# Этапы разработки проекта: заключительные стадии проектирования, схема базы данных

## Заключительные стадии проектирования

### Проектирование процесса тестирования

Проектирование процесса тестирования, как правило, следует за процессом функционального проектирования и проектирования схемы базы данных. На этом этапе можно использовать сложные схемы тестирования, а можно ограничиться и простыми. Здесь мы приведем некоторые принципы, которых  нужно придерживаться  при проектировании любой информационной системы.

Когда генерация модуля завершена, выполняют автономный тест, который преследует две основные цели:

* обнаружение отказов модуля (жестких сбоев);
* соответствие модуля спецификации (наличие всех необходимых функций, отсутствие лишних функций).

После того как автономный тест прошел успешно, группа сгенерированных модулей проходит тесты связей, которые должны отследить взаимное влияние модулей.

Далее группа модулей тестируется на надежность работы, то есть проходят, во-первых, тесты имитации отказов системы, а во-вторых, тесты наработки на отказ. Первая группа тестов показывает, насколько хорошо система восстанавливается после сбоев программного обеспечения, отказов аппаратного обеспечения. Вторая группа тестов определяет степень устойчивости системы при штатной работе и позволяет оценить время безотказной работы системы. В комплект тестов устойчивости должны входить тесты, имитирующие пиковую нагрузку на систему.

Затем весь комплект модулей проходит системный тест - тест внутренней приемки продукта, показывающий уровень его качества. Сюда входят тесты функциональности и тесты надежности системы.

Последний тест информационной системы - приемо-сдаточные испытания. Такой тест предусматривает показ информационной системы заказчику и должен содержать группу тестов, моделирующих реальные бизнес-процессы, чтобы показать соответствие реализации требованиям заказчика.

### Требования к безопасности, доступу, обслуживанию системы

Каждая информационная система содержит определенные требования к защите от несанкционированного доступа, к регистрации событий системы, аудиту, резервному копированию, восстановлению информации, которые в начале проектирования должны быть формализованы аналитиками. Проектировщики строят стратегию безопасности системы. В частности, ими должны быть определены категории пользователей системы, которые имеют доступ к тем или иным данным посредством соответствующих компонентов. Кроме того, определяются объекты и субъекты защиты. Следует отметить, что стратегия безопасности не ограничивается только ПО - это должен быть целый комплекс мер и правил ведения бизнеса. Нужно четко определить, какой уровень защиты данных необходим для каждого из компонентов информационной системы, и выделить критичные данные, доступ к которым строго ограничен. Пользователи информационной системы регистрируются, поэтому проектируются модули, отвечающие за идентификацию и аутентификацию пользователя. В большинстве СУБД реализована дискреционная защита данных, то есть регламентирован доступ к объектам данных (например, к таблицам, представлениям). Если же требуется ограничение доступа собственно к данным (к отдельным записям в таблице, к отдельным полям записи в таблице и т.п.), то следует реализовать мандатную защиту. Проектировщики должны иметь четкое представление о том, какой уровень защиты той или иной единицы информации является необходимым, а какой достаточным.

Вопросы восстановления, хранения резервных копий базы данных, архивов базы данных относятся к мероприятиям поддержки бесперебойного функционирования информационной системы. Необходимо внимательно изучить возможности, предоставляемые СУБД, а затем проанализировать, как следует использовать возможности СУБД для обеспечения требуемого уровня бесперебойной работы системы.

### Составление спецификаций

Результаты проектирования отражаются в документе - функциональной спецификации. Этот документ пишется для заказчика, чтобы получить его санкцию на завершение проектирования и начало разработки, и обычно не содержит большого количества технических деталей. Второй документ - техническая спецификация, являющаяся основным документом для разработчиков моделей и групп тестирования, и здесь описаны детали проекта. Если использовались CASE-средства, то техническая спецификация обязательно содержит ряд отчетов из репозитария.

### Полнота проектирования

Перед началом разработки модулей нужно еще раз проверить полноту проектирования. Один из полезных инструментов - матрица использования таблиц схемы базы данных по модулям.

### Переход к реализации

Итак, начата реализация модулей. Означает ли это, что работа проектировщиков на этом завершена полностью? На практике это далеко не так. Довольно часто разработчик сталкивается с медленно работающими или не реализуемыми в данной схеме запросами. Подобные ситуации инициируют изменение модели данных, а значит, и информационной модели. Однако изменение информационной модели производится не только по этой причине. Хорошему проектировщику необходим практический опыт работы с аппаратным и программным обеспечением - вот одна из причин участия проектировщиков в составе групп разработчиков. Нередко ведущие сотрудники групп разработчиков одновременно являются проектировщиками.

Как можно использовать проектировщиков на этапе разработки? Приведем некоторые примеры:

* Проектировщик, написавший спецификацию модуля, проводит семинар с разработчиками и демонстрирует необходимые прототипы (семинар предполагает диалог двух сторон).
* Когда модуль передан разработчику, проектировщик может участвовать в его пересмотре, а также выполнять контрольные функции по реализации проектных решений.
* Для крупных проектов характерно поэтапное выполнение работ, так что вполне вероятно, что после завершения реализации группы модулей и сдачи очередного этапа процесс проектирования будет продолжен для новой группы модулей.
* Проектировщики должны обеспечить быстрое реагирование на возможные изменения требований заказчика, поскольку своевременная обработка такой информации является их обязанностью. Кроме того, необходимо и участие системных аналитиков, так как именно они общаются с заказчиком проекта.

