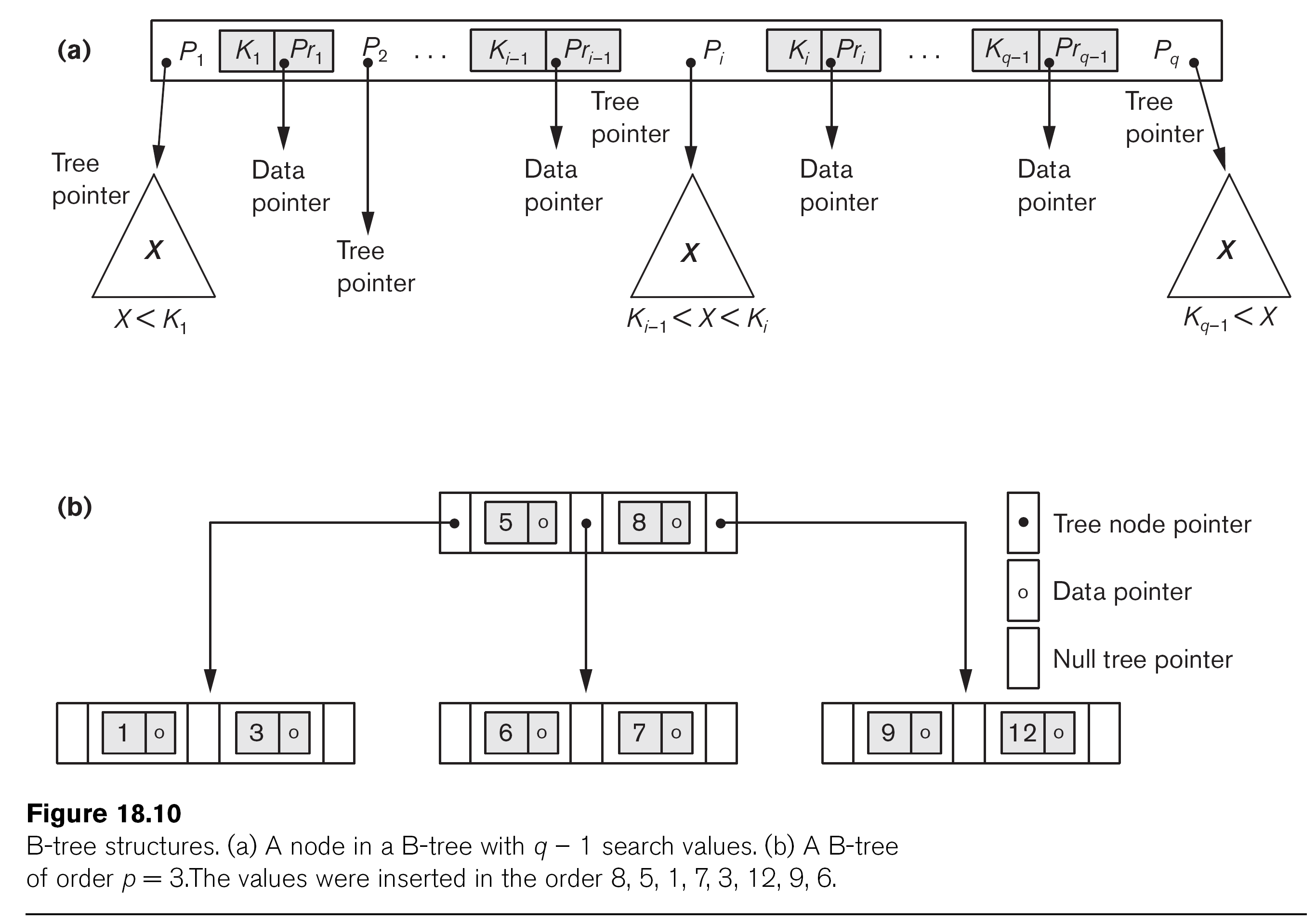
CONNECT TO home\_office IDENTIFIED BY secret USING 'NJ';

