ООП

Основные понятия в ООП

ООП – Объектоно Ориентированное Программирование.

Класс в ООП — это шаблон по которому строятся объекты (Класс «Автомобили»).

Объект (экземпляр класса) – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом (Грузовик - является объектом класса - автомобиль).

Атрибуты – для примера, двигатель, подвеска, 4 колеса являют атрибутами.

Функции (Методы) – для примера, открывание дверей, поворот руля, звуковой сигнал являются методами.

Тип – это область определения некой величины, т.е. множество её возможных значений (int, long, short, byte, string и т.д.). Тип может задаваться классом.

Интерфейс – это набор методов класса, доступных для использования другими классами.

4 кита ООП

Наследование — свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником, дочерним или производным классом.
В качестве примера можно привести тот факт, что когда выпускается новая модель автомобиля она наследует некоторые методы предыдущей модели.

Полиморфизм подтипов (в ООП называемый просто «полиморфизмом») — свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.
В качестве примера можно привести тот факт что если у вас имеются права на вождение, независимо от того каким образом будет реализовываться движение автомобиля, интерфейс его остаётся прежним. И вы знаете, как им управлять, т.е. вам неважно это седан, внедорожник либо купе, либо автомобиль будет красного цвета или черного вы все равно будете уметь на нём ездить.

Инкапсуляция — свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе. Некоторые языки (например, C++) отождествляют инкапсуляцию с сокрытием, но большинство (Smalltalk, Eiffel, OCaml) различают эти понятия.

В качестве примера можно привести тот факт, что вся механика автомобиля от вас скрыта.

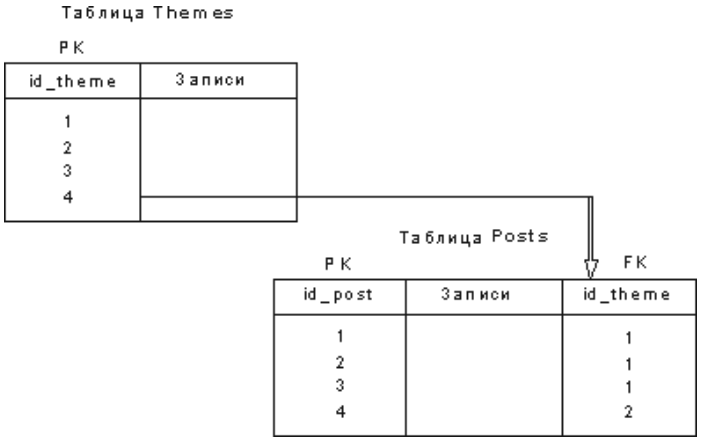
Абстрагирование означает выделение значимой информации и исключение из рассмотрения незначимой. В ООП рассматривают лишь абстракцию данных (нередко называя её просто «абстракцией»), подразумевая набор значимых характеристик объекта, доступный остальной программе. К примеру, вы знаете абстрактное представление автомобиля – самое главное, что он должен иметь – это 4 колеса, ходовую часть, двигатель и кузов. А такие элементы как на пример кондиционер, подогрев сидений и т.п., выделять не обязательно.

БД

Основные понятия в БД

Реляционная база данных — база данных, основанная на реляционной модели данных. Понятие «реляционный» основано на англ. relation («отношение», «зависимость», «связь»). Использование реляционных баз данных было предложено доктором Коддом из компании IBM в 1970 году. Для работы с реляционными БД применяют реляционные СУБД.

Первичный ключ (primary key) представляет собой уникальный индекс и применяется для уникальной идентификации записей таблицы. Никакие из двух записей таблицы не могут иметь одинаковых значений первичного ключа. Первичный ключ обычно сокращенно обозначают как PK (primarykey). Как мы уже говорили, в реляционных базах данных практически всегда разные таблицы логически связаны друг с другом. Первичные ключи как раз используются для однозначной организации такой связи. Первичный ключ не допускает значений Null и всегда должен иметь уникальный индекс. Первичный ключ используется для связывания таблицы с внешними ключами в других таблицах. К примеру, в базе данных Forum таблицы themes и posts связаны между собой следующим образом:



Вторичный (внешний) ключ (foreign key) - это одно или несколько полей (столбцов) в таблице, содержащих ссылку на поле или поля первичного ключа в другой таблице. Внешний ключ определяет способ объединения таблиц. Для сравнения первичный ключ – это одно или несколько полей (столбцов), комбинация значений которых однозначно определяет каждую запись в таблице.

