**达内郑州嵩山路中心java方向各阶段重点知识点(二)** 2017.8.9

**该文档由达内郑州嵩山路中心java项目经理何老师进行整理,供学员学习使用,中心之外使用请保留该出处,您身边如有想要学习技术的请联系嵩山路java项目经理:何老师,电话:13838585543**

**注:文档中带有 "面" 标识的为重点面试题,,带有"笔试"标识的重点笔试题,未标注的为各阶段重点知识点**

1. **数据库SQL语言的分类？(面)**

数据定义语言（DDL）：用于建立，修改，删除数据库对象，包含：表，视图，索引等。

比如：创建语句：CREATE 修改语句：ALTER 删除语句：DROP 删除数据：TRUNCATE

数据操纵语言（DML）：用户改变数据库数据

比如：数据插入：INSERT 更新数据：UPDATE 删除数据：DELETE

事务控制语言（TCL）：用来维护数据的一致性的语句

比如：提交:COMMIT 回滚：ROLLBACK 保存点：SAVEPOINT

数据查询语言（DQL）：查询数据

比如：SELECT

数据库控制语言（DCL）：用来执行权限的授予和收回操作

1. **数据库对象有哪些？(面)**

1)表：（TABLE)存储结构性数据。

2)视图：(VIEW)也被称作虚表，即虚拟的表，它本身并不包含任何数据，只包含映射到基表的一个查询语句

3)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，是一种高效率获得唯一键值的途径。

4)索引：（INDEX）索引是一种允许直接访问数据表中某一数据行的树型结构，为了提高查询效率而引入，是独立于表的对象，可以存放在与表不同的表空间（TABLESPACE）中。

注意:oracle有序列,mysql没有序列,有相同功能类型的自动编号AUTO\_INCREMENT.

用法:创建表时必须和主键约束结合使用,不能单独使用,默认从1开始编号,自动增长

如:

CREATE TABLE   test(

id INT PRIMARY KEY AUTO\_INCREMENT,  
 username VARCHAR(15) NOT NULL  
) AUTO\_INCREMENT = 100;

可在建表时可用“**AUTO\_INCREMENT=n**”选项来指定一个自增的初始值。

另外还可以在建表之后对自动编号设置自增的初始值:如下

alter table tbname auto\_increment = x ;  
设置表tbname的唯一auto\_increment字段起始值从x开始

1. **Oracle中的数据类型?**

**数字类型:**

**NUMBER**

NUMBER表示数字类型，经常被定义成NUMBER（P，S）形式，其中：

P表示数字的总位数 S表示小数点后面的位数,比如:



表示sal 列中是一个小数,数字总长度为6,其中4位整数,2位小数,最大支持9999.99

**INTEGER**

INTEGER是NUMBER的子类型, 它等同于NUMBER（38,0），用来存储整数。若插入、更新的数值有小数，则会被四舍五入,在书写时也可以写成INT.

**FLOAT**

FLOAT类型也是NUMBER的子类型, Float(n),数 n 指示位的精度，可以存储的值的数目。N 值的范围可以从 1 到 126。

注:还有一些其他数字类型,不常用不一一列举

**字符串类型:**

**CHAR**: 存放定长字符，如果数据存不满定长长度，则补齐空格, 最大取值为2000字节，也就是定义为CHAR（2000）。其中最多保存2000个英文字符，1000个汉字（GBK）, 如果不指定长度，默认为1个字节.

**VARCHAR2**: 存放变长字符，实际数据有多少长度则占用多少, 最大取值为4000字节，也就是VARCHAR2（4000），最多保存4000个英文字符，2000个汉字（GBK）,必须指定字节.

**LONG**:可以认为是VARCHAR2的加长版，用来存储变长字符串，最多达2GB的字符串数据，但是LONG类型有诸多限制,比如:每个表只能有一个LONG类型列, 不能作为主键,不能建立索引等,所以不建议使用.

**CLOB:**用来存储定长或变长字符串，最多达4GB的字符串数据，ORACLE建议开发中使用CLOB替代LONG类型

**时间类型:**

**DATE:** 用来保存日期和时间。表示范围从是公元前4712年1月1日至公元9999年12月31日, 在数据库中的实际存储固定为7个字节，格式分别为:

第1字节：世纪+100

第2字节：年

第3字节：月

第4字节：天

第5字节：小时+1

第6字节：分+1

第7字节：秒+1

**TIMESTAM**:表示时间戳，与DATE的区别是不仅可以保存日期和时间，还能保存小数秒，可指定为0-9位，默认6位，最高精度可以到ns(纳秒)级别。格式为:

第1字节-第7字节：和DATE相同

第8-11字节：纳秒，采用4个字节存储，内部运算类型为整型

1. **ORACLE数值类型的相关函数?**

数值函数指参数是数值类型的函数, 常用的有ROUND、TRUNC、MOD、CEIL和FLOOR。

**ROUND**: ROUND(n,m)用于将参数n按照m的数字要求四舍五入,

m必须是整数

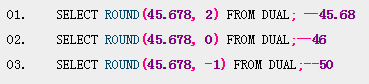
m取正数则四舍五入到小数点后第m位

m取0值则四舍五入到整数位

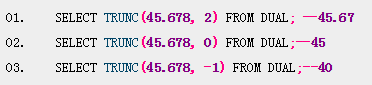
m取负数，则四舍五入到小数点前m位

m缺省，默认值是0

比如:



**TRUNC** :TRUNC(n,m)的功能是截取，其中n和m的定义和ROUND(n[, m])相同，不同的是功能上按照截取的方式处理数字n.比如:



**MOD**:MOD(m, n)是取模函数，返回m除以n后的余数，如果n为0则直接返回m。

比如: 假设sal是1500,MOD(sal,1000)取余后结果为500



**CEIL**:向上取整,

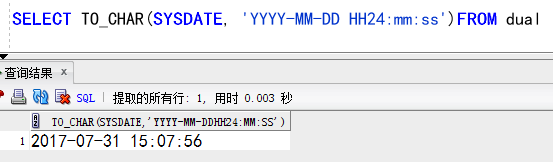
**FLOOR:**向下取整

比如: 数字n = 4.5，那么它的CEIL是5.0，它的FLOOR是4.0



1. **ORACLE时间类型的相关函数?**

**SYSDATE:** Oracle的内部函数，用来返回当前的系统时间，精确到秒，默认显示格式是DD-MON-RR，只有年月日并不显示时间,如果想要显示年月日时分秒的格式可以使用TO\_CHAR函数进行格式转换,例如:



**日期转换函数:**

**TO\_CHAR:** 将其它类型（日期，数值）的数据转换为字符类型,一般作用在日期格式上.

**TO\_DATE:**是将字符串按照定制格式转换为日期类型

**日期常用函数:**

**LAST\_DAY:** LAST\_DAY(date)返回日期date所在月的最后一天

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

注: 如果i是小数，将会被截取整数后再参与运算

如果i是负数，则获得的是减去i个月后的日期值

**MONTHS\_BETWEEN**: MONTHS\_BETWEEN(date1, date2)：计算date1和date2两个日期值之间间隔了多少个月，实际运算是date1-date2，如果date2时间比date1晚，会得到负值.

**NEXT\_DAY**:NEXT\_DAY(date, char)：返回date日期数据的下一个周几,周几是由参数char来决定的, 可以直接用数字1-7表示周日-周六.

**GREATEST**: GREATEST(参数1,参数2,参数3......):可以传入多个参数,返回其中时间最大的.

**LEAST:** LEAST(参数1,参数2,参数3......): 可以传入多个参数,返回其中时间最小的.

**EXTRACT**:EXTRACT(date FROM datetime),从datetime中提取date指定的数据,比如年月日.

1. **ORACLE字符串类型的相关函数?**

**CONCAT:**是字符串连接函数, CONCAT(char1, char2), concat只能有两个参数，所以如果连接三个字符串时，需要两个concat函数,比较麻烦,可以使用 **||,比如:**

****

**LENGTH:** 返回参数字符串的长度。如果字符类型是VARCHAR2，返回字符的实际长度，如果字符类型是CHAR，长度还包括后补的空格。

**UPPER**:用于将字符转换为大写形式.

**LOWER**:用于将字符转换为小写形式.

**INITCAP**:用于将字符串中每个单词的首字符大写.

**TRIM(c2 FROM c1)**: 表示从c1的前后截去c2.

**LTRIM(c1[, c2]):** 表示从c1的左边（Left）截去c2.

**RTRIM(c1[, c2]):** 表示从c1的右边（Right）截去c2.

**LPAD(char1, n, char2):**左补位函数,在char1的左端使用char2补足到n位.

**RPAD(char1, n, char2):**右补位函数在char1的右端使用char2补足到n位.

**SUBST(char,m,n):**表示在一个字符串中截取子串, 用于返回char中从m位开始取n个字符的子串,如果m = 0，则从首字符开始，如果m取负数，则从尾部开始.未设置n则表示截取到末尾.

**INSTR(char1, char2[, n [, m]]):** 返回子串char2在源字符串char1中的位置, 从n的位置开始搜索，没有指定n，从第1个字符开始搜索, m用于指定子串的第m次出现次数，如果不指定取值1,在char1中未找到char2则返回0.

1. **空值函数有哪些?(面)**

**NVL(expr1, expr2)：**如果expr1为NULL，则取值expr2, expr2是非空值。其中expr1和expr2可以是任何数据类型，但两个参数的数据类型必须是一致的.

例如: 计算员工月收入，如果comm列为空值的话,将空值转换为0进行计算.



**NVL2(expr1, expr2, expr3):**如果expr1不为NULL则取expr2的值,如果expr1为NULL,则取expr3的值.

例如:查询员工有没有奖金,如果comm不为NULL则显示有奖金,如果comm为NULL则显示没有奖金.

SELECT ename,comm,NAL2(comm,'有奖金','没有奖金')FROM emp;

1. **数据库中的别名?**

SQL语句中可以使用通过列的别名改变标题的显示样式,或者表示计算结果的含义,使用语句是列的别名跟在列名后,中间可加可不加"AS"关键字,别名可以直接写,不必使用双引号括起来,但如果希望别名区分大小写或者包含字符或者空格,则必须使用双引号括起来.

例如:

SELECT ename AS "EName",sal\*12 sal FROM emp\_hexf;

1. **数据库中的条件查询?**

**>, <, >=, <=, !=, <>, =**

以上条件查询都各自代表各自含义,不再一一解释,重点说一下!=和<>,这两个都代表不等于的意思.

在查询条件中还会用到并且和或者的关系.可以使用

**AND:并且**

**OR:或者**

**IN:**在某个传入参数列表中

**NOT IN:**不在传入参数的列表之中

例如查询职位是MANAGER或者CLERK的员工：

SELECT ename, job FROM emp WHERE job IN ('MANAGER', 'CLERK');

查询不是部门10或20的员工：

SELECT ename, job FROM emp WHERE deptno NOT IN (10, 20);

**BETWEEN…AND…:** 用来查询符合某个值域范围条件的数据，最常见的是使用在数字类型的数据范围上，但对字符类型和日期类型数据也同样适用.

比如: 查询薪水在1500-3000之间的职员信息：

SELECT ename, sal FROM emp WHERE sal BETWEEN 1500 AND 3000;

**IS NULL:**是空

**IS NOT NULL:**不是空

空值NULL是一个特殊的值，比较的时候不能使用”=”号，必须使用IS NULL，否则不能得到正确的结果

例如: 查询哪些职员的奖金数据为NULL

SELECT ename, sal, comm FROM emp WHERE comm IS NULL;

**ANY**:任一,不能单独使用,只能和比较操作符>、>=、<、<=一起使用

**ALL**: 全部,不能单独使用,只能和比较操作符>、>=、<、<=一起使用

比如:

> ANY : 大于最小 < ANY：小于最大

> ALL：大于最大 < ALL：小于最小

**DISTINCT:** 过滤去重,可单独一个字段使用,也可以多个字段联合使用.

比如: 查询员工的部门编码，去掉重复值

SELECT DISTINCT deptno FROM emp;

deptno和job联合起来不重复:

SELECT DISTINCT deptno, job FROM emp;

1. **结果集数据排序?(面)**

对查询结果集排序可使用ORDER BY子句,ORDER BY后跟着要排序的字段,然后跟着升序或者降序的关键字, ASC为升序,DESC为降序,ASC可以不写,默认升序.

例如: 按员工的薪水倒序排序

SELECT ename, sal FROM emp ORDER BY sal DESC;

还可以多个字段排序

例如:对职员表中的职员排序，先按照部门编码正序排列，再按照薪水降序排列

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

1. **数据库中聚合的数据函数是做什么用的?有哪几个?(面)**

聚合函数是对将结果集按照指定字段进行统计然后得到一条记录,聚合函数有5个.

**MAX**:统计最大值

**MIN:**统计最小值

**AVG**:统计平均值

**SUM**:统计总和

**COUNT**: 统计指定字段值不为NULL的记录总和

注:COUNT(\*):统计包含NULL的记录总和

1. **数据库中对结果集进行分组?(面)**

对结果集按照指定的字段值相同的记录分组,可以使用GROUP BY,后面跟要分组的字段.

注意: 当SELECT中含有聚合函数时,那么不是聚合函数的其他单独字段都需要在GROUP BY子句中,因为聚合函数查询出的结果是单列显示,非聚合函数的其他字段查询出来是多列显示,结果不能匹配显示,所以需要将非聚合函数字段放在分组中让其成为单行显示,这样最终的结果才能正常显示. 但是反过来则不是必须的,GROUP BY中的字段不一定要都出现在SELECT语句中.

比如: 查看每个部门的平均工资?

SELECT AVG(sal),deptno FROM emp

GROUP BY deptno

另外如果该查询结果有条件,那么可以在GROUP BY 后跟上HAVING过滤条件对结果集进行过滤.

比如:查看除10号部门之外的平均工资?

SELECT AVG(sal),deptno FROM emp

GROUP BY deptno

HAVING deptno<>10

注: HAVING子句不能独立存在,必须跟在GROUP BY子句后

另外: HAVING可使用聚合函数作为过滤条件,它是用来根据统计结果决定保留那些分组的

比如: 查看部门人数大于3人的部门的平均工资?

SELECT deptno,AVG(sal)FROM emp

GROUP BY deptno

HAVING COUNT(\*)>3

1. **一条查询语句中包含所有的子句，那么各子句之间的执行顺序?(面)**

**FROM-->WHERE-->GROUP BY-->HAVING-->SELECT-->ORDER BY**

FROM 子句：执行顺序为从后往前、从右到左。数据量较少的表尽量放在后面。

WHERE子句：执行顺序为自下而上、从右到左。将能过滤掉最大数量记录的条件写在

WHERE 子句的最右。

GROUP BY：执行顺序从左往右分组，最好在GROUP BY前使用WHERE将不需要的记录

在GROUP BY之前过滤掉。

HAVING 子句：消耗资源。尽量避免使用，HAVING 会在检索出所有记录之后才对结果

集进行过滤，需要排序等操作。

SELECT子句：少用\*号，尽量取字段名称。ORACLE 在解析的过程中, 通过查询数据字典

将\*号依次转换成所有的列名, 消耗时间。

ORDER BY子句：执行顺序为从左到右排序，消耗资源。

1. **WHERE过滤子句和HAVING过滤子句的区别?(面)**

WHERE过滤子句是查询过程中对表中数据的过滤条件, 不允许使用聚合函数作为过滤条件,原因在于时机不对,聚合函数是对表中数据查询后的结果集进行统计处理的,两者的执行时机不一致.

HAVING过滤子句是对查询结果集进行过滤的,可以在该子句后使用聚合函数,因为时机相同,聚合函数是处理结果集的,HAVING子句又是过滤结果集的,可以在一起使用,另外HAVING不能单独使用,只能跟在GROUP BY分组子句后面使用.

如下为错误案例:

查看部门人数高于3人的这些部门的平均工资?

SELECT deptno,AVG(sal) FROM emp

WHERE COUNT(\*)>3

GROUP BY deptno

正确案例为:

查看部门人数高于3人的这些部门的平均工资?

SELECT deptno,AVG(sal)FROM emp

GROUP BY deptno

HAVING COUNT(\*)>3

1. **TRUNCATE和DELETE以及DROP的区别?(面)**

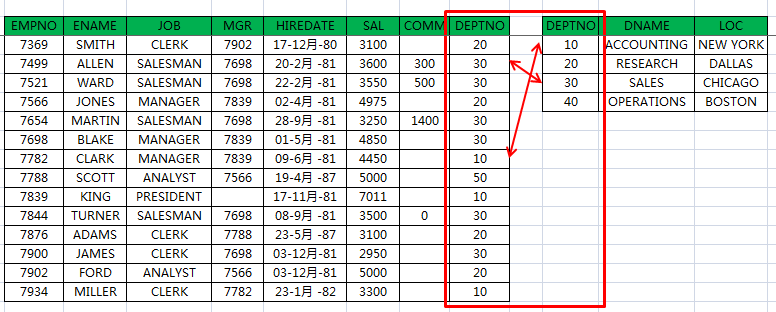
--TRUNCATE语句属于DDL语句,删除表中数据保留表结构,立即生效,不可以回滚,大数据量删除记录时效率高.

--DELETE语句属于DML语句,删除表中数据保留表结构,逐行删除记录,大数据量删除效率低,可执行条件删除,若不指定条件是全表删除.

--DROP语句是DDL语句,是删除数据库中表的操作语句.

1. **数据库中关联查询?**

在实际工作,经常要查询两个或两个以上数据表或视图的查询叫做连接查询，连接查询通常建立在存在相互关系的父子表之间.两张表之间需要有相同的字段作为连接条件.



上表:员工表和部门表中都有一个相同的字段DEPTNO,在做连接查询时可作为连接条件.

比如: 查看每个员工的名字以及所在部门的名字?

SELECT e.ename,d.dname

FROM emp e,dept d

WHERE e.deptno=d.deptno

该条查询语句就是建立在两张表的连接查询,FROM子句后跟两张表的表名,用逗号隔开,WHERE过滤子句将两张表中相同的字段作为连接条件进行关联.为了区别连接条件中分别是哪张表的数据,会只要别名来区分.

在关联查询中WHERE子句除了可以进行相同字段连接,同时也可以再次进行过滤,如下:

查看SALES部门的员工信息:

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

FROM emp e,dept d

WHERE e.deptno=d.deptno

AND d.dname='SALES'

1. **笛卡尔积?(面)**

在关联查询中,若不指定连接条件,则会产生笛卡尔积,该结果集会将每张表中的每条记录分别连接一次并组成一条记录,开销巨大,通常是一个没有意义的结果集,结果集条数是表中记录数的乘积,非特殊应用需要避免.

比如:A表中有10条数据,B表中有8条数据,产生的笛卡尔积结果集条目数是80条.

1. **内连接?(面)**

在做多表关联查询时,除了连接查询以外,内连接也可以实现关联查询. 无论是关联查询还是内连接,都忽略不满足连接条件的记录.

内连接关键字:JOIN ... ON...,其中JOIN 关键字关联两张表,ON是两张表的关联条件

比如: 查看SALES部门的员工信息

SELECT e.ename,d.dname

FROM emp e JOIN dept d

ON e.deptno=d.deptno

WHERE d.dname='SALES';

1. **外连接? (面)**

内连接是将满足连接条件的记录查询出来,外连接是将不满足条件的记录也查询出来,具体分为左外连接,右外连接,全外连接.

左外连接:LEFT OUTER JOIN ... ON... 以JOIN左侧的表为驱动表(主要显示数据的表),该表中的所有记录都会被查询出来,那么当某条记录不满足连接条件时,那么来自JOIN右侧表的字段全部以NULL作为值, 比如:

SELECT e.ename,d.dname

FROM emp e LEFT OUTER JOIN dept d

ON e.deptno=d.deptno

注:OUTER关键字可以省略

右外连接:RIGHT OUTER JOIN ... ON... 以JOIN右侧的表为驱动表(主要显示数据的表),该表中的所有记录都会被查询出来,那么当某条记录不满足连接条件时,那么来自JOIN左侧表的字段全部以NULL作为值,比如:

SELECT e.ename,d.dname

FROM emp e RIGHT OUTER JOIN dept d

ON e.deptno=d.deptno

注:OUTER关键字可以省略

全外连接:FULL OUTER JOIN ... ON... 将两张表中满足条件的数据和不满足条件的数据都查询出来,结果是左外连接和右边连接查询结果的综合.

SELECT e.ename,d.dname

FROM emp e FULL OUTER JOIN dept d

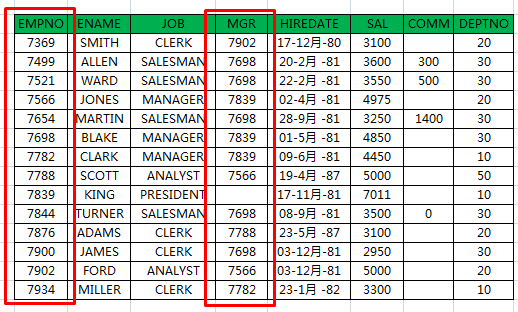
ON e.deptno=d.deptno

注:OUTER关键字可以省略

1. **自连接? (面)**

除了多张表关联之外,在查询中还会遇到数据的来源是一个表，将这张表通过别名虚拟成两张不同的表,关联关系来自于这张表中的多个列,可以使用等值或者不等值进行连接.

简单的说就是自己连接自己.



比如:员工表中员工的编号是另外一个员工的领导编号,现查询每个员工的领导信息?

SELECT e.ename,w.empno,w.ename

FROM emp e JOIN emp w

ON e.empno=w.mgr

1. **数据库表设计三范式?(面)**

范式(NF)简单的来说就是在关系型数据库中一张数据表的表结构所符合的某种设计标准的级别。注:关系型数据库: SQL Server，Oracle，MySQL等.

**第一范式(1NF)**: 表中字段不可再次拆分.

简单说就是表中的字段要符合原子性,不可再分,如果非要拆分,可分成两个不同的字段表示.

如下:



上表中城市和工资字段违背了第一范式,不可再次拆分,如果非要拆分可以将省份/县市分成两个独立的字段,月薪和奖金分成两个独立的字段.如下表:



**第二范式(2NF)**: 基于第一范式(1NF)基础上1表中有主键，非主键字段依赖于主键,一个表只说明一个事物

如下表:虽满足第一范式(1NF),但仍会出现**数据冗余,** **插入异常，删除异常，修改异常**的问题.



**数据冗余**:上表中每一个学生的学号,姓名,系名,系主任出现较多数据重复,造成数据冗余过大.

**插入异常**:关系中的某个属性或者某几个属性的组合，用于区分每个元组（可以把“元组”理解为一张表中的每条记录，也就是每一行）称为码,要求码相关的属性不能为空,假设增加了新的系--计算机系,但是没有学生,不能单独将该系插入到表中,因为码有不能为空的约束.则会造成插入异常.

**删除异常**:如果将某个系下的学生的数据都删除,那么该系及系主任的数据也会随之消失.

**修改异常:**若李小明同学要转到法律系,那么需要将其系名,系主任做3次修改才能修改完成.

为了避免如上异常,对表重新进行调整, 让表中有主键，非主键字段依赖于主键,调整后如下:



上表将一张表拆分为2张表,生成了学号作为主键,让其他非主键字段都依赖该主键.两张表各自表示各自的事物,一个是学生信息表,一个是成绩表.

虽然满足了第二范式(2NF)的规范,但问题并未一次性解决:

数据冗余: 学生的姓名、系名与系主任，不再像之前一样重复那么多次了---有改进

插入异常: 插入一个尚无学生的计算机系的信息,由于码信息不能为空---无改进

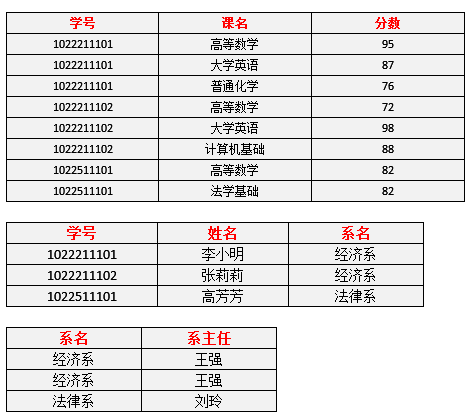
删除异常: 删除某个系中所有的学生记录,系名,系主任还会消失---无改进

修改异常: 李小明转系到法律系,只需要修改一次就可以了---有改进.

以上仅仅是满足了第二范式(2NF)的规范,但问题还是存在的, 出现问题的原因，在于仍然存在非主属性系名,系主任对于学号的传递依赖.

**第三范式(3NF)**: 基于第二范式(2NF)基础上非主键字段不能相互依赖, 每列都与主键有直接关系，不存在传递依赖.

再次对表进行修改,如下:



上表经过重新设计后:

选课表:符合3NF的要求，之前已经分析过了。

学生表:码为学号，主属性为学号，非主属性为系名，不存在传递依赖，符合3NF的要求。

系表:码为系名，主属性为系名，非主属性为系主任，不存在传递依赖（至少要有三个属性才可能存在传递依赖关系），符合3NF的要求。

那么再开看一下,第二范式(2NF)所未改进的部分内容是否通过第三范式(3NF)得到改进:

删除异常:删除某个系中所有的学生记录,该系的信息不会丢失。——有改进

插入异常:插入一个尚无学生的新系,系表与学生表是独立的两张表不影响。——有改进

数据冗余:数据冗余更加少了。——有改进

1. **子查询?(需自行多分析,多练习,笔试题必考,因需求不同本题仅告知需求思路分析)**

子查询是嵌套在其他查询语句中的作用是为了外层SQL语句提供数据,子查询常用于DQL,但是也可以用于DML和DDL之中.

子查询根据查询结果不同,有如下分类(给出一些参考方式,供选择95%的问题都可以解决):

子查询返回单行单列:HAVING WHERE

子查询返回单行多列:WHERE

子查询返回多行多列:FROM

**1).在WHERE子句中使用子查询**

WHERE控制数据行的显示,在WHERE子句中,子查询的返回一般是单行单列,单行多列,

多行单列

**1.1)子查询返回单行单列**

**例:要求统计出所有高于公司平均工资的员工信息?**

第一步:查询出公司平均工资,返回的是一个数值(单行单列)

SELECT AVG(sal) FROM emp

第二步:以上的查询返回的是单行单列数据,可以直接在WHERE子句中使用

SELECT ename,sal FROM emp

WHERE sal>( )

第三步:合并

SELECT ename,sal FROM emp

WHERE sal>( SELECT AVG(sal) FROM emp )

**例:查询出公司最早入职的员工信息?**

第一步:查询出公司最高的入职日期,返回的是一个日期(单行单列)

SELECT MIN(hiredate) FROM emp

第二步:以上的查询返回的是单行单列,所以在WHERE子句中使用

SELECT \* FROM emp

WHERE hiredate=( )

第三步:合并

SELECT \* FROM emp

WHERE hiredate=( SELECT MIN(hiredate) FROM emp)

**1.2)子查询返回单行多列(了解)**

**例:统计公司入职最早,工资最低的员工?(巧合数据库中刚好有一员工入职最早,工资最低)**

第一步:查询出入职时间最早,工资最低

SELECT MIN(hiredate),MIN(sal) FROM emp

第二步:查询结果返回的是单列两列,只能在WHERE中使用

SELECT \* FROM emp

WHERE (hiredate,sal)=( )

第三步:合并

SELECT \* FROM emp

WHERE (hiredate,sal)=( SELECT MIN(hiredate),MIN(sal) FROM emp)

**例:查询与SCOTT工资相同,职位相同的员工信息?**

第一步:查询出SCOTT工资和职位,返回的是单行多列,只能使用WHERE

SELECT sal,job FROM emp

WHERE ename='SCOTT'

第二步:查询出的结果是单行多列

SELECT \* FROM emp

WHERE (sal,job)=( )

第三步:合并

SELECT \* FROM emp

WHERE (sal,job)=( SELECT sal,job FROM emp

WHERE ename='SCOTT')

第四步:结果集中包含SCOTT,不符合业务需求,去除SCOTT

SELECT \* FROM emp

WHERE (sal,job)=( SELECT sal,job FROM emp

WHERE ename='SCOTT')

AND ename<>'SCOTT';

**1.3)子查询返回多行单列(比较麻烦的一种查询)**

如果子查询返回的是多行单列的数据,实际上相当于一个数据查询的查询范围,那么如果要针对范围查询则要使用三个判断符号:

IN(在范围内) ANY(其中之一) ALL(所有)

1)例:IN操作:此功能在某个范围之内

如下SQL返回的结果是单行多列

**查询职位是MANAGER(经理职位)的工资?**

SELECT sal FROM emp WHERE job='MANAGER';

**查询工资在MANAGER(经理职位)范围内的员工信息?**

SELECT \* FROM emp

WHERE sal IN(SELECT sal FROM emp WHERE job='MANAGER');

注:也可以使用NOT IN 不在某个范围内, 但需要注意一个问题:在使用NOT IN时,子查询中不能包含NULL,否则不会有任何数据返回

**查询工资不在MANAGER(经理职位)范围内的员工信息?**

SELECT \* FROM emp

WHERE sal NOT IN(SELECT sal FROM emp WHERE job='MANAGER');

如下案例:MANAGER(经理职位)的奖金comm为NULL,会查询出3条NULL

SELECT comm FROM emp WHERE job='MANAGER';

执行如下SQL会出现未选定行

SELECT \* FROM emp

WHERE sal NOT IN(SELECT comm FROM emp WHERE job='MANAGER');

2)ANY操作,此操作一共分为三种方式

=ANY(等于其中之一),此功能结果与IN的结果一样

SELECT \* FROM emp

WHERE sal =ANY(SELECT sal FROM emp WHERE job='MANAGER');

>ANY:比子查询的最小值要大

SELECT \* FROM emp

WHERE sal >ANY(SELECT sal FROM emp WHERE job='MANAGER');

<ANY:比子查询的最大值要小

SELECT \* FROM emp

WHERE sal <ANY(SELECT sal FROM emp WHERE job='MANAGER');

3)ALL操作:此操作分为两类  
>ALL:比子查询范围的最大值还要大

SELECT \* FROM emp

WHERE sal >ALL(SELECT sal FROM emp WHERE job='MANAGER');

<ALL:比子查询的最小的最小值还要小

SELECT \* FROM emp

WHERE sal <ALL(SELECT sal FROM emp WHERE job='MANAGER');

**例:要求查询出工资比30部门还要高的员工信息?**

第一步:查询出30部门的工资,返回多行单列

SELECT sal FROM emp WHERE deptno=30;

第二步:以上结果为多行单列,查询出比子查询的最大值还要大的员工信息

SELECT \* FROM emp WHERE sal>ALL( )

第三步:合并

SELECT \* FROM emp

WHERE sal>ALL( SELECT sal FROM emp WHERE deptno=30);

对于给定的多行单列查询,只能通过以上的三个符号完成

**2)在HAVING 子句中使用子查询**

如果使用了HAVING子句一定意味着进行了分组,而且进行了统计查询,在HAVING之中

出现的子查询**,**只能够返回单行单列的数据,没有向WHERE这么复杂的应用

**例:查询出高于公司平均工资的部门编号,平均工资?**

第一步:查询出公司的平均工资,返回的结果是单行单列

SELECT AVG(sal) FROM emp

第二步:要在分组之后进行过滤,查询出高于如上结果的部门编号和平均工资

SELECT deptno,AVG(sal)

FROM emp

GROUP BY deotno,AVG(sal)

HAVING AVG(sal)>( )

第三步:合并

SELECT deptno,AVG(sal)

FROM emp

GROUP BY deptno

HAVING AVG(sal)>( SELECT AVG(sal) FROM emp)

**例:查询出平均工资最低的职位信息,人数及平均工资?**

第一步:找到平均工资最低的职位,这个操作需要使用聚合函数嵌套

SELECT MIN(AVG(sal)) FROM emp

GROUP BY job

第二步:嵌套之后无法在出现分组字段,所以现在可以将以上单行单列的结果定义在HAVING子句之中

SELECT COUNT(job) ,AVG(sal) FROM emp

GROUP BY job

HAVING AVG(sal)<( )

第三步:合并

SELECT COUNT(job),AVG(sal) FROM emp

GROUP BY job

HAVING AVG(sal)<( SELECT MIN(AVG(sal)) FROM emp

GROUP BY job)

**3)在FROM子句中使用子查询**

在FROM 子句中使用子查询通常返回的都是多行多列(包含单行多列),常常被当做一张

表使用.

**例:查看比自己所在部门平均工资高的员工?**

第一步:先查到每个部门的平均工资和部门编号,返回多行多列数据,作为一张表来使用,需

要用GROUP BY针对部门编号分组.

SELECT AVG(sal) avg\_sal,deptno

FROM emp

GROUP BY deptno

第二步:再查看每个部门的员工信息以及每个部门的平均工资,需要将刚才查询出来的部

门平均工资表进行关联查询

SELECT e.ename,e.sal,e.deptno, t.avg\_sal

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

FROM emp

GROUP BY deptno) t  
 WHERE e.deptno=t.deptno

第三步:拿到了每个部门的员工信息及平均工资,就可以将每个员工工资低于部门平均工

资的员工给过滤掉,只显示每个部门员工工资高于部门平均工资的员工.

SELECT e.ename,e.sal,e.deptno, t.avg\_sal

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

1. **EXISTS和NOT EXISTS的用法?**

**EXISTS:**用在过滤条件中,其后需要跟一个子查询,只要该子查询可以查询出一条记录,那

么EXISTS表达式就返回真.

**NOT EXISTS:**没有查到记录返回真.

**比如:查看有员工的部门信息:**

SELECT deptno, dname

FROM dept d

WHERE EXISTS

(SELECT \* FROM emp e

WHERE e.deptno=d.deptno)

**比如:查看没有员工的部门信息:**

SELECT deptno, dname

FROM dept d

WHERE NOT EXISTS

(SELECT \* FROM emp e

WHERE e.deptno=d.deptno)

1. **ROWNUM是什么? (面)**

ROWNUM是Oracle数据库中的一个伪列,他不存在与任何一张表中,但是每张表都可以查询该字段,该字段的值为查询到的结果集中每条记录的行号,只能查到结果后才会对结果集从1开始为每条记录指定行号, 不能通过ROWNUM做>1以上数字的判断的,否则将查询不出任何结果.

先看错误案例:

SELECT ROWNUM,ename,sal,deptno

FROM emp

WHERE ROWNUM >1

该案例查询不到任何结果,.

正确案例如下:

SELECT ROWNUM,ename,sal,deptno

FROM emp

WHERE ROWNUM<5

此案例能查出4条结果,其中ROWNUM当做一个字段显示在结果集中.

1. **Oracle和MySQL的端口号? (面)**

Oracle端口号:1521

MySQL端口号:3306

1. **数据库分页查询中的真分页和假分页?(面)**

假分页：从数据库中取出所有的数据，然后分页在界页面上显示。访问一次数据库，但由于选择的数据量比较大，所以第一次花费时间比较长，但之后每一页的显示都是直接、快速的，避免对数据库的多次访问。

真分页：确定要显示的数量和内容，然后每次都去数据库取出该少量数据，优点是数据量小，缺点是访问数据库频繁。在大型网站中往往采用真分页。

注:假分页，如果数据量较多，在首次页面加载的时候会比较慢，严重影响用户体验.

真分页, 对于大数据量可以从容的面对,每次点击下一页，客户端访问服务器的时间基本一样，提高用户体验，更加的人性化。

1. **Oracle和MySQL的分页查询?(面):重点务必代码写熟**

**Oracle分页:**需要用到ROWNUM:

**查看公司工资排名的6-10名?**

SELECT \*

FROM(SELECT ROWNUM rn,t.\*

FROM(SELECT ename,sal,deptno

FROM emp

ORDER BY sal DESC) t

WHERE ROWNUM <=10)

WHERE rn >=6

第一步:先将要查询的数据进行排序若在分页中有排序需求时,那么应当先进行排序,因为排序的优先级是最低的.

SELECT ename,sal,deptno

FROM emp

ORDER BY sal DESC

第二步:拿到排序后的结果集后将该结果集作为一张表进行查询,并使用ROWUNM进行编号并过滤出只显示前10条数据

SELECT ROWNUM rn,t.\*

FROM(SELECT ename,sal,deptno

FROM emp

ORDER BY sal DESC) t

WHERE ROWNUM <=10

第三步:拿到前10条数据的结果集后再把该结果当做一张表对该表进行过滤查询,过滤出第6条到第10条的数据.

SELECT \*

FROM(SELECT ROWNUM rn,t.\*

FROM(SELECT ename,sal,deptno

FROM emp

ORDER BY sal DESC) t

WHERE ROWNUM <=10)

WHERE rn >=6

实际开发业务中<=10和>=6是被通过公式计算出来的,<=10等于start,>=6等于end

pageSize:每页显示的条目数

page:第几页

根据上述两个参数,计算结果集的范围:

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

end:pageSize\*page

**MySQL的分页:**需要用到limit

MySQL的分页查询相对于Oracle简单很多,只需要使用limit关键字后面跟2个参数,第一个参数是起始的查询位置,第二个参数是每次查询多少条,需要注意,MySQL中第一条数据是从0开始.

**假设查询第前10条数据,SQL语句为:**

SELECT \* FROM emp LIMIT 0,10;

1. **DECODE函数? (面)**

DECODE函数,可以实现分支效果语法含义解释：

DECODE(条件,值1,翻译值1,值2,翻译值2,...值n,翻译值n,缺省值)

其中第一个参数为条件,之后的每两个参数为做一组,如果条件等于后面的每组中的值,那么返回后面的翻译值,相当于如下写法：

IF 条件=值1 THEN

RETURN(翻译值1)

......

ELSIF 条件=值n THEN

RETURN(翻译值n)

ELSE

RETURN(缺省值)

END IF

**比如:给不同职位的员工涨工资,按照不同的职位涨幅不同**

SELECT ename, job, sal,

DECODE(job,

'MANAGER',sal\*1.2,

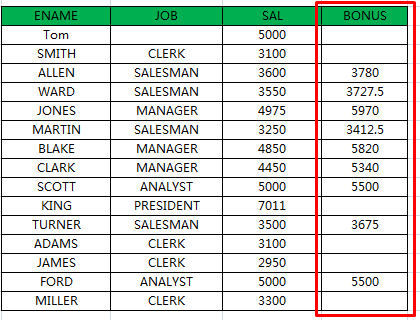
'ANALYST',sal\*1.1,

'SALESMAN',sal\*1.05

) bonus

FROM emp;

如上案例,DECODE得出的结果会被当做一个字段显示在查询结果中,字段名为bonus,符合job条件的职位会被涨薪,没有职位或者不符合职位条件的都不涨薪.



**CASE也可以实现DECODE相同的效果:**

SELECT ename,job,sal,

CASE job WHEN 'MANAGER' THEN sal \*1.2

WHEN 'ANALYST' THEN sal \* 1.1

WHEN 'SALESMAN' THEN sal \* 1.5

ELSE null

END

bonus

FROM emp

如上案例和DECODE函数结果一致,当CASE后面的job=WHEN后面的职位时则返回

THEN后的值,不等于这三个职位的话则执行ELSE后面的值返回NULL值.

1. **排序函数?**

排序函数可以根据结果集按照指定的字段分组,然后在按照指定的字段在各组内排序,然后生成组内编号.共3个:

**ROW\_NUMBER**:生成组内连续且唯一的数字

**RANK**:生成组内不连续也不唯一的数字

**DENSE\_RANK**:生成组内连续但不唯一的数字

具体案例如下:

1)ROW\_NUMBER:生成组内连续且唯一的数字

**查看各部门的工资排名?**

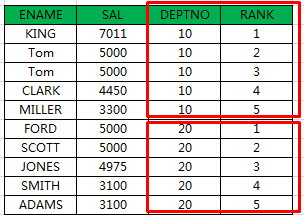
SELECT ename,sal,deptno,

ROW\_NUMBER() OVER(

PARTITION BY deptno ORDER BY sal DESC ) rank

FROM emp

以上案例OVER中的PARTITION BY后的字段是要分组的字段,ORDER BY 后的字段是根据前面分组的字段进行组内排序,然后生成组内连续且唯一的数字

.

2)RANK:生成组内不连续也不唯一的数字

**查看各部门的工资排名?**

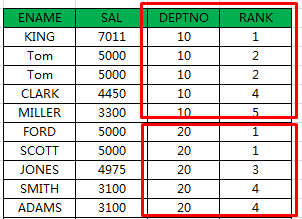
SELECT ename,sal,deptno,

RANK() OVER(

PARTITION BY deptno ORDER BY sal DESC ) rank

FROM emp

以上案例OVER中的PARTITION BY后的字段是要分组的字段,ORDER BY 后的字段是根据前面分组的字段进行组内排序,然后生成组内不连续也不唯一的数字



3)DENSE\_RANK:生成组内连续但不唯一的数字

**查看各部门的工资排名?**

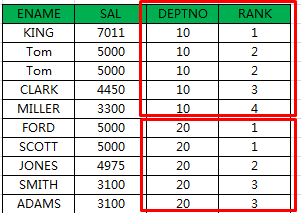
SELECT ename,sal,deptno,

DENSE\_RANK() OVER(

PARTITION BY deptno ORDER BY sal DESC ) rank

FROM emp

以上案例OVER中的PARTITION BY后的字段是要分组的字段,ORDER BY 后的字段是根据前面分组的字段进行组内排序,然后生成组内连续但不唯一的数字



1. **数据库中集合操作? (面)**

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

并集: UNION、UNION ALL

交集: INTERSECT

差集:MINUS

语法:

SELECT 查询语句1

交集 | 并集 | 差集

SELECT 查询语句2

在两个SELECT查询语句之间是有UNION、UNION ALL、INTERSECT、MINUS关键字实现两个结果集的合并操作,ORDER BY子句只能放在最后的一个查询语句中。

并集: UNION、UNION ALL

UNION: 返回两个以上结果集的并集会自动去掉合并后的重复记录。

UNION ALL: 返回两个结果集中的所有行，包括重复的行。

**例如，合并职位是’MANAGER’的员工和薪水大于2500的员工集合.**

UNION:去重复

SELECT ename, job, sal FROM emp

WHERE job = 'MANAGER'

UNION

SELECT ename, job, sal FROM emp

WHERE sal> 2500;

UNION ALL:不去重

SELECT ename, job, sal FROM emp

WHERE job = 'MANAGER'

UNION ALL

SELECT ename, job, sal FROM emp

WHERE sal> 2500;

交集: INTERSECT

获得两个结果集的交集，两个结果集中都有的数据，才被显示输出, 结果集会以第一列的数据作升序排列.

**例如：显示职位是’MANAGER’的员工和薪水大于2500的员工的交集**

SELECT ename, job, sal FROM emp

WHERE job = 'MANAGER'

INTERSECT

SELECT ename, job, sal FROM emp

WHERE sal> 2500;

差集:MINUS

MINUS函数获取两个结果集的差集,在结果集1中有的,结果集2中没有的才会被显示输出.

**例如：列出职位是MANAGER但薪水低于2500的员工记录**

SELECT ename, job, sal FROM emp

WHERE job = 'MANAGER'

MINUS

SELECT ename, job, sal FROM emp

WHERE sal> 2500;

1. **什么是视图? (面)**

**VIEW:**视图是数据对象之一,不是一张真实存在的表,而是一张虚表,是一条SELECT语句结果集数据的逻辑显示,给这个结果集起个名字就是视图的名字.

创建一个视图:

CREATE OR REPLACE VIEW v\_emp

AS

SELECT empno,ename,sal,deptno

FROM emp

WHERE deptno=10;

注:OR REPLACE:有了替换没有了创建

查看视图的结构:

DESC v\_emp

查看视图内的数据:

SELECT \* FROM v\_emp;

视图的分类:根据子查询种类分为简单视图和复杂视图

简单视图:SELECT语句是基于单表创建的且不包含任何函数运算,表达式或者分组函数的视图称为简单视图.

复杂视图:除简单视图之外的都是复杂视图.

需要注意两个问题:

对复杂视图进行DML操作时其实操作的都是基表中的数据,这样会对基表中数据造成污染,DML中的DELETE操作除外.

为了避免出现污染情况可以对视图添加检查选项,因为添加选项后要求对视图进行DML操作后的数据必须对视图课间才允许操作.

添加检查选项:

CREATE OR REPLACE VIEW v\_emp

AS

SELECT empno,ename,sal,deptno

FROM emp

WHERE deptno=10

WITH CHECK OPTION;

上述案例中:WITH CHECK OPTION短语表示，通过视图所做的修改，必须在视图的可见范围内才可以执行操作,否则执行失败：

假设INSERT，新增的记录在视图仍可查看

假设UPDATE，修改后的结果必须能通过视图查看到

假设DELETE，只能删除现有视图里能查到的记录

对于简单视图进行DML虽然合法,但也不安全,如果没有在视图上进行DML操作的必要可以对其设置只读模式: READ ONLY.

创建只读视图:

CREATE OR REPLACE VIEW v\_emp

AS

SELECT empno, ename, sal, deptno FROM emp

WHERE deptno = 10

WITH READ ONLY;

删除视图:

DROP v\_emp;

视图虽然是放在数据字典中的独立对象,但视图仅仅是基于表的一个查询定义,所以对视图的删除不会影响到基表中的数据.

1. **什么是序列? (面)**

**序列(SEQUENCE)**是一种用来生成唯一数字值的数据库对象,可以按照指定的方式生成唯一序列数字,最常用于给表的主键提供数据.

创建序列:

CREATE SEQUENCE seq\_emp

START WITH 1

INCREMENT BY 1

上述案例是创建了一个叫seq\_emp的序列,START WITH 1指的是从1开始,INCREMENT BY 1指的是每次增加1

序列支持两个伪列,以获取该序列的数字:

**NEXTVAL**:使序列生成一个数字,若是刚创建的序列,这是返回START WITH后指定的数字,以后按照INCREMENT BY 1的步进值生成一个数字并返回

**CURRVAL**:获得当前序列租后生成的数字

**案例:**SELECT seq\_emp.NEXTVAL FROM dual;

SELECT seq\_emp.CURRVAL FROM dual;

注:CURRVAL要在新创建的序列至少调用一次NEXTVAL后可以使用,NEXTVAL会导致序列步进且不能回退.

在插入语句中使用序列:

INSERT INTO emp VALUES(seq\_emp.NEXTVAL,'zhansan',5000,20);

删除序列:

DROP SEQUENCE seq\_emp;

1. **什么是索引? (面)**

**索引(INDEX)**是为了提高查询效率的机制.

创建索引:

CREATE INDEX idx\_emp ON emp(ename);

上面案例时在对emp表中的ename字段创建一个叫idx\_emp的索引.

创建符合索引:

CREATE INDEX idx\_emp ON emp(job,sal);

上面案例时在对emp表中的job,sal两个字段创建一个叫idx\_emp的符合索引.

注:创建完毕符合索引后在做查询时如果有ORDER BY 后两个字段排序,排序时字段的顺序要和符合索引的字段顺序一致.

创建基于函数的索引:

如果需要在emp表的ename列执行大小写无关搜索,可以在此列建立一个基于UPPER函数的索引.

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

修改索引:

Oracle里大量删除记录后，表和索引里占用的数据块空间并没有释放,这是可以使用ALTER INDEX对索引进行重建,REBUILD 重建的意思.

ALTER INDEX idx\_emp REBUILD;

删除索引:

DROP INDEX idx\_emp;

1. **数据库中约束? (面)**

约束(CONSTRAINT):限制条件,均用于创建表时对字段进行添加限制条件.

约束类型:

非空约束(NOT NULL):简称NN

唯一性约束(UNIQUE):简称UK

主键约束(PRIMARY KEY):简称:PK

外键约束(FOREIGN KEY):简称:FK

检查约束(CHECK):简称:CK

非空约束(NOT NULL):确保字段值不为空

1. **非空约束**:确保字段值不为空

创建表时对字段添加约束:

CREATE TABLE emp(

id NUMBER(6),

name VARCHAR2(30) NOT NULL)

修改表时对字段添加约束:

ALTER TABLE emp MODIFY ( id NUMBER(6) NOT NULL);

取消非空约束:可采用重建表或者修改表的方式

ALTER TABLE emp MODIFY(id NUMBER(6) NULL);

1. **唯一性约束**:保证指定的字段不允许有重复的值(NULL除外)

创建表时添加唯一性约束:

CREATE TABLE emp(

id NUMBER(6) UNIQUE, --------------------创建表跟在字段后

name VARCHAR2(30),

email VARCHAR2(59),

CONSTRAINT emp\_uk UNIQUE(email) --------------字段写完单独设置,可设

--------------置多个,逗号隔开

);

修改表添加唯一性约束:

ALTER TABLE emp ADD CONSTRAINT emp\_uk UNIQUE(name);-----可设置多个

删除唯一性约束:

ALTER TABLE DROP CONSTRAINT emp\_uk;

注:emp\_uk是给唯一性约束起的名字.

1. **主键约束**:约定的字段非空且唯一,并且一个表中只能有一个主键或者联合主键

创建表时添加主键约束:

CREATE TABLE emp(

id NUMBER(6) PRIMARY KEY, ------------创建表时添加在字段后

name VARCHAR2(30));

创建表完毕在进行修改添加:

CREATE TABLE emp(

id NUMBER(6) ,

name VARCHAR2(30));

ALTER TABLE emp ADD CONSTRAINT emp\_pk PRIMARY KEY (id);---修改时添加

1. **外键约束**:一张表保存的是另一张表的主键字段的值, 两个方面的数据约束：

* 从表上定义的外键的列值，必须从主表被参照的列值中选取，或者为NULL；
* 当主表参照列的值被从表参照时，主表的该行记录不允许被删除。

注:因使用麻烦,关系复杂目前基本不使用这种方式.不再做详细叙述.