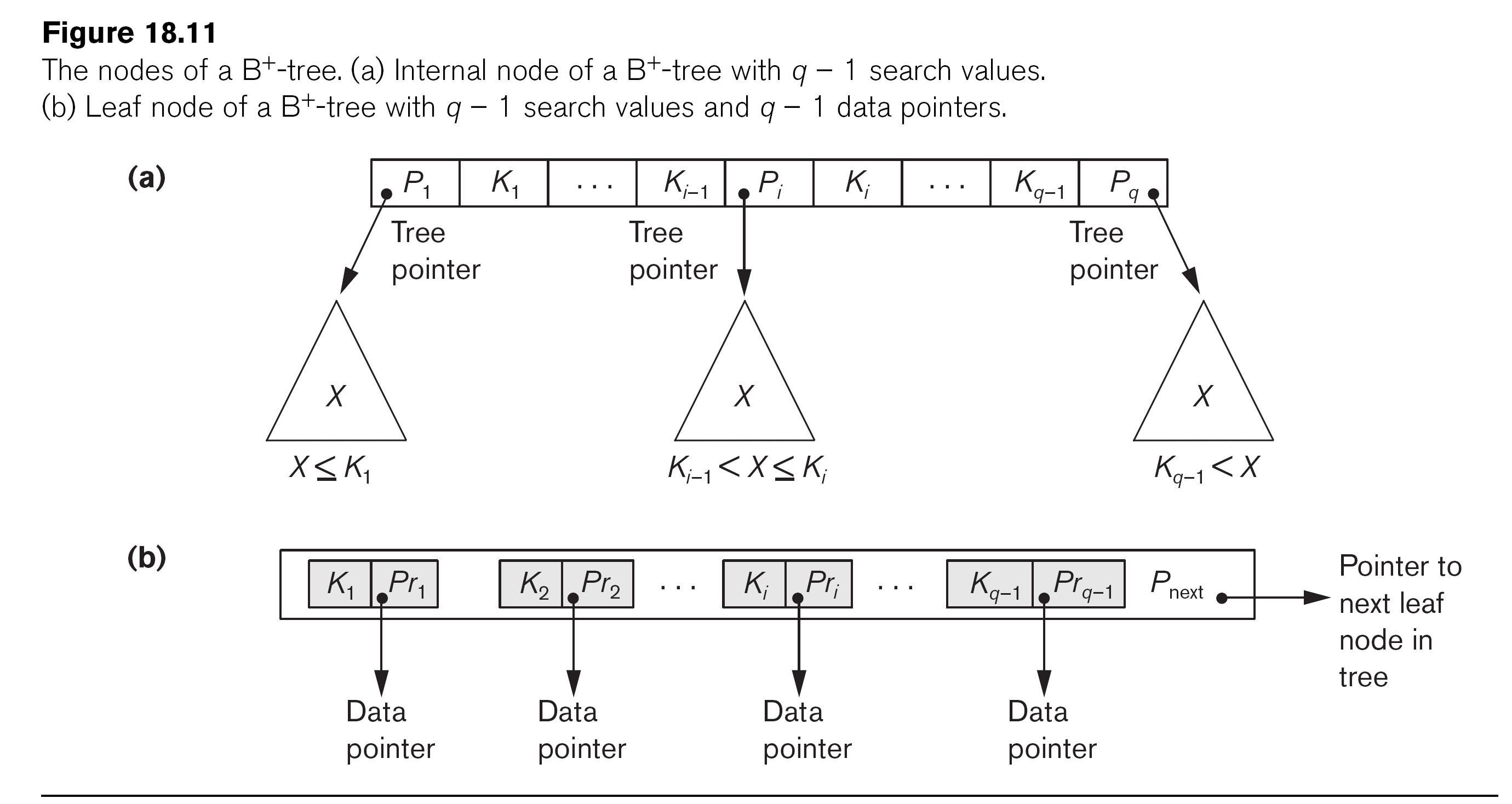
* OR HIDE PASSOWRDS
  + CREATE PUBLIC DATABASE LINK **new\_jersey** USING 'NJ';

1. Index

One form of an index is a file of entries <field value, pointer to record>, which is ordered by field value