Транза́кция (англ. transaction) — группа последовательных операций с базой данных, которая представляет собой логическую единицу работы с данными. Транзакция может быть выполнена либо целиком и успешно, соблюдая целостность данных и независимо от параллельно идущих других транзакций, либо не выполнена вообще и тогда она не должна произвести никакого эффекта. Транзакции обрабатываются транзакционными системами, в процессе работы которых создаётся история транзакций. Различают последовательные (обычные), параллельныеи распределённые транзакции. Распределённые транзакции подразумевают использование более чем одной транзакционной системы и требуют намного более сложной логики (например, two-phasecommit — двухфазный протокол фиксации транзакции). Пример: необходимо перевести с банковского счёта номер 5 на счёт номер 7 сумму в 10 денежных единиц. Этого можно достичь, к примеру, приведённой последовательностью действий:

Начать транзакцию

прочесть баланс на счету номер 5

уменьшить баланс на 10 денежных единиц

сохранить новый баланс счёта номер 5

прочесть баланс на счету номер 7

увеличить баланс на 10 денежных единиц

сохранить новый баланс счёта номер 7

Окончить транзакцию

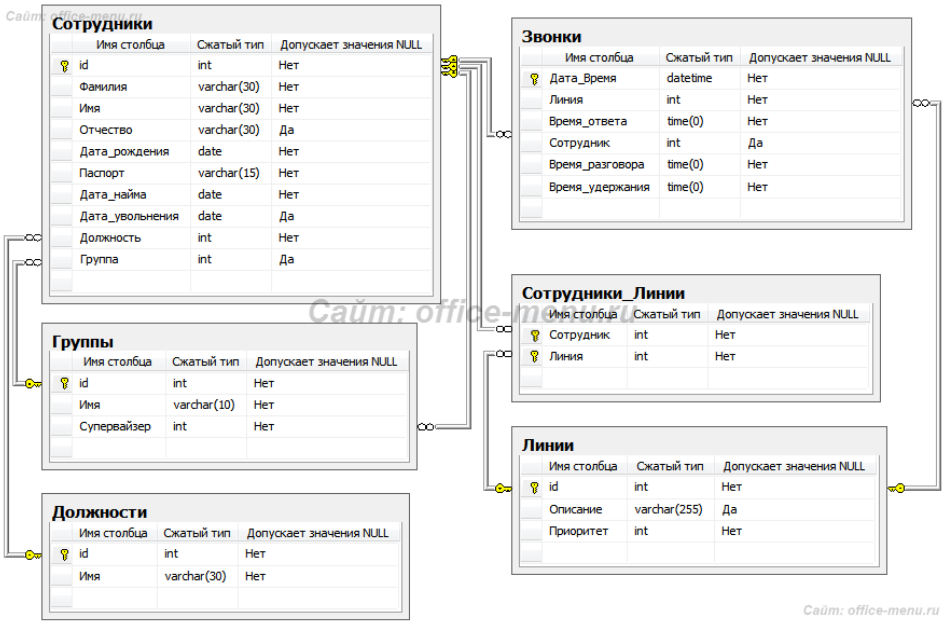
Эти действия представляют собой логическую единицу работы «перевод суммы между счетами», и таким образом, являются транзакцией. Если прервать данную транзакцию, к примеру, в середине, и не аннулировать все изменения, легко оставить владельца счёта номер 5 без 10 единиц, тогда как владелец счета номер 7 их не получит.

СВЯЗИ

Практически всегда БД не ограничивается одной таблицей. Сложно представить себе какой-либо бизнес-процесс на предприятии, который мог бы сконцентрироваться только на одном предмете в плане информации. Рассмотрим пример учебной базы данных. Имеется отдел, который

занимается обработкой звонков, поступающих на различные линии. Линии обслуживаются конкретными операторами. Операторы состоят в разных группах под присмотром супервайзеров. Только из данного краткого описания можно выделить несколько самостоятельных объектов:

Телефонные линии обслуживания;
Сотрудники отдела;
Должности сотрудников;
Группы, по которым распределены сотрудники;
Звонки.



Ознакомившись с диаграммой базы данных, можно обратить внимание на то, что некоторая информация из одних таблиц присутствует в других, т. е. между ними имеются связи. В нашем конкретном случае, все таблицы можно соединить между собой. Чтобы понять, как это правильно сделать, необходимо рассмотреть типы связей. Логику соединения таблиц в БД важно понять с самого начала изучения SQL, так как наверняка Вы не будете писать запросы только к одной таблице. Всего существует 3 типа связей:

Один к одному;
Один ко многим;
Многие ко многим.

Примечание:
в данном материале обозначения связей приводятся на примере MS SQL Server. В иных СУБД они могут обозначаться по-разному, но у Вас не должно возникнуть проблем с определением их типа, т. к. они либо очень похожи, либо интуитивно понятны.

