Cameron Khan, KHA16607446

Databases Assignment 2 Report

NoSQL:

A Dynamic Schema is a form of schema (database blueprint) that is well used in NoSql and one of it's key distinctions on what the term means compared to a Fixed Schema was actually mentioned previously. The idea of a "database blueprint" in a Dynamic Schema is almost non-existent as one of the benefits of Dynamic Schemas, meaning your not confined to a predefined structure as you would be when working with/ to a Fixed Schema, subquenstionaly this also means you wouldn't have to predefine things such as the datatypes of values before implementation either. NoSQL broadens the SQL means of querying data, therefore NoSQL is best suited to use where it is querying a large amount of data thanks to the fact NoSql has low complexity meaning it may have better means of scalability and speed. Put shortly a Dynamic Schema means a database without a fixed blueprint. This has a big advantage, take my database for example and let's say customers wanted the company to label more information about the accessories solt by the company, the accessory table would need editing to store that information. If I was wanting to add a new column that holds the information of the accessories weight or any new value doing this would change the Schema (blueprint) of the database using a Dynamic Schema I would not have a blueprint to affect or damage making it easier to implement things like the above or any other change for a matter of fact without having to alter/ fix/ update the blueprint.

Furthermore my database also has a lot of predefined structure thanks to things such as data types and integrity constraints such as foreign keys that have been created linking one piece of data from one table to another, an example of this is how all of my weak entities (purchase order, customer order) rely on a piece of information from my strong entities (customers, supplier) because of pre-defined relationships like these running queries and trying to retrieving data from the table could be effected in a negative way making the task's slow and sluggish. Where as if you was to denormalize the table and use a Dynamic Schema to store all the information in a single place no data would be duplicated or linked to one another in a constrained way this in itself is a big benefit but it also means retrieving data will also be fairly faster. However seeing as my database is fairly small this benefit of a Dynamic Schema isn't really needed nor would it be used to its fullest potential seeing as it wouldn't be as much of a slow process to fix/ update the current Fixed Schema than it would be with a much larger database.

You could also say this so called benefit of being able to alter tables and inject data with ease could also be a hindrance in situations where a Structured database is wanted or needed by a client that needs to fit a key blueprint or structure (for any which reason) NoSql however is not made out to be a better Schema more an alternative if it better suites the databases needs. Another rather pressing issue that could arise with using NoSql and Dynamic Schemas is that fact that the lack of constraints and structure could lead to the loss of quality and reliability of the data stored in the database meaning the information it provides can become "untrustworthy" and a lot harder to control as things such as constraints can be important in keeping the data from becoming volatile and hard to track/ see the reason why each bit of data is there.

Database Security:

An Injection Attack is a means of using user-defined input along with destructive and dangerous SQL commands and queries with the end goal of being able to manipulating a database to allow such things as access to parts of a database not intended to be seen that may be sensitive and private information for example data from my database that should be kept hidden for their safety is the Lincoln Gardens Staff's Home Addresses and more importantly there National Insurance Number. There are many other risk's than can arise from an Injection attack including things such as deletion of data and tables, database editing meaning the attacker has the ability to edit any part of the database they want and may even reach a point of risk of the attacker reaching a point where they gain administrative rights over a database which could lead to such things as the attacker denying anyone else access to the database.

One good example of a query that can be used as a means of injection attack that could be used on my database is the use of injection during logging into my database allowing the attacker to simply view or gain access to a database they otherwise should not be able to log-into. This can be done with the injection of a simple Select query. Take for example a Select * From users Where login = User-Defined Username and Password = User-Defined Password at the moment this piece of code is fine but in the same query you need to state the username and password to log in like so;

```
$login = “Username";
$password = “Password";
```

Here normally you would enter your log-in details of course an attacker would not have a login however with my database not having any protection really implemented they could revert to a method of tricking this security measure with a fairly simple bit of code where you would supply the login field with ' or 1=1 -- like so;

```
$login = " '  or 1=1 -- ";
```

And the select query;

```
SELECT * FROM users WHERE login = '' or 1=1 -- '
```

The server would read this as the username needing to simply be ; or 1=1 to login as it reads the or1==1 -- bit in the Select query as code, and only the ‘’ bit before the or as data this means in turn the Select statement is always going to True. As to get past the password field , the double hyphen at the end of the the Select query simply comments out the part of the statement that needs to check for a password and therefore one is rendered useless meaning the attacker does not even need to supply one to gain access. From here the attacker could view contents of the data stored in my database and use other queries and

Injections to further render and harm it such as being able to drop entire tables again using simple Select queries.

There is however way's I could implement precautions that can aid in the prevention of an Injection Attack on my database with simple things such as script that can identify illegal inputs from a user this however is just a short-term and fairly easily avoidable solution. Because the previous solution has issues I would rather use the implementation of both prepared statements and stored procedures to separate code from data in doing so all commands are then completely separated from the data from which they run on. Once commands such a the query above is no longer attached to the data the attacker is wanting to reach the Injection becomes useless and there prevents or even makes it pointless for the attacker to even try the Injection. One method I could use in prepared statements is by using place holders where data should be in a command, entering the code to the server, then using parameters when attaching data to the said command execute the command. There are small benefits if I was using this method of protection such as the fact it's a fairly easy method to implement and use and I wouldn't need to change change much of my code as the previously stated method only requires small changes.

References:

NoSQL:

Bankar, A. (2018). *Dynamic Schema in NoSQL*. [online] Blog.e-zest.com. Available at: http://blog.e-zest.com/dynamic-schema-in-nosql/ [Accessed 7 Jan. 2018].

Fowler, M. (2018). *Schemaless Data Structures*. [online] martinfowler.com. Available at: https://martinfowler.com/articles/schemaless/ [Accessed 6 Jan. 2018].

Database Security:

Incapsula.com. (n.d.). *SQL (STRUCTURED QUERY LANGUAGE) INJECTION*. [online] Available at: https://www.incapsula.com/web-application-security/sql-injection.html [Accessed 6 Jan. 2018].