Java-Success.com

Register | Login | Logout | Contact Us

Industrial strength Java/JEE Career Companion to open more doors

search here … Go

Home | Java FAQs | 600+ Java Q&As | Career | Tutorials | Member | Why?

Can u Debug? | Java 8 ready? | Top X | Productivity Tools | Judging Experience?

Home › Interview › Pressed for time? Java/JEE Interview FAQs › Java Key Area Essentials › ♦ 9 Java Transaction Management Interview Q&A

♦ 9 Java Transaction Management Interview Q&A

Posted on September 12, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

0 Like Share Tweet 0 G+1 Share

Q1. What is a Transaction?
A1. A transaction is a **set of operations that should be completed as a unit**. If one operation fails then all the other operations fail as well.

Example 1: If you transfer funds between two accounts there will be two operations in the set

Operation 1: Withdraw money from one account.
Operation 2: Deposit money into other account.

These two operations should be completed as a single unit. Otherwise your money will get lost if the withdrawal is successful and the deposit fails. There are four characteristics (ACID properties) for a Transaction.

— Atomicity: In a transaction involving two or more operations, either all operations are committed or none are. Either both "withdraw money from one account" and "deposit money into other account" are committed (i.e. written to the database tables) or none are written to the database tables.

— Consistency: A guarantee that your data will be consistent without violating the constraints you have on related data. For example,

1) An INTEGER field is guaranteed to not allow string data.
2) Setting NOT NULL for a given field.
3) A referential integrity constraint that says you cannot delete a row in Table A if Table B has records that refer to that row.
4) A consistency rule that dictates that a bank account number must follow a specific pattern- it must begin with a 'C' for checking account or 'S' for savings account, then followed by 12 digits.
5) Another consistency rule may state that the 'Customer Name' field cannot be empty when creating a customer.

If a certain transaction occurs that attempts to introduce inconsistent data, the entire transaction is rolled back and an error is returned to the user. If a transaction successfully executes, it will take the database from one state that is consistent with the rules to another state that is also consistent with the rules.

Note: In NoSQL (i.e. Not only SQL) world the term consistency does not mean "C" in ACID. NoSQL consistency is know as "**eventual**" consistency. For example

DAY 1: A share market update program writes a record to the database "Avg selling price for XYZ = $35.50 on 15/Jun/2015".
DAY 2: The next day, share market update program updates

Other Interview Q&A f
Free Java Interviev

16 Technical Key Areas

open all | close all
Best Practice (6)
Coding (26)
Concurrency (6)
Design Concepts (7)
Design Patterns (11)
Exception Handling (3
Java Debugging (21)
Judging Experience I
Low Latency (7)
Memory Managemen
Performance (13)
QoS (8)
Scalability (4)
SDLC (6)
Security (13)
Transaction Managen

80+ step by step Java Tutorials

open all | close all
Setting up Tutorial (6)
Tutorial - Diagnosis (2
Akka Tutorial (9)
Core Java Tutorials (2
Hadoop & Spark Tuto
JEE Tutorials (19)
Scala Tutorials (1)
Spring & HIbernate Tu
Tools Tutorials (19)
Other Tutorials (45)

the database so that “Avg selling price for XYZ = $32.00 on 16/Jun/2015”.

If someone queries the NoSQL database immediately after step 2, then there is a chance that a user will see that the average selling price on 16/Jun/2015 was $32.50, and NOT $32.00, which represents the latest data. However, “**eventually**” the data will, in fact, propagate such that everyone eventually sees $32.00.

Q. Why does this happen in NoSQL?
A. In a distributed system like NoSQL, the same piece of data is usually replicated to multiple machines. When you update that piece of information on one of the machines, it may take some time (usually milliseconds) to reach every machine that holds the replicated data. This creates the possibility that you might get information that hasn’t yet updated on the replica.

Q. Is this a serious anomally?
A. In many scenarios, the above anomaly is acceptable. In scenarios where this type of behavior is not acceptable, the NoSQL databases give the the choice, per transaction, whether data to be “eventually consistent” or “**strongly consistent**”. Strong consistency would guarantee that everyone saw $32.00 as the answer to their query after step 2.

— Isolation: Prevents data being corrupted by concurrent access by two different sources (e.g. 2 users accessing the same data). It keeps transactions isolated or separated from each other until they are finished. The reads and writes of successful transactions will not be affected by reads and writes of any other transactions. There are 4 type of isolation levels: **1)** READ UNCOMMITTED **2)** READ COMMITTED, **3)** REPEATABLE READ , and **4)** SERIALIZABLE.

— Durability: Ensures that the changes have been recorded to a durable medium (such as a hard disk) once a transaction is complete. Committed transactions will not be lost, even in the event of abnormal termination. Once it is notified that a transaction has been committed, you can be certain that the data will not be lost.