Связь «Один к одному»

Связь один к одному образуется, когда ключевой столбец (идентификатор) присутствует в другой таблице, в которой тоже является ключом либо свойствами столбца задана его уникальность (одно и тоже значение не может повторяться в разных строках).

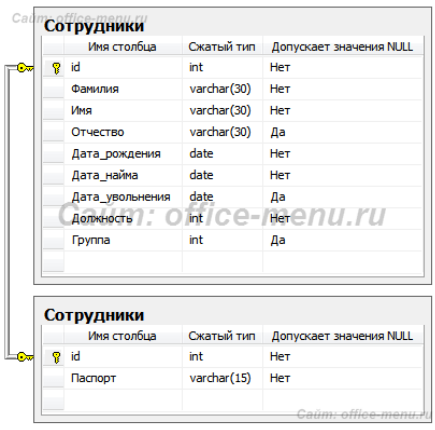
На практике связь «один к одному» наблюдается не часто. Например, она может возникнуть, когда требуется разделить данных одной таблицы на несколько отдельных таблиц с целью безопасности.

Вучебной безе данных нет подходящего примера, но гипотетически могла бы существовать необходимость разделения таблицы сотрудников.

Пример:
представьте, что базой данных пользуются несколько менеджеров и аналитиков, а таблица «Сотрудники» содержит те же столбцы, что и учебная база. Следовательно, доступ к персональным данным может получить любой из упомянутых работников.

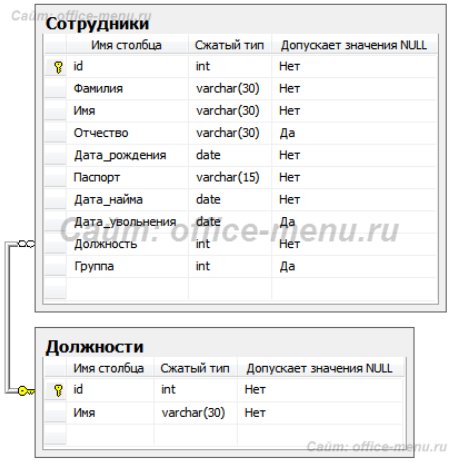
	id	Фамилия	Имя	Отчество	Дата_рождения	Паспорт	Дата_найма	Дата_увольнения	Должность	Группа
1	1	Мясников	Всеволод	Алексеевич	1987-01-02	6161 864808	2011-05-03	NULL	4	4
2	2	Беляев	Ярослав	Павлович	1990-01-02	8064 257116	2011-02-15	NULL	6	2
3	3	Степанов	Константин	Борисович	1981-02-05	9983 189856	2006-02-27	NULL	1	2
4	4	Калинина	Наталья	Анатольевна	1987-01-03	9453 359241	2013-03-08	NULL	4	1
5	5	Доронин	Олег	Леонидович	1992-04-04	1773 714978	2012-06-08	NULL	5	4
6	6	Никонов	Кирилл	Евгеньевич	1989-04-08	7428 261527	2011-10-31	NULL	4	1
7	7	Доронина	Дина	Вячеславовна	1993-01-04	9850 186129	2012-08-26	2014-08-16	5	3
8	8	Селезнёв	Владислав	Степанович	1992-02-12	5992 356004	2014-10-26	NULL	3	1
9	9	Дементьев	Иван	Степанович	1983-11-21	3543 037574	2010-11-23	NULL	5	3

Чтобы устранить возможность утечки конфиденциальной информации, принимается решение о переносе информации паспортных данных в отдельную таблицу, доступ к которой предоставляется ограниченному кругу лиц.



Связь «Один ко многим»

В типе связей один ко многим одной записи первой таблицы соответствует несколько записей в другой таблице. Рассмотрим связь учебной базы данных между должностями и сотрудниками, которая относится к рассматриваемому типу.

