

8

操纵数据

中国科学院西安网络中心 编译 2005

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

进度表:	时间	主题
	60 minutes	讲演
	30 minutes	练习
	90 minutes	总共

目标

完成本课后，您应当能够执行下列操作：

- 描述每个 DML 语句
- 插入行到表中
- 更新表中的行
- 从表中删除行
- 合并表中的行
- 控制事务

课程目标

在本课中，你将学习怎样插入行到表中，更新表中的行，从表中删除行。你也将学习怎样用 COMMIT、SAVEPOINT 和 ROLLBACK 语句来控制事务。

数据操纵语言

- 当你做下面操作时，DML 语句被执行：
 - 添加新行到表中
 - 修改表中的行
 - 删除表中的行
- 事务 由 DML 语句的集合组成，它组成工作的逻辑单元

数据操纵语言

数据操纵语言 (Data manipulation language DML) 是 SQL 的一个核心部分。当你想要添加、更新或者删除数据库中的数据时，你在执行 DML 语句。DML 依据的一个集合构成了一个被称为事务的逻辑单元。

考虑一个银行数据库，当一个银行客户从一个储蓄存款帐户转帐到一个经常帐户时，事务可以由三个单独的操作组成：减少存款帐户，增加经常帐户，在交易帐中记录交易。Oracle 服务器必须保证所有这三个 SQL 语句被执行以维护帐目余额的正确。当由于某种原因阻碍了交易中一条语句的执行，那么其它的交易语句都必须被撤消。

教师注释

DML 语句能够直接在 iSQL*Plus 中发布，由类似 Oracle Forms Services 这样的工具自动执行，或者用类似 3GL 预编译程序编程。

每一个表都有其关联的 INSERT、UPDATE 和 DELETE 权限，这些权限被自动地授予表的创建者，但在通常情况下这些权限必须被明确地授予其它用户。

从 Oracle 7.2 开始，你能够用一个子查询代替 UPDATE 语句中的表名，本质上和视图的使用方式一样。

添加一个新行到表中

70	Public Relations	100	1700
----	------------------	-----	------

新行

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

...插入
一个新行到
DEPARTMENTS
表中...

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700
70	Public Relations	100	1700

中国科学院西安网络中心 编译 2005

ORACLE

8-4

Copyright © Oracle Corporation, 2001. All rights reserved.

添加一个新行到表中

幻灯片中的图解举例说明了添加一个新的部门到 DEPARTMENTS 中的情形。

INSERT 语句语法

- 使用 **INSERT** 语句添加新行到表中

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

- 用该语法一次只能插入一行

添加一个新行到表中 (续)

你可以用 **INSERT** 语句添加新行到表中。

在语法中：

<i>table</i>	是表的名字
<i>column</i>	是表中的列名
<i>value</i>	是列的相应值

注：该语句用 **VALUES** 子句添加行到表中，一次仅一行。

插入新行

- 插入一个包含每一个列值的新行
- 值以表中列的默认顺序列表
- 在 **INSERT** 子句中字段可以随意列表

```
INSERT INTO departments(department_id, department_name,  
                        manager_id, location_id)  
VALUES      (70, 'Public Relations', 100, 1700);  
1 row created.
```

- 字符和日期要用单引号括起来

添加一个新行到表中 (续)

因为你能够插入一个包含每个列值的新行，因此在 **INSERT** 子句中字段列表不是必须的，可是，如果你不用字段列表，值必须按照表中字段的默认顺序排列，并且必须为每个列提供一个值。

```
DESCRIBE departments
```

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

为使语句更清楚，在 **INSERT** 子句中使用字段列表。

字符和日期值应放在单引号中；数字值不需要。

数字值不应放在单引号中，因为对于指定为 **NUMBER** 数据类型的字段，如果使用了单引号，可能会发生数字值的隐式转换。

插入带空值的行

- 隐式方法：省略字段列表中的列

```
INSERT INTO departments (department_id,  
                           department_name )  
VALUES      (30, 'Purchasing');  
1 row created.
```

- 显式方法：在 VALUES 子句中指定 NULL 关键字

```
INSERT INTO departments  
VALUES      (100, 'Finance', NULL, NULL);  
1 row created.
```

插入空值的方法

方法	说明
隐式 (Implicit)	从字段列表中省略列。
显式 (Explicit)	在 VALUES 列表中指定 NULL 关键字，对于字符串和日期，在在 VALUES 列表中指定空字符串 (' ')

用 iSQL*Plus DESCRIBE 命令核实目的列是否能够接受空值。
Oracle 服务器自动强制所有数据类型\数据范围和数据完整性约束。在插入新行时，任何没有被明确列出的字段都将获得一个空值。

在用户输入时经常出现的错误：

- 对于 NOT NULL 列缺少强制的值
- 重复值违反了唯一性约束
- 违反外键约束
- 违反 CHECK 约束
- 数据类型不匹配
- 值的宽度超过了列的限制

插入特殊值

SYSDATE 函数报告当前的日期和时间

```
INSERT INTO employees (employee_id,
                        first_name, last_name,
                        email, phone_number,
                        hire_date, job_id, salary,
                        commission_pct, manager_id,
                        department_id)
VALUES (113,
        'Louis', 'Popp',
        'LPOPP', '515.124.4567',
        SYSDATE, 'AC_ACCOUNT', 6900,
        NULL, 205, 100);

1 row created.
```

用 SQL 函数插入特殊值

你可以使用函数输入特殊值到表中。

幻灯片的例子录入雇员 Popp 的信息到 EMPLOYEES 表中，在 INSERT 语句中提供当前日期时间到 HIRE_DATE 列中，用 SYSDATE 函数得到当前日期时间。

在插入行到表中时，你也可以用 USER 函数，USER 函数记录当前用户名。

确认添加到表的信息

```
SELECT employee_id, last_name, job_id, hire_date, commission_pct
FROM   employees
WHERE  employee_id = 113;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	COMMISSION_PCT
113	Popp	AC_ACCOUNT	12-MAR-01	

插入特殊日期值

- 添加一个新雇员

```
INSERT INTO employees
VALUES      (114,
             'Den', 'Raphealy',
             'DRAPHEAL', '515.127.4561',
             TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
             'AC_ACCOUNT', 11000, NULL, 100, 30);
1 row created.
```

- 校验新添加的雇员

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_P
114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-99	AC_ACCOUNT	11000	

中国科学院西安网络中心 编译 2005

ORACLE

8-9

Copyright © Oracle Corporation, 2001. All rights reserved.

插入特殊日期和时间值

DD-MON-YY 格式通常用于插入日期值，用该格式，用默认世纪作为当前世纪。因为日期也包含时间信息，默认时间午夜 (00:00:00)。

如果日期必须以某种格式输入，而不是默认格式，例如，用另一个世纪，或一个特殊的时间，你必须使用 TO_DATE 函数。

幻灯片中的例子记录了雇员 Raphealy 的信息到 EMPLOYEES 表中，其中设置 HIRE_DATE 列为 February 3, 1999，如果你用下面的语句代替幻灯片中的例子插入，hire_date 中的年将被解释为 2099。

```
INSERT INTO employees
VALUES      (114,
             'Den', 'Raphealy',
             'DRAPHEAL', '515.127.4561',
             '03-FEB-99',
             'AC_ACCOUNT', 11000, NULL, 100, 30);
```

如果用 RR 格式，系统自动提供当前世纪，即使日期不是当前世纪。

教师注释

在 Oracle9i 中默认日期格式是 DD-MON-RR，在 8.16 版以前，默认格式是 DD-MON-YY。

使用替换变量

- 在 SQL 语句中用 & 替换变量提示用户输入值
- & 是一个用于变量值的占位符

