Не оставляйте закомментированного кода вообще. CVS хранит все изменения, не пропадет никуда. Боитесь забыть — пейте противотревожное.

Не добавляйте в общие модули ссылки, нужные только вам или даже не нужные хотя бы иногда. Создавайте новый модуль и пишите там.

В проектах, где это нужно, будет использоваться он, а остальные проекты будут избавлены от необходимости тащить все, на что ссылается добавленный в uses модуль.

Не пишите конкретику в абстрактных классах. Наследуйтесь. Примерно по тем же причинам.

В обработчиках событий форм пишите **inherited**, даже если форма наследуется от TForm. Когда перенаследуешься, пожалеешь что не писал.

О подписке на события. Выполнение произвольного кода чтение и запись произвольной памяти.

В конце блока if каментить условие это утопия. Условие поменяется, а камент стопудов старый останется. Не надо длинные блоки писать вообще. Пишите процедуры, если тело блока занимает больше пяти строчек. Лучше читать книгу написанную словами, чем ковыряться в "матрице".

Объявлять переменные стека обязательно в начале метода — нет. Никогда не поймешь, какие из них могут невидимо меняться в процедурах стека.

Не комментируйте, что делает код, а зачем он это делает. Если сказать о зачем нечего, значит контекст не нуждается в комментарии.

В проектировании базы очень плохая идея это:

1. Создавать составные ключи из полей, содержащих контекст.

Это очень ненадежная модель, поскольку ни один датасет не поддерживает адекватное изменение ключевых значений. И не может поддерживать. И не должен.

Кроме того, это в разы увеличивает затраты на разработку абстрактных обработок в клиентах. В случае, когда есть договоренности по ключевым полям, разработка клиента становится проще. Предельно надежный случай - все таблицы содержат одно ключевое поле типа GUID с одинаковым именем или его частью.

При проектировании структуры данных всегда нужно помнить, проектировать и оптимизировать базу в сотни раз проще и быстрее, чем писать обработки. Можно быстренько в базе наворотить черте чего и потом долго удивляться, почему клиенты не пишутся и выламываются годами.

2. Использовать любые неявные признаки.

Допустим, нам нужен порядок. Используем ключевое поле для его определения. Очень плохо по причинам из п.1.

Или у нас есть таблица-мультирасширение для многих других таблиц. Она должна или ссылаться на ключ GUID этих других таблиц, или содержать явный признак принадлежности. Например, в виде имени таблицы в отдельном поле.

Вообще, не только в базах, а и везде, если какой-либо один признак не совпадает прямо до одного сравнения с другим признаком, то не стоит использовать первый для второго. Необходимо завести два признака. Вплоть до случая, когда это просто разная логическая суть или "слишком другое" название признака.

Например, у нас есть объект Nose и его признак HasSnot. Мы пишем внешнюю относительно объекта процедуру WipeOut, которая вызывается по критерию IsWet. Даже если по статистике на сегодняшний день IsWet = HasSnot всегда, следует завести в объекте Nose отдельный признак IsWet с явной сутью, который будет возвращать HasSnot. Особенно, если IsWet = (HasSnot and (LastWipedOut > WipeOutInterval)). Проверку дополнительных признаков нельзя делать в месте вызова, даже если эти признаки внешние.