

Проектирование баз данных

Физические и логические сущности.

Тип мира	Физические объекты	Логические объекты
Мир людей	Шкаф с папками, бумага, карандаши	Мысли, идеи
Мир БД	Файл данных	Таблица, представление, индекс

Что же представляет собой таблица БД ? Лучше сравнить ее со шкафом, у которого есть выдвижные ящики. Ящики могут быть одного или разных размеров, в каждый из них можно что-то положить, а можно оставить его пустым на какое-то время (или навсегда). Таблица - это место, куда помещаются порции информации разных размеров.

Терминология

База данных представляет собой совокупность информации, организованной в виде множеств. Каждое множество содержит записи унифицированного вида. Сами записи состоят из полей. Обычно множества называют таблицами, а записи – строками таблиц.

Это – логическая модель данных. На жестком диске вся БД может находиться в одном файле.

Строки таблиц могут быть связаны друг с другом одним из трех способов:

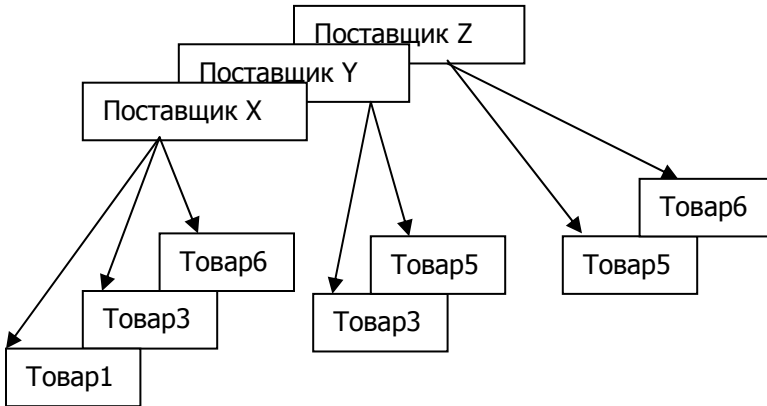
1:1 (один к одному)– строка одной таблице соответствует единственной строке второй таблицы.

1:N (один ко многим)– строка одной таблицы соответствует нескольким строкам другой таблицы.

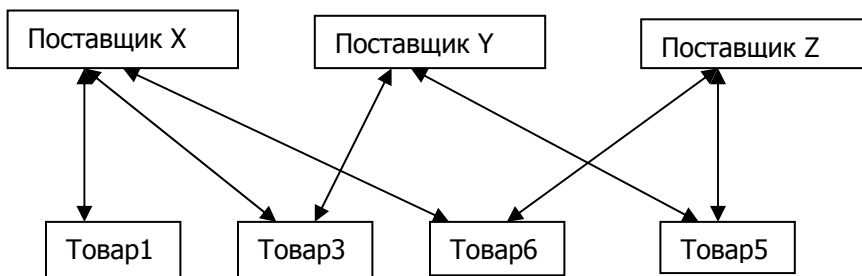
N:M (многие ко многим) – строки первой таблицы могут быть связаны с произвольным числом строк во второй таблице.

Основные концепции.

СУБД (система управления БД) управляет одной или несколькими базами данных. В первых СУБД была реализована иерархическая модель данных, в которой между одним объектом и подчиненными ему объектами устанавливаются иерархические связи. Например, между руководителем и его служащими или между поставщиками и поставляемыми ими товарами. В иерархических моделях всегда существует один путь от корня иерархии к каждой записи. Иерархические взаимосвязи непосредственно поддерживаются в физических конструкциях СУБД.



В сетевой модели иерархические взаимосвязи составляют частный случай сетевых. Например, в рыночном приложении каждый вид товара может иметь нескольких поставщиков, а каждый поставщик поставлять несколько видов товаров. Все эти взаимосвязи – иерархические, но общая взаимосвязь между поставщиками и товарами – сетевая.



Наконец, реляционная модель не вводит различия между объектами и взаимосвязями. Отношение может представлять объект, например товар, или взаимосвязь, например между товарами и поставщиками.

Поставщики	Товары	Поставщик-Товар	
X	1	X	1
Y	3	X	3
Z	5	X	6
	6	Y	3
		Y	5
		Z	5
		Z	6

Реляционную модель придумал научный сотрудник компании IBM доктор Эдгар Кодд в 1970 г. Данные теперь можно было свободно описывать в их естественном виде без каких-либо ограничений, накладываемых средой физического хранения. Это позволяло создать язык высокого уровня, способный работать с данными независимо от того, как они хранятся в

компьютере. Первоначально язык запросов назывался SEQUEL (Structured English Query Language). Позднее появилось название SQL (Structured Query Language).

СУБД или система управления базами данных.

Программист, работающий с БД, не заботится о том, как эти данные хранятся. В свою очередь, приложения, взаимодействующие с СУБД, не знают о способе записи данных на диск. Программист видит лишь логический образ БД. Подобная обработка данных осуществляется с помощью языка четвертого поколения, который поддерживает запросы, записываемые и исполняемые немедленно. При этом важна скорость доступа к данным. Кроме того, программист должен иметь возможность формировать новые нерегламентированные запросы к БД.

Язык четвертого поколения позволяет создавать схемы - точные определения данных и отношений между ними. Схема хранится как часть БД и предназначена для контроля целостности данных. Если объявлено, что в поле хранятся целочисленные значения, то СУБД не позволит записать в него строковые данные.

СУБД обеспечивает безопасность данных. Пользователям предоставлены определенные права доступа к информации. Одни могут лишь просматривать данные, другие – менять содержимое таблиц.

СУБД поддерживает параллельный доступ к БД. Приложения могут обращаться к БД одновременно, что повышает производительность системы.

СУБД помогает восстановить данные в случае непредвиденного сбоя, незаметно для пользователя создавая резервные копии данных.

Реляционная модель БД. Реляционная алгебра.

В реляционной модели база данных представляет собой централизованное хранилище таблиц, обеспечивающее безопасный одновременный доступ к информации со стороны многих пользователей. Реляционная модель пытается максимально избавить программиста от заботы о физической реализации БД. Доктор Кодд первоначально описал реляционную модель как область применения реляционной алгебры. Введем несколько понятий:

Алгебра – система определения множеств и операций над ними.

Множество – совокупность уникальных элементов, объединенных по какому-то признаку.

Оператор – это символическая запись правила преобразования, выполняемого над одним или несколькими элементами множества.

В традиционной алгебре элементами множества являются числа, над которыми можно выполнять операции сложения, вычитания, умножения, деления и др. **Реляционная алгебра** – это система манипулирования отношениями, которые являются элементами, группируемыми во множество.

Таблицы, строки, столбцы, ключи, отношения.

Реляционная модель скрывает детали физического хранения данных. Отношения, существующие между элементами данных, выявляются на логическом уровне. Реляционная БД состоит из таблиц. Аналогия между БД и ящиком картотеки: таблица в этом случае – папка, лежащая в ящике.

Таблица состоит из строк (записей) и столбцов (полей). Доктор Кодд называл таблицы отношениями, т.к. каждая строка таблицы неявно связана со всеми остальными строками, разделяя с ними общую форму записи.

Запись или кортеж - это набор взаимосвязанных атрибутов.

Атрибут – это характеристика объекта (цвет, вес, цена, размер и т.д.)

Каждый атрибут имеет название и тип. Пока скажем, что базовыми типами считаются

строковый, числовой и дата/время. Определение атрибута является строгим. Все значения атрибута должны иметь один и тот же тип. Строковым атрибутам дополнительно назначается максимальная длина.

Первичный ключ

<i>Имя атрибута</i>	<u>N служащего</u>	<u>Имя служащего</u>	<u>Дата Рождения</u>	<u>Отдел</u>	<u>Должность</u>
<i>Тип атрибута</i>	Number	Char	Date	Number	Char
<i>Значение атр.</i>	53730	Джонс Билл	1976-05-25	23	Бухгалтер
	15971	Смит Том	1971-01-08	93	Инженер
	72921	Холл Альберт	1975-07-14	93	Программист
Запись	53550	Лоранс Фред	1970-10-23	23	Бухгалтер

атрибуты сами представляют собой идентификаторы объекта в другой таблице

Ключ – это определенным образом помеченный столбец таблицы. Для того, чтобы его значения были связаны со столбцом другой таблицы, необходимо установить между столбцами виртуальную связь.

Отделы

N Отдела Название Отдела

23 Бухгалтерия
93 Технический отдел

Служащие

внешний ключ

N служащего Имя служащего Дата Рождения Отдел Должность

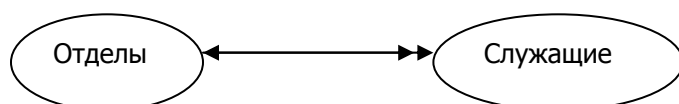
53730	Джонс Билл	1976-05-25	23	Бухгалтер
15971	Смит Том	1971-01-08	93	Инженер
72921	Холл Альберт	1975-07-14	93	Программист
53550	Лоранс Фред	1970-10-23	23	Бухгалтер

Столбец «N Отдела» в таблице «Отделы» - **первичный ключ** – он уникальным образом идентифицирует каждую строку. Столбец «Отдел» в таблице «Служащие» является **внешним ключом**, т.к. его значения берутся из внешней таблицы.

Когда столбец или группа столбцов помечается как ключ, дополнительно создается индекс. **Индекс** хранит значения ключа и указатели на записи, содержащие эти значения. Индексы ускоряют операции чтения, но замедляют выполнение операций записи.

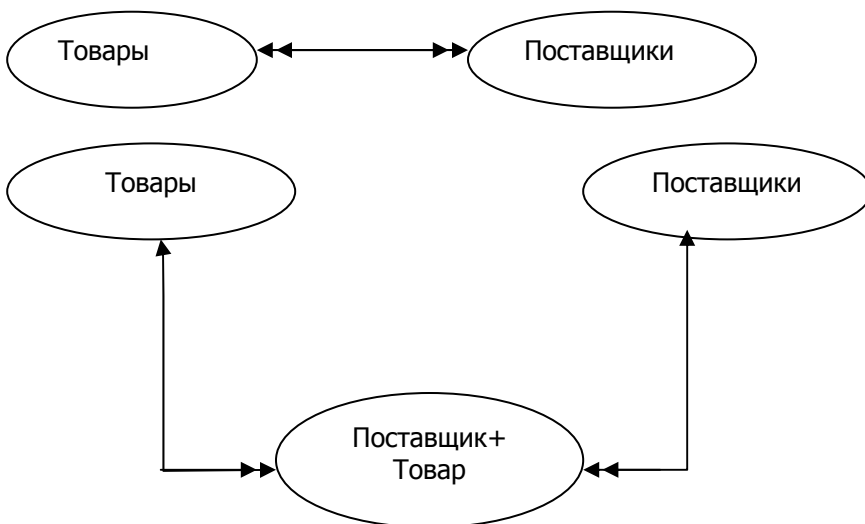
Между таблицами могут существовать отношения трех типов: «один ко многим», «один к одному» и «многие ко многим».

Отношение «**один ко многим**» (**1:N**) реализуется с помощью внешних ключей. При этом любой строке первой таблицы может соответствовать несколько строк второй таблицы. Строке же второй таблицы соответствует всего одна запись первой таблицы.



В идеально спроектированной БД отношение «**один к одному**» (**1:1**) не нужно. Если каждой строке одной таблицы соответствует одна строка другой таблицы, то такие таблицы обычно объединяют в одну.

Отношение «**многие ко многим**» (**M:N**) в реляционной БД напрямую создать нельзя. Оно преобразуется в два отношения 1:N с помощью промежуточной таблицы.



Реляционные операции

Для реляционной модели интересны только те операции, результатом которых являются множества. Отсюда имеем 8 простейших операций:

- выборка
- проекция
- пересечение
- сложение
- вычитание
- умножение
- деление
- переименование

Выборка – фильтрация строк. Результирующая таблица содержит все столбцы, но, возможно, не все строки. Напр., получим список сотрудников, родившихся до 1975 года:

<u>Ид служащего</u>	<u>Имя служащего</u>	<u>Дата Рождения</u>	<u>Отдел</u>	<u>Должность</u>
15971	Смит Том	1971-01-08	93	Инженер
53550	Лоранс Фред	1970-10-23	23	Бухгалтер

Проекция – фильтрация столбцов. Операция возвращает все строки, но, возможно не все столбцы. Получим список имен служащих:

Имя служащего

Джонс Билл
Смит Том
Холл Альберт
Лоранс Фред

Операция **пересечения** выполняется над двумя таблицами идентичной структуры. В результате получим только тезаписи, которые встречаются в обеих таблицах. Предположим, нужно узнать какие игроки принимали участие в финальном матче как в 2000, так и в 2001 году

<u>Финальный матч 2000 г.</u>	<u>Финальный матч 2000 г.</u>	<u>Пересечение</u>
Аломар	Голд	Родригес
Джонс	Лофтон	Селси
Родригес	Мансини	
Селси	Родригес	
Спото	Селси	
Фланел	Хилтон	
Шафт	Энсон	

Операция **сложения** выполняется над двумя таблицами идентичной структуры. В результирующую таблицу попадут все записи исходных таблиц, исключая дублируемые. Получим объединенный список участников финальных матчей в 2000 и 2001 г.

<u>Финальный матч 2000 г.</u>	<u>Финальный матч 2000 г.</u>	<u>Сложение</u>
Аломар	Голд	Аломар
Джонс	Лофтон	Голд
Родригес	Мансини	Джонс
Селси	Родригес	Лофтон
Спото	Селси	Мансини
Фланел	Хилтон	Родригес
Шафт	Энсон	Селси
		Спото
		Фланел
		Хилтон
		Шафт
		Энсон

Результирующая таблица содержит 12 строк, а не 14, т.к. два игрока принимали участие в обоих матчах.

Операция **вычитания** возвращает строки первой таблицы, отсутствующие во второй.

<u>Финальный матч 2000 г.</u>	<u>Финальный матч 2000 г.</u>	<u>Вычитание</u>
Аломар	Голд	Аломар
Джонс	Лофтон	Джонс
Родригес	Мансини	Спото
Селси	Родригес	Фланел
Спото	Селси	Шафт
Фланел	Хилтон	
Шафт	Энсон	

Операция **умножения** объединяет каждую строку первой таблицы с каждой строкой второй таблицы. Количество строк результирующей таблицы равно произведению строк исходных таблиц. Эта операция называется также декартовым произведением. Эту операцию называют также декартовым произведением.

Пусть есть 2 сотрудника, которые могут занять 3 вакантные должности. Изобразим схему табличного произведения сотрудников с должностями.

Служащие		Должности	
<u>N служащего</u>	<u>Имя служащего</u>	<u>N Должности</u>	<u>Должность</u>
53730	Джонс Билл	1	Бухгалтер
15971	Смит Том	2	Экономист
		3	Менеджер

Результат произведения

N служащего Имя служащего N Должности Должность

53730	Джонс Билл	1	Бухгалтер
53730	Джонс Билл	2	Экономист
53730	Джонс Билл	3	Менеджер
15971	Смит Том	1	Бухгалтер
15971	Смит Том	2	Экономист
15971	Смит Том	3	Менеджер

Операция **деления** выполняется над двумя таблицами. Первая состоит из двух столбцов, а вторая – из одного. Значения второй таблицы сравниваются со значениями первого столбца первой таблицы. При совпадении в результат запроса попадает значение второго столбца первой таблицы, т.е. происходит его фильтрация.

<u>Имя служащего</u>	<u>Должность</u>		<u>Имя служащего</u>	=	<u>Должность</u>
Джонс Билл	Бухгалтер	/	Холл Альберт	=	Программист
Смит Том	Инженер		Лоранс Фред		Бухгалтер
Холл Альберт	Программист				
Лоранс Фред	Бухгалтер				

Операция **переименования** назначает столбцу другое имя.

Это тривиальная операция. Но именно с ее помощью решается проблема с одинаковыми именами столбцов в результирующей таблице. Обычно двум одноименным столбцам назначаются разные имена, называемые **псевдонимами**.

Одним из классов более сложных операций являются объединения.

Объединение – это частный случай произведения, при котором к таблице произведения применяются дополнительные условия отбора.

Внутреннее или естественное объединение сравнивает связанные столбцы двух таблиц (столбцы, упомянутые в условии отбора). Строки, не удовлетворяющие условию сравнения, удаляются из результирующей таблицы.

Игроки**Команды**

<u>Игрок</u>	<u>N Команды</u>	<u>N Команды</u>	<u>Название</u>
Джонс	1	1	Колорадо
Литтл	1	2	Сан Франциско
Мейсон	1	3	Чикаго
Кларк	2		
Родригес	2		

Об'единение по столбцу «N Команды»

<u>Игрок</u>	<u>N Команды</u>	<u>Название</u>
Джонс	1	Колорадо
Литтл	1	Колорадо
Мейсон	1	Колорадо
Кларк	2	Сан Франциско
Родригес	2	Сан Франциско

Внешнее об'единение – расширение внутреннего. Существуют 3 типа внешних об'единений:

- **Левое внешнее об'единение** включает все строки левой (первой) таблицы
- **Правое внешнее об'единение** включает все строки правой (второй) таблицы
- **Полное внешнее об'единение** включает все строки обеих исходных таблиц

Игроки

<u>N Игрока</u>	<u>Игрок</u>
1	Джонс
2	Литтл
3	Мейсон
4	Кларк

Клички игроков

<u>N Игрока</u>	<u>Кличка</u>
1	Джо
2	Дуду
4	Мистер Твистер

Левое внешнее об'единение по столбцу «N игрока»

<u>N Игрока</u>	<u>Игрок</u>	<u>Кличка</u>
1	Джонс	Джо
2	Литтл	Дуду
3	Мейсон	[null]
4	Кларк	Мистер Твистер

SQL – Язык четвертого поколения.

SQL – это язык взаимодействия с базами данных, применяемый в большинстве реляционных СУБД. SQL напоминает человеческий язык, т.к. одна из его задач – быть понятным непрограммистам. Запросы читаются как обычные предложения. Словарь языка относительно невелик, а его команды являются словами английского языка. Таким образом, SQL несложно изучить. Основным стандартом SQL был принят в 1992 г. и называется SQL2 или SQL-92. Продолжается также работа над наиболее современным стандартом SQL3.

Определение данных.

Для определения структуры базы данных предназначена команда CREATE.

Базу данных можно создать с помощью инструкции CREATE DATABASE. Единственным ее аргументом является имя создаваемой БД(она создается пустой). Если БД с таким именем уже существует, будет выдано сообщение об ошибке.

Создание базы данных store: CREATE DATABASE store;

Инструкция DROP DATABASE удаляет базу данных со всеми таблицами. Это – необратимое действие.

Таблица создается с помощью инструкции CREATE TABLE. Ей нужно указать не только имя таблицы, но и ее полное определение, состоящее из определений отдельных полей. Общий формат инструкции CREATE TABLE таков:

```
CREATE TABLE имя_таблицы
```

```
    (определение столбца, ...)
```

Определение столбца включает в себя имя столбца и спецификацию его типа. Пусть создаваемая таблица item содержит 4 столбца: ID, NAME, PRICE, DESCRIPTION.

```
CREATE TABLE item (
```

```
    ID INT(6) NOT NULL AUTO_INCREMENT,
```

```
    NAME CHAR(32) NOT NULL,
```

```
    PRICE DECIMAL(4,2) NOT NULL,
```

```
    DESCRIPTION CHAR (255) DEFAULT 'No Description',
```

```
    PRIMARY KEY (ID),
```

```
    KEY (NAME)
```

```
);
```

Столбец ID содержит целые числа в диапазоне от 0 до 999999. Значения NULL в нем недопустимы, на что указывает спецификатор NOT NULL. Кроме того, столбец работает в режиме счетчика (флаг AUTO_INCREMENT). При вставке в таблицу новой строки значение ее столбца ID будет автоматически увеличено на основании значения в предыдущей строке. Нумерация начинается с 1. Это хороший способ присвоения записям уникальных идентификаторов.

Столбец NAME содержит строку из 32-х символов. У каждого элемента таблицы должно быть имя, поэтому столбец помечен спецификатором NOT NULL.

Столбец PRICE хранит значение стоимости в десятичном формате: 4 цифры до запятой и 2 – после.

Столбец DESCRIPTION - текстовое поле максимальной длины – 255 символов. Он может принимать значения NULL, но благодаря значению DEFAULT они будут преобразовываться в строку 'No Description'.

В конце инструкции определяются первичный и вторичный ключи. Кроме того, для столбцов ID и NAME будут созданы индексы, что ускорит поиск в них.

На имена баз данных, таблиц и столбцов в MySQL накладывается ряд ограничений:

- длина имен ограничена 64-мя символами
- все имена могут состоять из букв, чисел, знаков подчеркивания (_), и символов доллара (\$). Из соображений удобочитаемости желательно, чтобы имена начинались с буквы.

Вставка записей.

Инструкция INSERT добавляет строку в таблицу. Формат инструкции:

```
INSERT [INTO] имя_таблицы [(столбец,...)]
{VALUES (значение,...),(...),... | запрос |
  SET столбец = значение,...}
```

Список, приводимый после имени таблицы, может содержать имена столбцов в произвольном порядке. Столбцам, не указанным в списке, будут присвоены значения по умолчанию. Например, в поле-счетчик запишется следующее целое число.

После ключевого слова VALUES в скобках указывается набор значений, разделенных запятой. В MySQL одном предложении VALUES можно вводить значения сразу для нескольких записей.

```
INSERT INTO team (NAME) VALUES ('Real'), ('Milan'), ('Roma')
```

С помощью подчиненной инструкции SELECT можно отобразить группу записей для вставки в таблицу:

```
INSERT INTO team (NAME)
SELECT FROM oldteam
```

Обновление записей.

Для изменения полей существующих записей предназначена инструкция UPDATE. Формат инструкции:

```
UPDATE имя_таблицы
  SET столбец=значение [, столбец=значение,...]
[WHERE условие_отбора]
```

За один раз можно обновить только одну таблицу, но произвольное число столбцов. Имена столбцов и присваиваемые им значения приводятся в предложении SET. Предложение WHERE содержит условие отбора обновляемых записей.

```
UPDATE team SET NAME = 'Barselona' WHERE ID = 5
```

Удаление записей.

Инструкция DELETE удаляет строку из таблицы. Общий формат инструкции таков:

```
DELETE FROM имя_таблицы
```

```
[WHERE условия]
```

Предложение WHERE содержит условие отбора удаляемых записей. Оно имеет такой же формат, как и в инструкции SELECT (см. ниже).

Если предложение WHERE отсутствует, удаляются все записи таблицы. MySQL только помечает удаляемую строку как освобожденную. Пока она не будет затерта новой строкой, ее данные остаются на диске. Инструкция OPTIMIZE TABLE удаляет неиспользуемые строки и восстанавливает правильный формат таблицы.

Запросы к базе данных – команда Select

Команда **Select** заслуживает того, чтобы посвятить ей отдельную главу. Команда Select используется для запросов к базе данных с целью извлечения из нее информации. Синтаксис команды следующий:

```
SELECT [STRAIGHT_JOIN] [DISTINCT | ALL] select_expression,...
[FROM tables... [WHERE where_definition] [GROUP BY column,...]
[ORDER BY column [ASC | DESC], ...] HAVING full_where_definition
[LIMIT [offset,] rows] [PROCEDURE procedure_name]]
[INTO OUTFILE 'file_name'... ]
```

Как видно из вышеприведенного, вместе с командой Select используются ключевые слова, использование которых очень влияет на ответ сервера. Рассмотрим каждое из них.

- **DISTINCT**

Пропускает строки, в которых все выбранные поля идентичны, то есть устраняет дублирование данных.

- **WHERE**

Предложение команды Select, которое позволяет устанавливать предикаты, условие которых может быть верным или неверным для любой строки таблицы. Извлекаются только те строки, для которых такое утверждение верно. Например:

```
SELECT u_id, lname from publishers WHERE city = 'New York';
```

Выводит колонки u_id и lname из таблицы publishers для которых значение в столбце city - New York. Это дает возможность сделать запрос более конкретным.

- **Реляционные операторы.**

Реляционный оператор - математический символ который указывает на определенный тип сравнения между двумя значениями. Реляционные операторы которыми располагает MySQL :

= Равно
> Больше
< Меньше
>= Больше или равно
<= Меньше или равно
< > Не равно

Эти операторы имеют стандартные значения для числовых значений.

Предположим что вы хотите увидеть всех заказчиков с оценкой(rating) выше 200. Так как 200 - это скалярное значение, как и значение в столбце оценки, для их сравнения вы можете использовать реляционный оператор.

```
SELECT * FROM Customers WHERE rating > 200;
```

- **Булевы операторы.**

Основные Булевы операторы также распознаются в MySQL. Выражения Буля - являются или верными или неверными, подобно предикатам. Булевы операторы связывают одно или более верных/неверных значений и производят единственное верное или неверное значение. Стандартными операторами Буля распознаваемыми в SQL являются:AND,OR и NOT.

Предположим вы хотите видеть всех заказчиков в Далласе,которые имеют рейтинг выше 200:

```
SELECT * FROM Customers WHERE city = 'Dallas' AND rating > 200;
```

При использовании оператора AND,должны быть выполнены оба условия,то есть должны быть выбраны все заказчики из Далласа,рейтинг которых больше 200.

При использовании оператора OR,должно выполниться одно из условий.Например:

```
SELECT * FROM Customers WHERE city = 'Dallas ' OR rating > 200;
```

В данном случае будут выбраны все заказчики из Далласа и все имеющие рейтинг больше 200,даже если они и не из Далласа.

NOT может использоваться для инвертирования значений Буля.Пример запроса с NOT:

```
SELECT * FROM Customers WHERE city = 'Dallas' OR NOT rating > 200;
```

При таком запросе будут выбраны все заказчики из Далласа и все заказчики,рейтинг которых меньше 200.В этом запросе оператор NOT применяется только к выражению rating >200.Можно сделать более сложный запрос:

```
SELECT * FROM Customers WHERE NOT( city = 'Dallas' OR rating > 200 );
```

В этом запросе NOT применен к обеим выражениям в скобках. В данном случае, сервер читает выражения в скобках, определяет, соответствует ли истине равенство `city = 'Dallas'` или равенство `rating > 200`. Если любое условие верно, выражение Буля внутри круглых скобок верно. Однако, если выражение Буля внутри круглых скобок верно, предикат как единое целое неверен, потому что NOT преобразует верно в неверно и наоборот. То есть, будут выбраны все заказчики не находящиеся в Далласе и рейтинг которых меньше 200.

- **IN.**

Оператор IN определяет набор значений в которое данное значение может или не может быть включено. Например, запрос

```
SELECT * FROM Salespeople WHERE city = 'Barcelona' OR city = 'London';
```

может быть переписан более просто:

```
SELECT * FROM Salespeople WHERE city IN ('Barcelona', 'London');
```

IN определяет набор значений с помощью имен членов набора заключенных в круглые скобки и отделенных запятыми. Затем он проверяет различные значения указанного, пытаясь найти совпадение со значениями из набора. Если это случается, то предикат верен. Когда набор содержит значения номеров а не символов, одиночные кавычки опускаются.

- **BETWEEN.**

Оператор BETWEEN похож на оператор IN. В отличии от определения по номерам из набора, как это делает IN, BETWEEN определяет диапазон, значения которого должны уменьшаться что делает предикат верным. Вы должны ввести ключевое слово BETWEEN с начальным значением, ключевое AND и конечное значение. В отличии от IN, BETWEEN чувствителен к порядку, и первое значение в предложении должно быть первым по алфавитному или числовому порядку. Например:

```
SELECT * FROM Salespeople WHERE comm BETWEEN 10 AND 12;
SELECT * FROM Salespeople WHERE city BETWEEN 'Berlin' AND 'London';
```

- **LIKE.**

LIKE применим только к полям типа CHAR или VARCHAR, с которыми он используется чтобы находить подстроки. Т.е. он ищет поле символа чтобы видеть, совпадает ли с условием часть его строки. В качестве условия он использует групповые символы (wildcards) - специальные символы которые могут соответствовать чему-нибудь. Имеются два типа групповых символов используемых с LIKE:

- символ подчеркивания (`_`) замещает любой одиночный символ.
- знак `'%'`, замещающий любое количество символов.

Если мы зададим следующие условия:

```
SELECT * FROM Customers WHERE fname LIKE 'J%';
```

то будут выбраны все заказчики, чьи имена начинаются на J: John, Jerry, James и т.д.

- **COUNT.**

Агрегатная функция, производит подсчет значений в столбце или числа строк в таблице. При работе со столбцом использует DISTINCT в качестве аргумента:

```
SELECT COUNT (DISTINCT snum) FROM Orders;
```

При подсчете строк имеет синтаксис:

```
SELECT COUNT (*) FROM Customers;
```

- **GROUP BY.**

Предложение GROUP BY позволяет определять подмножество значений в особом поле в терминах другого поля, и применять функцию агрегата к подмножеству. Это дает возможность объединять поля и агрегатные функции в едином предложении SELECT. Например, предположим что вы хотите найти наибольшую сумму приобретений полученную каждым продавцом. Вы можете сделать отдельный запрос для каждого из них, выбрав MAX () из таблицы для каждого значения поля. GROUP BY позволит Вам поместить их все в одну команду:

```
SELECT snum, MAX (amt) FROM Orders GROUP BY snum;
```

- **ORDER BY.**

Эта команда упорядочивает вывод запроса согласно значениям в том или ином количестве выбранных столбцов. Многочисленные столбцы упорядочиваются один внутри другого, также как с GROUP BY.

- **DESC, ASC.**

DESC-DESCEDENT, вывод данных в обратном порядке (по алфавиту и численным значениям). По умолчанию используется ASC.

Проектирование базы данных.

Удачные базы данных редко возникают сами по себе, по мере добавления в них таблиц. Неизбежно приходится сталкиваться с множеством проблем, например с избыточностью. Процесс проектирования начинается с постановки исходных требований. Далее следует моделирование логической структуры данных. Обычно при этом создается графическое представление модели, называемое диаграммой отношения объектов. После этапа проектирования приступают к реализации БД. Здесь важно правильно выбрать типы столбцов таблиц. При проектировании нужно мыслить общими категориями, не привязываясь ни к одному языку программирования. Задачи системы должны быть функциональны по своей природе и понятны каждому. Хороший проект является простым, т.к. простые проекты легче понять, следовательно, реализация получится более качественной.

Диаграммы отношений объектов

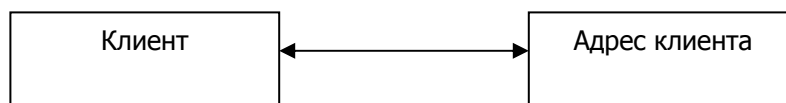
Диаграммы отношений объектов отображают 3 основных типа информации:

- объекты (сущности)
- отношения
- характер отношений

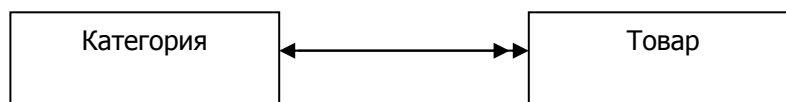
Объекты изображают прямоугольниками. Внутри прямоугольника – имя объекта. При проектировании реляционных БД объектом является таблица.



Прямые линии обозначают отношения между объектами. Характер отношения определяется числом строк, участвующих в нем с каждой стороны. Одной таблице может быть поставлена в соответствие одна или несколько строк другой таблицы. Одиночная стрелка означает единичную связь, а двойная – множественную.



Отношение «1:1»



Отношение «1:N»

Нормализация

Нормализация – метод организации реляционной БД с целью сокращения избыточности. В процессе нормализации неоптимальная таблица разбивается на две или более таблиц, между которыми создаются отношения.

Первая нормальная форма.

Первая нормальная форма требует, чтобы каждое поле таблицы было неделимым и не содержало повторяющихся групп. **Неделимость** означает, что наше поле не должно делиться на более мелкие. Например, речь идет о БД кадров. Будет ли поле «Фамилия» неделимым? Ясно, что будет. А поле «ФИО»? Тут все зависит от контекста – можно разделить на «Фамилия», «Имя», «Отчество», а можно оставить неделимым. «Адрес» тоже можно разделить на улицу, дом,

квартиру (тогда проще выбрать напр. всех живущих на одной улице) , а можно считать неделимым, если адрес – второстепенная информация. **Неповторяемость** означает, что мы не повторяем значение полей от одного поля к другому. Например, делаем записную книжку. Какие поля внесем ? Фамилия, Имя, Телефон, Email... А может несколько телефонов (рабочий, домашний, мобильный). А у некоторых может быть несколько рабочих или мобильных номеров или Email адресов. Правильнее всего создать еще одну таблицу с телефонами и поместить в нее ссылку на нашу таблицу.

ID	Фамилия	Адрес	ID	Вид телефона	Номер
1	Иванов	Адрес 1	1	Рабочий	22-33-44
2	Петров	Адрес 2	1	Рабочий	22-33-55
			1	Домашний	32-23-22
			2	Рабочий	45-67-89

Т.е. разбили таблицу на главную и подчиненную.

Вторая нормальная форма.

Вторая нормальная форма требует соответствия первой нормальной форме и чтобы каждая строка таблицы однозначно и избыточно определялась первичным ключом. Здесь 2 варианта : простой и очень простой. Первый – если например, накладные нумеруются каждый день, начиная с первой, то номер не может однозначно определить накладную, и не может быть первичным ключом. А номер+дата – может. Второй вариант – создаем поле-счетчик и указываем его в качестве первичного ключа.

Третья нормальная форма.

Третья нормальная форма требует, во-первых, соответствия второй нормальной форме. А во-вторых, чтобы только первичный ключ определял значения столбцов. Третья нормальная форма устраняет те столбцы, которые зависят от столбца, не являющегося первичным ключом. Это называется транзитивной зависимостью и ведет к дублированию данных.

ID	Фамилия	Адрес	Зарплата	№ проекта	Проект	Дата окончания
1	Иванов	Адрес 1	350	1	AAA	2004-11-03
2	Петров	Адрес 2	450	2	ССС	2004-10-06
3	Сидоров	Адрес 3	250	1	AAA	2004-07-06
4	Федоров	Адрес 4	300	2	ССС	2004-11-03

<u>ID</u>	<u>Фамилия</u>	<u>Адрес</u>	<u>Зарплата</u>	<u>N проекта</u>
1	Иванов	Адрес 1	350	1
2	Петров	Адрес 2	450	2
3	Сидоров	Адрес 3	250	1
4	Федоров	Адрес 4	300	2

<u>N проекта</u>	<u>Проект</u>	<u>Дата окончания</u>
1	AAA	2004-07-06
2	ССС	2004-10-06

Стратегия индексирования

Индекс по первичному ключу позволяет быстро найти нужную запись. Это пожалуй самый быстрый способ перейти к нужной записи. Кроме того, он незаменим при связывании таблиц друг с другом. **Правило 1 – каждой таблице по первичному ключу.**

Если приходится искать по неиндексированному полю, то СУБД переходит в режим сканирования, т е просто пробегает по каждой записи (поиск слова в словаре, пока не встретится нужное). **Правило 2 – поля, по которым ведется частый поиск должны быть проиндексированы.**

При каждом обновлении таблицы, индексы тоже обновляются. Если их слишком много, то это занимает определенное время. **Правило 3 – не загромождать таблицу лишними индексами.**