* Indexes can also be characterized as **dense** or **sparse** 
  + A dense index has an index entry for every search key value (and hence every record) in the data file.
  + A sparse (or nondense) index, on the other hand, has index entries for only some of the search values
* In B-Tree and B+-Tree data structures, each node corresponds to a disk block
* In a B-tree, pointers to data records exist at all levels of the tree
* In a B+-tree, all pointers to data records exists at the leaf-level nodes





* Primary and unique keys automatically have indexes, but you might want to create an index on a foreign key.
* Candidate columns for indexing
  + Values are relatively unique in the column.
  + Columns used frequently in the where clause
  + Columns with many NULLs and often queries check for NOT NULL
    - WHERE col\_x IS NOT NULL
  + Columns that are not suitable for Indexing
    - May NULLs in the columns and filtering for not null in the queries
    - LONG and LONG RAW columns cannot be indexed.

1. Packages

**create** or replace package BOOK\_MANAGEMENT

as

**function** OVERDUE\_CHARGES(aName IN VARCHAR2) return NUMBER;

**procedure** NEW\_BOOK (aTitle IN VARCHAR2, aPublisher IN VARCHAR2, aCategoryName IN VARCHAR2);

end BOOK\_MANAGEMENT;

**create or replace package body** BOOK\_MANAGEMENT as

function OVERDUE\_CHARGES (aName IN VARCHAR2) return NUMBER is

owed\_amount NUMBER(10,2);

**begin**

select SUM(((ReturnedDate-CheckoutDate) -14)\*0.20)

into owed\_amount

from BOOKSHELF\_CHECKOUT

where Name = aName and (ReturnedDate- CheckoutDate) > 14;

RETURN(owed\_amount);

EXCEPTION

when NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20100, 'No books borrowed.');

**end** **OVERDUE\_CHARGES;**

procedure NEW\_BOOK (aTitle IN VARCHAR2,

aPublisher IN VARCHAR2, aCategoryName IN VARCHAR2) is

**begin**

insert into BOOKSHELF

(Title, Publisher, CategoryName, Rating)

values   
(aTitle, aPublisher, aCategoryName, NULL);

delete from BOOK\_ORDER

where Title = aTitle;

end **NEW\_BOOK**;

begin

select User, SysDate

into User\_Name, Entry\_Date

from DUAL;

end BOOK\_MANAGEMENT;

end BOOK\_MANAGEMENT;

1. Cursor

SET SERVEROUTPUT ON

DECLARE

v\_drive\_no drive.driveno%TYPE;

v\_drive\_name drive.drivename%TYPE;

v\_contamt donation.contamt%TYPE;

v\_tot\_contamt cont\_info.contamt%TYPE;

CURSOR drive\_cursor IS

SELECT driveno, drivename FROM drive

ORDER BY driveno;

CURSOR donation\_cursor IS

SELECT contamt FROM donation

WHERE v\_drive\_no = driveno

ORDER BY driveno;

BEGIN

OPEN drive\_cursor;

LOOP

FETCH drive\_cursor INTO v\_drive\_no, v\_drive\_name;

EXIT WHEN drive\_cursor%NOTFOUND;

IF donation\_cursor%ISOPEN THEN

CLOSE donation\_cursor;

END IF;

OPEN donation\_cursor;

v\_tot\_contamt := 0;

LOOP

FETCH donation\_cursor INTO v\_contamt;

EXIT WHEN donation\_cursor%NOTFOUND;

v\_tot\_contamt := v\_tot\_contamt + v\_contamt;

dbms\_output.put\_line('The current amount is: '||v\_tot\_contamt);

END LOOP;

INSERT into cont\_info

VALUES(v\_drive\_no, v\_drive\_name, v\_tot\_contamt);

CLOSE donation\_cursor;

END LOOP;

CLOSE drive\_cursor;

END;

/

SET SERVEROUTPUT OFF

1. DBA (administration)

* Database system privileges let you execute specific sets of commands.
  + **CREATE TABLE privilege**
  + **GRANT ANY PRIVILEGE**
* Database object privileges give you the ability to perform some operation on various objects.
  + **DELETE** privilege
  + **SELECT** privilege