```
INSERT INTO departments
      (department_id, department_name, location_id)
VALUES  (&department_id, '&department_name', &location);
```

Define Substitution Variables

"department_id"

"department_name"

"location"

Submit for Execution

Cancel

1 row created.

中国科学院西安网络中心 编译 2005

ORACLE

8-10

Copyright © Oracle Corporation, 2001. All rights reserved.

创建一个脚本来操纵数据

你可以用替换变量保存命令到一个文件中，并且可以执行文件中的命令。上面的例子记录了 DEPARTMENTS 表中一个部门的信息。

运行脚本文件，你会被提示输入 & 替换变量，然后你输入的值被替换到语句中。使用替换变量允许你反复运行相同的脚本文件，但每次可以提供不同的值。

教师注释

关于脚本对学生提醒下面几点：

在 DEFINE 命令中，不要前置 iSQL*Plus 带 & 符号的替换参数。

用一个破折号 (-) 在下一行继续一个 iSQL*Plus 命令。

从另一个表中复制行

- 用一个子查询写 **INSERT** 语句

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM employees
WHERE job_id LIKE '%REP%';

4 rows created.
```

- 不用 **VALUES** 子句
- 在子查询中列的数目要匹配 **INSERT** 子句中列的数目

从另一个表中复制行

你可以用 **INSERT** 语句添加行到表中，插入的值来自已存在的表，在 **VALUES** 子句的位置用一个子查询。

语法

```
INSERT INTO table [ column (, column) ] subquery;
```

在语法中：

<i>table</i>	是表名
<i>column</i>	是表中的列名
<i>subquery</i>	是返回行的子查询

在 **INSERT** 子句的字段列表中列的数目和它们的数据类型必须与子查询中的值的数目及其数据类型相匹配。为了创建一个表的行的拷贝，在子查询中用 **SELECT ***。

```
INSERT INTO copy_emp
SELECT *
FROM employees;
```

更多信息，见 *Oracle9i SQL Reference*，“**SELECT**”子查询部分。

教师注释


在验证代码例子之前，请运行脚本 8_cretabs.sql 来创建 **COPY_EMP** 和 **SALES_REPS** 表。从另一个表拷贝行时不要获取过多的详细信息。

改变表中的数据

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSION_F
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	60	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	60	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	60	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

更新 EMPLOYEES 表中的行



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSIO
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	30	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	30	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	30	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

改变表中的数据

幻灯片的图解举例说明了雇员表中部门号的改变，将部门 60 中的雇员改为部门 30 中的雇员。

UPDATE 语句的语法

- 用 UPDATE 语句修改已存在的行

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

- 如果需要，一次更新多行

更新行

你可以用 UPDATE 语句修改已经存在的行。

在语法中：

table 是表的名字

column 是表中列的名字

value 是相应的值或对应列的子查询

condition 确定要被更新的行，由列名、表达式、常数和比较操作符组成
用查询表来显示受更新的行以确认更新操作。

更多信息，见 *Oracle9i SQL Reference*，“UPDATE”。

注：通常，用主键标识一个单个的行，如果用其他列，可能会出乎意料的引起另一些行被更新。例如，在 EMPLOYEES 表中用名字标识一个单个的行是危险的，因为不同的雇员可能有相同的名字。

更新表中的行

- 如果使用了 **WHERE** 子句，指定的一行或多行将被修改

```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
1 row updated.
```

- 如果遗漏 **WHERE** 子句，表中所有的行都会被修改

```
UPDATE    copy_emp
SET       department_id = 110;
22 rows updated.
```

更新行 (续)

如果 **WHERE** 子句被指定，**UPDATE** 语句将修改指定的行。幻灯片的例子中，将雇员 113 (Popp) 转到部门 70。

如果你遗漏了 **WHERE** 子句，表中的所有都会被修改。

```
SELECT last_name, department_id
FROM    copy_emp;
```

LAST_NAME	DEPARTMENT_ID
King	110
Kochhar	110
De Haan	110
Hunold	110
Ernst	110
Lorentz	110
Mourgos	110
Gietz	110

22 rows selected.

注：COPY_EMP 表与 EMPLOYEES 表有相同的数据。

用子查询更新两列

更新雇员 114 的工作和薪水，使其和雇员 205 相同

```
UPDATE employees
SET   job_id = (SELECT job_id
                FROM   employees
                WHERE  employee_id = 205),
      salary = (SELECT salary
                FROM   employees
                WHERE  employee_id = 205)
WHERE employee_id = 114;
1 row updated.
```

用子查询更新两列

你可以在 UPDATE 语句的 SET 子句中写多个子查询来更新多列。

语法

```
UPDATE table
SET   column =
      (SELECT column
       FROM table
       WHERE condition)
[ ,
  column =
      (SELECT column
       FROM table
       WHERE condition)]
[WHERE condition];
```

注：如果没有行被更新，一个“0 rows updated.”消息被返回。

更新基于另一个表的行

在 **UPDATE** 语句中用子查询来更新基于另一个表中值的那些行

```
UPDATE copy_emp
SET    department_id = (SELECT department_id
                        FROM employees
                        WHERE employee_id = 100)
WHERE  job_id        = (SELECT job_id
                        FROM employees
                        WHERE employee_id = 200);

1 row updated.
```

更新基于另一个表的行

你可以在 **UPDATE** 语句中用子查询来更新表中的多行。在幻灯片的例子中用基于来自 **EMPLOYEES** 表的值更新 **COPY_EMP** 表，它用 employee 100 的部门号；改变所有工作岗位与 employee 200 的工作岗位相同的那些雇员的部门号。

更新行：完整性约束错误

```
UPDATE employees  
SET    department_id = 55  
WHERE  department_id = 110;
```

```
UPDATE employees  
*  
ERROR at line 1:  
ORA-02291: integrity constraint (HR.EMP_DEPT_FK)  
violated - parent key not found
```

部门号 55 不存在

完整性约束错误

如果你企图用有完整性约束依赖的值更新记录，会出现错误。

在幻灯片的例子中，部门号 55 在父表 DEPARTMENTS 中不存在，所以，你会收到违反父键约束错误 ORA-02291。

注：完整性约束确保数据遵守预先设定的规则集。后面的课程将更深入地讲述数据的完整性约束。

教师注释

解释数据完整性约束，并且回顾主键和外键的概念。

从表中删除行

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
100	Finance		
50	Shipping	124	1500
60	IT	103	1400

从 DEPARTMENTS 表中删除一行

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400

从表中删除行

幻灯片图解了从 DEPARTMENTS 表中删除 Finance 部门。(假定在 DEPARTMENTS 表上没有定义约束)。

教师注释

在所有的行用 DELETE 语句删除后，只有表的数据结构被保留，清空表的另一种更有效的方法是用 TRUNCATE 语句。

你可以用 TRUNCATE 语句快速删除表中所有的行，用 TRUNCATE 语句删除行比用 DELETE 语句更快一些，原因如下：

TRUNCATE 语句是数据定义语言 (statement is a data definition language DDL) 语句，不产生回退信息，TRUNCATE 语句在子查询课程中讲述。

截断表不触发表或删除触发器。

如果表是引用完整性约束的父表，你不能截断该表，在发布 TRUNCATE 语句之前先禁用约束。

DELETE 语句

使用 **DELETE** 语句从表中删除已存在的行

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

删除行

你可以用 **DELETE** 语句删除已经存在的行。

在语法中：

table 是表名

condition 标识被删除的行，由字段名、表达式、常数和比较操作符组成

注：如果没有行被删除，消息 “0 rows deleted.” 被返回。

更多信息，见 *Oracle9i SQL Reference*，“DELETE”。

教师注释

DELETE 语句执行前没有确认提示，但是，删除操作直到数据处理事务被提交之前不会被持久化，因此，如果你做错了，可以用 **ROLLBACK** 语句撤消删除操作。

从表中删除行

- 如果指定了 **WHERE** 子句，则指定的行被删除

```
DELETE FROM departments
WHERE department_name = 'Finance';
1 row deleted.
```

- 如果遗漏了 **WHERE** 子句，表中所有的行都被删除

```
DELETE FROM copy_emp;
22 rows deleted.
```

删除行 (续)

在 **DELETE** 语句中你可以用 **WHERE** 子句删除指定的行。幻灯片的例子从 **DEPARTMENTS** 表中删除 **Finance** (财务) 部门。你可以用 **SELECT** 语句显示被删除的行以确认删除操作。

```
SELECT *
FROM departments
WHERE department_name = 'Finance';
no rows selected.
```

如果你遗漏了 **WHERE** 子句，在表中的所有行都被删除。幻灯片中的第二个例子从 **COPY_EMP** 表中删除所有的行，因为没有指定 **WHERE** 子句。

例

删除在 **WHERE** 子句中指定的行。

```
DELETE FROM employees
WHERE employee_id = 114;
1 row deleted.
```

```
DELETE FROM departments
WHERE department_id IN (30, 40);
2 rows deleted.
```

删除基于另一个表的行

在 **DELETE** 语句中用子查询来删除表中的基于另一个表中值的行

```
DELETE FROM employees
WHERE department_id =
    (SELECT department_id
     FROM departments
     WHERE department_name LIKE '%Public%');

1 row deleted.
```

删除基于另一个表的行

你可以用子查询从表中删除基于另一个表的值的行。幻灯片中的例子删除所有雇员，这些雇员所在的部门是部门名字中包含字符串“Public”的那些部门。子查询查找 DEPARTMENTS 表以找出基于部门名中包含字符串“Public”的部门的部门号。子查询然后将部门号馈入主查询，主查询基于该部门号从 EMPLOYEES 删除数据行。

删除行：完整性约束错误

```
DELETE FROM departments
WHERE      department_id = 60;
```

```
DELETE FROM departments
          *
ERROR at line 1:
ORA-02292: integrity constraint (HR.EMP_DEPT_FK)
violated - child record found
```

不能删除包含主键的行，该主键被用做另一个表的外键

完整性约束错误

如果你尝试删除一条记录，该记录中的值依赖一个完整性约束，一个错误被返回。幻灯片中的例子试图从 DEPARTMENTS 表中删除部门号 60，但执行该语句将返回一个错误，因为部门号在 EMPLOYEES 表中被用做外键。如果你试图删除的父记录有子记录，那么，你将收到 *child record found violation* ORA-02292。

下面的语句可以正常工作，因为在部门 70 中没有雇员：

```
DELETE FROM departments
WHERE      department_id = 70;
```

```
1 row deleted.
```

教师注释

如果使用了引用完整性约束，当你试图删除一行时，你可能收到一个 Oracle 服务器错误信息。但是，如果引用完整性约束包含了 ON DELETE CASCADE 选项，那么，可以删除行，并且所有相关的子表记录都被删除。

在 INSERT 语句中使用子查询

```
INSERT INTO
    (SELECT employee_id, last_name,
            email, hire_date, job_id, salary,
            department_id
     FROM employees
     WHERE department_id = 50)
VALUES (99999, 'Taylor', 'DTAYLOR',
        TO_DATE('07-JUN-99', 'DD-MON-RR'),
        'ST_CLERK', 5000, 50);

1 row created.
```

在 INSERT 语句中使用子查询

你可以在 INSERT 语句的 INTO 子句中用一个子查询代替表名，该子查询的选择列表必须与 VALUES 子句的字段列表有相同的字段数，基表的列上的所有规则必须遵循 INSERT 语句的顺序。例如，你不可能插入一个重复的 employee_id，也不能遗漏强制为非空列的值。

子查询用于为 INSERT 语句标识所要操作的表。

在 INSERT 语句中使用子查询

```
SELECT employee_id, last_name, email, hire_date,  
       job_id, salary, department_id  
FROM   employees  
WHERE  department_id = 50;
```

EMPLOYEE_ID	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
124	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50
141	Rajs	TRAJS	17-OCT-95	ST_CLERK	3500	50
142	Davies	CDAVIES	29-JAN-97	ST_CLERK	3100	50
143	Matos	RMATOS	15-MAR-98	ST_CLERK	2600	50
144	Vargas	PVARGAS	09-JUL-98	ST_CLERK	2500	50
99999	Taylor	DTAYLOR	07-JUN-99	ST_CLERK	5000	50

6 rows selected.

在 INSERT 语句中使用子查询

上面的例子显示了在 INSERT 语句中使用子查询的结果。

在 DML 语句中使用 WITH CHECK OPTION 关键字

- 一个子查询被用于确定表和 DML 语句的列
- **WITH CHECK OPTION** 关键字禁止改变那些不在子查询中的行

```
INSERT INTO (SELECT employee_id, last_name, email,  
                hire_date, job_id, salary  
            FROM employees  
            WHERE department_id = 50 WITH CHECK OPTION)  
VALUES (99998, 'Smith', 'JSMITH',  
        TO_DATE('07-JUN-99', 'DD-MON-RR'),  
        'ST_CLERK', 5000);  
INSERT INTO  
      *  
ERROR at line 1:  
ORA-01402: view WITH CHECK OPTION where-clause violation
```

WITH CHECK OPTION 关键字

指定 WITH CHECK OPTION 预示，如果子查询被用于在 INSERT、UPDATE 或 DELETE 语句中代替表，那些没有包括在子查询中提供给该表的行将不会改变。

如例子中显示，使用了 WITH CHECK OPTION 关键字，子查询确定部门号为 50 的行，但部门号不在 SELECT 列表中，并且在 VALUES 列表中也并没有提供相应的值。插入该行将导致一个空的部门号，该行不在子查询中。

显式默认特性概览

- 用显式默认特性，你能够用 **DEFAULT** 关键字作为一个列值，该列要求一个默认值
- 该附加特性是与 SQL: 1999 标准兼容的
- 该特性允许用户控制在什么地方和什么时候默认值应该被提供给数据
- 显示默认值能够被用于 **INSERT** 和 **UPDATE** 语句

显式默认

DEFAULT 关键字可以被用于 **INSERT** 和 **UPDATE** 语句来确定默认的列值。如果不存在默认值，将使用空值。

使用显示默认值

- 在 INSERT 中的 DEFAULT:

```
INSERT INTO departments  
  (department_id, department_name, manager_id)  
VALUES (300, 'Engineering', DEFAULT);
```

- 在 UPDATE 中的 DEFAULT:

```
UPDATE departments  
SET manager_id = DEFAULT WHERE department_id = 10;
```

使用确切的默认值

指定 DEFAULT 用预先指定的值作为该列的默认值。如果相应的列无指定的默认值，Oracle 设置该列为空。

第一个例子显示，INSERT 语句的 VALUES 子句给 MANAGER_ID 字段一个默认值，如果对该列无默认值定义，作为代替将插入一个空值。

第二个例子中，UPDATE 语句的 SET 子句赋给部门 10 的 MANAGER_ID 列一个默认值，如果对该列无默认值定义，该值将变为空。

注：在创建表时，你可以为一个字段指定默认值。默认值将在课程“创建和管理表”中讨论。

MERGE 语句

- 提供有条件地更新和插入数据到数据库表中的能力
- 如果行存在，执行 **UPDATE**；如果是一个新行，执行 **INSERT**：
 - 避免分散更新
 - 增进性能和易用性
 - 在数据仓库应用中有用

合并语句

SQL 的扩展包括了 **MERGE** 语句，使用该语句，你可以有条件地更新或插入行到表中，这样能避免多重 **UPDATE** 语句。是执行对目的表的更新操作还是执行对目的表的插入操作，取决于基于 **ON** 子句中的条件。

由于 **MERGE** 命令组合了 **INSERT** 和 **UPDATE** 命令，你需要有对目的表的 **INSERT** 和 **UPDATE** 权限，以及对源表的 **SELECT** 权限。

MERGE 语句确定性的。在同一个 **MERGE** 语句中，你不能多次更新目的表中相同的行。

一个可供选择近似方法是用 **PL/SQL** 循环和多重 **DML** 语句，然而，**MERGE** 语句易于使用，并且作为一条单个的 **SQL** 语句会更简单。

MERGE 语句在许多数据仓库应用中是适用的。例如，在一个数据仓库应用程序中，你可能需要来自多个源的数据工作，其中的一些可能是完全相同的。用 **MERGE** 语句，你可以有条件地添加或修改行。

MERGE 语句的语法

你能够用 **MERGE** 语句有条件地插入或更新表中的行

```
MERGE INTO table_name table_alias
  USING (table/view/sub_query) alias
  ON (join condition)
  WHEN MATCHED THEN
    UPDATE SET
      col1 = col_val1,
      col2 = col2_val
  WHEN NOT MATCHED THEN
    INSERT (column_list)
    VALUES (column_values);
```

合并行

用 **MERGE** 语句，你可以更新已经存在的行，并且有条件地插入新行。

在语法中：

INTO 子句	指定你正在更新或插入的目的表
USING 子句	指定数据源要被更新或插入的数据的源；可以是一个表、视图或者子查询
ON 子句	是一个条件，在此条件上 MERGE 操作即可以更新也可以插入
WHEN MATCHED WHEN NOT MATCHED	通知服务器怎样响应连接条件的结果

更多信息，见 *Oracle9i SQL 参考*, “MERGE.”

合并行

插入或更新在 **COPY_EMP** 表中的行以匹配 **EMPLOYEES** 表

```

MERGE INTO copy_emp c
  USING employees e
    ON (c.employee_id = e.employee_id)
  WHEN MATCHED THEN
    UPDATE SET
      c.first_name      = e.first_name,
      c.last_name       = e.last_name,
      ...
      c.department_id  = e.department_id
  WHEN NOT MATCHED THEN
    INSERT VALUES(e.employee_id, e.first_name, e.last_name,
                  e.email, e.phone_number, e.hire_date, e.job_id,
                  e.salary, e.commission_pct, e.manager_id,
                  e.department_id);

```

中国科学院西安网络中心 编译 2005

ORACLE

8-30

Copyright © Oracle Corporation, 2001. All rights reserved.

合并行的例子

```

MERGE INTO copy_emp c
  USING employees e
    ON (c.employee_id = e.employee_id)
  WHEN MATCHED THEN
    UPDATE SET
      c.first_name      = e.first_name,
      c.last_name       = e.last_name,
      c.email           = e.email,
      c.phone_number    = e.phone_number,
      c.hire_date       = e.hire_date,
      c.job_id          = e.job_id,
      c.salary          = e.salary,
      c.commission_pct  = e.commission_pct,
      c.manager_id      = e.manager_id,
      c.department_id   = e.department_id
  WHEN NOT MATCHED THEN
    INSERT VALUES(e.employee_id, e.first_name, e.last_name,
                  e.email, e.phone_number, e.hire_date, e.job_id,
                  e.salary, e.commission_pct, e.manager_id,
                  e.department_id);

```

该例子显示匹配 **COPY_EMP** 表中的 **EMPLOYEE_ID** 列与 **EMPLOYEES** 表中的 **EMPLOYEE_ID** 列。如果找到了一个匹配，用 **EMPLOYEES** 表中匹配行的列值更新 **COPY_EMP** 表中匹配的列值。如果相匹配行没有找到，**EMPLOYEES** 表中的列值被插入到 **COPY_EMP** 表中。

合并行

```
SELECT *  
FROM COPY_EMP;  
  
no rows selected
```

```
MERGE INTO copy_emp c  
  USING employees e  
  ON (c.employee_id = e.employee_id)  
WHEN MATCHED THEN  
  UPDATE SET  
    ...  
WHEN NOT MATCHED THEN  
  INSERT VALUES...;
```

```
SELECT *  
FROM COPY_EMP;  
  
20 rows selected.
```

合并行的例子

条件 `c.employee_id = e.employee_id` 被测试，因为 `COPY_EMP` 表为空，条件返回：没有相匹配的行，逻辑进入到 `WHEN NOT MATCHED` 子句中，并且 `MERGE` 命令插入 `EMPLOYEES` 表的行到 `COPY_EMP` 表中。

如果 `COPY_EMP` 表中存在记录，并且 `employee IDs` 在两个表中匹配 (`COPY_EMP` 和 `EMPLOYEES` 表)，`COPY_EMP` 表中已经存在的行将被 `EMPLOYEES` 表中相匹配的行更新。

教师注释

在数据仓库环境中，你可能有一个大的事实表和一个较小维数的表，小表中的行需要有条件地插入到大的事实表中。在这种情况下，`MERGE` 语句是有用的。

数据库事务处理

数据库事务处理由下面的语句组成：

- DML 语句，对数据进行永久的改变
- DDL 语句
- DCL 语句

数据库事务处理

Oracle 服务器基于事务处理确保数据的一致性。在改变数据时，事务给你更多的灵活性和可控性，如果用户程序失败或者系统失败，事务可以确保数据的一致性。

事务由包装成一个整体更改数据的 DML 语句组成，例如，一个在两个帐号之间的解款业务应该包括相同数量的一个借方帐目和一个贷方帐目，这两个动作应该同时失败或者同时成功；即没有记入借方，贷方不应该提交。

事务类型

类型	说明
数据操纵语言 (Data manipulation language DML)	由许多 DML 语句组成，Oracle 服务器将他们视为一个单个整体或者一个逻辑工作单元
数据定义语言 (Data definition language DDL)	由单个的 DDL 语句组成
数据控制语言 (Data control language DCL)	由单个的 DCL 语句组成

数据库事务处理

- 执行第一个 DML SQL 语句时开始
- 遇到下面事件之一结束：
 - 一个 **COMMIT** 或 **ROLLBACK** 语句被发布
 - 一个 DDL 或 DCL 语句执行（自动提交）
 - 用户退出 *iSQL*Plus*
 - 系统崩溃

什么时候开始和结束一个事务？

当遇到第一个 DML 语句一个事务开始，当下面的情况发生时，事务结束：

- 一个 **COMMIT** 或 **ROLLBACK** 语句被发布
- 一个 DDL 语句，例如，**CREATE** 被发布
- 一个 DCL 语句被发布
- 用户推出 *iSQL*Plus*
- 机器失效或者系统崩溃

在一个事务结束以后，下一个可执行的 **SQL** 语句自动开始下一个事务。

一个 DDL 语句或者一个 DCL 语句自动提交，并且因此一个事务隐式结束。

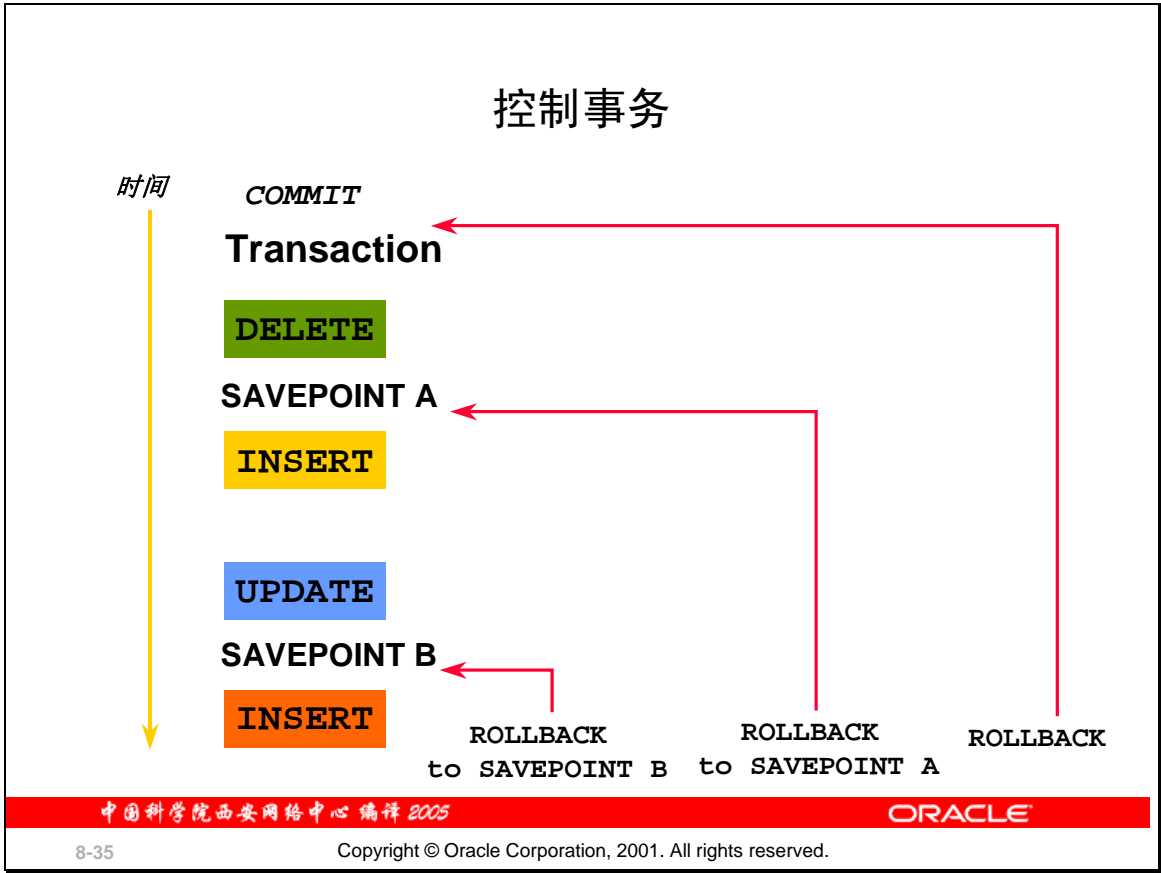
教师注释

请运行脚本 8_cretest.sql 创建测试表并且插入数据到表中。

COMMIT 和 ROLLBACK 语句优点

用 COMMIT 和 ROLLBACK 语句，你能够：

- 确保数据的一致性
- 在数据永久改变之前进行预览
- 分组逻辑相关的操作



显式事务控制语句

你能够用 COMMIT、SAVEPOINT 和 ROLLBACK 语句控制事务逻辑。

语句	说明
COMMIT	结束当前事务，使得所有未决的数据永久改变。
SAVEPOINT name	在当前事务中标记保存点。
ROLLBACK ROLLBACK	结束当前事务，丢弃所有未决的数据改变。
ROLLBACK TO SAVEPOINT name	回滚当前事务到指定的保存点，从而丢弃保存点创建后的任何改变。如果忽略了 TO SAVEPOINT 子句，ROLLBACK 语句回滚整个事务。由于保存点是逻辑的，因此，没有办法列出已经创建的保存点。

注：SAVEPOINT (保存点) 不是 ANSI 标准 SQL 的内容。

教师注释

保存点不是方案对象，并且不能在数据字典中被引用。

回退改变到一个标记

- 用 **SAVEPOINT** 语句在当前事务中创建一个标记
- 用 **ROLLBACK TO SAVEPOINT** 语句回退到该标记

```
UPDATE...  
SAVEPOINT update_done;  
Savepoint created.  
INSERT...  
ROLLBACK TO update_done;  
Rollback complete.
```

回退到保存点

你能够在当前事务中用 **SAVEPOINT** 语句创建一个标记，它把事务分为较小的部分。你可以用 **ROLLBACK TO SAVEPOINT** 语句丢弃未决的改变到该标记。如果你用与前面的保存点相同的名字创建了另一个保存点，哪个早一点时间创建的保存点就被删除了。

教师注释

保存点在 **PL/SQL** 和 **3GL** 程序中是特别有用的，在这些程序中，当前的改变能够基于运行时的条件被有条件地撤消。

隐式事务处理

- 在下面的情况下，一个自动提交发生：
 - DDL 语句被发送
 - DCL 语句被发送
 - 正常退出 iSQL*Plus，没有明确地发送 COMMIT 或 ROLLBACK 语句
- 当 iSQL*Plus 非正常退出时，或者发生系统故障时，一个自动回退发生

隐式事务处理

状态	详情
Automatic commit	DDL 语句或 DCL 语句被发布。 iSQL*Plus 正常退出，不显式地发布 COMMIT 或 ROLLBACK 命令。
Automatic rollback	iSQL*Plus 异常终止，或系统故障。

注：在 iSQL*Plus 环境中可以用第三个命令。AUTOCOMMIT 命令可以被触发或关闭。如果 AUTOCOMMIT 被设置为 on，每个单个的 DML 语句在执行后被立即提交，你不能回滚所做的改变。如果 AUTOCOMMIT 被设置为 off，COMMIT 仍然能够被显式地发布。当 DDL 语句被发布时，或当你从 iSQL*Plus 退出时 COMMIT 语句被发布。

系统故障

当一个事务被系统故障中断时，整个事务被自动回滚。该回滚防止不必要的数据改变错误发生，并且返回表到他们上一次提交时的状态，以这种方式，Oracle 服务器保护表的完整性。

在 iSQL*Plus 中，单击退出按钮，从会话正常的退出。在 SQL*Plus 中输入命令 EXIT 被时，窗口像正常退出一样关闭。

COMMIT 或 ROLLBACK 之前数据的状态

- 以前的数据状态能够被恢复
- 当前用户能用 **SELECT** 语句查看 DML 操作的结果
- 其他用户不能 *观察* 当前用户 DML 语句的结果
- 受影响的行被 *锁定*；其他用户不能改变受影响的行中数据

提交改变

在事务中所做的每一个数据改变在事务被提交之前都是临时的。

在 COMMIT 或 ROLLBACK 语句发布之前数据的状态：

- 数据操纵操作首先影响数据库缓冲区，因此，数据以前的状态可以被恢复。
- 当前用户可以查询表观察到数据操纵操作的结果。
- 其他用户不能观察到当前用户所做的数据操纵操作的结果。Oracle 服务器用读一致性来确保每个用户看到的数据和上次提交时相同。
- 受影响的行被锁定；其他用户不能改变受影响的行中的数据。

教师注释

就 Oracle 服务器来说，数据的改变在事务被提交之前可能实际上已被写入数据库文件，但他们仍然是临时的。

如果许多用户同时对相同的表作了修改，那么，直到用户提交他们的修改之前，每个用户只能看到他自己的修改。

默认情况下，Oracle 服务器有行级 (*row-level locking*)。改变默认的锁机制是可能的。

在 COMMIT 之后数据的状态

- 数据在数据库中被永久地改变.
- 数据的以前状态被永久地丢失
- 所有用户都能观察该结果
- 受影响行的锁定被释放；其它用户可以操纵那些行
- 所有保存点被擦除

提交改变 (续)

用 COMMIT 语句使得未决的改变永久化，在 COMMIT 语句执行后：

- 数据的改变被写到数据库中。
- 数据以前状态永久地丢失。
- 所有用户都可以观察到事务的结果。
- 受影响的行上的锁被释放；其他用户现在可以对行进行新的数据改变。
- 所有保存点被释放。

提交改变

- 产生改变

```
DELETE FROM employees
WHERE employee_id = 99999;
1 row deleted.

INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
1 row inserted.
```

- 提交改变

```
COMMIT;
Commit complete.
```

提交改变 (续)

幻灯片的例子从 EMPLOYEES 表中删除一行，并且插入一个新行到 DEPARTMENTS 表中，然后，发布 COMMIT 语句，使得改变永久化。

例

删除 DEPARTMENTS 表中的部门 290 和 300，并且在 更新 COPY_EMP 表中的一行，使得数据的改变永久化。

```
DELETE FROM departments
WHERE department_id IN (290, 300);
2 rows deleted.
```

```
UPDATE copy_emp
SET department_id = 80
WHERE employee_id = 206;
1 row updated.
```

```
COMMIT;
Commit Complete.
```

教师注释

用这个例子解释 COMMIT 怎样确保两个相关的操作同时发生或根本不同时发生，COMMIT 防止创建空部门。

ROLLBACK 后的数据状态

用 ROLLBACK 语句丢弃所有未决的改变：

- 数据的改变被撤消
- 数据的以前状态被恢复
- 受影响行的锁定被释放

```
DELETE FROM copy_emp;  
22 rows deleted.  
ROLLBACK;  
Rollback complete.
```

回退改变

用 ROLLBACK 语句放弃所有未决的改变，在一个 ROLLBACK 语句执行后：

- 数据的改变被还原。
- 数据以前的状态被恢复。
- 受影响的行上的锁被释放。

例

在试图从 TEST 表中移除一条记录时，你可能以外地清空了整个表，你可以纠正这个错误，然后重新发布正确的语句，并且使得数据改变永久化。

```
DELETE FROM test;  
25,000 rows deleted.  
ROLLBACK;  
Rollback complete.  
DELETE FROM test  
WHERE id = 100;  
1 row deleted.  
SELECT *  
FROM test  
WHERE id = 100;  
No rows selected.  
COMMIT;  
Commit complete.
```

语句级 Rollback

- 如果一个单个的 DML 语句在执行期间失败，仅仅该语句被回退
- Oracle 服务器实现一个隐式的保存点
- 所有其他的改变被保留
- 用户应该执行一个 **COMMIT** 或 **ROLLBACK** 语句来显式地结束事务

语句级回退

如果一个语句的执行错误被发现，一个事务的一部分可以用隐式的回退丢弃。如果一个单个的 DML 语句在一个事务的执行期间失败，它的影响被一个语句级的回退撤消，但在事务中的以前已经由 DML 语句完成的改变不能丢弃，他们可以由用户显示地提交或回滚。

Oracle 在任何数据定义语言 (data definition language DDL) 语句之前和之后发布一个隐式的提交。所以，即使你的 DDL 语句执行不成功，你也不能回退前面的语句，因为服务器已经发布了提交命令。

执行 **COMMIT** 或 **ROLLBACK** 语句来明确地结束事务。

教师注释

Oracle 服务器在数据上实现锁以防止对数据库中数据的并发操作，当某些事件发生时 (例如系统故障) 或当事务完成时，那些锁被释放。当一个 DML 语句成功执行时，数据库上的隐式锁被获得，默认情况下，Oracle 服务器在尽可能的最低级别锁定数据。

执行带 **FOR UPDATE** 子句的 **LOCK TABLE** 语句或 **SELECT** 语句可以手动获得数据库表上的锁。

从 Oracle9i 开始，DBA 有管理回退段的选择，或让 Oracle 自动管理在回退表空间中的回退数据。

请阅读 8-52 页的教师注释。

关于所的更多信息，参考 *Oracle9i Concepts*, “Data Concurrency and Consistency”。

读一致性

- 读一致性在所有时间保证对数据的一致观察
- 一个用户所做的改变不与另一个用户所做的改变冲突
- 读一致性确保下面的操作有同样的数据：
 - 读者不等待写者
 - 写者不等待读者

读一致性

数据库用户用两种方法访问数据库：

- 读操作 (SELECT 语句)
- 写操作 (插入、更新、删除 语句)

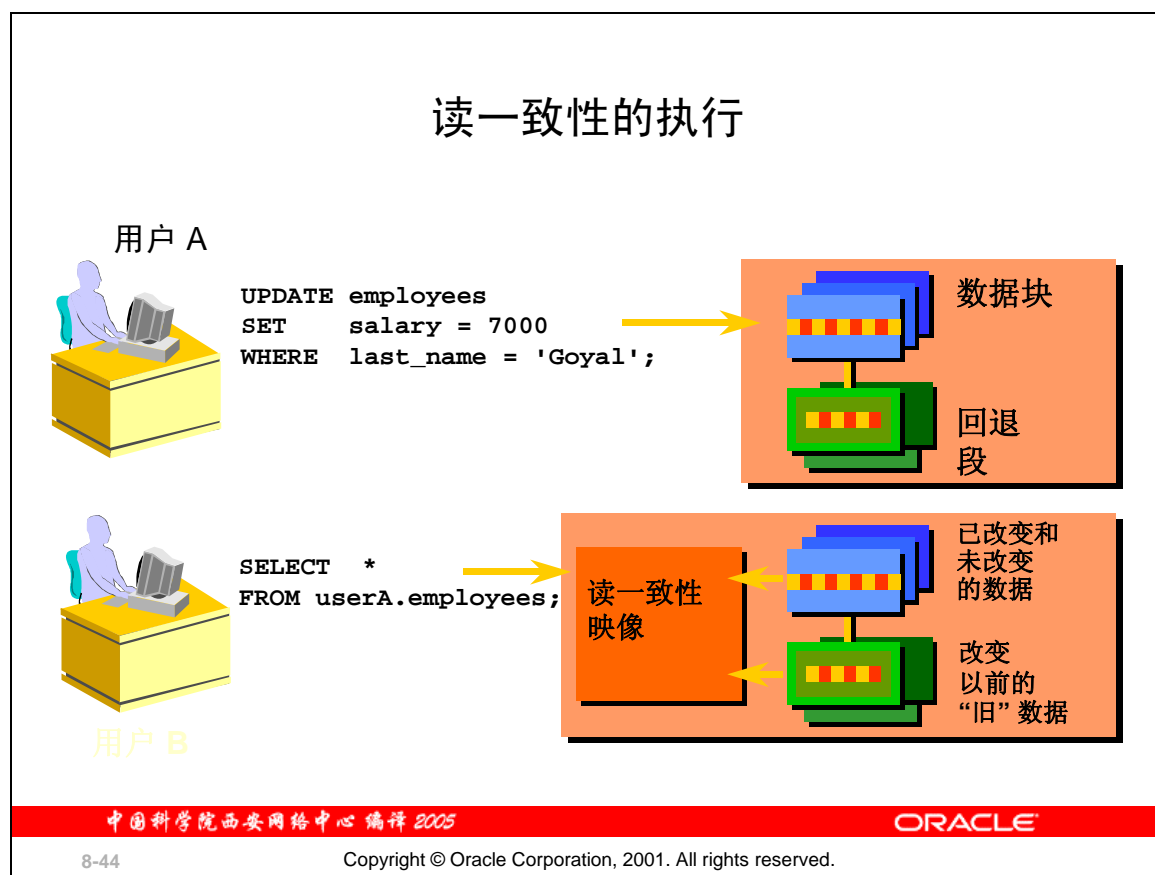
你需要读一致性，所以有下面事发生：

- 数据库读者和写者被确保对数据观察的一致性。
- 读者不能观察正在变化过程中的数据。
- 写者被确保对数据库的改变以一致的方式进行。
- 一个写者所做的改变不破坏另一个写者所做的改变或与其冲突。

读一致性的目的是确保每个用户看到的数据和他最后一次提交，并且在一个 DML 操作开始之前的数据一样。

教师注释

请阅读 8-53 页的教师注释。



读一致性的执行

读一致性是一种自动的执行，该操作在回退段保持一个数据库的局部的拷贝。

在对数据库进行一个插入、更新或者删除时，Oracle 服务器在数据改变之前获得相关数据的拷贝，并且将这些数据写到一个回退段 (undo segment)。

所有读数据者，除了发布修改命令的用户，看到的数据还是改变之前的状态；他们看到的数据是回退段中的数据快照 (snapshot)。

在改变被提交到数据库之前，只有正在修改数据的用户能看见数据库的改变；除他之外的任何人看到的是回退段中的快照，这样就确保数据的读者读到一致的数据，而不是当前正在被修改的数据。

当一个 DML 语句被提交时，对数据库所做的改变对任何执行 SELECT 语句的人成为可见的。在回退段中文件中被 /# 数据占用的空间被释放以重新使用。

如果事务被回滚，修改就被回退：

- 在回退段中原来的，更旧的数据版本被写回到表中。
- 所有用户看到的数据库就像事务开始之前那样。

教师注释

当你提交一个事务时，Oracle 服务器释放回滚信息，但并不立即销毁它，该信息保留在回退段中用来为事务提交之前就已经启动的查询创建相应数据的读一致查看。

从 Oracle9i 开始，DBA 有管理回退段的选择，或让 Oracle 自动管理在回退表空间中的回退数据。相关内容在 DBA 课程中被讨论。

锁定

在 Oracle 数据库中，锁：

- 在并发事务之间防止破坏性的交互作用
- 不需要用户的动作
- 自动使用最低的限制级别
- 在事务处理期间保持
- 有两种类型：显式锁定和隐式锁定

什么是锁？

锁是防止访问相同资源的事务之间的破坏性交互的机制。既可以是用户对象（例如表或行），也可以是对用户不可见的系统对象（例如共享数据结构和数据字典行）。

Oracle 数据库怎样锁定数据

Oracle 锁被自动执行，并且不要求用户干预。对于 SQL 语句隐式锁是必须的，依赖被请求的动作。隐式锁定除 SELECT 外，对所有的 SQL 语句都发生。

用户也可以手动锁定数据，这叫显式锁定。

教师注释

见 8-52 页的教师注释。

隐式锁定

- 两种锁模式：
 - 独占锁：不允许其他用户访问
 - 共享锁：允许其他用户访问
- 高级数据并发操作：
 - DML：表共享，行独占
 - 查询：不需要锁
 - DDL：保护对象定义
- 锁保持直到 commit 或 rollback

中国科学院西安网络中心 编译 2005

ORACLE

8-46

Copyright © Oracle Corporation, 2001. All rights reserved.

DML 锁定

当执行数据操纵语言 (DML) 操作时，Oracle 服务器通过 DML 所防止数据被同时操纵。DML 锁发生在两个级别：

- 共享锁是在表级在 DML 操作期间自动获得的。用共享锁模式，几个事务可以在相同的资源上获得共享锁。
- 对于用 DML 语句修改的每一行，独占锁被自动获得。独占锁在本事务被提交或被回滚之前防止行被其他事务修改。该锁确保无其他用户能够在相同的时间修改相同的行，并且覆盖另一个用户还没有提交的改变。

注：当你修改数据库对象（例如表）时，DDL 锁发生。

教师注释

SELECT...FOR UPDATE 语句也实现锁，相关的内容含盖在 Oracle9i PL/SQL 课程中。

小结

在本课中，您应该已经学会如何用 DML 语句和控制事务：

语句	说明
INSERT	添加一个新行到表中
UPDATE	修改在表中存在的行
DELETE	从表中删除存在的行
MERGE	条件插入或更新表中的数据
COMMIT	使得所有未决的改变永久改变
SAVEPOINT	被用于回退到该保存点标记
ROLLBACK	丢弃所有未决的数据改变

小结

在本课中，你应该已经学会如何用 INSERT、UPDATE 和 DELETE 语句操纵 Oracle 数据库中的数据。用 COMMIT、SAVEPOINT 和 ROLLBACK 语句控制数据的改变。

Oracle 服务器保证在所有时间数据的一致性观察。

锁定可以是隐式的或显式的。

练习 8 概览

本章练习包括下面的主题：

- 插入行到表中
- 更新和删除表中的行
- 控制事务

练习 8 概览

在本章的练习中，你添加行到 MY_EMPLOYEE 表中，从表中更新和删除数据，并且控制事务。

练习 8

插入数据到 MY_EMPLOYEE 表中。

1. 运行在 lab8_1.sql 脚本中的语句来构造 MY_EMPLOYEE 表，用于本章练习的实验。

```
CREATE TABLE my_employee
(id NUMBER(4) CONSTRAINT my_employee_id_nn NOT NULL,
last_name VARCHAR2(25),
first_name VARCHAR2(25),
userid VARCHAR2(8),
salary NUMBER(9,2));
```

2. 查看 MY_EMPLOYEE 表结构，确认列名。

Name	Null?	Type
ID	NOT NULL	NUMBER(4)
LAST_NAME		VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
USERID		VARCHAR2(8)
SALARY		NUMBER(9,2)

```
DESCRIBE my_employee
```

3. 从下面的样本数据中添加第一行数据到 MY_EMPLOYEE 表中，在 INSERT 子句中不要字段列表。

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audrey	aropebur	1550

```
INSERT INTO my_employee
VALUES (1, 'Patel', 'Ralph', 'rpatel', 895);
```

4. 用前面的列表中样本数据的第二行组装 MY_EMPLOYEE 表，这次在 INSERT 子句中显式地列出字段列表。

```
INSERT INTO my_employee (id, last_name, first_name,
userid, salary)
VALUES (2, 'Dancs', 'Betty', 'bdancs', 860);
```

5. 确认你添加到表中的数据。

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860

```
SELECT *  
FROM my_employee;
```

6. 在一个名为 loademp.sql 文本文件中写一个插入语句，装载行到 MY_EMPLOYEE 表中。连接名字的第一个字母和姓的前 7 个字符构成用户 ID。

```
SET ECHO OFF  
SET VERIFY OFF  
INSERT INTO my_employee  
VALUES (&p_id, '&p_last_name', '&p_first_name',  
lower(substr('&p_first_name', 1, 1) ||  
substr('&p_last_name', 1, 7)), &p_salary);  
SET VERIFY ON  
SET ECHO ON
```

7. 运行前面创建的脚本中的插入语句，用下面两行样本数据组装表。

```
SET ECHO OFF  
SET VERIFY OFF  
INSERT INTO my_employee  
VALUES (&p_id, '&p_last_name', '&p_first_name',  
lower(substr('&p_first_name', 1, 1) ||  
substr('&p_last_name', 1, 7)), &p_salary);  
SET VERIFY ON  
SET ECHO ON
```

8. 确认添加到表中的数据。

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750

```
SELECT *  
FROM my_employee;
```

9. 使得数据添加永久化。

```
COMMIT;
```

更新和删除 **EMPLOYEE** 表中的数据。

10. 改 id 为 3 的雇员的名字为 Drexler。

```
UPDATE my_employee
SET last_name = 'Drexler'
WHERE id = 3;
```

11. 改变所有薪水少于 900 的雇员的工资为 1000。

```
UPDATE my_employee
SET salary = 1000
WHERE salary < 900;
```

12. 校验你对表的改变。

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	1000
2	Dancs	Betty	bdancs	1000
3	Drexler	Ben	bbiri	1100
4	Newman	Chad	cnewman	1000

```
SELECT last_name, salary
FROM my_employee;
```

13. 从 MY_EMPLOYEE 表中删除 Betty Dancs。

```
DELETE
FROM my_employee
WHERE last_name = 'Dancs';
```

14. 确认你对表的改变。

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	1000
3	Drexler	Ben	bbiri	1100
4	Newman	Chad	cnewman	1000

```
SELECT *
FROM my_employee;
```

15. 提交所有未决的改变。

```
COMMIT;
```

控制数据事务到 MY_EMPLOYEE 表。

16. 修改前面第 6 题创建的脚本中的语句，用样本数据的最后一行组装表，运行脚本中的语句。

```
SET ECHO OFF  
SET VERIFY OFF  
INSERT INTO my_employee  
VALUES (&p_id, '&p_last_name', '&p_first_name',  
lower(substr('&p_first_name', 1, 1) ||  
substr('&p_last_name', 1, 7)), &p_salary);  
SET VERIFY ON  
SET ECHO ON
```

17. 确认添加的数据。

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	1000
3	Drexler	Ben	bbiri	1100
4	Newman	Chad	cnewman	1000
5	Ropeburn	Audrey	aropebur	1550

```
SELECT *  
FROM my_employee;
```

18. 在事务的过程中标记一个中间点。

```
SAVEPOINT step_18;
```

19. 清空整个表。

```
DELETE  
FROM my_employee;
```

20. 确认表是空的。

```
SELECT *  
FROM my_employee;
```

21. 丢弃最近的 DELETE 操作，而不丢弃前面的插入操作。

```
ROLLBACK TO step_18;
```

22. 确认新行还是完整的。

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	1000
3	Drexler	Ben	bbiri	1100
4	Newman	Chad	cnewman	1000
5	Ropeburn	Audrey	aropebur	1550

```
SELECT *  
FROM my_employee;
```

23. 使得数据添加永久化。

```
COMMIT;
```

教师注释 (对 8-42 - 8-45 页)

演示: 8_select.sql

目的: 举例说明一个读者不锁定另一个读者的概念。

- 用 teach/oracle 帐号登录 iSQL*Plus。
- 用 oraxx/oracle 帐号登录 iSQL*Plus。
- 从 teach/oracle 帐号运行 8_select.sql 脚本。(该脚本从 DEPARTMENTS 表中选择所有行)。
- 在 oraxx/oracle 帐号中运行 8_select.sql 脚本。(该脚本从 DEPARTMENTS 选择所有记录)。

在上面两个登录中, 脚本执行成功。该示范证明概念: 一个读者不锁定另一个读者。

演示: 8_grant.sql、8_update.sql 和 8_select.sql。

目的: 举例说明写者不锁定读者。

- 在 teach/oracle 帐号中运行 8_grant.sql 脚本。(该脚本授予 DEPARTMENTS 表的 SELECT 和 UPDATE 权限给 oraxx 帐号)。
- 在 teach/oracle 帐号中运行 8_update.sql 脚本。(该脚本更新 DEPARTMENTS 表, 改变部门 ID 20 到位置 1500。更新操作在 DEPARTMENTS 表上放置一个锁)。
- 在 teach/oracle 帐号中运行 8_select.sql 脚本。(该脚本从 DEPARTMENTS 表中选择所有记录, 观察部门 ID 20 的位置被改变到位置 ID 1500)。
- 在 oraxx/oracle 帐号中运行 8_select.sql 脚本。(该脚本从 DEPARTMENTS 表中选择所有记录)。

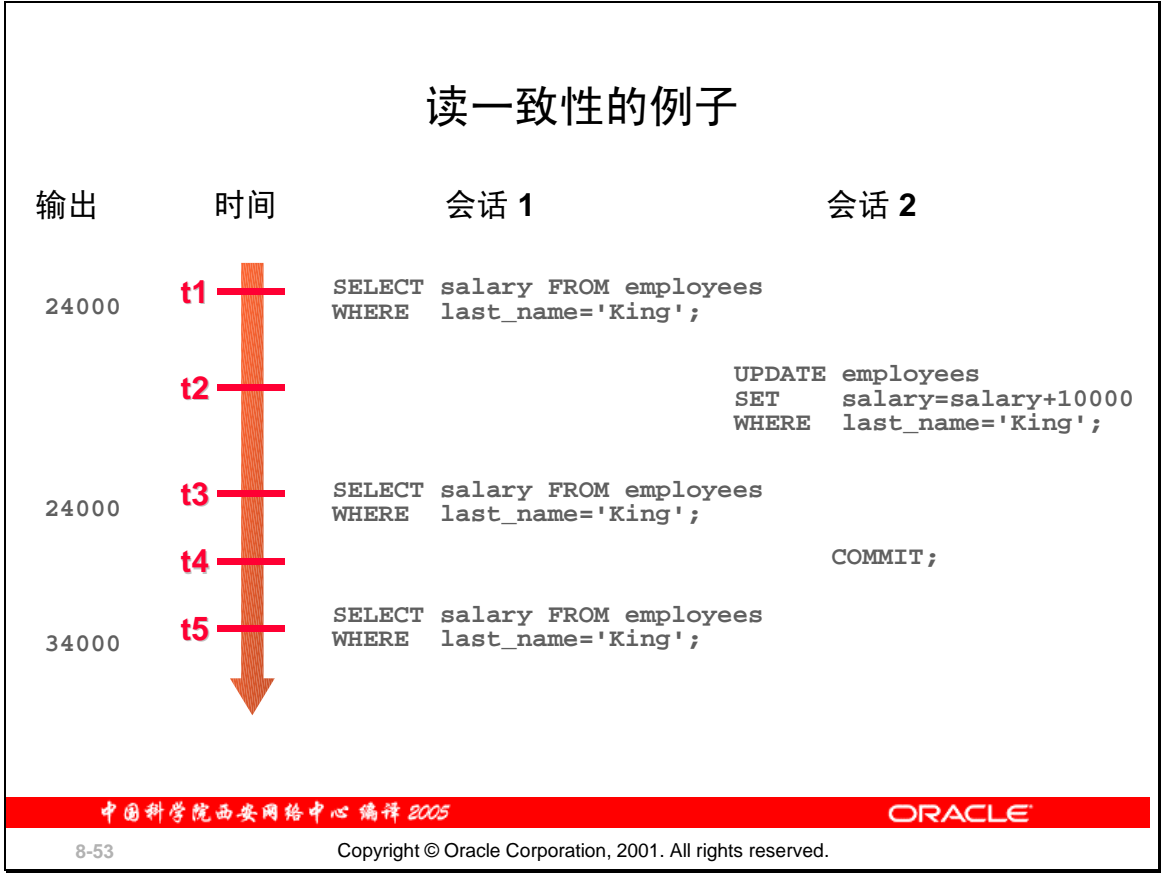
观察脚本在 oraxx/oracle 帐号中执行成功, 但部门 ID 20 的位置 ID 还是 1800。该演示证明概念: 一个写者不锁定一个读者。

演示: 8_update.sql、8_rollback.sql 和 8_select.sql。

目的: 举例说明一个写者锁定另一个写者。

- 在 oraxx/oracle 帐号中运行 8_update.sql 脚本。(该脚本不执行, 因为 DEPARTMENTS 表被 teach/oracle 帐号锁定)。
- 切换到 teach/oracle 帐号, 并且运行 8_rollback.sql 脚本。(该脚本回滚事务, 因此释放在 DEPARTMENTS 表上的锁)。
- 切换到 oraxx/oracle 帐号, 你看到 8_update.sql 脚本已经执行成功, 因为在 DEPARTMENTS 表上的锁已经被释放了。
- 在 oraxx/oracle 帐号中运行 8_select.sql 脚本。(该脚本从 DEPARTMENTS 表中选择所有记录。观察部门 ID 20 的位置已经被改到位置 ID 1500)。

本演示证明概念: 一个写者锁定另一个写者。



教师注释 (对 8-43 页)

读一致性例子

为了持续一个 SQL 语句，读一致性保证，当语句的处理已经开始时，被选择的数据对于时间点是一致的。

Oracle 服务器在回退段的数据块中保存未提交的数据（以前的数据映像）。只要改变没有被提交，所有用户其他看见的是原始数据。Oracle 服务器使用既使用表段中的数据也使用回退段中的数据来产生数据的读一致的观察。

在幻灯片的例子中，会话 2 的更新对会话 1 是不可见的，直到会话 2 提交更新（从 t4 往后）。对于在时间点 t3 的选择语句，King 的薪水必须被从一个回退段的数据库块中读出，而该回退段属于会话 2 的事务。