Oracle数据库

注意：ORACLE中一个中文表示3个字符

ORACLE查询格式：

SELECT 分组字段 | \*

FROM 表，多个表

WHERE 筛选条件，注意内外连接连接waw

GROUP BY 分组字段，多个分组字段。。。

HAVING 分组后的筛选过滤条件

ORDER BY 排序字段 [ASC|DESC]

;

# Oracle（甲骨文）发展史：

·Oracle8/ Oracle8i(1CD):”i”表示的是Internet,表示Oracle开始进军网络；

·Oracle9i（3CD）：属于Oracle8i的稳定版本，现在依然大范围的使用；

·Oracle10g(630M)：使用了网格计算的方式，提升了数据库的分布式的访问性能；

·Oracle11g（1.7G）：属于Oracle10g的稳定版本；

·Oracle12C：指的是云服务的支持。

# 启动：

由于我们将Oracle的服务设置为了手工启动：

所以我们在每次启动的时候只需要启动两个服务即可：

## ·OracleOraDbllg\_hone1TNSListener:

是监听服务，如果要连接数据库的时候这个服务必须要启动。

## ·OracleServiceOracle:

数据库的实例服务，它的命名标准：OracleServiceSID，每当为系统增加一个数据库的实例的时候都会自动的出现一个类似的服务名称。

# Oracle卸载；

Oracle的卸载之后有可能无法立刻安装完成；

|-使用Oracle提供的卸载程序；

|-重新启动电脑，进入到安全模式（启动时按住F8）；

|-找到系统上的对应磁盘路径，并且删除；

|-运行注册表命令“regedit.exe”，搜索所有与Oracle的注册表路径然后删除。

实在不行就只能重新安装系统；

**我们通常使用括号进行命令的分割**

# SQLPlus命令：

所有固定的语法用大写，可变的就用小写

## SQL语句语法：

SELECT、FROM、WHERE、GROUP BY、HAVING、ORDER BY、INSERT、UPDATE、DELETE、CREATE、DROP、ALTER（变化、改变）、GRANT（赠款，拨款，准许）、REVOKE(撤销)。

## 两种方式：

### |-1、

窗口键加上R，然后输入sqlplus就可以进入到输入用户名密码的界面，

### |-2、

然后是命令行模式，直接cmd进入，然后输入

例如：sqlplus scott/tiger

开始查询后格式显示的混乱可以通过控制格式进行改善：

·设置每行显示的数据长度：SET LINESIZE 300; (300是一个长度)

·设置每页显示的数据条数：SET PAGESIZE 30; (每页显示的条数)

两个指令叫格式化指令。

COL是用来定义列宽的

SET是用来设置的

#### EG: （COL）：

col 字段名（列名） for [a8](https://www.baidu.com/s?wd=a8&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)（A8是指字符长度为8，如果是数字则不用啊）

## 编辑：

我们可以利用命令：“ed 文件名称”（如果不写后缀，默认的后缀就是\*.sql），即输入：ed wjn 不用加;这个符号，回车后会直接弹出一个记事本界面，然后在里面直接输入命令即可，（在经过调用后，就相当于直接输入命令行到界面了），输入完成后要关闭界面，否则会进入堵塞状态。

使用“@文件名称”（默认找到\*.sql的后缀名文件）也不用加上;这个符号

## 切换用户：

CONN 用户名/密码[AD SYSDBA]

如果现在使用的是sys用户登录，那么必须要写上AS SYSDBA，否则无法登录

例如：

首先：CONN system/manger

然后：CONN sys/change\_on\_install

(将无法进行连接)

若是在尾部加上AS SYSDBA

即：CONN sys/change\_on\_install AS SYSDBA即可进行连接

可以利用HOST命令来调用本机命令

例如：HOST echo helloworld

## 查看表的结构：

DESC 加表名;

## Scott表的结构：

Scott表一共4张表，但是3张表没有数据

# 简单的查询：

语句：

SELECT [DISTINCT]\*|列名称[别名],列名称[别名],…FROM 表名称[别名]；

简单查询是不能控制数据行的，只能够在SELECT语句里面控制列，如果是全部的查询列，那么就使用通配符“\*”完成。

简答的查询就是查询全部的数据数据行记录，但是但是由SELECT子句控制列。

先写FROM，后SELECT，只是为了方便，但是还是要SELECT在前。就实现摆出来，然后在套入的意思

## 注意：

如果要消除掉重复的数据，就可以使用“DISTINCT”完成，

EG：SELECT DISTINCT job FROM emp;

但是需要说明的是，如果查询的数据是多个列的数据，只有在这多个列的数据都相同的时候才可以消除。

EG:SELECT DISTINCT empno,job FROM emp;

查询雇员的编号、姓名、基本年薪

SELECT empno,ename,sal\*12 FROM emp;

**以下为别名定义：**

**SELECT empno,ename,sal\*12income FROM emp;**

**别名不建议使用中文**

EG：要求显示出每个雇员的编号、姓名、基本年薪（每个月可以领取15个月的工资，每个月有200元的饭食补贴、100元的汽车补贴、100元的电话补贴，每年还有5个月的高温补贴）

SELECT empno,ename,(sal\*15+(200+100+100)\*12+200\*5)income FROM emp;

# 常量的编写说明：

在使用SELECT子句查询数据的时候，除了查询呢列世界上也可以设置一些常量：

EG：

·如果常量是字符串，则要求使用“’”声明，例如：’hello’;

·如果常量是数字，则直接编写，例如：10；

·如果常量是日期，则按照日期风格格式编写，使用“xx日-xx月-xx年”，例如：’17-12月-80’;

## 直接查询常量：

SELECT’雇员’,empno,eame FROM emp;