CREATE USER **myuser** IDENTIFIED BY **password**

DEFAULT TABLESPACE users TEMPORARY TABLESPACE temp

QUOTA UNLIMITED ON users

QUOTA 100M ON my\_data;

Create user command:

* + Define the user name
  + Define the password
  + Define the tablespace for the user
  + Define the temporary tablespace for the user
  + Allocate space quotes to various tablespaces to the uwer
  + Assign attributes to the user

**Profiles** enforce:

* + - The “lifetime” of your password, which determines how frequently you must change it
    - The grace period following your password’s “expiration date” during which you can change the password
    - The number of consecutive failed connect attempts allowed before the account is automatically “locked”
    - The number of days the account will remain locked
    - The number of days that must pass before you can reuse a password
    - The number of password changes that must take place before you can reuse a password

**create profile** LIMITED\_PROFILE limit FAILED\_LOGIN\_ATTEMPTS 5;

**create user** JANE identified by EYRE profile LIMITED\_PROFILE;

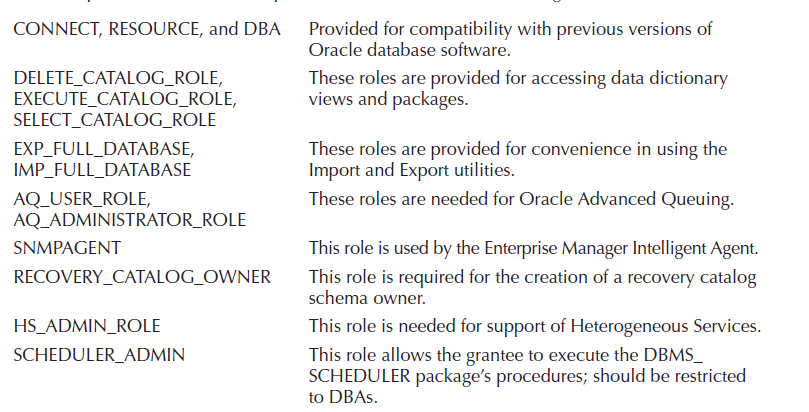
**grant** CREATE SESSION to JANE;

alter profile LIMITED\_PROFILE limit

**PASSWORD\_REUSE\_MAX** 3

**PASSWORD\_REUSE\_TIME** UNLIMITED;

**注: PASSWORD\_REUSE\_MAX** or **PASSWORD\_REUSE\_TIME** 两个是二选 一



create role role\_name

[not identified

|identified {by password | using [schema.]package

|externally | globally }];

Here are some examples of object privilege assignment:

* + grant select, insert on customer to fred, mary, joe;
  + grant insert on order\_table to update\_role;
  + grant all on customer to fred;
  + grant select on customer\_view to mary;

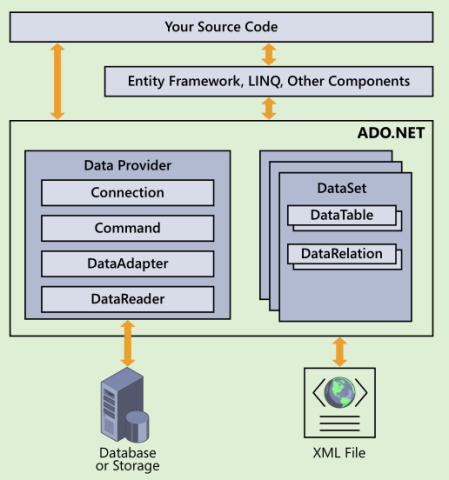
revoke {*system privilege* | *role* | all [privileges] }

[, {*system privilege* | *role* | all [privileges] }. . .]

from {*user* | *role*} [, {*user* | *role*}]. . .

* + drop user username [cascade];
* grant select on MY\_CHECKOUT to public;
* create public synonym MY\_CHECKOUT for Practice.MY\_CHECKOUT;

1. ADO.NET

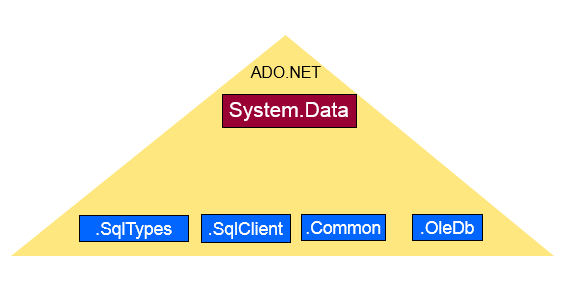


The ADO.NET Object Model

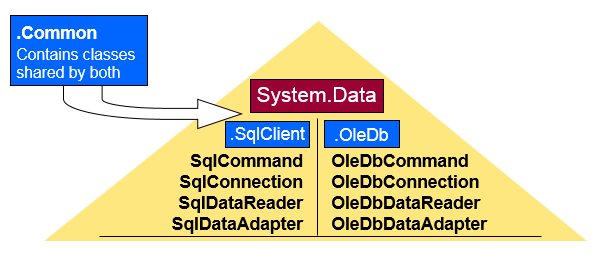
* 1. Objects of System.Data
  2. .NET data providers

ADO.NET namespace hierarchy

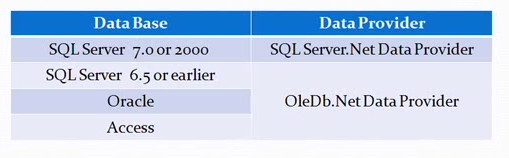
* Includes:
  + 1. **System.Data**
    2. **System.Data.OleDb**
    3. **System.Data.Common**
    4. **System.Data.SqlClient**
    5. **System.Data.SqlTypes**



* two main components of ADO.NET 3.0 for accessing and manipulating data are:
  1. .NET Framework Data Providers
  2. DataSet.



|  |  |
| --- | --- |
| .NET Framework data provider | Description |
| .NET Framework Data Provider for SQL Server | Provides data access for Microsoft SQL Server.  Uses the [System.Data.SqlClient](http://msdn.microsoft.com/en-us/library/system.data.sqlclient(v=vs.110).aspx) namespace. |
| .NET Framework Data Provider for OLE DB | For data sources exposed by using OLE DB.  Uses the [System.Data.OleDb](http://msdn.microsoft.com/en-us/library/system.data.oledb(v=vs.110).aspx) namespace. |
| .NET Framework Data Provider for ODBC | For data sources exposed by using ODBC.  Uses the [System.Data.Odbc](http://msdn.microsoft.com/en-us/library/system.data.odbc(v=vs.110).aspx) namespace. |
| .NET Framework Data Provider for Oracle | For Oracle data sources. The .NET Framework Data  Provider for Oracle supports Oracle client software  version 8.1.7 and later, and uses the  [System.Data.OracleClient](http://msdn.microsoft.com/en-us/library/system.data.oracleclient(v=vs.110).aspx) namespace. |
| EntityClient Provider | Provides data access for Entity Data Model (EDM)  applications. Uses the [System.Data.EntityClient](http://msdn.microsoft.com/en-us/library/system.data.entityclient(v=vs.110).aspx)  namespace. |
| .NET Framework Data Provider for SQL Server Compact 4.0. | Provides data access for Microsoft SQL Server  Compact 4.0. Uses the [System.Data.SqlServerCe](http://msdn.microsoft.com/en-us/library/system.data.sqlserverce.aspx)  namespace. |



* **SqlClient** :
  + C#
    - using System.Data.SqlClient;
  + VB
    - Imports System.Data.SqlClient

|  |  |  |
| --- | --- | --- |
| Driver | Provider | |
| SQLOLEDB | Microsoft OLE DB provider for SQL Server | |
| MSDAORA | Microsoft OLE DB provider for Oracle | |
| Microsoft.Jet.OLEDB.4.0 | OLE DB provider for Microsoft Jet | |
| Driver | |
| SQL Server | |
| Microsoft ODBC for Oracle | |
| Microsoft Access Driver (\*.mdb) | |

* OracleClient:
  + C#
    - using System.Data;
    - using System.Data.OracleClient;
  + VB
    - Imports System.Data
    - Imports System.Data.OracleClient

Sample Connect:



