<https://dev.mysql.com/downloads/windows/installer/8.0.html>

<https://introcs.cs.princeton.edu/java/85application/jar/jar.html>

<https://www.youtube.com/watch?v=9-dDfgOprYU>

<https://www.tutorialspoint.com/json/json_java_example.htm>

<https://www.guru99.com/json-tutorial-example.html>

<https://support.smartbear.com/alertsite/docs/monitors/web/selenium/export-eclipse-java-project-as-runnable-jar.html>

https://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html

<https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm>

0. What is SQL?

->SQL stands for Structured Query Language

->SQL lets you access and manipulate databases

->SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

->SQL can execute queries against a database

->SQL can retrieve data from a database

->SQL can insert records in a database

->SQL can update records in a database

->SQL can delete records from a database

->SQL can create new databases

->SQL can create new tables in a database

->SQL can create stored procedures in a database

->SQL can create views in a database

->SQL can set permissions on tables, procedures, and views

RDBMS:

RDBMS stands for Relational Database Management System.

RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

Look at the "Customers" table:

SELECT \* FROM Customers;

Every table is broken up into smaller entities called fields. The fields in the Customers table consist of CustomerID, CustomerName, ContactName, Address, City, PostalCode and Country. A field is a column in a table that is designed to maintain specific information about every record in the table.

A record, also called a row, is each individual entry that exists in a table. For example, there are 91 records in the above Customers table. A record is a horizontal entity in a table.

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

The various types of Keys are:

o   **Super Key** - A set of attributes (one or more) that collectively identifies an entity in an entity set. It’s any combination of column(s) for which that combination of values will be unique across all rows in a table.

   Example: EmployeeID is always unique in Employee table. Combination of EmployeeID and EmployeeName there will be no two rows in the table that share the exact same combination of values, because the EmployeeID will always be unique and different for each row.

o    **Candidate Key** - A minimal super key is called a candidate key. An entity set may have more than one candidate key. It is minimum number of columns that can be used to uniquely identify a single row. In other words, the minimum number of columns, which when combined, will give a unique value for every row in  the table.

Example: EmployeeID is always unique in Employee table.

o   **Primary Key** - Only one super key is used to uniquely identify a single row of table is called primary key.

Example: EmployeeID is always unique in Employee table.

o    **Foreign Key** -  It is used to define relationship between two tables.

o    **Alternate Key** - Any table have more than one candidate key, then after choosing primary key from those candidate key, rest of candidate keys are known as an alternate key of that table.

   Example: We have a table named Employee which has two columns EmployeeID and EmployeeEmail, both have not null attributes and unique value. So both columns are treated as candidate key. Now we make EmployeeID as a primary key to that table then EmployeeEmail is known as alternate key.

o   **Composite Key** - Create keys with more than one column then that key is known as composite key. It is used to uniquely identify a single row of table.

     Example: Combination of (EmployeeID, DegreeID) columns uniquely identified as single row of EmployeeDegree table.

* 1. Why SQL preferred instead of file saving?

This is because modern DBMS has focused on improving the storage of large blobs.

Pros of the File system:

Performance can be better than doing it in db. To justify this, If you store large files in db then it may slow down the performance because a simple query to retrieve the list of files or filename will also load the file data if you used Select \* in your query. While Files system accessing a file is quite simple and light weight.

Saving the files and downloading them in the file system is much simpler than database since a simple Save as function will help you out. Downloading can be done by addressing an URL with the location of the saved file.

Migrating the data is an easy process here. You can just copy and paste the folder to your desired destination while ensuring that write permissions are provided to your destination.

Cost effective as It is Economical in most of the cases to expand your web server rather than paying for certain Databases.

Easy to migrate it to Cloud storage like Amazon S3 or CDNs etc in the future.

Cons of the File system:

Loosely packed. No ACID (Atomicity, Consistency, Isolation, Durability) operations relational mapping which mean there is no guarantee. Consider a scenario if your files are deleted from the location manually or by some hacking dudes, you might not know whether the file exists or not. Painful right?

