Oracle 数据库

1.数据库简介

1.1 什么是数据库

简单的说，数据库（database）就是一个存放数据的仓库，这个仓库是按照一定的数据结构（数据结构是指数据的组织形式或数据之间的联系）来组织、存储的，我们可以通过数据提供的多种方法来管理数据库里的数据。数据结构有解构数据的功能吗？

当人们收集了大量的数据后，应该把它们保存起来进入近一步的处理，进一步的抽取有用的信息。当年人们把数据存放在文件柜中，可现在随着社会的发展，数据量急剧增长，现在人们就借助计算机和数据库技术科学的保存大量的数据，以便能更好的利用这些数据资源。

1.2 数据库的分类

数据结构不同？

数据库通常分为层次式数据库，网络式数据库和关系式数据库三种，但是现在主流的是关系型数据库。关系型数据库是把复杂的数据结构归结为简单的二元关系（即二维表格形式）。对数据的操作几乎全部建立在一个或多个关系表格上，通过对这些关联的表格分类、合并、连接或选取等运算来实现数据库的管理。

不同的数据关系存放为不同的表格，表格之间可以相互调用

关系型数据库诞生 40 多年了，从理论产生发展到现实产品，例如：Oracle 和 MySQL，Oracle 在数据库领域上升到霸主地位，形成每年高达数百亿美元的庞大产业市场

硬盘中的数据本身是什么状态？是以表的形式存在吗？也许是，但是关系型数据库在对数据进行管理时会将这些数据的数据关系拆分成二元关系，然后在表空间中对表进行操作，对吗？也许不是，那么数据库会根据什么标准对数据的数据关系进行解构呢？数据的存放是根据不同的数据结构来完成的？

1.3 常用的关系型数据库

1.3.1 Oracle 数据库

这些数据库的数据是怎么写入进去的？以什么方式，通过什么手段？

ORACLE 数据库系统是美国 ORACLE 公司（甲骨文）提供的以分布式数据库为核心的一组软件产品，是目前最流行的客户/服务器(CLIENT/SERVER)或 B/S 体系结构的数据库之一。比如 SilverStream 就是基于数据库的一种中间件。ORACLE 数据库是目前世界上使用最为广泛的数据库管理系统，作为一个通用的数据库系统，它具有完整的数据管理功能；作为一个关系数据库，它是一个完备关系的产品；作为分布式数据库它实现了分布式处理功能。但它的所有知识，只要在一种机型上学习了 ORACLE 知识，便能在各种类型的机器上使用它。

Oracle 数据库最新版本为 Oracle Database 12c。Oracle 数据库 12c 引入了一个新的多承租方架构，使用该架构可轻松部署和管理数据库云。此外，一些创新特性可最大限度地提高资源使用率和灵活性，如 Oracle Multitenant 可快速整合多个数据库，而 Automatic Data Optimization 和 Heat Map 能以更高的密度压缩数据和对数据分层。这些独一无二的技术进步再加上在可用性、安全性和大数据支持方面的主要增强，使得 Oracle 数据库 12c 成为私有云和公有云部署的理想平台。

1.3.2 MySQL 数据库

MySQL 是一种开放源代码的关系型数据库管理系统（RDBMS），MySQL 数据库系统使用最常用的数据库管理语言--结构化查询语言（SQL）进行数据库管理。

由于 MySQL 是开放源代码的，因此任何人都可以在 General Public License 的许可下下载并根据个性化的需要对其进行修改。MySQL 因为其速度、可靠性和适应性而备

受关注。大多数人都认为在不需要事务化处理的情况下，MySQL 是管理内容最好的选择。

1.3.3 SQL Server 数据库

SQL Server 是由 Microsoft 开发和推广的关系数据库管理系统（DBMS），它最初是由 Microsoft、Sybase 和 Ashton-Tate 三家公司共同开发的，并于 1988 年推出了第一个 OS/2 版本。Microsoft SQL Server 近年来不断更新版本，1996 年，Microsoft 推出了 SQL Server 6.5 版本；1998 年，SQL Server 7.0 版本和用户见面；SQL Server 2000 是 Microsoft 公司于 2000 年推出，目前最新版本是 2012 年 3 月份推出的 SQL SERVER 2012。

1.4 Oracle 数据库的基本概念

1.4.1 Oracle 服务器

Oracle 服务器是一个数据管理系统（RDBMS），它提供开放的，全面的，近乎完整的信息管理。是由 Oracle 数据库和 Oracle 实例组成。

1.4.2 Oracle 数据库和 Oracle 实例

Oracle 数据库：就是位于硬盘上实际存放数据的文件（控制文件，数据文件，日志文件等等），Oracle 数据库必须要与内存里的实例合作，才能对外提供数据管理服务。

Oracle 实例：位于内存里的数据结构。它由一个共享的内存池和多个后台进程组成，共享的内存池可以被所有的进程访问。用户如果要存取数据库（也就是硬盘上的文件）里的数据，必须通过实例才能实现，不能直接读取硬盘上的文件。

区别：实例可以操作数据库；在任何时刻一个实例只能与一个数据库关联；大多数情

况下，一个数据库只有一个实例对其进行操作。

1.4.3 Oracle 数据库的体系结构

用户进程可以是一般的客户端软件，像 Oracle 的 sqlplus，sql developer，或者是一些驱动程序（比如我们后面需要学习的 JDBC）等都属于用户进程。

服务器进程有时也会称为前台进程，当然这是相对于后台进程（比如 DBWR：数据库读写器，LGWR：日志写入器等）来说的，服务器进程的主要作用就是处理连接到当前实例的用户进程的请求，对客户端发来的 sql 进行执行并返回执行结果。

数据缓冲区：是 Oracle 用来执行 sql 的工作区域，在更新数据时，用户会话不会直接去更新磁盘上的数据，而是把数据缓存到数据缓冲区，等到用户提交事务，数据才会通过 DBWR（数据读写器）写入到硬盘的数据文件中。

日志缓冲区：是一块比较小的内存区域，它是用来短期存储将要写入到磁盘中的重做日志文件的。日志缓冲区存在的意义依然是为了减小磁盘 IO，减少用户的等待时间。

共享池：是最复杂的 SGA 结构，它有许多的子结构：

- 库缓存：库缓存这块内存区域会按已分析的格式缓存最近执行的代码，这样，同样的 sql 代码执行多次，就不用重复去进行代码分析，可以很大程度上提高系统性能。
- 数据字典缓存：存储 oracle 中对象定义（表，视图，同义词，索引等数据库对象），这样在分析 sql 代码时，就不用频繁的去磁盘读取数据字典中的数据了。
- PL/SQL 区：缓存存储过程，函数，触发器等数据库对象

数据库文件： 从物理上划分，数据库文件有三种：

- ctl 后缀文件： 控制文件，用来存放登录信息
- dbf 后缀文件：数据库文件，用来存放数据
- log 后缀文件：日志文件，用来记录日志

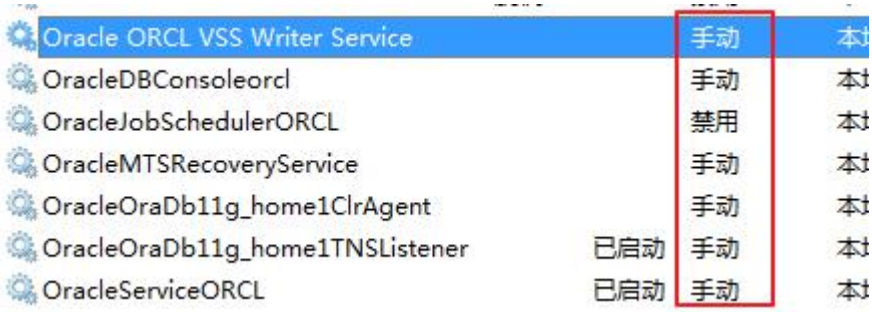
2 Oracle 数据库的安装

详情参考《Orcale 安装指南.docx》

2.1 登录 oracle 数据库

2.1.1 启动 oracle 服务

为了节省资源，建议把其中自动启动的服务修改为手动启动。如下图：



在使用 Oracle 的时候，需要启动 TNSListener 和 OracleService<SID>这两个服务。

- **OracleService<SID>：数据库服务**
- **OracleOraDb11g_hom1TNSListener ： 数据库监听服务**

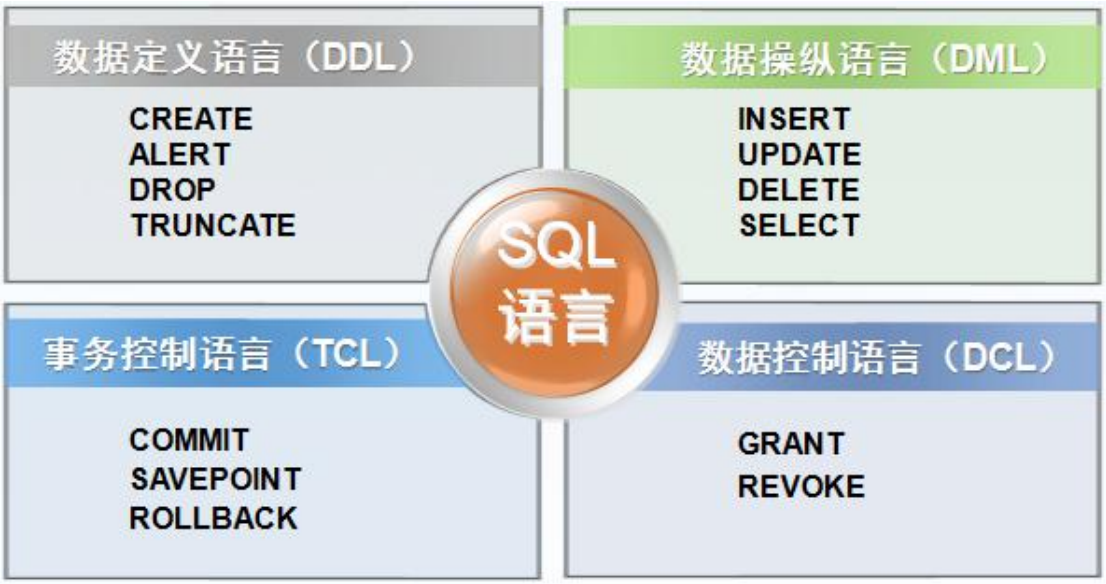
2.1.2 Oracle 默认用户

sys，system，scott 用户都是 Oracle 的系统默认用户，sys，与 system 使用的是 System 表空间，scott 使用的是 users 表空间。

- sys 用户：是 Oracle 的一个超级管理员，主要用来维护系统信息和管理实例，只能以 SYSDBA 或者 SYSOPER 角色登录。
- system 用户：Oracle 默认的系统管理员，拥有 DBA 权限，通常用来管理 Oracle 数据库的用户，权限和存储等，只能以 Normal 方式登录。
- scott 用户：oracle 中普通用户，里面提供了一些测试和学习用的数据。只能以 Normal 方式登录

3.SQL 语句之 DDL

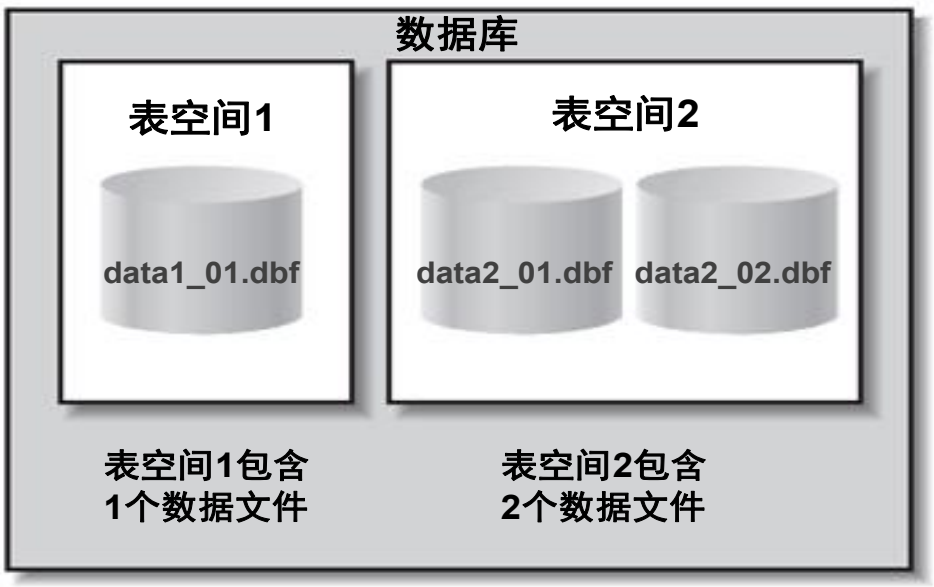
SQL：结构化查询语言，是用于访问和处理数据库的标准的计算机语言。SQL 语言可以分为：



DDL 语言：就是对数据库的对象的操作，比如说创建表，创建用户，创建表空间等...

3.1 表空间的管理

在 oracle 中，从逻辑上划分，是由表空间组成的，所有的表都保存在表空间中。一个 ORACLE 数据库能够有一个或多个表空间,而一个表空间则对应着一个或多个物理的数据库文件。表空间是 ORACLE 数据库恢复的最小单位,容纳着许多数据库实体,如表、视图、索引、聚簇、回退段和临时段等。



数据存储在硬盘上，Oracle数据库在使用时将数据放入表空间中，在数据库中对数据进行操作时，就在表空间内进行。我们可以将表空间删除但是不删除数据，但是此时数据处于什么状态我就不清楚了，应该可以再create表空间将相应的数据放进去。

oracle 中除了系统表空间外，还有临时表空间（temp）。临时表空间用来管理数据库排序操作以及用于存储临时表、中间排序结果等临时对象

oracle 中在安装实例时会默认创建若干个表空间，在创建新的用户时，该用户默认保存在一个叫做 users 的表空间中；此时使用该用户登录后，创建的表也都保存在这个 users 表空间下.

1.创建表空间：

根据业务需要，我们也可以创建自己的表空间，创建语法如下：

```
CREATE TABLESPACE mytest
DATAFILE 'D:\OracleDatabase\mytest.dbf'      --数据文件存放的路径
SIZE 20m                                     --数据文件的初始大小20M  --不自动扩展
AUTOEXTEND OFF;
```

其中，mytest 表示表空间的名字，是自定义的

datafile 表示创建该表空间的物理文件保存的位置和名字

size 表示创建表空间时分配的空间大小

autoextend：表示是否自动扩充　on 表示自动　off 表示关闭

注意：

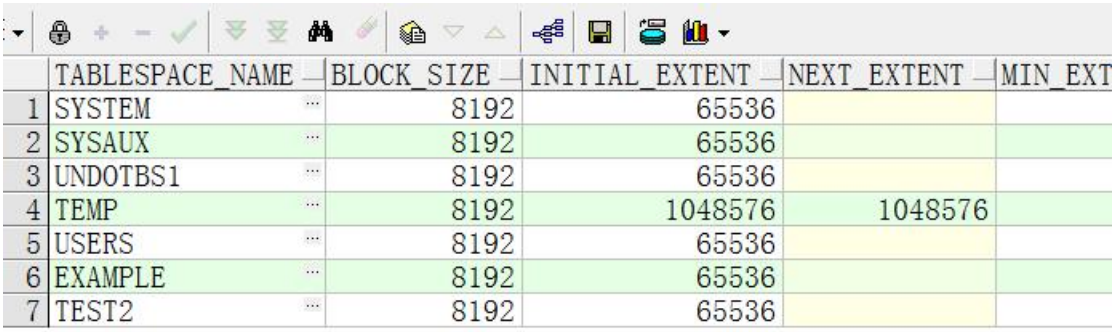
1.oracle 里面，是不认识双引号的，只能使用单引号

2.创建表空间需要管理员权限，否则是无法创建成功的

2.查询所有的表空间：

```
SELECT * FROM dba_tablespaces;
```

查询结果为：



	TABLESPACE_NAME	BLOCK_SIZE	INITIAL_EXTENT	NEXT_EXTENT	MIN_EXT
1	SYSTEM	8192	65536		
2	SYSAUX	8192	65536		
3	UNDOTBS1	8192	65536		
4	TEMP	8192	1048576	1048576	
5	USERS	8192	65536		
6	EXAMPLE	8192	65536		
7	TEST2	8192	65536		

3.删除表空间：

语法：

```
DROP TABLESPACE 表空间名 INCLUDING CONTENTS AND DATAFILES;
```

比如删除 mytest 表空间：

```
DROP TABLESPACE mytest INCLUDING CONTENTS AND DATAFILES;
```

注意：如果没有加“INCLUDING CONTENTS AND DATAFILES”这句代码，那么只是把表空间从 oracle 中删除，但是表空间对应的数据文件是没有删除的。

3.2 用户的管理

数据存储在硬盘中，使用Oracle数据库对数据进行管理时会将数据放入表空间中，平常使用Oracle管理数据也是在表空间中进行，如果仅仅删除表空间，那么数据依然存在，应该可以创建新的表空间将这部分数据放进去。

Oracle 在安装时有一些默认用户供我们使用，比如之前我们看到过的 sys，system，scott 等，但是我们也可以根据需求，自行创建用户。

1. 创建用户：

```
CREATE USER  用户名 IDENTIFIED BY 密码
DEFAULT TABLESPACE 默认表空间(永久保存数据)
TEMPORARY TABLESPACE temp(临时存储数据);
```

比如我们创建一个用户：用户名为 zhangsan,密码为 123456，默认表空间为我们刚刚创建的 mytest 表空间，临时表空间为系统定义的 temp：

```
CREATE USER  zhangsan IDENTIFIED BY 123456
```

```
DEFAULT TABLESPACE mytest

TEMPORARY TABLESPACE temp ;
```

2. 修改用户密码

对于已经存在的用户，我们可以去修改其密码：

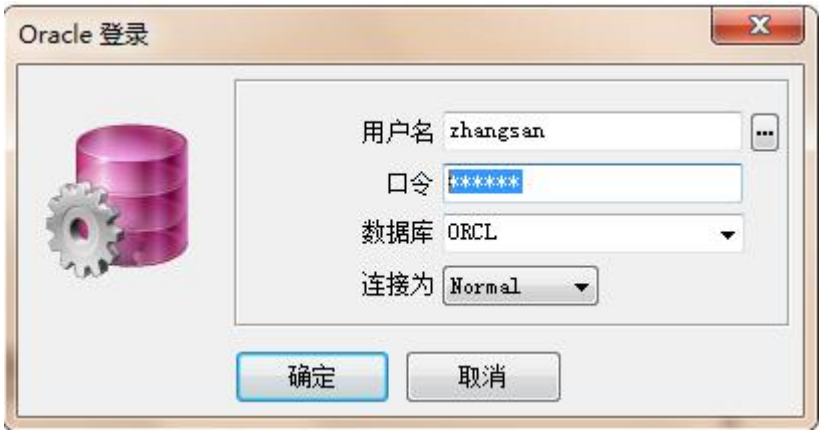
```
ALTER USER 用户名 IDENTIFIED BY 新密码;
```

比如，把刚刚创建的 zhangsan 用户的密码修改为 666666

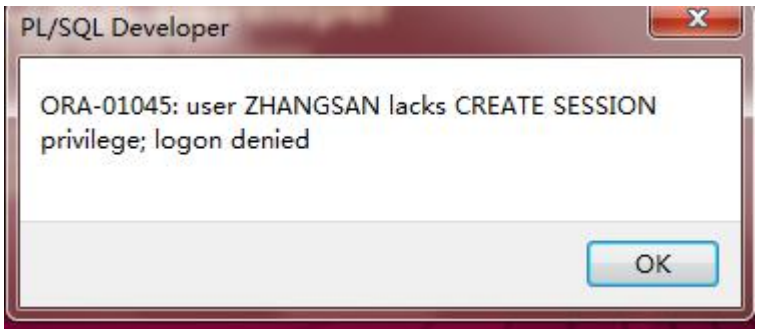
```
ALTER USER zhangsan IDENTIFIED BY 666666;
```

3. 授予权限

我们已经创建了一个用户，那我们就可以使用该用户对数据库进行操作，比如说登录，那我们来进行登录，



但是，很不幸，会报如下错误：



这是因为，oracle 数据库为了安全性，对所有的用户进行的权限控制，新创

建的用户是没有任何权限的，就连最基本的登录权限都没有，所以，我们要对新创建的用户进行授予权限，授予权限的用户一定是级别高的用户才能给级别低的用户授予权限，比如我们现在使用 sys 用户对 zhangsan 用户授予权限。

授予权限的语法：

```
GRANT 权限名[,权限名 2...] TO 用户名
```

比如我们给 zhangsan 用户授予连接的权限

```
GRANT CONNECT TO zhangsan;
```

此时 zhangsan 用户就可以登录到数据库了。

我们常用的权限有如下：

- connect：允许用户创建会话权限
- resource：允许用户对表进行增，删，改，查的权限
- dba：管理员权限

也可以把某个用户所创建的对象权限授予给其他用户比如：

```
--对象权限
--授予查询scott用户的emp表的权限  insert ，  delete  ，
update ,select ，  all
grant all on scott.emp  to zhangsan;
--收回权限 revoke  to:  到哪里去   from:从哪里来
revoke select on scott.emp  from zhangsan;
```

4. 撤销权限

如果我们发现某一用户权限太大，很危险，那我们也可以撤销它的权限。撤销权限

的语法：

```
REVOKE 权限名 [,权限名 2...] FROM 用户名
```

比如我们把 zhangsan 用户的 connect 权限撤销：

```
REVOKE CONNECT  FROM zhangsan
```

那现在 zhangsan 又不能登录了。

5. 删除用户

删除用户的语法：

```
DROP USER 用户名
```

比如我们把 zhangsan 用户给删除：

```
DROP USER zhangsan
```

注意：1,不能对已经连接的用户进行删除。

2,上述的 sql 只能对该用户没有创建任何对象（比如：表，视图等），如果用户创建的其他对象，则需要在后面加 cascade 关键字，例如：

```
DROP USER zhangsan CASCADE;
```

3.3 表的操作

表：基本的数据存储集合，由行列组成。oracle 数据库中，我们的业务数据都是保存在数据库表中，所以为了保存数据，我们需要先创建表.

3.3.1 表的数据类型：

注意：

- 不建议使用：VARCHAR、 INTEGER、FLOAT、DOUBLE 等类型
- char 与 varchar2 的区别，char 表示定长，而 varchar2 表示可变长度，比如我们有一个数据类型为 char（10），那我们存储的的数据为“abc”，因为数据只有三个，而我们用 char（10）来存储，那存在数据库，必须是保证长度为 10，所以也就会在 abc 后面再添加 7 个空格，但是如果我们使用 varchar2（10）来存储，因为 vatchar2 是可变长度，如果存储的是“abc”，那么在数据库中就是三位，不会添加空格。
- number（p，s） 其中 p 表示有效数字长度，包括整数和小数的个数，s 表示小数位数，所以整数的位数为：p-s， 比如说我们定义了一个数据类型为 number（4,2），那么表示小数位必须是 2 位，有效数字位为 4 位，整数位最多为：4-2=2 位，也就是说 number（4,2）最大存储的数字是：99.99

- date 表示日期，如果想获取当前系统日志，可以使用 sysdate 关键字

3.3.2 创建表

创建表的语法：

```
CREATE TABLE 表名(
    列名 1   数据类型 1,
    列名 2   数据类型 2,
    ...
)
```

比如创建一个学生表，包含的列为：学号，姓名，性别，生日，身份证号码：

```
CREATE TABLE t_student(
        stuNo NUMBER(7),
        sName VARCHAR2(50),
        sex CHAR(2),
        birthday DATE,
        cordId CHAR(18)
);
```

3.3.3 修改表结构

针对表本身进行的操作要在表名前加‘table’，且都要使用‘alter table’

1.增加新的一列

语法： alter table 表名

add 新的字段名 字段的类型;

例如： alter table tb_student

add email varchar2(50);

2.修改已存在的列

语法：alter table 表名

modify 字段的名 新的字段类型；

例如： alter table tb_student

modify address varchar2(200);

3.删除一列

语法：alter table 表名

drop column 字段名;

例如：alter table tb_student

drop column iphone;

4.修改字段名

语法： alter table 表名

rename column 字段名 to 新的字段名；

例如： alter table tb_student

rename column iphone to tel;

5.修改表名

语法： rename 表名 to 表名

例如： rename tb_student to student;

3.3.4 删除表

语法： drop table 表名　　可以还原

例如：drop table tb_student;

--还原以删除的表

Flashback table tb_class to before drop;

--彻底删除，不能回滚

drop table 表名 purge

只删除表的数据，不会删除表的结构

语法：truncate table 表名；

例如：　truncate table tb_student;

注意：truncate 不能删除某一行，如果要删除某一行需要使用 delete

Drop:

Truncate:

Delete:　　where

4.处理数据：　DML

针对表中的数据进行操作不用在表名前加‘table’

数据操作语言（DML）可以对表进行如下操作：

- 向表中插入数据
- 修改已存在的数据
- 删除表中的数据

4.1 插入数据

使用 insert 关键字来进行向表中插入数据，insert 的语法如下：

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

使用上述的语法，一次只能向表中插入一条数据。

insert all
into tbname values( )
.
.
select * from dual;

比如向之前创建的 t_student 表添加数据：

```
INSERT INTO t_student   VALUES(1000001,'张三','男','01-11 月
-1999','430112450987342345');
```

注意：

1. 如果表名后面没有写列名，则表示对表的所有的列添加数据，赋值的顺序与创建表的列的顺序要一致。
2. 字符和日期型数据赢包含在单引号中，数字类型可以加单引号或者不加
3. Oracle 默认的日期格式为“DD-MMM-YYYY”，如果想自定义日期格式，可以使用to_date()函数。

to_date可以保证时间按照数据库默认时间格式插入，如果插入时的格式跟数据库默认格式一致也可以直接插入。

```
INSERT  INTO  t_student    VALUES(1000002,'李四','男
','to_date('1998-10-11','yyyy-mm-dd'),'430112450987349
087');
```

后面的‘yyyy-mm-dd’是用来对应前面的日期格式进行转换的，转换过后转换过后按照数据库默认格式进行填充，前面是什么格式，后面就是什么格式，这个格式跟数据库本身的时间格式无关。

也可以给表的部分列添加数据，比如给学生表的学号和姓名添加数据，其他列不赋值，

此时其他列的值为 null

```
INSERT INTO t_student(stuNo,sname)   VALUES(1000003,'王五');
```

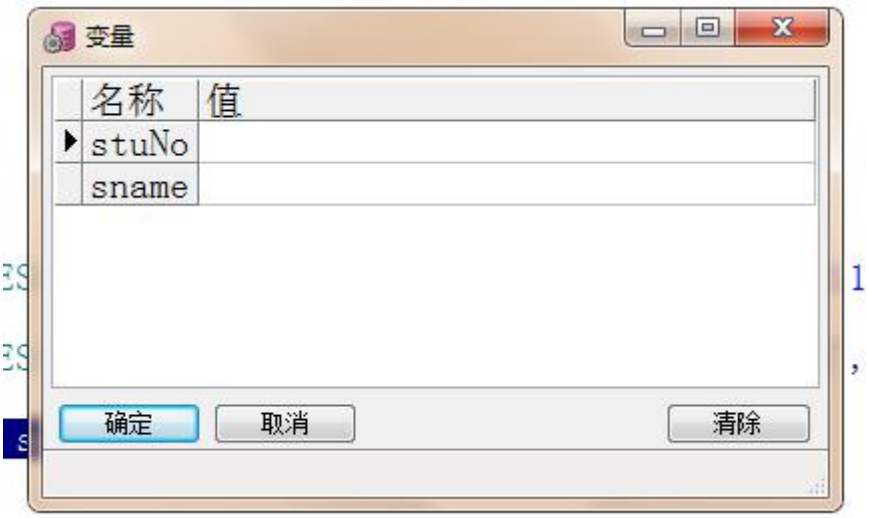
注意：如果某列设置为非空，那么向表中插入数据时，一定要给非空列赋值。

也可以在 sql 语句中使用&变量指定列的值， &变量名是放在 values 中，这样可以弹

出输入框，让我们进行手动输入值了，如下：

```
INSERT INTO t_student(stuNo,sname)  VALUES(&stuNo,&sname);
```

运行效果：



4.2 更新数据

我们也可以对表中已存在的数据进行修改，修改数据使用 update 关键字，语法如下：

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

比如我们要把 t_student 表中的数据的 sex 修改为 ‘女’

```
UPDATE t_student SET sex =’女’
```

注意：上面的 sql 语句会把表中的所有数据的 sex 都修改成女，这是很危险的，请慎用，在实际开发中，我们会在 update 语句后面添加 where 子句，where 子句就是添加条件的，如果我们要给学号是 1000001 的学生的 sex 修改为男：

```
UPDATE t_student SET sex =’男’ WHERE stuNo = 1000001
```

4.3 删除数据

我们也可以对已存在的数据进行删除，但是我们只能一行一行的删除，使用 delete 关键字就可以了，语法如下：

```
DELETE [FROM]   table
[WHERE          condition];
```

删除某张表中所有的数据，就不需要加 where 子句，比如把 t_student 表中的数据全部删除：

```
DELETE FROM t_student
```

但是大家要慎用，如果想删除指定条件的数据，那就可以再 delete 语句后面加 where，比如我们要删除性别为女的信息：

```
DELETE FROM t_student  WHERE sex = '女'
```

如果要删除表中的所有的数据，也可以使用 truncate

```
TRUNCATE TABLE t_student
```

delete 与 truncate 的相同与不同：

相同点：都是删除表中的数据，不能删除表的结果 **结构**

不同点：1. delete 操作可以回滚，而 truncate 是不能回滚的

2.delete 操作可能产生碎片，并且不释放空间，而 truncate 会释放表空间

5.约束

约束是表一级的限制，是指在表上强制执行的数据校验规则。确保数据的准确性和可靠

性

大部分数据库支持下面五类完整性约束：

NOT NULL　--非空

UNIQUE KEY　--唯一键

PRIMARY KEY --主键

FOREIGN KEY --外键

CHECK　　--检查

创建约束的时机：

-在建表的同时创建

-建表之后创建（修改表）

1.非空约束（NOT NULL）　确保字段值不允许为空，只能在字段之后定义。

NULL 值，所有数据类型的值都可以是 NULL

空字符串不等于 NULL "" != NULL

0 也不等于 NULL　0 != NULL

例如：　name varchar2(10) not null, --非空约束

2.主键约束：　从功能上看相当于非空并且唯一。

一个表中只能允许一个主键（并且一定要有一个主键）

主键是表中唯一确定一行数据的字段

主键字段可以是单个字段或者是多个字段的组合。

主键约束建议与业务无关

例如：id number primary key, --主键约束

3.唯一约束：确保所在的字段或字段组合不出现重复值，

同一张表可建多个唯一约束，

唯一约束可以由多个字段组成。

唯一约束条件的字段允许出现多个 NULL. 因为null并不代表空值，只是代表不知道

例如： email varchar2(50) unique, -- 唯一约束

4.Check 约束：定义在字段上的每一个记录都要满足的条件

即可写在字段后面，也可写在字段都定义之后。

age number not null check(age > 18 and age < 60), --check 约束

5.外键约束

外键是构建于一个表的两个字段或者两个表的两个字段之间的关系。

子（从）表外键列的值必须在主表参照列值的范围内。

外键只能参照主键或者唯一键

当主表的记录被子表参照时，主表记录不允许被删除。

例如： clazz_id number not null references tb_clazz(id) --外键约束

还能这样写？

6.默认值： 跟某字段添加默认值（如果没输入，则就是默认值，如果输入，则是输入的值）

例如： address varchar2(200) default '地址不详',

第一种，在创建表的时候添加约束：

```
CREATE TABLE tb_student(
    stuId NUMBER,
```

```
    sname VARCHAR2(10) NOT NULL, --非空约束
    sex VARCHAR2(6)    NOT NULL,
    age NUMBER NOT NULL,
    email VARCHAR2(50) ,
    address VARCHAR(200),
    tell VARCHAR2(11),
    clazz_id NUMBER NOT  NULL ,
     CONSTRAINTS  tb_student_pk PRIMARY  KEY(stuId),
     CONSTRAINTS  tb_student_uk UNIQUE(email),
     CONSTRAINTS  tb_student_ck_sex CHECK(sex='男' OR sex='女'),
     CONSTRAINTS  tb_student_ck_age CHECK(age>18 AND age<60),
     CONSTRAINTS  tb_student_fk_class_cid FOREIGN KEY(clazz_id)
REFERENCES tb_clazz(cid)
);
```

第二种，如果表已经创建,可以以如下方式进行添加：

添加约束：

语法：Alter table 表名

add constraints 约束名 约束类型

例如：alter table tb_clazz

add constraints tb_clazz_pk primary key(id);

删除约束：

语法：alter table 表名

drop constraints 约束名；

例如：alter table tb_clazz

drop constraints tb_clazz_pk;

----当表创建完了之后 给列添加非空约束;

alter table 表名 modify(列名 not null);

------修改约束名称

alter table 表名 rename constraint old_name to new_name;

6.查询语句

DQL 就是数据查询语言，数据库执行 DQL 语句不会对数据进行改变，而是让数据库发送结果集给客户端。

语法：

```
SELECT selection_list /*要查询的列名称*/
  FROM table_list /*要查询的表名称*/
  WHERE condition /*行条件*/
  GROUP BY grouping_columns /*对结果分组*/
  HAVING condition /*分组后的行条件*/
  ORDER BY sorting_columns /*对结果分组*/
```

where和having的区别是：where是针对分组之前的表中数据进行过滤，而having则是针对分组后的数据进行过滤

创建名：

- 学生表：stu

字段名称	字段类型	说明
sid	char(6)	学生学号
sname	varchar(50)	学生姓名
age	int	学生年龄
gender	varchar(50)	学生性别

```
CREATE TABLE stu (
    sid    CHAR(6),
    sname    VARCHAR2(50),
    age   NUMBER,
    gender     VARCHAR2(50)
);
```

```
INSERT INTO stu VALUES('S_1001', 'liuYi', 35, 'male');
INSERT INTO stu VALUES('S_1002', 'chenEr', 15, 'female');
INSERT INTO stu VALUES('S_1003', 'zhangSan', 95, 'male');
INSERT INTO stu VALUES('S_1004', 'liSi', 65, 'female');
INSERT INTO stu VALUES('S_1005', 'wangWu', 55, 'male');
INSERT INTO stu VALUES('S_1006', 'zhaoLiu', 75, 'female');
INSERT INTO stu VALUES('S_1007', 'sunQi', 25, 'male');
INSERT INTO stu VALUES('S_1008', 'zhouBa', 45, 'female');
INSERT INTO stu VALUES('S_1009', 'wuJiu', 85, 'male');
INSERT INTO stu VALUES('S_1010', 'zhengShi', 5, 'female');
INSERT INTO stu VALUES('S_1011', 'xxx', NULL, NULL);
```

6.1 基础查询

6.1.1 查询所有列

```
SELECT * FROM stu;
```

6.1.2　查询指定列

```
SELECT sid, sname, age FROM stu;
```

6.1.3 条件查询

6.1.3.1　条件查询介绍

条件查询就是在查询时给出 WHERE 子句，在 WHERE 子句中可以使用如下运算符及关键字：

- =、!=、<>、<、<=、>、>=；
- BETWEEN…AND；
- IN(set)；
- IS NULL；
- AND；
- OR；
- NOT；

6.1.3.2　查询性别为女，并且年龄 50 的记录

```
SELECT * FROM stu
WHERE gender='female' AND ge<50;
```

6.1.3.3　查询学号为 S_1001，或者姓名为 liSi 的记录

```
SELECT * FROM stu
WHERE sid ='S_1001' OR sname='liSi';
```

6.1.3.4　查询学号为 S_1001，S_1002，S_1003 的记录

```
SELECT * FROM stu
WHERE sid IN ('S_1001','S_1002','S_1003');
```

6.1.3.5　查询学号不是 S_1001，S_1002，S_1003 的记录

```
SELECT * FROM tab_student
WHERE s_number NOT IN ('S_1001','S_1002','S_1003');
```

6.1.3.6　查询年龄为 null 的记录

如何将表数据更新为null？

```
update stu set age=null where sid='s001';
```

```
SELECT * FROM stu
WHERE age IS NULL;
```

6.1.3.7　查询年龄在 20 到 40 之间的学生记录

```
SELECT *
FROM stu
WHERE age>=20 AND age<=40;
```

或者

```
SELECT *
FROM stu
WHERE age BETWEEN 20 AND 40;
```

6.1.3.8　查询性别非男的学生记录

```
SELECT *
```

```
FROM stu
WHERE gender!='male';
```

或者

```
SELECT *
FROM stu
WHERE gender<>'male';
```

或者

```
SELECT *
FROM stu
WHERE NOT gender='male';
```

6.1.3.9 查询姓名不为 null 的学生记录

```
SELECT *
FROM stu
WHERE NOT sname IS NULL;
```

或者

```
SELECT *
FROM stu
WHERE sname IS NOT NULL;
```

6.1.4 模糊查询

当想查询姓名中包含 a 字母的学生时就需要使用模糊查询了。模糊查询需要使用关键字 LIKE。

6.1.4.1 查询姓名由 5 个字母构成的学生记录

```
SELECT *
```

```
FROM stu
WHERE sname LIKE '_____';
```

模糊查询必须使用 LIKE 关键字。其中 “_”匹配任意一个字母，5 个“_”表示 5 个任意字母。

6.1.4.2 查询姓名由 5 个字母构成，并且第 5 个字母为“i”的学生记录

```
SELECT *
FROM stu
WHERE sname LIKE '____i';
```

6.1.4.3 查询姓名以“z”开头的学生记录

```
SELECT *
FROM stu
WHERE sname LIKE 'z%';
```

其中“%”匹配 0~n 个任何字母。

6.1.4.4 查询姓名中第 2 个字母为“i”的学生记录

```
SELECT *
FROM stu
WHERE sname LIKE '_i%';
```

6.1.4.5　查询姓名中包含“a”字母的学生记录

```
SELECT *
FROM stu
WHERE sname LIKE '%a%';
```

6.1.5 正则表达式 regexp_like 查询

6.1.5.1 查询姓名以 J 开头，以 S 结尾，中间包含 3 位任意字符

```
SELECT *
FROM emp
WHERE regexp_like(ename ,'J...S');
或者
SELECT *
FROM emp
WHERE regexp_like(ename ,'J\w{3}S');
```

6.1.5.2 查询姓名中以 A 开头的，A 后面有 2 到 3 个字符

```
SELECT *
FROM emp
WHERE regexp_like(ename ,'^A.{2,3}$');
```

6.1.5.3 忽略大小写并且名字中以 a 开始，至少有 2 个字符的名字

```
SELECT *
FROM emp
WHERE regexp_like(ename ,'^a.{2}','i');
```

6.1.5.4 常用的正则表达式符号

^ 匹配一个字符串的开始

$ 匹配字符串的结尾

* 表示匹配零个或多个字符

+ 号表示匹配一个或多个

？ 表示匹配 0 次或 1 次

. 匹配任何字符，除了空 **.在没有匹配次数的前提下匹配任何一个字符,我猜的哈哈**

\. 匹配小数点

| 用‘OR’来指定多个选项

[ ] 匹配指定列表，比如[0-9]匹配 0 到 9 任意一个数据，[a-z] 匹配小写字母

[^] 匹配不在指定列表｛m｝匹配 m 次

{m,} 匹配至少 m 次

{m,n}匹配至少 m 次，不超过 n 次

{m} 匹配刚好 m 次

\d 表示一个数字

\D 表示一个非数字

\w 匹配包括下划线的任何单词字符

6.1.6 字段控制查询

6.1.6.1 去除重复记录

去除重复记录（两行或两行以上记录中系列的上的数据都相同），例如 emp 表中 sal 字段就存在相同的记录。当只查询 emp 表的 sal 字段时，那么会出现重复记录，那么想去除重复记录，需要使用 DISTINCT：

```
SELECT DISTINCT sal FROM emp;
```

6.1.6.2 查看雇员的月薪与奖金之和

因为 sal 和 comm 两列的类型都是数值类型，所以可以做加运算。如果 sal 或 comm 中有一个字段不是数值类型，那么会出错。

```
SELECT emp. *,sal+comm FROM emp;
```

comm 列有很多记录的值为 NULL，因为任何东西与 NULL 相加结果还是 NULL，所以结算结果可能会出现 NULL。下面使用了把 NULL 转换成数值 0 的函数 NVL：

```
SELECT  emp.*,sal+NVL(comm,0) FROM emp;
```

6.1.6.3 连接

我们在编写 sql 语句的时候,有时需要把多列进行拼接,或者给某个拼接其他字符串,那么就需要使用 oracle 提供的连接符操作,oracle 提供的连接操作有如下两种:

- 使用 || 连接符号,类似与 java 中的 +,例如

```
SELECT  ename || '的工资:'||sal  FROM  emp;
```

- 使用 concat 函数 concat(内容 1,内容 2) 例如

```
SELECT  concat('hello','word')  FROM  dual
```

6.1.6.4 给列名添加别名

在上面查询中出现列名为 sal+NVL(comm,0)，这很不美观，现在我们给这一列给出一个别名，为 total：

```
SELECT  emp.*, sal+NVL(comm,0) AS total FROM emp;
```

给列起别名时，是可以省略 AS 关键字的：

```
SELECT emp.*,sal+NVL(comm,0) total FROM emp;
```

除了给列取别名，也可以给表取别名：

```
SELECT e.*,sal+NVL(comm,0) total FROM emp e;
```

6.1.7 排序

6.1.7.1 查询所有学生记录，按年龄升序排序

```
SELECT *
FROM stu
ORDER BY sage ASC;
```

默认升序

或者

```
SELECT *
FROM stu
ORDER BY sage;
```

6.1.7.2 查询所有学生记录，按年龄降序排序

```
SELECT *
FROM stu
ORDER BY age DESC;
```

6.1.7.3 查询所有雇员，按月薪降序排序，如果月薪相同时，按编号升序排序

```
SELECT * FROM emp
ORDER BY sal DESC,empno ASC;
```

6.1.8 聚合函数

聚合函数是用来做纵向运算的函数：

- COUNT()：统计指定列不为 NULL 的记录行数；

如果统计整个表的行数，由于rownum函数的存在，会在结果集中给每一行分配一个伪行数，因此count函数此时也会统计null行，效果与赋值的count(1)相同

- MAX()：计算指定列的最大值，如果指定列是字符串类型，那么使用字符串排序运算；
- MIN()：计算指定列的最小值，如果指定列是字符串类型，那么使用字符串排序运算；
- SUM()：计算指定列的数值和，如果指定列类型不是数值类型，那么计算结果为 0；
- AVG()：计算指定列的平均值，如果指定列类型不是数值类型，那么计算结果为 0；

6.1.8.1 COUNT

当需要纵向统计时可以使用 COUNT()。

- 查询 emp 表中记录数：**空的记录能查到吗？可以**

```
SELECT COUNT(*) AS cnt FROM emp;
```

- 查询 emp 表中有奖金的人数：

```
SELECT COUNT(comm) cnt FROM emp; as cnt
```

注意，因为 count()函数中给出的是 comm 列，那么只统计 comm 列非 NULL 的行数。

- 查询 emp 表中月薪大于 2500 的人数：

```
SELECT COUNT(*) FROM emp
WHERE sal > 2500;
```

- 统计月薪与佣金之和大于 2500 元的人数：

```
SELECT COUNT(*) AS cnt FROM emp WHERE sal+NVL(comm,0) > 2500;
```

6.1.8.2 SUM 和 AVG

当需要纵向求和时使用 sum()函数。

- 查询所有雇员月薪和：

```
SELECT SUM(sal) FROM emp;
```

- 查询所有雇员月薪和，以及所有雇员奖金和：

```
SELECT SUM(sal), SUM(comm) FROM emp;
```

- 查询所有雇员月薪+奖金和：

```
SELECT SUM(sal+NVL(comm,0)) FROM emp;
```

- 统计所有员工平均工资：

```
SELECT SUM(sal)/ COUNT(sal) FROM emp;
```

或者

```
SELECT AVG(sal) FROM emp;
```

6.1.8.3 MAX 和 MIN

- 查询最高工资和最低工资：

```
SELECT MAX(sal), MIN(sal) FROM emp;
```

6.1.9 分组查询

当需要分组查询时需要使用GROUP BY子句，例如查询每个部门的工资和，这说明要使用部分来分组。

部门

6.1.9.1 分组查询

- 查询每个部门的部门编号和每个部门的工资和：

```
SELECT deptno, SUM(sal)
```

```
FROM emp
GROUP BY deptno;
```

- 查询每个部门的部门编号以及每个部门的人数：

```
SELECT deptno,COUNT(*)
FROM emp
GROUP BY deptno;
```

因为已经对表中数据进行分组，因此无法查询到某个字段的结果，因为此时该字段已经包含该组的数据，但分组的依据例如deptno由于分组之后不影响，所以还可以显示。其余能显示的是分组之后的数据，例如使用聚合函数count，max，avg等对该组数据计算出的结果

- 查询每个部门的部门编号以及每个部门工资大于 1500 的人数：

```
SELECT deptno,COUNT(*)
FROM emp
WHERE sal>1500
GROUP BY deptno;
```

在对分组中的数据进行比较时不能直接使用having，只能使用where先对表中的数据进行处理，然后再根据部门编号进行分类。而对分组组间数据直接进行统计时使用having。

6.1.9.2 HAVING 子句

- 查询工资总和大于 9000 的部门编号以及工资和：

```
SELECT deptno, SUM(sal)
FROM emp
GROUP BY deptno
HAVING SUM(sal) > 9000;
```

注意，WHERE 是对分组前记录的条件，如果某行记录没有满足 WHERE 子句的条件，那么这行记录不会参加分组；而 HAVING 是对分组后数据的约束。

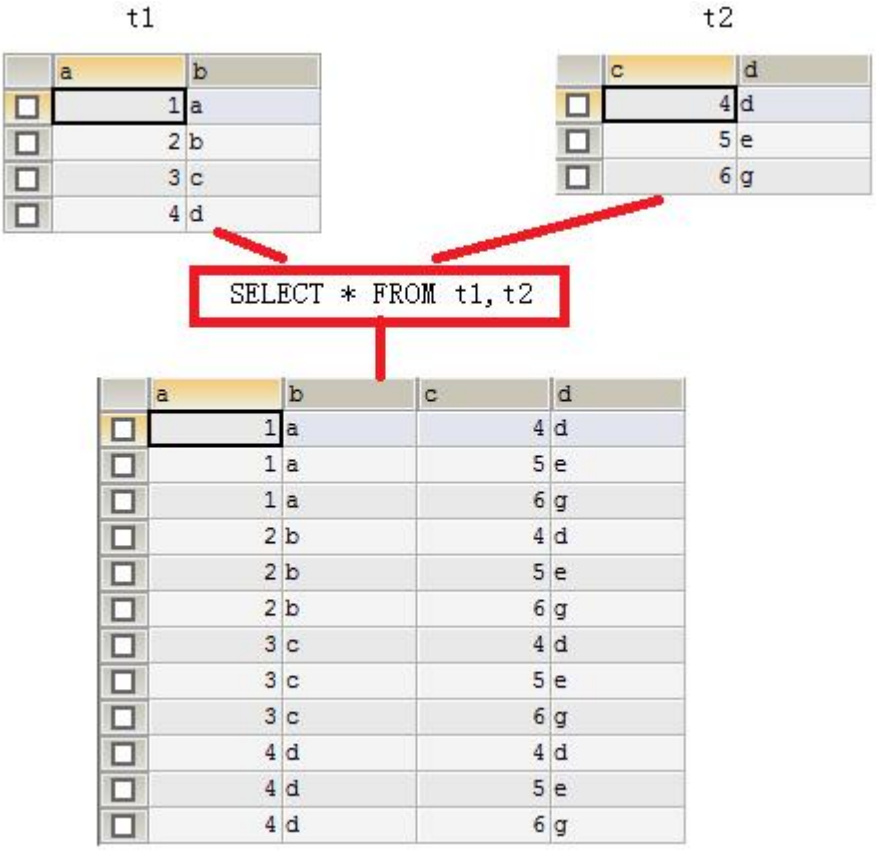
6.2 多表联查

多表查询有如下几种：

- 连接查询
  - 内连接
  - 外连接
    - 左外连接
    - 右外连接
  - 自然连接
- 子查询
- 合并结果集；

6.2.1 连接查询

连接查询就是求出多个表的乘积，例如 t1 连接 t2，那么查询出的结果就是 t1*t2。



连接查询会产生笛卡尔积，假设集合 A={a,b}，集合 B={0,1,2}，则两个集合的笛卡尔积为{(a,0),(a,1),(a,2),(b,0),(b,1),(b,2)}。可以扩展到多个集合的情况。

那么多表查询产生这样的结果并不是我们想要的，那么怎么去除重复的，不想要的记录

呢，当然是通过条件过滤。通常要查询的多个表之间都存在关联关系，那么就通过关联关系去除笛卡尔积。

你能想像到 emp 和 dept 表连接查询的结果么？emp 一共 14 行记录，dept 表一共 4 行记录，那么连接后查询出的结果是 56 行记录。

也就你只是想在查询 emp 表的同时，把每个员工的所在部门信息显示出来，那么就需要使用主外键来去除无用信息了。

使用主外键关系做为条件来去除无用信息

```
SELECT * FROM emp,dept WHERE emp.deptno=dept.deptno;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno	deptno	dname	loc
7782	CLARK	MANAGER	7839	1981-06-09	2450.00	*(NULL)*	10	10	ACCOUNTING	NEW YORK
7839	KING	PRESIDENT	*(NULL)*	1981-11-17	5000.00	*(NULL)*	10	10	ACCOUNTING	NEW YORK
7934	MILLER	CLERK	7782	1982-01-23	1300.00	*(NULL)*	10	10	ACCOUNTING	NEW YORK
7369	SMITH	CLERK	7902	1980-12-17	800.00	*(NULL)*	20	20	RESEARCH	DALLAS
7566	JONES	MANAGER	7839	1981-04-02	2975.00	*(NULL)*	20	20	RESEARCH	DALLAS
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	*(NULL)*	20	20	RESEARCH	DALLAS
7876	ADAMS	CLERK	7788	1987-05-23	1100.00	*(NULL)*	20	20	RESEARCH	DALLAS
7902	FORD	ANALYST	7566	1981-12-03	3000.00	*(NULL)*	20	20	RESEARCH	DALLAS
7499	ALLEN	SALESMAN	7698	1981-02-20	1600.00	300.00	30	30	SALES	CHICAGO
7521	WARD	SALESMAN	7698	1981-02-22	1250.00	500.00	30	30	SALES	CHICAGO
7654	MARTIN	SALESMAN	7698	1981-09-28	1250.00	1400.00	30	30	SALES	CHICAGO
7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	*(NULL)*	30	30	SALES	CHICAGO
7844	TURNER	SALESMAN	7698	1981-09-08	1500.00	0.00	30	30	SALES	CHICAGO
7900	JAMES	CLERK	7698	1981-12-03	950.00	*(NULL)*	30	30	SALES	CHICAGO

上面查询结果会把两张表的所有列都查询出来，也许你不需要那么多列，这时就可以指定要查询的列了。

```
SELECT emp.ename,emp.sal,emp.comm,dept.dname
FROM emp,dept
WHERE emp.deptno=dept.deptno;
```

ename	sal	comm	dname
CLARK	2450.00	(NULL)	ACCOUNTING
KING	5000.00	(NULL)	ACCOUNTING
MILLER	1300.00	(NULL)	ACCOUNTING
SMITH	800.00	(NULL)	RESEARCH
JONES	2975.00	(NULL)	RESEARCH
SCOTT	3000.00	(NULL)	RESEARCH
ADAMS	1100.00	(NULL)	RESEARCH
FORD	3000.00	(NULL)	RESEARCH
ALLEN	1600.00	300.00	SALES
WARD	1250.00	500.00	SALES
MARTIN	1250.00	1400.00	SALES
BLAKE	2850.00	(NULL)	SALES
TURNER	1500.00	0.00	SALES
JAMES	950.00	(NULL)	SALES

还可以为表指定别名，然后在引用列时使用别名即可。

```
SELECT e.ename,e.sal,e.comm,d.dname
FROM emp AS e,dept AS d
WHERE e.deptno=d.deptno;
```

6.2.1.1 内连接

上面的连接语句就是内连接，但它不是 SQL 标准中的查询方式，可以理解为方言！SQL 标准的内连接为：

```
SELECT *
FROM emp e
INNER JOIN dept d
ON e.deptno=d.deptno;
```

内连接的特点：查询结果必须满足条件。例如我们向 emp 表中插入一条记录：

8888	张三	CLERK	7782	1983-07-07	1500.00	(NULL)	(NULL)

其中 deptno 为 null，而在 dept 表中只有 10、20、30、40 部门，那么上面的查询结果

中就不会出现“张三”这条记录，因为它不能满足 e.deptno=d.deptno 这个条件。

6.2.1.2 外连接（左连接、右连接）

外连接的特点：查询出的结果存在不满足条件的可能。

左连接：

```
SELECT * FROM emp e
LEFT OUTER JOIN dept d
ON e.deptno=d.deptno;
```

左连接是先查询出左表（即以左表为主），然后查询右表，右表中满足条件的显示出来，不满足条件的显示 NULL。

这么说你可能不太明白，我们还是用上面的例子来说明。其中 emp 表中“张三”这条记录中，部门编号为 null，而 dept 表中不存在部门编号为 null 的记录，所以“张三”这条记录，不能满足 e.deptno=d.deptno 这条件。但在左连接中，因为 emp 表是左表，所以左表中的记录都会查询出来，即“张三”这条记录也会查出，但相应的右表部分显示 NULL。

empno	ename	job	mgr	hiredate	sal	comm	deptno	deptno	dname	loc
7369	SMITH	CLERK	7902	1980-12-17	800.00	(NULL)	20	20	RESEARCH	DALLAS
7499	ALLEN	SALESMAN	7698	1981-02-20	1600.00	300.00	30	30	SALES	CHICAGO
7521	WARD	SALESMAN	7698	1981-02-22	1250.00	500.00	30	30	SALES	CHICAGO
7566	JONES	MANAGER	7839	1981-04-02	2975.00	(NULL)	20	20	RESEARCH	DALLAS
7654	MARTIN	SALESMAN	7698	1981-09-28	1250.00	1400.00	30	30	SALES	CHICAGO
7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	(NULL)	30	30	SALES	CHICAGO
7782	CLARK	MANAGER	7839	1981-06-09	2450.00	(NULL)	10	10	ACCOUNTING	NEW YORK
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	(NULL)	20	20	RESEARCH	DALLAS
7839	KING	PRESIDENT	(NULL)	1981-11-17	5000.00	(NULL)	10	10	ACCOUNTING	NEW YORK
7844	TURNER	SALESMAN	7698	1981-09-08	1500.00	0.00	30	30	SALES	CHICAGO
7876	ADAMS	CLERK	7788	1987-05-23	1100.00	(NULL)	20	20	RESEARCH	DALLAS
7900	JAMES	CLERK	7698	1981-12-03	950.00	(NULL)	30	30	SALES	CHICAGO
7902	FORD	ANALYST	7566	1981-12-03	3000.00	(NULL)	20	20	RESEARCH	DALLAS
7934	MILLER	CLERK	7782	1982-01-23	1300.00	(NULL)	10	10	ACCOUNTING	NEW YORK
8888	张三	CLERK	7782	1983-07-07	1500.00	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

6.2.1.3 右连接

右连接就是先把右表中所有记录都查询出来，然后左表满足条件的显示，不满足显示 NULL。例如在 dept 表中的 40 部门并不存在员工，但在右连接中，如果 dept 表为右表，那么还是会查出 40 部门，但相应的员工信息为 NULL。

```
SELECT * FROM emp e
```

```
RIGHT OUTER JOIN dept d
ON e.deptno=d.deptno;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno	deptno	dname	loc
7782	CLARK	MANAGER	7839	1981-06-09	2450.00	(NULL)	10	10	ACCOUNTING	NEW YORK
7839	KING	PRESIDENT	(NULL)	1981-11-17	5000.00	(NULL)	10	10	ACCOUNTING	NEW YORK
7934	MILLER	CLERK	7782	1982-01-23	1300.00	(NULL)	10	10	ACCOUNTING	NEW YORK
7369	SMITH	CLERK	7902	1980-12-17	800.00	(NULL)	20	20	RESEARCH	DALLAS
7566	JONES	MANAGER	7839	1981-04-02	2975.00	(NULL)	20	20	RESEARCH	DALLAS
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	(NULL)	20	20	RESEARCH	DALLAS
7876	ADAMS	CLERK	7788	1987-05-23	1100.00	(NULL)	20	20	RESEARCH	DALLAS
7902	FORD	ANALYST	7566	1981-12-03	3000.00	(NULL)	20	20	RESEARCH	DALLAS
7499	ALLEN	SALESMAN	7698	1981-02-20	1600.00	300.00	30	30	SALES	CHICAGO
7521	WARD	SALESMAN	7698	1981-02-22	1250.00	500.00	30	30	SALES	CHICAGO
7654	MARTIN	SALESMAN	7698	1981-09-28	1250.00	1400.00	30	30	SALES	CHICAGO
7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	(NULL)	30	30	SALES	CHICAGO
7844	TURNER	SALESMAN	7698	1981-09-08	1500.00	0.00	30	30	SALES	CHICAGO
7900	JAMES	CLERK	7698	1981-12-03	950.00	(NULL)	30	30	SALES	CHICAGO
(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	40	OPERATIONS	BOSTON

losal和hisal那张表也可以连接

连接查询心得：

连接不限与两张表，连接查询也可以是三张、四张，甚至 N 张表的连接查询。通常连接查询不可能需要整个笛卡尔积，而只是需要其中一部分，那么这时就需要使用条件来去除不需要的记录。这个条件大多数情况下都是使用主外键关系去除。

两张表的连接查询一定有一个主外键关系，三张表的连接查询就一定有两个主外键关系，所以在大家不是很熟悉连接查询时，首先要学会去除无用笛卡尔积，那么就是用主外键关系作为条件来处理。如果两张表的查询，那么至少有一个主外键条件，三张表连接至少有两个主外键条件。这里说的使用主外键关系进行筛选，不是指表中必须要有主键和外键，而是表和表在连接的时候，连接条件是主外键关系，比如stu.sno=sc.sno，两者的两列相同，相当于一个外键关系。

