Oracle表设计

- What?Why?How?

本章概述

- 视图
- 序列
- Insert delete update
  - 事物
    - acid
- 建表
  - 三范式
- 约束
- 做练习
  - 设计表

VIEW视图的定义

- 视图(view)，也称虚表, 不占用物理空间，这个也是相对概念，因为视图本身的定义语句还是要存储在数据字典里的。视图只有逻辑定义。每次使用的时候, 只是重新执行SQL.
- 视图是从一个或多个实际表中获得的，这些表的数据存放在数据库中。那些用于产生视图的表叫做该视图的基表。一个视图也可以从另一个视图中产生。
- 视图的定义存在数据库中，与此定义相关的数据并没有再存一份于数据库中。通过视图看到的数据存放在基表中。
- 视图看上去非常象数据库的物理表，对它的操作同任何其它的表一样。当通过视图修改数据时，实际上是在改变基表中的数据；相反地，基表数据的改变也会自动反映在由基表产生的视图中。由于逻辑上的原因，有些Oracle视图可以修改对应的基表，有些则不能（仅仅能查询）。

创建视图

- 在CREATE VIEW语句后加入子查询.

```
CREATE [OR REPLACE] VIEW view
[(alias[, alias]...)]
AS subquery
[WITH READ ONLY];
```

```
create or replace view v$_emp_dept
as
select emp.deptno,ename,dname from emp
join dept  on  emp.deptno=dept.deptno
with read only
```

视图

- 在查询时，不需要再写完全的Select查询语句，只需要简单的写上从视图中查询的语句就可以了
- select * from v$_emp_dept
- 当视图不再需要的时候，用“drop view” 撤销。删掉视图不会导致数据的丢失，因为视图是基于数据库的表之上的一个查询定义.
- Drop view v$_emp_dept;

创建视图

```
create or replace view v_test01 as
 select * from emp
```

查找用户视图

- SELECT *
- FROM user_views
- WHERE view_name ='V_TEST01';

修改视图里面的数据

- insert into v_test01(ename,EMPNO,job,DEPTNO) values('cai30',9921,'SALESMAN',10)

修改视图对应基表数据

- 修改数据:
- UPDATE view_name SET ...
- 若一个视图依赖于多个基本表, 则一次修改该视图只能修改一个基本表的数据.
- 删除数据:
- Delete from view_name where ...
- 同样, 当视图依赖多个基表时, 不能使用此语句来删除基表中的数据. 只能删除依赖一个基表的数据.
- insert into v_test01(ename,EMPNO,job,DEPTNO) values('cai30',9921,'SALESMAN',10)
- delete from v_test01 where ename='cai30';
- update v_test01 set ename='cai10' where ename='cai30';

删除视图

- drop view v_test01;

只读视图

- 创建视图

```
create or replace view v_test01 as (
select * from emp  with read only
```

测试插入数据

```
insert into v_test01(ename,EMPNO,job,DEPTNO)
values('cai30',9921,'SALESMAN',10)
```

题目

- 我们要求平均薪水的等级最低的部门，它的部门名称是什么，我们完全使用子查询

题目分析

```
select dname, grade from
  (select deptno, avg_sal, grade from
    (select deptno, avg(sal) avg_sal from emp group by deptno) t,
    salgrade s
   where
    t.avg_sal between s.losal and s.hisal )
  t1,
  dept
where t1.deptno = dept.deptno
  and
  t1.grade =
     (select min(grade) from
         (select deptno, avg_sal, grade from
                (select deptno, avg(sal) avg_sal from emp group by deptno) t,
                salgrade s
            where
                t.avg_sal between s.losal and s.hisal )
     );
```

创建视图

```
create view   v$_temp   as
    (select deptno, avg_sal, grade from
        (select deptno, avg(sal) avg_sal from emp group by deptno) t,
        salgrade s
     where
        t.avg_sal between s.losal and s.hisal );
```

使用视图

```
select dname from dept, v$_temp where
v$_temp.deptno = dept.deptno
and grade = (select min(grade) from v$_temp);
```

授权视图

- **使用system用户为scott增加权限：**grant create view,create table to scott;
- **使用system用户为scott解锁：**alter user scott account unlock;

用户管理

1．创建用户
语法：create user username identified by password
红色字体为用户名密码。
create user bjmsb identified by bjmsb;
2.查看用户是否创建
SQL>select username from dba_users;

使用管理员账号创建用户
Sys
system

用户授权

账户授权语法：

grant privileges [ON object_name] to username

将权限privileges授予用户username

SQL>grant create session to John;

--授权：连接权限

登录：

SQL>conn John/johnpsw@test;

将scott用户的emp表所有权限授予John，则使用下列命令：

SQL>grant all on scott.emp to John;

select * from scott.emp

如果要收回授予用户John的scott用户表emp的所有权限，使用下列SQL语句：

SQL>revoke all on scott.emp from John;

- 创建用户
  - Create user 用户名 identified by 密码
  - 链接登录
    - Grant create session to 用户名
  - 授予表的权限
    - Grant all on scott.emp to 用户名
  - 收回权限
    - Revoke all on scott.emp from 用户名

修改用户密码

将John用户的口令修改为 newpsw。

SQL> alter user John identified by newpsw;

删除用户

使用drop user删除用户，关键字cascade删除用户模式中包含的数据对象。

删除用户John，并同时删除John拥有的所有表、索引等对象。

切换为system账户登录：

SQL> conn system/test123@test

删除John操作：

SQL> drop user John cascade;

测试John是否存在

SQL> conn John/newpsw@test;

查看自己的权限

- select * from user_sys_privs;

GRANT

- **1.GRANT 赋于权限**

  常用的系统权限集合有以下三个:

  CONNECT(基本的连接), RESOURCE(程序开发), DBA(数据库管理)

  常用的数据对象权限有以下五个:

  ALL ON 数据对象名, SELECT ON 数据对象名, UPDATE ON 数据对象名,

  DELETE ON 数据对象名, INSERT ON 数据对象名, ALTER ON 数据对象名

  GRANT CONNECT, RESOURCE TO 用户名;

  GRANT SELECT ON 表名 TO 用户名;

  GRANT SELECT, INSERT, DELETE ON表名 TO 用户名1, 用户名2;

grant

2.REVOKE 回收权限

REVOKE CONNECT, RESOURCE FROM 用户名;

REVOKE SELECT ON 表名 FROM 用户名;

REVOKE SELECT, INSERT, DELETE ON 表名 FROM 用户名1, 用户名2;

3.删除用户

drop user 用户名 cascade

4、设置用户密码登录后失效，并要求修改密码

alter user 用户名 password expire;

5、账户锁定和解锁

alter user 用户名 account lock; （锁定）

alter user 用户名 account unlock;（解锁）

序列sequence

- 序列是oracle专有的对象，它用来产生一个自动递增的数列
- 创建序列的语法：

```
create sequence seq_name
increment by n
start with n
maxvalue n|nomaxvalue 10^27 or -1
minvalue n|no minvalue
cycle|nocycle
cache n|nocache
```

- 实例：create sequence seq_empcopy_id start with 1 increment by 1;

序列

实际使用

create sequence 序列名称 start with 1 increment by 1;

序列的使用:

序列.nextval -->下个值

查看序列状态

select seq_score_id.currval from dual -->当前值

删除序列

drop sequence 序列名;

如:

insert into tb_course values(seq_course.nextval,'dsaf','dsaf);

序列

- 使用序列
  - select seq_empcopy_id.nextval from dual
  - insert into empcopy (empno,ename)

    values (seq_empcopy_id.nextval, ‘TEST’);
- 查看序列状态
  - select seq_empcopy_id.currval from dual
- 删除序列
  - drop sequence seq_empcopy_id;

SQL数据更新

- Sql的数据更新包括数据插入、删除和修改3个操作.
- 往表中插入数据的语句是insert语句，方式有两种，一种是元组值的插入，一种是查询结果的插入
- 元组值的插入语法如下：

```
INSERT INTO      table [(column [, column...])]
VALUES           (value [, value...]);
```

一次插入操作只插入一行

Insert语句

- insert into emp (empno,ename,job,mgr,hiredate,sal,comm,deptno)
  values(1111,'gao','clerk',7902,sysdate,10000,3000,40)
- 此处插入的元组中列的个数、顺序与emp的结构完全一致，因此表名之后的列名可以省略不写
- insert into emp
  values(2222,'gaohs','clerk',7902,sysdate,10000,3000,40)
- 可以只插入部分列
- insert into emp(empno,ename)
  values (3333,'xiaozhang')
- 但要求省略的列必须满足下面的条件：
  1.该列定义为允许Null值。
  2.在表定义中给出默认值，这表示如果不给出值，将使用默认值。
- 如果不符合上面两个条件，将会报错。不能成功插入。

Insert语句

可以用insert语句把一个select语句的查询结果插入到一个基本表中，语法如下：

```
Insert into tablename(column,..)
select * from tablename2
```

1. 创建一个临时表

```
create table temp
as
select * from emp
where 1 = 2
```

2. 执行插入

```
insert into ss  select * from emp;
```

例子

- create table test02 as
- select * from emp where 1=2
- create table test01 as
- select * from emp where 1=1
- insert into test02
- select * from emp where emp.deptno=10

DELETE 语句

- SQL的删除操作是指从基本表中删除元组，语法如下：

  DELETE [FROM] *table*

  [WHERE *condition*];

- 其语义是从基本表中删除满足条件表达式的元组
- Delete from table 表示从表中删除一切元组
- 如果想从表中删除所有的行，不要使用delete，可使用**truncate table** 语句，完成相同的工作，但是速度更快（没有事务）。

UPDATE 语句

- Update语句用于修改基本表中元组的某些列，其语法如下：

```
UPDATE      table
SET  column = value [, column = value] …
[WHERE      condition];
```

- 其语义是：修改基本表中满足条件表达式的那些元组的列值，需修改的列值在set子句中指出。

事务处理

- 事务（Transaction）是一个操作序列。这些操作要么都做，要么都不做，是一个不可分割的工作单位，是数据库环境中的逻辑工作单位。
- 事务是为了保证数据库的完整性
- 事务不能嵌套
- 在oracle中，没有事务开始的语句。一个Transaction起始于一条DML(Insert、Update和Delete)语句，结束于以下的几种情况：
  - 用户显式执行Commit语句提交操作或Rollback语句回退。
  - 当执行DDL(Create、Alter、Drop)语句事务自动提交。
  - 用户正常断开连接时，Transaction自动提交。
  - 系统崩溃或断电时事务自动回退。

Ddl语句执行自动提交事物

- insert into test02
- select * from emp where emp.deptno=10;

- create table test04 as
- select * from emp where 1=2

Commit & Rollback

- Commit表示事务成功地结束，此时告诉系统，数据库要进入一个新的正确状态，该事务对数据库的所有更新都以交付实施。每个Commit语句都可以看成是一个事务成功的结束，同时也是另一个事务的开始。
- Rollback表示事务不成功的结束，此时告诉系统，已发生错误，数据库可能处在不正确的状态，该事务对数据库的更新必须被撤销，数据库应恢复该事务到初始状态。每个Rollback语句同时也是另一个事务的开始。
- 一旦执行了commit语句，将目前对数据库的操作提交给数据库（实际写入DB），以后就不能用rollback进行撤销。
- 执行一个 DDL ，dcl语句或从 SQL*Plus正常退出，都会自动执行commit命令。
- savepoint test01
- rollback to test01

事物测试例子

```
insert into test02(ename,empno,deptno ) values('cai10',1010,10);
insert into test02(ename,empno,deptno ) values('cai20',1010,10);
select * from test02;
savepoint sp01
insert into test02(ename,empno,deptno ) values('cai30',1010,10);
insert into test02(ename,empno,deptno ) values('cai40',1010,10);
select * from test02;
rollback to sp01
commit;
```

事务的ACID属性

- 事务四大特征：原子性，一致性，隔离性和持久性。
- 1. 原子性（Atomicity）
- 一个原子事务要么完整执行，要么干脆不执行。这意味着，工作单元中的每项任务都必须正确执行。如果有任一任务执行失败，则整个工作单元或事务就会被终止。即此前对数据所作的任何修改都将被撤销。如果所有任务都被成功执行，事务就会被提交，即对数据所作的修改将会是永久性的。
- 2. 一致性（**Consistency**）
- 一致性代表了底层数据存储的完整性。它必须由事务系统和应用开发人员共同来保证。事务系统通过保证事务的原子性，隔离性和持久性来满足这一要求; 应用开发人员则需要保证数据库有适当的约束(主键，引用完整性等)，并且工作单元中所实现的业务逻辑不会导致数据的不一致(即，数据预期所表达的现实业务情况不相一致)。例如，在一次转账过程中，从某一账户中扣除的金额必须与另一账户中存入的金额相等。支付宝账号100 你读到余额要取，有人向你转100 但是事物没提交（这时候你读到的余额应该是100，而不是200） 这种就是一致性
- 3. 隔离性（**Isolation**）
- 隔离性意味着事务必须在不干扰其他进程或事务的前提下独立执行。换言之，在事务或工作单元执行完毕之前，其所访问的数据不能受系统其他部分的影响。
- 4. 持久性（**Durability**）
- 持久性表示在某个事务的执行过程中，对数据所作的所有改动都必须在事务成功结束前保存至某种物理存储设备。这样可以保证，所作的修改在任何系统瘫痪时不至于丢失。

