# Проектирование баз данных

1. Ведение

В настоящее время нельзя представить вебе информационную систему (ИС) и ядре которой нет базы данных (БД), встроенной или внешней, содержащей все необходимые данные для работы приложения и сохраняющую все результаты работы. От того как спроектирована эта БД во многом зависят все важные параметры ИС, начиная с быстродействия и кончая надежностью. В свете этого, особенно важно уметь проектировать БД таким образом, чтобы обеспечить максимальные характеристики ИС. Проектирование БД всегда было и остается творческой задачей, скорее ремеслом, искусством, нежели строго алгоритмизированной работой. В этом документе мы обсудим основные принципы и приемы проектирования БД, которые практикуются в настоящее время.

* 1. Примеры (синтаксис и нотации)

В данном документе присутствуют примеры скриптов (программ) на языке SQL и различные схемы. Тексты скриптов приведены в соответствии со стандартом ANSI SQL2003 (SQL3) [1], схемы бизнес-процессов в нотации IDEF0 (отступления оговариваются отдельно), а схемы сущностей и таблиц БД в виде, принятом в ORACLE SQL Developer Data Modeler.

* 1. Вопросы терминологии

В связи с развитием теории, практики проектирования БД, а также развитием стандартов СУБД и языка SQL родилось множество наслоений терминов, которые невольно порождают путаницу при изучении проектирования и работе с БД. Основные термины сведены в таблицу 1.

Таблица1. Термины, применяемые в литературе по базам данных.

|  |  |  |
| --- | --- | --- |
| **Реляционная алгебра**  **(Теория)** | **Базы данных (СУБД)**  **(Практика)** | **ISO и ГОСТ**  **(Стандарты)** |
| Отношение | Таблица | Таблица |
| Кортеж | Запись | Строка |
| Атрибут | Поле | Столбец |

В настоящей работе используются термины, принятые в литературе и документации на системы управления базами данных (СУБД), хотя в настоящее время все чаще применяется терминология ГОСТ, несмотря на то, что эта терминология появилась в результате небрежного перевода англоязычных стандартов. Часть определений терминов и сокращений приведены в Приложении А.

* 1. Исторический экскурс

Почти 50 лет прошло с тех пор как Эдгар Кодд опубликовал свою статью, положившую начало реляционным БД. Он показал, что любое представление данных может сводится к совокупности двумерных таблиц, которые он назвал отношениями - relation (англ.). В настоящее время реляционные базы данных являются самыми распространенными, а реляционные СУБД проникли во все сферы жизни человечества. Мы будем далее говорить именно о реляционных базах данных, хотя многие положения справедливы и для других моделей данных.

Вместе с развитием СУБД развивались и изменялись подходы к проектированию БД: сначала методы проектирования базировались на представлении, что СУБД сама по себе является информационной системой, а пользователь обращается к ней на языке SQL, самостоятельно формируя необходимые запросы. Именно поэтому язык SQL проектировался как непроцедурный, декларативный и максимально приближенный к простому английскому.

*В большинстве случаев проектирование рассматривается независимо от приложения. Иначе говоря, интерес представляют сами данные, а не то, как они будут использоваться. Независимость от приложения в этом смысле желательна по той простой причине, что в момент проектирования базы данных обычно еще неизвестны все возможные способы использования ее данных.* [1]

При таком подходе БД должна содержать как можно больше данных, относящихся к какой-либо предметной области, при этом структура БД с точки зрения работы пользователя не оптимизируется, т.к. нет практических критериев оптимизации – нет ясных алгоритмов работы пользователя, нет предпочтительных запросов, неизвестна цель пользователя при работе с данными. Можно предложить только абстрактные, теоретические методы проектирования БД, которые, будут приводить БД к виду, обеспечивающему минимальную логическую избыточность, позволяющую избежать некоторых аномалий обновления.

*Нормализация — это разбиение таблицы на две или более, обладающих лучшими свойствами при включении, изменении и удалении данных. Окончательная цель нормализации сводится к получению такого проекта базы данных, в котором каждый факт появляется лишь в одном месте, т. е. исключена избыточность информации. Это делается не столько с целью экономии памяти, сколько для исключения возможной противоречивости хранимых данных.* [2]

Когда заходит разговор о разбиении таблицы, сразу встает вопрос – откуда она берется, эта самая первая ненормализованная таблица? Вопрос не праздный и имеет вполне определенный ответ – в далекие времена зарождения реляционных БД, проектирование начиналось с определения всех атрибутов (данных) предметной области и из этих атрибутов и получалась первая таблица. Работать с такой таблицей, естественно, было нельзя, и вот тогда вступали в действия правила нормализации или нормальные формы, при помощи которых таблица разбивалась на части (на связанные между собой таблицы), с которыми уже можно было работать.

В настоящее время процесс проектирования БД не рассматривается в отрыве от процесса проектирования ИС. Если раньше СУБД были дорогими, редкими, сложными и громоздкими, то сейчас это не так. СУБД стали широко распространены, среди них есть системы с открытым кодом, такие как MySQL и PostgreSQL, так что и цена тоже перестала быть значимым фактором. Вместе с изменением идеологии использования БД, естественно изменились и принципы проектирования, о них мы и будем говорить в дальнейшем.

1. Основные цели проектирования БД

Прежде чем начать проектирование надо определить основные цели, к которым разработчик должен стремиться в процессе проектирования. Обратите внимание, мы говорим здесь о проектировании БД, как о подзадаче в процессе проектирования ИС и часть требований к ИС непременно оказывает влияние на цели, методы и подходы к проектированию БД, например, проблемы целостности данных сильно связаны с требуемым уровнем отказоустойчивости и быстродействия ИС. Итак, основные цели проектирования БД:

1. Обеспечение хранения в БД всей необходимой информации.
2. Обеспечение возможности получения данных по всем необходимым запросам.
3. Сокращение избыточности и дублирования данных.
4. Обеспечение целостности данных.

Надо понимать, что цель №1 должна быть достигнута непременно, а вот достижение остальных целей — это путь из компромиссов между достижением требуемых характеристик БД, ИС, мощностью аппаратуры, стоимостью, быстродействием, защищенностью и т.д.

Еще несколько слов необходимо сказать по каждой из целей:

С первой целью все ясно – старайтесь не упустить никаких сущностей и никаких атрибутов из предметной области – изучайте предметную область, станьте специалистом по предметной области, станьте продвинутым пользователем проектируемой ИС, станьте, наконец, администратором проектируемой ИС и тогда, может быть, вы ничего не упустите.

Второй пункт, на самом деле, о связях в базе данных – все сущности (таблицы) в БД должны быть связаны между собой так, чтобы понадобился всего один запрос, при помощи которого можно было достать любые данные необходимые для функционирования ИС.

Третий пункт совсем не об экономии памяти, а о том, что *«каждая часть данных должна иметь единственное, непротиворечивое и авторитетное представление в рамках системы»* [5]. Принцип DRY.(Don’t repeat yourself (рус. *не повторяйся*) — это принцип разработки программного обеспечения, нацеленный на снижение повторения информации различного рода. Если какая-либо информация в базе данных будет дублироваться: будьте уверены, содержимое этих дублированных полей разойдутся в соответствии с законом Мерфи [6], а вы получите неработоспособную базу данных.

Четвертая цель – это соблюдение ограничений, накладываемых на данные в БД. Целостность (согласованность) БД — соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам. «Явно заданные правила» это и есть ограничения, которые накладываются на значения полей в БД. В большинстве случаев ограничения целостности определяются особенностями предметной области. Механизмы обеспечения целостности закладываются при проектировании базы данных, и мы обсудим их позднее.

Перечисленные цели кажутся тривиальными, но без постоянной «оглядки» на них создать хорошую БД не удается.

1. Основные задачи и результат проектирования БД
   1. Основные задачи

В процессе проектирования БД разработчик должен решить следующие задачи:

1. Разработка бизнес-процессов
2. Определение ролей пользователей
3. Определение состояний и переходов
4. Определение сущностей
5. Создание схемы сущностей и их связей
6. Создание таблиц и связей в рамках конкретной СУБД
7. Создание объектов БД
8. Разработка скриптов создания БД

Далее мы подробно обсудим каждую из вышеприведенных задач, а сейчас необходимо сделать несколько важных замечаний.

Первые 5 задач относятся к логическому проектированию БД т.е. здесь проектировщик решает общие задачи организации БД, не привязываясь к конкретной СУБД, хотя чаще всего уже на ранних стадиях разработки ИС принимается решение о применяемой СУБД. Поэтому в реальной жизни уже все решения принимаются с учетом особенностей конкретной СУБД. Выбор СУБД почти всегда диктуется не техническими, а экономическими или конъюнктурными соображениями (см. раздел «Выбор СУБД»).

Последние три задачи относятся к техническому проектированию, на этом этапе логические схемы и правила выливаются в конкретные объекты выбранной СУБД. Конструкция таблиц, ограничений, представлений, и других объектов БД оформляется в виде операторов языка SQL или характерного для выбранной СУБД расширения языка SQL. Полученная таким образом программа и является результатом проектирования БД.

* 1. Результат проектирования БД

Результатом проектирования БД является программа (скрипт) на SQL (или его расширении), который содержит:

1. Операторы создания таблиц
2. Операторы создания представлений
3. Операторы создания индексов
4. Операторы создания ролей пользователей
5. Операторы создания хранимых процедур и других объектов БД

Запустив этот скрипт в конкретной СУБД, мы получим полную схему всей БД, роли и механизмы разграничения прав пользователей, механизмы обеспечения целостности, безопасности и быстродействия БД.

1. Выбор СУБД

Несколько слов надо сказать о выборе СУБД. Чаще всего выбор СУБД не является технической задачей, но представляет собой сложный компромисс между множеством факторов не являющихся техническими ограничениями. Например, в банке должна быть промышленная СУБД сопровождаемая разработчиком или всемирно известной компанией, иначе банк не пройдет международный аудит и его действия на зарубежных рынках будут ограничены, или в большой компании есть специалисты, подготовленные к работе с ORACLE и есть ряд соответствующих лицензий, поэтому вопрос о выборе СУБД не стоит. Надо запомнить одно правило:

«Если проектируемая БД небольшая, то любая известная СУБД подойдет, если же БД большая и критичная для работы предприятия, то выбор СУБД чаще всего не связан с техническими вопросами».

1. Логическое проектирование БД

В настоящее время разработка БД неотделима от разработки ИС, с которой эта база будет работать, поэтому проектирование БД начинается вместе с проектированием ИС. В подавляющем большинстве проектов по разработке ИС осуществляется автоматизация бизнес-процессов (БП) [8], поэтому основная работа по анализу и разработке требований сводится к изучению существующих и созданию новых БП.

В этом документе термин «бизнес-процесс» понимается как «совокупность взаимосвязанных действий пользователей и самой ИС, направленных на создание определённого продукта или услуги для потребителей».

Упрощенный бизнес процесс логического проектирования БД в нотации IDEF0 представлен на рис.1.

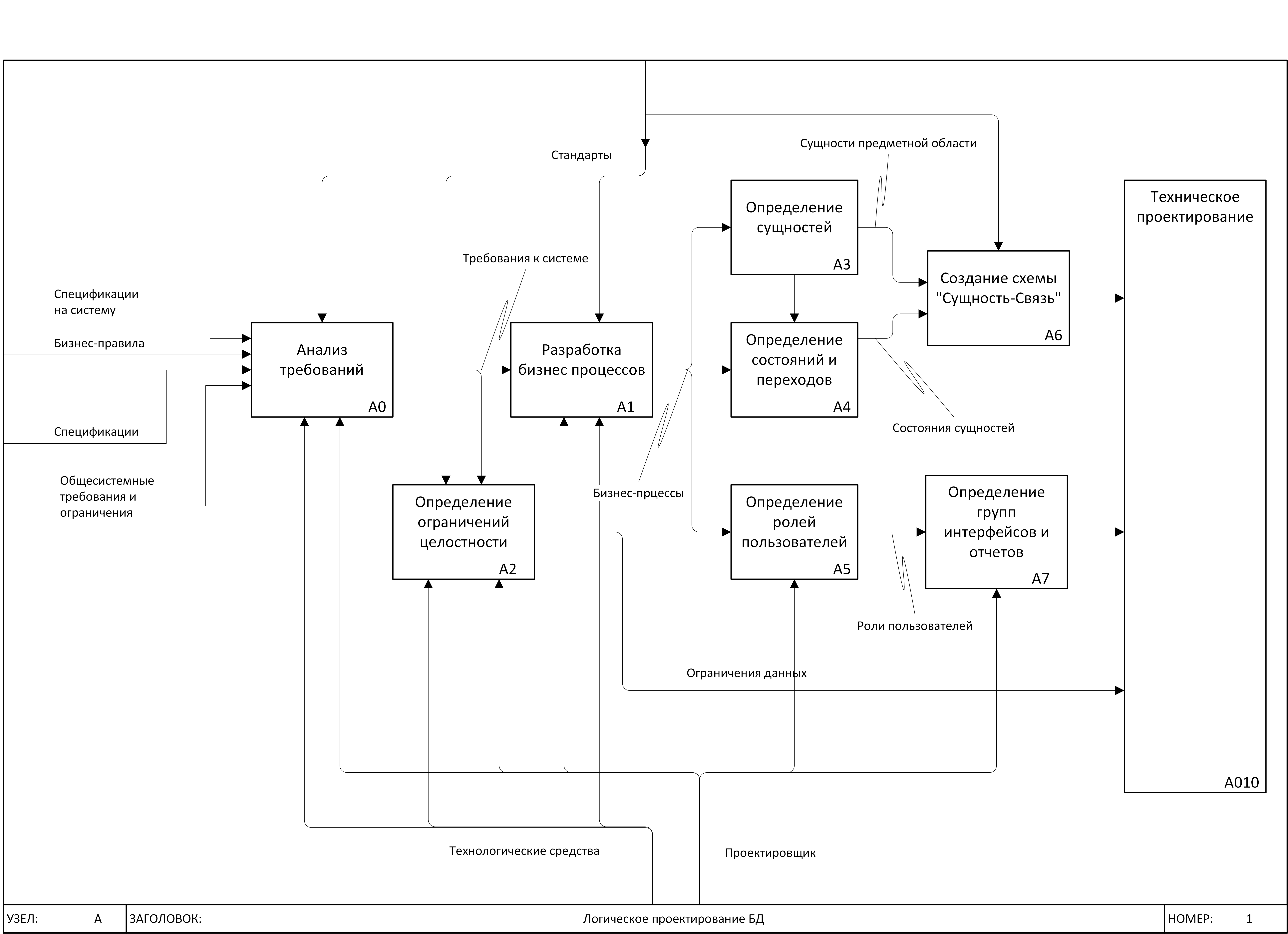
******

Рис.1. Бизнес процесс логического проектирования БД

Сбор информации о предметной области, а также анализ и разработку требований к ИС мы оставим за границами этого документа, тем более, что этот процесс подробно и исчерпывающе раскрыт в литературе [8]. Собственно, проектирование БД лучше всего начать с анализа БП, полученных после разработки требований к ИС.

* 1. Анализ бизнес процессов

Поскольку БП это действия пользователей в ИС и действия самой ИС, нашим первым вопросом будет – с какими объектами (или описаниями объектов) работает пользователь, какие документы поступают на вход БП и какие формируются на выходе? Это и будут сущности нашей ИС. (см. раздел «Определение сущностей»)

Второй, не менее важный вопрос – какие пользователи выполняют те или другие действия в БП? (см. раздел «Создание ролей пользователей»)

И, наконец, третий вопрос – в каких состояниях могут находиться объекты, с которыми работает пользователь в этом бизнес процессе (см. раздел «Создание атрибутов») и как эти состояния изменяются?

* 1. Определение сущностей

На рис 2. приведен фрагмент действующего документа, описывающего БП «Выдача кредита безналичными в другой банк» из проекта автоматизированной банковской системы (нотация BPMN). Посмотрим с какими объектами работают сотрудники банка при выполнении своих должностных обязанностей.

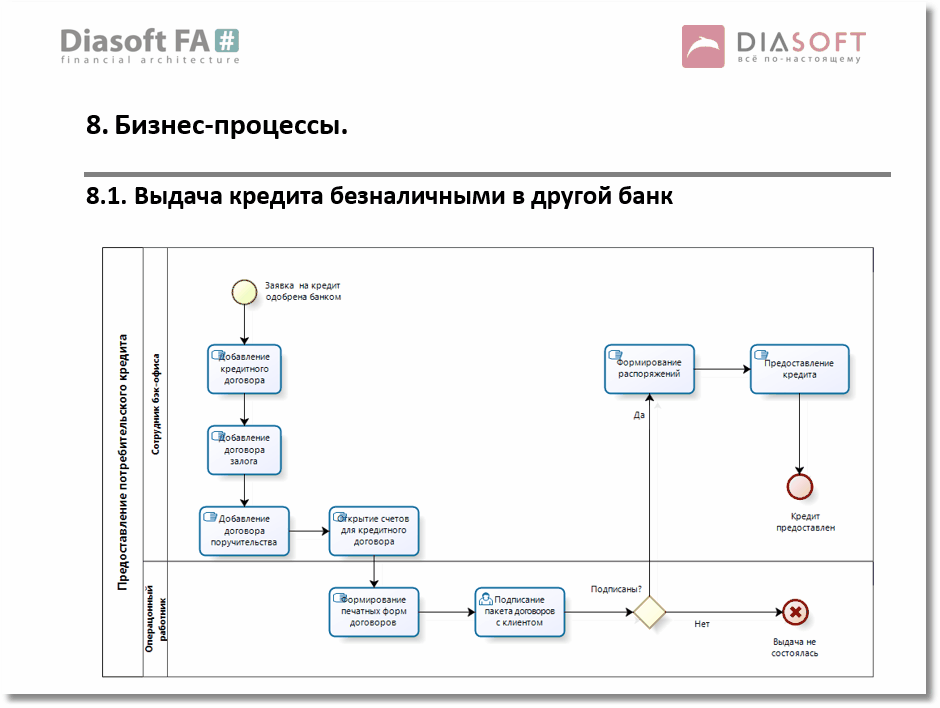
****

Рис.2. Бизнес процесс «Выдача кредита безналичными в другой банк»

Из этих объектов и родятся сущности будущей БД:

* Кредитный договор

Мы изучили предметную область и знаем, что в любом договоре есть как минимум 2 стороны, эти стороны и будут следующими двумя сущностями:

* Банк (кредитор)
* Клиент (заемщик)
* Договор залога
* Счета кредитного договора
* Платежные распоряжения

Рассмотрим еще один пример – БП записи на прием в поликлинике (см. приложение Б). Этот процесс представлен на рис. 3.

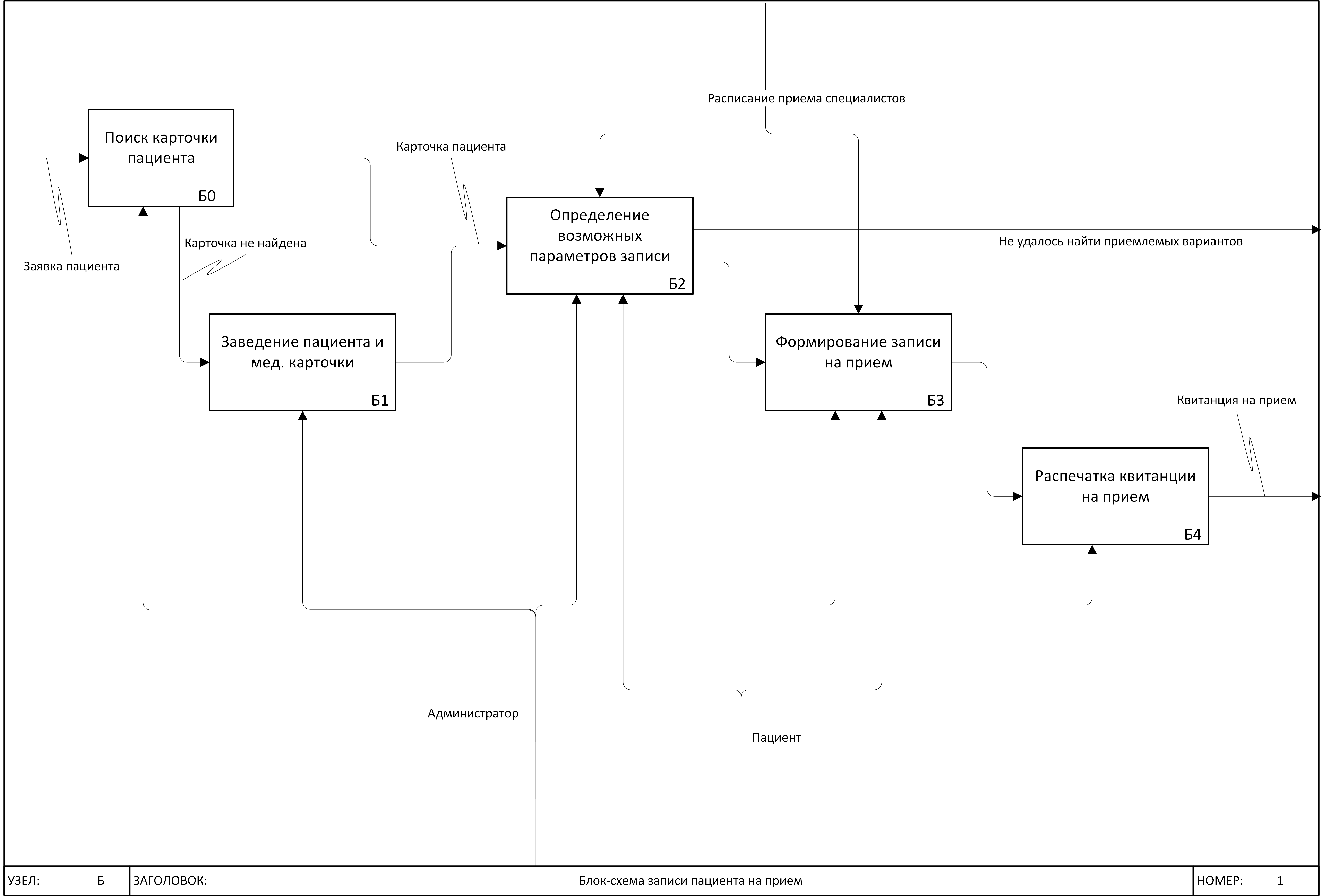


Рис. 3. БП «Запись на прием в поликлинике»

Здесь можно выделить следующие сущности, с которыми работает администратор и пациент:

* *Карточка пациента*

Здесь содержаться результаты приема пациента у различных врачей

* *Пациент*

Данные о пациенте

* *Расписание приема специалистов*

Эта сущность пришла в БП извне… наверно есть другой БП где она формируется

* *Специалист*

Данные о враче-специалисте

* *Запись на прием*

Данные о резервировании времени приема пациента

И, наконец, последний пример – БП приема у врача в поликлинике на рис. 4.

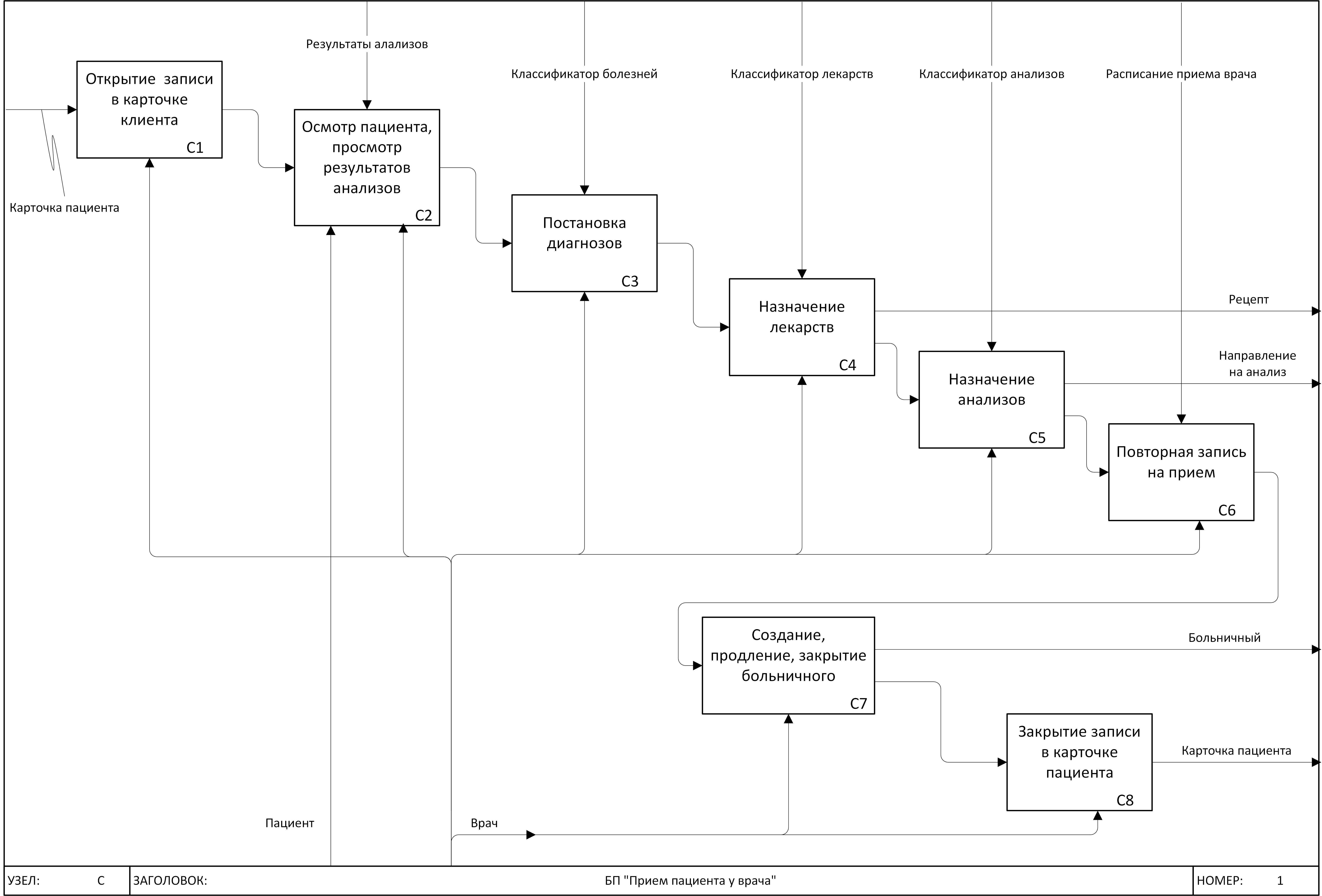


Рис. 4. БП «Прием пациента врачом»

Выделим и из этого БП сущности, с которыми работает врач:

* *Запись в карточке пациента*
* *Запись на прием*

Данные о резервировании времени приема пациента.

* *Классификатор (справочник) болезней*

Эта сущность пришла в БП извне, также, как и справочник анализов и лекарственных препаратов… изучая предметную область выясняем, что сейчас действует «Международная классификация болезней десятого пересмотра МКБ-10 (принята 43-ей Всемирной Ассамблеей Здравоохранения)».

* *Поставленный (уточненный) диагноз*

В соответствии со справочником болезней, анализируя результаты анализов и симптомы, врач ставит диагноз и записывает его в Карточку пациента.

* *Классификатор (справочник) анализов*
* *Классификатор (справочник) лекарственных препаратов*

Аналогично классификатору болезней существуют и другие справочники предметной области.

* *Назначенные анализы*

В соответствии со справочником анализов, врач назначает необходимые анализы и записывает их в Карточку пациента.

* *Результаты анализов*

После выполнения анализов их результаты поступают в систему и привязываются к карточке пациента.

* *Назначенные лекарственные препараты (Рецепт)*

Документ регламентированный законом РФ, удостоверяет возможность платного или бесплатного получения лекарств из перечня Минздрава.

* *Больничный лист*

Документ регламентированный законом РФ, удостоверяет временную неработоспособность пациента.

Из полученного перечня сущностей мы можем заключить, что сущностями чаще всего бывают:

* 1. **объекты** (Кабинет врача, Назначенный лекарственный препарат) - предметы окружающего мира;
  2. **субъекты** (Пациент, Специалист) - группа людей, роль;
  3. **события** (Запись в карточке пациента) - сущность фиксирующая событие, в данном случае это прием врачом пациента;
  4. **документы** (Больничный лист, Рецепт) - материальный носитель с зафиксированной на нём информацией, который имеет реквизиты, и предназначен для общественного использования;
  5. **справочники** (Справочник диагнозов) - перечни возможных типов сущностей или значений атрибутов.
  6. Вопросы для сущностей

После того, как определились сущности предметной области, надо проверить их корректность, для чего существуют специальные вопросы, ответив на которые вы убедитесь в том, что сущности на первой стадии проработки определены правильно.

1. Отражает ли имя сущности суть данного объекта?
2. Есть ли синонимы/омонимы данной сущности?
3. Нет ли уже такой сущности, возможно, под другим именем?
4. Нет ли пересечения с другими сущностями?
5. Имеются ли хотя бы два атрибута?
6. Сущность определена полностью?
7. Имеется ли у сущности хотя бы одна связь?
8. Существует ли хотя бы одна функция по созданию, поиску, корректировке, удалению, и использованию значения сущности?
9. Ведется ли история изменений?
10. Не имеет ли сущность слишком общий смысл?
11. Достаточен ли уровень обобщения, воплощенный в ней?

Разберем эти вопросы более подробно.

1. и 2. Вопросы помогут правильно назвать сущность, чтобы коды ИС и запросы к БД выглядели осмысленно и логично – это очень поможет при верификации и отладке кода программ.
2. Поинтересуйтесь, возможно такая же сущность уже была выделена в результате анализа другого БП, подумайте, полностью ли они идентичны.
3. Не повторяются ли группы атрибутов в разных сущностях и не вынести ли эти группы в отдельные сущности. Например, в сущностях «Студент», «Преподаватель», «Сотрудник» существует группа одинаковых полей: Фамилия, Имя, Отчество, Дата рождения, номер паспорта, адрес и т.д. (то, что называется «Личные данные»). Хорошо бы выделить их в отдельную сущность – «Физическое лицо», тем более, что студент может быть и сотрудником кафедры, а его личные данные будут храниться в одной таблице, а не в двух. Вспомните про принцип DRY и закон Мерфи.
4. Один обязательный атрибут – первичный ключ и нужен еще хотя бы один, для характеристики экземпляра сущности.
5. Подумайте, все ли атрибуты сущности упомянуты и определены.
6. Связана ли сущность с другими сущностями, а если не связана, может быть она просто лишняя?
7. А используется ли вообще данная сущность в проектируемой ИС, и, если нет ни одной функции работающей с ней, может эта сущность не нужна?
8. Нужно ли сохранять данные о предыдущих значениях атрибутов сущности, например, человек, на протяжении всей жизни, меняет несколько раз паспорт, надо ли хранить параметры предыдущих паспортов? В каких-то системах да, в каких-то нет, а если да, то где хранить предыдущие значения?
9. Может в этой сущности заложено слишком много смыслов и надо разбить ее на несколько более простых? Например, стоит разбить сущность «Транспортное средство», на «Гужевой транспорт», «Железнодорожный транспорт», «Автомобильный транспорт» и т.д.?
10. Может в этой сущности заложено слишком узкое понятие? Это вопрос обратный предыдущему - может быть стоит объединить сущности «Гвоздь», «Шуруп», «Саморез» в сущность - «Крепеж»?
    1. От автоматизации процессов к автоматизации компании

Процесс проектирования ИС это не процесс автоматизации одного БП, автоматизировать иногда приходится десятки, а то и сотни БП, составляющих ландшафт предприятия. Сущности, выделенные в процессе анализа различных БП, необходимо объединить в одну или несколько схем, руководствуясь принципом DRY. Объединение необходимо начинать с поиска одинаковых (общих) сущностей присутствующих в разных БП и выстраивать общую схему связывая новые сущности с найденными общими сущностями. Обычно общая схема БД предприятия распадается на несколько «кустов», что связано с административным делением предприятия, понятно, что в одном подразделении решаются схожие, смежные задачи, сущности, которых тесно связаны между собой. Связи между «кустами» обычно осуществляются через небольшое количество общих сущностей, пронизывающих всю ИС и создающих информационный каркас предприятия. После формирования единого перечня сущностей, необходимо снова проверить каждую при помощи вышеприведенных вопросов.

Определение сущностей завершает первый этап разработки БД, но надо помнить, что на дальнейших этапах количество и состав сущностей будет уточнятся и, возможно, дополняться.

* 1. Определение атрибутов сущностей

Следующим важным этапом проектирования является определение атрибутов сущностей

* 1. Первичный ключ – важный атрибут

Одним из атрибутов сущности должен быть первичный ключ – это атрибут однозначно определяющий экземпляр сущности, например, первичным ключом сущности «Студент» может быть номер паспорта, а первичным ключом сущности «Врач специалист» - табельный номер. Ясно, что значения первичного ключа должны быть уникальными и неизменными, иначе с экземплярами сущностей произойдет путаница, но это значит, что номер паспорта как ключ не подойдет - ведь он может меняться при замене паспорта (например, при утрате), а табельный номер при слиянии компаний становится неуникальным. Можно привести огромное количество примеров, которые показывают, что реальные атрибуты не подходят на роль первичного ключа и, хотя теоретики утверждают обратное – «первичным ключом может быть только реальные атрибуты или их суперпозиция» [3], в настоящее время при проектировании БД первичным ключом таблицы (и сущности) всегда служит специальный атрибут, не несущей никакой информации об экземпляре сущности, а только идентифицирующий ее. Такой ключ называется суррогатным – его значение генерируется СУБД искусственно.

Все размышления и утверждения, относящиеся к ключам сущностей переносимы на ключи таблиц реляционной БД, ведь каждой отдельной сущности соответствует отдельная таблица в реляционной базе данных, что следует из определения самой таблицы (см. приложение А).

* 1. Вопросы для атрибутов

1. Является ли наименование атрибута существительным единственного числа, отражающим суть обозначаемого атрибутом свойства?
2. Имеет ли атрибут только одно значение в каждый момент времени?
3. Есть ли необходимость в истории изменений?
4. Отсутствуют ли повторяющиеся значения (или группы)?
5. Описаны ли формат, длина, допустимые значения, алгоритм получения и т.п.?
6. Если значение атрибута является обязательным, всегда ли оно известно?
7. Не может ли этот атрибут быть пропущенной сущностью?
8. Не может ли он быть пропущенной связью?
9. Не является ли его значение вычислимым т.е. производным от других атрибутов?
10. Зависит ли его значение только от данной сущности?
11. Зависит ли его значение только от уникального ключа?
12. Зависит ли его значение от значений некоторых атрибутов, не включенных в уникальный ключ?

Вопросы

Необходимо раскрыть каждый вопрос и привести примеры

1. Связи между сущностями и связи между таблицами
   1. Виды связей между сущностями

Объекты реального мира находятся друг с другом в сложных взаимоотношениях. В БД такие взаимоотношения представляются как связи между сущностями. Под понятием «связь между сущностями А и В» мы будем понимать соответствие одному или нескольким экземплярам сущности А одного или нескольким экземплярам сущности В, например,

* Много студентов учатся в одной группе (Студенты – Группы);
* Студент изучает некоторые предметы (Студенты – Предметы);
* Врач является пациентом своей поликлиники (Врач – Пациент);
* Пациенты записываются на прием к врачам (Пациент – Врач).

Можно различать связи между двумя сущностями по количеству экземпляров присутствующих в связи с разных сторон:

1. Связь много к одному - **много** **студентов** учатся в **одной группе;**
2. Связь один ко многим – **один студент** изучает **много предметов;**
3. Связь один к одному - **врач** является **пациентом**;
4. Связь много ко многим - **пациент** ходит на прием к **врачу,**

т.е. один пациент может ходить на прием ко многим врачам, а к одному врачу может ходить на прием много пациентов.

Первый и второй варианты равнозначны, просто направлены в разные стороны, а третий вариант является вырожденным случаем предыдущих. Рассмотрим, как осуществляются связи первых трех типов между двумя сущностями. В таких сущностях присутствуют особые атрибуты, при помощи которых и можно отыскать связанные экземпляры. Например, в сущности «Студент» есть атрибут содержащий значение определяющее группу, к которой принадлежит студент (этот атрибут содержит номер группы). В общем случае связывающий атрибут первой сущности должен содержать значение однозначно определяющее экземпляр второй сущности. Лучше всего для такого значения подходит первичный ключ.

* 1. Первичные и внешние ключи

Ключи, существующие в таблицах (а каждая таблица соответствует сущности), однозначно определяют экземпляры сущностей:

* первичный ключ - это поле таблицы (атрибут сущности) однозначно определяющий ее запись (ее экземпляр);
* вторичный ключ - это поле таблицы однозначно определяющее ее запись и не являющееся первичным ключом;
* внешний ключ - это поле дочерней таблицы однозначно определяющее запись связанной родительской таблицы, т.е. содержащий значение первичного или вторичного ключа родительской таблицы.

Основное предназначение ключей - связывание сущностей. Пример на рис. 9.

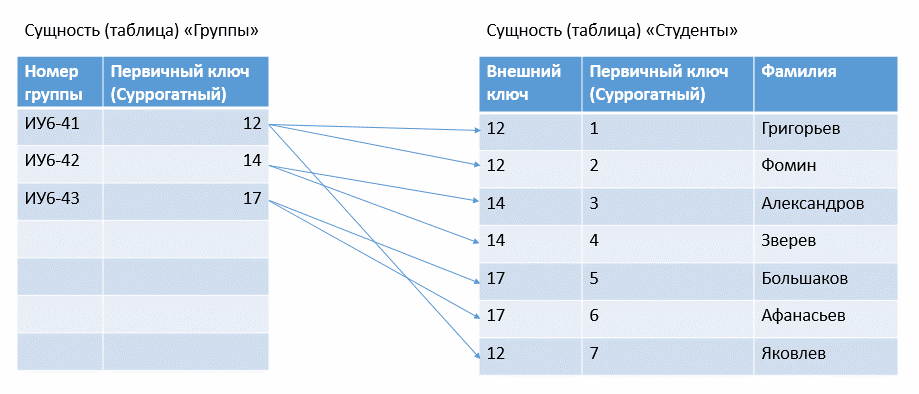


Рис. 9. Связи между сущностями (таблицами)

В таблицах на рис. 9 присутствуют «Номер группы» в таблице «Группа» - это первичный ключ, а в таблице «Студент» поле «Номер группы» - это внешний ключ, определяющий группу, в которой учится студент. Зная экземпляр группы и, следовательно, значение первичного ключа – «Номер группы» (НГ) можно по значениям вторичного ключа равным НГ выбрать всех студентов этой группы и наоборот, по значению вторичного ключа (номеру группы конкретного студента) можно найти нужный экземпляр сущности «Группа».

Так работают связи типа «один ко многим» для таблиц (сущностей) в реляционной БД.

* 1. «Один к одному» - редкий тип связи

Для иллюстрации и понимания сути связи «один к одному» рассмотрим пример связи «Физическое лицо – Паспорт». Конечно, здесь все кажется понятным: паспортные данные – атрибут сущности «Физическое лицо», тут и связи-то никакой нет. Однако, рассмотрим следующие примеры.

Пример 1. ИС Банка.

Процесс банковского обслуживания можно коротко описать так: «Операционист по предъявленному паспорту удостоверяет личность клиента и оказывает ему требуемую услугу». Здесь явно выделяются сущности – клиент, паспорт и услуга. Первые две связаны между собой ОБЯЗАТЕЛЬНОЙ связью «один к одному». Обязательность этой связи заключается в том, что у каждого клиента обязательно есть паспорт, причем только один. Такая связь - это прямое указание, что одна из сущностей является всего лишь атрибутом другой, Т.Е. паспортные данные – атрибут сущности «Физическое лицо».

Пример 2. ИС Паспортного стола.

В этой системе тоже есть связь «Физическое лицо – Паспорт», однако, здесь она не обязательная, потому как в паспортном столе есть еще не выданные паспорта (чистые бланки) и есть физические лица, которые еще не имеют паспортов, но уже зарегистрированы, как подавшие заявление на получение паспорта. В этой системе «Физическое лицо» и «паспорт» – две отдельные сущности, связанные между собой необязательной связью «один к одному». Необязательная связь «один к одному» говорит о том, что мы имеем дело с разными сущностями.

Пациент – карточка пациента – запись в карточке пациента

Физ. Лица – служащие, преподаватели, студенты

Разделение пространства ключа (дополнительная информация в первичном ключе)

* 1. «Много ко многим» - несуществующий тип связи

Еще раз посмотрев на рис. 9 можно понять, что связи между таблицами в реляционной БД бывают только одного типа – «один ко многим», это связано с единственным механизмом реализации связи через первичный и внешний ключ. Между сущностями могут быть и другие связи, так как механизмы связей между сущностями не регламентированы. Однако, с другой стороны, таблицы — это отражение сущностей и, следовательно, сущности тоже не могут иметь никаких других связей, кроме «один ко многим».

В реальной жизни (в реальной предметной области) не существует связей «много ко многим»! Но что делать если проектировщик все-же наталкивается на необходимость связи отличающейся от «много к одному»? Это значит он пропустил какую-то важную, для данной предметной области сущность, которая разрешает «странный» тип связи и преобразует ее в несколько связей «один ко многим». Вот хрестоматийный пример связи «много ко многим» – авторы и книги, каждый автор написал одну или более книг и у каждой книги один или более авторов. В реальной же системе, например, библиотечной ИС, присутствует сущность «Библиография автора» и таким образом получаем связи «один ко многим» (см. рис. 10).

Другой классический пример – преподаватель преподает несколько предметов, а один и тот же предмет может вести несколько преподавателей. Однако, в реальной БД ВУЗА обязательно присутствует сущность «Расписание занятий» связывающая преподавателей и предметы. Ведь только из расписания можно, понять кто и что читает. Таким образом мы опять получаем связи «один ко многим» (см. рис. 10).

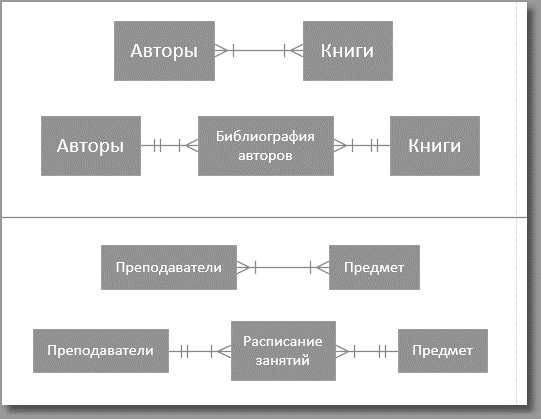


Рис. 10. Примеры связей «много ко многим»

В некоторых источниках упоминаются совсем уж экзотические «тринарные» связи, которые связывают между собой сразу три сущности, например, Продавцы, Покупатели и Товары. Однако, после анализа предметной области становится понятно, что эта связь всего лишь пропущенная сущность «Договор купли-продажи» (см. Рис 11).

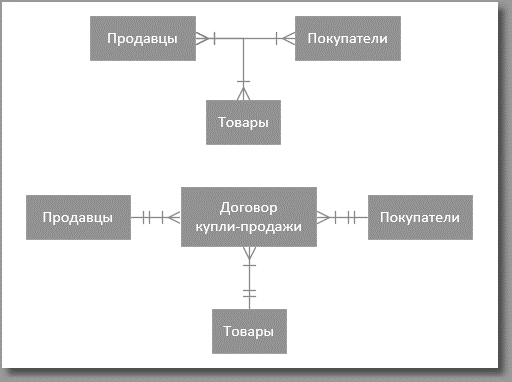


Рис. 11. Пример экзотической связи

Помните – если у вас на схеме появилась связь отличная от типов «один к одному» и «один ко многим», значит вы упустили какую-то важную сущность.

* 1. Вопросы для связей

1. Имеется ли ее описание для каждой участвующей стороны, точно ли оно отражает содержание связи и вписывается ли в принятый синтаксис?
2. Участвуют ли в ней только две стороны?
3. Не является ли связь пропущенной сущностью?
4. Заданы ли степень связи и обязательность для каждой стороны?
5. Допустима ли конструкция связи?
6. Не относится ли конструкция связи к редко используемым?
7. Не является ли она избыточной?
8. Не изменяется ли связь с течением времени?
9. Если связь обязательная, всегда ли она отражает отношение к сущности, представляющей противоположную сторону?

Необходимо раскрыть каждый вопрос и привести примеры

* 1. Диаграмма «Сущность-связь»

Одновременно с определением связей разрабатывается диаграмма «Сущность-связь», которая содержит все сущности и все связи. Начинать разработку этой диаграммы надо с основной сущности, той, которую вы считаете самой главной, той, которая имеет больше всего связей. В случае с системой «Кафедра» эта сущность – «Учебный план», а в системе «Поликлиника» – «Карточка пациента». Определяя связи и отображая новые сущности, проектировщик постепенно получает целостную картину БД, выраженную в диаграмме «Сущность-связь». Несколько следующих рисунков показывают процесс построения диаграммы.

На рис. 12 представлена первая итерация разработки схемы «Сущность-связь» БД «Поликлиника», здесь видно только начало бизнес-процесса: Врач осматривает клиента, записывает в БД симптомы и ставит диагноз. Центральная сущность в схеме «Запись в медкнижке»

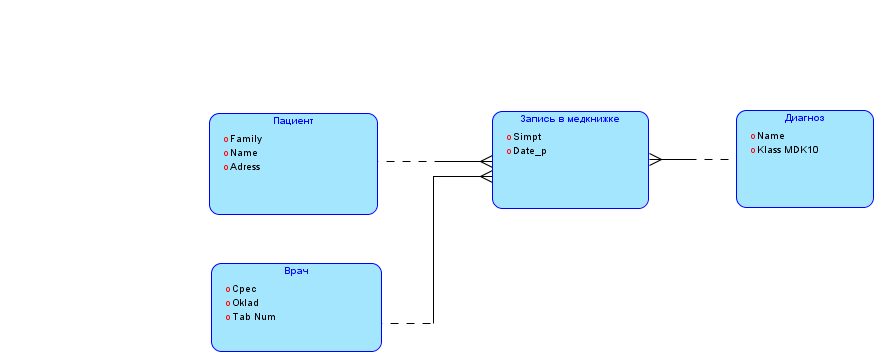


Рис. 12. Схема «Сущность-связь» БД «Поликлиника»

Доработаем схему последовательно присоединяя сущности БД

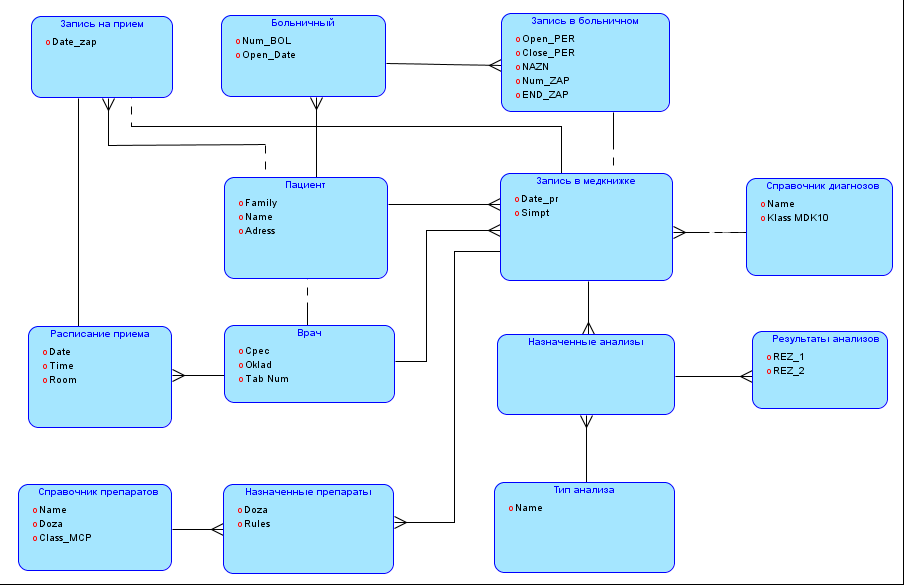


Рис. 13. Полная логическая схема БД для БП «Прием пациента врачом»

Внимательно изучите рис. 13 и вспомните БП «Прием пациента врачом», приведенный на рис. 4, посмотрите еще раз как мы выделяли из этого БП сущности и как определяли связи между этими сущностями. Разберитесь каким образом получили свое отображение на схеме вышеупомянутые сущности и связи. Обратите внимание на атрибуты сущностей, убедитесь, что здесь нет первичных и внешних ключей (они «родятся» потом, в процессе формирования таблиц для конкретной СУБД»). Обратите внимание, что сущность «Назначенные анализы» совсем не содержит атрибутов - она предназначена для связей.

* 1. Формирование схемы таблиц

После проработки схемы «сущность-связь» можно переходить к техническому проектированию БД, т.е. к созданию схемы таблиц и связей между ними. Если логическое проектирование было корректным, эта схема будет очень похожа на схему «сущность-связь», ведь каждой сущности соответствует таблица. Сравните рис. 13 и рис 14. Схема таблиц реляционной базы данных сгенерирована автоматически программой SQL Data Modeler фирмы Oracle, она не зависит еще от реализации (марки) СУБД, ведь здесь реализованы таблицы исходя из общих принципов реляционной модели: появились таблицы, первичные и внешние ключи, при помощи которых осуществляются связи, добавлены ограничения уникальности, индексы и т.д. Часто опытные разработчики минуют фазу составления схемы «сущность-связь» и сразу делают схему таблиц. Такое возможно только при неукоснительном следовании принципу – «Одна сущность – одна таблица» и при большом опыте ручного создания элементов связей.

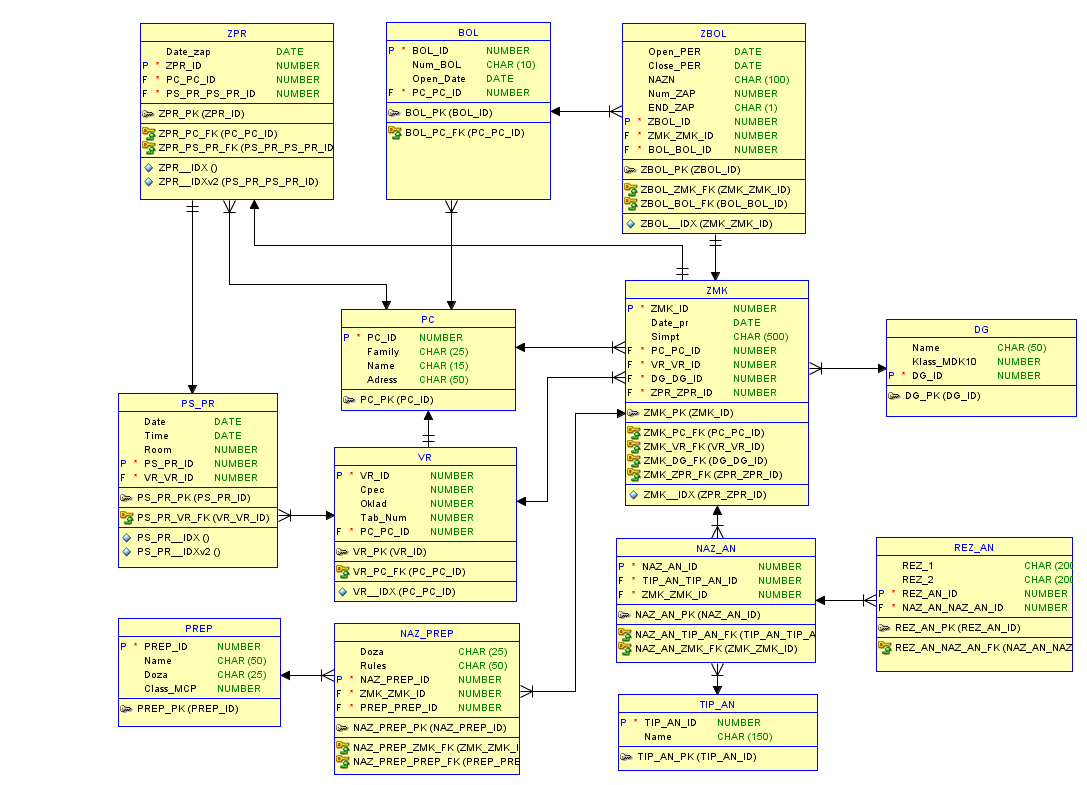


Рис. 14. Полная схема таблиц БД для БП «Прием у врача»

Дальше генератор текста языка SQL программы «ORACLE SQL Developer Data Modeler» по этой схеме сформирует скрипт создания БД.

1. Создание представлений

На вышеприведенной схеме мы видим только таблицы, однако у разработчика реляционных БД есть еще один мощный инструмент, позволяющий улучшить характеристики БД, это – представления.

Представление (англ. view,) — виртуальная (логическая) таблица, представляющая собой поименованный запрос (синоним к запросу), который будет подставлен как подзапрос при использовании представления. Представления скрывают от прикладной программы сложность запросов и саму структуру таблиц БД.

База данных может использоваться разными приложениями, каждое из которых может иметь свой «взгляд» на предметную область, и этот «взгляд» обеспечивается представлениями, показывающими БД в разных разрезах, скрывая физическую сложность как самой структуры, так и преобразований к различным разрезам предметной области. Вот что дает применение представлений:

1. Представления позволяют использовать запрещенные в обычных таблицах элементы, например, вычислимые поля.
2. Использование представлений позволяет отделить прикладную схему представления данных от схемы хранения.

В словаре СУБД, например, очень сложная структура хранения данных, она оптимизирована на скорость выполнения запросов ядра СУБД, а вот для пользователей доступ к словарю осуществляется на основе представлений, чтобы человек легко мог найти необходимые ему данные.

1. С помощью представлений обеспечивается ещё один уровень защиты данных.

С одной стороны, мы закрываем таблицы и данные в них от случайных изменений (по умолчанию данные в представлении нельзя изменить, ведь это не настоящие таблицы, а результат выполнения запроса), с другой стороны мы можем закрыть для некоторых пользователей часть таблицы, разрешив доступ к этой таблице через ограничивающее представление. Например, словарь СУБД изменить вручную практически невозможно, т.к. пользователь видит только представления, которые изменить нельзя.

1. СУБД получает возможность применить к запросу, на котором строится представление, оптимизацию или предварительную компиляцию, а значит представления работают намного быстрее, чем обычные запросы.

Примеры представлений

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 1. Представление – Средний оклад по возрастным категориям сотрудников

CREATE VIEW СР\_Оклад as

SELECT ROUND(EXTRACT(Year from SYSDATE)- EXTRACT(Year from ДATA\_РОЖД),-1) возраст,

-- Вычисляется возраст пациента и округляется до десятков лет

COUNT(\*) кол, ROUND(avg(оклад)) "средний оклад"

FROM "СОТРУДНИКИ"

GROUP BY ROUND(EXTRACT(Year from SYSDATE)- EXTRACT(Year from ДATA\_РОЖД),-1)

ORDER BY 1;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 2. Представление – Курсы валют на заданную дату

CREATE VIEW Курсы as

SELECT КВ."КУРС", ' ', СП."КРАТКОЕ\_ИМЯ" Валюта, ' за ',

КВ."МНОЖИТЕЛЬ" Кол\_единиц, ' ', СП2."КРАТКОЕ\_ИМЯ" Валюта2, КВ."ДАТА"

FROM "КУРСЫ\_ВАЛЮТ" КВ, "СПР\_ВАЛЮТ" СП, "СПР\_ВАЛЮТ" СП2

-- участвуют две таблицы "КУРСЫ\_ВАЛЮТ" "СПР\_ВАЛЮТ" связанные по коду валюты

WHERE КВ."ДАТА" =

(SELECT MAX(КВ2."ДАТА")

FROM "КУРСЫ\_ВАЛЮТ" КВ2

WHERE КВ."ВАЛ1\_ID" = КВ2."ВАЛ1\_ID"

and КВ2."ДАТА" < TO\_date('19.09.2017','DD.MM.YYYY'))

-- подзапрос возвращает дату последнего изменения валюты перед заданной датой

AND КВ."ВАЛ1\_ID" = СП."ВАЛ\_ID"

AND КВ."ВАЛ2\_ID" = СП2."ВАЛ\_ID";

-- таблицы связываются, чтобы найти наименования валют по коду

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Запрос представленный в листинге 2 подробно разбирается в п. 12.3.

1. Создание хранимых процедур
   1. Типы встроенных процедур
2. Процедуры обработки ошибок
3. Процедуры, обеспечивающие целостность и согласованность БД
4. Процедуры реализующие особо сложные запросы
5. Триггеры

Привести примеры процедур. Сделать акцент на том, что процедуры пишутся не на SQL а на расширениях SQL, специфичных для каждой СУБД (PL SQL)

* 1. Триггер

**Триггер** - хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой инициируется СУБД при попытке изменения данных в таблице, с которой он связан. Триггеры применяются для обеспечения целостности данных и реализации сложной бизнес-логики. Все производимые модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера.

* 1. Зачем нужны Триггеры

1. Для автоматической генерации значений полей
2. Для логирования при ограничении доступа
3. Для предотвращения dml операций
4. Для реализации сложных ограничений целостности данных
5. Для организации всевозможных видов аудита и сбора статистики
6. Для оповещения других модулей о том, что делать в случае изменения информации в БД
7. Для реализации бизнес логики
8. Для организации каскадных воздействий на таблицы БД

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 1. Триггер – Заполнение первичного ключа

CREATE OR REPLACE TRIGGER VR\_VR\_ID\_TRG

BEFORE INSERT ON VR FOR EACH ROW

WHEN (NEW.VR\_ID IS NULL)

BEGIN

:NEW.VR\_ID := VR\_VR\_ID\_SEQ.NEXTVAL;

END;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 2. Триггер – Запрет записи пациента не в рабочее время

CREATE OR REPLACE TRIGGER Огр\_записи

BEFORE INSERT ON ZPR

BEGIN

IF TO\_CHAR (SYSDATE, 'HH24:MI') NOT BETWEEN '08:00' AND '18:00'

OR TO\_CHAR (SYSDATE, 'DY') IN ('SAT', 'SUN') THEN

RAISE\_APPLICATION\_ERROR (-20205, 'Запись производится только с 8:00 до 18:00');

END IF;

END Огр\_записи;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 3. Триггер – Сохранение истории записи врача

CREATE OR REPLACE TRIGGER Ист\_вр

AFTER UPDATE OF job\_id, department\_id ON VR

FOR EACH ROW

BEGIN

INSERT INTO job\_history (VR\_id, date\_upd, Oklad, Cpec)

VALUES(:old.VR\_id, sysdate, :old. Oklad, :old. Cpec);

END Ист\_вр;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 4. Триггер – Запрет удаления записи врача

CREATE OR REPLACE TRIGGER Огр\_удал

BEFORE DELETE ON VR

BEGIN

RAISE\_APPLICATION\_ERROR (-20205,

‘Удаление записи о враче невозможна');

END Огр\_удал;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 5. Триггер – Каскадное удаление записей из таблицы «Расписание»

CREATE OR REPLACE TRIGGER Каск\_удал

BEFORE DELETE ON VR

BEGIN

DELETE FROM ZPR

WHERE :old.VR\_id = ZPR.VR\_VR\_id;

END Каск\_удал;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Привести примеры, акцентировать внимание на работе триггера в транзакции и на возможности прерывания работы транзакции.

1. Создание индексов
   1. Что такое индексы

Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и поиск нужной строки путём последовательного перебора строк таблицы занимает много времени. Чтобы ускорить поиск строк используются индексы. Индекс — это вспомогательная таблица, которая формируется из значений одного или нескольких столбцов индексируемой таблицы и указателей на соответствующие строки. Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индексная таблица имеет структуру, оптимизированную под поиск — например, сбалансированного дерева. При разработке БД необходимо выделить часто исполняемые запросы и проанализировав таблицы, используемые в запросах, выделить поля, по которым чаще всего осуществляется поиск. Внимания так же заслуживают поля, по которым производится связывание таблиц. Заметим, что СУБД обычно автоматически создает индексы по первичным ключам, а по наиболее часто используемым внешним ключам индексы должен создать разработчик БД.

Пример создания индекса

* 1. Что нужно знать об индексах

Необходимо запомнить ряд правил, которые обязательны к исполнению при работе с индексами:

1. Индексы создаются для ускорения поиска и выборки.
2. Индексы замедляют все операции кроме выборки.

Все операции с таблицами, кроме выборки, изменяют информацию и, следовательно, ведут к перестроению всех индексов, связанных с этими таблицами, что может занять достаточно большое время.

1. Индексы занимают место.

Для больших таблиц индексы могут занимать много памяти, а если индексов много, то они могут занять места больше, чем сама таблица.

1. Для маленьких таблиц индексы не нужны.

Простой перебор для небольших таблиц (менее 100 записей) эффективней, чем работа с индексом, а значит индексы для таких таблиц не нужны.

1. Для таблиц, которые часто обновляются используйте как можно меньше индексов.

Частое обновление таблицы ведет к частому перестроению индексов и замедляет работу системы.

1. Для временных таблиц индексы не нужны.

Временные таблицы часто изменяются и часто очищаются – это ведет к огромной работе по обновлению индексов, которая сводит на нет все преимущества индексации.

1. Для составного индекса НЕ важен порядок полей в индексе.

В СУБД начала века, действительно, можно было получить различные выигрыши при правильной комбинации полей в составном индексе и это упоминается в разной литературе. В настоящее время все СУБД оптимизированы так, что порядок полей в составном индексе не важен.

1. Разграничение доступа и создание ролей пользователей
   1. Что такое разграничение доступа

В современных СУБД достаточно развиты средства дискреционной защиты. Дискреционное управление доступом (discretionary access control) — разграничение доступа между поименованными субъектами и поименованными объектами. Под *субъектами* здесь понимаются пользователи СУБД, а под *объектами* – объекты БД, такие как таблицы, представления, индексы и т.д., под *доступом* понимается возможность использования данных из объектов БД (содержимое таблиц), так и использование самих объектив (индексы, хранимые процедуры и т.д.). Т.е. каждый пользователь имеет конкретные права на работу с конкретными объектами БД, например, доступ только по чтению к таблицам «Страны», «Города» и «Улицы» или доступ к созданию, изменению и удалению всех объектов БД «Адреса».

В настоящем документе разграничение доступа рассматривается только с точки зрения дискреционной защиты, реализуемой средствами СУБД. В настоящее время все СУБД поддерживают разграничение прав доступа, однако, все они делают это по-разному – всеохватывающую систему дискреционной защиты имеет только ORACLE, поэтому мы рассмотрим проектирование системы доступа на примере СУБД ORACLE.

* 1. Пользователи, их схемы и роли

При создании пользователя СУБД создается и его схема – именованный набор объектов и данных, принадлежащих конкретному пользователю. Понятие схемы примерно совпадает с понятием базы данных. Сразу после создания схема пустая и создать в этой схеме объекты может только хозяин схемы или тот пользователь, которому будут даны права на объекты этой схемы. Права на объекты или данные схемы может дать либо владелец схемы, либо администратор СУБД. При развертывании и эксплуатации большой БД приходится раздавать десятки прав на сотни, а то и тысячи объектов, сотням пользователей.

Чтобы упростить и упорядочить процедуру раздачи прав создаются роли пользователей. По сути, роль - это набор прав на конкретные объекты БД, позволяющие выполнять некий набор БП. Например, все преподаватели–пользователи ИС ВУЗа обладают одинаковыми правами и им присваивается роль – «преподаватель», а некоторые еще и являются кураторами групп и им присваивается роль – «куратор», в результате права этих ролей складываются. Аналогично, некоторые студенты кроме роли «студент» обладают еще и ролью «староста». Таким образом не нужно раздавать права на конкретные объекты БД конечным пользователям, достаточно назначить пользователям набор ролей для выполнения их должностных обязанностей.

* 1. Предопределённые роли пользователей СУБД

В СУБД ORACLE, сразу после ее развертывания автоматически создаются три предопределённые роли:

* CONNECT — работает с данными (не может создавать и изменять объекты БД).
* RESOURCE — работает с объектами БД (может их создавать, модифицировать, удалять и раздавать на них права).
* DBA — работает с объектами СУБД (пользователями, их схемами, табличными пространствами, устройствами хранения данных и т.д.).

Здесь мы видим четкое распределение прав по работе с данными, объектами БД и объектами СУБД.

Пользователь системы, работающий с данными и имеющий все права на эти данные, обладает правами роли CONNECT, эта роль и дается ему при создании.

Пользователь, обладающий правами RESOURCE, это обычно владелец схемы БД, разработчик ИС, имеет все права на все объекты БД. Он также обладает всеми правами роли CONNECT, хотя никогда не правит данные в работающей системе, работать с данными он может только при отладке системы и при устранении сбойных ситуаций

Пользователь, обладающий правами DBA, это владелец всей схемы СУБД, и имеет все права на все объекты всех схем всех пользователей и, соответственно обладает всеми правами ролей CONNECT и RESOURCE.

* 1. Создание ролей пользователей

Присвоив пользователю одну из ролей, описанных в п 1.3. необходимо дать ему права, определяющие его роль в ИС.

На основании анализа БП мы можем определить какие роли пользователей нам нужны для обеспечения работы ИС.

Таблица 2. Роли пользователей в проекте банковской БД

|  |  |
| --- | --- |
| Роль | Зона ответственности (разрешения доступа) |
| Операционный работник / клиентский менеджер | Оформление договора потребительского кредита, предоставление кредита, прием и обработка обращений клиентов |
| Сотрудник бэк-офиса | Выполнение плановых операций по договорам потребительских кредитов (начисление процентов, погашения, оформление досрочных погашений, оформление и погашение просроченных задолженностей, завершение договоров) |
| Кассир | Прием и выдача наличных денежных средств, связанных с предоставлением или погашением потребительского кредита |

Для проекта БД поликлинике также можно выделить роли пользователей исходя из описанных БП (см. рис. 3,4).

Таблица 3. Роли пользователей в проекте БД поликлиники

|  |  |
| --- | --- |
| Роль | Зона ответственности (разрешения доступа) |
| Администратор | Регистрация пациентов, заведение карточки пациента, поддержание в актуальном состоянии справочников ИС |
| Врач специалист | Прием больных, заполнение карточки пациента, оформление больничных |
| Главврач | Составление расписаний, графиков дежурств и отпусков, работа с врачами |

На рис. 15 представлено распределение доступа в соответствии с ролями, перечисленными в таблице 3. Розовым фоном выделены таблицы, к которым имеет полный доступ «Врач специалист», салатовым фоном – «Администратор», без фона - «Главврач»,

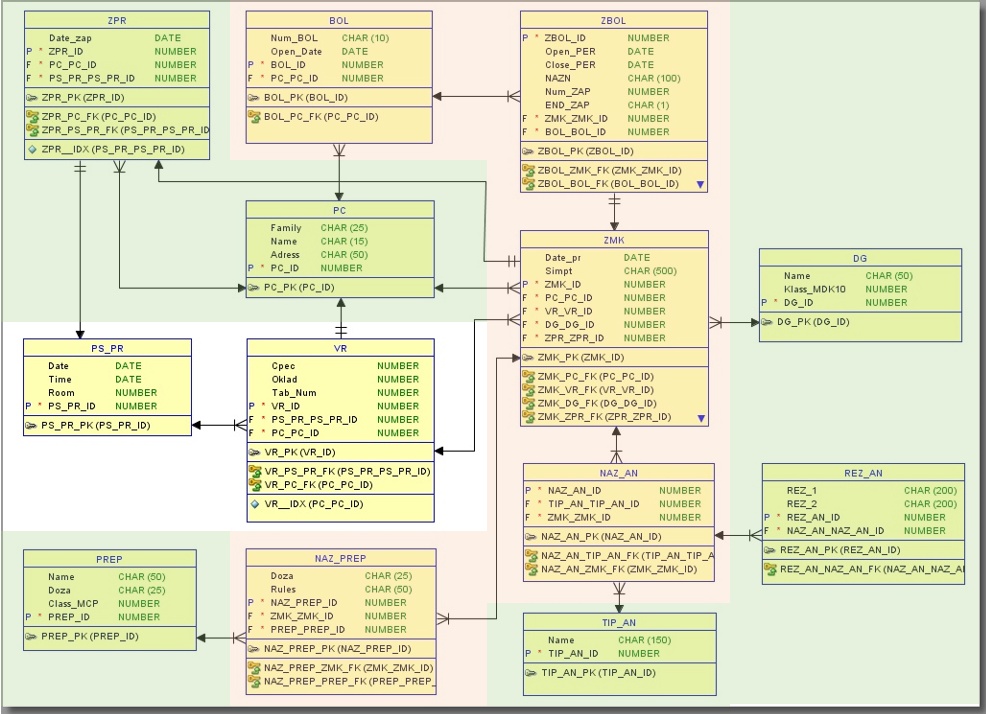


Рис. 14. Полная схема таблиц БД для БП «Прием у врача» с распределением доступа

Работа с ролями пользователей, указанными в таблице 3 представлена в листинге 1.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 1. Создание ролей в проекте БД поликлиники

CREATE ROLE ГлВрач; -- Создание роли «ГлВрач»

GRANT ALL ON VR TO ГлВрач; -- Полные права на таблицу «VR»

GRANT ALL ON PS\_PR TO ГлВрач;

GRANT SELECT ON ZMK TO ГлВрач; -- Права на чтение таблицы «ZMK»

… здесь перечисляются все таблицы ИС, которые доступны роли «ГлВрач» по чтению

GRANT SELECT ON ZMK TO ГлВрач; -- Права на чтение таблицы «TIP\_AN»

GRANT ГлВрач TO smirnova; -- роль «ГлВрач» назначена пользователю «smirnova»

CREATE ROLE Врач;

GRANT ALL ON BOL TO Врач; -- Создание роли «Врач»

GRANT ALL ON ZBOL TO Врач; -- Полные права на таблицу «ZBOL»

GRANT ALL ON ZMK TO Врач;

GRANT ALL ON NAZ\_AN TO Врач;

…

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Обеспечение целостности БД
   1. Типы целостности БД

Целостность базы данных (database integrity) — соответствие имеющейся в [базе данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) информации её внутренней логике, структуре и всем явно заданным правилам. Каждое правило, налагающее некоторое ограничение на возможное состояние базы данных, называется [ограничением целостности](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D1%86%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B8&action=edit&redlink=1) (integrity constraint). В большинстве случаев ограничения целостности определяются особенностями предметной области.

Примеры правил: в неделе семь дней; вес детали должен быть положительным; количество знаков в телефонном номере не должно превышать 15; возраст родителей не может быть меньше возраста их биологического ребёнка и т. д.

Понятие согласованности, или целостности данных является ключевым понятием БД. Одной из важнейших задач, решаемой СУБД, является поддержание в любой момент времени взаимной непротиворечивости, правильности и точности данных, хранящихся в БД. Этот процесс называется обеспечением целостности базы данных.

Очевидно, что ограничения должны быть формально объявлены для [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), после чего СУБД должна отслеживать их выполнение. Объявление ограничений сводится просто к использованию соответствующих средств языка базы данных, а соблюдение ограничений осуществляется с помощью контроля со стороны СУБД операций обновления, которые могут нарушить эти ограничения, и запрещения тех операций, которые их действительно нарушают.

Разделяют несколько видов целостности БД:

1. Сущностная целостность

Поддержание сущностной целостности состоит в недопущении многократного попадания данных об одной сущности в таблицу БД. Если использовать «естественные» уникальные ключи, такие как номер паспорта или СНИЛС то сущностная целостность поддерживается автоматически. При использовании суррогатных первичных ключей добиться этого можно используя уникальные «естественные» вторичные ключи, такие как, уже упомянутые, номер паспорта или номер зачетки и т.д.

1. Доменная целостность

Доменная целостность -- это достоверность записей в конкретном столбце. Она включает в себя ограничения типа данных, ограничения формата при помощи ограничений CHECK и правил, а также ограничения диапазона возможных значений при помощи ограничений FOREIGN KEY, CHECK, DEFAULT, определений NOT NULL и т.д.

1. Ссылочная целостность

Ссылочная целостность гарантирует согласованность значений первичных и внешних ключей во всех таблицах. Обычно поддерживается триггерами, автоматически заполняющими значение ключей или обеспечивающие невозможность их изменения.

1. Пользовательская целостность

Определяется бизнес-правилами, не входящие ни в одну из категорий целостности, эти правила определяются и отслеживаются не на уровне СУБД, а на уровне ИС. Например, баланс банка должен сходится или число проданных и непроданных билетов на авиарейс должно в сумме равняться вместимости самолета. Пользовательская целостность поддерживается при помощи транзакций и использования триггеров.

* 1. Использование транзакций – путь к обеспечению целостности БД.

**Транзакция** — логическая единица работы, состоящая из одного или нескольких операторов манипулирования данными, которую СУБД рассматривает и обрабатывает как неделимое действие, переводящее БД из одного целостного (согласованного) состояния в другое целостное состояние.

Из этого определения видно, что механизм транзакций надо использовать везде, где необходимо обеспечить целостность БД.

Постой пример – добавление записи в таблицы – родительскую и дочернюю

* 1. Использование триггеров – путь к обеспечению целостности БД.

Использование триггеров – это реализация [ограничений целостности](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D1%86%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B8&action=edit&redlink=1)

Ссылки на п 8.3 – примеры триггеров

* 1. Удаление данных

В различной литературе довольно большое место уделяется процессу удаления данных и возникающим вследствие этого нарушениям ссылочной целостности базы данных. Рассматриваются многочисленные методы и способы приведения БД к согласованному состоянию после удаления данных.

В противовес этим источникам необходимо со всей категоричностью заявить, что из БД ничего удалять нельзя. Любое удаление данных из БД приводит к невозможности получения аналитических отчетов за период, в котором были удалены данные. Если из базы данных кафедры ВУЗА удалить, например, отчисленных студентов, то нельзя будет получить списки групп, статистику посещаемости и успеваемости за прошлые годы. Исходя из невозможности удаления данных, надо предусмотреть методы поддержания изменений состояния сущностей, атрибутов и даже связей между сущностями в процессе жизни и развития ИС (см. раздел «Поддержка историчности»), а также методы ускорения работы БД при постоянном росте ее объемов (см. разделы «Секционирование таблиц» и «Кластеризация таблиц»). В редких случаях в процессе эксплуатации ИС приходится удалять некоторые данные, но это делается вне ИС, администратором БД и последствия этих действий целиком ложатся на его плечи.

Конечно, тезис «из БД ничего удалять нельзя» не относится к процессу разработки и отладки ИС, в эти периоды жизненного цикла удаление данных из БД – обычное дело, только надо помнить о согласованности данных.

1. Обеспечение быстродействия БД
   1. Секционирование таблиц

Наиболее **часто решаемой задачей** при секционировании является повышение производительности работы SQL-запросов по выборке и модификации строк таблицы. Это достигается за счет того, что поиск и модификация строк в таблице идут не по всей таблице, а только в ее части (в одной или нескольких секциях). Эти секции могут храниться на разных носителях, отличающихся как быстродействием, так и стоимостью. Важным шагом в создании секционированной таблицы является определение ключа секционирования. В качестве ключа секционирования может выступить столбец или несколько столбцов, относительно значений которых, будет делаться разнесение таблицы на секции. Например, документы, датируемые текущим и предыдущим годом, составляют одну секцию, а документы, датируемые более ранними годами – другую секцию таблицы, т.к. система чаще всего работает с текущей информацией, и аналитические документы чаще всего не собираются на глубину более года.

Секционирование необходимо указывать при создании таблицы, поэтому принципы разделения на секции необходимо продумать и предусмотреть при проектировании БД.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 1. Создание секционированной таблицы товарных чеков

CREATE TABLE sales\_data

(ticket\_no NUMBER,

sale\_year INT NOT NULL,

sale\_month INT NOT NULL,

sale\_day INT NOT NULL)

PARTITION BY RANGE (sale\_year, sale\_month, sale\_day)

(PARTITION sales\_q1 VALUES LESS THAN (2008, 04, 01)

TABLESPACE ts1,

PARTITION sales\_q2 VALUES LESS THAN (2008, 07, 01)

TABLESPACE ts2,

PARTITION sales\_q3 VALUES LESS THAN (2008, 10, 01)

TABLESPACE ts3,

PARTITION sales\_q4 VALUES LESS THAN (2009, 01, 01)

TABLESPACE ts4);

Более подробно про секционирование можно прочитать

<https://oracle-patches.com/блоги/70-базы-данных-oracle/3006-секционированные-таблицы>

<https://www.oracle.com/technetwork/ru/database/partitioning-wp-12c-1896137-ru.pdf>

* 1. Кластеризация таблиц

Еще один способ увеличения быстродействия БД — это кластеризация таблиц…

Кластеризация— это попытка разместить рядом в одном физическом блоке данные тех записей, доступ к которым осуществляется при помощи одинаковых значений индекса.

* 1. Поддержка историчности

В этой главе мы обсудим каким образом осуществляется сохранение истории изменений данных. Итак, у нас есть поле Oklad содержащее размер оклада врача на текущий момент. При изменении значения этого поля необходимо сохранить предыдущее значение в отдельной таблице, которая и будет хранилищем истории изменения данных о врачах. Для осуществления такого сохранения идеально подойдет механизм триггеров. Триггер запоминающий предыдущее значение полей таблицы «VR» приведен на листинге 2.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 2. Представление – Сохранение истории зарплат

CREATE OR REPLACE TRIGGER Ист\_вр

AFTER UPDATE OF job\_id, department\_id ON VR

FOR EACH ROW

BEGIN

INSERT INTO job\_history (VR\_id, date\_upd, Oklad, Cpec)

-- сохранение значений таблицы «VR» в таблице истории «Ист\_вр»

VALUES(:old.VR\_id, sysdate, :old. Oklad, :old. Cpec);

END Ист\_вр;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нахождение зарплаты на дату

Более сложный пример – нахождение курсов валют на заданную дату приведен в листинге

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 2. Представление – Курсы валют на заданную дату

SELECT КВ."КУРС", ' ', СП."КРАТКОЕ\_ИМЯ" Валюта, ' за ',

КВ."МНОЖИТЕЛЬ" Кол\_единиц, ' ', СП2."КРАТКОЕ\_ИМЯ" Валюта2, КВ."ДАТА"

FROM "КУРСЫ\_ВАЛЮТ" КВ, "СПР\_ВАЛЮТ" СП, "СПР\_ВАЛЮТ" СП2

-- участвуют две таблицы "КУРСЫ\_ВАЛЮТ" "СПР\_ВАЛЮТ" связанные по коду валюты

WHERE КВ."ДАТА" =

(SELECT MAX(КВ2."ДАТА")

FROM "КУРСЫ\_ВАЛЮТ" КВ2

WHERE КВ."ВАЛ1\_ID" = КВ2."ВАЛ1\_ID"

and КВ2."ДАТА" < TO\_DATE('19.09.2017','DD.MM.YYYY'))

-- подзапрос возвращает дату последнего изменения валюты перед заданной датой

AND КВ."ВАЛ1\_ID" = СП."ВАЛ\_ID"

AND КВ."ВАЛ2\_ID" = СП2."ВАЛ\_ID";

-- таблицы связываются, чтобы найти наименования валют по коду

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Результат работы запроса представлен на рис. 15

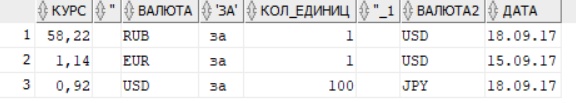


Рис. 15. Курсы валют на заданную (19.09.2017) дату

1. Нормализация базы данных

Во многих источниках большое значение придается нормализации, она объявляется чуть ли не краеугольным камнем проектирования БД, однако в настоящее время разработчики БД обычно и не вспоминают про нормальные формы и сам процесс нормализации. Почему? Давайте разберем это подробно:

1. Понятия нормальных форм относятся к отношениям, переменным отношений, кортежам и другим понятиям реляционной алгебры… переложить их на язык таблиц и СУБД весьма проблематично, например, первая нормальная форма запрещает в отношении одинаковые строки, но не одна СУБД этого не поддерживает (если мы говорим о таблицах).
2. Мнение многих авторов-теоретиках расходятся по поводу «атомарности» полей таблиц и тогда понятие первой нормальной формы становится неопределенным.
3. При использовании таких простых понятий, как суррогатный первичный ключ, мы гарантированно получаем таблицы, которые находятся во второй нормальной форме.
4. Если в качестве единственного потенциального ключа рассматривать первичный суррогатный ключ, то мы получим таблицу в усиленной третьей нормальной форме

При использовании правил и приемов, содержащемся в этом руководстве проектируемая база данных автоматически получается в третьей нормальной форме, а даже теоретики утверждают, что дальнейшая нормализация излишня. С другой стороны, справедливо утверждение:

*Если база данных нуждается в нормализации – значит она неправильно спроектирована.*

Если вы научитесь правильно проектировать базы данных, то нормализовать их не надо.

Список литературы

1. Кляйн К., Кляйн Д., Хант Б. SQL. Справочник, 3\_е издание. – Пер. с англ. – СПб: Символ\_Плюс, 2010. –656 с., ил. ISBN 978\_5\_93286\_165\_3
2. Дейт, К. Дж. Введение в системы баз данных, 8-е издание. Пер. с англ. — М.: Издательский дом "Вильямс", 2005. — 1328 с.: ил. ISBN 5-8459-0788-8 (рус.)
3. Дейт, К. Дж. SQL и реляционная теория. Как грамотно писать код на SQL. – Пер. с англ. –

СПб.: Символ-Плюс, 2010. – 480 с., ил. ISBN 978-5-93286-173-8

1. В.В. Кириллов, Г.Ю. Громов. Введение в реляционные базы данных. — СПб.: БХВ-Петербург, 2009. — 464 с.: ил. ISBN 978-5-94157-770-5
2. Туманов В.Е. Основы проектирования реляционных баз данных. - M.: Национальный Открытый Университет "ИНТУИТ", 2016 (Основы информационных технологий)

ISBN 978-5-94774-713-3

1. Электронный ресурс: «97 вещей, которые должен знать каждый программист» <https://www.gitbook.com/book/97-things-every-x-should-know/97-things-every-programmer-should-know/details/ru>
2. Электронный ресурс: «Закон Мерфи» - <http://murphy-law.net.ru/catalog.html>
3. Вигерс Карл, Битти Джой Разработка требований к программному обеспечению. 3-е изд., дополненное Пер. с англ. — М.: Издательство «Русская редакция»; СПб.: БХВ-Петербург, 2014. — 736 стр.: ил. ISBN 978-5-7502-0433-5
4. Райордан Р. Основы реляционных баз данных Пер. с англ. — М.: Издательско-торговый дом ≪Русская Редакция≫, 2001. — 384 с.: ил. ISBN 5-7502-0150-3
5. Иван Шамаев Электронный ресурс: «Основы PL/SQL» -

<https://ivan-shamaev.ru/pl_sql_functions_procedures_variables_cursors_cycles/>

Приложение А. Глоссарий.

***Информационная система (ИС) –*** система, предназначенная для своевременного обеспечения пользователей надлежащей информацией в рамках определенной предметной области, при этом результатом функционирования информационных систем является информационная продукция — документы, информационные массивы, и информационные услуги.

***База данных (БД)*** ‑ организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей.

***Система управления базами данных*** ***(СУБД)***‑ совокупность программных, лингвистических и методических средств, обеспечивающих управление созданием и использованием БД.

***Таблица*** – объект базы данных хранящий данные об одной сущности: столбцы таблицы (поля) - это атрибуты сущности, а строки таблицы (записи) - это данные о конкретном экземпляре сущности.

***Хранилище (склад) данных (ХД)*** ‑ извлеченные данные из оперативных и архивных БД, преобразованные в формат, приемлемый для сложных нерегламентированных запросов и многомерного анализа.

***Первичный ключ*** – атрибут однозначно определяющий экземпляр сущности.

***Предметная область базы данных (ПО)*** ‑ это та часть реального мира, о которой информационная система хранит, собирает и анализирует информацию.

***Приложение***‑ комплекс программ, автоматизирующий обработку данных для прикладной задачи.

***Сущность (****объект****)*** - предмет, человек или событие, о котором собирается и хранится информация.

***Экземпляр сущности (****кортеж, запись****)*** - единичное значение (экземпляр) объекта.

Пример. Объект “СОТРУДНИК”, экземпляр объекта ‑ данные об Иванове И.И.

***Атрибут (****поле записи, столбец таблицы****)*** ‑ отдельная характеристика (свойство) объекта.

***Домен*** ‑ область значений одного или нескольких атрибутов.

***Индекс*** (англ. index) — объект БД, создаваемый с целью повышения производительности поиска данных.

***SQL***(*structured query language* - «язык структурированных запросов») - формальный непроцедурный язык программирования, применяемый для создания, модификации и управления данными в произвольной реляционной базе данных, управляемой соответствующей СУБД.

**Бизнес-процесс (БП)** — это совокупность взаимосвязанных действий пользователей и самой ИС, направленных на создание определённого продукта или услуги для потребителей.

Приложение Б. Большой пример.

В настоящем документе рассмотрен пример проектирования БД на основе создания взаимосвязанных фрагментов ИС поликлиники. Важно, что эта предметная область всем знакома и не требует специального изучения. Здесь будут проанализированы всего два бизнес-процесса

* Запись на прием к врачу
* Прием у врача

Простейшие схемы бизнес процессов приведены ниже. Примерно такие эскизы получаются после интервьюирования персонала предприятия.

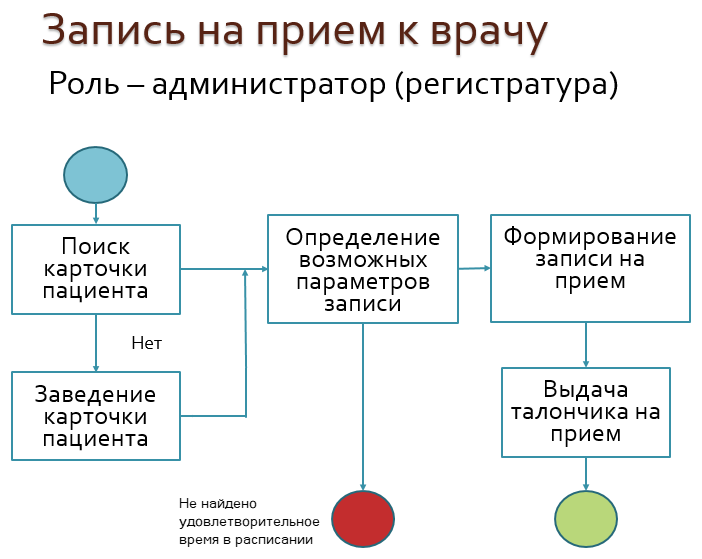


Рис Б1. Бизнес-процесс записи пациента на прием

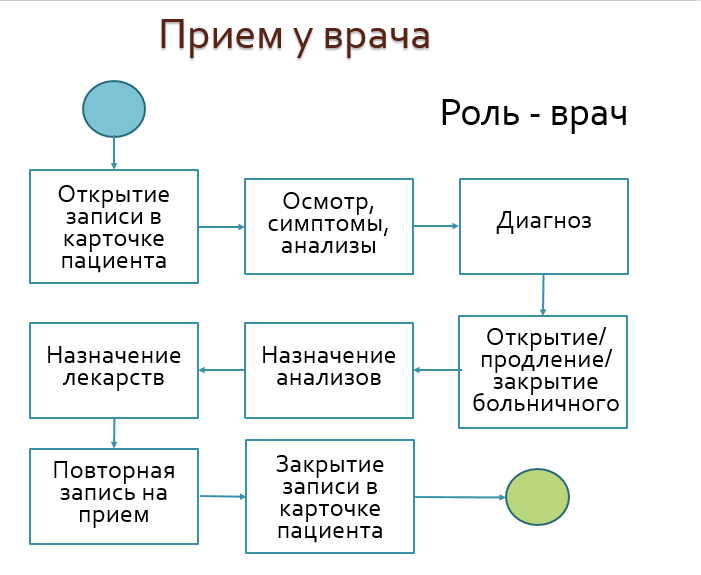


Рис Б2. Бизнес-процесс приема пациента у врача