1. **检查约束**:用来强制在字段上的每个值都满足CHECK 中定义的条件

比如:设置员工工资必须大于2000的约束

ALTER TABLE emp ADD CONSTRAINT emp\_check CHECK(sal>2000);

对sal字段设置过检查约束时对sal字段执行的插入,更新语句该字段的值小于2000时会报错违反检查性约束.

1. **表级约束和列级约束? (面)**

**列级约束:**直接跟在字段后定义的约束只能当前字段有效,称为列级约束:非空,唯一,主键,外键,检查.

**表级约束**:可以单独定义,并且定义时可传入多个字段,逗号隔开,对多个字段同时有效的约束: 唯一,主键,外键,检查.

换言之简单的说就是:除了非空约束只能作为列级约束外,其他约束类型都可以作为列级又可以作为表级约束.

比如:能够通过ALTER TABLE ... ADD CONSTRAINT定义的约束都是表级约束:

ALTER TABLE emp ADD CONSTRAINT emp\_uk UNIQUE(id,name,email);

注意:主键约束作为表级约束时是联合主键,并不是添加多个主键约束,因为每个表中只能有一个主键.

1. **SQL语句的优化方式? (面)**

1)避免使用SELECT \* ,需要哪些字段将字段写在SELECT可节省查询时间,避免将不需

要的数据也查询出来.

2)对经常需要查询操作的字段设置索引,因为索引会增加查询效率.

3)模糊查询尽量不以%开头,当模糊匹配以%开头时,该字段索引将失效.

4)避免在索引字段上使用<>,!==,使用该操作索引会失效,造成全表扫描

5)避免在索引字段上使用IS NULL 和IS NOT NULL,该操作索引会失效,造成全表扫描

6)使用WHERE子句替代HAVING子句,HAVING对结果集进行过滤,能够使用WHERE

过滤掉的就不放在结果集中过滤.

7)exists代替in,oracle中in后子查询返回的结果不能超过1000条,使用exists替代in.

8)oracle的解析器是按照从右往左的顺序处理,多表的情况下,应将记录少的表放在右边

9)where语句是自上而下解析,关联查询中如果where后有关联条件又有过滤条件,应

将关联条件放在前边,过滤条件放在后面,让关联条件先过滤掉一部分

还有其他很多不在一一总结,但至少要给面试官说出来5条以上.

**以下为笔试题:请自行练习,无答案(笔试)**

1. **已知以下表结构（括号内为表名和字段名，用SQL写出以下题目答案）**

** **

****

（1）查询所有学生的信息（学号、姓名、性别、班级名称）

（2）查询所有人的课程分数（学号、姓名、性别、班级名称、语文分数、数学分数）

（3）查询所有班级的人数（班级编号、班级名称、人数）

（4）查询总分数（语文+数学）>=150的学生信息（学号、姓名、班级名称、总分数）

（5）查询所有班级的平均分数（班级编号、班级名称、语文平均分数、数学平均分数）

（6）查询各科都合格【分数>=60分】的学生（学号、姓名、语文分数、数学分数）

（7）查询有挂科【分数<60分】现象的学生（学号、姓名、语文分数、数学分数）

（8）查询班级人数>=30的班级（班级编号、班级名称、人数）

（9）查询没有参加考试【没有成绩表】的学生（学号、姓名、性别、班级名称）

（10）假设分数>=60分合格，分析学生的成绩是否合格 （学号、姓名、语文合格情况[合

格/不合格]、数学合格情况[合格/不合格]）

1. **有如下3张表,完成如下需求:**



(1)用户表中添加一条数据,请写出SQL语句

(2)在用户表中把姓名为"李四"的在职用户的年龄修改为2

(3)查询所有的角色名称为"技术人员",性别是女的在职人员

(4)把角色名称为"研发人员"的用户"张三"从用户表中删除

(5)查询年龄在18到24岁之间的男性在职人员

(6)查询年龄为18,19,20的男性和女性员工的人数,结果如下图所示:



(7)查询前50名姓张的男员工

(8)查询所有根角色员工,根角色就是有父角色为null的角色.

(9)查询员工姓名为"张三"的员工对应的角色信息

1. **有如下表,请按照需求写出对应SQL语句:**



(1)创建studen t表,并以下插入数据:

1,张三,20,男;

2,李四,21,女;

3,王五,22,男;

4,赵六,23,男;

5,钱七,24,女;

6,孙八,25,男;

(2)创建course表,并插入以下数据:

001,语文,1;

002,数学,2;

003,英语,3;

004,化学,4;

005,物理,5;

(3)创建sc表并插入以下数据:

1,001,60;

1,002,75;

1,003,85;

2,001,70;

2,002,85;

2,003,55;

2,005,70;

3,001,72;

3,004,68;

3,005,80;

4,002,70;

4,003,80;

4,004,69;

4,005,73;

5,001,75;

5,003,69;

5,004,89;

(4)创建teacher表并插入如下数据:

1,苍老师;

2,李老师;

3,王老师;

4,范老师;

5,叶平;

(5) 修改张三同学的语文成绩为80分

(6) 删除张三同学的数学课程的成绩

(7) 查询语文课程比数学课程成绩高的所有学生的学号

(8) 查询没学过叶平老师课的同学的学号和姓名

(9) 查询前50名学生平均成绩及其名次

(10) 查询出只选修了一门课程的全部学生的学号和姓名

(11) 查询姓张的学生名单

1. **有如下student表,请按照以下需求写出SQL语句**



(1)修改id为10的学生信息

(2)复制student表到student1中,仅复制表结构,不复制表数据

(3)复制student表到student2中,包含表结构及数据

(4)如果该数据库是mysql,请使用分页查询出age从大到小排名5到10名的学生信息

(5)如果该数据库是oracle,请使用分页查询出age从小到大排名5到10名的学生信息

1. **有如下3张表,请按照以下需求写出SQL语句**



(1)使用标准SQL嵌套语句查询选修课程为"税收基础"的学员学号和姓名

(2)使用标准SQL嵌套语句查询选修课程编号为"C2"的学员姓名和所属单位

(3)使用标准SQL嵌套语句查询不选修课程编号"C5"的学生姓名和所属单位

(4)查询选修了课程的学员人数

(5)查询选修课程超过5门的学员学员和所属单位

1. **有如下一张表,按照右侧的结果格式写出生成右侧结果的SQL语句**



(1)按照右侧的结果格式写出生成右侧结果的SQL语句

1. **有如下3张表,请按照以下需求写出SQL语句**



(1)删除员工表中重复的员工数据,只保留重复数据中的一条

(2)按部门名称查询出员工的人数大于等于100部门

(3)把所有员工的姓名、性别和年龄显示出来

(4)查询部门表名称，当部门表的名称为空时，用0来代替

(5)按员工编号进行从小到大排序，要求查询出第15行到第20行的员工编号和姓名

(6)查询哪个部门没有员工（要求使用使用表关联）

1. **有如下3张表,请按照需求写出对应SQL**



(1)写sql语句查询出表a中有学生成绩重复的记录(两条记录科目和名称两个字段的值

相同即视为重复记录)

(2)写sql语句查询出表a中学生成绩重复次数大于2以上的学生

(3)写sql语句删除a表中重复记录,保留重复记录中分数最大的.

(4)写sql语句在表b中求出各年级各科目的平均分数和最大分数

(5)写sql语句在表b中 查询语文平均分数大于3的班级

(6)写sql语句将表b行列转换为表c

(7)写sql语句将表c行列转换为表b