Low Security. Since your files can be saved in a folder where you should have provided write permissions, it is prone to safety issues and invites troubles like hacking. So it is best to avoid saving in fs if you cannot afford to compromise in terms of security.

When is it most preferred:

If your application is liable to handle Large files of size more than 5MB and the massive number say thousands of file uploads.

If your application can land you on cloud nine, I mean your application will have a large number of users.

Best way to do: Though File System comes with some cost and certain cons, A good Internal Folder Structure and choosing a folder location which may be a little difficult to access by others.

Pros of Database:

ACID consistency which includes a rollback of an update that is complicated when the files are stored outside the database.

Files will be in sync with the database so cannot be orphaned from it which gives you an upper hand in tracking transactions.

Backups automatically include file binaries.

More Secure than saving in a File System.

Cons of Database:

You may have to convert the files to blob in order to store it in db.

Database Backups will become more hefty and heavy.

Memory ineffective. To add more, often RDBMS’s are RAM driven. So all data has to go to RAM first. Yeah, that’s right. Had you ever thought about what happens when an RDBMS has to find and sort data? RDBMS tracks each data page even lowest amount of data read/written, and it has to track if it’s in memory or if it’s on disk if it’s indexed or sorted physically etc.

P.S: I have skipped some contradictory points to curtail the content because while Comparing two things we often end up finding the Pros and Cons of one will be the opposite of other.

When is it most preferred:

If your user’s file needs to be more tightly coupled, secured and confidential.

If your application will not demand a large number of files from a large number of users.

1. Create Document about JOIN in Mysql

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

eg: SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate

FROM Orders

INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

(INNER) JOIN: Returns records that have matching values in both tables

LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table

RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table

FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table

2)Various SQL Db

Oracle 12c. It's no surprise that Oracle is consistently at the top of lists of popular databases. ...

MySQL. MySQL is one of the most popular databases for web-based applications. ...

Microsoft SQL Server. ...

PostgreSQL. ...

MongoDB. ...

MariaDB. ...

DB2. ...

SAP HANA.

3)Diff between SQL vs NOSQL

SQL Databases

Also known as relational databases, define and manipulate data based on structured query language (SQL). These are most popularly used and useful for handling structured data that organizes elements of data and standardizes how they relate to one another and to different properties.

NoSQL Databases

Also known as non-relational databases, allow you to store and retrieve unstructured data using a dynamic schema. NoSQL is popularly used for its flexible ability to create a unique structure, and can be document, graph, column, or even KeyValue organized as a data structure.

SQL has had a large lead over the non-relational alternatives for decades, but NoSQL is quickly closing the gap with popular databases such as MongoDB, Redis, and Cassandra. Though many organizations are choosing to migrate from legacy databases, such as Oracle, not all are moving to NoSQL way. Based on our findings, SQL still holds 60% with rising demand for systems such as PostgreSQL:

4)ACID property

Next →← Prev

ACID Properties in DBMS

DBMS is the management of data that should remain integrated when any changes are done in it. It is because if the integrity of the data is affected, whole data will get disturbed and corrupted. Therefore, to maintain the integrity of the data, there are four properties described in the database management system, which are known as the ACID properties. The ACID properties are meant for the transaction that goes through a different group of tasks, and there we come to see the role of the ACID properties.

In this section, we will learn and understand about the ACID properties. We will learn what these properties stand for and what does each property is used for. We will also understand the ACID properties with the help of some examples.

ACID Properties

The expansion of the term ACID defines for:

ACID Properties in DBMS

1) Atomicity: The term atomicity defines that the data remains atomic. It means if any operation is performed on the data, either it should be performed or executed completely or should not be executed at all. It further means that the operation should not break in between or execute partially. In the case of executing operations on the transaction, the operation should be completely executed and not partially.

Example: If Remo has account A having $30 in his account from which he wishes to send $10 to Sheero's account, which is B. In account B, a sum of $ 100 is already present. When $10 will be transferred to account B, the sum will become $110. Now, there will be two operations that will take place. One is the amount of $10 that Remo wants to transfer will be debited from his account A, and the same amount will get credited to account B, i.e., into Sheero's account. Now, what happens - the first operation of debit executes successfully, but the credit operation, however, fails. Thus, in Remo's account A, the value becomes $20, and to that of Sheero's account, it remains $100 as it was previously present.

ACID Properties in DBMS

In the above diagram, it can be seen that after crediting $10, the amount is still $100 in account B. So, it is not an atomic transaction.

The below image shows that both debit and credit operations are done successfully. Thus the transaction is atomic.

ACID Properties in DBMS

Thus, when the amount loses atomicity, then in the bank systems, this becomes a huge issue, and so the atomicity is the main focus in the bank systems.

2) Consistency: The word consistency means that the value should remain preserved always. In DBMS, the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always. In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.

Example:

ACID Properties in DBMS

In the above figure, there are three accounts, A, B, and C, where A is making a transaction T one by one to both B & C. There are two operations that take place, i.e., Debit and Credit. Account A firstly debits $50 to account B, and the amount in account A is read $300 by B before the transaction. After the successful transaction T, the available amount in B becomes $150. Now, A debits $20 to account C, and that time, the value read by C is $250 (that is correct as a debit of $50 has been successfully done to B). The debit and credit operation from account A to C has been done successfully. We can see that the transaction is done successfully, and the value is also read correctly. Thus, the data is consistent. In case the value read by B and C is $300, which means that data is inconsistent because when the debit operation executes, it will not be consistent.

4) Isolation: The term 'isolation' means separation. In DBMS, Isolation is the property of a database where no data should affect the other one and may occur concurrently. In short, the operation on one database should begin when the operation on the first database gets complete. It means if two operations are being performed on two different databases, they may not affect the value of one another. In the case of transactions, when two or more transactions occur simultaneously, the consistency should remain maintained. Any changes that occur in any particular transaction will not be seen by other transactions until the change is not committed in the memory.

Example: If two operations are concurrently running on two different accounts, then the value of both accounts should not get affected. The value should remain persistent. As you can see in the below diagram, account A is making T1 and T2 transactions to account B and C, but both are executing independently without affecting each other. It is known as Isolation.

ACID Properties in DBMS

4) Durability: Durability ensures the permanency of something. In DBMS, the term durability ensures that the data after the successful execution of the operation becomes permanent in the database. The durability of the data should be so perfect that even if the system fails or leads to a crash, the database still survives. However, if gets lost, it becomes the responsibility of the recovery manager for ensuring the durability of the database. For committing the values, the COMMIT command must be used every time we make changes.

Therefore, the ACID property of DBMS plays a vital role in maintaining the consistency and availability of data in the database.

Thus, it was a precise introduction of ACID properties in DBMS. We have discussed these properties in the transaction section also.

5) Class loader

The Java Class Loader is a part of the Java Runtime Environment that dynamically loads Java classes into the Java Virtual Machine. The Java run time system does not need to know about files and file systems because of class loaders. Java classes aren't loaded into memory all at once, but when required by an application.

6)DAO

In computer software, a data access object (DAO) is a pattern that provides an abstract interface to some type of database or other persistence mechanism. By mapping application calls to the persistence layer, the DAO provides some specific data operations without exposing details of the database.

7) POJO/Java Beans

As we know that in Java POJO refers to the Plain old Java object. POJO and Bean class in Java shares some common features which are as follows −

Both classes must be public i.e accessible to all.

Properties or variables defined in both classes must be private i.e. can't be accessed directly.

Both classes must have default constructor i.e no argument constructor.

Public Getter and Setter must be present in both the classes in order to access the variables/properties.

The only difference between both the classes is Java make java beans objects serialized so that the state of a bean class could be preserved in case required.So due to this a Java Bean class must either implements Serializable or Externalizable interface.

Due to this it is stated that all JavaBeans are POJOs but not all POJOs are JavaBeans.

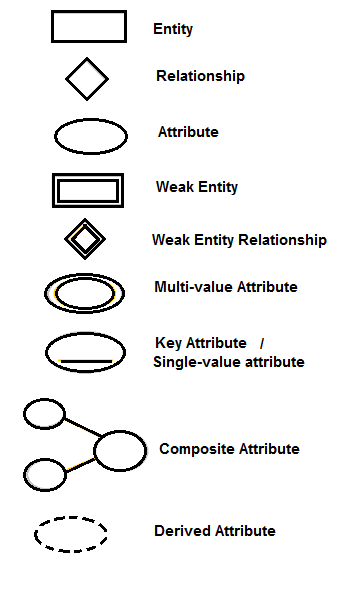
8) What is Entity?

**Entity** – **Entity** can be real-world object. It is a collection of related attributes or properties. Example: Employee, Department, etc. o Strong **Entity** –An **entity** that has a primary key is called as Strong **entity**. Rectangle represents strong **entity**.

|  |  |  |
| --- | --- | --- |
| **#** | **Strong Entity** | **Weak Entity** |
| 1 | It has its own primary key. | It does not have sufficient attributes to form a primary key on its own. |
| 2 | It is represented by a rectangle. | It is represented by a double rectangle. |
| 3 | It contains a primary key represented by an eclipse with underline. | It contains partial key or discriminator represented by an eclipse with dashed underline. |
| 4 | The member of strong entity set is called as dominant entity set | The member of weak entity set is called as subordinate entity set. |
| 5 | The primary key is one of its attributes which uniquely identifies its member. | The primary key of weak entity set is combination of partial key and primary key of the strong entity set. |
| 6 | The relationship between two strong entity set is represented by a diamond symbol. | The relationship between one strong and weak entity set is represented by a double diamond symbol. |
| 7 | The line connecting strong entity set with identifying the relationship is single. | The line connecting weak entity set with identifying the relationship is double. |
| 8 | Total participation in the relationship may or may not exists | Total participation in the identifying relationship always exists. |

ER Diagram:

Symbols and Notations for ER Diagram are as follows.

[m](https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm)[](https://4.bp.blogspot.com/-kW1jr530B9k/WCbTTWJj3SI/AAAAAAAAANU/AQk-dvK70qAyMz_KEYWwMqRd5be1dQRwwCEw/s1600/ERD_Symbols.png)

JDBC Drivers

**Java Database Connectivity (JDBC)** is an application programming interface (API) for the programming language Java, which defines how a client may access any kind of tabular data, especially relational database. It is part of Java Standard Edition platform, from Oracle Corporation. It acts as a middle layer interface between java applications and database.

The JDBC classes are contained in the Java Package **java.sql** and **javax.sql**.  
JDBC helps you to write Java applications that manage these three programming activities:

1. Connect to a data source, like a database.
2. Send queries and update statements to the database
3. Retrieve and process the results received from the database in answer to your query

**Structure of JDBC**



**JDBC Drivers**

JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand. There are 4 types of JDBC drivers:

1. Type-1 driver or JDBC-ODBC bridge driver
2. Type-2 driver or Native-API driver
3. Type-3 driver or Network Protocol driver
4. Type-4 driver or Thin driver

**Type-1 driver**

Type-1 driver or JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. Type-1 driver is also called Universal driver because it can be used to connect to any of the databases.

* As a common driver is used in order to interact with different databases, the data transferred through this driver is not so secured.
* The ODBC bridge driver is needed to be installed in individual client machines.
* Type-1 driver isn’t written in java, that’s why it isn’t a portable driver.
* This driver software is built-in with JDK so no need to install separately.
* It is a database independent driver.

**Type-2 driver**

The Native API driver uses the client -side libraries of the database. This driver converts JDBC method calls into native calls of the database API. In order to interact with different database, this driver needs their local API, that’s why data transfer is much more secure as compared to type-1 driver.