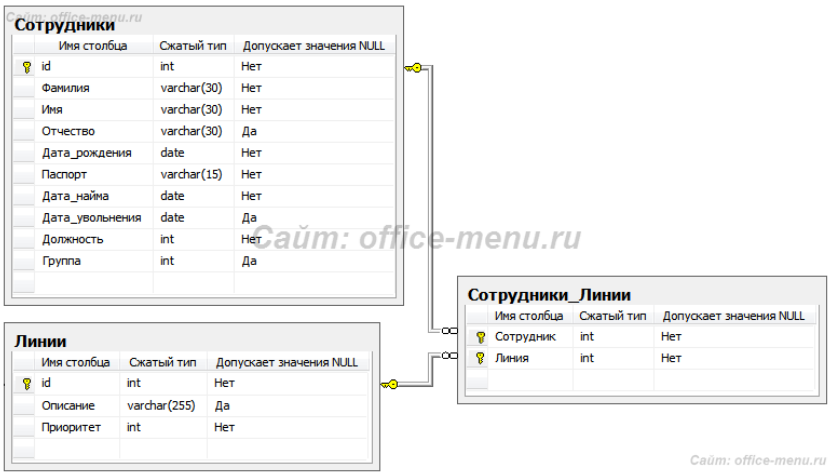


Записи должностей в таблице «Должность» уникальны, так как нет смысла повторно создавать имеющуюся запись. Записи в таблице «Сотрудники» также уникальны, но несколько различных сотрудников могут находиться на одинаковой должностной позиции. Символ ключа на конце связи указывает, что таблица, к которой этой конец прилегает, находится на стороне «один» (связанный столбец является первичным ключом), а символ бесконечности находится на стороне «многие» (такой столбец является внешним ключом).

Связь «Многие ко многим»

Если нескольким записям из одной таблицы соответствует несколько записей из другой таблицы, то такая связь называется «многие ко многим» и организовывается посредством связывающей таблицы.

В нашей базе подобное наблюдается только между таблицами с сотрудниками и линиями.



Из диаграммы видно, что имеются две связи «один ко многим» (один сотрудник может обрабатывать несколько телефонных линий, и одну линию могут обрабатывать несколько сотрудников), но в совокупности они образуют связь «многие ко многим».

Для чего все это нужно? Связи выполняют более важную роль, чем просто информация размещения данных по таблицам. Прежде всего, они требуются разработчикам для поддержания целостности баз данных. Правильно настроив связи, можно быть уверенным, что ничего не потеряется. Представьте, что Вы решили удалить одну из групп в таблице учебной базы данных. Если бы связи не было, то для тех сотрудников, которые к ней были определены, остался идентификатор несуществующей группы. Связь не позволит удалить группу, пока она имеется во внешних ключах других таблиц. Для начала следовало определить сотрудников в другие имеющиеся или новые группы, а только затем удалить ненужную запись. Поэтому связи называют еще ограничениями.

ACID (АСИУ)

ACID описывает требования к транзакционной системе (например, к СУБД), обеспечивающие наиболее надёжную и предсказуемую её работу. Требования ACID были в основном сформулированы в конце 70-х годов Джимом Греем.

Atomicity — Атомарность
Атомарность гарантирует, что никакая транзакция не будет зафиксирована в системе частично. Будут либо выполнены все её подоперации, либо не выполнено ни одной. Поскольку на практике невозможно одновременно и атомарно выполнить всю последовательность операций внутри транзакции, вводится понятие «отката» (rollback): если транзакцию не удаётся полностью завершить, результаты всех её до сих пор произведённых действий будут отменены и система вернётся во «внешне исходное» состояние — со стороны будет казаться, что транзакции и не было. (Естественно, счётчики, индексы и другие внутренние структуры могут измениться, но, если СУБД запрограммирована без ошибок, это не повлияет на внешнее её поведение.)

Consistency — Согласованность
Транзакция, достигающая своего нормального завершения (EOT — end of transaction, завершение транзакции) и, тем самым, фиксирующая свои результаты, сохраняет согласованность базы данных. Другими словами, каждая успешная транзакция по определению фиксирует только допустимые результаты. Это условие является необходимым для поддержки четвёртого свойства.
Согласованность является более широким понятием. Например, в банковской системе может существовать требование равенства суммы, списываемой с одного счёта, сумме, зачисляемой на другой. Это бизнес-правило и оно не может быть гарантировано только проверками целостности, его должны соблюсти программисты при написании кода транзакций. Если какая-либо транзакция произведёт списание, но не произведёт зачисление, то система останется в некорректном состоянии и свойство согласованности будет нарушено.
Наконец, ещё одно замечание касается того, что в ходе выполнения транзакции согласованность не требуется. В нашем примере, списание и зачисление будут, скорее всего, двумя разными подоперациями и между их выполнением внутри транзакции будет видно несогласованное состояние системы. Однако не нужно забывать, что при выполнении требования изоляции, никаким другим транзакциям эта несогласованность не будет видна. А атомарность гарантирует, что транзакция либо будет полностью завершена, либо ни одна из операций транзакции не будет выполнена. Тем самым эта промежуточная несогласованность является скрытой.

Isolation — Изолированность
Во время выполнения транзакции параллельные транзакции не должны оказывать влияние на её результат. Изолированность — требование дорогое, поэтому в реальных БД существуют режимы, не полностью изолирующие транзакцию (уровни изолированности Repeatable Read и ниже).

Durability — Устойчивость
Независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбои в оборудовании) изменения, сделанные успешно завершённой транзакцией, должны остаться сохранёнными после возвращения системы в работу. Другими словами, если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя.