

第 3 章 SQL 语言

SQL 语言是操作关系型数据库的通用语言，与关系型数据库一样，至今也有了 40 多年的历史。与其他编程语言相比，SQL 语言与平时常用的英语更接近，比较容易掌握。SQL 语言可以交互执行，也可以嵌入到其他编程语言中执行。从事数据库应用开发和数据库管理工作都需要具备快速编写 SQL 语句的能力。

本章主要包括：

- SQL 概览
- select, insert, update, delete 语句
- null 值的处理
- 创建表与修改表的结构
- identity 列
- 约束
- 视图
- 序列
- 同义词

3.1 SQL 概览

SQL 语言由 IBM 的 Donald D. Chamberlin 和 Raymond F. Boyce 开发，用于在 System R 项目中实现数据访问，于 1974 年发布，其最初目的是让非计算机专业人员能够使用 SQL 语言操作数据库。SQL 使用面向集合的方式而不是传统的面向过程的方式查询数据。1979 年，Oracle 公司(当时的公司名称为 Relational Software, Inc)推出了第一个商业化关系数据库产品，并使用了与 IBM 公司兼容的 SQL 语言。

随着关系型数据库的广泛使用，SQL 语言也成为其标准操作语言，并于 1986 年由 ANSI 进行了标准化，称为 SQL-86，1987 年被 ISO 接受。SQL 语言的功能不断扩充，其标准也随之不断修改，当前最新标准为 2011 年发布的 SQL:2011。

当前各个数据库产品的 SQL 语言一般都支持 SQL-92 标准的主要部分，但在语法上还不能完全兼容。

SQL 表示 Structured Query Language，但其功能已远远不限于提供查询。传统上，按其功能，SQL 一般分为以下三类：

DDL: Data Definition Language，数据定义语言

DML: Data Manipulation Language，数据操纵语言

DCL: Data Control Language，数据控制语言

以上三类语言的功能和主要语句如下表所示：

表 3-1 SQL 语言分类

分类	主要语句	功能
DML	select	查询表的数据
	update	修改表中的列值
	delete	删除表中的记录
	insert	对表添加新记录
	merge	有条件地添加或更新记录

DDL	create	创建数据库对象，如表、视图、索引等。
	alter	修改数据库对象定义，如对表添加或删除列
	drop	删除数据库对象
	truncate	删除表中的所有记录，不会触发 delete 触发器
DCL	grant	对用户赋予权限
	revoke	收回用户的权限

DML 语言中的几个语句除了 insert 语句，其他几个都会涉及查询。在底层，对于 DML 语句，需要在执行语法和语义检查后，制定执行计划，然后根据执行计划完成任务，而 DDL 语句只需执行语法和语义检查，然后直接执行。

Oracle 把 SQL 语言分为以下几类：

- DDL 语句
- DML 语句
- 事务控制语句
- 会话控制语句
- 系统控制语句
- 嵌入式 SQL 语句

在 Oracle 中，DCL 包含在 DDL 中。除了以上几个语句，Oracle 的 DDL 还包括执行审计功能的 audit、noaudit 语句，以及添加注释的 comment 语句。

Oracle 中的 DML 还包括查看执行计划的 explain plan 以及临时禁止用户访问指定表的 lock table 语句。

在事务处理方面，Oracle 中执行 DDL 时，会在其前后自动执行 commit 操作，从而不能回滚 DDL 语句。而在 SQL Server 中，DDL 与 DML 的处理方式相同，即执行 DDL 时，并不会自动执行 commit 操作，DDL 的执行效果也可以回滚。

3.2 select 语句

select 语句用于对表的查询，是 SQL 语言中最重要、最常用的部分，关系代数中的运算都是指查询操作。本节介绍 Oracle 和 SQL Server 在 select 各种子句的细微差异。

3.2.1 简单查询

这里所说的简单查询指不包含子查询的单表查询，其典型的形式包括下面 6 部分，也称为 6 个子句：

select ... from ... where ... group by ... having ... order by ...

对于这种简单查询，Oracle 与 SQL Server 的语法形式基本是相同的，这里不再赘述。两者的不同点主要是列别名及字符串条件的语法形式，下面分别说明。

Oracle 中支持两种形式的列别名语法形式，一种直接把别名附加在列名后面：

```
SQL> select ename emp_name, sal salary
2  from emp
3  where deptno=20
4  /
```

EMP_NAME	SALARY
SMITH	800

JONES	2975
FORD	3000

另外一种是在列名与列别名之间再附加 as 关键字：

```
SQL> select ename as emp_name, sal as salary
2  from emp
3  where deptno=20
4  /
```

EMP_NAME	SALARY
SMITH	800
JONES	2975
FORD	3000

SQL Server 除了支持上述两种语法形式外，还支持把列别名置于列名之前，并附加等号：

```
1> select emp_name=ename, salary=sal
2> from emp
3> where deptno=20
4> go
emp_name    salary
-----
SMITH       800.00
JONES       2975.00
FORD        3000.00
(3 行受影响)
```

Oracle 与 SQL Server 都支持 like 关键字、“%”及“_”作为通配符以及使用 escape 关键字指定转义字符进行模糊匹配查询，如查询 dept 表中 dname 列的第二个字母为 A 的记录，以下命令 Oracle 与 SQL Server 都是支持的：

```
select * from dept where dname like '_A%'
```

除了支持以上字符串模糊匹配的用法以外，SQL Server 还支持正则表达式中的方括号用法，以匹配指定范围或指定集合中的任意单个字符。Oracle 使用 regexp_like 函数对正则表达式实现了更完整的支持。

SQL Server 支持的方括号用法有两种形式，[]与[^]，前者用于包含某些字符，后者用于排除某些字符，举例如下：

[amd]：表示包含 a、m、d 三个字符中的任意一个。

[^amd]：表示不包含 a、m、d 三个字符中的任意一个。

[b-f]：表示英文字母表中 b 到 f 之中的任意一个。

[0-9]：表示 0 到 9 这 10 个数字中的任意一个。

如查询 dept 表的 dname 列中第一及第二个字符为数字，第三个字符为小写英文字母的记录，可以使用如下语句：

```
1> select *
2> from dept
3> where dname like '[0-9][0-9][a-z]%'
4> go
```

3.2.2 多表连接

本节内容的查询除非特殊说明，都是查询 emp 表中的 ename 及其在 dept 表中对应的 dname，即每个员工的姓名及其所在的部门的名称。

交叉连接(cross join)即不附加连接条件的连接形式，返回两个表的所有行的两两拼接结

果,也称为笛卡尔连接(Cartesian join),在关系代数术语中也称为笛卡尔积(Cartesian product)。交叉连接的语法有 SQL-89 及 SQL-92 两种形式,Oracle 与 SQL Server 对这两种形式都支持。

SQL-89 的语法形式:

```
select e.ename, d.dname
from emp e, dept d
```

SQL-92 的语法形式

```
select e.ename, d.dname
from emp e cross join dept d
```

交叉连接的查询结果中,多数都是没有任何意义的拼接结果,若从中获得有意义的结果,则需要附加连接条件。交叉连接附加连接条件后,称为内连接,其查询结果为两个表中满足连接条件的记录拼接而成。

在 SQL-89 标准中,内连接的连接条件与过滤条件都放置于 where 子句中,这种形式在一些文档中被称为 old-style join。

在 SQL-92 标准中,使用[inner] join 关键字(inner 可省略),把连接条件放置于 on 子句中,在 where 子句放置针对单表的过滤条件,从而把连接条件与过滤条件分开,使得内连接的语法形式可读性更好。

Oracle 和 SQL Server 对这两种内连接语法形式都支持,下面命令查询 emp 表中的 ename 及其在 dept 表中对应的 dname,即每个员工名称及其部门名称。

SQL-89 的语法形式,连接条件使用 where 关键字:

```
select e.ename, d.dname
from emp e, dept d
where e.deptno=d.deptno
```

SQL-92 的语法形式,使用 join ... on 的形式:

```
select e.ename, d.dname
from emp e join dept d
on e.deptno=d.deptno
```

对于内连接的 SQL-86 和 SQL-92 两种语法形式,当前的 SQL 标准都支持,也会一直支持下去,Oracle 的相关帮助文档并未表现出对 SQL-2 语法形式的特别偏好,但 SQL Server 2016 的帮助文档中,涉及多表连接的示例已经都使用 SQL-92 语法形式。

自然连接(natural join)是内连接的特殊形式,在 SQL-92 标准引入,即对两个表执行连接查询时,连接条件中的字段在两个表中是同名的。对于自然连接,Oracle 支持 natural join 以及 using 关键字的用法,而 SQL Server 不支持。

在 Oracle 中使用 natural join 查询 emp 表中的 ename 及其在 dept 表中对应的 dname:

```
SQL> select e.ename, d.dname
2   from emp e natural join dept d
3   /
```

同样的语法形式在 SQL Server 中执行却是不支持的:

```
1> select e.ename, d.dname
2> from emp e natural join dept d
3> go
消息 102, 级别 15, 状态 1, 服务器 LAW_X240, 第 2 行
'natural' 附近有语法错误。
```

在 Oracle 中也可以使用 using 关键字:

```
SQL> select e.ename, d.dname
2   from emp e join dept d
3   using (deptno)
4   /
```

同样的语法形式 SQL Server 则不支持：

```
1> select e.ename, d.dname
2> from emp e join dept d
3> using (deptno)
4> go
消息 102, 级别 15, 状态 1, 服务器 LAW_X240, 第 3 行
'using' 附近有语法错误。
```

内连接的结果是两表中符合连接条件的数据，不满足连接条件的数据则未显示，如果需要把两者都显示出来，则需要使用外连接。外连接分为左外连接、右外连接及全外连接，在语法上，也有 SQL-92 标准形式和传统形式两种，Oracle 12c 与 SQL Server 2016 都支持 SQL-92 标准。

SQL-92 标准的三种外连接分别使用 left outer join，right outer join 及 full outer join(outter 可以省略)，其含义如下：

- 左外连接：除了把符合内连接的数据显示出来以外，也把 SQL 命令中 left outer join 左侧表中不满足连接条件的数据显示出来。
- 右外连接：除了把符合内连接的数据显示出来，也把 SQL 命令中 right outer join 右侧表中不满足连接条件的数据显示出来。
- 全外连接：除了把符合内连接的数据显示出来，也把 SQL 命令中 full outer join 两侧的表中不满足条件的数据显示出来。

下面命令可以在两个产品中执行。

左外连接：

```
select e.ename, d.dname
from emp e left join dept d
on e.deptno=d.deptno
```

右外连接：

```
select e.ename, d.dname
from emp e right join dept d
on e.deptno=d.deptno
```

全外连接：

```
select e.ename, d.dname
from emp e full join dept d
on e.deptno=d.deptno
```

对于外连接的传统语法形式，两种产品只支持左外连接和右外连接。Oracle 12c 支持在 where 子句中使用 “(+)” 实现外连接。SQL Server 早期版本使用 “*” 实现外连接，从 2005 版本开始不再支持这种传统语法形式。

在 Oracle 中使用传统语法形式执行左外连接查询：

```
SQL> select e.ename, d.dname
2   from emp e, dept d
3   where e.deptno=d.deptno(+)
4   /
```

在 Oracle 中使用传统语法形式执行右外连接：

```
SQL> select e.ename, d.dname
      2  from emp e, dept d
      3  where e.deptno(+) = d.deptno
      4  /
```

在 SQL Server 中使用传统语法形式执行左外连接，可以发现不再支持：

```
1> select e.ename, d.dname
2> from emp e, dept d
3> where e.deptno(*) = d.deptno
4> go
消息 102, 级别 15, 状态 1, 服务器 LAW_X240, 第 3 行
“*” 附近有语法错误。
```

3.2.3 子查询

子查询是在 DML 语句中包含的查询，这里我们只说明最常用的 select 语句中的子查询。

Oracle 与 SQL Server 都支持在 select 子句、from 子句、where 子句中使用子查询，在 select 子句与 where 子句中使用子查询的用法是相同的，在 from 子句使用子查询稍有区别。

以下两个子查询在 Oracle 与 SQL Server 中都可以成功执行。

在 select 子句中使用子查询，查询每个员工的 sal 值与平均 sal 值的差距：

```
select ename, sal, sal - (select avg(sal) from emp) from emp
```

在 where 子句使用子查询，查询获得最高 sal 值的员工名称：

```
select ename, sal
from emp
where sal = (select max(sal) from emp)
```

from 子句包含子查询时，Oracle 并未要求必须使用表别名，下面子查询未使用表别名：

```
SQL> select ename from (select * from emp where deptno=20);

ENAME
-----
SMITH
JONES
FORD
```

若使用表别名，则不能在表别名之前附带 as 关键字：

```
SQL> select ename from (select * from emp where deptno=20) e;

ENAME
-----
SMITH
JONES
FORD
```

若 from 子句中只有一个子查询，也没有另外的表，即不涉及表连接，子查询不使用表别名会使语法显得更直观、简洁。若 from 子句中不止一个子查询，还有另外的子查询或表，则使用表别名限制查询语句中出现的各个列名称。

from 子句使用子查询，SQL Server 要求使用表别名，不使用表别名，则语法不能通过：

```
1> select ename from (select * from emp where deptno=20)
2> go
消息 102, 级别 15, 状态 1, 服务器 LAW_X240, 第 1 行
“)” 附近有语法错误。
```

使用表别名时，对是否使用 as 关键字不加限制，下面语法形式都是正确的：

```
select ename from (select * from emp where deptno=20) as e
select ename from (select * from emp where deptno=20) e
```

3.2.4 分页查询

分页查询是指取出排序后的从指定行开始的指定行数的记录。出于在客户端界面中分页显示查询结果的需要，分页查询一直是开发人员迫切需要的功能。MySQL 和 PostgreSQL 使用 limit ... offset ... 形式的子句很早就完美地支持分页查询功能，但此功能直到 SQL:2011 才标准化，其标准语法形式由 offset 和 fetch 两部分组成(offset 部分可选)，分为下面两种形式：

- offset *m* rows fetch next *n* [percent] rows only
- offset *m* rows fetch next *n* [percent] rows with ties

其中的 rows 可以替换为 row，next 可以替换为 first，可根据语法可读性选择其一。

此子句一般附加在 order by 子句之后，表示排序后，略过前 *m* 行，再取出 *n* 行。若在 fetch 部分使用 percent 关键字，则以百分比指定取出记录的数量。

若最后一行的排序列值重复，需要把重复的行也取出，则把 only 替换为 with ties。

若省略 offset 部分，则取出排序后的前 *n* 行，即 *m* 默认为 0，这也是 SQL:2008 引入的标准形式。

依照 SQL:2011 标准，若得到 emp 表中按 sal 值由低到高排序后，由第 10 行开始的 4 行记录，可执行下面命令：

```
select ename, sal from emp
order by sal asc
offset 9 rows fetch next 3 rows only
```

若以上结果中，最后一行的 sal 值有重复，则可以把 only 关键字替换为 with ties 把重复记录也显示出来：

```
select ename, sal from emp
order by sal asc
offset 9 rows fetch next 3 rows with ties
```

若得到排序后的前 3 行，则省略 offset 部分(为提高可读性，把 next 替换为 first)：

```
select ename, sal from emp
order by sal asc
fetch first 3 rows with ties
```

若不需取出第 3 行的重复行，则把 with ties 替换为 only。

在 12c 之前版本，Oracle 使用 rownum 伪列或 row_number() 分析函数实现此功能，语法形式较为繁琐，12c 版本完整支持上述 SQL:2011 的分页语法标准，上述 3 个示例语句可以不加修改地在 Oracle 12c 执行。

下面查询执行上述第二个示例语句：

```
SQL> select ename, sal from emp
2  order by sal asc
3  offset 9 rows fetch next 3 rows with ties
4  /
```

ENAME	SAL
BLAKE	2850
JONES	2975

SCOTT	3000
FORD	3000

在 2012 版本之前，SQL Server 使用 `top n` 子句可以间接实现分页查询功能，SQL Server 2012 部分支持 SQL:2011 的分页查询标准，SQL Server 2016 版本与 SQL:2011 的分页查询标准存在以下三个方面的区别：

- 不能省略 `offset` 部分。
- 不支持以百分数指定取出记录的数量。
- 不支持 `with ties` 选项。

下面查询执行上述第一个示例语句：

```
1> select ename, sal from emp
2> order by sal asc
3> offset 9 rows fetch next 3 rows only
4> go
ename      sal
-----
BLAKE      2850.00
JONES      2975.00
SCOTT      3000.00
```

3.2.5 集合运算

集合运算主要包括并、交、差三种形式。

Oracle 使用：

- `union`：返回两个查询结果的并集，剔除重复记录
- `union all`：返回两个查询结果的并集，包含重复记录
- `intersect`：求交集，剔除重复记录
- `minus`：求差集，剔除重复记录

SQL Server 使用：

- `union`：返回两个查询结果的并集，剔除重复记录
- `union all`：返回两个查询结果的并集，包含重复记录
- `intersect`：求交集，剔除重复记录
- `except`：求差集，剔除重复记录

对于集合运算，Oracle 与 SQL Server 支持的前三个运算符相同，用法也相同；最后的差集运算符不同，SQL Server 的 `except` 符合 SQL-92 标准。

以下示例均可以在 Oracle 与 SQL Server 中执行：

```
select deptno from emp
intersect
select deptno from dept

select deptno from dept
union all
select deptno from emp

select deptno from dept
union
select deptno from emp
```

`minus` 操作符只能用于 Oracle：

```
SQL> select deptno from dept
2 minus
```

```

3 select deptno from emp
4 /

DEPTNO
-----
40

```

except 操作符只能用于 SQL Server:

```

1> select deptno from dept
2> except
3> select deptno from emp
4> go
deptno
-----
40

```

3.2.6 时态数据库相关查询

时态数据库(也称为时态表)除了提供对数据库中当前数据的查询以外,也可以指定时刻查询过去的历史数据。时态数据库是 SQL:2011 的新增功能, Oracle 从 9.2 版本开始支持时态数据库功能, SQL Server 从 2016 版本开始支持时态数据库。Oracle 的 as of timestamp 语法形式基本符合 SQL:2011 标准。

下面先介绍 Oracle 的时态数据库功能。

Oracle 用 undo 数据支持时态表,不需额外配置。可回溯的历史数据与 undo_retention 相关,这个参数用于设置事务提交后,其 undo 数据在 undo 表空间保留的时间,其默认值为 900 秒,即 15 分钟。因为空间问题,过去某个时刻的 undo 数据可能已经被覆盖,这个时刻的数据就不能查询出来了。改变表的结构后,之前的 undo 数据就不能用来支持时态表查询了。

创建测试表 em:

```

SQL> create table em
2 (
3     eno number(4),
4     ename varchar2(10),
5     sal number(5)
6 )
7 /

```

执行下面 SQL 脚本文件添加一行记录,然后每隔 2 分钟对其执行一次 update 操作:

```

select to_char(sysdate, 'yyyy-mm-dd hh24:mi:ss') as now from dual;
insert into em values(7369, 'SMITH', 800);
commit;
exec dbms_lock.sleep(120)
update em set sal=1000 where eno=7369;
commit;
exec dbms_lock.sleep(120)
update em set sal=1200 where eno=7369;
commit;
exec dbms_lock.sleep(120)
update em set sal=1400 where eno=7369;
commit;

```

直接查询 em 表,可以得到当前数据:

```

SQL> select * from em;

```

ENO	ENAME	SAL
7369	SMITH	1400

查询 10 分钟以内的所有版本数据(先设置客户端时间格式，以方便查看时间信息):

```
SQL> alter session set nls_timestamp_format='yyyy-mm-dd hh24:mi:ss';
会话已更改。
SQL> select eno,ename,sal,versions_starttime,versions_endtime
2  from em
3  versions between timestamp
4  systimestamp-10/1440 and systimestamp
5  /
```

ENO	ENAME	SAL	VERSIONS_STARTTIME	VERSIONS_ENDTIME
7369	SMITH	1400	2016-08-10 21:24:23	
7369	SMITH	1200	2016-08-10 21:22:22	2016-08-10 21:24:23
7369	SMITH	1000	2016-08-10 21:20:23	2016-08-10 21:22:22
7369	SMITH	800	2016-08-10 21:18:22	2016-08-10 21:20:23

versions_starttime, versions_endtime 是两个伪列，用于记录行的有效时间范围。

查询旧版本的数据，可以指定时刻或时间范围，也可以指定 SCN 号或 SCN 范围。下面两个示例分别使用 as of timestamp 和 versions between timestamp 子句，查询满足指定时刻和时间范围的记录。

使用 as of timestamp 子句查询指定时刻的有效记录:

```
SQL> select eno,ename,sal
2  from em
3  as of timestamp timestamp'2016-08-10 21:21:00'
4  /
```

ENO	ENAME	SAL
7369	SMITH	1000

使用 versions between timestamp 查询指定时间范围的有效记录:

```
SQL> select eno,ename,sal
2  from em
3  versions between timestamp
4  timestamp'2016-08-10 21:20:30' and
5  timestamp'2016-08-10 21:23:00'
6  /
```

ENO	ENAME	SAL
7369	SMITH	1200
7369	SMITH	1000

SQL Server 使用两个表实现时态表功能，一个用于保存当前数据(current table)，一个用于保存历史数据(history table)。两个表各有两个列用来保存记录的有效时间范围，对应 Oracle 的 versions_starttime 和 versions_endtime 列，与 Oracle 不同的是，这两个列的名称由用户指定。

下面命令创建具备时态功能的 em 表:

```
1> create table em
```

```

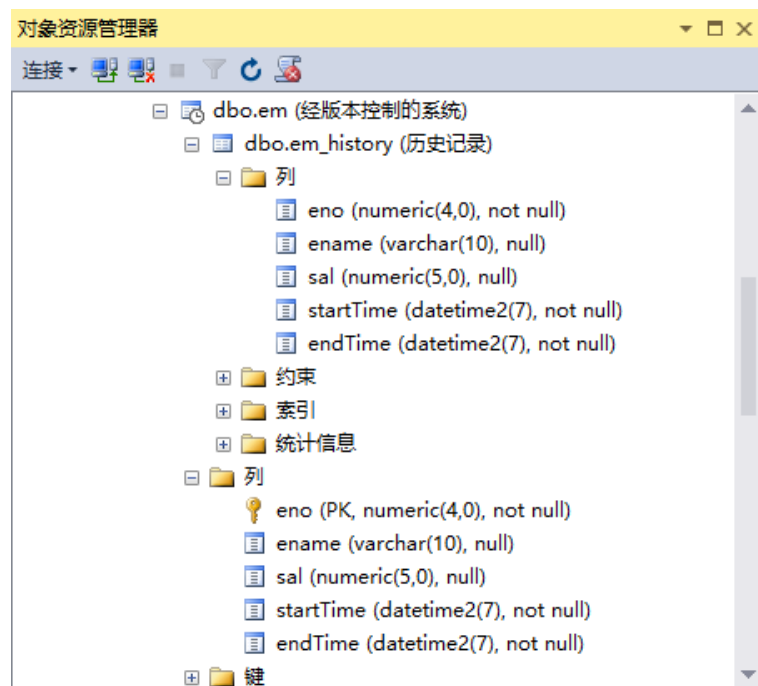
2> (
3>     eno numeric(4) primary key,
4>     ename varchar(10),
5>     sal numeric(5),
6>     startTime datetime2 generated always as row start hidden not null,
7>     endTime datetime2 generated always as row end hidden not null,
8>     period for system_time(startTime, endTime)
9> )
10> with (system_versioning=on (history_table=dbo.em_history))
11> go

```

下面简单说明以上命令涉及时态表的几个属性：

- 第 3 行：时态表要附加主键约束。
- 第 6 行：记录行有效期的起始时刻。其类型为 datetime2。
- 第 7 行：记录行有效期的结束时刻。其类型为 datetime2。
- 第 8 行：period 列用于指定行有效期起讫时刻的列。
- 第 10 行：system_versioning 属性设置为 on，开启表的时态属性。
- 第 10 行：history_table 指定与此表配对的历史记录表，其结构拷贝自 em 表，表名前的架构名称不能省略。若省略此部分，则由 SQL Server 指定其表名。

可以在 SSMS 中看到 em 表与其历史表的关系：



执行以下 SQL 脚本文件，对 em 表添加一行记录后每隔 2 分钟对其执行 update 操作：

```

select getdate() as now
go
insert into em values(7369, 'SMITH', 800)
go
waitfor delay '00:02:00'
go
update em set sal=1000 where eno=7369
go
waitfor delay '00:02:00'
go

```

```

update em set sal=1200 where eno=7369
go
waitfor delay '00:02:00'
go
update em set sal=1400 where eno=7369
go

```

查询包括当前记录在内的所有记录，可以执行下面命令：

```

1> select sal,startTime,endTime from em
2> for system_time all
3> go

```

sal	startTime	endTime
1400	2016-08-09 00:41:43.1894918	9999-12-31 23:59:59.9999999
800	2016-08-09 00:35:43.0277558	2016-08-09 00:37:43.0922101
1000	2016-08-09 00:37:43.0922101	2016-08-09 00:39:43.1720506
1200	2016-08-09 00:39:43.1720506	2016-08-09 00:41:43.1894918

(4 行受影响)

查询 em_history 表，可以发现每次执行 update 操作时，其更新之前的原记录及此记录有效的起讫时刻被存入了 em_history 表：

```

1> select sal,startTime,endTime from em_history
2> go

```

sal	startTime	endTime
800	2016-08-09 00:35:43.0277558	2016-08-09 00:37:43.0922101
1000	2016-08-09 00:37:43.0922101	2016-08-09 00:39:43.1720506
1200	2016-08-09 00:39:43.1720506	2016-08-09 00:41:43.1894918

(3 行受影响)

查询 em 表，可以得到其当前记录：

```

1> select * from em
2> go

```

eno	ename	sal
7369	SMITH	1400

(1 行受影响)

下面几个命令得到其不同时刻的数据。

查询 00:36 时有效的数据：

```

1> select * from em
2> for system_time as of '2016-08-09 00:36'
3> go

```

eno	ename	sal
7369	SMITH	800

查询 00:38 时有效的数据：

```

1> select * from em
2> for system_time as of '2016-08-09 00:38'
3> go

```

eno	ename	sal
7369	SMITH	1400

7369 SMITH	1000
------------	------

查询 00:40 时有效的数据:

```

1> select * from em
2> for system_time as of '2016-08-09 00:40'
3> go

```

eno	ename	sal
7369	SMITH	1200

查询在'2016-08-09 00:37:43' 与 '2016-08-09 00:39'之间有效的数据:

```

1> select * from em
2> for system_time from '2016-08-09 00:37:43' to '2016-08-09 00:39'
3> go

```

eno	ename	sal
7369	SMITH	800
7369	SMITH	1000

(2 行受影响)

除了可以用 `from ... to ...` 子句指定起始时刻外，还可以使用 `between ... and ...` 和 `contained in()` 子句指定查询范围，其区别主要是在是否包括范围边界。四种表示时间范围的子句，其确切界限如下表所示，其中的 `startTime` 和 `endTime` 表示时态表的起始时刻列：

范围子句	具体界限
<code>as of t</code>	<code>startTime <= t AND endTime > t</code>
<code>from t1 to t2</code>	<code>startTime < t2 AND endTime > t1</code>
<code>between t1 and t2</code>	<code>startTime <= t2 AND endTime > t1</code>
<code>contained in(t1, t2)</code>	<code>startTime >= t1 AND endTime <= t2</code>

对于时态表及其历史记录表有以下两点需要注意：

- 对时态表不能执行 `drop table` 和 `truncate table` 操作。
- 对历史表不能执行除查询之外的 DML 操作。

执行以上操作，需要把时态表的 `system_versioning` 设置为 `off`，设置为 `off` 后，时态表及其历史表即成为普通表，两者也不再有关联关系。

关闭 `system_versioning` 的命令如下：

```

1> alter table em set (system_versioning=off)
2> go

```

执行下面命令可以查询数据库中的时态表及其对应的历史表：

```

1> select name as temporal_table, object_name(history_table_id) as history_table
2> from sys.tables
3> where temporal_type=2
4> go

```

temporal_table	history_table
em	em_history

3.3 insert 语句

insert 语句的基本形式为：

```
insert into table_name(column_list) values(value_list)
```

```
insert into table_name values(value_list)
```

这两种基本用法，Oracle 与 SQL Server 都支持。

另外，如果对应列的类型一致，可以把 select 语句查询的结果使用 insert 语句填入表中，Oracle 与 SQL Server 对这种用法也都支持，语法形式如下：

```
insert into table_name(column_list) select_clause
```

下面命令可以在 Oracle 与 SQL Server 执行。

先创建 t 表，包含两个列 a、b，分别与 emp 表的 ename 及 sal 列的类型相同：

```
create table t(a varchar(10), b numeric(7,2));
```

然后使用 insert ... select 语句把由 emp 表查询的数据填入 t 表：

```
insert into t
select ename,sal
from emp where deptno=30
```

SQL Server 还支持以下两种用法。

一是可以在一个 insert 语句中对表添加多条记录。如下面示例所示：

```
1> insert into dept values(50,'a','b'),(60,'d','e')
2> go
```

二是可以把存储过程的结果添加到表中。

如果存储过程的执行结果是表的查询结果，则可以结合 insert 语句把这些结果添加到一个表中，要求表中列的数据类型及顺序与存储过程的结果一致。

下面的存储过程由 emp 表得到 ename 及 sal 列的值：

```
1> create procedure select_enameandsal
2> as
3> select ename, sal from emp
4> go
```

创建 t 表，包含两个列，与上述存储过程执行结果中的列的数据类型及顺序一致：

```
1> create table t(a varchar(15), b numeric(7,2))
2> go
```

然后可以把存储过程的执行结果以下面命令添加到 t 表中：

```
1> insert into t
2> execute select_enameandsal
3> go
```

3.4 update 语句

update 语句的用法，Oracle 和 SQL Server 是相同的，即：

```
update table_name set column_name=value1, column_name=value2 ...where condition
```

下面示例修改 emp 表的 sal 与 comm 列的值，在 Oracle 与 SQL Server 上都可以执行：

```
update emp set sal=sal+1000, comm=500
where deptno=30
```

SQL Server 还支持以下类似 C 语言中的赋值语法形式:

```
1> update emp set sal+=1000, comm=500
2> where deptno=30
3> go
```

类似的运算符还包括: “-=”, “*=”, “/=” 等。

3.5 delete 语句

delete 语句在 Oracle 与 SQL Server 中的用法也是相同的:

delete from *table_name* where *condition*

下面命令在 Oracle 与 SQL Server 中都可以正确执行:

```
delete from emp where deptno=30
```

3.6 null 值的处理

对表添加记录时, 若某个列未指定值, 则称此列值为 null, 一般翻译为空。若一个表达式包含 null 值, 则其结果也为 null, 即不确定, 不确定的布尔值当做 FALSE。查询某个列为空或不为空的值, 不能使用 “=” 和 “<>” 这些普通的关系运算符, 而要使用 is null 或 is not null。

以上这些内容, Oracle 和 SQL Server 都是适用的, 下面介绍两种产品对待空值方面的一些区别。

3.6.1 null 值在排序(order by)中的处理

在 Oracle 中的查询中, 如果在 order by 子句中附加了 asc 选项, 则 null 值排在其他非空值之后, 如果附加 desc, 则 null 值排在其他非空值之前, 可以认为, Oracle 中的 null 值比非空值大, 如下面示例所示。

附加 asc 的情形:

```
SQL> select ename,comm
2  from emp
3  order by comm asc
4  /
```

ENAME	COMM
TURNER	0
ALLEN	300
WARD	500
MARTIN	1400
KING	
JAMES	
FORD	
CLARK	
MILLER	
JONES	
SMITH	

BLAKE

附加 desc 的情形:

```
SQL> select ename,comm
2  from emp
3  order by comm desc
4  /
```

ENAME	COMM
SMITH	
BLAKE	
FORD	
JAMES	
KING	
JONES	
MILLER	
CLARK	
MARTIN	1400
WARD	500
ALLEN	300
TURNER	0

这种默认形式可以通过在 order by 子句再附加 nulls last 而改变:

```
SQL> select ename,comm
2  from emp
3  order by comm desc nulls last
4  /
```

ENAME	COMM
MARTIN	1400
WARD	500
ALLEN	300
TURNER	0
KING	
JAMES	
FORD	
CLARK	
MILLER	
JONES	
SMITH	
BLAKE	

在 SQL Server 的查询中,如果 order by 附加了 asc 选项(这也是默认选项),即升序排列,则 null 值排在其他非空值之前,如果 order by 子句附加了 desc,则 null 值排在其他非空值之后,也可以认为 SQL Server 中 null 值最小,这与 Oracle 的处理方式正好相反,如下面示例所示。

附加 asc 的情形:

```
1> select ename,comm
2> from emp
3> order by comm asc
4> go
ename      comm
-----
SMITH      NULL
```

JONES	NULL
.....	
MILLER	NULL
TURNER	.00
ALLEN	300.00
WARD	500.00
MARTIN	1400.00

附加 desc 的情形：

```

1> select ename,comm
2> from emp
3> order by comm desc
4> go

```

ename	comm
-----	-----
MARTIN	1400.00
WARD	500.00
ALLEN	300.00
TURNER	.00
JAMES	NULL
FORD	NULL
MILLER	NULL
SMITH	NULL
JONES	NULL
BLAKE	NULL
CLARK	NULL
KING	NULL

3.6.2 null 处理函数

Oracle 的 null 处理函数包括 nvl()及 nvl2()。nvl()包含两个参数，nvl2()包含三个参数。

- nvl(expr1, expr2): 如果 expr1 为 null, 则返回 expr2, 否则返回 expr1。
- nvl2(expr1, expr2, expr3): 如果 expr1 非 null, 则返回 expr2, 否则返回 expr3。

如查询 emp 表中员工的每个月总收入(即 sal 与 comm 之和), 可分别使用这两个函数:

```

SQL> select ename, sal+nvl(comm, 0) income
2 from emp
3 /

```

ENAME	INCOME
-----	-----
SMITH	800
ALLEN	1900
WARD	1750
.....	

已选择 12 行。

```

SQL> select ename, nvl2(comm, sal+comm, sal) income
2 from emp
3 /

```

ENAME	INCOME
-----	-----
SMITH	800
ALLEN	1900
WARD	1750
.....	

对应 Oracle 的 `nvl()` 函数，SQL Server 提供的函数为 `isnull()`，其用法与 `nvl()` 相同：

`isnull(expr1, expr2)`：如果 `expr1` 为 `null`，则返回 `expr2`，否则返回 `expr1`。

与上节示例效果相同，下面示例返回 `emp` 表中每个员工的月总收入：

```
1> select ename, sal+isnull(comm, 0)
2> from emp
3> go
ename
-----
SMITH      800.00
ALLEN     1900.00
WARD      1750.00
JONES     2975.00
MARTIN    2650.00
.....
```

3.7 管理表

管理表包括创建、删除及修改表，Oracle 和 SQL Server 的语法基本相同。

修改表的结构主要包括修改列的数据类型，添加或删除列，添加或删除约束，修改表名称等。Oracle 完成上述任务使用 `alter table` 语句，SQL Server 使用 `sp_rename` 命令修改对象名称(包括修改表名)，其他任务也是使用 `alter table` 命令。

3.7.1 创建表

Oracle 和 SQL Server 创建表的基本语法是相同的，如下面语句两者都可以成功执行：

```
create table t
(
    a int,
    b int,
    primary key(a)
)
```

若指定存储表的表空间或文件组，则稍有区别。

Oracle 建表 `t` 并指定其存放的表空间为 `tbs`：

```
SQL> create table t
2 (
3     a int,
4     b int,
5     primary key(a)
6 )
7 tablespace tbs
8 /
```

SQL Server 建表 `t` 并指定其存放的文件组为 `fg`：

```
1> create table t
2> (
3>     a int,
4>     b int,
5>     primary key(a)
6> )
7> on fg
8> go
```

另外, Oracle 和 SQL Server 都可以拷贝现有表的结构以创建新表, Oracle 使用 `create table as` 语句, SQL Server 使用 `select into` 语句。

在 Oracle 中拷贝 emp 表以创建 emp_copy:

```
SQL> create table emp_copy
2 as
3 select * from emp
4 where 1=2
5 /
```

在 SQL Server 中拷贝 emp 表以创建 emp_copy:

```
1> select * into emp_copy
2> from emp
3> where 1=2
4> go
```

`where 1=2` 的结果为假, 目的是只拷贝表的结构, 而不拷贝表的数据, 若要连同数据一起拷贝, 可以去除 `where` 条件, 或添加合适的 `where` 条件, 只拷贝满足指定条件的数据。

3.7.1 修改列的数据类型

Oracle 和 SQL Server 分别使用 `modify` 与 `alter column` 关键字修改列的数据类型, 两者的语法如下:

- Oracle: `alter table table_name modify column_name datatype`
- SQL Server: `alter table table_name alter column column_name datatype`

SQL Server 采用的方式是 SQL:2003 的标准语法形式。

以修改 dept 表的 dname 列为 `varchar(20)` 为例, 分别说明 Oracle 与 SQL Server 的语法形式(Oracle 的 `varchar(20)` 表示为 `varchar2(20)`)。

Oracle 的情形:

```
SQL> alter table dept modify dname varchar2(20);
```

SQL Server 的情形:

```
1> alter table dept alter column dname varchar(20)
2> go
```

3.7.2 添加及删除列

添加和删除列的语法, Oracle 与 SQL Server 相同, 都符合 SQL:2003 标准。

添加及删除列分别使用下面语法:

- 添加列: `alter table table_name add column_name datatype`
- 删除列: `alter table table_name drop column column_name`

如对 dept 表添加列 phone_number, 数据类型为 `char(12)`:

```
alter table dept add phone_number char(12)
```

删除以上的 phone_number:

```
alter table dept drop column phone_number
```

3.7.3 修改列名

Oracle 使用 `alter table` 附加 `rename column` 子句来修改列名, SQL Server 用 `sp_rename` 系

统存储过程修改列名(及其他对象)。下面以修改 dept 表的 loc 列为 location 为例说明两者的用法。

Oracle 修改列名的语法形式为:

`alter table table_name rename column old_column_name to new_column_name`

```
SQL> alter table dept rename column loc to location;
```

SQL Server 使用 sp_rename 存储过程时, 要附加三个参数, 语法形式如下:

`sp_rename 'old_name', 'new_name', 'type'`

前两个参数分别是修改对象的旧名称及新名称, 第三个参数用于指定修改对象的类型, 可以为 database、object、column、index、userdatatype, 分别用于修改数据库名称、表或约束、列、索引、用户定义数据类型等名称。若修改列名, 则第三个参数为 column。

如修改 dept 表的 loc 列名称为 location:

```
1> sp_rename 'sch.dept.loc', 'location', 'column'
2> go
```

3.7.4 修改表名

Oracle 可以使用两种命令修改表的名称:

- `alter table table_name rename to new_name`
- `rename table_name to new_name`

如修改 dept 表名称为 department:

```
SQL> alter table dept rename to department;
```

使用 rename 命令再修改回来:

```
SQL> rename department to dept;
```

SQL Server 修改表名的方式与修改列名类似, 即使用 sp_rename 存储过程, 只是要把第三个参数指定为 object, 如修改 dept 表为 department:

```
1> sp_rename 'sch.dept', 'department', 'object'
2> go
```

3.7.5 删除表

删除表使用 drop table 语句, 如删除 t 表:

```
drop table t
```

Oracle 有回收站功能, 表被删除后, 其数据并未真正删除, 而是移入了回收站, 以后可以执行 flashback table 再恢复回来, 其目的与操作系统的回收站功能类似, 都是为了防止数据被误删除。

下面实验说明如何恢复被删除的表。

删除表之前, scott 模式的表的情况如下:

```
SQL> conn scott/tiger
已连接。
SQL> select * from tab;
```

TNAME	TABTYPE	CLUSTERID
SALGRADE	TABLE	

BONUS	TABLE
DEPT	TABLE
EMP	TABLE

删除其中的 emp 表：

```
SQL> drop table emp;
```

表已删除。

然后查询 scott 模式下的表的情况：

```
SQL> select * from tab;
```

TNAME	TABTYPE	CLUSTERID
BIN\$ZAlfuzhUTgOqqC+rNFJoSg==\$0	TABLE	
SALGRADE	TABLE	
BONUS	TABLE	
DEPT	TABLE	

可以发现，第一行的表名类似乱码，并且 emp 表不见了。

查询数据字典视图 user_recyclebin，可以看到 emp 表及其索引被移入了回收站：

```
SQL> select object_name, original_name from user_recyclebin;
```

OBJECT_NAME	ORIGINAL_NAME
BIN\$Jrs0/pbNSx2md00/VzICPw==\$0	PK_EMP
BIN\$ZAlfuzhUTgOqqC+rNFJoSg==\$0	EMP

执行下面 flashback 命令恢复 emp 表：

```
SQL> flashback table emp to before drop;
```

闪回完成。

回收站内的被删除对象已不存在，emp 表也被恢复了：

```
SQL> select object_name, original_name from user_recyclebin;
```

未选定行

```
SQL> select * from tab;
```

TNAME	TABTYPE	CLUSTERID
SALGRADE	TABLE	
BONUS	TABLE	
DEPT	TABLE	
EMP	TABLE	

如果要关闭回收站功能，可以设置初始化参数 recyclebin 为 off，这是一个静态初始化参数，修改时要附加 scope=spfile 子句，重启数据库使其生效：

```
SQL> alter system set recyclebin=off scope=spfile;
```

系统已更改。

3.9 关于 identity 列

列的 identity 属性用于实现添加记录时的列值自增，一般用于没有明确意义的主键列。SQL Server 各版本一直支持这个功能，Oracle 从 12c 版本开始支持。

在 Oracle 中，创建包含 identify 列的表，默认初始值和递增步长均为 1：

```
SQL> create table t(a int generated as identity, b int);  
表已创建。
```

添加记录时，可以忽略 identity 列：

```
SQL> insert into t(b) values(10);  
已创建 1 行。  
SQL> insert into t(b) values(20);  
已创建 1 行。  
SQL> insert into t(b) values(30);  
已创建 1 行。
```

下面命令可以验证 a 列是如何自增的：

```
SQL> select * from t;
```

A	B
1	10
2	20
3	30

可以附加 start with 和 increment by 子句手工指定初始值和步长，如下面命令指定初始值为 10，递增步长为 2：

```
SQL> create table tt(a int generated as identity(start with 10 increment by 2), b int);
```

SQL Server 创建表时，在列后附加 identify 关键字使列具备 identify 属性，默认初始值和递增步长均为 1，语法形式相比 Oracle 较为简单，如下面命令所示：

```
1> create table t(a int identity, b int)  
2> go
```

也可以如下面方式，直接在括号内指定初值和步长：

```
1> create table tt(a int identity(10, 2), b int)  
2> go
```

3.10 关于约束

Oracle 与 SQL Server 都支持以下类型的约束：

- 非空(not null)
- 唯一(unique)
- 检查(check)
- 主键(primary key)
- 外键(foreign key)
- 默认(default)

除外键约束的语法稍有区别外，其他几种约束的附加方式是相同的。如下面命令可以在 Oracle 和 SQL Server 成功执行：

```
create table t  
(  
    a int,  
    b int,
```

```

constraint pk_t primary key(a),
constraint ck_t check(b>100),
constraint uq_t unique(b)
)

```

Oracle 附加外键约束时,可以使用下面子句指定当主表被子表外键引用的记录被删除时,对子表的处理方式:

- on delete set null: 指定子表外键的对应值设置为空。
- on delete cascade: 级联删除子表中对应的记录。

下面示例,在附加外键约束时,附加了 on delete cascade 子句:

```

SQL> alter table emp add constraint fk_deptno
2 foreign key(deptno) references dept(deptno)
3 on delete cascade
4 /

```

如果未指定外键约束的附属子句,则 Oracle 默认不允许删除或更新主表中被子表引用的记录,相当于 SQL Server 中的 on delete no action 或 on update no action 子句的效果。

对于外键约束,SQL Server 可以指定更多的选项:

- on delete no action 或 on update no action
- on delete cascade 或 on update cascade
- on delete set null 或 on update set null
- on delete set default 或 on update set default
- no action: 删除或更新主表被引用的记录时报错,回滚删除操作,是默认选项。
- cascade: 级联删除或更新子表中引用的记录。
- set null: 把子表中引用的外键值设置为空。
- set default: 把子表外键值设置为默认值,在外键列上需要预先设置 default 约束,如果未附加 default 约束,则外键的对应值会设置为空。

如下面示例在主表删除子表引用到的记录时,把子表外键的对应值设置为空,当更新主表的 deptno 列值时,级联更新子表外键的对应值:

```

1> alter table emp
2> add constraint fk_emp foreign key(deptno) references dept(deptno)
3> on delete set null
4> on update cascade
5> go

```

3.11 视图

视图是由查询的结果集构成的虚拟表,虚拟的意思是视图不保存数据,表的意思是视图可以像表一样进行操作,数据库中对视图存储的信息只有名称及其对应的查询命令。

SQL Server 视图的查询语句不能包含 order by 子句(除非指定列 top、offset),Oracle 无此限制。

下面命令创建视图 v_emp:

```

create view v_emp
as
select ename, sal from emp

```

Oracle 支持实体化视图(也称为物化视图)以提高查询效率,实体化视图不是虚拟表,保存了视图定义中的查询结果集,SQL Server 中的类似对象是索引视图,从 SQL Server 2000

开始引入。实体化视图和索引视图的数据都会自动与基表同步。

在 Oracle 中创建实体化视图：

```
SQL> create materialized view mv_emp
2 refresh on commit
3 as
4 select ename, sal from emp
5 /
```

refresh on commit 子句指定当表中的数据修改提交后，刷新物化视图使其与表同步。

下面说明如何在 SQL Server 中创建索引视图，这里的索引必须是唯一聚集索引，索引视图的意思是添加了唯一聚集索引的视图。

附加 with schemabinding 子句(作用是不能修改视图基表的结构)，创建视图 v_emp：

```
1> set quoted_identifier on
2> go
1> create view v_emp
2> with schemabinding
3> as
4> select ename, sal from dbo.emp
5> go
```

创建索引视图时，要先开启 quoted_identifier 参数，另外要注意查询语句的表名之前要附加架构名称。创建索引视图的其他注意事项，请参考联机丛书。

然后创建唯一聚集索引 idx_v_emp：

```
1> create unique clustered index idx_v_emp
2> on v_emp(ename, sal)
3> go
```

索引视图把索引定义中的查询结果集保存在磁盘上，占用空间比表上的聚集索引要少，可以使查询效率提高更显著。

3.12 序列

序列的目的是产生满足指定条件的常数。

Oracle 和 SQL Server 使用 create sequence 命令创建序列，下面命令可以在 Oracle 和 SQL Server 成功执行。

```
create sequence seq
start with 1
increment by 1
```

start with 子句指定序列的初始值，increment by 子句指定递增步长。

Oracle 使用序列对象的伪列 nextval 使用序列产生的常数。

下面示例先创建 t 表，然后对其添加三行记录，使用序列 seq 对其主键列指定值：

```
SQL> create table t(a int primary key, b char(10));
表已创建。
SQL> insert into t values(seq.nextval, 'a');
已创建 1 行。
SQL> insert into t values(seq.nextval, 'b');
已创建 1 行。
SQL> insert into t values(seq.nextval, 'c');
已创建 1 行。
```

SQL Server 使用 next value for 函数在序列中产生常数。

下面示例先创建 t 表，然后对其添加三行记录，使用序列 seq 对其主键列指定值：

```
1> create table t(a int primary key, b char(10))
2> go
1> insert into t values(next value for seq, 'a')
2> insert into t values(next value for seq, 'b')
3> insert into t values(next value for seq, 'c')
4> go
```

修改序列的属性，Oracle 和 SQL Server 使用 alter sequence 命令：

```
alter sequence seq increment by 10;
```

Oracle 使用 dba_sequences 查询序列属性：

```
SQL> select sequence_name, min_value, increment_by
  2   from dba_sequences
  3   where sequence_name='SEQ'
  4   /
```

SEQUENCE_NAME	MIN_VALUE	INCREMENT_BY
SEQ	1	10

SQL Server 使用 sys.sequences 查询序列属性：

```
1> select start_value, increment
2> from sys.sequences
3> where name='seq'
4> go
start_value    increment
-----
1              10
```

删除序列，Oracle 和 SQL Server 使用 drop sequence 命令：

3.13 同义词

同义词用来简化数据库对象名称。Oracle 的同义词分为私有和公共两类，私有即只对其模式所对应的用户有效，公共对所有用户均有效。当公共同义词与用户模式内的对象重名时，公共同义词失效。相对 Oracle，SQL Server 只有公共同义词。

Oracle 中创建私有同义词：

```
SQL> create synonym emp for scott.emp;
```

Oracle 中创建公共同义词：

```
SQL> create public synonym emp for scott.emp;
```

下面是 SQL Server 创建同义词：

```
1> create synonym syn_emp for sch.emp
2> go
```

Oracle 使用数据字典视图 dba_synonyms 查询同义词对应的对象：

```
SQL> select synonym_name, table_owner, table_name
```

```

2  from dba_synonyms
3  where synonym_name='EMP'
4  /

```

SYNONYM_NAME	TABLE_OWNE	TABLE_NAME
EMP	SCOTT	EMP

SQL Server 使用目录视图 sys.synonyms 查询同义词对应的对象:

```

1> select name,base_object_name from sys.synonyms
2> where name='syn_emp'
3> go
name                base_object_name
-----
syn_emp             [sch].[emp]

```

在 Oracle 中修改同义词的定义使用 create or replace 命令，无则新建，有则替换，不能使用 alter 命令:

```
SQL> create or replace synonym emp for scott.dept;
```

SQL Server 不能修改同义词定义，只能通过删除后重建间接实现。

删除同义词的命令相同，下面命令可以在 Oracle 和 SQL Server 成功执行:

```
drop synonym emp;
```