* Driver needs to be installed separately in individual client machines
* The Vendor client library needs to be installed on client machine.
* Type-2 driver isn’t written in java, that’s why it isn’t a portable driver
* It is a database dependent driver.

**Type-3 driver**

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. Here all the database connectivity drivers are present in a single server, hence no need of individual client-side installation.

* Type-3 drivers are fully written in Java, hence they are portable drivers.
* No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
* Network support is required on client machine.
* Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.
* Switch facility to switch over from one database to another database.

**Type-4 driver**

Type-4 driver is also called native protocol driver. This driver interact directly with database. It does not require any native database library, that is why it is also known as Thin Driver.

* Does not require any native library and Middleware server, so no client-side or server-side installation.
* It is fully written in Java language, hence they are portable drivers.

**Which Driver to use When?**

* If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is type-4.
* If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.
* Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.
* The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

**Operations in Transaction-**

The main operations in a transaction are-

1. Read Operation
2. Write Operation

**1. Read Operation-**

* Read operation reads the data from the database and then stores it in the buffer in main memory.
* For example- **Read(A)** instruction will read the value of A from the database and will store it in the buffer in main memory.

**2. Write Operation-**

* Write operation writes the updated data value back to the database from the buffer.
* For example- **Write(A)** will write the updated value of A from the buffer to the database.

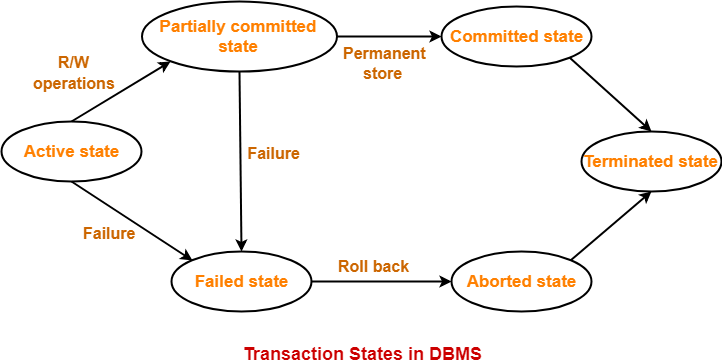
**Transaction States-**

A transaction goes through many different states throughout its life cycle.

These states are called as **transaction states**.

Transaction states are as follows-

1. Active state
2. Partially committed state
3. Committed state
4. Failed state
5. Aborted state
6. Terminated state



**1. Active State-**

* This is the first state in the life cycle of a transaction.
* A transaction is called in an **active state** as long as its instructions are getting executed.
* All the changes made by the transaction now are stored in the buffer in main memory.

**2. Partially Committed State-**

* After the last instruction of transaction has executed, it enters into a **partially committed state**.
* After entering this state, the transaction is considered to be partially committed.
* It is not considered fully committed because all the changes made by the transaction are still stored in the buffer in main memory.

**3. Committed State-**

* After all the changes made by the transaction have been successfully stored into the database, it enters into a **committed state**.
* Now, the transaction is considered to be fully committed.

**NOTE-**

* After a transaction has entered the committed state, it is not possible to roll back the transaction.
* In other words, it is not possible to undo the changes that has been made by the transaction.
* This is because the system is updated into a new consistent state.
* The only way to undo the changes is by carrying out another transaction called as **compensating transaction** that performs the reverse operations.

**4. Failed State-**

* When a transaction is getting executed in the active state or partially committed state and some failure occurs due to which it becomes impossible to continue the execution, it enters into a **failed state**.

**5. Aborted State-**

* After the transaction has failed and entered into a failed state, all the changes made by it have to be undone.
* To undo the changes made by the transaction, it becomes necessary to roll back the transaction.
* After the transaction has rolled back completely, it enters into an **aborted state**.

**6. Terminated State-**

* This is the last state in the life cycle of a transaction.
* After entering the committed state or aborted state, the transaction finally enters into a **terminated state** where its life cycle finally comes to an end.