## Схема базы данных

Схема базы данных содержит описание всех объектов базы данных: пользователей, их привилегий, таблиц, представлений, индексов, кластеров, ограничений, пакетов, хранимых процедур, триггеров и т.п. При этом создаются не только определения этих объектов, но и сами объекты, с которыми потом работают разработчики.

### ER-модель и ее отображение на схему данных

Результат этапа анализа - построение информационной модели. Казалось бы, дело это простое: сущности становятся таблицами, а атрибуты сущностей - столбцами таблиц; ключи становятся первичными ключами, для возможных ключей определяется ограничение unique, внешние ключи становятся декларациями ссылочной целостности. Аналитики, как правило, не вникают в особенности реализации той или иной СУБД, поэтому при проектировании схемы базы данных проектировщик сталкивается с конструкциями в информационной модели, которые не реализуемы или трудно реализуемы в выбранной СУБД.

Приведем несколько примеров ограничений реализации СУБД:

* В информационной модели описаны три сущности - A, B, C. Сущности B и C содержат внешние ключи, ссылающиеся на сущность A. В СУБД поддерживается возможность определения внешнего ключа только для первичного ключа, а для возможного ключа определить декларативную ссылочную целостность нельзя. В этом случае отображение ER-модели на физическую модель данных невозможно без изменения информационной модели.
* В информационной модели описан внешний ключ с каскадным удалением и модификацией. В СУБД поддерживаются внешние ключи только для варианта действия no action (то есть каскадные изменения в явном виде не поддерживаются). Реализация ссылочной целостности посредством триггеров ограничена уровнем каскадирования триггеров (например, 32 вызовами триггера). В этом случае потребуется также изменение информационной модели.
* В информационной модели определен атрибут, представляющий собой строку длиной в 500 символов. По этому атрибуту часто осуществляется поиск в информационной системе; объем данных велик. В СУБД можно индексировать строки символов не длиннее чем 128 или 256 символов. Если осуществлять поиск без индекса, то время ответа информационной системы существенно превышает допустимое, вследствие чего придется изменить описание сущности.
* В информационной модели описана сущность A, которая содержит по крайней мере два атрибута BLOB (например, требуется отдельно хранить и звук и изображение). В СУБД невозможно создать таблицу с двумя атрибутами BLOB и в этом случае нужно изменить описание сущности. Из отношения A исключаются два атрибута BLOB, и добавляется один атрибут AK типа integer. Добавляются две дополнительные сущности - и . Каждая из них будет содержать один атрибут BLOB и один ключевой атрибут K типа integer, который станет внешним ключом у каждого из новых отношений и будет ссылаться на атрибут AK в отношении A (тип внешнего ключа - on delete cascade, on update cascade).
* Жизненный цикл сущности определен в информационной модели соответствующей диаграммой. В описании сущности отсутствует атрибут, который отражает изменение состояния сущности. В этой ситуации проектировщики добавляют атрибут status, для которого определяется ограничение допустимых значений (из списка допустимых состояний), а изменение состояния сущности описывается триггером, проверяющим допустимость сочетания нового и старого значения атрибута.
* Две диаграммы потока данных описывают различные бизнес-процессы, работающие над одними и теми же данными. Допустим, первая диаграмма описывает выписку товара со склада, а вторая -  сложный отчет, отражающий состояние склада. Один процесс интенсивно модифицирует данные, второй работает в режиме чтения данных, но требует согласованности данных в течение длительного времени. Каждый из процессов описывается транзакцией над данными. В СУБД уровни изолированности транзакций реализованы так, что читающие транзакции конфликтуют с модифицирующими транзакциями. Это приводит к остановке выписки товаров со склада на время выполнения отчета, что неприемлемо для заказчика. Здесь может потребоваться очень серьезное изменение информационной модели.

Подобных примеров, когда не только ER-модель, но и другие продукты анализа не могут быть перенесены автоматически на модель данных, можно привести множество. Каждый такой случай инициирует изменение информационной модели. Решение проблемы определяется возможностями СУБД, выбранной для реализации проекта. Если проблем, не разрешаемых в рамках данной СУБД, накапливается очень много, то проектировщики могут поставить вопрос о смене СУБД. Такой вопрос поднимается именно на стадии проектирования, поскольку если уже разработчики столкнутся с подобными проблемами, то цена смены СУБД будет выше. Ясно, что одинаковых СУБД не бывает: то, что хорошо работает в одной, может плохо работать или вообще не работать в другой, несмотря на уверения производителей СУБД в поддержке стандартов SQL. Что касается хранимых процедур и триггеров, то здесь вообще трудно говорить о поддержке SQL92/PSM.