提交或回滚前数据的状态

- 以前的数据可恢复
- 当前的用户可以看到DML操作的结果
- 其他用户不能看到DML操作的结果
- 被操作的数据被锁住，其他用户不能修改这些数据

提交后数据的状态

- 数据的修改被永久写在数据库中.
- 数据以前的状态永久性丢失.
- 所有的用户都能看到操作后的结果.
- 记录锁被释放,其他用户可操作这些记录.

回滚后数据的状态

- 语句将放弃所有的数据修改
  - 修改的数据被回退.
  - 恢复数据以前的状态.
  - 行级锁被释放.

常用数据类型

① number(x,y) ： 数字类型 ， 最长x位，y位小数

② varchar2 (maxlength) ： 变长字符串,这个参数的上限是32767字节

- 声明方式如下VARCHAR2 (L) ，L为字符串长度，没有缺省值，作为变量最大32767个字节

③ char(max_length)　　定长字符串　最大2000字节

④ DATE:日期类型（只能精确到秒。）

⑤ TIMESTAMP:时间戳（精确到微秒）

⑥ long：长字符串，最长2GB

了解类型

① CLOB：最大长度4G -->大对象很少使用:如果存在大对象，一般的解决方案存入文件地址（地址为程序所在应用服务器的相对路径）。

② BLOB：存二进制文件

数据库的对象

对象名称	描述
表	基本的数据存储对象，以行和列的形式存在，列也就是字段，行也就是记录
约束	执行数据校验，保证了数据完整性的
视图	一个或者多个表数据的逻辑显示
索引	用于提高查询的性能
Sequence	自增序列

数据库对象的命名规则

- 必须以字母开头
- 可包括数字和三个特殊字符（# _ $）
- 不要使用oracle的保留字
- 同一用户下的对象不能同名

ORACLE常用数据类型

数据类型	含义
Varchar2(n)	变长字符串，存储空间等与实际空间的数据大小，最大为4K，长度以字节为单位指定（注意中文字符）
Char(n)	定长字符串，存储空间大小固定
Number(p,s)	整数或小数，p是精度（所有数字位的个数，最大38），s是刻度范围（小数点右边的数字位个数，最大127）
Date /timestamp	年、月、日、时、分、秒

表的创建

- 标准的建表语法：

  CREATE TABLE [*schema*.] *table*

  (*column datatype* [DEFAULT *expr*] , ...

  );
- 在创建新表时，指定的表名必须不存在，否则将出错。
- 使用默认值：当插入行时如果不给出值，dbms将自动采用默认值。
- 在用Create语句创建基本表时，最初只是一个空的框架，用户可以使用insert命令把数据插入表中。

表的创建

- 设计要求：建立一张用来存储学生信息的表，表中的字段包含了学生的学号、姓名、年龄、入学日期、年级、班级、email等信息，并且为grade指定了默认值为1，如果在插入数据时不指定grade得值，就代表是一年级的学生
  - create table stu
    (id number(6),
    name varchar2(20) not null unique,
    sex number(1) not null,
    age number(3),
    sdate date,
    grade number(2) default 1,
    class number(4),
    email varchar2(50)
    );

表的创建

- 使用子查询创建表的语法

  CREATE TABLE *table* [*column*(, *column*...)]

  AS *subquery*;

  1. 新表的字段列表必须与子查询中的字段列表匹配
  2. 字段列表可以省略

- create table emp2 as select * from emp;

表结构的修改

- 在基本表建立并使用一段时间后，可以根据实际需要对基本表的结构进行修改
  - 增加新的列用“alter table … add …”语句

    alter table emp add address varchar(20)

    新增加的类不能定义为“not null”，基本表在增加一列后，原有元组在新增加的列上的值都定义为空值。
  - 删除原有的列用“alter table … drop…”语句，语法格式：alter table 表名 drop column 列名

    alter table emp drop column address
  - 修改字段“alter table...modify...”

    alter table emp modify(job varchar(50))

表结构的修改

- 在基本表不需要时，可以使用“drop table”语句撤消。在一个基本表撤消后，所有的数据都丢弃。所有相关的索引被删除

```
drop table emp cascade constraints
```

- 可以使用RENAME语句改变表名（视图），要求必须是表（视图）的所有者
  - RENAME old_name TO new_name

约束 constraint

- 当我们创建表的时候，同时可以指定所插入数据的一些规则，比如说某个字段不能为空值，某个字段的值（比如年龄）不能小于零等等，这些规则称为约束。约束是在表上强制执行的数据校验规则.
- Oracle 支持下面五类完整性约束:

1. NOT NULL非空
2. UNIQUE Key　　唯一键
3. PRIMARY KEY　　主键
4. FOREIGN KEY　　外键
5. CHECK　　自定义检查约束

约束 constraint

- Oracle使用SYS_Cn格式命名约束，也可以由用户命名
- 创建约束的时机
  - 在建表的同时创建
  - 建表后创建
- 约束从作用上分类，可以分成两大类：
  - 表级约束：可以约束表中的任意一列或多列。可以定义除了Not Null以外的任何约束。
  - 列级约束：只能约束其所在的某一列。可以定义任何约束。

约束简介

- 约束用于确保数据库数据满足特定的商业逻辑或者企业规则，如果定义了约束，并且数据不符
- 合约束，那么DML操作（INSERT、UPDATE、DELETE）将不能成功执行。约束包括NOT NULL、UNIQUE、PRIMARY KEY、FOREING KEY 以及CHECK等五种类型
- 定义约束
- 列级约束：column [CONSTRAINT constraint_name] constraint_type
- 表级约束：
- column ,...,
- [CONSTRAINT constraint_name] constraint_type (column,...)

约束简介

- 1.定义NOT NULL约束
- NOT NULL 约束只能在列级定义，不能在表级定义
- 例:
- CREATE TABLE emp01(
- eno INT NOT NULL,
- name VARCHAR2(10) CONSTRAINT nn_name2 NOT NULL,
- salary NUMBER(6,2)
-);

约束简介

- 列级约束：从形式上看，在每列定义完后马上定义的约束，在逗号之前就定义好了。
- carete table parent(c1 number primary key);
- create table child (c number primary key , c2 number references parent(c1));
- 表级约束： 从形式上可以看出与列级约束的区别了吧。
- create table child(c number , c2 number , primary key (c2), foreign key(c2) references parent(c1));
- 有些时候，列级约束无法实现某种约束的定义，比如联合主键的定义，就要用到表级约束:
- create table test(id1 number , id2 number, primary key(id1, id2));

主键约束（PRIMARY KEY）

- 主键约束是数据库中最重要的一种约束。在关系中，主键值不可为空，也不允许出现重复，即关系要满足实体完整性规则。
  - 主键从功能上看相当于非空且唯一
  - 一个表中只允许一个主键
  - 主键是表中能够唯一确定一个行数据的字段
  - 主键字段可以是单字段或者是多字段的组合
  - Oracle为主键创建对应的唯一性索引

建议命名

- 建议命名
- 约束_表名_字段 可以保证唯一性。 如果太长，可用缩写。
- 同一字段可以有多个约束，但是约束之间不要冲突

主键约束

- 主键可用下列两种形式之一定义
  1. 主键子句

     在表的定义中加上如下子句 primary key(列)
  2. 主键短语

     在主属性的定义之后加上primary key字样。
- 上述形式Oracle会自动命名约束，可自己给约束起名
  - create table t3(

    id number(4),

    constraint t3_pk primary key(id)

)

非空约束（NOT NULL）

- 确保字段值不允许为空
- 只能在字段级定义

```
CREATE TABLE employees(
        employee_id    NUMBER(6),
        name      VARCHAR2(25) NOT NULL,
        salary        NUMBER(8,2),
        hire_date      DATE CONSTRAINT emp_hire_date_nn NOT NULL
)
```

唯一性约束（UNIQUE）

- 唯一性约束条件确保所在的字段或者字段组合不出现重复值
- 唯一性约束条件的字段允许出现空值
- Oracle将为唯一性约束条件创建对应的唯一性索引

```
CREATE TABLE employees(
        id      NUMBER(6),
        name        VARCHAR2(25) NOT NULL UNIQUE,
        email           VARCHAR2(25),
        salary          NUMBER(8,2),
        hire_date       DATE NOT NULL,
        CONSTRAINT emp_email_uk UNIQUE(email)
);
```

CHECK 约束

- Check约束用于对一个属性的值加以限制
- 在check中定义检查的条件表达式，数据需要符合设置的条件

```
create table emp3
(    id number(4) primary key,
     age number(2)  check(age > 0 and age < 100),
     salary number(7,2),
     sex char(1),
     constraint salary_check check(salary > 0)
)
```

- 在这种约束下，插入记录或修改记录时，系统要测试新的记录的值是否满足条件

关系模型的三类完整性规则

- 为了维护数据库中的数据与现实世界的一致性，关系数据库的数据与更新操作必须遵循下列三类完整性规则：

1. 实体完整性规则

   这条规则要求关系中在组成主键的属性上不能有空值。

2. 参照完整性规则

   这条规则要求“不引用不存在的实体”。例如：deptno是dept表的主键，而相应的属性也在表emp中出现，此时deptno是表emp的外键。在emp表中，deptno的取值要么为空，要么等于dept中的某个主键值。

3. 用户定义的完整性规则

   用户定义的完整性规则反应了某一具体的应用涉及的数据必须满足的语义要求。

外键约束（FOREIGN KEY）

- 外键是表中的一个列，其值必须在另一表的主键或者唯一键中列出
- 作为主键的表称为“主表”，作为外键的关系称为“依赖表”
- 外键参照的是主表的主键或者唯一键
- 对于主表的删除和修改主键值的操作，会对依赖关系产生影响，以删除为例：当要删除主表的某个记录（即删除一个主键值，那么对依赖的影响可采取下列3种做法：

1. RESTRICT方式：只有当依赖表中没有一个外键值与要删除的主表中主键值相对应时，才可执行删除操作。
2. CASCADE方式：将依赖表中所有外键值与主表中要删除的主键值相对应的记录一起删除
3. SET NULL方式：将依赖表中所有与主表中被删除的主键值相对应的外键值设为空值

FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO)

[ON DELETE [CASCADE|SET NULL]] 如省略on短语，缺省为第一中处理方式。

约束的添加和撤销

- 可增加或删除约束，但不能直接修改

  alter table tablename

  增加

  add constraint con_name unique(col)

  删除

  drop constraint com_name [cascade]

查询constraint

- select constraint_name,constraint_type
- from user_constraints
- where table_name=upper('sxtstu05')
- 或者 where owner='SCOTT' 大写
- select constraint_name,constraint_type
- from user_constraints
- where owner='SCOTT'
- select constraint_name,column_name from user_cons_columns
- where table_name=upper('tablename')

索引

- 索引是为了加快对数据的搜索速度而设立的。索引是方案（schema）中的一个数据库对象，与表独立存放.
- 索引的作用：在数据库中用来加速对表的查询，通过使用快速路径访问方法快速定位数据，减少了磁盘的I/O
- Sql中的索引是非显示索引，也就是在索引创建以后，在用户撤销它之前不会在用到该索引的名字，但是索引在用户查询时会自动起作用。
- 索引的创建有两种情况

1. 自动: 当在表上定义一个PRIMARY KEY 或者UNIQUE 约束条件时,Oracle数据库自动创建一个对应的唯一索引.
2. 手动: 用户可以创建索引以加速查询

索引

- 开发中使用索引的要点：
  - 1. 索引改善检索操作的性能，但降低数据插入、修改和删除的性能。在执行这些操作时，DBMS必须动态地更新索引。
  - 2. 索引数据可能要占用大量的存储空间。
  - 3. 并非所有的数据都适合于索引。唯一性不好的数据（如省）从索引的到的好处不比具有更多可能值的数据（如姓名）从索引得到的好处多。
  - 4. 索引用于数据过滤和数据排序。如果你经常以某种特定的顺序排序数据，则该数据可能是索引的备选。
  - 5. 可以在索引中定义多个列（如省加城市），这样的索引只在以省加城市的顺序排序时有用。如果想按城市排序，则这种索引没有用处。

索引

- 在一列或者多列上创建索引.

  CREATE INDEX *index* ON *table* (*column*[, *column*]...);

- 下面的索引将会提高对EMP表基于 ENAME 字段的查询速度.

  CREATE INDEX emp_last_name_idx

  ON emp (ename)

- 通过DROP INDEX 命令删掉一个索引.
  - **DROP INDEX *index*;**
- 删掉 UPPER_LAST_NAME_IDX 索引.
  - **DROP INDEX upper_last_name_idx;**