

讲师: collen7788@126.com

P r e s e n t a t i o n

处理数据

本章目标

1

使用DML语句

2

控制事务

数据操作语言

- ❖ **DML(Data Manipulation Language – 数据操作语言)** 可以在下列条件下执行：
 - 向表中插入数据
 - 修改现存数据
 - 删除现存数据
- ❖ 事务是由完成若干项工作的**DML**语句组成的

插入数据

❖ INSERT 语句语法

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

使用这种语法一次只能向表中插入一条数据

插入数据

- ❖ 为每一列添加一个新值。
- ❖ 按列的默认顺序列出各个列的值。
- ❖ 在 **INSERT** 子句中随意列出列名和他们的值。
- ❖ 字符和日期型数据应包含在单引号中。

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

向表中插入空值

- ❖ 隐式方式：在列名表中省略该列的值。

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

- ❖ 显式方式：在**VALUES** 子句中指定空值。

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

插入指定的值

SYSDATE 记录当前系统的日期和时间。

```
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.
```

插入指定的值

- 加入新员工

```
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.
```


创建脚本

- ❖ 在**SQL** 语句中使用 **&** 变量指定列值。
- ❖ **&** 变量放在**VALUES**子句中。

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

Define Substitution Variables

"department_id" 40
"department_name" Human Resources
"location" 2500

Submit for Execution

Cancel

1 row created.

从其它表中拷贝数据

- ❖ 在 **INSERT** 语句中加入子查询。

```
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.


- ❖ 不必书写 **VALUES** 子句。
- ❖ 子查询中的值列表应与 **INSERT** 子句中的列名对应

更新数据

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



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	

UPDATE 语句语法

- ❖ 使用 **UPDATE** 语句更新数据。

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

- ❖ 可以一次更新多条数据。

更新数据

- ❖ 使用 **WHERE** 子句指定需要更新的数据。

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

- ❖ 如果省略**WHERE**子句，则表中的所有数据都将被更新。

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

在UPDATE语句中使用子查询

- ❖ 更新 **114**号员工的工作和工资使其与 **205**号员工相同。

```
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.
```

在UPDATE语句中使用子查询

- ❖ 在 **UPDATE** 中使用子查询，使更新基于另一个表中的数据。

```
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.
```

更新中的数据完整性错误

```
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
```

不存在 55 号部门

删除数据

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

从表DEPARTMENTS 中删除一条记录。

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

DELETE 语句

- ❖ 使用 **DELETE** 语句从表中删除数据。

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

删除数据

- ❖ 使用**WHERE** 子句指定删除的记录。

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

- ❖ 如果省略**WHERE**子句，则表中的全部数据将被删除。

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

在 DELETE 中使用子查询

- ❖ 在 **DELETE** 中使用子查询，使删除基于另一个表中的数据。

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

1 row deleted.
```

删除中的数据完整性错误

```
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
```

You cannot delete a row that contains a primary key that is used as a foreign key in another table.

Delete和Truncate

- ❖ 都是删除表中的数据
- ❖ **Delete**操作可以**rollback**，可以闪回
- ❖ **Delete**操作可能产生碎片，并且不释放空间
- ❖ **Truncate**:清空表



数据库事务

❖ 数据库事务由以下的部分组成:

- 一个或多个**DML** 语句
- 一个 DDL(Data Definition Language – 数据定义语言) 语句
- 一个 DCL(Data Control Language – 数据控制语言) 语句

数据库事务

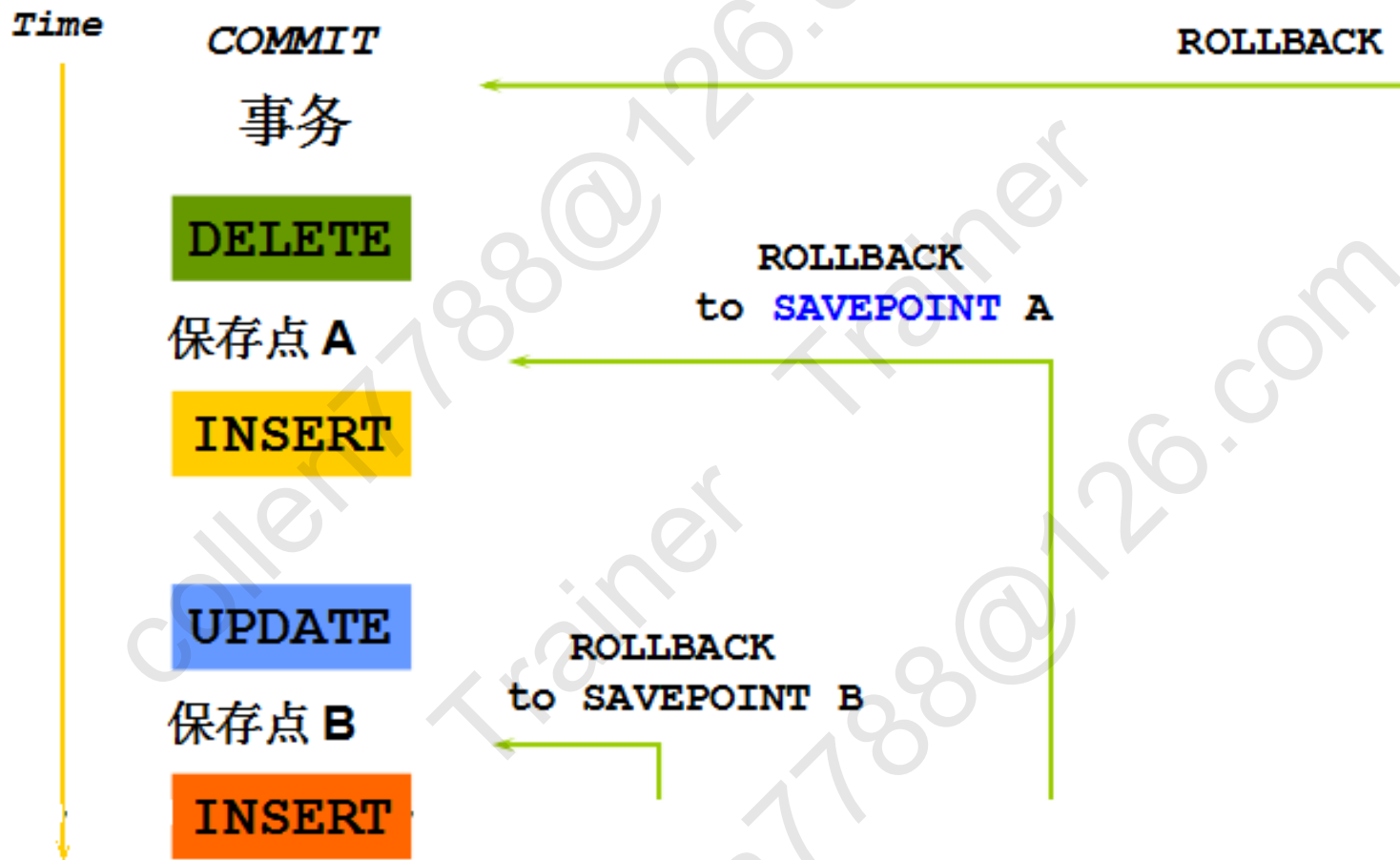
- ❖ 以第一个 **DML** 语句的执行作为开始
- ❖ 以下面的其中之一作为结束：
 - 显示结束: `commit` `rollback`
 - 隐式结束 (自动提交): `DDL`语言, `DCL`语言, `exit` (事务正常退出)
 - 隐式回滚 (系统异常终了): 关闭窗口, 死机, 掉电

COMMIT和ROLLBACK语句的优点

❖ 使用**COMMIT** 和 **ROLLBACK**语句,我们可以:

- 确保数据完整性。
- 数据改变被提交之前预览。
- 将逻辑上相关的操作分组。

控制事务



回滚到保留点

- ❖ 使用 `SAVEPOINT` 语句在当前事务中创建保存点。
- ❖ 使用 `ROLLBACK TO SAVEPOINT` 语句回滚到创建的保存点。

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

数据库的隔离级别

- ❖ 对于同时运行的多个事务, 当这些事务访问数据库中相同的数据时, 如果没有采取必要的隔离机制, 就会导致各种并发问题:
 - **脏读**: 对于两个事物 T1, T2, T1 读取了已经被 T2 更新但还没有被提交的字段. 之后, 若 T2 回滚, T1 读取的内容就是临时且无效的.
 - **不可重复读**: 对于两个事物 T1, T2, T1 读取了一个字段, 然后 T2 更新了该字段. 之后, T1 再次读取同一个字段, 值就不同了.
 - **幻读**: 对于两个事物 T1, T2, T1 从一个表中读取了一个字段, 然后 T2 在该表中插入了一些新的行. 之后, 如果 T1 再次读取同一个表, 就会多出几行.
- ❖ **数据库事务的隔离性**: 数据库系统必须具有隔离并发运行各个事务的能力, 使它们不会相互影响, 避免各种并发问题.
- ❖ 一个事务与其他事务隔离的程度称为隔离级别. 数据库规定了多种事务隔离级别, 不同隔离级别对应不同的干扰程度, 隔离级别越高, 数据一致性就越好, 但并发性越弱

数据库的隔离级别

❖ 数据库提供的 4 种事务隔离级别:

隔离级别	描述
READ UNCOMMITTED (读未提交数据)	允许事务读取未被其他事物提交的变更,脏读,不可重复读和幻读的问题都会出现
READ COMMITTED (读已提交数据)	只允许事务读取已经被其它事务提交的变更,可以避免脏读,但不可重复读和幻读问题仍然可能出现
REPEATABLE READ (可重复读)	确保事务可以多次从一个字段中读取相同的值,在这个事务持续期间,禁止其他事物对这个字段进行更新,可以避免脏读和不可重复读,但幻读的问题仍然存在.
SERIALIZABLE(串行化)	确保事务可以从一个表中读取相同的行,在这个事务持续期间,禁止其他事务对该表执行插入,更新和删除操作,所有并发问题都可以避免,但性能十分低下.

❖ Oracle 支持的 2 种事务隔离级别: **READ COMMITTED**, **SERIALIZABLE**. Oracle 默认的事务隔离级别为: **READ COMMITTED**

❖ Mysql 支持 4 中事务隔离级别. Mysql 默认的事务隔离级别为: **REPEATABLE READ**

总结

- ❖ 通过本章学习, 您应学会如何使用**DML**语句改变数据和事务控制

语句	功能
INSERT	插入
UPDATE	修正
DELETE	删除
COMMIT	提交
SAVEPOINT	保存点
ROLLBACK	回滚

讲师: collen7788@126.com

P r e s e n t a t i o n

Thank you