**Section A: Core Database Concepts**

**1. What is a Transaction? Explain with an Example.**

A **transaction** in a database is a sequence of operations performed as a single logical unit of work. A transaction ensures that either all operations execute successfully (commit), or none are applied (rollback), preserving data consistency. For example, transferring money between two bank accounts involves deducting an amount from one account and adding it to another. If any part of the process fails, the entire transaction is rolled back to prevent inconsistencies[1](https://en.wikipedia.org/wiki/Database_transaction)[2](https://www.dbvis.com/thetable/database-transactions-101-the-essential-guide/)[3](https://www.geeksforgeeks.org/dbms/transaction-in-dbms/).

**Example in SQL:**

sql

**START** **TRANSACTION**;

**UPDATE** accounts **SET** balance = balance - 100 **WHERE** account\_id = 1;

**UPDATE** accounts **SET** balance = balance + 100 **WHERE** account\_id = 2;

**COMMIT**;

If any update fails, a rollback ensures balances remain unchanged[2](https://www.dbvis.com/thetable/database-transactions-101-the-essential-guide/)[3](https://www.geeksforgeeks.org/dbms/transaction-in-dbms/).

**2. Problems That Arise Without Concurrency Control**

Without proper concurrency control, executing multiple transactions simultaneously can lead to several problems[4](https://www.studocu.com/row/messages/question/8684410/what-problem-can-come-if-there-is-no-concurrency-control-where-multiple-transactions-are-being)[5](https://www.geeksforgeeks.org/dbms/concurrency-problems-in-dbms-transactions/):

* **Lost Update:** Two transactions write to the same data, but only the last update persists, overwriting the other.
* **Dirty Read:** A transaction reads data modified by another transaction that has not yet committed, potentially using uncommitted, erroneous data.
* **Uncommitted Data (Temporary Update):** Transactions read or use data that is later rolled back, causing inconsistency.
* **Unrepeatable Read:** Same query in the same transaction returns different results if another transaction modifies the data in between.
* **Phantom Read:** A transaction reads a set of rows that another transaction modifies (adds or deletes) during its execution, leading to differing results on repeated queries.

**3. What is Locking? How Does It Help Manage Concurrent Transactions?**

**Locking** is a technique used to control access to database resources during concurrent transactions. Locks prevent multiple transactions from simultaneously modifying the same data, preserving consistency and preventing problems like lost updates and dirty reads[6](https://www.ibm.com/docs/en/informix-servers/14.10.0?topic=scope-database-locks)[7](https://www.dbwatch.com/blog/database-locks-how-to-monitor-and-manage-it/).

Types of locks include:

* **Row-level:** Locks individual rows.
* **Table-level:** Locks the entire table.
* **Shared locks:** Allow reading but not writing.
* **Exclusive locks:** Allow writing, blocking other access.

Locks ensure that only one transaction modifies a piece of data at a time, upholding database integrity when many users access data concurrently[7](https://www.dbwatch.com/blog/database-locks-how-to-monitor-and-manage-it/).

**+4. Define Serializability. Why Is It Important?**

**Serializability** is the property that ensures the outcome of executing transactions concurrently is the same as if transactions were run serially, one after the other. It is crucial for correctness, consistency, and data integrity when transactions are concurrent[8](https://www.upgrad.com/blog/serializability-in-dbms/)[9](https://vocal.media/education/understanding-serializability-in-dbms).

* Ensures consistency: The database remains accurate.
* Maintains isolation: Transactions don’t interfere with each other.-
* Guarantees reliable results: Prevents the issues described in concurrency problems.

**5. Two-Phase Locking Protocol (2PL)**

The **Two-Phase Locking Protocol** is a concurrency control method ensuring serializability. It divides the transaction's execution into two phases[10](https://www.tutorialspoint.com/explain-about-two-phase-locking-2pl-protocol-dbms)[11](https://www.scaler.com/topics/two-phase-locking-protocol/):

1. **Growing phase:** The transaction acquires all locks it needs without releasing any.
2. **Shrinking phase:** Once it releases a lock, it cannot acquire new ones.

This protocol enforces a “lock point” after which locks can only be released, guaranteeing that concurrent execution will be serializable and data consistent. Variants like strict and rigorous 2PL add further constraints for enhanced safety[10](https://www.tutorialspoint.com/explain-about-two-phase-locking-2pl-protocol-dbms)[11](https://www.scaler.com/topics/two-phase-locking-protocol/).

**Section B: Advanced Topics (Optional)**

**1. What Is Indexing? Why Is It Used in Databases?**

**Indexing** refers to creating data structures that improve the speed of data retrieval in a database. An index works like a book index — it lets the database find data without reading every row, dramatically improving query performance, especially for large tables[12](https://byjus.com/gate/indexing-in-dbms-notes/)[13](https://dev.to/shriyaexe/understanding-database-indexing-a-guide-with-sql-examples-17ik).

