**Transaction Management & Database Administration**

**CSC317 Database Systems II**

**Professor: Dr. Kurt Diesch**

**By Odiscious Dozier**

**Trident University**

**CASE 02 - Database Transactions -- I**

1. How many database requests can you identify for an inventory update for both PRODUCT and PART?

A database request is defined as a database query. This can be done by, choosing parameters from a menu, using query by example (QBE), or using a query language such as the structured query language (SQL) (Beal, 2016).

In order to add a part to the PRODUCT table a single UPDATE query would need to be executed.

In order to remove a part from each of the PART tables (3 in this example), the UPDATE query would need to be executed 3 times, once for each table.

1. Using SQL, write each database request you identified in step 1.

There are currently 205 products in the PRODUCT table. This update will add one product to the table by modifying the PROD\_QOH to 206.

//Note: from this example it’s impossible to determine the particular datatype. Here, I’m assuming nvarchar(n).

UPDATE PRODUCT  
SET PRODUCT. **PROD\_QOH** = 206   
WHERE PRODUCT. **PROD\_CODE** *=* ‘XYZ’*;*

There are currently 150 parts with code ‘X’ in the PART table. This update will subtract one part to the table by modifying the PART\_QOH to 149.

//Note: from this example it’s impossible to determine the particular datatype. Here, I’m assuming nvarchar(n).

UPDATE PART  
SET PART. PART **\_QOH** = 149   
WHERE PART. PART **\_CODE** *=* ‘X’*;*

There are currently 300 parts with code ‘Y’ in the PART table. This update will subtract one part to the table by modifying the PART\_QOH to 299.

//Note: from this example it’s impossible to determine the particular datatype. Here, I’m assuming nvarchar(n).

UPDATE PART  
SET PART. PART **\_QOH** = 299   
WHERE PART. PART **\_CODE** *=* ‘Y’*;*

There are currently 150 parts with code ‘X’ in the PART table. This update will subtract one part to the table by modifying the PART\_QOH to 149.

//Note: from this example it’s impossible to determine the particular datatype. Here, I’m assuming nvarchar(n).

UPDATE PART  
SET PART. PART **\_QOH** = 399   
WHERE PART. PART **\_CODE** *=* ‘Z’*;*

1. Write the complete transaction(s).

By default, all transactions are read write transactions. This means that a logical snapshot of the database is created; thus, guaranteeing “that all of the read within a single statement get consistent data from the database” (“Oracle Tutorials”, n.d.).

The COMMIT is executed, automatically, at the end of a successful query execution, but if it is set to off then the COMMIT statement must be explicitly ran (“Connection”, n.d.). If the query is unsuccessful then, it is rolled back.

These examples assume COMMIT is automatically executed.

There are currently 205 products in the PRODUCT table. This update will add one product to the table by modifying the PROD\_QOH to 206.

//Note: from this example it’s impossible to determine the particular datatype. Here, I’m assuming nvarchar(n).

SET TRANSACTION READ WRITE NAME ‘Update PROD\_QOH(205,206)’;

UPDATE PRODUCT  
SET PRODUCT. **PROD\_QOH** = 206   
WHERE PRODUCT. **PROD\_CODE** *=* ‘XYZ’*;*

There are currently 150 parts with code ‘X’ in the PART table. This update will subtract one part to the table by modifying the PART\_QOH to

149.

//Note: from this example it’s impossible to determine the particular datatype. Here, I’m assuming nvarchar(n).

SET TRANSACTION READ WRITE NAME ‘Update PART\_QOH(150,149)’;

UPDATE PART  
SET PART. PART **\_QOH** = 149   
WHERE PART. PART **\_CODE** *=* ‘X’*;*

There are currently 300 parts with code ‘Y’ in the PART table. This update will subtract one part to the table by modifying the PART\_QOH to 299.

//Note: from this example it’s impossible to determine the particular datatype. Here, I’m assuming nvarchar(n).

SET TRANSACTION READ WRITE NAME ‘Update PART\_QOH(300,299)’;

UPDATE PART  
SET PART. PART **\_QOH** = 299   
WHERE PART. PART **\_CODE** *=* ‘Y’*;*

There are currently 150 parts with code ‘X’ in the PART table. This update will subtract one part to the table by modifying the PART\_QOH to 149.

//Note: from this example it’s impossible to determine the particular datatype. Here, I’m assuming nvarchar(n).

SET TRANSACTION READ WRITE NAME ‘Update PART\_QOH(400,399)’;

UPDATE PART  
SET PART. PART **\_QOH** = 399   
WHERE PART. PART **\_CODE** *=* ‘Z’*;*

These examples assume COMMIT is not automatically executed.

There are currently 205 products in the PRODUCT table. This update will add one product to the table by modifying the PROD\_QOH to 206.

//Note: from this example it’s impossible to determine the particular datatype. Here, I’m assuming nvarchar(n).

SET TRANSACTION READ WRITE NAME ‘Update PROD\_QOH(205,206)’;

UPDATE PRODUCT  
SET PRODUCT. **PROD\_QOH** = 206   
WHERE PRODUCT. **PROD\_CODE** *=* ‘XYZ’

*COMMIT;*

There are currently 150 parts with code ‘X’ in the PART table. This update will subtract one part to the table by modifying the PART\_QOH to

149.

//Note: from this example it’s impossible to determine the particular datatype. Here, I’m assuming nvarchar(n).

SET TRANSACTION READ WRITE NAME ‘Update PART\_QOH(150,149)’;

UPDATE PART  
SET PART. PART **\_QOH** = 149   
WHERE PART. PART **\_CODE** *=* ‘X’

*COMMIT;*

There are currently 300 parts with code ‘Y’ in the PART table. This update will subtract one part to the table by modifying the PART\_QOH to 299.

//Note: from this example it’s impossible to determine the particular datatype. Here, I’m assuming nvarchar(n).

SET TRANSACTION READ WRITE NAME ‘Update PART\_QOH(300,299)’;

UPDATE PART  
SET PART. PART **\_QOH** = 299   
WHERE PART. PART **\_CODE** *=* ‘Y’

*COMMIT;*

There are currently 150 parts with code ‘X’ in the PART table. This update will subtract one part to the table by modifying the PART\_QOH to 149.

//Note: from this example it’s impossible to determine the particular datatype. Here, I’m assuming nvarchar(n).

SET TRANSACTION READ WRITE NAME ‘Update PART\_QOH(400,399)’;

UPDATE PART  
SET PART. PART **\_QOH** = 399   
WHERE PART. PART **\_CODE** *=* ‘Z’

*COMMIT;*

1. Write the transaction log, using the table below as your template. Note: this table is not to scale.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TRL**  **ID** | **TRX**  **NUM** | **PREV**  **PTR** | **NEXT**  **PTR** | **OPERATION** | **TABLE** | **ROW**  **ID** | **ATTRIBUTE** | **BEFORE**  **VALUE** | **AFTER**  **VALUE** |
| 341 | 101 | Null | 352 | START | \*\*\*\*Start Transaction |  |  |  |  |
| 352 | 101 | 341 | 355 | UPDATE | PRODUCT | 1558-QW1 | PROD\_QOH | 205 | 206 |
| 355 | 101 | 344 | NULL | COMMIT | \*\*\*\*End of Transaction |  |  |  |  |
| 360 | 101 | Null | 363 | START | \*\*\*\*Start Transaction |  |  |  |  |
| 363 | 101 | 352 | 365 | UPDATE | PART | 10011 | PART\_QOH | 150 | 149 |
| 365 | 101 | 363 | 369 | UPDATE | PART | 10012 | PART\_QOH | 300 | 299 |
| 369 | 101 | 376 | 373 | UPDATE | PART | 10013 | PART\_QOH | 400 | 399 |
| 373 | 101 | 420 | NULL | COMMIT | \*\*\*\*End of Transaction |  |  |  |  |

REFERENCES

Beal, V. (2016). Query. Retrieved February 18, 2016, from <http://www.webopedia.com/TERM/Q/query.html>

Connection (Java Platform SE 7 ). (n.d.). Retrieved February 18, 2016, from <http://docs.oracle.com/javase/7/docs/api/java/sql/Connection.html#setAutoCommit(boole> an)

Distributed Databases. (n.d.). Retrieved February 18, 2016, from <https://docs.oracle.com/cd/A57673_01/DOC/server/doc/SCN73/ch21.htm>

Oracle Tutorials - What Is a READ WRITE Transaction. (n.d.). Retrieved February 18, 2016, from <http://dba.fyicenter.com/faq/oracle/What-Is-READ-WRITE-Transaction.html>

SQL Transactions. (n.d.). Retrieved February 18, 2016, from <http://www.tutorialspoint.com/sql/sql-transactions.htm>

Why Oracle says 'SET TRANSACTION' is one means of transaction beginning. (n.d.). Retrieved February 18, 2016, from <http://stackoverflow.com/questions/23452782/why-oracle-says-> set-transaction-is-one-means-of-transaction-beginning