1. **Просмотр View. Действия над просмотром (срезы). Привести примеры на языке SQL.**

**2.9.1CREATE VIEW - создать новый вид на таблицу данных**

**Синтаксис**

CREATE VIEW [имя вида](http://aam.ugpl.de/node/984#tabname)   
[ **(** [поле](http://aam.ugpl.de/node/984#feldname) [,...] **)** ] AS  
[SELECT-команда](http://aam.ugpl.de/node/814#select) [;](http://aam.ugpl.de/node/984#semikol)

**Описание:**

CREATE VIEW создает новый вид (взгляд) на одну или несколько таблиц базы данных в форме новой виртуальной (находящейся в только в оперативной памяти) таблицы. В команде CREATE VIEW поддерживается, в зависимости от производителя базы данных, только часть возможностей команды SELECT.

Пример:

CREATE VIEW mydetail (

my\_ material,

my\_ description

) AS

SELECT

material,

description

FROM detail

WHERE price > 0;

Создает новый вид на таблицу 'detail', содержащий только поля ' material ' и ' description' из строк у которых price>0.

**2.9.2 DROP VIEW - удалить вид на таблицу данных**

**Синтаксис**

DROP VIEW [имя](http://aam.ugpl.de/node/984#tabname);

**Описание**

DROP VIEW удаляет вид на таблицы данных созданный до этого CREATE VIEW.

Пример:

DROP VIEW myorders;

Удаляет созданный до этого вид на таблицы данных с именем 'myorders'

1. **Концепции наборов DBTG в сетевой модели.**

Концепция *сетевой модели данных* связана с именем Ч. Бахмана, известного специалиста в области обработки данных, который оказал определяющее влияние на создание проекта DBTG CODASYL (1971 год). *Сетевая модель данных* является моделью объектов-связей, где допускаются только *бинарные связи* типа "многие-к-одному", что позволяет использовать для представления данных простую модель ориентированных графов. В некоторых определениях *сетевой модели* допускаются связи типа "многие-ко-многим", но требование бинарности связи остается в силе.

Для *сетевой модели* не существует общепринятой терминологии. Далее используется сложившая к настоящему времени группа понятий и терминов, которые используются для описания элементов *сетевой модели*.

Для моделирования представления данных в *сетевой модели* используются следующие элементы данных:

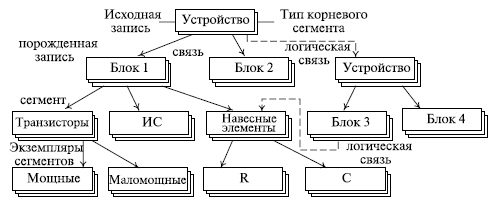
* простое поле (элемент данных, итем) - наименьшая единица структуры данных, имеет уникальное имя, размер и тип: (табельный номер служащего);
* множественное поле (*агрегат данных*, периодическая группа) - поименованная совокупность простых полей или агрегатов; (простой агрегат: Дата = (день, месяц, год)), (составной агрегат: Организация = (наименование, адрес = (почтовый\_индекс, город, улица, дома\_номер))), (повторяющаяся группа: зарплата (12) = (ФИО, оклад));
* запись (группа данных) - поименованный агрегат, который не входит в состав никакого другого агрегата и представляет сущность ПО БД (тип записи);
* групповое отношение (связь, набор) - *иерархическое отношение* между различными записями (графическое представление группового отношения в *сетевой модели* называется диаграммой Бахмана);
* БД - совокупность записей различного типа, объединенная системой групповых отношений различной направленности.

1. **Иерархические структуры хранения данных IMS и методы доступа.**

*Иерархические модели* баз данных исторически возникли одними из первых. В качестве примера рассмотрим иерархическую *БД* —*Information* *Management* Systems (*IMS*), разработанную в 1967 году. Существуют соответствия терминов, описывающих *сетевые модели* *БД* КОДАСИЛ и иерархическую *IMS*, в частности: тип *набора* — *связь*; тип *записи* — тип сегмента; *запись* — владелец*набора* — исходный сегмент; *запись* — член *набора* — порожденный сегмент и т. д. ([рис. 11.2](http://www.intuit.ru/studies/courses/2264/227/lecture/5903?page=2#image.11.2)).

На вершине *иерархической структуры* только один тип сегмента — тип *корневого сегмента*. *Связь* между типами сегментов указывают стрелкой. *Запись* *IMS* (или *запись* *иерархической базы данных*) означает, что экземпляр корневого сегмента связан со всеми экземплярами *записей* нижнего уровня.

В иерархической структуре любой тип *записи* не может иметь более одного владельца (порожденный сегмент). Поэтому и не нужно вводить понятие "тип *набора* ".



**Рис. 11.2.**Структура иерархической модели баз данных

Была рассмотрена *физическая* *база данных*. Однако существуют еще и *логические* отношения, связывающие две (или более) физические *БД* дерева ([рис. 11.2](http://www.intuit.ru/studies/courses/2264/227/lecture/5903?page=2#image.11.2)). С введением этого понятия могут быть связаны два исходных сегмента: один из них, находящийся в той же базе данных, что и рассматриваемый сегмент, называют физически исходным сегментом, другой — логически исходным сегментом. Логические связи на [рис. 11.2](http://www.intuit.ru/studies/courses/2264/227/lecture/5903?page=2#image.11.2) показаны пунктирными линиями. С помощью иерархических структур и логических отношений можно представить и некоторые сетевые структуры. Существуют некоторые ограничения на взаимное расположение физически и логически исходных и порожденных сегментов.

В памяти ЭВМ файлы *иерархической базы данных* *IMS* могут быть представлены четырьмя способами: *HSAM* — последовательный*метод доступа*, HISAM — индексно-последовательный, HDAM — *прямой*, HI *DAM* — индексно-прямой2.

При последовательном методе доступа *дерево* *БД* просматривается сверху вниз и слева направо. Для поиска нужной *записи*необходимо просматривать все *дерево*. Команды включения, удаления и замены сегментов ( *записей* ) не допускаются.

При индексно-последовательном способе *записи* *БД* хранятся в цепочке блоков памяти, в качестве ключей используются элементы корневого сегмента. Каждая *запись* расположена в одном блоке первичной области и при необходимости переходит в специальную область переполнения.

В случае прямого метода доступа *загрузка* сегментов в базу данных осуществляется в произвольном порядке и обращение к ним не последовательное, как в предыдущем способе, а прямое — *по* значению ключа. При удалении некоторых сегментов*пространство* памяти может быть использовано повторно.

В индексно-прямом методе доступа используются корневые *сегменты*, расположенные *по* возрастанию ключа в так называемой начальной базе данных и указывающие на те или иные рабочие *сегменты* *базы данных*.

1. **Даталогическое проектирование. Понятие функциональной зависимости (ФЗ). Полная и транзитивная ФЗ.**

*Функциональная зависимость (ФЗ)* – это соотношение проекций R[A] и R[B] при котором в каждый момент времени любому элементу R[A] соответствует только один элемент проекции R[B], входящий в вместе в какой-либо кортеж отношения R.

*Полная функциональная зависимость (ПФЗ)*. Набор атрибутов В функционально зависит от А, и не зависит функционально от любого подмножества А.

*Транзитивная функциональная зависимость (ТФЗ)*. Пусть существует набор атрибутов С, таких что:

1. С не является подмножеством А;
2. С не включает в себя В;
3. Существует функциональная зависимость С от А ();
4. Не существует функциональная зависимость А от С ();
5. Существует функциональная зависимость B от C ().

*Детерминант отношения.* Если в отношении существуют несколько ФЗ, то каждый атрибут или набор атрибутов, от которого зависит другой атрибут, называется детерминантом отношения.

Выявление смысловой зависимости между данными – один из способов формализации смысловой информации о данных.

Функциональная зависимость описывает связь типа многие-к-одному между атрибутами отношения, где много – детерминант функциональной зависимости. Функциональная зависимость является семантическим свойством атрибутов отношения.

Если в отношении R, содержащем атрибуты A и B, атрибут B функционально зависит от атрибута A (А является детерминантом атрибута B)

*A*→*B*,

то в каждом кортеже этого отношения каждое конкретное значение атрибута A всегда связано только с одним значением атрибута B. Атрибуты A и B могут быть составными атрибутами.

Особенности функциональных зависимостей, лежащие в основе процесса нормализации:

• функциональная зависимость является специализированным правилом целостности – она накладывает ограничения на допустимые значения атрибутов отношений; эту особенность можно использовать при обновлении БД, т.к. зная какие функциональные зависимости есть в отношении, можно проанализировать нарушат ли новые данные целостность данных отношения;

• функциональная зависимость является обобщением понятия потенциального ключа; функциональные зависимости позволяют определить все потенциальные ключи отношения (и соответственно – первичный ключ): все атрибуты отношения, которые не являются частью первичного (или потенциального) ключа, должны функционально зависеть от этого ключа; если не все остальные атрибуты отношения зависят от некоторого детерминанта, то этот детерминант не является потенциальным ключом этого отношения. Одни функциональные зависимости подразумевают другие зависимости. Для данного множества зависимостей S замыканием S+ называется множество всех функциональных зависимостей, подразумеваемых зависимостями множества S. Множество S обязательно является подмножеством собственного замыкания S+. Правила логического вывода Армстронга (аксиомы Армстронга) обеспечивают исчерпывающую и полную основу для вычисления замыкания S+ для заданного множества S:

- рефлексивность *X*→ *X;*

- пополнение *X*→*Y* => *XZ*→*Y;*

- аддитивность (*X*→*Y*)*and*(*X*→*Z*) => *X*→*YZ;*

- проективность *X*→*YZ* => *X*→*Y;*

- транзитивность (*X*→*Y*)*and*(*Y* →*Z*) => *X*→*Z;*

- псевдотранзитивность (*X*→*Y*)*and*(*YZ* →*W*) => *X Z* →*W;*

Два множества функциональных зависимостей S1 и S2 эквивалентны тогда и только тогда, когда они являются покрытиями друг друга, то есть S1+=S2+.

Каждое множество функциональных зависимостей эквивалентно, по крайней мере, одному неприводимому множеству. Множество функциональных зависимостей является неприводимым, если:

• каждая функциональная зависимость этого множества имеет одноэлементную правую часть;

• ни одна функциональная зависимость множества не может быть устранена без изменения замыкания этого множества;

• ни один атрибут не может быть устранен из левой части любой функциональной зависимости без изменения замыкания множества.

Если I является неприводимым множеством, эквивалентным множеству S, то проверка выполнения функциональных зависимостей из множества I автоматически проверяет выполнение всех функциональных зависимостей множества S.

1. **Сетевая модель. Базовые объекты. Языки описания данных и манипулирования данными.**

Стандарт сетевой модели впервые был определен в 1975 году организацией CODASYL (Conference of Data System Languages), которая определила базовые понятия модели и формальный язык описания.

Базовыми объектами модели являются:

* элемент данных;
* агрегат данных;
* запись;
* набор данных,

**Элемент данных**- то же, что и в иерархической модели, то есть минимальная информационная единица, доступная пользователю с использованием СУБД.

**Агрегат данных**- соответствует следующему уровню обобщения в модели. В модели определены агрегаты двух типов: агрегат типа *вектор*и агрегат типа*повторяющаяся группа.*

Агрегат данных имеет имя, и в системе допустимо обращение к агрегату по имени. Агрегат типа вектор соответствует линейному набору элементов данных.

Записью называется совокупность агрегатов или элементов данных, моделирующая некоторый класс объектов реального мира. Понятие записи соответствует понятию «сегмент» в иерархической модели. Для записи, так же как и для сегмента, вводятся понятия типа записи и экземпляра записи.

Следующим базовым понятием в сетевой модели является понятие «Набор».

**Набор**- это двухуровневый граф, связывающий отношением «один-ко-многим» два типа записи.

Набор фактически отражает иерархическую связь между двумя типами записей. Родительский тип записи в данном наборе называется владельцем набора, а дочерний тип записи — членом того же набора.

Для любых двух типов записей может быть задано любое количество наборов, которые их связывают. Фактически наличие подобных возможностей позволяет промоделировать отношение «многие-ко-многим» между двумя объектами реального мира, что выгодно отличает сетевую модель от иерархической. В рамках набора возможен последовательный просмотр экземпляров членов набора, связанных с одним экземпляром владельца набора.

Между двумя типами записей может быть определено любое количество наборов: например, можно построить два взаимосвязанных набора. Существенным ограничением набора является то, что один и тот же тип записи не может быть одновременно владельцем и членом набора.

Среди всех наборов выделяют специальный тип набора, называемый «Сингулярным набором», владельцем которого формально определена вся система. Сингулярный набор изображается в виде входящей стрелки, которая имеет собственно имя набора и имя члена набора, но у которой не определен тип записи «Владелец набора».

Сингулярные наборы позволяют обеспечить доступ к экземплярам отдельных типов данных, поэтому если в задаче алгоритм обработки информации предполагает обеспечение произвольного доступа к некоторому типу записи, то для поддержки этой возможности необходимо ввести соответствующий сингулярный набор.

В общем случае сетевая база данных представляет совокупность взаимосвязанных наборов, которые образуют на концептуальном уровне некоторый граф.

**Язык описания данных в сетевой модели**

Язык описания данных в сетевой модели имеет несколько разделов:

* описание базы данных — области размещения;
* описания записей — элементов и агрегатов (каждого в отдельности);
* описания наборов (каждого в отдельности).

SCHEMA IS <Имя БД>.

AREA NAME IS <Имя физической области>.

RECORD NAME IS <Имя записи (уникапьное)>

Для каждой записи определяется способ размещения экземпляров записи данного типа:

LOCATION MODE IS'{DIRECT (напрямую)

CALC <Имя программы> USING <[Список пер.>]

DUPLICATE ARE [NOT] ALLOWED

VIA <Имя на6ора> SET (рядом с записями владельца)

SYSTEM (решать будет система)}

Каждый тип записи должен быть приписан к некоторой физической области размещения:

WITHIN <Имя области размещения> AREA

После описания записи в целом идет описание внутренней структуры:

<Имя уровня> <Имя данного> <Шаблон> <Тип>

Номер уровня определяет уровень вложенности при описании элементов и агрегатов данных. Первый уровень — сама запись. Поэтому элементы или агрегаты данных имеют уровень начиная со второго. Если данное соответствует агрегату, то любая его составляющая добавляет один уровень вложенности.

Если агрегат является вектором, то он описывается как <Номер уровня> <Имя агрегата>.<Номер уровня> <Имя 1-й сост.> а если — повторяющейся группой, то следующим образом:

<Номер уровня> <Имя агрегата>.OCCURS <N> TIMES

где N — среднее количество элементов в группе.

Описание набора и порядка включения членов в него выглядит следующим образом:

SET NAME IS <Имя набора>:

OWNER IS (<Имя владельца> SYSTEM).

Далее указывается порядок включения новых экземпляров члена данного набора в экземпляр набора:

ORDER PERMANENT INSERTION IS {SORTED | NEXT | PREV | LAST FIRST}

После этого описывается член набора с указанием способа включения и способа исключения экземпляра — члена набора из экземпляра набора.

MEMBER IS <Имя члена набора> {AUTOMATIC | MANUAL} {MANDATORY OPTIONAL} KEY IS (ACCENDING | DESCENDING) <Имя элемента данных>

При автоматическом включении каждый новый; экземпляр члена набора автоматически попадает в текущий экземпляр набора в соответствии с заданным ранее Порядком включения. При ручном способе экземпляр члена набора сначала попадает в БД, а только потом командой CONNECT может быть включен в конкретный экземпляр набора.

Если задан способ исключения MANDATORY, то экземпляр записи, исключаемый из набора, автоматически исключается и из базы данных. Иначе просто разрываются связи.

*Внешняя модель*при сетевой организации данных поддерживается путем описания части общего связного графа.

**Язык манипулирования данными в сетевой модели**

Все операции манипулирования данными в сетевой модели делятся на *навигационные операции*и *операции модификации.*

Навигационные операции осуществляют перемещение по БД путем прохождения по связям, которые поддерживаются в схеме БД. В этом случае результатом является новый единичный объект, который получает статус *текущего объекта.*

Операции модификации осуществляют как добавление новых экземпляров отдельных типов записей, так и экземпляров новых наборов, удаление экземпляров записей и наборов, модификацию отдельных составляющих внутри конкретных экземпляров записей. Средства модификации данных сведены в табл. 3.1:

***Таблица 3.1.****Операторы манипулирования данными в сетевой модели*

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Операция** | **Назначение** |  |
|  | READY | Обеспечение доступа данного процесса или пользователя к БД (сходна по смыслу с операцией открытия файла) |  |
|  | FINISH | Окончание работы с БД |  |
|  | FIND | Группа операций, устанавливающих указатель найденного объекта на текущий объект |  |
|  | GET | Передача найденного объекта в рабочую область. Допустима только после FIND |  |
|  | STORE | Помещение в БД записи, .сформированной в рабочей области |  |
|  | CONNECT | Включение текущей записи в текущий экземпляр набора |  |
|  | DISCONNECT | Исключение текущей записи из текущего экземпляра набора |  |
|  | MODIFY | Обновление текущей записи данными из рабочей области пользователя |  |
|  | ERASE | Удаление экземпляра текущей записи |  |
|  |  |  |  |

В рабочей области пользователя хранятся шаблоны записей, программные переменные и три типа указателей текущего состояния:

* текущая запись процесса (код или ключ последней записи, с которой работала данная программа);
* текущая запись типа записи (для каждого типа записи ключ последней записи, с которой работала программа);
* текущая запись типа набор (для каждого набора с владельцем Т1 и членом Т2 указывается, Т1 или Т2 были последней обрабатываемой записью).

При необходимости возможно описание элементов данных, которые не принадлежат непосредственно данной записи, но при ее обработке часто используются. Для этого используется тип VIRTUAL с обязательным указанием источника данного элемента данных.

RECORD Цены

02 Цена TYPE REAL

02 Товар VIRTUAL

SOURCE IS Товары.НаименованиеТовара

OF OWNER OF Товар-Цены SET

Наиболее интересна операция поиска (FIND), так как именно она отражает суть навигационных методов, применяемых в сетевой модели. Всего существует семь типов операций поиска:

1. По ключу (запись должна быть описана через CALC USING ...):

FIND <Имя записи> RECORD BY CALC KEY <Имя параметра>

1. Последовательный просмотр записей данного типа:

FIND DUPLICATE <Имя записи> RECORD BY CALC KEY

1. Найти владельца текущего экземпляра набора:

FIND OWNER OF CURRENT <Имя набора> SET

1. Последовательный просмотр записей—членов текущего экземпляра набора:

FIND (FIRST | NEXT) <Имя записи> RECORD IN CURRENT <Имя набора> SET

1. Просмотр записей—членов экземпляра набора, специфицированных рядом нолей:

FIND [DUPLICATE] <Имя записи> RECORD IN CURRENT <Имя набора> SET USING <Список полей>

1. Сделать текущей записью процесса текущий экземпляр набора:

FIND CURRENT OF <Имя набора> SET

1. Установить текущую запись процесса:

FIND CURRENT OF <Имя записи> RECORD

1. **Языки описания данных и манипулирования данными в иерархических СУБД.**

*Иерархическая модель данных*является наиболее простой среди всех даталогических моделей. Исторически она появилась первой среди всех даталогических моделей: именно эту модель поддерживает первая из зарегистрированных промышленных СУБД IMS фирмы IBM.

Появление иерархической модели связано с тем, что в реальном мире очень многие связи соответствуют иерархии, когда один объект выступает как родительский, а с ним может быть связано множество подчиненных объектов. Иерархия проста и естественна в отображении взаимосвязи между классами объектов.

Основными информационными единицами в иерархической модели являются: *база данных (БД), сегмент и поле.*

**Поле данных**- это минимальная, неделимая единица данных, доступная пользователю с помощью СУБД.

Например, если в задачах требуется печатать в документах адрес клиента, но не требуется дополнительного анализа полного адреса, то есть города, улицы, дома, квартиры, то мы можем принять весь адрес за элемент данных, и он будет храниться полностью, а пользователь сможет получить его только как полную строку символов из БД. Если же в наших задачах существует анализ частей, составляющих адрес, например города, где расположен клиент, то нам необходимо выделить город как отдельный элемент данных, только в этом случае пользователь может получить к нему доступ и выполнить, например, запрос на поиск всех клиентов, которые проживают в конкретном городе, например в Париже. Однако если пользователю понадобится и полный адрес клиента, то остальную информацию по адресу также необходимо хранить в отдельном поле, которое может быть названо, например, Сокращенный адрес. В этом случае для каждого клиента в БД хранится как Город, так и Сокращенный адрес.

**Сегмент**- в терминологии Американской Ассоциации по базам данных DBTG (Data Base Task Group) называется *записью,*при этом в рамках иерархической модели определяются два понятия: *тип сегмента*или *тип записи*и *экземпляр сегмента*или *экземпляр записи.*

**Тип сегмента**- это поименованная совокупность типов элементов данных, в него входящих. ***Экземпляр сегмента***образуется из конкретных значений полей или элементов данных, в него входящих. Каждый тип сегмента в рамках иерархической модели образует некоторый набор однородных записей. Для возможности различия отдельных записей в данном наборе каждый тип сегмента должен иметь ключ или набор ключевых атрибутов (полей, элементов данных).

**Ключ**- это набор элементов данных, однозначно идентифицирующих экземпляр сегмента.

Например, рассматривая тип сегмента, описывающий сотрудника организации, мы должны выделить те характеристики сотрудника, которые могут его однозначно идентифицировать в рамках БД предприятия. Если предположить, что на предприятии могут работать однофамильцы, то, вероятно, наиболее надежным будет идентифицировать сотрудника по его табельному номеру. Однако если мы будем строить БД, содержащую описание множества граждан, например нашей страны, то, скорее всего, нам придется в качестве ключа выбрать совокупность полей, отражающих его паспортные данные.

В иерархической модели сегменты объединяются в ориентированный древовидный граф. При этом полагают, что направленные ребра графа отражают иерархические связи между сегментами: каждому экземпляру сегмента, стоящему выше по иерархии и соединенному с данным типом сегмента, соответствует несколько (множество) экземпляров данного (подчиненного) типа сегмента. Тип сегмента, находящийся на более высоком уровне иерархии, называется логически исходным по отношению к типам сегментов, соединенным с данным направленными иерархическими ребрами, которые в свою очередь называются логически подчиненными по отношению к этому типу сегмента. Иногда исходные сегменты называют сегментами-предками, а подчиненные сегменты называют сегментами-потомками.

**Схема иерархической БД**- это совокупность отдельных деревьев, каждое дерево в рамках модели называется физической базой данных.

Каждая физическая БД удовлетворяет следующим иерархическим ограничениям:

* в каждой физической БД существует один корневой сегмент, то есть сегмент, у которого нет логически исходного (родительского) типа сегмента;
* каждый логически исходный сегмент может быть связан с произвольным числом логически подчиненных сегментов;
* каждый логически подчиненный сегмент может быть связан только с одним логически исходным (родительским ) сегментом.

Экземпляры-потомки одного типа, связанные с одним экземпляром сегмента-предка, называют «близнецами». Так, для нашего примера экземпляры b1, b2 и bЗ являются «близнецами» , но экземпляр b4 подчинен другому экземпляру родительского сегмента, и он не является «близнецом» по отношению к экземплярам b1, b2 и bЗ. Набор всех экземпляров сегментов, подчиненных одному экземпляру корневого сегмента, называется *физической записью.*Количество экземпляров-потомков может быть разным для разных экземпляров родительских сегментов, поэтому в общем случае физические записи имеют разную длину.

**Язык описания данных иерархической модели**

В рамках иерархической модели выделяют языковые средства описания данных (DDL, Data Definition Language) и средства манипулирования данными (DML, Data Manipulation Language).

Каждая физическая база описывается набором операторов, определяющих как ее логическую структуру, так и структуру хранения БД. Описание начинается с*оператора определения базы* - DBD (Data Base Definition):

DBD Name = < имя БД>, ACCESS = < способ доступа>

Способ доступа определяет способ организации взаимосвязи физических записей.

Определено 5 способов доступа:

**HSAM**- *hierarchical sequential access method*(иерархически последовательный метод),

**HISAM**- *hierarchical index sequential access method*(иерархически индексно-последовательный метод),

**EDAM**- *hierarchical direct access method*(иерархически прямой метод),

**HID AM**- *hierarchical index direct access method*(иерархически индексно-прямой метод),

**INDEX**- индексный метод.

Далее идет описание наборов данных, предназначенных для хранения БД:

DATA SET D01 = < имя оператора, определяющего хранимый набор данных>.

DEVICE =< устройство хранения БД>, [OVFLW = < имя области переполнения>]

Так как физические записи имеют разную длину, то при модификации данных запись может увеличиться и превысит исходную длину записи до модификации. В этом случае при определенных методах хранения может понадобиться дополнительное пространство хранения, где и будут размещены дополнительные данные. Это пространство и называется областью переполнения.

После описания всей физической БД идет описание типов сегментов, ее составляющих, в соответстшш с иерархией. Описание сегментов всегда начинается с описания корневого сегмента. Общая схема описания типа сегмента такова:

SEGM NAME = < имя сегмента>. BYTES =< размер в байтах>.

FREQ = <средняя частота реализаций сегмента под одним исходным>

PARENT = <имя родительского сегмента>

Параметр FREQ определяет среднее количество экземпляров данного сегмента, связанных с одним экземпляром родительского сегмента. Для корневого сегмента это число возможных экземпляров корневого сегмента.

Для корневого сегмента параметр PARENT равен 0 (нулю). Далее для каждого сегмента дается описание полей:

FIELD NAME = {(<имя поля> [. SEQ].{U M}) | <имя поля> }.

START = < номер байта, с которого начинается значения поля >,

BYTES = <размер поля в байтах>,

TYPE = {X | Р | С}

Признак SEQ — задается для ключевого поля, если экземпляры данного сегмента физически упорядочены в соответствии со значениями данного поля.

Параметр U задается, если значения ключевого поля уникальны для всех экземпляров данного сегмента, М — в противном случае. Если поле является ключевым, то его описание задается в круглых скобках, в противном случае имя поля задается без скобок. Параметр TYPE определяет тип данных. Для ранних иерархических моделей были определены только три типа данных: X — шестпадцатеричиый, Р —упакованный десятичный, С — символьный.

Заканчивается описание схемы вызовом процедуры генерации:

* DBDGEN — указывает на конец последовательности управляющих операторов описания БД;
* FINISH — устанавливает ненулевой код завершения при обнаружении ошибки;
* END — конец.

В системе может быть несколько физических БД (ФБД), но каждая из них описывается отдельно своим DBD и ей присваивается уникальное имя. Каждая ФБД содержит только один корневой сегмент. Совокупность ФБД образует концептуальную модель данных.

Для доступа к базе данных у пользователя должна быть сформирована специальная среда окружения, поддерживающая в явном виде имеющиеся навигационные операции. Для этого в ней должны храниться:

* шаблоны всех записей логических баз данных, доступных пользователю;
* указатели на текущий экземпляр сегмента данного типа — для всех типов сегментов.

Язык манипулирования данными в иерархической модели поддерживает в явном виде навигационные операции. Эти операции связаны с перемещением указателя, который определяет текущий экземпляр конкретного сегмента.

Все операторы в языке манипулирования данными можно разделить на 3 группы. Первую группу составляют операторы поиска данных.

**Операторы поиска данных**

**Синтаксис:**

GET UNIQUE <имя сегмента> WHERE <список поиска>;

список поиска состоит из последовательности условий вида:

<имя сегмента>.<имя поля>ОС <constant или имя другого поля данного сегмента или имя переменной>:

ОС — операция сравнения;

условия могут быть соединены логическими операциями И и ИЛИ {& , V}.

**Назначение:**

Получить единственное значение.

**Пример:**

Найти типовую модель стоимостью не более $600, которая существует не менее чем в 10 экземплярах.

GET UNIQUE ТИПОВЫЕ МОДЕЛИ

WHERE Типовые модели.Стоимость <= $600

AND Типовые модели,Количество на складе >= 10

Данная команда всегда ищет с начала БД и останавливается, найдя первый экземпляр сегмента, удовлетворяющий условиям поиска.

**Синтаксис:**

GET NEXT <имя сегмента> WHERE <список аргументов поиска>

**Назначение:**

Получить следующий экземпляр сегмента для тех же условии.

**Пример:**

Напечатать полный список заказов стоимостью не менее $500.

GET UNIQUE ИНДИВИДУАЛЬНЫЕ МОДЕЛИ

WHERE Индивидуальные модели.Стоимость >- $500

WHILE NOT EAIL (пока не конец поиска) DO

PRINT № заказа. Стоимость, Количество

GET NEXT ИНДИВИДУАЛЬНЫЕ МОДЕЛИ

END

**Синтаксис:**

GET NEXT <имя сегмента> WITHIN PARENT [ where <дополн.условия>]

**Назначение:**

Получить следующий для того же исходного.

**Пример:**

Получить перечень винчестеров, имеющихся на складе номер 1, в количестве не менее 10 с объемом 10 Гбайт.

GET UNIQUE СКЛАД WHERE Склад.Номер = 1

GET NEXT ИЗДЕЛИЕ WITHIN PARENT

WHERE Изделие.Наименование = "Винчестер"

GET NEXT ХАРАКТЕРИСТИКИ WITHIN PARENT

WHERE ХАРАКТЕРИСТИКИ.Параметр = 10 AND

ХАРАКТЕРИСТИКИ.Единицы Измерения = Гб AND

ХАРАКТЕРИСТИКИ.Величина > 10

While Not Fail (пока поиск не завершен) DO

Get Next Within Parent

end

**Операторы поиска данных с возможностью модификации**

1. Найти и удержать единственный экземпляр сегмента. Эта операция подобна первой операции поиска GET UNIQUE, единственным отличием этой операции является то, что после выполнения этой операции пал найденным экземпляром сегмента допустимы операции модификации (изменения) данных.

**Синтаксис:**

GET HOLD UNIQUE <имя сегмента> WHERE <список поиска>

1. Найти и удержать следующий с теми же условиями поиска. Аналогично операции 4 эта операция дублирует вторую операции поиска GET NEXT с возможностью выполнения последующей модификации данных.

**Синтаксис:**

GET HOLD NEXT [WHERE <дополнительные условия>]

1. Получить и удержать следующий для того же родителя. Эта операция является аналогом операции поиска 3, но разрешает выполнение операций модификации данных после себя.

**Синтаксис:**

GET HOLD NEXT WITHIN PARENT [ where <дополн.условия>]

**Операторы модификации данных**

1. Удалить : Это первая из трех операций модификации.

**Синтаксис:**

DELETE

Эта команда не имеет параметров. Почему? Потому что операции модификации действуют на экземпляр сегмента, найденный командами поиска с удержанием. А он всегда единственный текущий найденный и удерживаемый для модификации экземпляр конкретного сегмента. Поэтому при выполнении команды удаления будет удален именно этот экземпляр сегмента.

1. Обновить

**Синтаксис:**

UPDATE

Как же происходит обновление, если мы и в этой команде не задаем никаких параметров. СУБД берет данные из рабочей области пользователя, где в шаблонах записей соответствующих внутренних переменных находятся значения полей каждого сегмента внешней модели, с которой работает данный пользователь. Именно этими значениями и обновляется текущий экземпляр сегмента. Значит, перед тем как выполнить операции модификации UPDATE, необходимо присвоить соответствующим переменным новые значения.

Ввести новый экземпляр сегмента.

INSERT <имя сегмента>

Эта команда позволяет ввести новый экземпляр сегмента, имя которого определено в параметре команды. Если мы вводим данные в сегмент, который является подчиненным некоторому родительскому экземпляру сегмента, то он будет внесен в БД и физически подключен к тому экземпляру родительского сегмента, который в данный момент является текущим.

1. **Типы данных в СУБД My SQL. Привести примеры.**

# Типы данных в БД MySQL

[Базы данных MySQL](http://kbss.ru/blog/bd_mysql/)

В этой публикации я рассмотрю подробно, на сколько это возможно, основные типы данных используемые в СУБД MySQL.  
  
MySQL поддерживает несколько различных типов данных:  
**Числовые данные** — это все целые числа (без дробной части) и вещественные числа (с дробной частью).  
**Строковые данные** — последовательность символов, заключенная в кавычки. В MySQL в качестве стандарта используются одинарные кавычки и для совместимости с другими базами данных рекомендуется именно их использование.  
**Календарные данные** — тип для обозначение даты и времени, принимает различные формы, например строковую «2007-11-23» или числовую 20071123. Особенностью этого типа данных является их хранение в едином внутреннем формате, что позволяет производить операции сложения и вычитания, в независимости от внешнего представления.  
**NULL** — специальный тип данных, обозначает отсутствие информации.  
  
Итак, теперь о каждом типе данных подробнее.  
  
**Числовые данные**  
  
Числовые данные делятся на точечные (это BOOLEAN, INTEGER, DECIMAL) и приближенные (FLOAT, REAL, DOUBLE PRESISION).

В {} - таких скобках заключены необязательные элементы синтаксиса.  
  
TINYINT{(N)} - 1 байт  
// от -128 до 127 (от -2^7 до 2^7-1)  
// от 0 до 255 (от 0 до 2^8-1)  
  
SMALLINT{(N)} - 2 байта  
// от -32768 до 32767 (от -2^15 до 2^15-1)  
// от 0 до 65535 (от 0 до 2^16-1)  
  
MEDIUMINT{(N)} - 3 байта  
// от -8388608 до 8388608 (от -2^23 до 2^23-1)  
// от 0 до 16777215 (от 0 до 2^24-1)  
  
INT{(N)}, INTEGER{(N)} - 4 байта  
// от -2147683648 до 2147683648 (от -2^31 до 2^31-1)  
// от 0 до 4294967295 (от 0 до 2^32-1)  
  
BIGINT{(N)} - 8 байта  
// от -2^63 до 2^63-1  
// от 0 до 2^64  
  
BIT{(N)} - (N\_7)/8 байт  
// от 1 до 64 бит, зависит от значения N  
  
BOOL, BOOLEAN - 1 байт  
// либо 0, либо 1  
  
DECIMAL{(N{,D})}, DEC{(N{,D})}, NUMERIC{(N{,D})} - N+2 байта  
// Повышенная точность, зависит от значений N и D.

Итого, имеем: пять целых типов TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT. Различающихся между собой диапазоном хранимых в них величин. Чем больше диапазон значений, тем больше памяти для его хранения потребуется.  
  
Целые типы данных могут быть объявлены положительными. Для этого после объявления необходимо использовать ключевое слово UNSIGNED. В таком случае элементам столбца нельзя будет присвоить отрицательные значения, а допустимый диапазон положительных чисел которые может принимать элемент, удваивается. Например: TINYINT принимает от -128 до 127, а TINYINT UNSIGNED — от 0 до 255.  
  
При объявлении значения целого типа можно задать количество отводимых под число символов N (от 1 до 255). Делать это не обязательно, данное указание используется для дополнения пробелами слева от выводимых значений символов, меньших чем заданная ширина (при этом ограничений на диапазон значений это ни каких не налагает). Если указать дополнительно необязательный атрибут ZEROFILL, то вместо пробелов свободные позиции будут заполнены нулями слева. Например INT(6) ZEROFILL, значение 34 отобразится как 000034.  
  
Тип BIT{(N)} предназначен для хранения битовых полей. Параметр N обозначает число битовых значений, которое может принимать поле (от 1 до 64). Если параметр N не указан, то он принимает значение по умолчанию равное 1. (Тип BIT появился в MySQL с версии 5.0.3)  
  
Тип BOOLEAN это аналог TINYINT(1). Значение 1 это истина (true), значение 0 ложь (false).  
  
Тип DECIMAL, и его синонимы NUMERIC и DEC представляют величины повышенной точности, к примеру для денежных данных. Точность задается при объявлении столбца данных одного из этих типов, к примеру:

summarub DECIMAL(5,2)

В примере цифра 5 это общее число символов, отведенных под число, а цифра 2 это количество знаков после запятой. В данном случае получается что интервал величин которые могут храниться в столбце summarub, составляет от -99,99 до 99,99 (однако на самом деле для данного столбца можно хранить значения до 999,99, потому что допускается не хранить знак для положительных чисел).  
  
**Важно:** *Первый параметр N может принимать максимальное значение, равное 64, второй равное 30.*  
  
Величины DECIMAL, DEC и NUMERIC хранятся как строки, а не как двоичное число с плавающей точкой, для сохранения точности представления этих величин в десятичном виде. Если значение второго параметра равно 0, то величины DECIMAL и NUMERIC не содержат десятичного знака или дробной части.  
  
Для представления вещественных типов данных MySQL содержит три типа: FLOAT, DOUBLE и DECIMAL.

В {} - таких скобках заключены необязательные элементы синтаксиса.  
  
FLOAT{(N,D)} - 4 байта  
// минимальное значение ± 1.175494351\*10^(-39)  
// максимальное значение ± 3.402823466\*10^38  
  
DOUBLE{(N,D)}, REAL{(N,D)}, DOUBLE PRECISION{(N,D)} - 8 байт  
// минимальное значение ± 2.2250738585072014\*10^(-308)  
// максимальное значение ± 1.797693134862315\*10^308

Параметр N — задает количество символов для всего числа, D — количество символов дробной части.  
  
Числовые типы данных с плавающей запятой, тоже могут иметь параметр UNSIGNED. Как и в целочисленных данный параметр предотвращает хранение в указанном столбце отрицательных величин, но в отличие от целочисленных типов, максимальный размер столбца при этом не увеличивается.  
  
*Вещественные числа могут быть заданы как в обычной форме, к примеру 47.85, и в научной нотации, например 4.785E+04.*  
  
При выборе столбцов для создания структуры таблицы, обязательно необходимо обращать внимание на размер, который занимает тот или иной тип данных: в случае если значения, размещаемые в базе данных, никогда не выйдут за 100, не надо выбирать тип более TINYINT. При хранении в поле только положительных значений, применение параметра UNSIGNED позволит увеличить диапазон значений в два раза.  
  
Пример создания таблицы:

create table table1 (  
number1 int default 0,  
number2 float default 1.0  
)

Из примера видно, что после типа столбца задано ключевое слово DEFAULT. Это поле позволяет задать значение поля по умолчанию, так для поля number1 это значение равно 0, а для number2 1.0. Ключевое слово DEFAULT является не обязательным и может не указываться, то-есть мы можем создать таблицу и так:

create table table1 (  
number1 int,  
number2 float  
)

**Строковые данные**  
  
Максимальные размеры и требования к памяти, строковых типов данных:

Здесь L - это длина хранимой в ячейке строки, байты приплюсованные к этому значению,  
это накладные расходы.  
  
CHAR(N)  
// Объем памяти - N символов  
// Максимальный размер - N символов  
  
VARCHAR(N)  
// Объем памяти - L+1 символов  
// Максимальный размер - N символов  
  
TINYBLOB, TINYTEXT  
// Объем памяти - L+1 символов  
// Максимальный размер - 2^8-1 символов  
  
BLOB, TEXT  
// Объем памяти - L+2 символов  
// Максимальный размер - 2^16-1 символов  
  
MEDIUMBLOB, MEDIUMTEXT  
// Объем памяти - L+3 символов  
// Максимальный размер - 2^24-1 символов  
  
LONGBLOB, LONGTEXT  
// Объем памяти - L+4 символов  
// Максимальный размер - 2^32-1 символов  
  
ENUM('value1','value2',...)  
// Объем памяти - 1 или 2 байта  
// Максимальный размер - 65535 элементов  
  
SET('value1','value2',...)  
// Объем памяти - 1, 2, 3, 4 или 8 байт  
// Максимальный размер - 64 элемента

Тип данных CHAR может хранить строку фиксированной длины N, его дополняет тип VARCHAR, который позволяет хранить переменные строки длиной L. Значение N может принимать значения от 0 до 65535. (До версии MySQL 5.0.3 значение N могло быть только от 0 до 255)  
  
При выборе строкового типа данных для столбца, следует учитывать что для переменных строк VARCHAR требуется количество символов, равное длине строки плюс один байт, а для типа CHAR(N) в независимости от длины строки будут сохранены все N символов.  
  
При создании таблицы нельзя комбинировать столбцы типов CHAR и VARCHAR. Если такое произойдет, то MySQL изменит тип столбцов CHAR на тип VARCHAR.  
  
*С версии MySQL 4.1.2, в типах CHAR и VARCHAR строки рассматриваются как последовательности символов. Таким образом, при использовании многобайтных кодировок, например UNICODE, размер строки в байтах будет больше, чем в символах. Для совместимости со старыми версиями MySQL введены два специальных типа данных: BINARY и VARBINARY, которые эквивалентны типам CHAR и VARHAR, однако строка в них рассматривается как последовательность байтов, а не символов. К BINARY строкам не применимы кодировки и сортируются они как обычные последовательности байтов.*  
  
Типы данных BLOB и TEXT в MySQL отличаются лишь в небольших деталях. Например, при выполнении действий над столбцами типа TEXT учитывается кодировка, а типа BLOB — нет.  
  
Тип TEXT как правило используется для хранения больших объемов текстовой информации, а BLOB — для хранения больших двоичных объектов, например изображений или звуков.  
  
*Главным отличием типа TEXT от CHAR и VARCHAR является поддержка возможностей полнотекстового поиска.*  
  
Особыми типами данных являются ENUM и SET. Строки данных типов принимают значения из заранее заданного списка значений. Основное отличие между этими двумя типами в том, что значение типа ENUM должно содержать только одно значение из указанного множества, в то время как столбцы SET могут содержать любой или все элементы заранее заданного множества одновременно. К примеру, значения для столбца объявленного как ENUM('z','x'), могут принимать только два значения: либо 'z', либо 'x'.  
  
Для типа SET, как и для типа ENUM, при объявлении задается список возможных значений, только ячейка может принимать любое значение из списка, а пустая строка означает, что ни один из элементов списка не выбран. К примеру, значения для столбца SET('z','x') могут принимать значения ('z','x'), ('z'), ('x') и пустое множество ().  
  
Типы данных ENUM и SET строковые лишь отчасти, так как при объявлении они задаются списком строк, однако во внутреннем представлении базы данных элементы множества сохраняются в виде чисел. Элементы типа ENUM нумеруются последовательно, начиная с 1. В зависимости от числа элементов в списке под столбец может отводиться 1 байт (до 256 элементов в списке) или 2 байта (от 257 до 65536 элементов в списке).  
  
Элементы из множества SET обрабатываются как биты, размер типа при этом тоже определяется числом элементов в списке: 1 байт (от 1 до 8 элементов), 2 байта (от 9 до 16 элементов), 3 байта (от 17 до 24 элементов), 4 байта (от 25 до 32 элементов) и 8 байт (от 33 до 64 элементов).  
  
**Календарные данные**  
  
В MySQL есть 5 видов столбцов для хранения календарных типов данных: DATE, DATETIME, TIME, TIMESTAMP и YEAR. Тип DATE — для хранения даты, TIME — для хранения времени, TIMESTAMP — для представления даты и времени в виде числа секунд, которые прошли с полуночи 1 января 1970 года. Тип данных YEAR — позволяет хранить только год.

В {} - таких скобках заключены необязательные элементы синтаксиса.  
  
DATE - 3 байта  
// от '1000-01-01' до '9999-12-31'  
  
TIME - 3 байта  
// от '-828:59:59' до '828:59:59'  
  
DATETIME - 8 байт  
// от '1000-01-01 00:00:00' до '9999-12-31 00:00:00'  
  
TIMESTAMP{(N)} - 4 байта  
// от '1970-01-01 00:00:00' до '2038-12-31 59:59:59'  
  
YEAR{(N)} - 1 байт  
// от '1901' до '2155' для YEAR(4)  
// от '1970' до '2069' для YEAR(2)

Значения типов DATE и DATETIME, в качестве первого числа принимают год либо в формате «YYYY», например '2010-12-10', либо в формате «YY», например '10-12-10'. Затем через дефис указывается месяц в формате «MM» (12), а следом день аналогично через дефис день в формате «DD» (10).  
  
В типах TIME и DATETIME время приводится в формате hh:mm:ss, где hh — часы, mm — минуты, ss — секунды. Дни, месяцы, часы, минуты и секунды можно записывать как с ведущим нулем: 01, так и без него: 1. К примеру все следующие записи идентичны:

2010-02-09 03:05:04  
2010-02-09 03:05:4  
2010-02-09 03:5:4  
2010-02-09 3:5:4  
2010-02-9 3:5:4  
2010-2-9 3:5:4

В качестве разделителя между годами, месяцами, днями, часами, минутами и секундами может выступать любой символ, отличный от цифры. К примеру, следующие значения идентичны:

10-02-09 03:05:04  
10.02.09 03+05+04  
10/02/09 03\*05\*04  
10@02@09 03^05^04

Дата и время суток могут быть представлены и в форматах «YYYYMMDDhhmmss» и «YYMMDDhhmmss». Например, строки '20100209030504' и '100209030504' аналогичны '2010-02-09 03:05:04'. Вместо строк допустимы и целочисленные значения, например, 20100209030504 и 100209030504 интерпретируются как '2010-02-09 03:05:04'.  
  
*С версии MySQL 4.1.1, при указании времени, после секунд через точку можно указать микросекунды, использовать расширенный формат вида 'hh:mm:ss.ffffff', например '03:05:04.000002'. Помимо расширенного формата, можно использовать краткие форматы 'HH:MM' и 'HH' — вместо пропущенных величин будут подставлены нулевые значения.*  
  
Формат типа TIMESTAMP совпадает с DATETIME, но во внутреннем представлении дата хранится в виде секунд, прошедших с полуночи 1 января 1970 года.  
  
Представление нулевых значений:

DATE '000-00-00'  
TIME '00:00:00'  
DATETIME '0000-00-00 00:00:00'  
TIMESTAMP 00000000000000  
YEAR 0000

Интересной особенностью поля TIMESTAMP является возможность автоматического получения текущей даты и времени при создании новой записи и изменении уже существующей оператором UPDATE.  
  
*До версии MySQL 4.0 включительно, формат типа TIMESTAMP представлял собой число вида YYYYMMDDhhmmss. В более поздних версиях формат был изменен на YYYY-MM-DD hh:mm:ss.*  
  
При наличии в таблице нескольких столбцов TIMESTAMP, при модификации записи текущее время будет записываться только в один из столбцов (по умолчанию в первый). Можно также указать явно столбец, которому необходимо назначать текущую дату при создании новой записи, следует после определения столбца добавить запись DEFAULT CURRENT\_TIMESTAMP. Если требуется, чтобы текущее время выставлялось при модификации уже существующей записи, при использовании оператора UPDATE необходимо добавить конструкцию ON UPDATE CURRENT\_TIMESTAMP.  
  
Для столбцов TIMESTAMP можно указать только одно из ключевых слов: либо DEFAULT CURRENT\_TIMESTAMP, либо ON UPDATE CURRENT\_TIMESTAMP.   
  
Пример:

create table tabl1(  
mytime1 timestamp default current\_timestamp,  
mytime2 timestamp on update current\_timestamp  
);

Здесь поле mytime1 — получает текущее время при создании записи, а поле mutime2 — получает текущее время при изменении записи.  
  
Столбцы типа данных YEAR предназначены для хранения только года, указание параметра N позволяет задать формат года: двузначный CHAR(2) «YY» или четырехзначный CHAR(4) «YYYY». Если параметр N не указан то по умолчанию считается что он равен 4.  
  
**Тип данных NULL**  
  
При создании таблицы, возможны ситуации когда данных для заполнения всех полей недостаточно, и для части данных нельзя определить, какое значение они примут. Такие данные обозначают специальным типом — NULL.  
  
Для указания того, что поле может принимать значение NULL, в определении столбца, после типа данных требуется указать ключевое слово NULL, если поле ни при каких обстоятельствах не должно принимать значение NULL, необходимо указать ключевое слово NOT NULL.  
  
*Атрибут NOT NULL можно не указывать, поскольку он присваивается столбцу по умолчанию, если никакой из атрибутов не указан. Совместно с атрибутами NOT NULL и NULL можно использовать DEFAULT, который имеет больший приоритет.*

1. **CASE – средства моделирования и конфигурационного управления**

CASE-средств, реализующих CASE-технологию создания и сопровождения ИС. Термин CASE (Computer Aided Software Engineering) используется в настоящее время в весьма широком смысле. Первоначальное значение термина CASE, ограниченное вопросами автоматизации разработки только лишь программного обеспечения (ПО), в настоящее время приобрело новый смысл, охватывающий процесс разработки сложных ИС в целом. Теперь под термином CASE-средства понимаются программные средства, поддерживающие процессы создания и сопровождения ИС, включая анализ и формулировку требований, проектирование прикладного ПО (приложений) и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы. CASE-средства вместе с системным ПО и техническими средствами образуют полную среду разработки ИС.

Появлению CASE-технологии и CASE-средств предшествовали исследования в области методологии программирования. Программирование обрело черты системного подхода с разработкой и внедрением языков высокого уровня, методов структурного и модульного программирования, языков проектирования и средств их поддержки, формальных и неформальных языков описаний системных требований и спецификаций и т.д. Кроме того, появлению CASE-технологии способствовали и такие факторы, как:

* подготовка аналитиков и программистов, восприимчивых к концепциям модульного и структурного программирования;
* широкое внедрение и постоянный рост производительности компьютеров, позволившие использовать эффективные графические средства и автоматизировать большинство этапов проектирования;
* внедрение сетевой технологии, предоставившей возможность объединения усилий отдельных исполнителей в единый процесс проектирования путем использования разделяемой базы данных, содержащей необходимую информацию о проекте.

CASE-технология представляет собой методологию проектирования ИС, а также набор инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать эту модель на всех этапах разработки и сопровождения ИС и разрабатывать приложения в соответствии с информационными потребностями пользователей. Большинство существующих CASE-средств основано на методологиях структурного (в основном) или объектно-ориентированного анализа и проектирования, использующих спецификации в виде диаграмм или текстов для описания внешних требований, связей между моделями системы, динамики поведения системы и архитектуры программных средств.

Согласно обзору передовых технологий (Survey of Advanced Technology), составленному фирмой Systems Development Inc. в 1996 г. по результатам анкетирования более 1000 американских фирм, CASE-технология в настоящее время попала в разряд наиболее стабильных информационных технологий (ее использовала половина всех опрошенных пользователей более чем в трети своих проектов, из них 85% завершились успешно). Однако, несмотря на все потенциальные возможности CASE-средств, существует множество примеров их неудачного внедрения, в результате которых CASE-средства становятся "полочным" ПО (shelfware). В связи с этим необходимо отметить следующее:

* CASE-средства не обязательно дают немедленный эффект; он может быть получен только спустя какое-то время;
* реальные затраты на внедрение CASE-средств обычно намного превышают затраты на их приобретение;
* CASE-средства обеспечивают возможности для получения существенной выгоды только после успешного завершения процесса их внедрения.

Ввиду разнообразной природы CASE-средств было бы ошибочно делать какие-либо безоговорочные утверждения относительно реального удовлетворения тех или иных ожиданий от их внедрения. Можно перечислить следующие факторы, усложняющие определение возможного эффекта от использования CASE-средств:

* широкое разнообразие качества и возможностей CASE-средств;
* относительно небольшое время использования CASE-средств в различных организациях и недостаток опыта их применения;
* широкое разнообразие в практике внедрения различных организаций;
* отсутствие детальных метрик и данных для уже выполненных и текущих проектов;
* широкий диапазон предметных областей проектов;
* различная степень интеграции CASE-средств в различных проектах.

Вследствие этих сложностей доступная информация о реальных внедрениях крайне ограничена и противоречива. Она зависит от типа средств, характеристик проектов, уровня сопровождения и опыта пользователей. Некоторые аналитики полагают, что реальная выгода от использования некоторых типов CASE-средств может быть получена только после одно- или двухлетнего опыта. Другие полагают, что воздействие может реально проявиться в фазе эксплуатации жизненного цикла ИС, когда технологические улучшения могут привести к снижению эксплуатационных затрат.

Для успешного внедрения CASE-средств организация должна обладать следующими качествами:

* *Технология.*

Понимание ограниченности существующих возможностей и способность принять новую технологию;

* *Культура.*

Готовность к внедрению новых процессов и взаимоотношений между разработчиками и пользователями;

* *Управление.*

Четкое руководство и организованность по отношению к наиболее важным этапам и процессам внедрения.

Если организация не обладает хотя бы одним из перечисленных качеств, то внедрение CASE-средств может закончиться неудачей независимо от степени тщательности следования различным рекомендациям по внедрению.

Для того, чтобы принять взвешенное решение относительно инвестиций в CASE-технологию, пользователи вынуждены производить оценку отдельных CASE-средств, опираясь на неполные и противоречивые данные. Эта проблема зачастую усугубляется недостаточным знанием всех возможных "подводных камней" использования CASE-средств. Среди наиболее важных проблем выделяются следующие:

* достоверная оценка отдачи от инвестиций в CASE-средства затруднительна ввиду отсутствия приемлемых метрик и данных по проектам и процессам разработки ПО;
* внедрение CASE-средств может представлять собой достаточно длительный процесс и может не принести немедленной отдачи. Возможно даже краткосрочное снижение продуктивности в результате усилий, затрачиваемых на внедрение. Вследствие этого руководство организации-пользователя может утратить интерес к CASE-средствам и прекратить поддержку их внедрения;
* отсутствие полного соответствия между теми процессами и методами, которые поддерживаются CASE-средствами, и теми, которые используются в данной организации, может привести к дополнительным трудностям;
* CASE-средства зачастую трудно использовать в комплексе с другими подобными средствами. Это объясняется как различными парадигмами, поддерживаемыми различными средствами, так и проблемами передачи данных и управления от одного средства к другому;
* некоторые CASE-средства требуют слишком много усилий для того, чтобы оправдать их использование в небольшом проекте, при этом, тем не менее, можно извлечь выгоду из той дисциплины, к которой обязывает их применение;
* негативное отношение персонала к внедрению новой CASE-технологии может быть главной причиной провала проекта.

Пользователи CASE-средств должны быть готовы к необходимости долгосрочных затрат на эксплуатацию, частому появлению новых версий и возможному быстрому моральному старению средств, а также постоянным затратам на обучение и повышение квалификации персонала.

Несмотря на все высказанные предостережения и некоторый пессимизм, грамотный и разумный подход к использованию CASE-средств может преодолеть все перечисленные трудности. Успешное внедрение CASE-средств должно обеспечить такие выгоды как:

* высокий уровень технологической поддержки процессов разработки и сопровождения ПО;
* положительное воздействие на некоторые или все из перечисленных факторов: производительность, качество продукции, соблюдение стандартов, документирование;
* приемлемый уровень отдачи от инвестиций в CASE-средства.

1. **Проблемы и направления оптимизации БД.**

**Оптимизация запросов** — это 1) функция [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94), осуществляющая поиск оптимального [плана выполнения запросов](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%BD_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%B0) из всех возможных для заданного запроса, 2) процесс изменения запроса и/или структуры БД с целью уменьшения использования вычислительных ресурсов при выполнении запроса. Один и тот же результат может быть получен [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94) различными способами ([планами выполнения запросов](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%BD_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%B0)), которые могут существенно отличаться как по затратам ресурсов, так и по времени выполнения. Задача оптимизации заключается в нахождении оптимального способа.

В реляционной СУБД оптимальный план выполнения запроса — это такая последовательность применения операторов реляционной алгебры к исходным и промежуточным отношениям, которая для конкретного текущего состояния БД (её структуры и наполнения) может быть выполнена с минимальным использованием вычислительных ресурсов.

В настоящее время известны две стратегии поиска оптимального плана:

* грубой силы путём оценки всех перестановок соединяемых таблиц, используемых способов входа в таблицы и типов соединения (т. е. полный перебор вариантов);
* на основе генетического алгоритма путём оценки ограниченного числа перестановок.

Также некоторые СУБД позволяют программисту вмешиваться в поиск оптимального плана в различной степени, от минимального влияния до полного и чёткого указания какой именно план запроса использовать.

Планы выполнения запроса сравниваются исходя из множества факторов (реализации в различных СУБД отличаются), в том числе:

* потенциальное число строк, извлекаемое из каждой таблицы, получаемое из статистики;
* наличие [индексов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85));
* возможность выполнения [слияний](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%81%D0%BE%D0%B5%D0%B4%D0%B8%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F_%D1%81%D0%BB%D0%B8%D1%8F%D0%BD%D0%B8%D0%B5%D0%BC_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_%D1%81%D0%BF%D0%B8%D1%81%D0%BA%D0%BE%D0%B2) (merge-join);
* способ чтения записей/блоков таблиц/индексов.

В общем случае соединение выполняется [вложенными циклами](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%81%D0%BE%D0%B5%D0%B4%D0%B8%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F_%D1%81%D0%BB%D0%B8%D1%8F%D0%BD%D0%B8%D0%B5%D0%BC_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_%D1%81%D0%BF%D0%B8%D1%81%D0%BA%D0%BE%D0%B2). Однако этот алгоритм может оказаться менее эффективен, чем специализированные алгоритмы. Например, если у сливаемых таблиц есть индексы по соединяемым полям, или одна или обе таблицы достаточно малы, чтобы быть отсортированными в памяти, то исследуется возможность выполнения [слияний](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%81%D0%BE%D0%B5%D0%B4%D0%B8%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F_%D1%81%D0%BB%D0%B8%D1%8F%D0%BD%D0%B8%D0%B5%D0%BC_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_%D1%81%D0%BF%D0%B8%D1%81%D0%BA%D0%BE%D0%B2).

**55. Этапы развития БД:**

1. БД на больших эвм:

Все СУБД базируются на мощных мультипрограммных ОС, поэтому в основном поддерживается работа с централизованно БД в режиме распределенного доступа. функции управления распределением ресурсов в основном осуществляются ОС. поддерживаются языки низкого уровня манипулирования данными, ориентированные на навигационные методы доступа к данным. значительная роль отводится администрированию данных. Проводятся серьезные работы по обоснованию и формализации модели данных( System R). Проводятся теоретические работы по оптимизации запросов и управлению распределенным доступом к БД, было введено понятие транзакций.  появляются первые языки высокого уровня для работы с реляционной моделью данных, однако отсутствуют стандарты для этих языков.

2. Эпоха ПК:

Все субд были рассчитаны на создание бд с монопольным доступом. Большинство СУБД имели удобный и развитый пользовательский интерфейс. в большинстве существовал интерактивный режим работы с БД, как в рамках описания БД, так и в рамках проектирования запросов.  Во всех настольных СУБД поддерживался только внешний уровень представления реляционной модели( только внешний табличный вид структуры данных). при наличии высокоуровневых языков манипулирования данными типа реляционной алгебры, в настольных СУБД поддерживались низкоуровневые языки манипулирования данными, на уровне строк и таблиц. в настольных субд отсутствовали средства поддержки ссылочной и структурной целостности БД. Эти функции должны были выполнять приложения, однако, скудность средств разработки приложений никогда не позволяла это сделать, и в этом случае функции выполнялись пользователем. Наличие монопольного режима работы фактически привело к рождению функций администрирования БД. В связи с этим отсутствовали инструментальные средства администрирования БД. Сравнительно скромные требования к аппаратному обеспечению со стороны настольных СУБД. (D Base, Fox Pro, Paradox, Cleepper).

3. Распределенные БД:

Практически все современные СУБД обеспечивают поддержку полной реляционной модели(структурная целостность(допустимыми являются только данные, представленные в виде отношения реляционной модели), языковой целостности, ссылочной целостности( контроль за соблюдением ссылочной целостности, и гарантия невозможности со стороны СУБД нарушить ограничения). Большинство современных СУБД рассчитаны на многоплатформенную архитектуру. Необходимость поддержки многопользовательской работы с БД. возможность децентрализованного хранения данных  потребовалось развитие средств администрирования с реализацией общей концепции средств защиты данных. Для того, чтобы не потерять клиентов, которые ранее работали на настольных СУБД, практически все современные СУБД имеют средства подключения клиентских приложений, разработанных с использованием настольных СУБД и средств экспорта данных из форматов настольных СУБД 2 этапа развития. Разработан ряд стандартов, в рамках языков описания и манипулирования данными технологии по обмену данными между различными данными, к которым можно отнести протокол ODBC, предложенной фирмой Microsoft. К этому этапу можно отнести начало работ, связанных с концепцией объектно-ориентированных БД. Oracle7.3, 8.4., System 10-11, informix, DB2.

4. Перспективы развития СУБД.

Банк данных - система, специальным образом организованных данных - баз данных, программных, технических, языковых, организационно-методических, предназначенных для коллективного, многоцелевого использования данных.

 БД - именованная совокупность данных, отражающих состояние объектов, и их отношений в рассматриваемой предметной области.

СУБД-совокупность языковых программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.