## 注意：

在简单查询操作里面，如果有需要也可以在SELECT子语句里面使用“||”连接查询结果。

## 转换显示格式：

·效果：雇员编号：7369，姓名：SMITH，收入：800；

SELECT '雇员编号:' || empno || '，姓名:' || ename || ',收入：' || sal info FROM emp;

# 限定查询（即WHERE）：

（WHERE进行行控制，SELECT进行列控制）

## 运算符：

·关系运算符：>、<、>=、<=、<>(!=)；

·逻辑运算符：AND、OR、NOT；

·范围运算符：BETWEEN…AND;

·谓词范围：IN、NOTIN;

·空判断：IS NULL、IS NOTNULL;

·模糊查询：LIKE。

·eg：查询所有基本工资高于1500的雇员信息

SELECT \* FROM emp WHERE sal>1500;

查询职位不是销售人员，指的是针对于数据行过滤；

Eg: 查询工作不等于销售的

SELECT empno,ename,job FROM emp WHERE job<>’SALESMAN’;

BETWEEN…AND可以用逻辑运算符表示

## 注意：

**查询日期：**

SELECT \* FROM emp WHERE hiredate BETWEEN ’01-1月-1981’AND’31-12月-81’;

空在数据库上解释为不确定的内容。在数字列上使用NULL那么绝对不表示0.

空的操作只能够使用IS NULL 或者是IS NOT NULL(NOT IS NULL)表示。

## EG：判断是否为空的操作（查询雇员佣金信息）

SELECT \* FROM emp WHERE NOT comm IS NULL;

SELECT \* FROM emp WHERE comm IS NOT NULL;

## IN操作符：

例子：

SELECT \* FROM emp WHERE empno IN(7369,7566,7788,9999);

SELECT \* FROM emp WHERE empno=7369 OR empno=7566 OR empno=7788 OR empno=9999;

以上举例，即可获知在指定范围里面使用IN，那么如果不在指定的范围之中可以使用NOT IN操作。

**在使用NOT IN 进行范围判断的时候，如果范围里包含有NULL，那么不会有任何的结果返回。**

使用NOT IN的目的是为了查询部分数据行，但是如果有了NULL（某些数据永远不可能为空），就变成了查询全部了。（这样就宣判了服务器的死刑），若数据量太多，内存家在不过来，就会导致崩溃

## 模糊查询：

·“\_”：匹配任意的一位字符；

·“%”：匹配任意的零位、一位或多位字符。

### 范例：查询姓名是以字母A开头的雇员信息

SELECT \* FROM emp WHERE ename LIKE ‘A%’;

### 查询第二字母为A的雇员信息

SELECT \* FROM emp WHERE ename LIKE '\_A%';

### ·查询包含字母A（任意位置）的雇员信息

SELECT \* FROM emp WHERE ename LIKE '%A%';

### 注意：

SELECT \* FROM emp WHERE ename LIKE '%%';

虽然以上的代码可以查询全部数据，但是与与不使用WHERE子句相比，不使用WHERE子句的查询一定性能更好。

WHERE是要一个个进行盘查，但是这个不用

## 注意：

WHERE子句一般都写在FROM子句之后，但是紧跟着FROM子句之后执行的；

SELECT子句要落后于WHERE子句执行，所以在SELECT子句中定义的别名，无法在WHERE子句里面使用；

使用NOT IN查询时，查询范围里面不允许出现NULL，否则不会有任何查询结果返回；

数据库的的查询通常是按照数据的升序来保存的。

# 排序：

|  |
| --- |
| |-③控制显示的数据列:SELECT[DISTINCT]\*|列名称[别名],列名称[别名];  |-①确定数据来源:FROM 表名称[**别名**]  |-②确定满足条件的数据行:[WHERE 过滤条件(s)]  |-④针对查询结果进行排序[ORDER BY字段 [ASC|DESC]];  （不写也是ASC，默认升序） |

EG：(SAL是工资字段)

SELECT \* FROM emp ORDER BY sal DESC;

## EG:

要求按照工资由高到低排序，如果工资相同，则按照雇佣日期由早到晚排序。

**SELECT\* FROM emp ORDER BY sal DESC ,hiredate ASC;**

**在整个SQL查询的结构之中哪个，只有ORDER BY子句可以调用SELECT子句中定义的别名。**

|  |  |  |
| --- | --- | --- |
| No. | 函数名称 | 描述 |
| 1 | 字符串UPPER(列 | 字符串) | 将传入的字符串变为大写字母的形式 |
| 2 | 字符串LOWER(列 | 字符串) | 将传入的字符串变为小写字母的形式 |
| 3 | 字符串INITCAP(列 | 字符串) | 开头首字母大写，其他字母全部变为小写 |
| 4 | 字符串LENGTH(列 | 字符串) | 取得指定字符串的长度 |
| 5 | 字符串SUBSTR(列 | 字符串,开始索引,[长度]) | 进行字符串截取，如果字符串没有设置长度，表示从开始索引一直截取到结尾 |
| 6 | 字符串REPLACE(列 | 字符串，旧内容,新内容) | 将指定字符串的数据以新数据替换旧数据 |

# 单行函数：

**在Oracle里面所有的函数要想进行验证，也必须要编写SQL语句，为了方便用户进行验证或者是一些不需要查询表的一些操作。提供了一个叫 dual 的表**

## 使用例子：

1. 转大小写：

SELECT UPPER(‘hello’) FROM dual;

结果：

UPPER('HEL

----------

HELLO

SQL>

1. 转小写：

SELECT LOWER(ename)

FROM emp

;

注意看这两个结果的区别：

在Oracle数据库里面本身的数据是区分大小写的，但是为什么要提供大小写呢？

在实际的开发之中啊，会发现许多的代码本身是不区分大小写关系的，所以用户在信息查询的时候往往不会考虑到大小写的问题。

## 例子说明：

### 比如说查询一个一个叫smith的：

//查询一个叫smith的

SELECT \*

FROM emp

WHERE ename='SmitH'

;

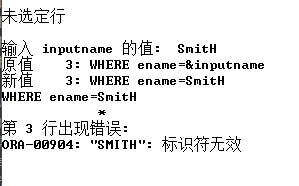
//上面是会报错的

### //另一种方式

SELECT \*

FROM emp

WHERE ename=&inputname;



请输入的时候，要是没有输入到正确的（一个字母大小写也是），就会报错。

### 但是看另外一种方式

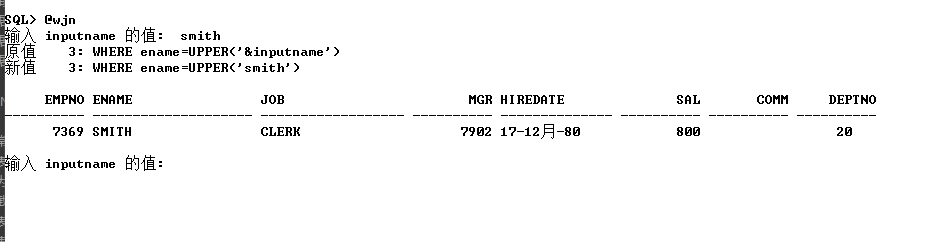
//第二种

SELECT \*

FROM emp

WHERE ename=UPPER('&inputname')

;



将代码转换为，这样就可以避免查询错误

## INITCAP函数说明：

//首字母大写INITCA, 利用新旧作比较

SELECT ename,INITCAP(ename)

FROM emp

;

控制的是首字母大写，其余全部改小写

## LENGTH取得字符串的长度：

可以，

在Oracle中，如果索引设置的是0，他也会按照1来进行处理，若设置的是负数，就是指倒数的后几位

# 数值函数：

|  |  |  |
| --- | --- | --- |
| NO. | 函数名称 | 描述 |
| 1 | 数字ROUND(列 | 数字 [,小数位]) | 实现数据的四舍五入 |
| 2 | 数字TRUNC(列 | 数字[,小数位]) | 数显数据的截取，即不进位 |
| 3 | 数字MOD(列 | 数字,列 | 数字]) | 求模（取余） |

EG：

**ROUND函数用起来很奇怪，尽量别用dual表来进行验证**

## TRUNC例子：

//数值函数，截取，观察区别

SELECT TRUNC(729.5671234),

TRUNC(729.5671234,2),

TRUNC(729.5671234,-2)

FROM dual;

以后中只要看到ROUND就是指四舍五入

# 日期函数：

如果想处理任何的日期，那么都有一个基本的前提，就是必须知道当前的日期是什么。如果想要取得当前的日期时间，在Oracle里面，专门提供了一个伪列“SYSDATE”（SYSTIMESTAMP）

## 获取时间：

EG:

//观察下列日期的区别

SELECT SYSDATE FROM DUAL;

SELECT SYSTIMESTAMP FROM dual;

结果：

SYSDATE

--------------

07-10月-18

SYSTIMESTAMP

-----------------------------------------------------------

07-10月-18 12.23.59.307000 上午 +08:00

## SYSDATE是伪列：可以直接SELECT

## 3个日期的操作公式：

·日期+数字=日期（表示若干天之后的日期）；

·日期-数字=日期（表示若干天之前的日期）；

·日期-日期=数字（天数）；

**天数只有日期减日期**

|  |  |  |
| --- | --- | --- |
| NO. | 函数名称 | 描述 |
| 1 | 日期ADD\_MONTHS(列 | 日期，月数) | 在制定的日期上增加若干个月之后的日期 |
| 2 | 数字MONTHS\_BETWEEN(列 | 日期，列 | 日期) | 返回两个日期之间的所经历的月数 |
| 3 | 日期LAST\_DAY(列 | 日期) | 取得指定日期所在月的最后一天 |
| 4 | 日期NEXT\_DAY(列 | 日期,星期 X) | 返回下一个指定的一周时间数对应的日期 |

## 四个日期处理函数：

### 例子：

**//增加指定的月份**

SELECT ADD\_MONTHS(SYSDATE,4),

ADD\_MONTHS(SYSDATE,20),

ADD\_MONTHS(SYSDATE,300)

FROM dual;

在进行月数增加的时候是不会限制数据的大小的（只要别太大都行）。

**//计算所有雇员到今天为止雇佣的月数**

Hiredate是雇佣日期，注意前后

SELECT empno,ename,hiredate,MONTHS\_BETWEEN(SYSDATE,hiredate)

FROM emp;

**//计算当前时间所在月的最后一天日期；**

SELECT LAST\_DAY(SYSDATE)

FROM dual;

**//要求查询所有在雇用所在月倒数第三天雇佣的雇员信息**

SELECT empno,ename,hiredate,LAST\_DAY(hiredate)-2

FROM emp

WHERE hiredate=LAST\_DAY(hiredate)-2

;

注意看是减几，3天所以是-2

一般而言，日期函数的操作过程是有些麻烦，但是有一个前提：利用日期函数操作的日期是最准确的。

**//NEXT\_DAY函数**

SELECT NEXT\_DAY(SYSDATE,'星期二')

FROM dual

;

注意观察，中间的是中文星期二

# 转换函数：

数字型、字符串型、日期型三类数据相互转换

|  |  |  |
| --- | --- | --- |
| NO、 | 函数名称 | 描述 |
| 1 | 字符串TO\_CHAR(列 | 日期 | 数字，转换格式) | 将日期或数字格式化为指定结构的字符串 |
| 2 | 日期TO\_DATE(列 | 字符串，转换格式) | 按照指定的转换格式编写字符串后将其变为日期型数据 |
| 3 | 数字TO\_NUMBER(列 | 字符串) | 将字符串变为数字 |

·日期：年（yyyy）、月（mm）、日（dd）

·时间：时（hh、hh24）、分（mi）、秒（ss）;

·数字：任意数字（9）、本地货币符号（L）

## //将日期显示格式化

SELECT TO\_CHAR(SYSDATE,'yyyy-mm-dd') FROM dual;

SELECT TO\_CHAR(SYSDATE,'yyyy-mm-dd hh24:mi:ss') FROM dual;

日期拆分也是用TO\_CHAR

EG: TO\_CHAR(SYSDATE,’YYYY’)或者TO\_CHAR(SYSDATE,’MM’)

除了转换日期以外还可以转换字符串

# 通用函数：

|  |  |  |
| --- | --- | --- |
| NO、 | 函数名称 | 描述 |
| 1 | 数字 NVL(列 | NULL,默认值) | 如果传入的内容是null，则使用默认数值处理，如果不是空则使用原始数值处理 |
| 2 | 数值类型 DECODE(列 | 字符串 | 数值,比较内容1,显示内容1,比较内容2,显示内容2,…[，默认显示内容]) | 设置内容会与每一个比较内容进行比较，如果内容相同，则会使用现实内容进行输出，如果都不相同，则是用最后的默认显示内容进行进行输出 |

## 处理NULL函数

要求查询每个雇员的编号、姓名、基本工资、佣金、年薪

SELECT empno,ename,sal,comm,(sal+comm)\*12 FROM emp;

要求年薪不为0，则：

//要求查询每个雇员的编号、姓名、基本工资、佣金、年薪

SELECT empno,ename,sal,comm,(sal+comm)\*12,NVL(comm,0) FROM emp;

## DECODE()函数

//DECODE函数

SELECT ename,job,DECODE(job,'CLERK','办事员','SALEMAN','销售','经理','MANGER','经理','---')

FROM emp；

# 多表查询：

笛卡尔积问题

多表查询的结果可以说其实就是一张临时表

## 建议表使用别名：

SELECT \* FROM emo e,dept p WHERE e.deptno=d.deptno;

e p是两个表的别名

**多表查询要注意笛卡尔积问题**

实现多表查询一定要有一个前提：**要关联的数据表一定要存在有关联字段，或者是关联条件，如果不存在这些要求的，一定不能使用多表查询。**

在查询每增加一张表就要消除笛卡尔积。

即：e.deptno=d.deptno;

SELECT e.empno,e.ename,e.job,e.sal,d.dname,d.loc

取一个表，共有属性也要写出来

一般情况下，如果是多个消除笛卡尔积的条件都会使用AND进行连接

# 表的连接：

·内连接（等值连接）：所有满足条件的数据都会被显示出来；

·外连接（左外连接、右外连接、全外连接）：控制左表与右表的数据是否全部显示。

## 外连接：

让等值判断左右两百斤有一遍的数据可以全部显示出来，如果相爱那个药实现外连接

## 左外连接：

用（+）来实现， 这个+号可以这样来理解： + 表示补充，即哪个表有加号，这个表就是匹配表。**如果加号写在右表，左表就是全部显示，所以是左连接。**

## 右外连接：

用（+）来实现， 这个+号可以这样来理解： + 表示补充，即哪个表有加号，这个表就是匹配表。**如果加号写在左表，右表就是全部显示，所以是右连接。**

## 全连接：

**左表和右表都不做限制，所有的记录都显示，两表不足的地方均为NULL。 全外连接不支持（+）写法。**

## 关联自身的查询：

自身的查询去除笛卡尔积会导致数据显示不完全，

然后用外连接来处理：

即：

SELECT e.ename,e.job,m.ename FROM emp e,emp m WHERE e.mgr=m.empno**(+);**

# 分组函数：（分组查询）

WHERE子句是不允许使用分组函数的，

GROUD BY实现分组，**HAVING实现分组后的过滤**

WHERE之前不允许使用GROUD BY，

**如果子查询在HAVING子句里面出现，那么只有一种情况，子查询返回的是单行单列数据，并且需要进行统计函数计算的时候**

·--**having 子句中的每一个元素也必须出现在select列表**中。有些数据库例外，如oracle.  
·--having子句和where子句都可以用来设定限制条件以使查询结果满足一定的条件限制。  
·--having子句限制的是组，而不是行。where子句中不能使用聚集函数，而having子句中可以。

**HAVING的子查询返回的结果一定不会特别复杂**

# 统计函数：

# 子查询：

**一般在SELECT子句里面使用子查询（）**

能用子查询寻也不能说就要用子查询，反正是很常出现的

子查询可以出现在任何位置

一个查询语句中

子查询可以说在任何该出现的位置上

子查询必须要用括号声明

## ANY操作：

一共分三种形式：

=ANY：功能与IN一样

>ANY：比子查询返回的最小值要大

<ANY：比子查询返回的最大值要小

不包含本身

### EG：

SELECT \*FROM emp

WHERE sal>ANY(

SELECT sal FROM emp WHERE job=’MANAGER’);

红色为子查询，已用括号括起

## ALL操作：一共分两种形式

·>ALL：比子查询返回的最大值还要大；

·<ALL：比子查询返回的最小值还要小；

不包含本身

# 表的创建：

## 复制表：(将EMP表复制到myemp)

CREATE TABLE myemp AS SELECT \* FROM emp;

# 增加数据：

## 完整型插入格式：

INSERT INTO 表（数据列，逗号隔开）

VALUES(AAAA,BBB,TO\_DATE(‘CCC’),’ DDD’ );

## 简写型：

（省略掉增加数据列的名称（所编写的数据的顺序一定要和表中定义的列顺序一致））

Eg:INSERT INTO myemp

VALUES(9999,’李四’,’清洁工’,7369,TO\_DATE(‘1979-10-10’,’yyyy-mm-dd’),9000.0,10.0,40)

·当数据增加成功之后，所有的数据库都会提示用户，当前更新语句所影响的数据行数量。

### 注意：

如果只是从代码代码长度上来看，基本肯定会认为不写列名称是简单的但是从代码的维护角度来说，强烈建议使用完整格式，不然以后维护会相当麻烦，不然的话会扣工资

### 数据设置空值：

Null代替，

如果没设置默认值，他就是空。

如果编写的是完整的格式，那么就可以通过不写字段的方式操作，如果写的是省略格式，那么就必学明确的设置null，否则会出现数据个数不一致的错误。直接提示你没有完整的值

# 数据修改：（更新数据）

## 格式：

**UPDATE 表名称 SET 字段1=值，字段2=值，…WHERE 更新条件;**

在使用更新的时候，WHERE子句里面的内容可以使用IN、BETWEEN。

更新的内容也可以子查询用子查询

**更新操作一定要谢谢WHERE子句，不然就是更新全部，更新全部是一个灾难，注意NOT IN 是不能接NULL的**

# 数据删除：

DELETE FROM 表名称 WHERER 子句;

删除了以后再删除同样的东西会返回已删除0行

EG:

DELETE FROM myemp WHERE empnno IN(7782,7902);

删除7782和7902的雇员信息。

任何系统之中的删除操作都属于及其危险的操作

·物理删除：直接执行DELETE FROM语句，彻底的从表中删除记录；

·逻辑删除：增加一个逻辑字段，例如FLAG字段，如果flag=0表示没有被删除，如果真执行的是删除操作那么久相当于修改了flag字段的内容，同时在每次查询的时候只需要增加一个WHERE限定条件WHERE 限定条件 (SELECT \* FROM … WHERE flag=1)。

只有一些无关紧要的信息才用被删除，（垃圾）日志之类的。

# 事务处理：

Sql:**rollback，基础数据库的回档**

现在有一种业务：

**A同学要付给你B同学1000元**

·第一步：从A同学上减少1000元

·第二步：在B同学上增加1000元

·第三步：A同学给出50元转账费用

如果说A第二步出现了错误B同学没有收到1000元，那么A同学的也要还原。

也就是这三条要一起进行更新操作，属于同一个操作业务，为了保证三个更新的同时性，即是说要么同时完成，要么同时失败，

## Session概念：

（会话，表示为宜的）在ORACLE中，么一个登录到数据库上的用户都会给分配一个session，即每一个session都表示不同的用户，每一个session都有独立操作的能力。

任何Session的事务处理上都可以使用两个命令：

## ·commit：

**事物提交，即：如果执行了多条更新操作，那么只有执行了commit之后更新才会真正的发出。在没有执行commit之前，所有的更新操作都会保存在缓冲区之中。**

## ·rollback：

事务回滚操作，即：如果发现更新的操作又问题，则回复所有的操作，以保证原始数据不被破坏。

### 每个Session操作完后，

首先是保存到缓冲区（保存所有的更新），若有问题则rollback，提交更新，COMMIT操作之后缓冲区将被清空，就到物理层了

EG：

先执行DELETE FROM语句后，在执行SELECT语句，记录是不能显示的。但是会存在缓冲区，所以要进行rollback或者是commit，执行了commit之后是不能再执行了，真正的修改了物理层，回滚也没有用。

# 死锁：

SessionA和SessionB共同执行一个操作的时候，就会死锁，就是两个一起抢占资源。但是都不能占有资源，就会出现死锁。第二个会一直等待这第一个一直更新完毕后才会发出更新。这时候没有commit没有rollback,就会一直在缓冲区，第二个session执行不了，就造成了死锁。

# 注意：

DDL不支持事务处理，而且在Oracle里面，会出现严重的问题：所有未提交的更新事务将会被自动提交；

# 数据的伪列：

## ROWNUN:（行号）

与程序开发有关：

SYSDATE获取时间的时候

自动分行

其实就是每行前面隐藏的编号，但又不是排序，且是**动态生成**的。

作用：（对数据分析很有用）

·取出第一行记录；

SELECT \*FROM EMP WHERE ROWNUM=1;

·取出第N行记录；

SELECT \*FROM EMP WHERE ROWNUM<=N;

BETWEEN …AND …是不能的。

ROWNUM是自动生成的，BETWEEN是要到实体数据中查的。

**如果真的想中间截断的取出的话，可以将ROWNUM作为子查询，应为这样取出的是行数据，然后我们可以作为子查询的条件；**

**EG:**

**SELECT \* FROM**

**(SELECT empno,ename,job,sal,hiredate,ROWNUM rn FROM EMP WHERE ROWNUM<=10**

**)**

**WHERE temp.rn>5;**

**即可实现**

**还可以实现分页功能。**

## 部分记录的显示：

往往取出第一行来判断这个是什么表，有什么数据。而不是全取；

就像是刚来到公司里，然后上来就上手数据库，直接把所有的数据全给你弄出来了

## ROWID:

与分析数据库对象有直接的联系

表中所有数据行保存后都会有一个物理地址编号（且唯一），然后可以通过ROWID找到

EG:

AAAR3qAAEAAAACHAAA

·数据的对象编号：AAAR3q 那个图书馆

·数据保存文件编号：AAE 图书馆的那个区

·数据保存的块号：AAAACH 哪个区的那个书架

·数据保存行号：AAA 那个书架的那个行

可以通过ROWID来找到数据 ，在WHERE子句中用

# 数据分页：

格式：

SELECT \* FROM

(//子查询

SELECT内容,ROWNUM rn FROM EMP

WHERE ROWNUM<=**N currentPage\*lineSize=N**

ORDER BY

GROUD BY等

)

WHERE temp.rn>M; (**CurrentPage-1**)**\*lineSize=M**

**M你每页显示的数据量**

**N啦实现**

即可实现

# 创建表：

格式

CREATE TABLE 表名称(

列名称 类型 [DEFAULT 默认值],

列名称 类型 [DEFAULT 默认值],

列名称 类型 [DEFAULT 默认值],

)

默认时间的话，就

DEFAULT SYSDATE

# 表的重命名：

在Oracle里面提供有能记录下所有对象的信息，提供了数据字典的概念

·用户级别：user\_\*开头，指的是一个用户可以使用的数据字典；

·管理员级别：dba\_\*开头，指的是由数据库管理员使用的数据字典;

·全部级别：all\_\*开头，表示不管是用户还是管理员都可以都可以使用。

EG:SELECT \* FROM user\_tables。

查询的是表的各种乱七八糟的信息，详细的资料

# 截断表：

DELETE一张数据表的记录，那么第一个反应就是DELETE，但是在使用DELETE的时候严格来说表所占有的资源（约束、索引）并不会马上就释放掉，因为可以rollback，现在有一种方式可以直清除掉所占用的全部资源，那么就使用截断表的操作。

## 语法：

TRUNCATE TABLE 表名称;

注意，这样rollback也是回不来的。

# 复制表：

## 语法：

CTEATE TABLE 表名称 AS 子查询;

复制一张什么都没有的表：

SELECT \* FROM emp WHERE 1=2;

# 删除表：

DELETE TABLE 表名称;

删除表是不能一次性全部删除的。

只能一张一张的删

# 闪回技术：

## 查看回收站：

SELECT \* FROM user\_recyclebin;

但是一般不这么干，

EG：从回收站查看3个基本信息

SELECT original\_name,object\_name,droptime FROM user\_recyclebin;

## 闪回 格式：

FLASHBACK TABLE 表名称 TO BEFORE DROP;

## 强制删除数据表：

DROP TABLE deptstat PURGE;

## 删除回收站里的一张数据表：

PUGER TABLE 表名;

## 清空回收站：

PUGER RECYLEBIN;

# 修改表结构：

·脚本文件的后缀名称发是\*.sql;

·要编写删除数据表的语法；

·创建数据表的用法；

·测试数据；

·执行事务提交。

## 格式：

ALTER TABLE 表名称 MODIFY(

列名称 类型 [DEFAULT 默认值],

列名称 类型 [DEFAULT 默认值]

)

表出问题了，最好是表删除，然后重新建

对于数据库的操作对象，只有三类语法：

创建对象，删除对象，修改对象

# 约束：

约束保证书了数据表数据的完整性。

六种（数据类型、非空约束、唯一约束、主键约束、检查约束、外键约束）

所有的月数约束也是一个对象

EG：查看数据库所有的约束，

SELECT \* FROM user\_constraints;

只能告诉用户那张表上有约束，但是并不知道这个约束对象对应的的还是那个列

想要知道约束对象是哪一个列的可以使用user\_cons\_columns;

我们可以在设置约束的时候为约束条件设置一个对象名字。

格式：约束简写+下划线+字段

EG：

在建表的时候字段下面，补上：

CONSTRAINT 约束名字缩写\_字段 约束(字段).

复合主键EG：

CONSTRAINT pk\_mid\_name PRIMARY KEY(mid,name)

这样如果还是出现错误的话就会很明显的显示出报错的位置，不会再。

## 非空约束：(NOT NULL,NK)

非空约束指的就是数据表中的某一个列的内容不允许为空。就是在建表的是后，数据字段后面加个NOT NULL

## 唯一约束：（UNIQUE,UK）

在后面加上UNIQUE，表示的是：在该数据段上不允许保存重复内容。

空是不受重复的限制的。

## 主键约束：（PRIIMARY KEY,PK）

主键约束=唯一约束+非空约束，既不能否重复，也不能够为空，例如：身份证号。主键使用PRIMARYKEY表示（PK）。通常一张表只会有一个主键，但是有时候也会设置有多个主键。这样就叫复合主键。

## 检查约束：（CHECK,CK）

就是指在进行数据更新操作前设置一些过滤条件，满足此条件的数据就实现更新。

EG:

CONSTRAINT ck\_age CHECL(age BETWEEN 0 AND 250);

注意：检查约束多了也不好，用户量一大的时候就速度很明显的就出来了

## 外键约束：（FOREIGN KEY，FK）

他是作用在两张数据表上的约束。父表和子表的关系，例如：

dept-emp,所以在子表emp里面就设置有deptno.表示和哪一个父表的关系关联

EG:

一个人有多本书

两个表：

--建立人表

CREATE TABLE member(

mid NUMBER,

name VARCHAR2(20) NOT NULL,

CONSTRAINT pk\_mid\_PRIMARY KEY(mid)

) ;

--建立书表

CREATE TABLE book(

bid NUMBER,

title VARCHAR2(20),

mid NUMBER,

CONSTRAINT pk\_bid\_PRIMARY KEY(bid)

**CONSTRAINT fk\_mid\_FOREIGN KEY(mid) REFERENCES member(mid)**

) ;

在BOOK里面创立外键

外键约束在这里就是说，book子表中的mid的取值范围应该由父表中member中的mid来定。 然后可以通过book表的mid可以找到人表中的人。理论上就是：

控制子表中某一列的内容 与父表中的数据范围相匹配，，

使用FOREIGN KEY表示。

报错就会说违反完整约束条件（SCOTT.FK\_MID）-未找到父项关键字

**外键的约束是 对于两张数据表的**，增删改，都会有联系

### 限制一、

如果表中存在有外键关系，在删除父表前一定要先删除子表，不然是不给删除的，

假如：两表A和B，A表有B表的外键，B表有A表的外键，DROP是不能强制删除的。

**强制删除：**

**DROP TABLE member CASCADE CONSTRAINT;**

不在关注子表是否存在，父表直接强制删除，如果不到万不得已，不要使用此类操作。

### 限制二、

父表中作为子表关联的外键字段，**必须设置为主键约束主键约束或者是唯一约束**否则无法成为子表的外键。

### 限制三、

默认情况下，如果父表记录中有对应的子表记录，那么父表 记录无法被删除。先删除子表数据在删除父表数据。

有一种方法，希望父表一删除对应的子表，数据就自动删除完了，就是数据的级联操作。

## ·数据的级联删除：ON DELETE CASCADE;

在外键后面加ON DELETE CASCADE;

## 数据的级联更新：ON DELETE SET NULL完成

一删除完，然后级联的自动设置NULL

## 修改约束：

**已经有数据了**，但是还要为其中增加约束

ALTER TABLE 表名称 ADD CONSTRAINT 约束名称 约束类型(字段)[选项]:

EG:为表增加主键

ALTER TABLE 表名 ADDCONSTRAINT 主键名字 PRIMARY KEY(字段名);

然后要先删除才可以继续增加。

以上的语法可以用检查、唯一、主键、外键四类约束上，但是不能用在非空约束上。如果 真的要使用非空约束，只有在修改表结构上完成。

# 数据字典：

user\_views

只要是在ORACLE中说是对象的，都会保存在数据字典中

# 序列：

就是 自动增长列

序列也是数据库对象，只要是对象就需要用哪个CREATE来创建

格式：

CREATE SEQUENNNCE 序列名称

[INCREMENT BY 步长][START WITH 开始值]

[MAXVALUE 最大值|NOMINVALUE]

[CYCLE | NOCYCLE] 循环或者不循环

[CACHE 缓存数据 | NOCACHE];

大多数时候都只会创建默认序列，默认序列就是指最小值为0，并没有最大值，开始值为0；

·序列已经创建成功，随后就需要去使用此序列，对于序列的使用可以采用两个伪列操作：

·**序列对象**.nextval：表示进行序列的增长，每调用一次，序列加上指定的步长

·序列对象.currval：表示取得当前的序列内容，不管如何调用，序列内容不发生改变。

**当序列对象创建完成之后，严格来讲序列是无法直接执行currval的，必须要先执行nextval之后才可以执行。不然报的错是对象未执行定义。**

只要系统提供了缓存的概念，那么基本上都是指的提升代码运行性能的一种手段。

如果使用缓存可能会出现跳号的问题。

这个是正常的问题 ，数据库很少说连着的

序列实现队列 的自动增长操作，要在INSERT语句里面完成，不能再数据 库的构建中完成

EG:

INSERT INTO mytab(id,name) VALUES (自己创建的序列对象 .nextval,’1111’)，

因为此时是在一个SESSION下的操作，还算在正常，如果以后出现顺序不对是正常的 现象

删除序列就是：

DROP SEQUENCE myseq;

缓存个数乘与步长

淘宝的一长串 订单就是

START WITH 1000000000000000000000;

然后逐步累加上去的。

EG:

设置循环：在

CREATE SEQUENCE myseq

CREATE SEQUENCE myseq

INCREMENT BY 2 --步长

START WITH 1

MINVALUE 1

MAXVALUE 9

CYCLE CACHE 3;

# 同义词：

# 视图的定义与使用：

所谓的视图就是包装了复杂查询的SQL语句对象。

语法：

CREATE[OR REPLACE]VIEW 视图名称 AS 子查询；

然后**只需要查询视图**就可以实现数据查询的操作了。

视图一定会存在的，但是是一个理想的状态

视图保存的是查询结果的数据，是不能修改的；

OR REPLACE ：

就是

视图可以更新。

WITH CHECK OPTION;

一般不直接替换视图数据。

创建视图的时候使用WITH READ ONLY;

就是为了保护视图，表示只读视图。

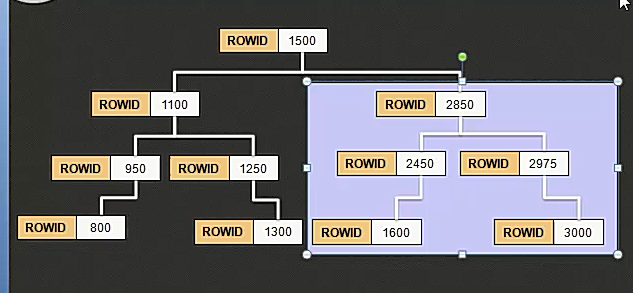
更新视图的非创建条件：

更新视图把数据改了就出现问题了。

# 索引：

set AUTOTRACE ON;打开跟踪器，

索引的作用就是不要进行全表扫描；



此时不用查询全部的数据记录，而只需要根据数值不叫查询有限多个数据即可 ，然而这就是索引的支持，也就是说如果使用了索引，那么只需要在数据之外维持一个内存空间，这个空间保留有这个树的排序关系。

## EG：创建索引

CREATE INDEX 表名\_（字段）\_ind（即是索引名称） ON 用户**.**表(字段);

之后再使用跟踪器就会显示的是不是全表扫描，在该表的某一个字段中定义了索引，但是在其他字段中没有定义，查询其他字段的时候，还是会执行全表扫描。

通过分析，索引可以提升是数据的查询性能，索引使用扫描的的并不是数据表，而是扫描索引数据（B\*Tree）。

假如工资作为索引的例。

如果是雇员的工资数据改变了，所以索引树的内容一定要发生改变。

每一次树的改变一定会消耗掉大量的内存空间。如果表中的数据要被频繁修改的话，那么索引就变成了彻底的拖累。

## 假如：

现在有一个数据库，里面的数据 平均每s更新2~3次，但是由于信息检索的需要，此数据库必须保持高性能的数据快速检索的需要与数据统计的需要，改怎么解决。

### 矛盾点：

如果想要更新迅速，那么不能够使用索引，如果想要检索迅速必须要使用索引。

·方式（一）：装备一台超级电脑；

·方式（二）：性能和速度是不能兼得的，此时只有牺牲实时性，就是延缓新数据的统计，在数据库里面可以准备出两张表，有一张表专门留给用户进行数据的更新使用，并且记录好更新的日志。

而另外一张数据表，可以设置索引，实现信息的快速检索。可以在夜间的时候或者是人少的时候实现数据的汇总、统计。

# 用户管理：

SQL中DCL是数据库的控制语言，使用DL

1. 连接到超级管理员

CONN sys/change\_on\_install AS SYSDBA;

1. 创建一个新的用户：

CREATE USER 名字 IDENTIFIED BY 密码；

1. 创建SESSIONN权限才可以进行数据库的连接；

GRANT CREATE SESSION TO dog;  
然后还是无法创建表；

1. 为 用户分配创建表的权限

GRANT CREATE TABLE TO dog;

GRANT就是分配权限，但是一个一个的分配很累，

所以有CONNECT\RESOURCE。

1. 为用户分配角色； （任何系统里面如果一个用户的权限或者角色发生了变化，一定要重新登录）

GRANT CONNECT,RESOURCE TO dog;

即是CON 用户名/密码

1. 用户管理

修改密码

ALTER USER 用户名 IDENTIFIED BY 密码；

或者：

ALTER USER dog PASSWORD EXPIRE;

·锁定用户：(连都连不上)

ALTER USER dog ACCOUNT LOCK;

·解锁用户：

ALTER USER dog ACCOUNT UNLOCK;

1. 将对象操作的权限授予其他的用户；

对象的权限只有四种：INSERT、UPDATE、SELECT、DELETE。

GRANT SELECT,INSERT ON scott.emp TO dog;

EG;

GRANT SELECT,INSERT ON scott.emp TO dog;

授予

1. 回收权限：

REVOKE CONNECT,RESOURCE FROM dog;

1. 删除用户；

DROP USER dog CASCADE;(有数据和字段的时候)

OR用

DROP USER dog;

以上是一些管理员权限的操作。

# 数据备份；

数据库的热备份；

数据的导出：

·建立一个文件夹：e:\backup;

进入到命令行

·通过命令行方式进入到此目录下；

·输入exp命令。此命令指的是导出数据的操作；

现在只会导出有数据的表。不用动，一直回车

## 数据的导入：

imp，指令；

以上的方式只适合于小数据的情况，只适合于你项目更换数据库的情况。如果在数据库还在运行的情况下，你去数据备份，就会出现很大的问题。占用内存，很可能导致瘫痪。

# 数据库的冷备份：

所谓的冷备份局势归档备份，即：数据库服务停止，专门去备份。

我们还是专门的取备份，要备份一下的内容：

·控制文件：可以通过“v$controlfile”数据字典找到；

·重做日志文件：可以通过“v$logfile”数据字典找到；

·数据文件（表空间）：通过“v$datafile”数据字典找到；

·核心配置文件（pfile）：SHOW PARAMTER pfile找到；

1. 使用sys登录

CONN sys/change\_on\_install AS SYSDBA;

1. 找到控制文件信息

SELECT \* FROM v$controlfile;

1. 重做日志文件

SELECT \* FROM v$logfile

1. 数据文件

SELECT \* FROM v$datafile；

1. 找到pfile文件

SHOW PARAMTER pfile;

1. 记录好以上所有文件的路径
2. 关闭数据库服务

SHUTDOWN IMMEDIATE;

1. 重新启动数据库服务；

STARTUP

注意以上的操作使用只能用在稍微没那么严格的数据库

# 第一范式：

软件项目的开发一定会经历的阶段；

·获取需求阶段；

**·需求分析与业务设计**

**·数据库设计（这两个以后项目经理一定要把握在自己手里）**

·程序开发与业务实现；

·程序测试；

·程序的维护。

**业务很重要。**

所有的范式只能够作为一种参考设计。

初期才能根据范式进行设计。

**数据库的设计最大原则：**

根据需要竟可能的减少多表查询或复杂查询。

## 内容：

数据库中的每个字段都不可再分。（单表）

使用标准数据类型定义的表。

# 第二范式：

## 内容：

数据表中不存在非关键字段对任意一候选关键字段的部分函数依赖

## 解释：

说谁函数与函数依赖的关系

就是两个列之间不要存在有数学关系。

比如说：单价、总价之类的。

### 依赖：

指的是通过某几个字段推成功推演出唯一的一个字段。

比如：电影的年份，演员，片名，推演出导演的信息。

要有合适的东西作为主键。

多对多是大多数问题的解决办法。就是3张表

一个学生可以选多门课，一门课可以被多门课选，然后关系在是一张表。

# 第三范式：

数据表中不存在非关键字段对任意一候选关键字段的传递函数依赖。

EG:

知道一个人可以找到他的学校，找到班级，找到对应的班主任。

第三范式一定是优先考虑的

# 项目实践：

用户名密码的判断：

可以通过这样来实现：

SELECT COUNT(mid) FROM member WHERE mid=?AND password=?;

·这样输入正确的用户名和密码：数据量的统计为1；

·输入错误的用户名或密码：数据量的统计为0；