Вопросы производительности информационной системы также влияют на отображение ER-модели на модель данных. За счет мощного сервера баз данных можно добиться большей скорости реакции системы, но мощность аппаратного комплекса ограниченна. Производительность системы в целом зависит в том числе и от нормализации. Часто до 80% запросов к базе данных являются выборками данных, а соединение по тому или иному атрибуту относится к затратным операциям, в первую очередь соединение по нечисловым атрибутам. Увеличить производительность системы можно посредством введения избыточной информации - денормализации. Следует отметить, что решение об этом не принимается на основе одной ER-модели - требуется внимательно проанализировать потоки данных. Критичные процессы являются хорошими кандидатами для денормализации: по времени выполнения, по частоте выполнения, по большому объему обрабатываемых данных, по частоте изменения обрабатываемой информации, по явному приоритету. Часто к денормализации прибегают в целях ускорения выполнения отчетов. Для проверки эффективности той или иной денормализации привлекаются тестеры.

### Типы данных

Как правило, СУБД поддерживают небольшой набор базовых типов данных: числовые типы (целые, вещественные с плавающей и фиксированной точкой), строки (символов и байт), дата и время (или комбинированный тип datetime), BLOB (и его разновидности, например BLOB-поля для хранения только текста). В информационной модели каждому атрибуту соответствует домен. Поскольку не все реализации СУБД поддерживают домены, то в этом случае при определении модели данных ограничения домена описывают как ограничения столбца таблицы (если такое возможно); в частности используют check constraints, триггеры. Следует отметить, что при определении типов столбцов таблиц нужно учитывать, какие типы данных поддерживаются в словаре данных СУБД. Например, в Oracle ключевые слова integer, smallint, real поддерживаются транслятором SQL, но в словаре данных им соответствуют number(38), number(38), float(63), так как Oracle хранит данные в двоично-десятичном формате с плавающей точкой, а не в двоичном формате с плавающей точкой, и  38-восьмизначное число никак нельзя назвать словом smallint.

СУБД поддерживают два вида строковых типов: с фиксированной длиной (например, char), когда хранится ровно столько символов, сколько указано в описании атрибута, и с переменной длиной (например, varchar), когда хранится реальная длина значения атрибута, а концевые пробелы строки усекаются. Семантика сравнения строк в СУБД также различная, и если ваше мнение о сравнении строк расходится с тем, как это реализовано в СУБД, то придется смириться с этим как с особенностью СУБД. Например (описано поведение Oracle 7.x), если сравниваются значения A равное ‘ab’ и B равное ‘ab’ двух атрибутов типа varchar разной длины, то sql сообщит, что . Чтобы избежать подобных «фокусов», нужно, в частности, следить за тем, чтобы приложение не вставляло незначащие концевые пробелы в значения атрибутов этих типов.

### Индексы, кластеры

В правильно спроектированной базе данных каждая таблица содержит первичный ключ, что означает наличие индекса. В большинстве СУБД используются индексы . Отметим, что если используется составной индекс, то поиск по всем атрибутам, входящим в индекс, начиная со второго, будет медленным. Допустим, определен индекс index1(id1, id2), в этом случае поиск значений, удовлетворяющих условию id2=1, будет медленным (не исключено, что оптимизатор вообще не будет использовать этот индекс для обработки данного условия и будет принято решение о полном сканировании данных), а поиск значений, удовлетворяющих условию id1=1 and id2=1, будет быстрым. Данные особенности следует учитывать при определении индексов в схеме базы данных, а именно:

* индексировать нужно атрибуты, по которым наиболее часто осуществляется поиск или соединение. Наличие индекса замедляет операции модификации, но ускоряет поиск;
* наличие индекса обязательно, если для атрибута или набора атрибутов указано ограничение unique. Такие индексы СУБД создает автоматически, если в описании таблицы указаны ограничения unique;
* индекс может быть использован для выборки данных в заданном порядке. В этом случае не вызывается процесс сортировки ответа, а используется уже готовый индекс;
* атрибуты, входящие во внешний ключ, также следует индексировать, если СУБД не делает эту операцию автоматически при декларации внешнего ключа;
* в некоторых СУБД поддерживаются bitmap-индексы, которые очень эффективны при поиске на равенство, но для поиска на  этот тип индексов не годится;
* в некоторых СУБД поддерживаются хеш-индексы, например для кластеров. Такие индексы эффективно используются при поиске на равенство.

Кластеризация - это попытка разместить рядом в одном физическом блоке данных те строки, доступ к которым осуществляется при помощи одинаковых значений ключа. Индексные кластеры, например, удобно использовать для хранения родительской и дочерних строк таблиц, связанных ссылочной целостностью. Кластеры удобно определять для тех наборов атрибутов, соединение по которым проводится наиболее часто, поскольку это увеличивает скорость поиска. Следует отметить, что в реализациях СУБД существуют жесткие ограничения на количество кластеров для таблицы, как правило, это один кластер. Особенности реализации кластеров в СУБД необходимо учитывать при проектировании критичных по времени выполнения модулей. Нужно обратить внимание, насколько сильно влияет наличие кластера на производительность DML-операций. Чаще всего это оказывает отрицательное влияние, которое в некоторых реализациях распространяется на DML-операции над любой таблицей базы данных, а не только над той, для которой определен кластер. Эти особенности СУБД также следует учитывать при проектировании.

Для того чтобы выбрать тот или иной тип индекса, требуется внимательно изучить руководство администратора СУБД. Оптимизатор SQL использует различные типы доступа к данным при обработке запросов, и индексирование существенно влияет на выбор оптимизатора.

Приведем некоторые способы доступа к данным на примере выборки select id, name from xtable where id=10:

* Таблица не индексирована. В этом случае применяется полное сканирование, которое не является эффективным, если объем данных большой, в таблице много данных, не удовлетворяющих условию, размер кортежа существенно превосходит размер атрибута id.
* Атрибут id индексирован; тип индекса - . Тогда применяется индексное сканирование; полное сканирование может быть выбрано только в случае, если объем данных, удовлетворяющих данному условию, является большим (эта информация анализируется статистическим оптимизатором) и сравним с количеством записей в таблице. Индексное сканирование применяется лишь тогда, когда размер индексированных атрибутов меньше размера кортежа и все необходимые для обработки запроса данные могут быть получены из индекса; в остальных случаях может быть применено полное сканирование.
* Атрибут id входит в составной индекс, и является первым (лидирующим) атрибутом индекса, при этом тип индекса - . Аналогично предыдущему примеру,  применяется индексное сканирование.
* Атрибут id входит в составной индекс, он не первый атрибут индекса; тип индекса - . Индексное сканирование применяется лишь тогда, когда размер индексированных атрибутов меньше размера кортежа и все необходимые для обработки запроса данные могут быть получены из индекса; в остальных случаях применяется полное сканирование.
* Атрибут id индексирован; тип индекса - хеш. Здесь применяется индексное сканирование для поиска на =; если бы условие было id <=10, то применение хеш-индекса для такого поиска не эффективно.
* Атрибут id индексирован; тип индекса - bitmap. Здесь применяется индексное сканирование для поиска на =; если бы условие было id <=10, то применение bitmap-индекса для такого поиска не эффективно.
* Атрибут id является ключом хеш-кластера. В этом случае применяется алгоритм хеширования при поиске блока данных для чтения. При хорошем алгоритме и правильном размере кластера поиск может быть осуществлен за одно чтение; при ошибках в выборе алгоритма и блока кластера это может составить до нескольких тысяч операций чтения блоков.
* Атрибут id является ключом индексного кластера. Здесь применяется индексное сканирование, почти аналогично случаю с индексом .
* Таблица кластеризована, id не является ни кластерным ключом, ни лидирующим в составном индексе . В этой ситуации для кластера применяется полное сканирование.

Мы привели только некоторые правила выполнения операции поиска в зависимости от наличия и типа индекса. В реализации используемой вами СУБД могут быть приняты иные принципы. Подробности использования типов сканирования при поиске данных даются в руководстве по настройке СУБД и в руководстве администратора СУБД.

А почему бы не проиндексировать все, если индексный поиск быстрее полного сканирования? Очевидно, что индекс занимает место на диске, вопрос в том - сколько. Например, индексируется атрибут integer - это 4 байта. Но в  кроме собственно значения ключа в индексе хранятся и внутренний идентификатор кортежа, и некоторая служебная информация, так что все вместе может составлять 4-8 байт. Чтобы точно посчитать эту величину для используемой вами СУБД, следует обратиться к руководству администратора: посмотрите размер идентификаторов ROWID (Oracle), RID (DB2) и т.д., а также размер страницы индекса (как правило, это 4 Кбайт).

При выборе стратегии индексации следует придерживаться двух простых принципов:

* чем больше индексов, тем выше затраты на выполнение DML-операций. По грубым оценкам затрат: если принять за 1 работу по вставке строки в таблицу, то работа по вставке той же записи в один индекс равна 2 или 3 (для разных СУБД);
* в любое значение может быть найдено за такое количество операций чтения, сколько уровней у дерева (дерево трех уровней для значений integer, например, содержит порядка 533 731 324 ключей, если страница дерева 4 Кбайт). Такие индексы отлично используются при поиске на =, <, >, <=, >=, between, и достаточно хорошо модифицируются. В bitmap-индексах содержатся готовые битовые векторы, отражающие вхождение или невхождение значения в ответ при поиске на равенство, но такие индексы плохо модифицируются и больше подходят для хранилищ данных, например для индексирования вхождения слов в текстовый документ. Хеш-индексы позволяют осуществлять поиск на равенство, хеш-функция используется для поиска блока кластеризованных данных, содержащего нужные значения. Если алгоритм хеш-функции хорош и размер кластера указан верно, то поиск может быть осуществлен за одно чтение. Эти индексы, как правило, используют для создания кластеров.

Обратите внимание, хранит ли СУБД в индексах NULL. Если NULL в индексе не хранится, то вероятность использования полного сканирования для атрибутов без декларации not NULL резко повышается. Если NULL хранится в индексе (обычно его считают самым большим или самым маленьким при построении  и специальным значением при построении bitmap и хеш-индексов), то выясните, для каких операций поиска индексы будут использованы оптимизатором SQL. Эта информация, как правило, содержится в руководстве администратора. Можно проверить и экспериментально, создав тестовую таблицу с объемом данных примерно 20 тыс. записей (чтобы оптимизатор не выбирал полное сканирование по причине малого объема) и выполнив исследуемый запрос, а затем произвести explain плана запроса (если подобный сервис предоставляется СУБД).