Example 2: Booking an airline ticket where actions required are to reserve a seat and pay for the ticket. Both of these actions must either succeed or if one fails, both actions should be rolled back. Isolation is one of the basic tenets of the ACID model to prevent any anomalies relating to **dirty reads**, **non-repeatable reads**, and **phantom reads**. These anomalies can leave a database in inconsistent state when accessed concurrently. Improper isolation levels could end up double booking a seat in an airline booking system.

Dirty Reads occur when one transaction reads data written by another, uncommitted, transaction. The danger with dirty reads is that the other transaction might never commit, leaving the original transaction with "dirty" data. **Non Repeatable Reads** occur when one transaction attempts to access the same data twice and a second transaction modifies the data between the first transaction's read attempts. This may cause the first transaction to read two different values for the same data, causing the original read to be non-repeatable.

Q2. What does auto commit do?
A2. Transactions maintain **data integrity**. A transaction has a beginning and an end like everything else in life. The setAutocommit(….), commit() and rollback() are used for marking the transactions (aka known as **transaction demarcation**). When a connection is created, it is usually in auto-commit mode. This means that each individual SQL statement is treated as a transaction and will be automatically committed immediately after it is executed. The way to allow two or more statements to be grouped into a transaction is to disable auto-commit mode.

```
try{
    Connection myConnection = dataSource.getConn

    // set autoCommit to false
    myConnection.setAutoCommit(false);

   withdrawMoneyFromFirstAccount(.............);
```

```
    depositMoneyIntoSecondAccount(............);

    myConnection.commit();
}
catch(Exception sqle) {
    try{
      myConnection .rollback();
    }catch( Exception e){}
}
finally{
     try{
       if( conn != null) {
           conn.close();
       }
     } catch( Exception e) {}
}
```

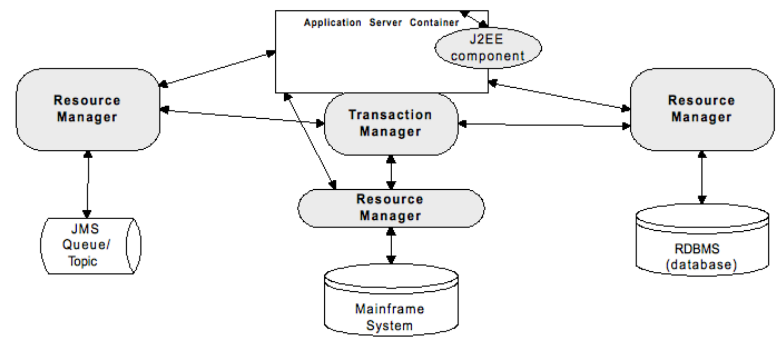
The above code ensures that both operation 1 and operation 2 succeed or fail as an atomic unit and consequently leaves the database in a consistent state.

Q3. What is a distributed (aka JTA/XA) transaction? How does it differ from a local transaction?
A3. here are two types of transactions:

1) **Local transaction:** Transaction is within the same database. As we have seen above, with JDBC transaction demarcation, you can combine multiple SQL statements into a single transaction, but the transactional scope is limited to a single database connection. A JDBC transaction cannot span multiple databases.

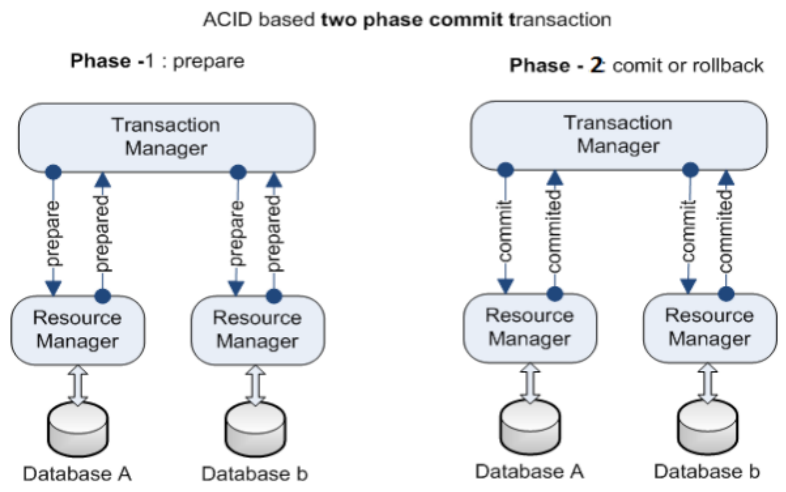
2) **Distributed Transaction** (aka **Global Transaction**, **JTA/XA transaction**): The transactions that constitute a distributed transaction might be in the same database, but more typically are in different databases and often in different locations. For example, A distributed transaction might consist of money being transferred from an account in one bank to an account in another bank. You would not want either transaction committed without assurance that both will complete successfully. The Java Transaction API (JTA) and its sibling Java Transaction Service (JTS), provide distributed transaction services for the JEE platform. A distributed transaction (aka JTA/XA transaction) involves a **transaction manager** and **one or more resource managers**. A resource manager represents any kind of data store. The transaction manager is responsible for coordinating communication

between your application and all the resource managers. A transaction manager decides whether to commit or rollback at the end of the transaction in a distributed system. A resource manager is responsible for controlling of accessing the common resources in the distributed system.



