**一．ORACLE基础**

SELECT \* FROM(

SELECT ROWNUM rn , t.\* FROM (

SELECT ename,sal,JOB

FROM emp

ORDER BY sal DESC ) t

)

where rn between 6 and 10;

主流关系型数据库：关系型数据库就是数据表(一张二维的表格)

Oracle DB2【IBM】 SyBase SQL-Server【MS】 MySql

端口：1521 1433 3306

三大免费数据库：MySQL SQLite Access

结构化查询语言SQL：关系型数据库操作标准语言

**DDL数据定义语言：用于建立、修改、删除数据库对象【立即生效，无法回退】**

【CREATE ALTER DROP TRUNCATE】

DML数据操纵语言：改变数据表中的数据，和事务相关，执行完事务控制语句提交生效

【INSERT UPDATE DELETE】

TCL事务控制语言：维护数据一致性的语句

COMMIT【提交】 ROLLBACK【回滚】 SAVEPOINT【保存点】

**数据库事务具备以下特性：一致性、原子性、持久性、分离性**

DQL数据查询语言：SELECT

DCL数据控制语言：执行权限的授予和收回操作

GRANT【授权】 REVOKE【回收权限】 CREATE USER【创建用户】

DBMS:数据库管理系统-管理数据库的软件

DB数据库概念：是数据库和其他子对象（数据表/索引/视图）的集合。

ORACLE安装：

steup.exe以管理员身份运行—不接受更新—创建和配置数据库—桌面类

两个不锁定账户SYS-SYSTEM 密码为上一步设置的

验证安装成功：

DATABASE-Control网页管理器，https://localhost:1158/em/console/logon/logon

ORACLE卸载：

11g以后版本：安装目录dbhome\_1 - deinstall - deinstall.bat

ORACLE登录：用户名不区分大小写，密码区分大小写

1. 运行SQL PLUS，安装完第一次先用系统用户连接到数据库，密码为安装时设定的密码

四个系统自带账户：sys—system—sysman—scoot密码tiger(权限最低)

请输入用户名：sys/system as sysdba 输入口令：无需

请输入用户名：system 输入口令：clear【安装时设定】

2.再创建自定义用户，并授权以后可以登录等操作

创建用户：SQL: create user xxx identified by key;

用户授权：SQL: grant connect ,resource to xxx; 授权连接

grant create view to xxx; 授权视图

1. 已登录情况下用SQL语句切换用户：

① connect 用户名/密码； 【自定义用户或system】

② connect 用户名 as sysdba;【自定义用户或sys用户，切换并授权至最高权限管理员，无需密码】

综合： 登录-用户名：[username/paseword] [@server] [as sysdba|sysoper] 安装不在本机： [服务名或IP地址] [@server] [as sysdba|sysoper]

查看登录用户：show user

SQLdeveloper安装登录：

搭配jdk1.6.0\_18文件内容，路径：jdk-bin-java

新建连接： 修改服务名orcl

服务名查看方式：cmd-system32>lsnrctl status ,服务摘要第二个

oracle版本号查看： select \* from v$version;

SID/CORE查看：select instance\_name from v$instance;

数据字典： 是只读类型表的集合，存放数据库所用的有关信息。

包括数据库中所有框架对象的定义： 表 视图 簇 索引 分配及使用的空间 字段的缺省 用户的名字 用户的权限

管理员级用户查看的数据字典

dba\_tablespaces表空间数据字典 dba\_users用户信息数据字典

SQL:desc dba\_tablespaces

select tablespace\_name from dba\_tablespaces；

SYSTEM SYSTEM系统表空间，存放系统的表 视图...

SYSAUX EXAMPL的辅助表空间

UNDOTBS1 存储撤销信息

TEMP 临时表空间

USERS 用户表空间，存放用户的表 视图...

EXAMPLE 用与安装Oracle事例使用的表空间

普通用户查看的数据字典user\_tablespaces user\_users

通过数据字典查看其他用户信息：【普通用户无权限查看sys/system可以】

SQL:desc dba\_users

SQL:select username from dba\_users； 查看所有用户名

启用scoot用户：alter user scoot account unlock

1. **ORACLE数据类型**

UTF-8编码：一个中文占2-4个字节，其余都是一个字节

GBK：一个中文两个字节

字符型-n表示占用的字节数

数据库中的字符串用单引号，严格区分大小写，空格代表一个字符

N开头的，任何一个字符占两个字节，按照Unicode格式存放

非N开头的，Unicode中文字符占3个字节，其他占1个字节

CHAR(n) 定长n字节 默认最大2000字节

NCHAR(n) 存储任意1000个字符

VARCHAR2(n) 变长n字节 默认最大4000字节

NVARCHAR2(n) 任意2000个字符

Char 不指定长度默认为1，VARCHAR2必须指定长度

Length函数求占用字符数 lengthb/vsize求占用字节数

Lengthb（N‘xxx’）将字符串转换为Unicode字符串，任意每个字符占两个字节，返回6

Length（N‘xxx’）返回3

Name char(10 char) 20个字节

Name varchar2(10 char) ; Name可以保存10个字符 20个字节

数值型

NUMBER(p,s) p总位数，s小数点后的位数

FLOAT(n) 存储二进制数据，表示的二进制的位数是1-126位

转换10进制 \* 0.30103

存放大对象的数据类型：

BLOB 4G 二进制存储

CLOB 4G 字符串存储

把文件存放在数据库里有很多问题：

对数据库的读/写的速度永远都赶不上文件系统处理的速度

数据库备份变的巨大，越来越耗时间

对文件的访问需要穿越你的应用层和数据库层

日期类型：

日期，建议使用date类型，时间，建议使用它datetime  
因为如果用varchar 来存储年月日，那么需要10个字节，而date类型只需4个字节，而datetime类型也只需要8个字节，都小于varchar类型。  
其次，在进行查找、比较时，由于date和datetime本质上存储在数据库中是一个数字，所以直接通过数值比较效率很高，而varchar进行比较必需要一个字符一个字符比较，所以速度很慢。  
一条记录少了几个字节，总体节省的字节数就会很多，另外，加载到内存后，加载到内存更少，同时也只需要更少的IO，查询速度更快

DATE 长度7字节 默认格式DD-MON-RR 11-APR-90

TIMESTAMP 时间戳类型 精确到ns

SYSDATE 内部函数，返回当前系统时间，默认格式

SYSTIMESTAMP 内部函数 表示一个时间戳类型的当前系统时间的值,毫秒级

LAST\_DAY(date) 返回日期date 所在月的最后一天

ADD\_MONTHS(date,i) 返回日期date加上i个月后的日期值

MONTHS\_BETWEEN(date1,date2) 计算两个日期之间隔了几个月

NEXT\_DAY(date,char) 返回date日期数据的下一个周char

EXTRACT(date FROM datetime)

从参数datetime中提取参数date指定的数据，年、月、日

TO\_DATE():把字符串转换为日期格式

TO\_CHAR():把日期转换为字符串格式

--时间类型之间是可以比较大小的，时间越晚，值越大。

--时间之间可以做减法，差为相差的天数。

--日期可以加减数字，等同于加减指定的天数。

SELECT SYSDATE+4 FROM dual;

SELECT SYSTIMESTAMP FROM dual;

INSERT INTO tb\_name

(birth) VALUES(to\_date(‘1990-05-01’, ‘YYYY-MM-DD’))

SELECT to\_char(birth,'YYYY-MM-DD') FROM tb\_name;

SELECT to\_char(birth,'YYYY"年"MM"月"DD"日"') FROM tb\_name;

SELECT CEIL(SYSDATE-TO\_DATE('1987-12-10','yyyy-mm-dd') )FROM dual;

SELECT TO\_CHAR(SYSTIMESTAMP , 'YYYY-MM-DD')FROM dual;

SELECT EXTRACT(YEAR FROM SYSDATE) FROM dual;

SELECT EXTRACT(HOUR FROM TIMESTAMP ‘2008-10-10 10:10:10’) FROM dual;

SELECT LAST\_DAY(SYSDATE) FROM DUAL;

SELECT NEXT\_DAY(SYSDATE,4) FROM DUAL; 查询下一个周三的日期

SELECT hiredate,add\_months(hiredate,20\*12)AS "20周年后"

from emp;

VALUES( to\_date('1990-05-01','YYYY-MM-DD'));

