Note: most of this lecture has been removed due to copyright restrictions.

1.204 Lecture 4

JDBC

Code examples from JDBC API Tutorial and Reference

JDBC API

- **Package (library) of classes and methods to connect from a Java application to DBMS, execute SQL statements and retrieve results**
  - **SQL syntax primarily based on SQL-92 standard**
  - **Standard set of error codes**
  - **Standard way to connect and log on to DBMS**
  - **Standard representation of data types**
  - **Standard methods for data type conversions**
  - **Standard methods to send SQL queries and receive result sets**
  - **JDBC has level 1-4 functionality to deal with simple and sophisticated interfaces.**
  - **It can interface to files and other data sources as well**

Transactions

- Group of operations often must be treated as atomic unit
  - Start transaction
    - Insert OrderHeader
    - While more OrderDetail (line items) exist:
      - Select Part
      - Update Part inventory
      - Insert OrderDetail row
  - Commit transaction if everything succeeds
  - Roll back transaction if any error occurs:
    - In Order Header
    - In OrderDetail
    - Server crashes
    - Disk crashes
    - Network dies
    - Etc.

Transaction properties (ACID)

- Atomicity. Either all of transactions are executed or all are rolled back
  - Account transfer debit and credit both succeed or fail
- Consistency. Only legal states can exist
  - If order detail cannot be written, order header is rolled back
- Isolation. Results not seen by other transactions until the transaction is complete
  - Account transfer debit and credit either both seen or neither is seen
- Durability. Data is persistent even if hardware or software crashes: What is written on the disk is correct
  - Account balance is maintained

Transactions

- Multi-user databases have other transaction issues
- Two database actions conflict if one or both are write operations. Examples of problems:
  - Lost updates:
    - 7 parts in inventory
    - Transactions 1 and 2 simultaneously read 7 as the current quantity
    - Transaction 1 finishes first, adds 3 parts, writes 10 as quantity
    - Transaction 2 finishes second, subtracts 5 parts, writes 2 as quantity!
  - Uncommitted changes:
    - Transaction 1 adds 3 parts, writes 10 as quantity
    - Transaction 2 reads 10 as quantity
    - Transaction 1 aborts (rolls back), leaving transaction 2 with wrong data

Transactions

- Databases use locks for concurrency. One simple scheme is pessimistic locking:
  - Writes obtain an exclusive lock, preventing reads or writes
  - Reads obtain nonexclusive locks, allowing other reads but preventing a writer from obtaining an exclusive lock
- Or you can use optimistic locking (logs)
  - No locks are used. Check if row exists, is same after operation
  - If not, issue error and program must retry. Better performance.
- Databases use logs for recovery.
  - Log file of all changes is written in addition to making the changes in the database. (This is a key bottleneck in architecture.)
  - Change can't be committed until the log is written to stable storage.
    - Changes usually committed before tables actually updated on disk
  - If a change is rolled back, the log is read to reverse the transactions.
  - If a system or disk crashes, the log is rerun from the last checkpoint to restore the database.
  - Turn off logs when loading batch data or recovering