### Временные данные

Временными данными, или временными рядами, называют данные, содержащие дату и время. Неправильная обработка таких данных в некоторых СУБД может служить одной из основных причин низкой функциональности и производительности информационной системы. Временные ряды не очень хорошо вписываются в двухмерную реляционную модель. SQL поддерживает соединения, не основанные на равенстве, но большинство разработчиков СУБД ограничиваются эквисоединением. Для временных данных часто приходится соединять таблицы на основе перекрытия одного диапазона дат другим. В SQL не существуют операции, которая позволяла бы задать такое соединение непосредственно.

Ряд атрибутов, например курс валюты или цена товара, изменяются во времени. Такие атрибуты действительны по дате, то есть актуальны только в течение определенного интервала времени, например дня для курса валюты.

Если СУБД позволяет обрабатывать многомерные данные, то обработка временных рядов может использовать эти механизмы и тогда время будет являться одним из измерений. Подобные многомерные процессоры применяются для обработки геофизических и географических данных. В таких системах используются индексы ,  и их клоны.

Приведем пример обработки цены товара (код товара, начальная дата, конечная дата, цена):

create table prices

(id integer, date\_from date not null,

date\_to date, price decimal not null,

constraint p\_range check date\_from < date\_to);

Отметим, что здесь в отношении не задан первичный ключ, а сама задача определения ключа в таких отношениях отличается сложностью. Известно, что момент изменения цены заранее не известен, этим и объясняется отсутствие ограничения not null для атрибута date\_to:

select price from prices

where id = :PRODUCT\_CODE

and date\_from < :WHEN\_DATE

and date\_to >=

nvl(:WHEN\_DATE, to\_date(’01/12/4721’, ’DD/MM/YYYY’);

Здесь :PRODUCT\_CODE и :WHEN\_DATE обозначают переменные включающего языка, дата ’01/12/4721’ является самой большой из поддерживаемых СУБД (эта дата  может быть и другой). Подобные операции лучше оформлять в виде хранимых процедур, функций или претранслированных запросов. В хранилищах данных часто обрабатываются архивные данные, для которых обработка временных рядов также актуальна.

Возникают также вопросы по поводу точности дат, например: они актуальны для курсов валют, биржевых сделок. Следует определить, достаточна ли точность даты до секунды или нет. Существует проблема и с тем, как зарегистрировать сделку, если цена акции постоянно меняется. Некоторые проектировщики в этом случае обрабатывают временной ряд, но есть и более простое решение -  отследить цену акции на момент прохождения биржевой транзакции и сохранить эту цену как часть информации, которую модифицировала транзакция. Поэтому сначала следует рассмотреть простые варианты решения задач, и если ни один из них не подходит, то использовать временные ряды.

А также как нужно показывать не известную на текущий момент дату, например момент, когда цена товара перестанет быть актуальной? Сделайте это значение равным NULL или самым большим значением (определите его как default-значение для атрибута). Следует отметить, что значения default в большинстве реализаций СУБД работают только при вставке новых записей. Можно, конечно, создать триггер, который срабатывает после выполнения операции insert или update и преобразует null в наибольшее из допустимых значений даты. Но в этом случае, когда пользователь будет работать с подобной информацией, то может забыть, откуда взялась та или иная дата. Если вы спроектировали схему базы данных и запросы таким образом, то все приложения, работающие с выборками данных, должны отвечать следующим требованиям:

* принудительно конвертировать в null такие даты и не показывать их пользователю;
* использовать соответствующие представления, содержащие decode (или иное средство конвертации «больших» дат);
* вызывать хранимую процедуру, которая выполнит все нужные преобразования.

Выше только что были перечислены возможные проектные решения. Отметим, что для эффективного поиска следует создать составной индекс с атрибутами (date\_from, date\_to), но не все СУБД будут использовать такой составной индекс, если для одного из атрибутов допустимы значения null. Поэтому довольно простая для аналитиков задача представления временных рядов может повлечь за собой множество неприятных моментов при проектировании.

Теперь рассмотрим проблему поиска первичного ключа для подобных отношений. Оказывается, что единственный разумный вариант предотвращения дубликатов таков: каждому единичному интервалу соответствует отдельная строка. Для биржевых операций этот интервал может быть равен, например, одной секунде, а в некоторых системах он еще меньше. У таких таблиц только одно преимущество -  результат из них можно выбирать при помощи эквисоединения, однако объемы обрабатываемых данных велики.

Использование временных рядов, как правило, является одной из наиболее актуальных тем в разговоре с аналитиками. Какое решение будет лучшим - зависит от используемой СУБД и от ее особенностей, а именно: оптимизатора запросов, особенностей использования индексов, мощности SQL, хранимых процедур и триггеров.

### Хранение объектов данных

Это одна из самых сложных задач проектирования схемы базы данных, для решения которой привлекаются администраторы баз данных. Универсальных решений, которые подошли бы для любой СУБД, не существует, так каждый производитель СУБД создает свой способ хранения и доступа к данным, считает его лучшим и очень им гордится. На проектирование схемы базы данных влияют следующие параметры, общие для большинства СУБД:

* размер табличных пространств для хранения таблиц;
* размер табличных пространств для хранения индексов;
* размер табличных пространств для хранения BLOB;
* кластеры и их параметры;
* размер словаря данных, включая код всех хранимых процедур, функций, триггеров, пакетов, статического SQL (реализован только в DB2);
* управляющие файлы;
* файлы журнала;
* интенсивность потока запросов, модифицирующих данные и индексы;
* фалы временных табличных пространств (для хранения временных таблиц, которые строятся, например, при выполнении group by, а также других временных объектов);
* интенсивность потока запросов, инициирующих создание временных таблиц;
* потоки транзакций read-write, read-only, объем модифицируемых и считываемых ими данных, характеристики параллельной работы транзакций (какие и сколько их);
* количество приложений, работающих параллельно с базой данных;
* количество соединений с базой данных для каждого приложения;
* файлы параметров старта ядра СУБД;
* загрузочные модули ядра СУБД и утилиты СУБД;
* входные и выходные данные, генерируемые пользовательскими программами;
* скрипты управления СУБД.

Постарайтесь описать эти параметры хотя бы в общем виде, обсудите все вопросы с администратором баз данных и внимательно выслушайте его рекомендации. Если же его мнение может расходится с вашим представлением о том, как должна работать СУБД, попытайтесь понять его аргументы и запомните, с какими особенностями СУБД это связано. Все данные следует отразить в журнале проектирования. Следует иметь в виду, что многие нюансы размещения объектов данных и конфигурации сервера баз данных не могут быть учтены на этапе проектирования, так как требует полномасштабного тестирования. Конечно, избежать некоторых ошибок проектирования можно, но приготовьтесь к тому, что схема базы данных будет меняться и на этапе реализации, причем  неоднократно.

### Защита данных

Стратегия защиты определяется на этапе анализа, а на этапе проектирования предстоит реализовать эту стратегию, спроектировав соответствующие структуры в схеме базы данных и модули. Большинство СУБД имеют развитые средства дискреционной защиты, а ряд СУБД имеют встроенные подсистемы аудита, что освобождает от необходимости создания собственных средств защиты.

Обычно СУБД предоставляют набор пакетированных привилегий для управления данными, например: connect, которая разрешает соединение с базой данных; resource, которая дополнительно разрешает создание собственных объектов базы данных, dba, которая позволяет выполнять функции администратора конкретной базы данных, и др. Дискреционная защита предполагает разграничение доступа к объектам данных (таблиц, представлений, и т.п.), а не собственно к данным, которые хранятся в этих объектах. Дискреционная защита также обеспечивает создание пользовательских пакетированных привилегий - ролей или групп привилегий. В этом случае набор привилегий на те или иные объекты данных назначается группе или роли, а затем эта группа или роль назначается пользователю; таким образом пользователь получает привилегии на выполнение тех или иных операций над объектами данных косвенно - через группу или роль.

В некоторых реализациях допускается выполнение определенного набора операций от имени другого пользователя и с его привилегиями - в частности вызовы пакетов и хранимых процедур. Пользователь, вызывающий такой объект, не может ни изменить пакет или хранимую процедуру, ни получить доступ к ее коду, но операции над данными, которые выполняются в хранимой процедуре или пакете, будут выполнены от имени владельца хранимой процедуры или пакета. То же самое относится и к триггерам.

Некоторые проектировщики очень любят использовать описанный выше метод защиты данных - это похоже на инкапсуляцию. Иногда проектировщики строят на каскадирующих вызовах хранимых процедур весьма сложные протоколы доступа к данным, имитируя мандатную защиту. Такой подход имеет свои преимущества и недостатки. С одной стороны, любой доступ к данным скрыт хранимой процедурой или пакетом, но с другой - в этом случае словарь данных сильно перегружен. Однако далеко не все реализации СУБД хорошо работают с курсорами в хранимых процедурах и пакетах, поскольку это вызывает  чрезмерную загрузку процессора. Кроме того, в большинстве реализаций СУБД предложения SQL, выполняемые из хранимой процедуры или пакета, имеют более высокий приоритет, чем операции SQL, выполняемые из приложения пользователя. В связи с этим большое количество вызовов хранимых процедур может существенно замедлить выполнение запросов непосредственно из приложений пользователя. В любом случае, чтобы сделать защиту данных, «закрывая» любой запрос хранимой процедурой, требуется, чтобы СУБД имела достаточно развитый язык и позволяла, например, выполнять синтаксический разбор и чтобы внутри самой хранимой процедуры можно было строить как статические, так и динамические предложения SQL.

К сфере защиты данных относятся также сохранность данных и восстановление их после сбоя системы. Для обеспечения бесперебойной работы часто применяют архивирование (в том числе инкрементное) базы данных и журнала транзакций, а в случае отказа системы при следующем старте операции над данными восстанавливают по журналу транзакций (например, производят их откат до определенного момента времени). Применяют также методы горячего резервирования, когда работают два сервера: основной, обрабатывающий запросы пользователей, и резервный, который продолжает работу основного сервера в случае его отказа. Состояние хранилищ данных на основном и резервном серверах согласовано и поддерживается СУБД автоматически, что позволяет проектировщикам не разрабатывать собственные механизмы репликации данных.

Работа серверов в режиме горячего резервирования не избавляет от необходимости хранения резервных копий данных, это может быть и не очевидно для аналитиков и не предусмотрено ими. Некоторые бизнес-процессы по своей природе требуют от информационной системы работы в режиме 24x7, и любой простой стоит очень дорого. В этих случаях работают две или три параллельные системы, и при отказе одного из серверов резервные серверы немедленно принимают управление на себя. Эффективным, но дорогостоящим способом реализации таких задач являются предоставляемые СУБД технологии симметричной репликации. Еще один вариант - архивирование журналов транзакций на резервном узле на специальное устройство и немедленный докат по этому журналу резервного узла в случае отказа основного. Разные СУБД предлагают разные механизмы реализации подобной бесперебойной работы, и для принятия верного проектного решения необходимы консультации проектировщиков с администраторами баз данных.

В простых ситуациях, когда информационная система используется в основном для операций чтения данных, а сами данные меняются редко, резервное копирование может вообще не требоваться, если данные одной такой системы могут быть легко восстановлены из данных других работающих систем. Достаточно будет обеспечить наличие образа базы данных (архив всех файлов базы данных, а также управляющих файлов - это должен быть снимок базы данных на определенный момент времени; проще всего такой снимок получить, остановив СУБД и сделав резервную копию всех указанных файлов).

В большинстве информационных систем необходимо обеспечение безотказной работы системы в ситуации, представляющей собой нечто среднее между описанными выше двумя крайними случаями. Проектировщики должны пересмотреть результаты анализа, исходя из ответов на следующие вопросы:

* каков график необходимой доступности системы для запросов пользователя (то есть когда система обязательно должна работать);
* допустимы ли вообще и когда допустимы периоды профилактического простоя системы;
* допустимы ли и когда допустимы периоды ограничения доступа к системе;
* какие данные после отказа системы нельзя получить из других источников (часто это ввод новых документов, например накладных, операции со счетами, заказы на телефонные переговоры, информация с автоматизированных датчиков и т.п.);
* если данные можно получить, то каков объем повторно вводимой информации;
* каково допустимое время восстановления системы после сбоя;
* имеется ли график пакетных суточных заданий;
* какие еще приложения, кроме информационной системы, работают на данном оборудовании;
* имеются ли резервные аппаратные средства на случай отказа основных;
* имеется ли запас мощности оборудования, на котором функционирует информационная система;
* какова скорость передачи данных при резервном копировании;
* имеются ли специальные отказоустойчивые носители для хранения резервных копий;
* имеется ли правило циклического использования резервных носителей;
* имеется ли специальное защищенное место для хранения резервных носителей.

Ответы на эти вопросы позволят более реально оценить ситуацию и уточнить требования заказчика, формализованные аналитики. Бывает, что заказ работоспособности системы в режиме 24х7 вовсе не является обоснованным и система простаивает, например, 50% времени. Если же требование 24х7 действительно отражает особенности данного бизнеса, то эти вопросы помогут построить соответствующую стратегию защиты данных от сбоев. Качество построенной при проектировании стратегии защиты должно быть проверено тестерами, причем их работа по генерации и проведению тестов, имитирующих отказы оборудования, должна проводиться как на этапе проектирования, так и в течение всего этапа разработки - в целях раннего обнаружения дефектов стратегии защиты данных от сбоев.

### Обмен данными с внешними системами

Большинство аналитиков считают задачи обмена с внешними системами чисто физическими, решенными априори и, как следствие, не уделяют этим вопросам внимания при анализе. Поэтому зачастую проектировщики вынуждены нести на себе всю нагрузку по созданию таких систем обмена.

Импорт и экспорт данных во внешние системы могут обеспечиваться как утилитами, поставляемыми в составе СУБД, так и специальными средствами обмена данными проектируемой информационной системы. Прежде чем писать код собственно системы обмена данными, внимательно изучите сервис, который предоставляет СУБД, и используйте его полностью. Если обмен данными идет между двумя однородными базами данных (одного производителя и одной версии), то средства импорта и экспорта данных во внутреннем формате данной СУБД могут быть использованы без ограничений. Если же обмен данными осуществляется между СУБД разных производителей, то для импорта и экспорта собственно данных можно использовать текстовые файлы. Следует при этом обратить особое внимание на особенности представления null, пустых строк символов и BLOB, так как здесь потенциально присутствует расхождение интерпретации содержимого файлов загрузчиками данных.

Импорт и экспорт схем может оказаться более сложным делом. Производители СУБД часто заявляют о поддержке стандартов ANSI, но на практике большинство баз данных не удовлетворяют им полностью. Синтаксисы операций создания таблиц существенно различаются, а синтаксисы определения хранимых процедур и триггеров большинства СУБД вообще несовместимы. Информация о схеме базы данных может быть получена из системного каталога посредством анализа данных в системных представлениях, но и здесь нет полной поддержки стандарта. Когда информационной системе требуется обмен данными между СУБД разных производителей, то, как правило, проектировщики принимают решение о создании специализированного средства. Даже если оно не будет полнофункциональным и не может быть применено для решения задачи конвертации данных между любыми СУБД, то решить локальную задачу оно способно.

Интерфейсы обмена с внешними системами можно разбить на следующие категории:

* одноразовый импорт данных, унаследованных, как правило, из старой системы;
* периодический обмен данными между узлами информационной системы (внутренний обмен);
* периодический обмен данных с другими информационными системами (внешний обмен).

Если обмен данными должен осуществляться в режиме, близком к реальному времени, то это будет задача о распределенной базе данных, а не о простой передаче данных.

При анализе задач загрузки и выгрузки данных проектировщик должен рассмотреть:

* каким подсистемам нужен интерфейс выгрузки данных и каков должен быть интерфейс загрузки данных из внешней системы;
* каковы периодичность обмена данными и объем передаваемых данных;
* какая требуется степень синхронизации двух систем;
* каковы возможные методы транспортировки данных;

а также:

* согласовать формат данных для обмена;
* определить зависимости загрузки и выгрузки, например порядок выполнения операций;
* определить мероприятия, которые необходимо выполнить при сбое во время загрузки и выгрузки данных;
* сформулировать правила определения ошибочных записей (при загрузке);
* определить правила регистрации операций передачи и приема данных;
* определить график передачи данных (в большинстве информационных систем эти операции выполняются в ночное время);
* составить график разработки и тестирования собственных утилит или скриптов обмена данными;
* составить график разовой загрузки данных, наследуемых из старой системы, и подготовить методику проверки корректности этой операции.

Следует отметить, что при наследовании данных из старой системы проектировщикам не приходится надеяться на то, что кто-то создаст утилиту, позволяющую достать данные из старой системы, - обычно это становится задачей самих проектировщиков новой системы. Может случиться так, что вам придется работать в жестких условиях, когда не будет возможности выделить время для тестирования новой программы извлечения данных. В этом случае нужно разработать набор тестовых данных. Если в старой системе имеется какое-то средство извлечения данных - используйте его; часто это самый разумный выход.

При загрузке данных из старой системы проектировщики могут столкнуться с большим объемом неочищенных данных - с нарушениями целостности данных, возникшими из-за сбоев системы, «заплаток» разработчиков, иных неприятностей. Возможно, что на вас будет оказано давление с тем, чтобы допустить наличие неочищенных данных в новой системе. Если не принять мер по очистке данных, то, вероятно, большинство спроектированных ограничений целостности нужно будет ослабить, чтобы загрузить хоть какую-то часть данных. Цена такой уступки достаточно высока: данные вы приняли, но ослабленные ранее ограничения уже нельзя восстановить, так как они уже нарушены (это отслеживается СУБД автоматически). Отсюда следует вывод: поддаваться давлению нельзя, так как несколько дней, потраченных на очистку данных, стоят так мало по сравнению с наличием в информационной системе данных, не обладающих элементарной целостностью.

Что делать с данными, которые содержат ошибки или не согласованы? Самое простое решение -  пропускать такие данные, собирать их отдельно и анализировать. Здесь вас могут ждать некоторые проблемы: не все СУБД при загрузке данных их собственными утилитами позволяют в случае  возврата кода ошибки указать запись, на которой произошел сбой. Если это так, то данные следует загружать небольшими порциями, чтобы можно было легче найти запись, которая повлекла сбой. Можно разместить данные с нарушениями целостности в отдельных таблицах, а потом обработать их. Подобную операцию (которую аналитики, как правило, не предусматривают) лучше автоматизировать посредством отдельного компонента. Проектировщикам придется либо озадачить аналитиков исследованием правил корректности данных, либо выполнить эту работу самим, причем необходима помощь опытных пользователей старой информационной системы. Здесь крайне важно найти данные, которые являются надежными, то есть те, которые с большой вероятностью указаны правильно. От таких данных и надо отталкиваться при создании программ проверки корректности данных.

Далеко не все СУБД предоставляют средства, позволяющие выгрузить схему данных, хранимые процедуры и триггеры, но имеется достаточное количество ПО сторонних фирм, способного на это. Самым привлекательным будет ПО, осуществляющее выгрузку указанных объектов в виде скриптов, описывающих их создание. Если ПО позволяет сначала создать объекты данных без ограничений ссылочной целостности, а потом изменить их, добавив эти ограничения, то такой опцией рекомендуется пользоваться всегда. Дело в том, что большинство утилит выгрузки схемы опрашивают системный каталог и генерируют скрипт создания таблиц в том порядке, в котором имена таблиц встречаются в системном каталоге. Но вы не застрахованы от того, что имя таблицы-потомка будет считано раньше имени таблицы-предка. От исправления операций создания объектов данных вы также не застрахованы, так как у большинства СУБД разный синтаксис определения ограничений ссылочной целостности, серверных ограничений, индексов. Например, перечисленные ниже определения ссылочной целостности для таблицы tx:

**create table t(id int primary key, name char(10);**

**create table tx(i int, j int references t(id));**

**create table tx(i int, j int references t(id) on delete no action on update no action);**

**create table tx(i int, j int, foreign key j references t(id) on delete no action on update no action);**

**create table tx(i int, j int, foreign key j references t on delete no action on update no action);**

**create table tx(i int, j int, constraint t\_ref\_cascade foreign key j references t(id) on delete no action on update no action);**

все означают в точности одно и то же, но синтаксис у них разный. Для преобразования предложений SQL в требуемый формат могут быть созданы специальные утилиты. Поскольку в этом случае требуется разбор текста, то весьма нецелесообразно применение инструмента Perl. К сожалению, операция переноса хранимых процедур и триггеров не может быть в такой же степени автоматизирована вследствие существенных различий в языках их написания в разных СУБД. Некоторые СУБД совместимы по синтаксису хотя бы частично, но если этого нет, то большой доли ручной работы не избежать.