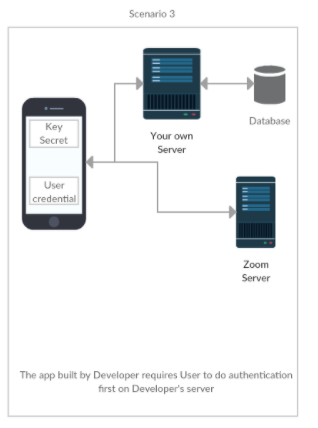
**Virtual Classroom Dashboard 🡪 Research Documentation:**

**01/03/2021 – 01/09/2021 🡪 Week 1**

**Zoom API SDK Client:** [**https://marketplace.zoom.us/docs/sdk/custom/introduction**](https://marketplace.zoom.us/docs/sdk/custom/introduction)

* Zoom API uses Zoom access tokens: investigate Access Credentials for zoom.
* Can build modules to enable meetings within a web browser.
* Access Credentials: SDK key & Secret.
* Registering and activating any Basic Zoom account will automatically provide free-trial Developer access to the Zoom API and SDKs.
* Use free trial to test the services and SDK functionality
* User Tokens and Zoom Access Tokens are required to start a meeting on behalf of a [Non-login user](https://marketplace.zoom.us/docs/sdk/native-sdks/user-login). These dual tokens are required for additional layers of security.
* Web SDK is authenticated using an API key and Secret instead of SDK.
* For Web SDK: [*Create a JWT App*](https://marketplace.zoom.us/docs/guides/getting-started/app-types/create-jwt-app)*on the Marketplace.*
* User Tokens: used to start meetings for users.
* Requesting user tokens need to send GET requests with a userID to /users/{userId}/token
* UserID is either User API or user email.
* More on Access Credentials: <https://marketplace.zoom.us/docs/sdk/native-sdks/credentials>
* Login User/SSO User – Each person needs school credentials if university or personal credentials.
* Login SSO is more beneficial User Type
* Connection type:



* How to Integrate with web App: <https://marketplace.zoom.us/docs/sdk/native-sdks/web>
* Chrome, Firefox, Edge all work with all the features within Zoom. Safari and Internet Explorer have limitations.

**AWS vs Azure hosting:** [**https://stackify.com/azure-vs-aws-comparison/**](https://stackify.com/azure-vs-aws-comparison/) **&** [**https://insanelab.com/blog/web-development/microsoft-azure-vs-amazon-web-services/**](https://insanelab.com/blog/web-development/microsoft-azure-vs-amazon-web-services/)

* AWS categorizations: content delivery and storage, compute, networking, and database.
* Azure categorizations: data management and databases, compute, networking, and performance.
* Amazon includes identity and security services such as key storage and active directory.
* Amazon also includes AWS Config, Cloudtrail, and Cloudwatch.
* Azure includes security and management tools such as Active Directory, Azure Active Directory, Multi-Factor Auth, and Azure monitoring and performance tweaks.
* Azure offers an easy-to-use Hybrid clouds and substantial support.
* Azure: Windows Server, SQL Server, Exchange, etc..
* Azure makes simple deployment for .Net apps.
* AWS is great for .Net as well but only if a certain AWS feature is needed.
* Since .NET is easier to integrate with Azure prob best to use Azure.

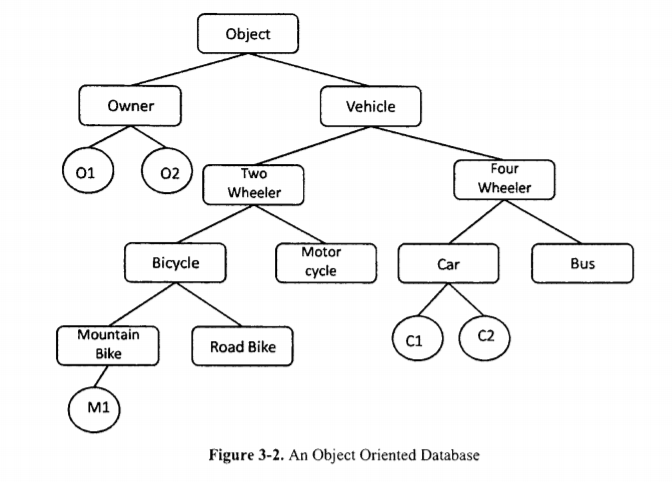
**Implementing large scale databases: LinkedIn Learning video 🡪 Database Foundations Core Concepts & Elmasri, Ramez, and Shamkant Navathe. *Fundamentals of Database Systems*. 7th ed., Pearson, 2015 &** [**https://www.freecodecamp.org/news/database-indexing-at-a-glance-bb50809d48bd/**](https://www.freecodecamp.org/news/database-indexing-at-a-glance-bb50809d48bd/) **&** **Self-adjusting multi-granularity locking protocol for object-oriented databases**

* Databases are highly structured data files that allow data input, organization, and retrieval.
* We use tables for the storing, sorting, and filtering of the data.
* A database is just a structured data storage system
* DBMS 🡪 structural rules, data entry, and data protection
* Retrieving data 🡪 creates one or more indexes, queries data
* Managing Databases 🡪 DBMS supports highly structed and efficient data
* Flat File DB 🡪 2-D tables, rows, and columns, stored as delimited text files
* Flat File Structure 🡪 rows store records(individual items), columns store attributes.
* A simple data structure that follows basic organizational principles.
* Hierarchical Databases 🡪 consists of tables that are related by some piece of data.
* Limitations are when they need to display more complicated relationships.
* Relational Databases 🡪 No restrictions and is a combination of flat file and hierarchical databases.
* Uses unique identifiers keys: Primary, foreign, candidate, and super keys
* Foreign keys reference primary keys in another table.
* Database Fundamentals🡪 data types (Consistent data is entered): an attribute must only have a single data type, Constraints(no dup values, rules, default values, NULL values, etc) What are the rules of the data, Referential Integrity 🡪 Ensures the validity and completeness of the data, SQL 🡪 language that is used to create structures, update, modify, delete data
* Microsoft uses T-SQL which is just their version of SQL
* Database Server 🡪 dedicated or virtual machine
* Users log in over a network and commands are used remotely, processed, and the results are returned by the DBMS
* Server Hardware 🡪 extremely fast r & w disk speeds, large amounts of memory, and fast network connections
* Multiple server machines tired together in clusters and physical proximity not needed.
* SQL Server Management Studios(SSMS) – connect to the instance of SQL server
* Views 🡪 Multiple tables data no actual data storage.
* “A proper index can be created only when you know exactly what your query & data access patterns look like”(free code camp)
* Indexing maps search keys that correspond to data on disks by using in memory & on disk data structures.
* Possible keys 🡪 what all available indices are there
* Key column 🡪 which index is going to be used
* In ANSI SQL standards 🡪 PK’s should be comparable
* Define id field as AUTO INCREMENT
* EXTENDED 🡪 shows all indices not usable
* Differences between key and index 🡪 Key: constraint on the behavior of the column & index: special data structure that facilitates data.
* Non\_unique🡪 1 not unique, 0 is unique
* Key\_name 🡪 name of index, PK is always PRIMARY
* Seq\_in\_index 🡪 sequence # of col in index, if multiple columns are in the index them, they will be assigned based on how they were ordered during the creation.
* Collation 🡪 how col is sorted with the index. A ascending, D descending, NULL not sorted
* Cardinality 🡪 est. # of unique values in index. More Cardinality: higher chance query optimizer will pick index for queries
* Sub\_part 🡪 index prefix, NULL if the entire col is indexed. Otherwise it shows the # of indexed bytes if partially indexed.
* Packed 🡪 how key is packed, NULL if not
* NULL 🡪 YES if may contain null values and blank if not
* Index\_type 🡪 which indexing structure is used: BTREE, HASH, RTREE, FULLTEXT
* Clustered Index 🡪 collocated with data in the same table or same disk file & can be a BTREE whose leaf nodes are the actual data blocks on disk.
* This kind of index physically organizes the data on disk as per the logical order of the index key.
* Physical data organization 🡪 data is organized on disks across thousands or more of disk/data blocks.
* Cluster indexes do not require that all the disk blocks are contagiously stored.
* Advantage of Clustered Indexes 🡪 faster, possibility data is buffered in memory.
* Reduces # of disk IO by collocating related data as much as possible causing improvement in performance.
* If queries are based on PK’s performance will be faster.
* Constraints with Clustered indexes 🡪 impacts physical organization of data so only one clustered index per table is allowed.
* Structure of PK Index(Clustered) 🡪 indexes are maintained as B+ Tree on disk & in memory and stored in blocks on disks.’
* Advantages of Primary Index 🡪 locality of fata can be provided & any query that uses PK is faster.
* Disadvantages 🡪 since uses direct reference to data blocks through virtual addressing space and dick blocks are physically organized, every time the OS does some disk page split due to DML operations(SELECT, UPDATE, DELETE), the P index needs to be updated.
* DML operations places pressure on the performance of the p index.
* Secondary indexing 🡪 any indexing but clustered indexing. Does not impact physical storage locations.
* EX of Query: CREATE INDEX secondaryIndexName ON tableName(col)
* Structure of secondary index 🡪 also uses B+ Tree and is sorted as per the key. Leaf nodes contain the primary index.
* Secondary index references primary index
* Secondary Index 🡪 Primary Index 🡪 Data Blocks
* Advantages 🡪 can create infinite secondary indices. (Not recommended to use a lot, only what is required)
* Disadvantages 🡪 DML operations(DELETE & INSERT), secondary index also needs to be updates.
* Secondary indices can essentially create issues.
* UNIQUE key index 🡪 can contain null values and are similar to primary indices.
* Composite index 🡪 defines indices on multiple columns(MAX = 16)
* Columns using composite indices are concatenated together and are stored in sorted order in a B+ Tree.
* When would you use a composite index?
* If certain fields appear together in multiple queries
* If there is a high cardinality
* Covering Index 🡪 a type of composite index
* SELECT & WHERE are part of composite index
* Indexing Guidelines 🡪 indices take up more memory so use only the number of indices you need, write operations are costly with indices, Cardinality is important, and indices require maintenance.
* OODBMS 🡪 Object-Oriented Database Management System: supports modeling and creation of data as objects
* Uses Objects(“Real world entities”), Object identifiers(“unique identifier associated with the object”), Class(“objects are instances of a class and classes consist of methods and attributes”), and Class Inheritance(“use of subclasses that belong to one superclass – the subclasses inherit attributes from the superclass.”)
* Concurrency Control 🡪 Updates of data must be done in concurrent executions and failures
* A 🡪 atomicity: effects of transactions are reflected in the database or none are
* C 🡪 consistency: ensures the state of the database is consistently maintained before and after transactions
* I 🡪 isolation: concurrent transactions are isolated from eachother
* D 🡪 durability: no transactions are lost after system failures
* Concurrency Control in RDB 🡪 responsible for resolving conflicts among transactions
* Serializability guarantee consistency
* “A Lock is a database system object associated with a database object that prevents undesired operations of other transactions by blocking them”.

**01/10/2021 – 01/16/2021** 🡪 **Week 2 (6 hours): 2 hour(s) down**

**Database Continuation from previous week: Self-adjusting multi-granularity locking protocol for object-oriented databases**

* Shared and Exclusive locking 🡪 with share a data item can be read but cannot write and with exclusive locks it can both read and write items.
* Deadlock protocol 🡪 2 or more transactions are waiting on each other to finish so there is a protocol to prevent such an event from occurring
* Two-phase locking 🡪 growing and shrinking phases
* Growing: locks accumulated, shrinking locks released
* Rigorous two-phase locking 🡪 locks held until the transactions are committed
* Two-phase locking in general is to help prevent deadlock and is known as a pessimistic method because it requires locks to be acquired before given access to data
* Timestamping is another method that prevents deadlocks where a number is assigned to each transaction before anything is executed.
* Concurrency Control with Object Oriented DB’s 🡪 database functionality combined with OO concepts
* Design & Development of SAML 🡪 SAML: self-adjusting multi-granularity locking protocol improves concurrency control of OODB’s
* Adjusts locking granularity bases on requirements of transaction
* 3 levels of granularity locking: instance lock, class lock, and hierarchy lock
* RDBMS lock hierarchy 🡪 Database – Areas – Relations – Tuples
* Pg. 19 OODBMS



* Explicit lock 🡪 explicitly requested by and granted to transactions directly
* Implicit locks 🡪 need to be acquired for either direction of the hierarchy of the database

**Advanced Threading in C#:** [**https://www.linkedin.com/learning/advanced-threading-in-c-sharp/two-way-signaling?u=52983649**](https://www.linkedin.com/learning/advanced-threading-in-c-sharp/two-way-signaling?u=52983649)

* Multithreaded code needs thread safety
* Thread safety 🡪 Code is thread safe if shared data structures are modified to ensure all threads behave properly.
* Items must be locked before making any modifications
* Code is all about predictability
* All tasks run concurrently, and the order doesn’t matter because one task can take precedent over the other
* In the .Net Framework the static methods are thread safe but not all static method that you create are thread safe.
* When you develop a static method make sure you create them to be thread safe
* Thread Affinity 🡪 thread that instantiates an object is the only thread that can access its members
* With this you may not need to use a lock to access objects inside the thread
* Synchronizing among threads is signaling.
* AutoResetEvent 🡪 used when a thread needs exclusive access to a resource, one thread access at a time, automatically closes
* Two-way signaling 🡪 more interaction between the threads.