**Benefits:**

* Faster searches and queries
* Enforcing uniqueness (e.g., primary keys)
* Support rapid sorting and filtering

**2. Clustered vs. Non-Clustered Indexes**

| **Feature** | **Clustered Index** | **Non-Clustered Index** |
| --- | --- | --- |
| Data Storage | Sorts and stores table rows in order of the index key | Stores the index separately; uses pointers to data rows |
| Number Allowed | One per table | Multiple allowed |
| Ideal Use | Range queries, primary key | Fast lookups on non-key columns, multiple indexes |
| Speed | Fast for range-based search | Fast for specific value lookups |
| Example | Primary key by default | Explicitly created on columns frequently searched |
| Maintenance | Data rearranged on changes | Slower on updates due to extra structure |
| Disk Space | Does not require extra space | Requires additional storage |
| [14](https://dev.to/devcorner/clustered-vs-non-clustered-index-in-databases-5gfj)[15](https://www.guru99.com/clustered-vs-non-clustered-index.html) |  |  |

**3. What Is a View in SQL? How Does It Differ from a Table?**

A **view** is a virtual table based on a query from one or more tables. It does not store data itself but provides a way to represent data for ease of access or security[16](https://beginnersbook.com/2022/08/difference-between-view-and-table-with-examples/)[17](https://blog.devart.com/difference-between-views-and-tables-in-sql.html):

| **Aspect** | **Table** | **View** |
| --- | --- | --- |
| Data storage | Physically stores data | No data storage—retrieves from tables |
| Persistence | Permanent unless deleted | Temporary—defined by query |
| Modification | Can insert, update, delete rows | Usually read-only (some updates possible) |
| Usage | Holds core data | Presents filtered or joined data |
| Security | No access restriction per se | Can limit access, expose only needed columns |

**4. Data Warehouse and OLAP**

A **data warehouse** is a centralized repository designed for storing integrated data from multiple sources, optimized for analytical reporting and data analysis. OLAP (Online Analytical Processing) refers to techniques that enable complex queries, aggregations, and data insights quickly across large datasets.

* **Data Warehouse:** Stores historical data, supports business intelligence, and is optimized for read-heavy queries.
* **OLAP:** Enables multi-dimensional analysis (like sales by product, region, and time), supporting decision-making.

**5. Briefly Describe NoSQL Databases. How Do They Differ from Relational Databases?**

**NoSQL databases** are non-relational systems designed for specific use cases such as large-scale data, flexible schemas, or high scalability. Key differences from relational databases:

* **Schema Flexibility:** NoSQL often uses flexible, dynamic schemas.
* **Data Models:** Includes key-value, document, column-family, and graph databases.
* **Scalability:** Designed for horizontal scaling and distributed architectures.
* **Transactions:** Often trade complex transactions for performance, with some NoSQL systems offering limited or eventual consistency rather than strict ACID properties.

In contrast, **relational databases** enforce strict schemas, support complex SQL queries, and provide strong ACID guarantees, making them ideal for traditional business applications.

1. <https://en.wikipedia.org/wiki/Database_transaction>
2. <https://www.dbvis.com/thetable/database-transactions-101-the-essential-guide/>
3. <https://www.geeksforgeeks.org/dbms/transaction-in-dbms/>
4. <https://www.studocu.com/row/messages/question/8684410/what-problem-can-come-if-there-is-no-concurrency-control-where-multiple-transactions-are-being>
5. <https://www.geeksforgeeks.org/dbms/concurrency-problems-in-dbms-transactions/>
6. <https://www.ibm.com/docs/en/informix-servers/14.10.0?topic=scope-database-locks>
7. <https://www.dbwatch.com/blog/database-locks-how-to-monitor-and-manage-it/>
8. <https://www.upgrad.com/blog/serializability-in-dbms/>
9. <https://vocal.media/education/understanding-serializability-in-dbms>
10. <https://www.tutorialspoint.com/explain-about-two-phase-locking-2pl-protocol-dbms>
11. <https://www.scaler.com/topics/two-phase-locking-protocol/>
12. <https://byjus.com/gate/indexing-in-dbms-notes/>
13. <https://dev.to/shriyaexe/understanding-database-indexing-a-guide-with-sql-examples-17ik>
14. <https://dev.to/devcorner/clustered-vs-non-clustered-index-in-databases-5gfj>
15. <https://www.guru99.com/clustered-vs-non-clustered-index.html>
16. <https://beginnersbook.com/2022/08/difference-between-view-and-table-with-examples/>
17. <https://blog.devart.com/difference-between-views-and-tables-in-sql.html>
18. <https://fauna.com/blog/database-transaction>
19. <https://www.datacamp.com/tutorial/sql-transactions>
20. <https://dev.to/aharmaz/database-transactions-basic-concepts-2gl2>