* 学习目标

* 能够掌握Oracle的链接查询

  * 交叉链接：笛卡尔积

  * 内连接：inner join ...on

  * 外链接：left join on，right join，full join

* 能够理解数据结构之B,B+树

  * B树，平衡多路查找树，磁盘，‘矮胖’，减少IO读写次数

  * B+树

* 能够掌握Oracle的索引

  * index，提供查询性能

  * ...

* 能够掌握Oracle的序列

  * Sequence(start with increment by maxvalue cycle cache)

* 能够掌握Oracle的分区表

  * partition by range|list

---

* 回顾

  * E-R:矩形，椭圆形，菱形

    * 一对一，一对多，多对一，多对多

  * 类与类之间的关系

    * 泛化（继承），实现，关联，组合，聚合，依赖

  * PowerDesinger 画一个ER图（CDM）

  * Oracle约束：primary key（唯一非空），not null，unqiue,check,foreign key

    int cid, constraint c_p_fk foreign key(cid) references compay(id) on delete cascade,on delete set null

  * PowerDesinger画了表与表之间的关系(PDM)

* 一对多

* 数据库的设计范式：

* 1NF：原子性

* 2NF:在第一范式基础上， 要求有主键，不能产生部分依赖，要完成依赖

* 3NF：在第二范式的基础上，要求不能产生传递依赖

* 能够掌握Oracle的链接查询

16. 用一条 SQL 语句 查询出每门课都大于 80 分的学生姓名

name	kecheng	fenshu
张三	语文	81
张三	数学	75
李四	语文	76
李四	数学	90
王五	语文	81
王五	数学	100
王五	英语	90

```
create table student(
     name varchar2(10),
     kecheng varchar2(10),
     fengshu number(3,0)
)

insert into student values('张三','语文',81);
insert into student values('张三','数学',75);
insert into student values('李四','语文',76);
insert into student values('李四','数学',90);
insert into student values('王五','语文',81);
insert into student values('王五','数学',100);
insert into student values('王五','英文',91);
commit;
```

```
select name from student minus select name from student where fengshu<=80;
```

* Oracle连接查询分类

* 交叉连接

* 又称笛卡尔积链接，是两个或多个表间的无条件链接。

* 假如表A有5条数据，表B有3条数据，表A的每一条数据与表B的每一条数据链接，结果会为5*3=15条

* 内连接

* 内部链接是两个或多个表的链接，这些表只返回满足链接条件的行。

* 外连接

* 外部链接返回满足链接条件的所有行并且返回某个链接表不符合链接条件的行。

* 外链接的分类

* 左外链接

* 外部链接返回满足链接条件的所有行

* 并且返回左链接表不符合链接条件的行

* 右外链接

* 外部链接返回满足链接条件的所有行

* 并且返回右链接表不符合链接条件的行

* 全外链接

* 外部链接返回满足链接条件的所有行

* 并且返回坐右链接表不符合链接条件的行

```
连接查询
CREATE TABLE employee
(
     emp_no CHAR(8) PRIMARY KEY NOT NULL,   --工号，主键，非空
     emp_name VARCHAR2(30) NOT NULL,--姓名,非空
     emp_id VARCHAR2(18), --身份证号，代表18位整数
     emp_address varchar2(30),
     tea_no  CHAR(4)
);
```

```
10
11 create table teacher(
12   tea_no CHAR(8) PRIMARY KEY NOT NULL,   --工号，主键，非空
13   tea_name VARCHAR2(30) NOT NULL--姓名,非空
14 );
15 insert into teacher  values('t001','小黑');
16 insert into teacher  values('t002','小白');
17 insert into teacher  values('t003','小明');
18 commit;
19 insert into employee(emp_no,emp_name,emp_id,emp_address,tea_no) values('lg001',
20 insert into employee(emp_no,emp_name,emp_id,emp_address,tea_no) values('lg002',
21 insert into employee(emp_no,emp_name,emp_id,emp_address,tea_no) values('lg003',
22 insert into employee(emp_no,emp_name,emp_id) values('lg004','张四','44152119990
23 insert into employee(emp_no,emp_name,emp_id) values('lg005','张5','441521199909
24 commit;
25 select * from employee;
26 select * from teacher;
27
28 * 测试
29  * 交叉连接
30   * 又称笛卡尔积链接，是两个或多个表间的无条件链接。
31   * 假如表A有5条数据，表B有3条数据，表A的每一条数据与表B的每一条数据链接，结果会为5*3=
32   * select * from employee,teacher
33  * 内链接
34   * 内部链接是两个或多个表的链接，这些表只返回满足链接条件的行。
35   * 标准SQL的写法
36     * select * from employee e inner join teacher t on e.tea_no=t.tea_no;
37     * 可以省略（inner）：select * from employee e join teacher t on e.tea_no=t.t
38   * Oracle的写法
39     * select * from employee e,teacher t where e.tea_no=t.tea_no;
40  * 左外链接
41    * 标准SQL的写法
42     * select * from employee e left join teacher t on e.tea_no=t.tea_no;
43    * Oracle的写法
44     * select * from employee e,teacher t where e.tea_no=t.tea_no(+);
45  * 右外链接
46    * 标准SQL的写法
47     * select * from employee e right join teacher t on e.tea_no=t.tea_no;
48    * Oracle的写法
49     * select * from employee e,teacher t where e.tea_no(+)=t.tea_no;
```

```
50  * 全外链接
51     * select * from employee e full join teacher t on e.tea_no=t.tea_no;
52
53 * 温馨提醒：
54   * 连接查询的性能优于子查询，所以能用连接查询的地方尽量少用子查询
```

* 查询两个链接表

  * select * from employee;

	EMP_NO	EMP_NAME	EMP_ID	EMP_ADDRESS	TEA_NO
▶ 1	lg001	张大	441521199909092111	4401	t001
2	lg002	张二	441521199909092112	4403	t002
3	lg003	张三	441521199909092113	4401	t001
4	lg004	张四	441521199909092114		
5	lg005	张5	441521199909092115		

  * select * from teacher;

	TEA_NO	TEA_NAME
▶ 1	t001	小黑
2	t002	小白
3	t003	小明

* 交叉连接操作效果图

  * select * from employee,teacher

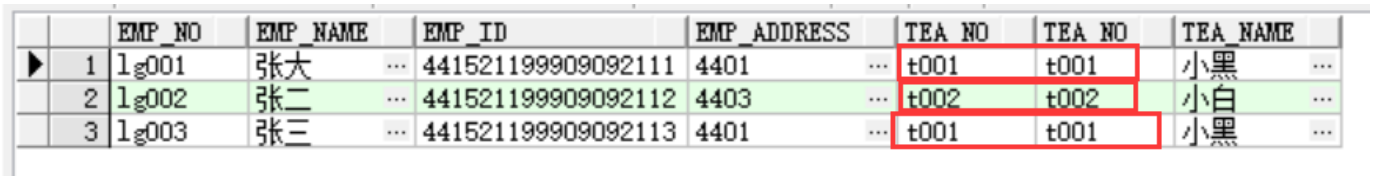
▶ 1	lg001	张大	441521199909092111	4401	t001	t001	小黑
2	lg002	张二	441521199909092112	4403	t002	t001	小黑
3	lg003	张三	441521199909092113	4401	t001	t001	小黑
4	lg004	张四	441521199909092114			t001	小黑
5	lg005	张5	441521199909092115			t001	小黑
6	lg001	张大	441521199909092111	4401	t001	t002	小白
7	lg002	张二	441521199909092112	4403	t002	t002	小白
8	lg003	张三	441521199909092113	4401	t001	t002	小白
9	lg004	张四	441521199909092114			t002	小白
10	lg005	张5	441521199909092115			t002	小白
11	lg001	张大	441521199909092111	4401	t001	t003	小明
12	lg002	张二	441521199909092112	4403	t002	t003	小明
13	lg003	张三	441521199909092113	4401	t001	t003	小明
14	lg004	张四	441521199909092114			t003	小明
15	lg005	张5	441521199909092115			t003	小明

* 内链接

  * select * from employee e inner join teacher t on e.tea_no=t.tea_no;

  * select * from employee e join teacher t on e.tea_no=t.tea_no;

* select * from employee e,teacher t where e.tea_no=t.tea_no;

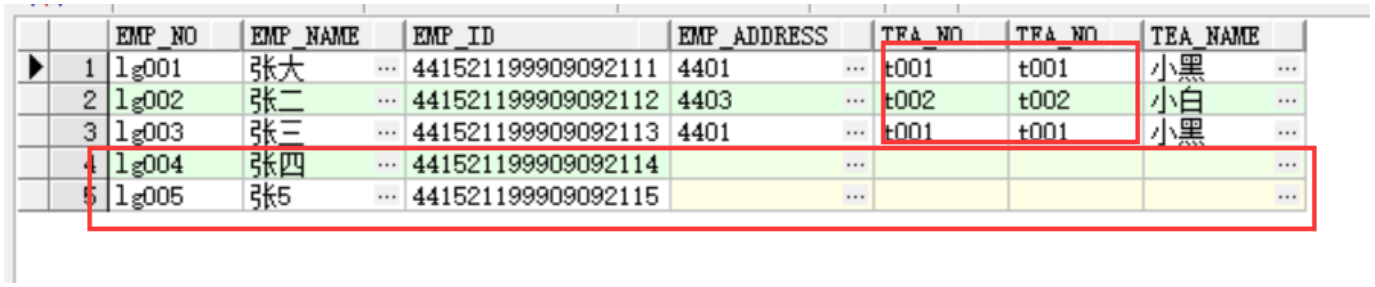


	EMP_NO	EMP_NAME	EMP_ID	EMP_ADDRESS	TEA_NO	TEA_NO	TEA_NAME
1	lg001	张大	441521199909092111	4401	t001	t001	小黑
2	lg002	张二	441521199909092112	4403	t002	t002	小白
3	lg003	张三	441521199909092113	4401	t001	t001	小黑

* 左链接

* select * from employee e left join teacher t on e.tea_no=t.tea_no;

* select * from employee e,teacher t where e.tea_no=t.tea_no(+);

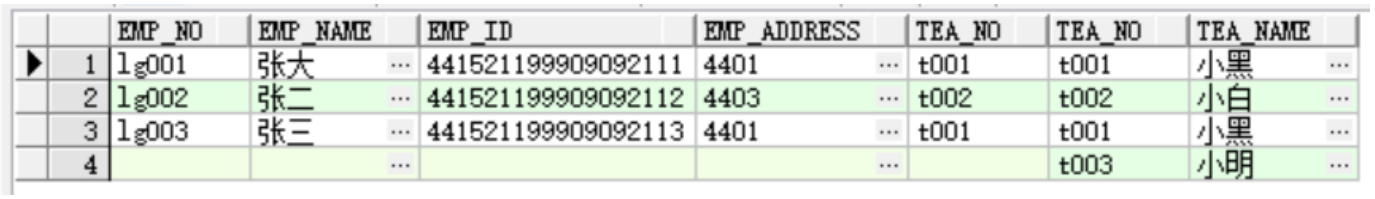


	EMP_NO	EMP_NAME	EMP_ID	EMP_ADDRESS	TEA_NO	TEA_NO	TEA_NAME
1	lg001	张大	441521199909092111	4401	t001	t001	小黑
2	lg002	张二	441521199909092112	4403	t002	t002	小白
3	lg003	张三	441521199909092113	4401	t001	t001	小黑
4	lg004	张四	441521199909092114				
5	lg005	张5	441521199909092115				

* 右链接

* select * from employee e right join teacher t on e.tea_no=t.tea_no;

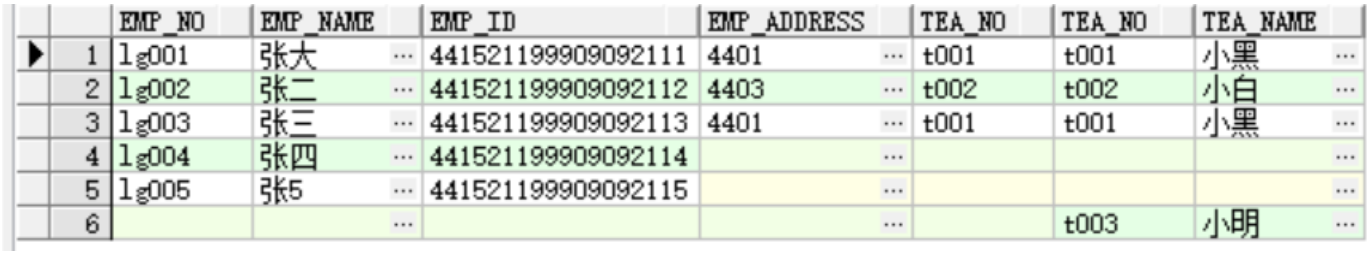
* select * from employee e,teacher t where e.tea_no(+)=t.tea_no;



	EMP_NO	EMP_NAME	EMP_ID	EMP_ADDRESS	TEA_NO	TEA_NO	TEA_NAME
1	lg001	张大	441521199909092111	4401	t001	t001	小黑
2	lg002	张二	441521199909092112	4403	t002	t002	小白
3	lg003	张三	441521199909092113	4401	t001	t001	小黑
4						t003	小明

* 全外链接

* select * from employee e full join teacher t on e.tea_no=t.tea_no;



	EMP_NO	EMP_NAME	EMP_ID	EMP_ADDRESS	TEA_NO	TEA_NO	TEA_NAME
1	lg001	张大	441521199909092111	4401	t001	t001	小黑
2	lg002	张二	441521199909092112	4403	t002	t002	小白
3	lg003	张三	441521199909092113	4401	t001	t001	小黑
4	lg004	张四	441521199909092114				
5	lg005	张5	441521199909092115				
6						t003	小明

* 能够理解数据结构之B,B+树

* B树概述（B-Balance-平衡）

* B树又叫B-树，平衡多路查找树

* 二叉查找树，平衡二叉树，红黑树是从内存中查找的。

* 假如有海量的数据，不可能一次性读取到内存中，这时候就要考虑的是，如何在磁盘中快速找到需要的数据。

* B树或者B+树可以帮助我们查找磁盘中的大量数据

* B树普遍运用在数据库和文件系统

* B树与平衡二叉树相同点

* B树的节点数据大小也是按照左小右大

* B树与平衡二叉树不同点

* 平衡二叉树节点最多有两个子树，而B树每个节点可以有多个子树，M阶B树表示该树每个节点最多有M个子树

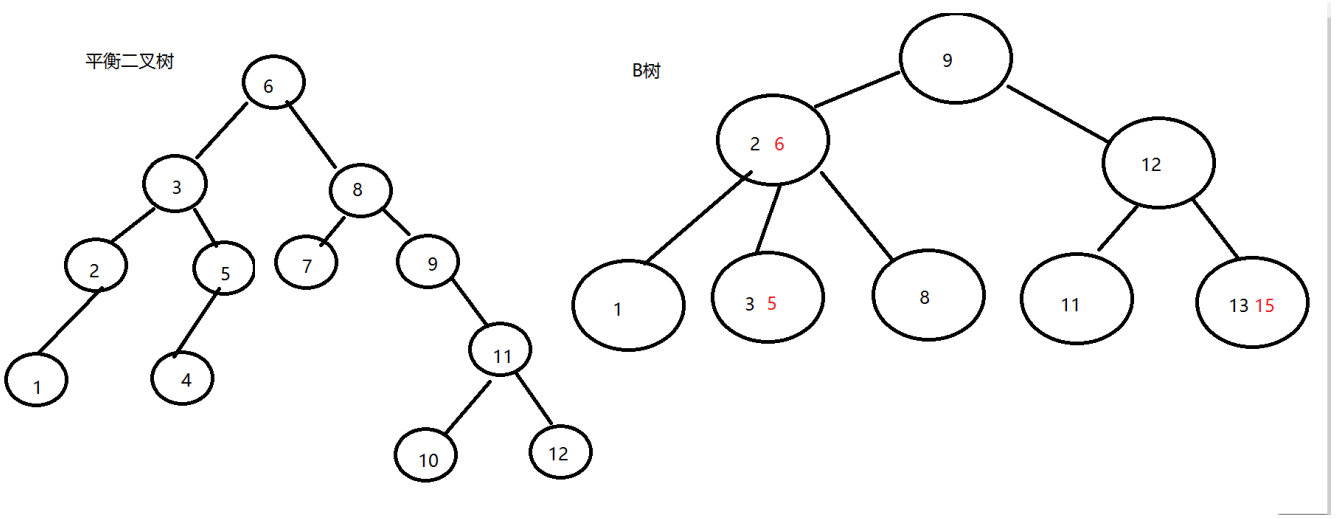
* 平衡二叉树每个节点只有一个数据和两个指向孩子的指针，而B树每个中间节点有k个子树（k介于阶数M和M/2之间，M/2向上取整）和k-1个关键字（可以理解为数据），B树的所有叶子节点都在同一层，并且叶子节点只有关键字，指向孩子的指针为 null

* 一棵B树必须满足以下条件：

* 若根结点不是终端结点，则至少有2棵子树

* 除根节点以外的所有非叶结点至少有 M/2 棵子树,至多有M个子树（关键字数为子树减一）

* 所有的叶子结点都位于同一层



* 总结：B树的每个节点可以表示的信息更多，因此整个树更加“矮胖”，这在从磁盘中查找数据（先读取到内存、后查找）的过程中，可以减少磁盘 IO 的次数，从而提升查找速度。

* B+树概述

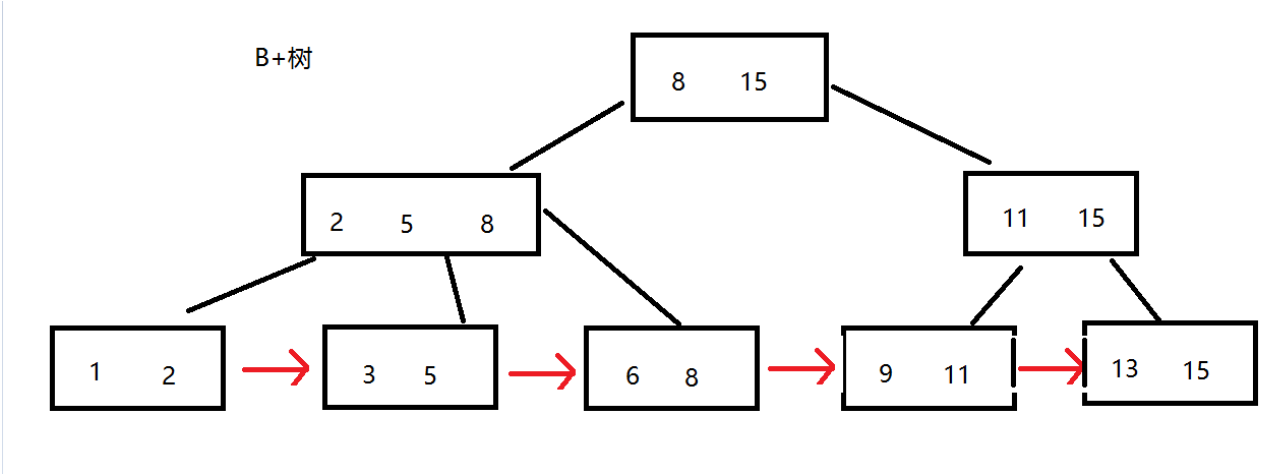
  * B+树查询性能比B树的更高。

  * 一棵B+树需要满足以下条件：

   * 节点的子树数和关键字数相同（B树是关键字数比子树数少一）

   * 节点的关键字表示的是子树中的最大数，在子树中同样含有这个数据

   * 叶子节点包含了全部数据，同时符合左小右大的顺序



* B+树的三个特点：

  * 关键字数和子树相同

  * 非叶子节点仅用作索引，它的关键字和子节点有重复元素

  * 叶子节点用指针连在一起

* B+树的三个优点：

 * 层级更低,IO次数更少

 * 每次都需要查询到叶子节点，查询性能稳定

 * 叶子节点形成有序链表，范围查询方便

* B树和B+树区别

  * B树和B+树最重要的一个区别就是B+树只有叶节点存放数据，其余节点用来索引，而B-树是每个索引节点都会有Data域。

* 温馨提醒：

    * Oracle 默认采用B树索引，MySql默认采用B+索引

* 能够掌握Oracle的索引

* 索引概述

* 在关系数据库中，索引是一种单独的、物理的对数据库表中一列或多列的值进行排序的一种存储结构，

它是某个表中一列或若干列值的集合和相应的指向表中物理标识这些值的数据页的逻辑指针清单。

* 索引作用

* 相当于图书的目录，可以根据目录中的页码快速找到所需的内容。

* 索引优点

* 大大加快数据的检索速度;

* 创建唯一性索引，保证数据库表中每一行数据的唯一性;

* 加速表和表之间的连接;

* 在使用分组和排序子句进行数据检索时，可以显著减少查询中分组和排序的时间。

* 索引缺点

* 索引需要占物理空间。

* 当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，增加维护难度。

* 建议建立索引的列

* 经常需要搜索的列上

* 主键，一般建立唯一性索引，保持数据的唯一性

* 主键和具有唯一性约束的列都会自动创建索引。

* 外键，提高表与表之间连接的速度

* 需要排序的列上

* where子句后边经常出现的字段

* 经常需要根据范围进行搜索的列上，比如日期

* 不建议建立索引的列

* 很少进行搜索的列上

* blob类型的列上

* 修改频率比较高的列上

* 建立索引，但是无法使用情况

* 使用不等于<>、 != ，（不等于操作符一定会进行全表扫描）

* 使用is null 、 is not null

* 只要索引中出现一个null，那么这个索引就报废了。所以在建立索引的时候，一定要将准备建立索引的列设置为not null

* where子句中，进行了数据类型不匹配的比较，比如(where row_num ='1')的时候

* 使用like'%dd%'，无法使用索引，造成全表扫描

* 索引按逻辑分类

* 单列索引

* 组合索引

* 唯一索引

* 非唯一索引

* 按Oracle的索引类型分类

* B树索引（默认）

* 位图索引

```
* 创建索引语法
CREATE [UNIQUE|BITMAP] INDEX  index_name ON table_name(column_list);
示例：
 --创建单列唯一索引
CREATE unique INDEX  index_emp_emp_id ON employee(emp_id);
--创建单列非唯一索引
CREATE  INDEX  index_emp_emp_id ON employee(emp_id);
--创建组合列、唯一索引
CREATE unique INDEX  index_emp_emp_name_emp_id ON employee(emp_name,emp_age);
--创建组合列、非唯一索引
CREATE INDEX  index_emp_emp_name_emp_id ON employee(emp_name,emp_age);
* 删除索引语法
DROP INDEX  index_name;
示例：
```

```
drop index index_emp_emp_name_emp_id;

* 测试不适用索引和使用索引的性能
 * 创建表
CREATE TABLE emp3
(
     emp_no VARCHAR2(30) PRIMARY KEY NOT NULL,   --工号，主键，非空
     emp_name VARCHAR2(30) NOT NULL,--姓名,非空
     emp_id VARCHAR2(18), --身份证号，代表18位整数
     emp_address varchar2(30)
);
 * 插入大量的数据
   @Test
  public void test3() throws SQLException {
      //124385
        long start = System.currentTimeMillis();
        // 1 获得连接
        Connection conn = ConnectionUtils.getConn();
        String sql = "insert into emp3(emp_no,emp_name,emp_id,emp_address) valu
        PreparedStatement psmt = conn.prepareStatement(sql);
        for (int i = 1; i < 5000000; i++) {
            psmt.setString(1, String.valueOf(i));
            psmt.setString(2, "xiaobai"+i);
            psmt.setString(3, "441521199909092111");
            psmt.setString(4, "4405");
            psmt.addBatch();
            // 每次发一万条
            if (i % 10000 == 0) {
                psmt.executeBatch();
                psmt.clearBatch();
            }
        }
        psmt.executeBatch();
        psmt.clearBatch();
        long end = System.currentTimeMillis();
        System.out.println("使用批处理花的时间:" + (end - start));
        ConnectionUtils.close(conn, psmt, null);
    }

 * 开启执行时间显示
```

```
  * set timing on
  * 执行语句后：Elapsed: 00:00:00.01
* select count(*) from emp3;
  * Elapsed: 00:00:07.03
* select emp_no,emp_name,emp_id,emp_address from emp3 where emp_name='xiaobai2
  * Elapsed: 00:00:03.87
* CREATE  INDEX  index_emp3_emp_name ON emp3(emp_name);
  * Elapsed: 00:00:21.84
* select emp_no,emp_name,emp_id,emp_address from emp3 where emp_name='xiaobai2
  * Elapsed: 00:00:00.09
* select emp_no,emp_name,emp_id,emp_address from emp3 where emp_name='xiaobai2
  * Elapsed:00:00:00.06
* select count(*) from emp3;
  * Elapsed: 00:00:05.58
* drop index index_emp3_emp_name;
  * Elapsed: 00:00:00.37
* CREATE  UNIQUE INDEX  index_emp3_emp_name ON emp3(emp_name);
  * Elapsed: 00:00:16.60
* select count(*) from emp3;
  * Elapsed: 00:00:01.70
  * Elapsed: 00:00:00.17
* select emp_no,emp_name,emp_id,emp_address from emp3 where emp_no=299998
  * Elapsed: 00:00:03.27
* select emp_no,emp_name,emp_id,emp_address from emp3 where emp_no='299998';
  * Elapsed: 00:00:00.05
*  CREATE  INDEX  index_emp3_emp_no ON emp3(emp_no);
  * ORA-01408: such column list already indexed

* 位图索引
  * 假如有五行数据，存储用户信息，性别分别是 男-男-女-女-男 (1-1-0-0-1)，1表示男，0表
    婚姻状况分别是 已婚-未婚-已婚-已婚-未婚 (1-0-1-1-0)，1表示已婚，0表示未婚
    在这两行上创建位图索引后，则生成相应的性别向量(11001)和婚姻状况向量(10110)
    当执行sql语句时：select * from t_user where gender='男'and marital='已婚';
    这时候两个向量执行逻辑与操作
    1 1 0 0 1   (性别向量)
    1 0 1 1 0   (婚姻状况向量)
    1 0 0 0 0   (逻辑与结果)
    这样可以直接看到第一条数据符合查询要求。
* 测试
  create table t_user(
```

```
    id int primary key not null,
    name varchar2(50) not null,
    gender varchar2(2),
    marital varchar2(8)
     )
 * 插入数据
 @Test
public void test5() throws SQLException {
      //52328
      long start = System.currentTimeMillis();
      // 1 获得连接
      Connection conn = ConnectionUtils.getConn();
      String sql = "insert into t_user(id,name,gender,marital) values(?,?,?,?
      PreparedStatement psmt = conn.prepareStatement(sql);
      for (int i = 1; i < 5000000; i++) {
          psmt.setString(1, String.valueOf(i));
          psmt.setString(2, "xiaobai"+i);
          psmt.setString(3, i%2==0?"男":"女");
          psmt.setString(4, i%3==0?"已婚":"未婚");
          psmt.addBatch();
          // 每次发一万条
          if (i % 10000 == 0) {
              psmt.executeBatch();
              psmt.clearBatch();
          }
      }
      psmt.executeBatch();
      psmt.clearBatch();
      long end = System.currentTimeMillis();
      System.out.println("使用批处理花的时间:" + (end - start));
      ConnectionUtils.close(conn, psmt, null);
  }
* 建立索引和没有建立索引进行测试
select * from t_user where gender='男'and marital='已婚';
 * Elapsed: 00:05:19.61
CREATE  INDEX  index_t_user_gender ON t_user(gender);
 * Elapsed: 00:00:10.90
CREATE  INDEX  index_t_user_marital ON t_user(marital);
 * Elapsed: 00:00:11.69
select * from t_user where gender='男'and marital='已婚';
```

```
 * Elapsed: 00:05:37.10
drop index index_t_user_gender;
 * Elapsed: 00:00:00.15
drop index index_t_user_marital;
 * Elapsed: 00:00:00.10
CREATE INDEX  index_t_user_gender_marital ON t_user(gender,marital);
 * Elapsed: 00:00:12.02
select * from t_user where gender='男'and marital='已婚';
 * Elapsed: 00:04:33.34

drop index index_t_user_gender_marital;
 * Elapsed: 00:00:00.02
CREATE BITMAP INDEX  index_t_user_gender ON t_user(gender);
 * Elapsed: 00:00:01.85
CREATE BITMAP INDEX  index_t_user_marital ON t_user(marital);
 * Elapsed: 00:00:01.83
select * from t_user where gender='男'and marital='已婚';
 * Elapsed: 00:03:19.51
drop index index_t_user_gender;
 * Elapsed: 00:00:05.74
drop index index_t_user_marital;
 * Elapsed: 00:00:00.03
CREATE BITMAP INDEX  index_t_user_gender_marital ON t_user(gender,marital);
 * Elapsed: 00:00:02.25
select * from t_user where gender='男'and marital='已婚';
 * Elapsed: 00:03:24.85
drop INDEX  index_t_user_gender_marital;
```

* 能够掌握Oracle的序列

* 序列概述

    * 序列是oracle用来生产一组等间隔的数值。

    * 序列是递增，而且连续的。

    * oracle主键没有自增类型，所以一般使用序列产生的值作为某张表的主键,实现主键自增。

    * 序列的编号不是在插入记录的时候自动生成的，必须调用序列的方法来生成（一般调

用nextval方法）。

```
语法：
CREATE SEQUENCE sequence_name
[START WITH integer]
[INCREMENT BY integer]
[MAXVALUE integer| NOMAXVALUE]
[MINVALUE integer| NOMINVALUE]
[CYCLE| NOCYCLE]
[CACHE integer|NOCACHE];
 * start with：生成第一个序列号,对于升序列,其默认值为序列最小值;对于降序序列,其默认值
 * increment by：用于指定序列号之间的间隔,其默认值为1,如果integer为正值,则生成的序列
 * maxvalue：序列可以生成的最大值。
 * nomaxvalue：oracle将升序序列的最大值设为1027,将降序序列的最大值设为-1.这是默认选项
 * minvalue：minvalue必须小于或等于start with的值,并且必须小于maxvalue的值。
 * nominvalue：oracle将升序的最小值设为1,或将降序序列的最小值设为-1026.这是默认值。
 * cycle：序列在达到最大值或最小值后,将继续从头开始生成值。
 * nocycle：序列在达到最大值或最小值后,将不能再继续生成值。不写默认为nocycle这是默认选
 * cache：预先分配一组序列号,并将其保留在内存中,这样可以更快地访问序列号.当用完缓存中的
 * nocache：不会加快访问速度而预先分配序列号,如果在创建序列时忽略了cache和nocache,or

* 创建序列
CREATE SEQUENCE seq_test
START WITH 1
INCREMENT BY 1
MAXVALUE 2000
NOCYCLE
CACHE 30;
* 使用序列
 * NEXTVAL:第一次访问时，返回序列的初始值，后继每次调用时，按步长增加的值返回。
    * select seq_test.nextval from dual;
 * CURRVAL:返回序列当前的值，创建新序列后，不能直接使用CURRVAL访问序列，使用过NEXTVAL
    * select seq_test.currval from dual;

* 测试案例
--drop table employee;
CREATE TABLE employee
(
     emp_no number(8) PRIMARY KEY NOT NULL,    --工号，主键，非空
```

```
    emp_name VARCHAR2(30) NOT NULL,--姓名,非空
    emp_id VARCHAR2(18), --身份证号，代表18位整数
    emp_age NUMBER(3,0)  --年龄
);
--创建序列
CREATE SEQUENCE seq_emp_no
START WITH 1
INCREMENT BY 1
MAXVALUE 2000
MINVALUE 1
NOCYCLE
CACHE 30;

--使用序列插入数据
insert into employee(emp_no,emp_name,emp_id,emp_age) values(seq_emp_no.nextval,
--查询数据,如果发现表中的数据从2开始，可以修改deferred segment creation（创建延迟片断
--执行alter session set deferred_segment_creation=false; 把参数的值高为false,此参
SELECT * FROM employee;
SELECT seq_emp_no.CURRVAL FROM dual;

--修改序列
ALTER SEQUENCE seq_emp_no
MAXVALUE 5
nocycle;
 * 假如超过最大值，又没有循环，会报错
    *  ORA-08004: sequence SEQ_EMP_NO.NEXTVAL exceeds MAXVALUE and cannot be in
ALTER SEQUENCE seq_emp_no
MAXVALUE 50
cycle;

--删除序列
DROP SEQUENCE seq_emp_no;

--序列与SYS_GUID函数（生成唯一标识符）
--使用SYS_GUID函数，32位，由时间戳和机器标识符生成，保证在不同数据库是唯一
SELECT sys_guid() FROM dual;