Global or Distributed Transaction

Q4. What is a 2-phase commit?
A4. A distributed (e.g. across a database and a messaging queue) transaction using a concept known as the **2-phase commit**.



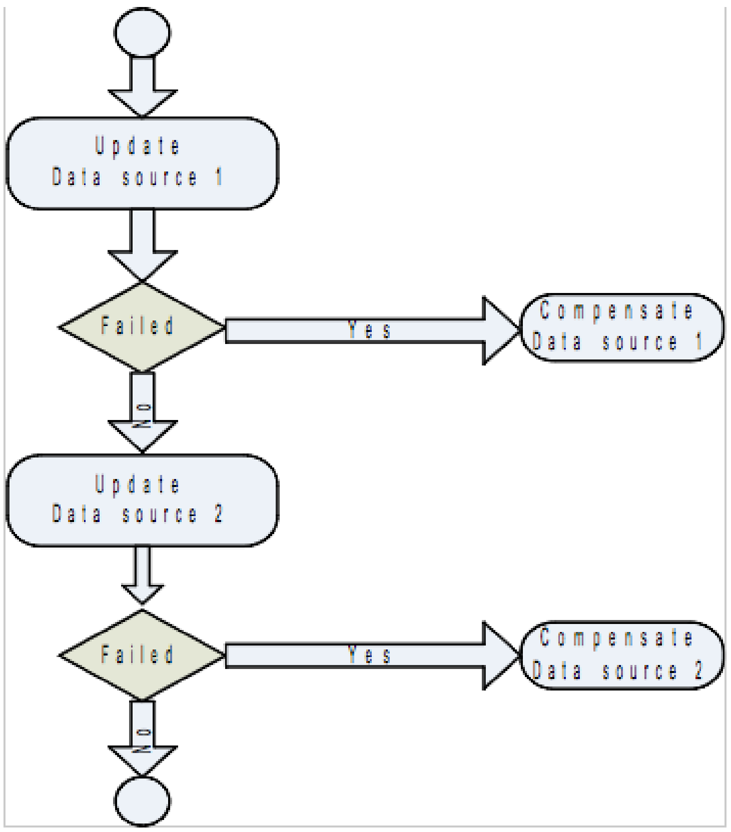
2-Phase Commit

Q5. What are the pros and cons of isolation levels?
A5. The isolation levels **lock database records at different granularity**. For example, a single row, set of rows or the whole table. This type of locking is known as the

"**pessimistic locking**" and you need to apply this type of locking judiciously as it can adversely impact performance. The "pessimistic locking" doesn't work well in a web application, as you don't want to hold on to the locks during the user think time or if a user locks a record and then forgets that s/he locked it, you will have a record that is locked forever.

he best way to handle this problem is to assume that you aren't going to have two people trying to update the same data at the same time too often. You do this by making only one person responsible for updating each data item. You're still going to have the occasional race condition, for example when two or more users update the same record, but you can deal with those on an ad-hoc basis with the help of a version or timestamp column in the database to detect if the record is dirty without needing to lock the record. This approach is known as the "**optimistic locking**". The optimistic locking is more scalable and the ORM tools do support the "**dirty checking**" feature with version numbers or timestamp columns in the database table.

Q6. Can you apply ACID based transaction management to Web services?
A6. Web services are used in business applications for third-party integration. For example, an online travel application calls a number of different external third party applications like an airline ticketing system, a hotel reservation, and a rental car reservation system. All of these services need to succeed, and if one of the services fail, then all the other services must be rolled back. If you use a traditional ACID based transaction management, you would run into the scenario where the third-party resources used by the web services would be locked for an indefinite period of time, until all the other participating services have executed. So, the ACID based transactions are not suited for the long running transactions , and **compensation based transactions** need to be used.



Compensation based transaction management

The **compensation based transaction management** has features such as:

— Managing data consistency without locking the resources. The coordinator (WS-C) is responsible for sending messages back and forth between the transaction participants.

— The participants register themselves with the coordinator to receive messages. The coordinator is also responsible for propagating transactional context information between participants.

— Unlike ACID based transactions where the complete authority is given to a single transaction manager, the transactions are coordinated among the various participants without giving complete authority to a single transaction manager. The coordinators and compensating tasks replace the transaction managers.

— Works well in scenarios where participants' availability or response is not guaranteed.

— The participants are notified via the life cycle methods to take the appropriate course of action. The participants do have life cycle methods like *compensate( )* to be executed when a transaction fails and a *close( )* method to indicate successful transaction completion.

Q7. In a Java enterprise application, which layer does the transaction demarcation?
A7. The service layer. Here is an example with Spring.

Step 1: Define your transaction manager in a Spring context file

```
<bean id="txnManager" class="org.springframework.
..
<tx:annotation-driven transaction-manager="txnMan

```

Step 2: The service class that handles business logic in a protocol agnostic manner. Annotations are used to wire up dependencies. This is the service layer that provides transaction demarcation with @Transactional annotation. **@Transactional** can take properties but we will go with default values, which are "Propagation : Required", "isolation level : default rollback of the underling resource managers", and "Rollback : Any runtime exception triggers a rollback".

```
@Service(value = "myapp_Service")
public class CashForecastServiceImpl implements
{
    @Resource(name = "myapp_Dao")
    private MyAppDao myAppDao;

    @Override
    @Transactional
    public void updatePortfolioSummaries(Portfol
        myAppDao.updatePortfolio(vo.getPortfolio
        myAppDao.updateAccount(vo.getAccount());
        myAppDao.updateTransactions(vo.getAccoun
    }
}
```

Step 3: The DAO class that makes database calls via a JDBC template. The configuration of JDBC template is not shown.

```
@Repository(value = "myapp_Dao")
public class CashForecastDaoImpl implements Cash
{

    @Resource(name = "myapp_JdbcTemplate")
    private JdbcTemplate jdbcTemplateSybase;  //


    public int  updatePortfolio(Account account)
        //.....logic to update database table us
    }

    public int  updateAccount(Portfolio portfoli
        //.....logic to update database table us
    }

    public int  updateTransactions(List<Transact
        //.....logic to update database table us
    }

}
```

Q8. Can you explain how NoSQL databases take a **BASE** approach to data storage, whereas relational database management systems take **ACID** based approach?
A8. You may already know that ways in which relational & NoSQL databases "store and structure" their data are very different. In addition, it is important to keep in mind that the relational databases adhere to ACID standard explained in Q1. NoSQL databases adhere to **BASE** standard described below.

1) Basically Available: Data is replicated and partitioned across multiple nodes on commodity (i.e. cheap) hardware, to ensure it will be available even in the event of multiple failures.

2) Soft state: Unlike ACID-based systems that strive for consistency, a NoSQL database permits data to be in an inconsistent state. This can occur after data is modified but not fully replicated to the other replicated nodes. The replication can usually take a few milliseconds. This approach can give better performance by not having to wait for all the nodes to be updated.

3) Eventual consistency: After it has been fully replicated to all the nodes, the data is brought to a consistent state.

The BASE standards and the different structure to store data allows NoSQL databases to support big data (i.e. in petabytes) and low-latency operations at the expense of not being able to handle granular control to the level that a relational database does in terms of ensuring data integrity and reducing data redundancy and inconsistencies. The moderen NoSQL databases at least partially support the "AID" part of the ACID properties. The modern NoSQL databases do give a choice to configure "strong consistency" on a case by case basis for scenarios requiring strong consistency.

Q9. How will you write **atomically** some data from your code to a file? In other words, Either all of your writes have to make it to the disk, or none of them.
A9. File writes are not atomic, but meta operations like rename, copy, creating a new file, etc are atomic. So, you need to write to a temp file, and once everything is written successfully, rename the file to actual name.

Q10. How will you handle producer-consumer scenario where a consumer polls for availability of a file, whilst the producer is responsible for producing a file?
A10. The key consideration here is that the consumer needs to read the file only once it is fully written and not read a partially read file while it is being written by the producer. This can be achieved as described below.

1. The **producer** writing to a file say customer.csv should create a new empty file named **cusomer.csv.end** file at the completion. This is an empty flag file that signals the file completion.

2. The **consumer** should be polling for presence of a file named **customer.csv.end** to trigger reading from a file named customer.csv by dropping ".end", which is used only as a signal file.

Popular Posts

♦ 11 Spring boot interview questions & answers

828 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

768 views

18 Java scenarios based interview Questions and Answers

400 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

389 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

296 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

286 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

280 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

240 views

001B: ♦ Java architecture & design concepts interview questions & answers

202 views

Bio | **Latest Posts**

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in

2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ ♥ 6 Aspects that can motivate you to fast-track your career & go places as a software engineer

8 Java memory management interview Q&A ›

Posted in Java Key Area Essentials**,** member-paid**,** NoSQL**,** Transaction Management

Tags: Architect FAQs

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from "Blah blah" to "Wow"? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer