# Design considerations

When designing a database there are many considerations which must be considered.

1. Who is going to use it?
2. How is it going to be used?
3. What guarantees can we have about the input?
4. What performance requirements are there?
5. …

# Choosing a database type

Most traditional databases are normalized. This means that data preferably only exist once and are linked via keys. This is referred to as normalized data. The advantage of this format is that it has a low footprint as duplicate data only exist in form of keys. To retrieve the data the different tables must be joined. Unfortunately the joining operations are costly, and if the throughput is very large, normalized data is actually not preferable.

Another important consideration is consistency. Traditional databases use ACID (Atomicity, Consistency, Isolation, Durability), which ensures that when data is written to the database subsequent requests is guaranteed to retrieve the new update. It also ensures that simultaneous updates are protected from each other and that transactions are supported. Unfortunately this form of access is also expensive, and sometimes not needed. An alternative is BASE (Basically Available, Soft state, Eventual consistency). Sometimes it is OK that we simply know that an update will eventually be completed, and that inconsistencies are OK for a limited time. This form of database is also used for very large project, and often combined with a distributed database. It often relies on NoSQL (Not Only SQL) as opposed to SQL.

As we are designing a database for a larger cloth wholesaler which includes payments, it is beneficial to have ACID ensured, however it would be quite possible to implement it on BASE. Also, since the database is to be used by a limited number of simultaneous users (< 10000000) an ACID DB should be sufficient, and normalized data will also be acceptable, and the databases indexing and performance optimization (keeping some tables joined in RAM to improve performance) should be sufficient.

# Who is going to use it

If the database is only used by fully trusted personnel then there is no need to place limitations on the access, however if the database is to be accessed by people we do not trust, then it is important to ensure that the users are not able to access more than he or she is allowed to.

Access to the database is not directly part of the SQL standard, but most database implementation works with Users, permissions and possibly Roles. The normal implementation allows for limiting a user’s permissions to one or more specific tables or views as well as whether the user is allowed to insert, update or select. Unfortunately this is insufficient when data from multiple individuals exist in the same table and it is therefore necessary to add another layer of protection. This could be on the form of a REST service with its own layer of authentication and exposing a limited interface to the user.

If we wish to use the database as is it gets more complicated. There are solutions, but not all of them are possible.

If we assign a GUID (e.g. 128bit) to each row and force a where clause then only a customer who knows the correct GUID would be able to read the row. Unfortunately forcing a where clause is more complicated than one might think.

As it is possible to limit a user to a given view, this might seem like a good idea, but unfortunately a view is static, and though where clauses is allowed, it is not allowed to supply a parameter.

However postgresql supports limiting users to specific functions, and functions support mandatory parameters, it is possible to use the built in permissions system of postgresql to limit a user’s access to a table to only those rows where the user knows the correct id.

There will be a performance penalty by having to go through a function first, but it is believed that it is acceptable.

As a minor note, if the users are allowed to insert arbitrary data it is important to encode the text to prevent them from inserting SQL-statements inside the text.

# How is it going to be used

As mentioned before if the database is used through a predefined interface it is simple to limit the access based on authentication. We are going to attempt to create the database so it may be access through the normal postgresql access and authentication, so the database may simply be exposed “online”.