```

* 能够掌握Oracle的分区表

* Oracle 分区表概述

* Oracle把表中的行分为不同部分，存储在不同的位置，每一部分称为一个分区，分区的表称为分区表。

* Oracle 分区表的作用

* 安全 ：分区存放于不同的磁盘，减少同时损坏

* 查询：查询可按分区

* 管理：可按分区加载、删除

* 备份和恢复：针对分区备份与恢复，方便

* Oracle分区表的使用

* 数据量大的表，一般大于2GB

* 数据有明显的界限划分

* 对于Long和Long Raw类型列不能使用分区。

```
* 范围分区
语法：
在Create Table语句后增加
     PARTITION BY RANGE(column_name)
        (
           PARTITION part1 VALUE LESS THAN (range1) [TABLESPACE tbs1],
           PARTITION part2 VALUE LESS THAN (range2) [TABLESPACE tbs2],
           ....
           PARTITION partN VALUE LESS THAN (MAXVALUE) [TABLESPACE tbsN]
        );
--drop table employee;
CREATE TABLE employee
(
     emp_no char(8) PRIMARY KEY NOT NULL,   --工号，主键，非空
     emp_name VARCHAR2(30) NOT NULL,--姓名,非空
     emp_id VARCHAR2(18), --身份证号，代表18位整数
     emp_record_no number(15), --记录号,表示该员工是公司的第几位入职的员工
     emp_birthday date  --出生日期
)
```

```
PARTITION BY RANGE (emp_birthday)
(
   PARTITION P1 VALUES LESS THAN (to_date('2011-07-1', 'yyyy-mm-dd')),
   PARTITION P2 VALUES LESS THAN (to_date('2011-10-1', 'yyyy-mm-dd')),
   PARTITION P3 VALUES LESS THAN (maxvalue)
);

* select dbms_metadata.get_ddl('TABLE','EMPLOYEE') from dual;

insert into employee(emp_no,emp_name,emp_id,emp_record_no,emp_birthday) values(
insert into employee(emp_no,emp_name,emp_id,emp_record_no,emp_birthday) values(
insert into employee(emp_no,emp_name,emp_id,emp_record_no,emp_birthday) values(
insert into employee(emp_no,emp_name,emp_id,emp_record_no,emp_birthday) values(
insert into employee(emp_no,emp_name,emp_id,emp_record_no,emp_birthday) values(
insert into employee(emp_no,emp_name,emp_id,emp_record_no,emp_birthday) values(
insert into employee(emp_no,emp_name,emp_id,emp_record_no,emp_birthday) values(
commit;

select * from employee;
select * from employee partition(P1);
select * from employee partition(P2);
select * from employee partition(P3);
--要查看在第三季度的数据
SELECT  *  FROM employee partition(P3);
--要删除第三季度的数据
DELETE FROM employee partition(P3);
--查看所有
SELECT  *  FROM employee;

* 作业按入读学校顺序分区
 * 提示建表时分区写法
PARTITION BY RANGE (emp_record_no)
(
   PARTITION P1 VALUES LESS THAN (3),
   PARTITION P2 VALUES LESS THAN (6),
   PARTITION P3 VALUES LESS THAN (maxvalue)
);

* 列表分区
语法：
```

```
 PARTITION BY LIST(column_name)
    (
           PARTITION part1 VALUES (values_list1),
           PARTITION part2 VALUES (values_list2),
           ....
           PARTITION partN VALUES (DEFAULT)
     );
     其中：column_name是以其为基础创建列表分区的列。
          part1...partN是分区的名称。
          values_list是对应分区的分区键值的列表。
          DEFAULT关键字允许存储前面的分区不能存储的记录。
--drop table employee;
CREATE TABLE employee
(
     emp_no char(8) PRIMARY KEY NOT NULL,   --工号，主键，非空
     emp_name VARCHAR2(30) NOT NULL,--姓名,非空
     emp_id VARCHAR2(18), --身份证号，代表18位整数
     emp_record_no number(15), --记录号,表示该员工是公司的第几位入职的员工
     emp_province VARCHAR2(50) NOT NULL --所属省份
)
partition by list (emp_province)
(
  partition bj values ('北京'),
  partition gx_hn values ('广西','海南'),
  partition gd values ('广东'),
  partition qt values (default)
);


insert into employee(emp_no,emp_name,emp_id,emp_record_no,emp_province) values(
insert into employee(emp_no,emp_name,emp_id,emp_record_no,emp_province) values(
insert into employee(emp_no,emp_name,emp_id,emp_record_no,emp_province) values(
insert into employee(emp_no,emp_name,emp_id,emp_record_no,emp_province) values(
insert into employee(emp_no,emp_name,emp_id,emp_record_no,emp_province) values(
insert into employee(emp_no,emp_name,emp_id,emp_record_no,emp_province) values(
insert into employee(emp_no,emp_name,emp_id,emp_record_no,emp_province) values(
commit;

select * from employee;
select * from employee partition(bj);
```

```
select * from employee partition(gx_hn);
select * from employee partition(gd);
select * from employee partition(qt);
delete from employee partition(bj);
--要查看在第三分区的数据
SELECT  *  FROM employee partition(qt);
--要删除第三分区的数据
DELETE FROM employee partition(bj);
--查看所有
SELECT  *  FROM employee;

```