SELECT ename，floor(months\_between(SYSDATE,hiredate))AS "一共入职多少月" from emp;

floor向上取整



1. **ORACLE表空间-TableSpace**

**表空间：是数据库的逻辑存储空间，【在数据库中开辟的一个空间，用来存放数据库的对象】 oracle的优化是通过表空间实现的，一个数据库可由多个表空间构成，这也是与** 其他数据库软件的区别

**是由一个或多个数据文件构成的，数据文件位置和大小由用户定义**

**分类：永久表空间：存储永久化存储的对象，表 视图 存储过程**

**临时表空间：数据库操作当中中间执行的过程，执行结束后自动释放掉**

**UNDO表空间 存储被修改之前的数据，便于数据回滚**

设置用户的默认或临时表空间

ALTER USER username DEFAULT|TEMPORARY TABLESPACE tablespace\_name

创建表空间

CREATE [TEMPORARY] TABLESPACE tbs\_name

TEMPFILE | DATAFILE 'XX.dbf' SIZE xx；

查看表空间存储路径

select file\_name from dba\_data\_files where

tablespace\_name='TEST1\_TABLESPACE';

修改表空间 修改状态：设置联机或脱机状态

ALTER TABLESPACE tablespace\_name ONLINE|OFFLINE;

查看状态

1.desc dba\_tablespaces

查看表空间，STATUS存放所有数据表的状态

2.select status from dba\_tablespaces where tablespace\_name='';

设置只读或可读写状态(必须是联机状态)

ALTER TABLESPACE tablespace\_name READ ONLY|READ WRITE;

修改数据文件：

ALTER TABLESPACE tablespace\_name

增加ADD DATAFILE 'xx.dbf' SIZE xx;

删除DROP DATAFILE 'filename.dbf';

eg:

ALTER TABLESPACE test1\_tablespace

ADD DATAFILE 'tset2\_file.dbf' SIZE 10m;

查看文件

select file\_name from dba\_data\_files where tablespace\_name='TEST1\_TAB;ESPACE';

删除表空间时，不能删除第一个创建的表空间，如果要删除就要把整个的表空间删掉

DROP TABLESPACE

tablespace\_name [INCLUDING CONTENTS]

如果要同数据文件一起删掉，需要加 [INCLUDING CONTENTS]

1. **ORACLE数据表-字段-记录**

数据表： 数据库最重要的组成部分之一，是其他对象的基础，基本的存储单位，存放于表 空间。

是一张二维的表格，有行、列columns，行称之为记录、列称之为字段

设计数据表以用来存储数据

SELECT 字段...：查找的字段可以是原字段经过函数或其他方法处理过后的，最后添加别名 显示出来最好

DUAL：伪表，可用于临时操作，包含系统时间等基本内容

表操作： CREATE SELECT ALTER DROP TRUNCATE DESC

字段操作：ADD DROP RENAME MODIFY

记录操作：INSERT DELETE SELECT UPDATE

eg:

CREATE TABLE tb1 ( 【父表】

Id NUMBER(5) PRIMARY KEY, 【参照列】

name VARCHAR2(20) NOT NULL ,

Gender CHAR(1) DEFAULT ‘M’,

Birth DATE DEFAULT sysdate, 【默认系统时间】

Salary NUMBER(6,2),

Job VARCHAR(30)

);

CREATE TABLE tb2 ( 【子表，外键约束】

id2 NUMBER(5) , 【物理外键列】

FOREIGN KEY (id2) REFERENCES tb1 (id)

);

表结构查询DESC t\_name; 【表的字段属性】

表查询： SELECT \* from t\_name 【字段-记录】

表创建：CREATE TABLE tb1( //同一个用户下表名唯一

c\_name data\_type 约束 ,

...);

表删除：DROP TABLE table\_name 数据和表一并删除

表更名：ALTER TABLE tbl\_name RENAME TO new\_name;

RENAME tbl\_name TO new\_tbl\_name;

复制表数据：CREATE TABLE tbl\_new AS SELECT (column1,..|\*) FROM tbl\_old;

部分字段|全部字段

字段添加：ALTER TABLE tbl\_name ADD (column\_name datatype 约束 , ...);

只能队尾

字段类型修改： ALTER TABLE tbl\_name MODIFY (col\_name datatype 约束，...）；

修改字段时，若表中有数据，类型尽量不改变，并且长度最好变长不要变短。

字段删除： ALTER TABLE tbl\_name DROP **COLUMN** col\_name；

字段更名： ALTER TABLE tbl\_name

RENAME **COLUMN** column\_name TO new\_column\_name ；

记录添加： INSERT INTO tbl\_name (column1,column2,...) VALUES (value1,value2...);

不写column默认给表的所有字段都添加，value应与表的所有字段对应

若有日期字段，需考虑日期的格式：

INSERT INTO tbl\_name (id,name,birth) VALUES (1,‘clear’,‘01-SEP-93’);

INSERT INTO tbl\_name (id,name,birth) VALUES (1,‘clear’,TO\_DATE(‘1993-03-03’,‘YYYY-MM-DD’));

记录复制：INSERT INTO tbl\_new (column1,...) SELECT ...；

更新记录：UPDATE tbl\_name SET (column1=value1,...)[WHERE condition];

不添加WHERE条件全表更新

删除记录：DELETE tb\_name [WHERE condition];

不添加WHERE删除表的全部记录，表结构仍在

TRUNCATE TABLE table\_name

删除全部记录，表结构仍在，不可回滚，速度快

**记录查询**：

SELECT \* FROM tb\_name; 查询表的所有字段

SELECT select\_expr [, select\_expr...]

[

FROM tb\_name

[WHERE where\_condition]

[GROUP BY {col\_name | position} [ASC |DESC],... ]

[HAVING where\_condition]

[ORDRE BY {col\_name | expr |position} [ASC |DESC],...]

]

执行顺序：

1. FROM:从后往前，从右到左 【数据量少的表放在后】
2. WHERE:自下而上，从右到左 【大量过滤的条件写在最右边】
3. GROUP BY从左往右分组
4. HAVING 尽量避免，耗资源
5. ORDRE BY 从左到右排序

逻辑查询处理阶段简介

FROM：对FROM子句中的前两个表执行笛卡尔积（Cartesian product)(交叉联接），生成虚拟表VT1

ON：对VT1应用ON筛选器。只有那些使<join\_condition>为真的行才被插入VT2。

OUTER(JOIN)：如 果指定了OUTER JOIN（相对于CROSS JOIN 或(INNER JOIN),保留表（preserved table：左外部联接把左表标记为保留表，右外部联接把右表标记为保留表，完全外部联接把两个表都标记为保留表）中未找到匹配的行将作为外部行添加到 VT2,生成VT3.如果FROM子句包含两个以上的表，则对上一个联接生成的结果表和下一个表重复执行步骤1到步骤3，直到处理完所有的表为止。

WHERE：对VT3应用WHERE筛选器。只有使<where\_condition>为true的行才被插入VT4.

GROUP BY：按GROUP BY子句中的列列表对VT4中的行分组，生成VT5.

CUBE|ROLLUP：把超组(Suppergroups)插入VT5,生成VT6.

HAVING：对VT6应用HAVING筛选器。只有使<having\_condition>为true的组才会被插入VT7.

SELECT：处理SELECT列表，产生VT8.

DISTINCT：将重复的行从VT8中移除，产生VT9.

ORDER BY：将VT9中的行按ORDER BY 子句中的列列表排序，生成游标（VC10).

TOP：从VC10的开始处选择指定数量或比例的行，生成表VT11,并返回调用者。

查询字段的顺序影响结果集出现的顺序，字段的别名影响结果集的别名

select\_expr：查询表达式

每一个表达式表示想要的一列，至少有一个，多个列之间用英文逗号隔开

\*代表所有列。 tbl\_name.\*可以表示表的所有列

from后返回可以是一张表或多个字段

别名：查询表达式可以使用[AS] alias\_name为列赋予别名

若希望别名区分大小写字符，或别名中包含空格，则必须使用双引号

别名可用于GROUP BY ，ORDRE BY 或 HAVING子句

WHERE:条件表达式

对记录进行过滤，如果没有指定WHERE子句，则显示所有记录

在WHERE表达式中，可以使用函数或运算符

SELECT \* FROM table\_name WHERE id > 3 AND age<30;

如果和字符及日期类型的数据比较，必须用单引号引起

比较运算符 = < > >=< ！= <>不等于

NOT AND OR 满足多条件 或 满足其中之一 优先级一次递减

OR的优先级低于AND的，若需要提高优先级，可以使用括号

SELECT \* FROM emp WHERE sal>1000 AND job='CLERK';

SELECT \* FROM emp WHERE sal>1250 AND (JOB='CLERK' OR JOB='SALESMAN');

SELECT \* FROM emp WHERE NOT(sal>1250 AND (JOB='CLERK' OR JOB='SALESMAN'));

GROUP BY:查询结果分组

--当select子句中出现了聚合函数，

--那么凡是不在聚合函数中的其他字段，

--必须出现在group by 子句中

--多字段分组：按照这几个字段的值组合一样的看作是一组。

--查看每个部门的平均工资

select AVG(sal),deptno from emp group by deptno;

--查看每个职位的工资的最大值，最小值，平均值和总和

select Max(sal),min(sal),avg(sal),sum(sal),job from emp group by job;

--查看平均工资高于2000的部门的最高工资和最低工资以及平均工资

select max(sal),min(sal),deptno from emp

group by deptno Having avg(sal)>2000;

HAVING:分组条件，对分组后的结果进行进一步限制，必须跟在GB后

要么是一个聚合函数，要么字段必须出现在开始的SELECT中，如上

--Having不能独立存在，必须跟在group by子句后面，用于进行

--分组统计后进行过滤使用。

--与where的区别：

--where是在第一次查表的时候进行过滤，只有满足where要求的记录

--才会被查出来，而having是在查询出来的数据基础上进行分组统计后

--得到的结果进行过滤。

--所以where先进行的过滤，having后进行的过滤。

ORDER BY:对查询结果进行排序，NULL值视为最大，默认ASC

必须出现在SELECT中最后一个子句

SELECT ename,deptno,sal FROM emp ORDER BY deptno,sal DESC;

优先使用deptno排序，当id字段存在数值相同时，数值相同的记录再使 用sal 排序

1. **ORACLE连接，子查询，分页查询**

--关联查询：查询的数据从多个表中得到，

--关联查询必须要写连接条件

--N张表进行连接，至少要有N-1个条件。

连接通常建立在存在相互关系的父子表之间，或称为驱动表和匹配表

--查询SALES部门的员工的姓名

SELECT ename from emp,dept

where emp.deptno=dept.deptno and dname='SALES';

--查看每个员工的信息以及其对应的部门名称及工作所在地

SELECT ename ,sal ,dname,loc from emp,dept

where emp.deptno=dept.deptno;

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

from emp e,dept d

where e.deptno=d.deptno;

--查看在“NEW YORK”工作的员工是谁？

SELECT ename,loc from emp e,dept d

where e.deptno=d.deptno

and d.loc='NEW YORK';

连接：在SELECT语句/多表更新/多表删除语句中支持JOIN操作

表的连接：其实就是外键的一种逆向操作，外键是把数据分开的存储，连接是把多张表连接在一起

等值连接：使用=连接相关的表

SELECT A.id,A.name,B.id,B.name FROM tb1 A,tb2 B WHERE A.age=B.age;

内连接：返回所有满足连接条件的记录

--内连接:也是一种关联查询

--内连接用join来连接，在on子句中书写连接条件,，多条件之间可以用and连接

--相对传统的关联查询，是将连接条件和过滤条件分开在不同的地方书写，

--语句表达更清晰

--查看每个员工的信息以及其对应的部门名称及工作所在地

SELECT ename ,sal ,dname,loc from emp,dept

where emp.deptno=dept.deptno;

SELECT ename ,sal ,dname,loc

from emp join dept

on emp.deptno=dept.deptno;

--查看经理所在部门

select \* from emp e join dept d

on e.deptno=d.deptno--连接条件

where e.job='MANAGER';--过滤条件

--查看"SALES"部门员工的信息

SELECT \* from emp e join dept d

on e.deptno=d.deptno

where d.dname='SALES';

外连接：

--当需要将不满足条件的记录也在关联查询中显示出来时，就要使用外链接

**--外链接分为左外链接，右外链接和全连接**

**--左外链接：以join左边的表为驱动表（显示所有信息），**

**--来自右边表中的字段当不满足条件时，全部为null。**

--外连接中如果存在对单表的条件后面使用WHERE子句，因为针对的是单表的条件

--显示所有部门员工信息，若有部门将部门名称显示出来。

--左外链接

select e.ename,e.deptno,d.dname

from emp e **LEFT OUTER** join dept d

ON e.deptno=d.deptno;

--右外链接

select e.ename,e.deptno,d.dname

from emp e **RIGHT OUTER** join dept d

ON e.deptno=d.deptno;

--全连接

select e.ename,e.deptno,d.dname

from emp e FULL OUTER join dept d

ON e.deptno=d.deptno;

**where子句中加(+) 是外连接的一种**

**带(+)侧的数据即使不存在，另一侧的数据依然可以显示出来。**

select e.ename,e.deptno,d.dname

from emp e ,dept d

where e.deptno(+)=d.deptno;

自连接：关联关系来自单表中的多个列

保存的数据相同，但是数据间又存在上下级关系，而且层级不确定

--查询员工姓名与其上级领导mgr的姓名

SELECT e.ename,m.ename

FROM emp e,emp m

WHERE e.mgr=m.empno;

--king的下属都有谁？

SELECT e.ename,m.ename

FROM emp e,emp m

WHERE e.mgr=m.empno

AND m.ename='KING';

--查看JONES下属的平均工资

SELECT AVG(e.sal)

FROM emp e,emp m

WHERE e.mgr=m.empno

AND m.ename='JONES';

表的参照关系：语法结构

tb\_name1

{ JOIN | {LEFT | RIGHT | FULL} JOIN }

tb\_name2

ON 连接条件

子查询(Subquery)：指出现在其他SQL语句内的SELECT子句.

eg: SELECT \* FROM t1 WHERE column1 = (SELECT column2 FROM t2);

子查询在WHERE子句中：先执行子查询，得到结果再进行主查询

子查询可以包含多个关键字或条件：必须出现在小括号内

DISTINCT GROUP BY ORDER BY 函数

子查询可以在HAVING子句中

子查询的外层可以是：SELECT DML DDL语句

子查询的返回值：标量 行 列 子查询

引发子查询的情况有三种：比较运算符， IN/NOT IN

--子查询根据查询的结果的字段数量与记录数分为：

--单行单列查询

--多行单列查询

--多行多列查询

--其中单列子查询多用于where中作为过滤条件

--而多列常当作一张表看待出现在from子句中

如果子查询返回多行，主查询要使用多行比较运算符，包括IN/ALL/ANY，ALL和ANY配合><=一起使用，使子查询返回一行

--在DDL中使用子查询，创建个表emp10,保存所有10号部门的员工

CREATE TABLE emp10 AS SELECT \* FROM emp WHERE deptno=10;

--在DML中使用子查询，删除BLAKE所在部门的所有员工

--1.先知道BLARK所在哪个部门

--2.根据部门去找其他员工

--3.再删除

DELETE FROM EMP

WHERE deptno=(SELECT deptno FROM emp WHERE ename='BLAKE' );

--将BLAKE所在部门的员工的工资提高10%

UPDATE emp SET sal=sal\*1.1

WHERE deptno=(SELECT deptno FROM emp WHERE ename='BLAKE' );

--查看和MANAGER相同部门的员工

SELECT \* FROM emp

WHERE deptno IN (SELECT deptno FROM emp WHERE JOB='MANAGER');

--查找薪水比整个公司平均工资高的员工信息

SELECT \* FROM emp

WHERE sal>(SELECT avg(sal) FROM emp);

--查找与SALESMAN同部门的其他职位的员工

SELECT \* FROM emp

WHERE deptno IN

(SELECT deptno FROM emp WHERE JOB='SALESMAN')

AND JOB <> 'SALESMAN';

--查询最低工资高于 （部门30的最低工资） 的部门编号

SELECT deptno FROM emp

GROUP BY deptno

HAVING MIN(sal)>(

SELECT MIN(sal)

FROM emp

WHERE deptno=30

);

--查看平均工资高于名叫CLERK的平均工资的职位

SELECT job FROM emp

GROUP BY JOB

HAVING avg(sal)>(

SELECT avg(sal)

FROM emp

WHERE JOB='CLERK');

--查看薪水比本（部门平均薪水）高的员工

SELECT e.ename,e.sal,e.deptno

FROM emp e,(SELECT avg(sal) avg\_sal , deptno

FROM emp

GROUP BY deptno

) t

WHERE e.deptno =t.deptno

AND e.sal>t.avg\_sal;

--select 子句，通常是实现外连接的效果，是外连接的另外一种表现形式

--查询员工ename，sal，deptno（从dept 表中获取的）

SELECT e.ename,e.sal,(SELECT t.deptno

FROM dept t

WHERE t.deptno=e.deptno

) deptno

FROM emp e;

EXISTS关键字

In中子查询和主查询可以不使用连接条件，子查询单独返回的字段的数据存在于主查询字段的数据即可，IN用于子查询时，子查询只能返回一个字段的数据。

Exists若要达到和in相同的效果，必须使用多表的连接并附加连接条件【见下方例句】

在子查询中需要引用到主查询的字段数据，使用exists关键字

后边的子查询至少返回一行数据，则整个条件返回true

Exists只能用于子查询，可以替代in，放在WHERE之后，Exists用于子查询时，子查询返回的可以是一张表，

【替代方法见例句】

若子查询结果集比较小，优先使用in

若外层查询比子查询小，优先使用exists。

因为若用in，则oracle会优先查询子查询，然后匹配外层查询，

若使用exists，则oracle会优先查询外层表，然后再与内层表匹配。

最优化匹配原则，拿最小记录匹配大记录。

--查看部门号存在于dept表中的员工姓名 1.2等效

1.SELECT ename FROM emp

WHERE deptno IN (SELECT deptno FROM dept);

2.SELECT ename FROM emp e

WHERE exists (select 1 from dept d where e.deptno=d.deptno);

3.SELECT ename FROM emp

WHERE exists (SELECT deptno FROM dept );

该句无效，子句为true，主查询返回所有数据

**--分页查询**

--当一条查询语句执行后的数据量过大，通常会采取分页措施

--一次只查询出一部分数据，分段获取，这样可以提高响应速度，以及减少系统资源的开销。

--分页的SQL语句没有标准，所以不同的数据库有不同的分页语句。

--ORACLE使用行号：rownum

--MYSQL使用：limit

--SQLSERVER使用：top

ROWNUM：被称作伪列，不是表中的一个真实的字段，可应用在任何一张表中，用于返回标识行数据顺序的数字,只能从1计数

--在结果集中体现的样子就是每一行记录出一个行号。

--行号是在查询数据的过程中生成的，只要SQL语句能查处一条数据，

--ROWNUM就会为该记录出一个行号，从1开始。

--由于ROWNUM只有查出一条数据，才会为其编号，然后自身进行增长，

--所以在第一次查询数据进行编号的过程中不要使用rownum进行大于1以上

--的数字作为过滤条件，否则查询不出任何数据。

--把查出来的数据按工资高低排序，拿到6-10的名单

SELECT \* FROM(

SELECT ROWNUM rn , t.\* FROM (

SELECT ename,sal,JOB

FROM emp

ORDER BY sal DESC ) t

)

where rn between 6 and 10;

--先排序，再编号，最后取范围

--oracle中的分页查询是靠两次子查询完成的。

--范围有一个算法：

--pageSize:每页显示的条目数

--page:页数,也叫第几页

--通过以上两个值计算行号的范围：

--min:(page-1)\*pageSize+1

--max:page\*pageSize

decode函数的使用：

查询职员表，根据职位计算奖励金额，没写到的职位奖金默认发放sal工资

SELECT ename,job,sal,

DECODE(job,

'MANAGER',sal\*1.2,

'ANALYST',sal\*1.1,

'SALESMAN',sal\*1.05,

sal) bonus FROM emp;

decode分组查询： COUNT(1) 1可以是任意数字，不能为null或不写

查询VIP和'OPERATION'的人数分布

SELECT COUNT(\*) job\_cnt , DECODE(job,

'ANALYST','VIP',

'MANAGER','VIP',

'OPERATION') job FROM emp

GROUP BY job;

查询VIP和'OPERATION'的人数总和

SELECT COUNT(\*) job\_cnt , DECODE(job,

'ANALYST','VIP',

'MANAGER','VIP',

'OPERATION') job FROM emp

GROUP BY DECODE(job,

'ANALYST','VIP',

'MANAGER','VIP',

'OPERATION');

按字段内容排序

SELECT deptno,dname FROM dept

ORDER BY

DECODE(dname,'OPERATIONS',1,

'ACCOUNTING',2,

'SALES',3);

case...when语句：查询语句中根据记录的不同显示不同的结果

SELECT ename,job,sal,

CASE job WHEN 'MANAGER' THEN sal\*1.2

WHEN 'ANALYST' THEN sal\*1.1

WHEN 'SALESMAN' THEN sal\*1.05

ELSE sal

END bonus FROM emp;

case之后也可以不写字段名，相当于查询语句

SELECT ename,job,sal,

CASE WHEN job= 'MANAGER' THEN sal\*1.2

WHEN job= 'ANALYST' THEN sal\*1.1

WHEN job='SALESMAN' THEN sal\*1.05

ELSE sal

END bonus FROM emp;

**排序函数**

ROW\_NUMBER

语法格式ROW\_NUMBER() OVER(

PARTITION BY col1 ORDER BY col2)

表示根据col1分组，在分组内部根据col2排序，组内连续且唯一

Rownum是伪列，ROW\_NUMBER功能更强，可以直接从结果集中取出子集

--按照部门编码分组显示，每组内按照职员编码排序，并赋予组内编码

SELECT deptno,ename,empno,

ROW\_NUMBER() OVER (PARTITION BY deptno ORDER BY empno)

AS emp\_id FROM emp;

RANK

语法格式RANK() OVER(

PARTITION BY col1 ORDER BY col2)

表示根据col1分组，在分组内部根据col2排序并给予等级标识

等级标识即排名，相同的数据返回相同的排名

跳跃排序，如果有相同的数据，则排名相同（并列），比如并列第二，则两行数据 都标记为2，但下一位将是第四名

和ROW\_NUMBER()的区别是有重复值，而ROW\_NUMBER() 没有

--按部门编码分组，同组内按薪水倒序排序，相同薪水按奖金正序排序，并给予组 内等级，用RANK\_ID表示

SELECT deptno,ename,sal,comm,

RANK() OVER

(PARTITION BY deptno ORDER BY sal DESC,comm)

AS "RANK\_ID" FROM emp;

DENSE\_RANK

语法格式DENSE\_RANK() OVER(

PARTITION BY col1 ORDER BY col2)

连续排序，如果有并列第二，下一个排序将是3，这一点和RANK不同

--关联emp和dept表，按照部门编码分组，每组内按照员工薪水排序，列出员 工的部门名字 姓名 和薪水

SELECT d.dname,e.ename,e.sal,

DENSE\_RANK() OVER

(PARTITION BY e.deptno ORDER BY e.sal) AS drank

FROM emp e join dept d ON e.deptno = d.deptno;

集合操作

为了合并多个SELECT语句的结果，可以使用集合操作符，实现集合的并、交、差

集合操作符包括UNION、UNION ALL、INTERSECT和MINUS

多条作集合操作的SELECT语句的列的个数和数据类型必须匹配

ORDER BY子句只能放在最后一个查询语句中

并UNION、UNION ALL

UNION自动去重复，并对查询结果进行排序

UNION ALL返回所有，不排序

SELECT ename，job,sal FROM emp WHERE job='MANAGER'

UNION/INTERSECT/MINUS

SELECT ename，job,sal FROM emp WHERE sal>2500;

交INTERSECT

INTERSECT返回的结果会以第一列的数据作为升序排列

差MINUS

第一个结果集减去第二个结果集

只有在第一个结果集存在，第二个不存在的数据才能被显现出来

高级分组函数

ROLLUP、CUBE、GROUPING SETS是GROUP BY 子句的扩展，

可以生成与使用UNION ALL来组合单个分组查询时相同的结果集，用来简化和高 效的实现统计查询

GROUP BY ROLLUP(a,b,c)假设有表test，有a,b,c,d四个列

SELECT a,b,c,SUM(d) FROM test GROUP BY ROLLUP(a,b,c);

等价于：（内部运行机制并不相同，效率要高很多）

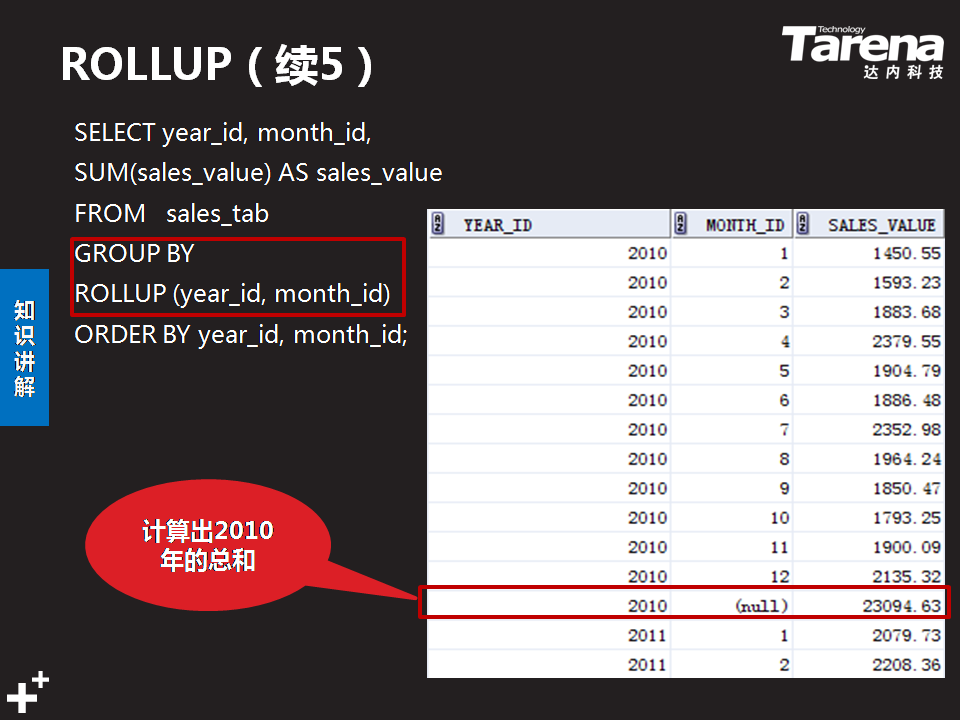
SELECT a,b,c,SUM(d) FROM test GROUP BY a,b,c UNION ALL

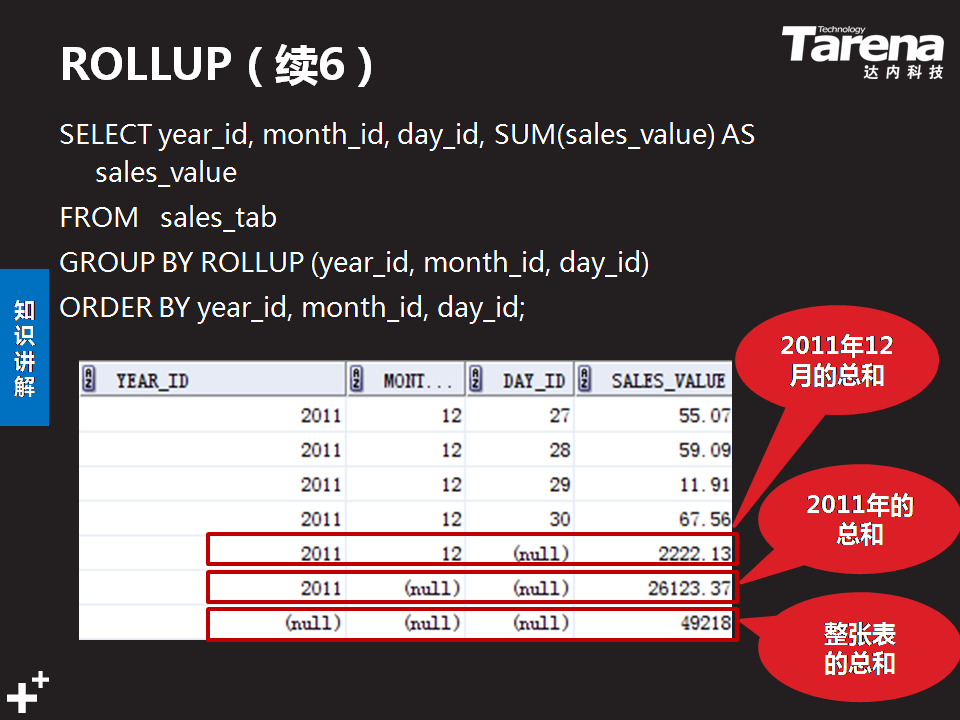
SELECT a,b,null,SUM(d) FROM test GROUP BY a,b UNION ALL

SELECT a,null,null,SUM(d) FROM test GROUP BY a UNION ALL

SELECT null,null,null,SUM(d) FROM test;

对ROLLUP的列从右到左以一次少一列的方式进行分组直到所有列都去掉后 的分组（也就是全表分组），对于n个参数的ROLLUP，有n+1次分组



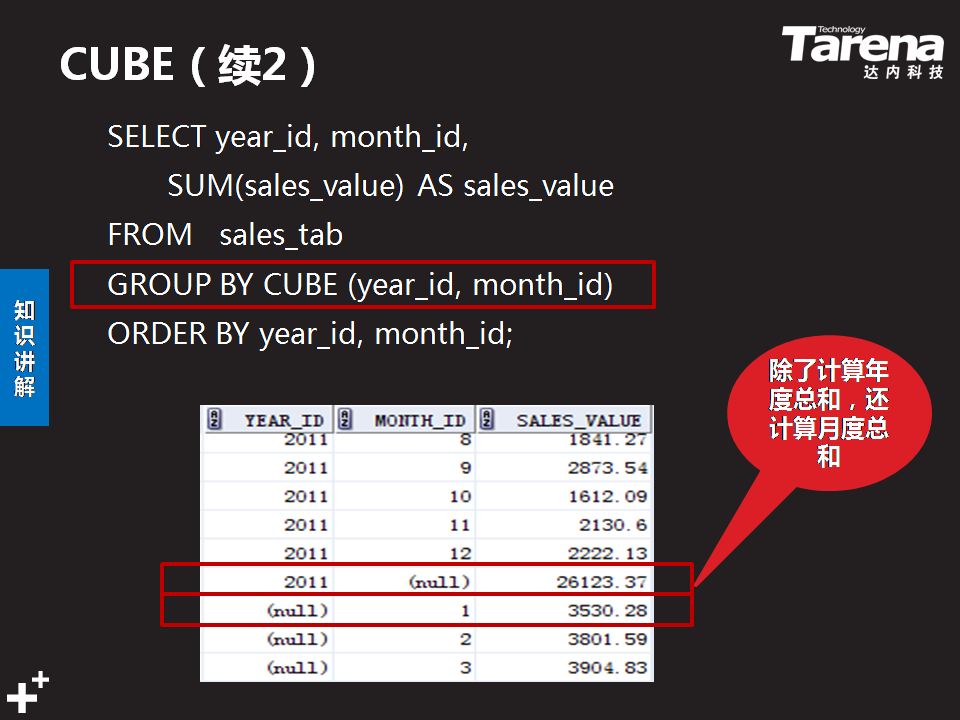


GROUP BY CUBE(a,b,c)

首先对（a,b,c）进行GROUP BY，然后依次是（a,b）,（a,c）,（a）,（b,c）,（b）,, （c），最后对全表进行GROUP BY 操作，一共是2^3=8次分组

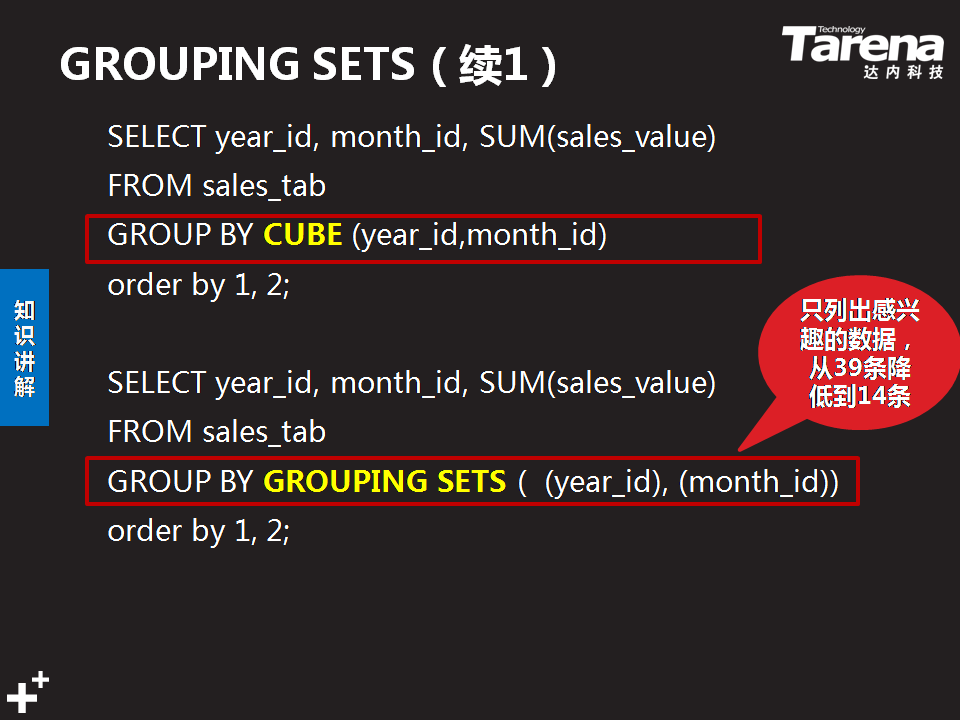
对CUBE的每个参数，都可以理解为（取值为参与分组）和（不参与分组）两个 值的一个维度，所有维度取值组合的集合就是分组后的集合。

对于n个参数的CUBE，有2^n次分组



GROUP BY GROUPING SETS

如果不需要获得由完备的ROLLUP或CUBE运算符生成的全部分组，则可以使用 GROUPING SETS仅指定所需的分组，GROUPING SETS列表可以包含重复分组





1. **ORACLE修饰字段的约束**

表级约束和列级约束：定义的方式划分

列级约束：创建表时直接在表后添加的约束，在列定义时后方声明，下1

表级约束：与列定义相互独立，不包含在列定义中，必须指出约束的别名，

可对多个数据列建立的约束，只能在列定义后声明。 下2

NOT NULL和 DEFAULT不存在表级约束

使用的方法：

1. 创建表时直接在字段名后添加 约束 关键字
2. 创建表时写完字段后，添加CONSTRAINT constraint\_name 约束 (column\_name,...)

此方法可以给约束设置别名

1. 都有启用禁用的方法ALTER TABLE emp DISABLE|ENABLE CONSTRAINT 约束名
2. 删除的通用方法：ALTER TABLE emp DROP CONSTRAINT 约束名 ;

作用：给字段定义规则，保证数据的完整性和一致性。

功能划分：NOT NULL(非空约束) 不添加默认允许为空

PRIMARY KEY(主键约束)

UNIQUE KEY(唯一约束)

DEFAULT(默认约束)

FOREIGN KEY(外键约束) CHECK 检查约束

**主键约束PRIMARY KEY，每张数据表只能存在一个主键，保证记录的唯一性，自动为NOT**  NULL。主键约束可以由多个字段构成（联合主键或复合主键）

--主键可以用序列作为值的提供方式，第一种

--主键还有一种提供值的方式是常见的UUID，第二种

【通用唯一识别码 (Universally Unique Identifier)】

--oracle的uuid是一个32位不重复的字符串，如果用uuid，主键的字段必须是字符 串类型。

--oracle提供了一个函数可以生成uuid，函数是：sys\_guid();

SELECT SYS\_GUID() FROM DUAL;

--主键有两种方式可以生成，通常数据量巨大的时候的会考虑使用UUID的生成方式。

修改已存在字段，添加主键约束：

ALTER TABLE emp MODIFY empno NUMBER(4) PRIMARY KEY；

ALTER TABLE emp MODIFY

( empno NUMBER(4), CONSTRAINT pkempno PRIMARY KEY(empno ) );

添加新字段，并设置主键约束：

ALTER TABLE emp ADD egirl varchar2(10) PRIMARY KEY;

ALTER TABLE emp ADD

( egirl VARCHAR2(10), CONSTRAINT ukegril PRIMARY KEY(egirl) );

添加(联合)主键约束：

CONSTRAINT constraint\_name PRIMARY KEY (column\_name,...)

删除主键约束：ALTER TABLE users2 DROP PRIMARY KEY [CASCADE];

启用/停止主键约束：ALTER TABLE users2 DISABLE|ENABLE PRIMARY KEY;

CASCADE级联约束，子表引用此主键约束的一并删除

**外键约束FOREIGN KEY：**

实现1对1或1对多的关系，先建驱动表，再给匹配表添加外键

关联不一定使用外键约束，很少使用物理外键约束，更多使用逻辑外键约束

条件：驱动表中的字段必须是主键字段

两个表参照关系的字段类型必须相同，NUMBER(2)，括号内值可以不同

匹配表中外键字段的值必须来自驱动表中的主键相应字段，或者写为null值亦可

当主表字段被从表参照时，主表的该行记录不允许被删除。

创建匹配表时直接定义外键约束，即写参照关系

CREATE TABLE tbn （coln datatype REFERENCES tb2(coln),...）;

1.ALTER TABLE dept MODIFY deptno NUMBER(2) PRIMARY KEY;

2.CREATE TABLE employees(

eid NUMBER(6) , NAME VARCHAR2(30), salary NUMBER(7,2),

deptno NUMBER(2) REFERENCES dept(deptno) 列级外键

);

//deptno NUMBER(2)， 表级外键

//CONSTRAINT fk FOREIGN KEY

// deptno REFERENCES dept(deptno ) [ON DELETE CASCADE];

ON DELETE CASCADE级联约束，主表删除从表一并删除，保证主从表完整性

修改表时添加外键约束

ALTER TABLE tbl\_name ADD FOREIGN KEY(pid) REFERENCES tbl\_name2 (id);

删除外键约束：ALTER TABLE users2 DROP CONSTRAINT constraint\_name;

启用/停止外键约束：ALTER TABLE users2 DISABLE|ENABLE constraint\_name ;

**唯一约束UNIQUE KEY，可以保证记录的唯一性，字段可以都为NULL，主键不可以为null，** 每张数据表可以存在多个唯一约束，主键只能有一个。

修改已存在字段，添加唯一约束：

ALTER TABLE emp MODIFY ename varchar2(10) UNIQUE;

ALTER TABLE emp MODIFY

( ename VARCHAR2(10), CONSTRAINT ukename UNIQUE(ename) );

添加新字段，并设置唯一约束：

ALTER TABLE emp ADD egirl varchar2(10) UNIQUE;

ALTER TABLE emp ADD

( egirl VARCHAR2(10), CONSTRAINT ukegril UNIQUE(egirl) );

**默认约束DEFAULT，插入记录时，如果没有明确为字段赋值，则自动赋予默认值**

数据库中，无论表中的字段是什么类型，在插入数据时只要不指定该字段的值，默认插 入的都为null,

若希望指定为特殊的默认值，可以使用default关键字。

需要注意的是，数据库中的字符串是用单引号括起来的，而且内容严格区别大小写。

**检查约束：CHECK**

检查约束条件用来强制在字段上的每个值都要满足check中定义的条件

CREATE TABLE tbn （salary number(5,0) check(salary>0)）;

ALTER TABLE employees4

ADD CONSTRAINT employees4\_salary\_check

CHECK(salary>2000);

**添加(非)空约束：ALTER TABLE emp MODIFY empno NUMBER(4) [NOT] NULL;**

默认情况，任何字段允许为NULL

添加删除默认约束：ALTER TABLE emp MODIFY aempno DEFAULT 15/[NULL];

约束改名：ALTER TABLE emp RENAME CONSTRAINT old\_name TO new\_name

查看约束信息：SELECT \* FROM user\_constraints; 用户约束所有信息

SELECT constraint\_name,constraint\_type,status from user\_constraints;

【直接使用无需改动】status状态：启用/禁用

删除约束：ALTER TABLE tb\_name DROP CONSTRAINT constraint\_name;

ALTER TABLE emp DROP CONSTRAINT ukename;

1. **ORACLE函数，作用于字段，返回新字段**

**字符串函数：**

CONCAT(char1,char2)字符连接，等价操作：连接操作符“||”

SELECT CONCAT(CONCAT(name, ‘:’),sal) FROM tb\_name;

SELECT name ||‘:’|| age FROM tb\_name;

LENGTH(char)返回字符串长度

VARCHAR2：返回字符的实际长度

CHAR：长度包括后补的空格

SELECT name,LENGTH(name) FROM tb\_name;

LOWER()转换成小写字母 UPPER()转换成大写字母

SELECT \* FROM tb\_name WHERE name= UPPER(‘clear’);

INITCAP()将字符串每个单词首字符大写，单词之间需有空格分隔

SELECT UPPER(‘hello world’),LOWER(‘HELLO WORLD’),INITCAP(‘hello world’) FROM DUAL;

LTRIM(c1[,c2]);c1的左边删除c2

RTRIM(c1[,c2]);c1的右边删除c2

TRIM(c2 FROM c1);c1的前后删除c2

如果没有c2，就去除空格

SELECT TRIM ('e'from 'elite')AS "t1",

LTRIM('elite','e')AS "t2",

RTRIM('elite','e')AS "t3" FROM DUAL;

LPAD(char1,n,char2)

RPAD(char1,n,char2)

补位函数，用于在字符串char1的左端或右端用char2补足到n位，可重复多次

SELECT username,LPAD(username,7,'CC')from tb2;

SUBSTR(char,m[,n])

获取子串，返回char中从m位开始【包括m位】取n个字符串

m=0，从首字符开始，等效于m=1;m取负数，从尾部开始，

字符串首位计数从1开始

如果没有设置n，或n的长度超过char的长度，则获取全部

SELECT SUBSTR (‘AAAABBBBCCCCDDDD’,8,25) FROM DUAL;

INSTR(char1,char2[,n,m])

返回子串char2在源字符串char1中的位置

n:从n的位置开始搜索，默认第一个字符开始

m:用于指定子串第m次出现次数，默认为1

Char1中没有找到char2，返回0

SELECT INSTR (‘AAAABBBBCCCCDDDD’, ‘CC’) FROM DUAL;

LIKE模糊查询

SELECT \* FROM emp WHERE ename LIKE '%A%'; ename字段包含A的记录

%代表通配符，任意多个字符，可以是0个；

\_下划线代表任意一个字符，而且必须存在

SELECT \* FROM emp WHERE ename LIKE '%9%%' ESCAPE '9'; 包含%的字符串

此处9可以是除%以外任意的字母或数字

正则表达式查询：

select \* from emp where regexp\_like(ename,'A'); ename字段包含A的记录

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

select \* from emp where regexp\_like(ename,'%'); 字段包含%的记录

**数值函数：**

ROUND(n[,m]):四舍五入n,保留m位小数

m必须是整数，默认值为0表示保留到整数位就是个位

正数表示保留小数点后多少位，负数表示保留到小数点前的位数

SELECT ROUND(45.678，-1) FROM DUAL;--50

TRUNC(n,m)不进行四舍五入，保留n位小数

SELECT TRUNC(45.678，-1) FROM DUAL; --40

MOD(m,n)用m/n取余数，若n为0，直接返回m

CEIL(n)返回大于给定值的最小整数（向上取整）

FLOOR(n)返回小于给定值的最大整数（向下取整）

SELECT CEIL(45.678) FROM DUAL;

SELECT FLOOR(45.678) FROM DUAL;

**比较函数：**

GREATEST(expr1,expr2,...)

LEAST(expr1,expr2,...)

返回结果是参数列表中最大或最小的值，参数类型必须一致，第一个以后的参数会被隐式转换为第一个参数类型，不能转换则报错

SELECT LEAST (SYSDATE, ‘10-10月-08’) FROM DUAL;

[NOT] BETWEEN...AND... [不]在范围之内 数据类型：数字/字符/日期

**[NOT] IN()** 取出[不]符合列表范围中的数据

IS [NOT] NULL [不]空

SELECT \* FROM emp WHERE JOB IN('MANAGER','CLERK');

SELECT \* FROM emp WHERE JOB =ANY('MANAGER', 'CLERK');

SELECT \* FROM emp WHERE JOB ='MANAGER' OR JOB='CLERK';

SELECT \* FROM emp WHERE JOB NOT IN('MANAGER','CLERK');

SELECT \* FROM emp WHERE JOB !=ALL('MANAGER', 'CLERK');

SELECT \* FROM emp WHERE JOB !='MANAGER' AND JOB!='CLERK';

SELECT \* FROM tb\_name WHERE id BETWEEN 0 AND 10;

SELECT \* FROM tb\_name WHERE job IS NULL；

**ANY ALL修饰的比较运算符**

作用：如果子查询在返回多个结果时，比较运算符无法选择其中一个结果进行比较，所以需要添加ANY ALL

>ANY() 大于括号内最小 <ANY() 小于括号内最大

>ALL() 大于括号内最大 <ALL() 小于括号内最小

=ANY运算符与IN等效

！=ALL或<>ALL与NOT IN等效

SELECT \* FROM WHERE sal>ANY(3000,3500,4000);

**NULL：**

空值，任何数据类型均可取值NULL

数据库中，无论表中的字段是什么类型，在插入数据时只要不指定该字段的值，默认插入的都为null,

--null与字符串拼接，相当于什么也没做

--null与数字运算结果为null

SELECT \* FROM student WHERE gender IS NOT NULL;

UPDATE tb\_name SET column1=NULL; 无非空约束下可以更新，更新后全部为NULL

**NVL**(expr1,expr2)

--NVL(a1,a2)空值函数，

--该函数的意义在于，将一个null值转换为非null值

--定义：当a1为null值时，函数返回a2的值，否则返回a1.

expr1和expr2可以是任何数据类型，但两个参数的数据类型必须一致

SELECT ename,sal,comm,nvl2(comm,'有奖金','没奖金') FROM emp;

**DISTINCT过滤重复**

SELECT DISTINCT JOB，deptno FROM emp; 查询职业，去除同行

--distinct 关键字，去除重复行

--distinct关键字必须紧跟在select 关键字之后

--不能保证结果集中某一个字段没有重复值，

--但能保证这几个字段的值的组合没有重复的

**聚合函数：只有一个返回值**

**又名多行函数，分组函数**

**AVG() COUNT() MAX() MIN() SUM()**

**自动忽略空值**

**MAX() MIN()**

用来取得列或表达式的最大最小值，可以统计任何数据类型

SELECT MAX(sal)"最高工资",MIN(sal)"最低工资"

select avg(sal),sum(sal) from emp;

select AVG(NVL(COMM,0))FROM EMP; 将null替换为0，来进行平均数统计

**AVG() SUM()**

平均值，和值，只能操作数字类型，忽略NULL值

SELECT AVG(age) , SUM(sal) FROM tb\_name;

SELECT ROUND(AVG(age),2) FROM tb\_name;

**COUNT()**

用来计算表中记录的条数，忽略null

SELECT COUNT(\*) FROM tb\_name;

select count(\*)"公司一共有几人" from emp;

SELECT COUNT(job) FROM tb\_name;

查看每个职位有多少人

SELECT COUNT(\*),job FROM emp GROUP BY job;

统计analyst和manager共计多少人，其他职位共计多少人

1. **视图、序列、索引**

**视图：VIEW也被称作虚表，是一组数据的逻辑表示**

**视图对应于一条SELECT语句subquery，结果集【对应的表被称作基表】被赋予** 【AS】一个名字，即视图名字

视图本身并不包含任何数据，它只包含映射到基表的一个查询语句，当基表数据发生 变化，视图数据也随之改变。

视图创建后，可以像操作表一样操作视图，主要是查询

根据视图所对应的子查询种类分为几种类型：

SELECT语句是基于单表建立的，且不包含任何函数运算、表达式或分组函数，叫 做简单视图，此时视图是基表的子集

SELECT语句同样是基于单表，但包含了单行函数、表达式、分组函数或GROUP BY 子句，叫做复杂视图，字段必须指定别名。

SELECT语句是基于多个表的，叫做连接视图

**作用**：如果需要经常执行某项复杂查询，可以基于这个复杂查询建立视图，此后查询 此视图即可，**简化复杂查询**

视图本质上就是一条SELECT语句，所以当访问视图时，只能访问到所对应的 SELECT语句中涉及到的列，对基表中的其他列起到安全和保密的作用，**限制 数据访问**

**授权：创建视图是CREATE VIEW ，用户必须有权限才可创建**

管理员授权：GRANT CREATE VIEW TO xxx

**视图操作**

CREATE OR REPLACE VIEW view\_name AS subquery;

CREATE VIEW v\_emp\_10 AS

SELECT empno,ename,sal,deptno

FROM emp WHERE deptno=10;

WITH READ ONLY;/WITH READ ONLY；

视图结构：DESC v\_emp\_10

--视图本身没有没有结构，而是将其子查询查询的基表

--对应的字段的类型长度列出来

查询视图：SELECT \* FROM v\_emp\_10;

--数据库会先将视图v\_emp\_10对应的select语句执行，

--然后得到结果集，再将该结果集当作一张'表'[其实是视图]，

--然后执行我们的查询工作。

修改视图：--由于视图本身没有结构，只是对应了一条子查询

--所以修改视图就是替换原有的子查询，和创建的语句相同

--使用语句create or replace 创建修改都可以，没有则创建

--有则替换，视图对应的子查询可以为字段添加别名，

--若视图AS查询的字段含有表达式，函数【即复杂视图】，则该字段必 须指定别名。

CREATE OR REPLACE VIEW v\_emp\_10 AS

SELECT empno ID,ename NAME, sal salary,deptno

FROM emp WHERE deptno=10; 【添加别名】

设置别名后只能通过别名查询

SELECT ID,NAME,salary,deptno FROM v\_emp\_10;

删除视图：DROP VIEW view\_name; 不会影响基表数据

**简单视图DML操作**：INSERT DELECT UPDATE

视图本身并不包含数据，只是基表数据的逻辑映射

当对视图执行DML操作时，实际上是对数据来源的基表的DML操作

只有简单视图能够执行DML操作,复杂视图不允许DML操作

对视图执行DML操作的基本原则：

简单视图执行DML操作影响到基表的数据，下列情况除外：

1. 在基表中定义了非空列，但简单视图对应的的SELECT语句并没有包含 这个非空列，此时无法对视图执行INSERT操作
2. DELECT操作时，只能删除现有视图里能查到的记录
3. 如果视图定义中包含了函数、表达式、分组语句、DISTINCT关键字或 ROWNUM伪列，不允许执行DML操作
4. DML操作不能违反基表的约束条件

INSERT --向视图中插入一条数据，就是将数据插入到基表中，

--并且，该条数据只有视图看得到的字段会有值

--其他都是插入默认null（单独指定默认值的除外），此时对基表产生了污染。

--所以，若是基表中视图看不到的字段含有not null约束的话，那么会插入失败。

INSERT INTO v\_emp\_10 VALUES (1001,'lihang',5000,10);

INSERT INTO v\_emp\_10 VALUES (1001,'lihang',5000,20);

此处第二条记录执行成功，基表存在记录，但视图查不到，因为视图查询 条件有deptno=10

UPDATE --更新同样存在对基表的污染

UPDATE v\_emp\_10

SET salary=salary+500 WHERE deptno=10;

UPDATE v\_emp\_10

SET salary=salary+500 WHERE deptno=20;

--第二条失败，没有效果，视图看不到20部门，所以无法更新

UPDATE v\_emp\_10 SET deptno=20;

--污染基表和视图

因为执行完会污染基表数据使基表的deptno=10变为20，此后再查看 视图会不存在数据，因为视图条件是deptno=20

DELETE 删除数据只能删除视图看得到的记录，同时删除基表的记录

DELETE v\_emp\_10 WHERE deptno=20;

SELECT \* FROM v\_emp\_10;

SELECT \* FROM emp;

**CHECK OPTION视图的约束**

CREATE OR REPLACE VIEW view\_name AS subquery

WITH CHECK OPTION;

对视图所做的修改【增、改】操作前、后，数据都必须在视图的可见范围之内

CREATE OR REPLACE VIEW v\_emp\_10 AS

SELECT empno ID,ename NAME,sal salary,deptno

FROM emp WHERE deptno=10

WITH CHECK OPTION;

INSERT INTO v\_emp\_10 VALUES (1003,'xiaomei',6000,20);

执行前，检查约束20数据对视图不可见，操作失败。对视图和基表无影响

UPDATE v\_emp\_10 SET deptno=20;

执行后，视图将不可见，所以禁止执行

**WITH READ ONLY 视图的约束**

将视图设置为只读的，该视图不允许执行DML操作，只允许DQL查询语句

**复杂视图（多表关联）**

必须为子查询的表达式或函数定义别名，复杂视图不允许DML操作

--创建一个视图V\_EMP\_SALARY，按部门名字和部门号分组

--获得每个部门的平均薪水，薪水总和，最高薪水和最低薪水。

CREATE OR REPLACE VIEW v\_emp\_salary AS

SELECT d.dname,d.deptno,

avg(e.sal) avg\_sal，sum(e.sal) sum\_sal,

MAX(e.sal) max\_sal , MIN(e.sal) min\_sal

FROM emp e JOIN dept d

ON d.deptno=e.deptno

GROUP BY d.dname,d.deptno;

--查看那些员工比所在部门平均薪水高？

SELECT e.ename,e.sal,e.deptno

FROM emp e,v\_emp\_salary s

WHERE e.deptno=s.deptno

AND e.sal>s.avg\_sal;

--查看那些员工的工资的一半都比30号部门的平均工资高

SELECT ename,sal,deptno

FROM emp

WHERE sal/2>(SELECT avg\_sal

FROM v\_emp\_salary

WHERE deptno=30);

**序列：SEQUENCE是一种用来生成唯一数字值的数据库对象**

序列的值由ORACLE程序按递增或递减顺序自动生成，通常用来自动产生表的主 键值，是一种高效获得唯一键值的途径

**序列是独立的数据库对象，和表是独立的存在，并不依附于表**

创建在schema方案下

通常一个序列为一个表提供主键，但也可以为多个表提供主键

**CREATE SEQUENCE sequence\_name**

[START WITH i] [INCREMENT BY j] 起始，步进

[MAXVALUE m | NOMAXVALUE]

[MINVALUE n | NOMINVALUE]

[CYCLE | NOCYCLE] [CACHE p | NOCACHE]

序列的第一个序列值是i，步进是j，整数递增，负数递减；

可生成的最大值m，最小值n；

CYCLE表示递增至最大或最小后是否重复；

CACHE用来指定先预取p个数据在缓存中，以提高序列值得生存效率，默认20

默认起始为1，步进为1；NOCYCLE

--序列提供两个伪列：

--nextval:获取序列的下一个值

--currval:获取序列的当前值（最后一次生成的值）

--nextval在新创建的序列中使用会首先获取start with指定的值，

--以后则是用最后一次生成的数字加上步长得到，序列不可以后退，

--所以当调用了nextval后就无法再得到之前生成的数字了。

--currval可以获取最后生成的数字，无论调用多少次值都是一样，

--除非调用过nextval生成的下一个数字。

--需要注意：刚创建的序列必须先执行一次nextval后才可以使用currval.

SELECT emp\_seq.nextval FROM dual;

SELECT emp\_seq.currval FROM dual;

序列操作：创建序列

**CREATE SEQUENCE emp\_seq**

START WITH 100 INCREMENT BY 10 ;

INSERT INTO emp (empno,ename,JOB,sal,deptno)

VALUES (emp\_seq.nextval , 'xiao宁' , 'CLERK',6000,10) ;

--删除序列

DROP SEQUENCE emp\_seq;

**索引：** 允许直接访问数据表中某一行数据的树形结构。

**用于提高查询效率，排序效率等操作的机制。**

**是独立于表的对象，可以存放在与表不同的表空间中**

**唯一约束和主键约束会自动创建索引**

--索引可以建立在一张表上的单列上，也可以建立在多列上。

--索引记录中存有索引关键字和指向表中数据的指针（地址）

--对索引进行的IO操作比对表进行操作要少很多

--为经常出现在WHERE子句中的列、ORDER BY 、DISTINCT后面的字段创建 索引，如果是复合索引，索引的字段顺序要和关键字后的顺序一致

--为经常作为表的连接条件的列上建立索引

--不在经常做DML操作的表上创建索引

--不在小表上创建索引

--限制索引数目，并不是越多越好

--删除很少被使用的、不合理的索引

--索引的算法是数据库自动维护的，我们无需关注，

--当我们进行查询或排序时，数据库会自动使用可用的索引。

**--简单索引**

**CREATE [UNIQUE] INDEX idx\_emp\_ename ON emp(ename);**

--复合索引/多列索引

若经常在ORDER BY子句中使用job和sal作为排序依据，建立复合索引

**CREATE INDEX idx\_emp\_job\_sal ON emp(JOB,sal);**

SELECT empno,ename,sal,job FROM emp ORDER BY job,sal;

--基于函数的索引

在emp表的ename列上执行(大小写)的搜索

CREATE INDEX idx\_emp\_ename\_upper ON emp(UPPER(ename));

SELECT \* FROM emp WHERE UPPER(ename) = 'KING';

--重建索引（修改索引）

如果经常在索引列上执行DML操作，需要定期重建索引，提高索引的空间利用率

ALTER INDEX idx\_emp\_ename REBUILD;

--删除索引

当表上有不合理的索引，会导致操作性能下降

drop index idx\_emp\_ename;