***Recoverable***

Reading data transaction should commit last

***Cascadless***

Should read only after writing transaction is committed

***Strict***

Should read, write only after writing transaction is committed

***Serial***

All operations of a transaction should be in sequence (consecutively)

***Conflict***

Different transaction, working on same data item, at least one of them is write

***Conflict Equivalent***

Change to other schedule after changing positions of non-conflicting operations

***Conflict Serializable***

Draw usage graph, cycle should not be present

***SQL Injection***

Manipulate SQL for malicious purposes

Methods –

1. String Manipulation: Parameter manipulation, function call
2. Code Injection

Goals –

1. Fingerprinting: learn about system, its behavior
2. DoS: Denial of Service
3. Privilege escalation/Authentication bypass
4. Remote execution

Mitigation –

1. Parameterized statements
2. Minimum use of custom functions

***Secure Password Storage***

1. Never save plain text password
2. Save salted /hashed passwords
3. Use modern hashing that requires high work factor for hashing
4. Use Honeywords for breach detection

Indexes created on the attributes that has higher selectivity reduces the execution time of the query. Selectivity is nothing but uniqueness of the attribute, primary key has all unique values the its selectivity is very high and hence default indexes make use of primary keys.

For any table ‘insertion time’ is directly proportional to number of indexes (on high cardinality column) created on that table, as it must create, maintain and then insert the data in the table. That’s why significant increase can be seen in the average increase time when index attribute has large domain values.

Space used is directly proportional to cardinality of attributes in that table. Thus, higher the domain range of an attribute higher the space taken by table in memory.

Query performance can be improved by creating an index on the attribute that has high cardinality. Such indexing decreases the query execution time significantly however the down side is it also increases the insertion time.

Creating indexes on many attribute had drastic effect on the query performance. The insertion time increased by more than 900% and the space consumption also increased by more than 200%. This is because number of indexes increases the maintenance cost of the table which increases in insertion time and to save the index information, table takes up more space on disk.

Query time decreased by more than 50% for two non-primary key attributes whereas, for three non-primary key attributes that are present in index the execution time decreased by more than 250%. *Maria DB uses ‘index\_merge’ pattern while multiple attribute query execution. That is db chooses the intersection of the attributes (most selective) in the where clause to expedite the execution process*.

When dealing with a database with high insert-to-query ratio, if number of queries executed on database is very high as compared to insert, it is better to include more column from where clause into index. However, if database has more insert queries or the insert-to-query ratio is very minute at that time it is safe to create indexes on only those columns which are very important.

*Multi attribute index reduced the insertion time as compared to the multiple single attribute indexes however, it did not benefit the query execution all the time*.

If all attributes of the where clause are covered as an index then the execution time is reduced significantly on the other hand when only few columns out of the where clause were present in the index, it did not help the execution at all.

In case of high query-to-insert ratio (queries fired are very high as compared to insert) making multiple-attribute index on columns that appear in the where clause most of the time is a good strategy.

Good rule of thumb is to place most selective attributes at the beginning of the index.

Creating redundant indexes has more cons as compared to the minute pros. So, I think if (only) a2 is not appearing specifically in where clause of most of the queries then creating single attribute index on will not help.