<u>NoSQL</u>

NoSQL simply means 'not only SQL' and is non-relational, it encompasses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications (MongoDB, 2018). The term 'database schema' refers to the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated, and it formulates all the constraints that are to be applied on the data (TutorialsPoint, 2018). Relational databases require that schemas be defined before you can add data, whereas NoSQL databases are built to allow the insertion of data without a predefined schema (MongoDB, 2018) and are therefore dynamic.

There are many advantages to the dynamic schemas permitted by NoSQL. For example, it makes it easy to make significant application changes in real-time, without worrying about service interruptions – which means development is faster, code integration is more reliable, and less database administrator time is needed (MongoDB). If a company such as LCS was to grow and start dealing with large amounts of various data, this would be highly advantageous since it would support a wider range of datatypes without the need for structural modification and allow you to add or change data quickly or easily. As is often the case in the fast-paced business world, saving time often means saving money so a dynamic schema is likely to make the creation and usage of a database far more efficient in a competitive business environment. Setting datatypes and integrity constraints in an SQL database can be a time-consuming process which requires a lot of thought and planning. As well as this, there is a high possibility that this initial set up may not be 'right first time' or may simply become unsuitable as business develops. For this reason, a lot of time and effort may be required to update these constraints in order to update and adapt a database. An example that arose during the planning on creation of the LCS database was setting the datatype for the house number column. The initial plan was to set this column to an 'int' since the majority of houses are assigned a number no more than 3 digits long. However, further thinking led to the realisation that not all houses have a number – some homes are given a name, and others may contain a letter as well as a number. This meant that the house number field had to be set as a 'varchar' which greatly reduces the possibility of error-checking as well as the efficiency of the database. For this example, a dynamic schema would be better since it would allow for any type of data to be entered in this field without the need for careful set-up and modifications.

On the other hand, the relatively new NoSQL designed for the 'hyper-competitive' global business environment may not be the best choice for everyone. "The best way to determine which database is right for your business is to analyse what you need its functions to be. NoSQL is a good choice for those companies experiencing rapid growth with no clear schema definitions. NoSQL offers much more flexibility than a relational database and is a solid option for companies who must analyse large quantities of data or whose data structures they manage are variable" (BMC, 2005-2019). This description does not describe the Lincoln Computer Surgery since they are a small local business unlikely to expand, and their data is likely to remain relatively small and consistent. For this reason, the fixed schema required by SQL may be most suitable for small companies such as LCS since it could be the most efficient option and allow for some degree of error prevention provided by the fixed datatypes. For example, the telephone number field in the LCS database is set to 'char(11)' which means that only a single type of character (numbers) will be accepted, and only 11 of these – no more, no less. This means that there is a higher chance that all telephone numbers stored in the system are valid because they must be in the correct UK telephone number format. Furthermore, unlike in NoSQL, relational databases are ACID compliant which means they are more reliable thanks to guaranteed transactions, consistency, and data validity.

Overall, it is clear that NoSQL and its dynamic schemas can greatly benefit large-scale, rapidly developing businesses due to its flexibility allowing quick and easy input and update of various data. However, there is no 'one size fits all' solution when it comes to choosing a database type – it is entirely dependent on the company's characteristics and needs. Due to the small and relatively simplistic nature of LCS, using fixed schemas is the best option since no business-damaging disadvantage would arise from this structure. The data they need to store is likely to remain largely consistent and straight-forward so the fixed schema could well provide the company with more advantages (such as error-prevention) than disadvantages.

<u>Database Security</u>

SQL Injection (SQLi) refers to an injection attack in which an attacker can execute malicious SQL statements that control a web application's database (acunetix, 2019). This could potentially destroy a database since it could result in unauthorised access to the system, database modification, and the reading or writing of files on the system. A common injection method is using the code 'OR 1 = 1' since this will always return true. An example of how this could happen on the LCS database could be someone gaining unauthorised access through login. This could look like: *SELECT * FROM users WHERE login = '' or 1=1 -- ' and password_hash = MD5('anything')*. This means that the SELECT statement will always return true since 'OR 1 = 1' will always be true. The double hyphen (--) will comment out the rest of the statement which checks the password, resulting in the attacker being able to log in without knowing the login or password. This would mean that a hacker could have all the privileges of an authorised user, allowing them to access sensitive information such as customers' addresses, telephone numbers, and email addresses, as well as being able to maliciously edit the database such as deleting or editing important entries.

There are some ways to prevent SQL attacks from occurring as well as protecting against them if they were to happen. One way of doing this is by using prepared statements which involves preparing SQL commands, setting related data variables, and binding data and execution. "They are simple to write, and easier to understand than dynamic queries. Parameterized queries force the developer to first define all the SQL code, and then pass in each parameter to the query later. This coding style allows the database to distinguish between code and data, regardless of what user input is supplied" (OWASP, 2018). An example of this that could be used for LCS could be preparing a statement for login. The first step is to prepare the statement which would look like; *Prepare stmt From 'SELECT * FROM users WHERE login = ? and password_hash = MD5(?)'*. Then, data must be set such as; *Set @a= $login Set @b= $password*. And finally, the statement can be executed; *EXECUTE stmt USING @a, @b*. This could prevent the type of injection attack mentioned previously since the parameterised query would no longer be vulnerable and would instead search for a user name that actually matched '1 = 1'. Another method of reducing the risk from injection attacks is using stored procedures and whilst they are not always safe from SQL injection, certain standard stored procedure programming constructs can have the same effect as parameterised queries when implemented safely (OWASP). A stored procedure has already been implemented in the LCS database to display all orders with a status other than 'complete'. Although this specific procedure was mainly designed for functionality and efficiency, some stored procedures can be a security measure. For example, users could be granted privileges only through stored procedures and denied access to the tables they are based upon which means if a hacker gained access to a user account, they would not be able to use the database without knowing the procedures.

In conclusion, despite LCS being a relatively small company, all businesses should take as many steps as possible to prevent and combat these attacks in order to guarantee privacy and confidentiality to their customers by keeping their sensitive data safe. As well as this, changes made to the database by a hacker could potentially seize business if important information or structure is edited or deleted which would cost valuable time and therefore money. In addition to code solutions, LCS should also take basic measures such as using strong passwords and regular backups to further prevent and reduce the effects of any attacks.

References

MongoDB | NoSQL Databases Explained (2018) [online] available at: https://www.mongodb.com/nosql-explained [accessed 8 Jan 2019]

TutorialsPoint | DBMS – Data Schemas (2018) [online] available at: https://www.tutorialspoint.com/dbms/dbms_data_schemas.htm [accessed 8 Jan 2019]

BMC blogs | The Business of IT Blog, NoSQL vs SQL: What's The Difference and How To Choose (2005-2019) [online] available at: http://www.bmc.com/blogs/sql-vs-nosql/ [accessed 8 Jan 2019]

Acunetix | SQL Injection (SQLi) (2019) [online] available at: https://www.acunetix.com/websitesecurity/sql-injection/ [accessed 8 Jan 2019]

OWASP.org | SQL Injection Prevention Cheat Sheet (06/02/2018) [online] available at: https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet [accessed 8 Jan 2019]