6.2.1.4 自然连接

大家也都知道，连接查询会产生无用笛卡尔积，我们通常使用主外键关系等式来去除它。而自然连接无需你去给出主外键等式，它会自动找到这一等式：

- 两张连接的表中名称和类型完成一致的列作为条件，例如 emp 和 dept 表都存在 deptno 列，并且类型一致，所以会被自然连接找到！

当然自然连接还有其他的查找条件的方式，但其他方式都可能存在问题！

```
SELECT * FROM emp NATURAL JOIN dept;
SELECT * FROM emp NATURAL LEFT JOIN dept;
```

```
SELECT * FROM emp NATURAL RIGHT JOIN dept;
```

6.2.2 子查询

子查询就是嵌套查询，即 SELECT 中包含 SELECT，如果一条语句中存在两个，或两个以上 SELECT，那么就是子查询语句了。

- 子查询出现的位置：
  - where 后，作为条件的一部分；
  - from 后，作为被查询的一条表；
- 当子查询出现在 where 后作为条件时，还可以使用如下关键字：
  - any
  - all
- 子查询结果集的形式：
  - 单行单列（用于条件）
  - 单行多列（用于条件）
  - 多行单列（用于条件）
  - 多行多列（用于表）

练习：

1. 工资高于 ALLEN 的员工。

分析：

查询条件：工资>ALLEN 工资，其中 ALLEN 工资需要一条子查询。

第一步：查询 ALLEN 的工资

```
SELECT sal FROM emp WHERE ename='ALLEN'
```

第二步：查询高于 ALLEN 工资的员工

```
SELECT * FROM emp WHERE sal > (${第一步})
```

结果：

```
SELECT * FROM emp WHERE sal > (SELECT sal FROM emp WHERE ename='ALLEN')
```

- 子查询作为条件
- 子查询形式为单行单列

2. 工资高于 30 部门所有人的员工信息

分析：

查询条件：工资高于 30 部门所有人工资，其中 30 部门所有人工资是子查询。高于所有需要使用 all 关键字。

第一步：查询 30 部门所有人工资

```
SELECT sal FROM emp WHERE deptno=30;
```

第二步：查询高于 30 部门所有人工资的员工信息

```
SELECT * FROM emp WHERE sal > ALL (${第一步})
```

结果：

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

- 子查询作为条件
- 子查询形式为多行单列（当子查询结果集形式为多行单列时可以使用 ALL 或 ANY 关键字）

3. 查询工作和工资与 MARTIN 完全相同的员工信息

分析：

查询条件：工作和工资与 MARTIN 完全相同，这是子查询

第一步：查询出 MARTIN 的工作和工资

```
SELECT job,sal FROM emp WHERE ename='MARTIN'
```

第二步：查询出与 MARTIN 工作和工资相同的人

```
SELECT * FROM emp WHERE (job,sal) IN (${第一步})
```

结果：

```
SELECT * FROM emp WHERE (job,sal) IN (SELECT job,sal FROM emp WHERE ename='MARTIN')
```

- 子查询作为条件
- 子查询形式为单行多列

4. 查询员工编号为 7369 的员工名称、员工工资、部门名称、部门地址

分析：

查询列：员工名称、员工工资、部门名称、部门地址

查询表：emp 和 dept，分析得出，不需要外连接（外连接的特性：某一行（或某些行）记录上会出现一半有值，一半为 NULL 值）

条件：员工编号为 7369

第一种方式：使用表连接查询

第一步：去除多表，只查一张表，这里去除部门表，只查员工表

```
SELECT ename, sal FROM emp e WHERE empno=7369
```

第二步：让第一步与 dept 做内连接查询，添加主外键条件去除无用笛卡尔积

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno=d.deptno AND empno=7369
```

第二种方式:使用子查询

第二步中的 dept 表表示所有行所有列的一张完整的表，这里可以把 dept 替换成所有行，但只有 dname 和 loc 列的表，这需要子查询。

第一步：查询 dept 表中 dname 和 loc 两列，因为 deptno 会被作为条件，用来去除无用

笛卡尔积，所以需要查询它。

```
SELECT dname,loc,deptno FROM dept;
```

第二步：替换第二步中的 dept

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e, (SELECT dname,loc,deptno FROM dept) d
WHERE e.deptno=d.deptno AND e.empno=7369
```

- 子查询作为表
- 子查询形式为多行多列

5. 查询薪资高于 3000 的部门信息

exists={ }此集合中必须有值，等号才能成立
not exists={ }同理此集合必须为空
“=”并不用来取值，而是用来进行判断是否相等
取值从外层进行

第一种方式:使用非相关子查询

```
select * from dept    where deptno in(
        select deptno from emp where sal > 3000
);
```

第二种方式:使用相关子查询

子查询中的deptno需要依赖主查询表中员工所在的部门，需要把主查询表中的d.deptno传递给子查询

```
select * from dept d where exists (
            select * from emp e where d.deptno = e.deptno and sal>3000)
```

exists 的相关子查询只关心内层查询是否有返回值;当内层没有返回值时，外层也没有结果. 如果内层有返回值，将内层的结果去查询外层数据

这个返回值应该是将内外两个表连起来的那个值，否则返回去之后也无法根据这个值进行查询，而是否返回该值与是否满足内层的条件有关

6.2.3 分页查询

在 Oracle 数据库中,使用 rownum 伪列来实现分页,oracle 数据库中对 rownum 的处理机制如下:

rownum 是在得到结果集的时候产生的,用于标记结果集中结果顺序的一个字段,这个字段称为”伪列”,也就是事实上不存在的一个列,它的特点是按”顺序标记”而且是”逐次

递加”,也就是说只有存在 rownum=1 的记录,才可能存在 rownum=2,假设我们的查询条件为 rownum>1,那么在查询出第一条记录的时候,oracle 标记此记录的 rownum 为 1,结果发现和 rownum>1 的条件不符,于是条件为假,没有结果.

所以如果想实现 rownum 大于某个值时,我们需要先把 rownum 固定,再进行大于的比较. 分页的实现:

第一种:两层 sql 嵌套实现分页:

```
--每页显示 3 条记录,查询第一页记录
select * from (select rownum r,t.* from emp t where rownum<=3 ) where  r>0
```

```
--每页显示 3 条，查第二页记录:
select * from (select rownum r,t.* from emp t where rownum<=6 ) where  r>3
```

```
--每页显示 pageSize 条，查第 page 页记录:  这里不能用别名，只能用rownum
select * from (select rownum r,t.* from emp t where rownum<=page*pageSize )
where  r>(page-1)*pageSize
```

第二种:三层 sql 嵌套实现分页

```
--每页显示 3 条，查第一页记录:
select * from (select rownum r,t.* from  (select * from emp) t where rownum<=3)
where r>0
```

```
--每页显示 3 条，查第二页记录:
select * from (select rownum r,t.* from  (select * from emp) t where rownum<=6)
where r>3
```

```
--每页显示 pageSize 条，查第 page 页记录:
select * from (select rownum r,t.* from  (select * from emp) t where
```

```
rownum<=page*pageSize) where r>(page-1)*pageSize
```

6.2.4 联合查询

1.联合查询就是把两个 select 语句的查询结果集合并到一起.

2.联合查询有如下方式:

- Union: 对两个结果集进行并集操作，不包括重复行，同时进行默认规则的排序
- Union All：对两个结果集进行并集操作，包括重复行，不进行排序;
- Intersect：对两个结果集进行交集操作，不包括重复行，同时进行默认规则的排序;
- Minus：对两个结果集进行差操作，不包括重复行，同时进行默认规则的排序

```
    SELECT deptno FROM emp
    UNION
    SELECT deptno FROM dept;
-- 查询结果: 10,20,30,40
```

```
    SELECT deptno FROM emp
    UNION ALL
    SELECT deptno FROM dept;
```

```
    SELECT deptno FROM emp
    INTERSECT
    SELECT deptno FROM dept;
     --结果: 10，,20,30
```

```
SELECT deptno FROM dept

MINUS

SELECT deptno FROM emp;

--结果:40
```

6.3 oracle 的系统函数

6.3.1 单行函数

单行函数对于从表中查询的每一行只返回一个值,可以出现在 select 子句和 where 子句中.

6.3.1.1 字符函数

--求字符串的长度 length

```
SELECT LENGTH('asdasdasd') FROM dual;
```

--首字母大写: 每一个单词的首字母大写 initcap

```
SELECT initcap('hello world') FROM dual;
```
Hello World

--小写: lower

```
SELECT LOWER('HELLO') FROM dual;
```
hello

--大写: upper

```
SELECT UPPER('ddd') FROM dual;
```
DDD

--左截 ltrim

```
SELECT LTRIM('        hello     ddd       ')    FROM dual;


--右截 rtrim

SELECT RTRIM('        hello     ddd       ')    FROM dual;

--左右截 trim（c1 from c2）

--去掉前后空格

SELECT TRIM('        hello     ddd       ')    FROM dual;

--abbcca  把左右的 a 截取

SELECT TRIM('a' FROM 'abbacca') as "xx"   FROM dual;


--替换 replace  全部替换

--  把 hello    cat 中的  cat  替换为 dog

SELECT REPLACE('hello cat','cat','dog') FROM dual;


--找子字符串出现的位置   第一次出现的位置    instr

-- aabbxxdd    b  第一次出现的位置

--java  字符串的下标从 0 开始

--oracle 字符串的下标从 1 开始

SELECT INSTR('aabbxxdd','b') FROM dual;


--截取指定位置的子字符串    substr

SELECT SUBSTR('helloxxworld',6) FROM dual;
```

```
SELECT SUBSTR('helloxxworld',6,2) FROM dual;


--连接  concat       ||   oracle 的连接

SELECT CONCAT('hello','world') FROM dual;

SELECT 'hello' || 'world'||'xx' FROM dual;
```

6.3.1.2.数字函数

```
--向下取整

SELECT FLOOR(12.1) FROM dual;   12

--  向上取整

SELECT CEIL(12.1) FROM dual;   13

--四舍五入

SELECT ROUND(12.48,3) FROM dual;12.48
```

6.3.1.3 日期函数

在函数内嵌套函数时不能加‘ ’，因为函数并不是字符串，不能识别，而如果是具体的值，则必须加‘ ’作为字符串进行识别

```
--SYSDATE 获取当前系统时间

SELECT SYSDATE FROM dual;

-- 默认的日期格式：  dd-m 月-yy

--获取两个日期相隔多少月  months_between（日期，日期）

-- 2012-12-01     2015-08-01

SELECT months_between('01-12 月-12','12-8 月-15') FROM dual;
```

--在日期加月份 add_months（日期，月份增量）

--获取当前的系统日期： sysdate

SELECT add_months(SYSDATE,-12) FROM dual;

--从指定日期开始，找到指定星期的日期 next_day(日期字符串,星期字符串)

SELECT next_day(SYSDATE, '星期六') FROM dual;

--查找指定日期这个月的最后一天的日期 last_day(日期)

SELECT last_day('01-2 月-15') FROM dual;

可以配合to_date函数使用

--练习 求去年这个月的最后一天的日期

6.3.1.4 转换函数

--转换格式的字符串 to_char(值，格式)

-- 12345678.212 --> $12,345,678.21

--9 任意一个数字

-- $12,345,678.21

SELECT to_char(12345678.212,'$999,999,999,999.99') FROM dual;

--0 任意一个数字

SELECT to_char(12345678.212,'$000,000,000,000.00') FROM dual;

--yyyy:四位的年　yy：两位的年，　mm:两位的月数字，month：月份，带月字

-- dd: 日 ddd:表示一年的第几天　day：星期　ww:一年的第几个星期　w:一月的第几个星期

--hh: 小时，　12 进制　hh24: 小时，24 进制　mi：分钟　ss:秒　ff:秒之后的时间

```
SELECT to_char(SYSDATE, 'yyyy') FROM dual;

SELECT to_char(SYSDATE, 'yy') FROM dual;

SELECT to_char(SYSDATE, 'yyy') FROM dual;

SELECT to_char(SYSDATE, 'month') FROM dual;

SELECT to_char(SYSDATE, 'dd') FROM dual;

SELECT to_char(SYSDATE, 'ddd') FROM dual;


SELECT to_char(SYSDATE, 'w') FROM dual;


-- 现在的日期是在一年的第多少天
```

对于to_date（‘年-月-日’，‘yyyy-mm-dd’），前面的时间和后面的格式一定要相符，如果我们只对年份进行转换，to_date('2019','yyyy')那么输入系统的时候，系统会将这个时间转换成系统格式，但是系统时间是有月份和日期的，在只有年份的情况下，系统会以当前系统时间的月份和日期对输入的时间进行填充

-- dd-m 月-yy: oracle 默认把这个格式认为是一个日期 ***

--把指定的格式转换为日期 to_date(值，格式)

把日期转换成指定的格式

-- 2012-12-12　2012/12/12　2012 年 12 月 12 日　20121212

```
SELECT to_date('2012-12-12','yyyy-mm-dd') FROM dual;
```

--转换为数字 to_number(值，[格式])

--计算 $88.34-12

select to_number('$88.34','$999.99') -12 from dual;

6.3.1.4 其他函数

--nvl(exp1,exp2) 如果 exp1 值为 null,则返回 exp2 的值，否则返回 exp1 的值

--查询每一个员工的一个月的收入

-- comm null， null+/-任意的数字 都变为 null

select * from emp;

select (sal+ nvl(comm,0)) "月收入" from emp;

--nvl2(exp1,exp2,exp3) 如果 exp1 值为 null,则返回 exp3 的值，否则返回 exp2 的值

select (sal+ nvl2(comm,comm,0)) "月收入" from emp;

--decode(value，if1,then1,if2,then2...,else) if --else if --else if -- else ****

--如果 value 的值等于 if1 则返回 then1 ,如果等于 if2,则返回 then2,,,否则返回 else

--加工资， 岗位是 clerk 的加 500， salesman 加 200 ， analyst 加 100，其他岗位不

加

```
select * from emp;

select ename, decode(job,
                     'CLERK',sal+500,
                     'SALESMAN',sal+200,
                     'ANALYST',sal+100,
                     sal
                    ) from emp;
```

--练习： 计算每一个员工年薪(工资加奖金)*12

6.3.1.5 分析函数

语法： 函数名([参数]) over ([分组子句] [排序子句]))

- row_number 使用：它会为查询出来的每一行记录生成一个序号，依次排序且不会重复，注意使用 row_number 函数时必须要用 over 子句选择对某一列进行排序才能生成序号。

row_number 的语法：row_number() over (partition by column1 order by column2)

表示根据 column1 进行分组，在分组内部根据 column2 排序，此函数计算的值表示每组内部排序后的顺序编号，组内编号连续

```
--根据 deptno 分组，每个部门薪水由高到低排序

select empno,ename,sal ,deptno,
```

```
row_number() over( partition by deptno order by sal desc ) "row_number"

from emp;
```

查询结果:

	EMPNO	ENAME	SAL	DEPTNO	row_number
1	7839	KING	5000.00	10	1
2	7782	CLARK	2450.00	10	2
3	7934	MILLER	1300.00	10	3
4	7902	FORD	3000.00	20	1
5	7788	SCOTT	3000.00	20	2
6	7566	JONES	2975.00	20	3
7	7876	ADAMS	1100.00	20	4
8	7369	SMITH	800.00	20	5
9	7698	BLAKE	2850.00	30	1
10	7499	ALLEN	1600.00	30	2
11	7844	TURNER	1500.00	30	3
12	7654	MARTIN	1250.00	30	4
13	7521	WARD	1250.00	30	5
14	7900	JAMES	950.00	30	6
15	1200	李四	5000.00		1
16	1100	张三	5000.00		2
17	149	张三	5000.00		3

- rank():具有相等值的行排位相同，序数随后跳跃

```
select empno,ename,sal ,deptno,

        rank () over( partition by deptno order by sal desc ) "rank"

        from emp;
```

加双引号表示引号内的字符，不加表示字母，系统默认大写，对汉字没有影响，但是针对列名数字开头必须加双引号，否则无法显示

结果:

	EMPNO	ENAME	SAL	DEPTNO	rank
1	7839	KING	5000.00	10	1
2	7782	CLARK	2450.00	10	2
3	7934	MILLER	1300.00	10	3
4	7902	FORD	3000.00	20	1
5	7788	SCOTT	3000.00	20	1
6	7566	JONES	2975.00	20	3
7	7876	ADAMS	1100.00	20	4
8	7369	SMITH	800.00	20	5
9	7698	BLAKE	2850.00	30	1
10	7499	ALLEN	1600.00	30	2
11	7844	TURNER	1500.00	30	3
12	7654	MARTIN	1250.00	30	4
13	7521	WARD	1250.00	30	4
14	7900	JAMES	950.00	30	6

- dense_rank:具有相等值的行排位相同，序号是连续的

```
select empno,ename,sal ,deptno,
        dense_rank () over( partition by deptno order by sal desc ) " dense_rank "
        from emp;
```

结果:

EMPNO	ENAME	SAL	DEPTNO	dense_rank
7839	KING	5000.00	10	1
7782	CLARK	2450.00	10	2
7934	MILLER	1300.00	10	3
7902	FORD	3000.00	20	1
7788	SCOTT	3000.00	20	1
7566	JONES	2975.00	20	2
7876	ADAMS	1100.00	20	3
7369	SMITH	800.00	20	4
7698	BLAKE	2850.00	30	1
7499	ALLEN	1600.00	30	2
7844	TURNER	1500.00	30	3
7654	MARTIN	1250.00	30	4
7521	WARD	1250.00	30	4
7900	JAMES	950.00	30	5

7.练习

1. 查询出部门编号为 30 的所有员工

2. 所有销售员的姓名、编号和部门编号

3. 找出奖金高于工资的员工

4. 找出奖金高于工资 60%的员工

5. 找出部门编号为 10 中所有经理，和部门编号为 20 中所有销售员的详细资料

6. 找出部门编号为 10 中所有经理，部门编号为 20 中所有销售员，还有即不是经理又不是销售员但其工资大或等于 2000 的所有员工详细资料

7. 有奖金的岗位

8. 无奖金或奖金低于 500 的员工

9. 查询名字由五个字组成的员工

10. 查询 1982 年入职的员工,

11. 查询所有员工详细信息，用编号升序排序

12. 查询所有员工详细信息，用工资降序排序，如果工资相同使用入职日期升序排序

13. 查询每个部门的平均工资

14. 求出每个部门的员工数量

15. 查询每个岗位的最高工资、最低工资、人数

16. 查询非销售人员工作名称以及从事同一工作雇员的月工资的总和，并且要满足从事同一工作的雇员的月工资合计大于 2000，输出结果按月工资的合计升序排列

17. 查出至少有一个员工的部门。显示部门编号、部门名称、部门位置、部门人数

18. 列出工资比 ALLEN 高的所有员工

19. 列出所有员工的姓名及其直接上级的姓名

20. 列出受雇日期早于直接上级的所有员工的编号、姓名、部门名称

21. 列出部门名称和这些部门的员工信息，同时列出那些没有员工的部门

22. 列出所有文员(CLERK)的姓名及其部门名称，部门的人数

23. 列出最低薪金大于 1500 的各种岗位及从事此岗位的员工人数

24. 列出在销售部(SALESMAN)工作的员工的姓名，假定不知道销售部的部门编号

25. 列出薪金高于公司平均薪金的所有员工信息，所在部门名称，上级领导

26. 列出与 SMITH 从事相同工作的所有员工及部门名称

27. 列出薪金高于在部门 30 工作的所有员工的薪金的员工姓名和薪金、部门名称

28. 列出在每个部门工作的员工数量、平均工资