**Основные функции СУБД, типовая организация СУБД.**

1) Непосредственное управление данными во внешней памяти

Эта функция обеспечивает поддержку необходимых структур внешней памяти как для хранения данных и метаданных, непосредственно входящих в БД, так и для служебных целей, например, для ускорения доступа к данным (индексы).

В некоторых реализациях СУБД используются возможности существующих ФС, в других работа производится на уровне устройств внешней памяти. Но развитых СУБД пользователи в любом случае не обязаны знать, использует ли СУБД ФС, и, если использует, то как организованы файлы. В СУБД обычно поддерживается собственная система именования объектов БД.

2) Управление буферами оперативной памяти.

СУБД обычно работают с БД значительного размера; этот размер обычно существенно больше объема доступной ОП.

Если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. Даже если операционная система производит общесистемную буферизацию данных (как, например, в случае ОС UNIX), этого недостаточно для целей СУБД, которая располагает гораздо большей информацией о полезности буферизации той или иной части БД. Поэтому в развитых СУБД поддерживается собственный набор буферов оперативной памяти с использованием собственной дисциплины замены буферов.

3) Управление транзакциями.

Транзакция - это последовательность операций над БД, рассматриваемых СУБД как единое целое:

1) либо транзакция успешно выполняется, и СУБД фиксирует (выполняет операцию COMMIT) изменения БД, произведенные этой транзакцией, во внешней памяти,

2) либо ни одно из этих изменений никак не отражается на состоянии БД.

Понятие транзакции необходимо для поддержания логической целостности БД. То свойство, что каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения, делает очень удобным использование понятия транзакции как единицы активности пользователя по отношению к БД.

С управлением транзакциями в многопользовательской СУБД связано важное понятие сериализации транзакций. Под сериализацией параллельно выполняемых транзакций понимается такой порядок планирования выполнения операций, при котором суммарный эффект всех транзакций эквивалентен эффекту их некоторого последовательного выполнения. Понятно, что если удается добиться действительно сериализованного выполнения всех транзакций, то для каждого пользователя, по инициативе которого образована транзакция, присутствие других транзакций будет незаметно.

Существует несколько базовых алгоритмов сериализации транзакций. Наиболее распространены алгоритмы, основанные на блокировках объектов БД. При использовании любого алгоритма сериализации возможны конфликты между двумя или более транзакциями по доступу к объектам БД. В этом случае для поддержки сериализации необходимо выполнить откат одной или более транзакций.

4) Журнализация.

Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Для обеспечения надежного хранения данных в БД требуется хранение избыточных данных, причем та часть данных, которая используется для восстановления БД, должна храниться особо надежно. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД. Журнал — это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью, в которую поступают записи обо всех изменениях основной части БД.

Принято придерживаться стратегии «упреждающей» записи в журнал. Эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД.

5) Поддержка языков БД

Для работы с базами данных используются специальные языки, в целом называемые языками баз данных. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. Чаще всего выделялись два языка: язык определения схемы БД (SDL - Schema Definition Language) и язык манипулирования данными (DML - Data Manipulation Language). SDL служил, главным образом, для определения логической структуры БД, т.е. той структуры БД, какой она представляется пользователям. DML содержал набор операторов манипулирования данными, т.е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные.

В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных. Стандартным языком наиболее распространенных в настоящее время реляционных СУБД является язык SQL (Standard Query Language). Прежде всего, язык SQL сочетает средства SDL и DML.

Именование объектов БД поддерживается на языковом уровне в том смысле, что компилятор языка SQL производит преобразование имен объектов в их внутренние идентификаторы на основании специально поддерживаемых служебных таблиц-каталогов. Внутренняя часть СУБД (ядро) вообще не работает с именами таблиц и их столбцов.

Язык SQL содержит специальные средства определения ограничений целостности БД. Ограничения целостности хранятся в специальных таблицах-каталогах, и обеспечение контроля целостности БД производится на языковом уровне.

Специальные операторы языка SQL позволяют определять представления БД, фактически являющиеся хранимыми в БД запросами (результатом любого запроса к реляционной БД является таблица) с именованными столбцами. Для пользователя представление является такой же таблицей, как любая базовая таблица, хранимая в БД, но с помощью представлений можно ограничить или наоборот расширить видимость БД для конкретного пользователя.

Логически в современной реляционной СУБД можно выделить наиболее внутреннюю часть – ядро СУБД, компилятор языка БД, подсистему поддержки времени выполнения, набор утилит. В некоторых системах эти части выделяются явно, но логически такое разделение можно провести во всех СУБД.

Ядро СУБД отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, управление транзакциями и журнализацию. Соответственно, можно выделить такие компоненты ядра: менеджер данных, менеджер буферов, менеджер транзакций и менеджер журнала.

Функции этих компонентов взаимосвязаны, и для обеспечения корректной работы СУБД все эти компоненты должны взаимодействовать по тщательно продуманным и проверенным протоколам.

Ядро СУБД обладает собственным интерфейсом, обычно не доступным пользователям напрямую и используемым в программах, производимых компилятором SQL и утилитах БД. При использовании архитектуры «клиент-сервер» ядро является базовой составляющей серверной части системы.

Основной функцией компилятора языка БД является компиляция операторов языка БД в некоторую выполняемую программу. Компилятор должен решить, каким образом выполнять оператор языка прежде, чем произвести программу.  
Применяются достаточно сложные методы оптимизации операторов. Результатом компиляции является выполняемая программа, представляемая в выполняемом внутреннем машинно-независимом коде. Реальное выполнение оператора производится с привлечением подсистемы поддержки времени выполнения, представляющей собой, интерпретатор этого внутреннего языка.

В отдельные утилиты БД обычно выделяют такие процедуры, которые слишком накладно выполнять с использованием языка БД, например, загрузка и выгрузка БД, сбор статистики, глобальная проверка целостности БД и т.д. Утилиты программируются с использованием интерфейса ядра СУБД, а иногда даже с проникновением внутрь ядра.

**Получение реляционной схемы из ER-диаграммы.**

Опишем типовую многошаговую процедуру преобразования ER-диаграммы в реляционную (более точно, в SQL-ориентированную) схему базы данных.

Каждый простой тип сущности превращается в таблицу. Имя сущности становится именем таблицы. Экземплярам типа сущности соответствуют строки соответствующей таблицы.

Каждый атрибут становится столбцом таблицы с тем же именем; может выбираться более точный формат представления данных. Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения. Столбцы, соответствующие обязательным атрибутам – не могут.

Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы . Если имеется несколько возможных уникальных идентификаторов, для первичного ключа выбирается наиболее характерный. Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи. Для именования этих столбцов используются имена концов связей или имена парных типов сущностей.

Связи «многие к одному » (и «один к одному») становятся внешними ключами, т.е. образуется копия уникального идентификатора сущности на конце связи «один» , и соответствующие столбцы составляют внешний ключ таблицы, соответствующей типу сущности на конце связи «многие». Необязательные связи соответствуют столбцам внешнего ключа, допускающим наличие неопределенных значений; обязательные связи – столбцам, не допускающим неопределенных значений . Если между двумя типами сущности A и B имеется связь «один к одному», то соответствующий внешний ключ по желанию проектировщика может быть объявлен как в таблице A, так и в таблице B. Чтобы отразить в определении таблицы ограничение, которое заключается в том, что степень конца связи должна равняться единице, соответствующий столбец должен быть дополнительно специфицирован как возможный ключ таблицы (в случае использования языка SQL для этого служит спецификация UNIQUE).

Для поддержки связи «многие ко многим» между типами сущности A и B создается дополнительная таблица AB с двумя столбцами, один из которых содержит уникальные идентификаторы экземпляров сущности A, а другой – уникальные идентификаторы экземпляров сущности B. Обозначим через УИД уникальный идентификатор экземпляра с некоторого типа сущности C. Тогда, если в экземпляре связи «многие ко многим» участвуют экземпляры a1, a2, ..., an типа сущности A и экземпляры b1, b2, ..., bm типа сущности B, то в таблице AB должны присутствовать все строки вида {УИД(ai), УИД(bj)} для i = 1, 2, ..., n, j = 1, 2, ..., m. Используя таблицы A, B и AB, с помощью стандартных реляционных операций можно найти все пары экземпляров типов сущности, участвующих в данной связи.

Индексы создаются для первичного ключа (уникальный индекс), внешних ключей и тех атрибутов, на которых предполагается в основном базировать запросы.

Представление в реляционной схеме супертипов и подтипов сущности.

Если в концептуальной схеме (ER-диаграмме) присутствуют подтипы, то возможны два способа их представления в реляционной схеме:

• (a) собрать все подтипы в одной таблице;

• (b) для каждого подтипа образовать отдельную таблицу.

При применении способа (a) таблица создается для максимального супертипа, а для подтипов могут создаваться представления.

Таблица содержит столбцы, соответствующие каждому атрибуту каждого подтипа. В таблицу добавляется, по крайней мере, один столбец, содержащий код типа, он становится частью первичного ключа . Для каждой строки таблицы значение этого столбца определяет тип сущности, экземпляру которого соответствует строка. Столбцы этой строки, которые соответствуют атрибутам и связям, отсутствующим в данном типе сущности, должны содержать неопределенные значения.  
При использовании второго метода для каждого подтипа первого уровня создается отдельная таблица.

Для более глубоких уровней наследования применяется первый метод. Супертип воссоздается с помощью объединения проекций таблиц, соответствующих подтипам, на заголовок таблицы супертипа.

У каждого способа есть свои достоинства и недостатки. К достоинствам первого способа (одна таблица для супертипа и всех его подтипов) можно отнести следующее:

• соответствие логике супертипов и подтипов; поскольку любой экземпляр  
любого подтипа является экземпляром супертипа, логично хранить вместе все строки, соответствующие экземплярам супертипа;

• обеспечение простого доступа к экземплярам супертипа и не слишком сложный доступ к экземплярам подтипов;

• возможность обойтись небольшим числом таблиц.

Недостатки метода (a):

• прикладная программа, имеющая дело с одной таблицей супертипа, должна включать дополнительную логику работы с разными наборами столбцов и разными ограничениями целостности;

• общая для всех подтипов таблица потенциально может стать узким местом при многопользовательском доступе по причине возможности блокировки  
таблицы целиком;

• для индивидуальных столбцов подтипов должна допускаться возможность  
содержать неопределенные значения; таким образом, потенциально в общей таблице будет содержаться много неопределенных значений , что при использовании некоторых РСУБД может потребовать значительного объема внешней памяти.

Достоинства метода (b) состоят в следующем:

• действуют более понятные правила работы с подтипами (каждому подтипу  
соответствует одноименная таблица);

• упрощается логика приложений;

• каждая программа работает только с нужной таблицей.

Недостатки метода (b):

• в общем случае требуется слишком много отдельных таблиц;

• работа с экземплярами супертипа на основе представления, объединяющего таблицы супертипов, может оказаться недостаточно эффективной;

• поскольку множество экземпляров супертипа является объединением множеств экземпляров подтипов, не все РСУБД могут обеспечить выполнение операций модификации экземпляров супертипа.

Представление в реляционной схеме взаимно исключающих связей  
Существуют два способа формирования схемы реляционной БД при наличии взаимно исключающих связей:

• (a) общее хранение внешних ключей;

• (b) раздельное хранение внешних ключей.

Если имеются взаимно исключающие связи упомянутой категории, то в таблице, соответствующей сущности, для которой связи являются взаимно исключающими, необходимо хранить внешние ключи. Если внешние ключи всех потенциально связанных таблиц имеют общий формат , то можно применить способ (a), т.е. создать два столбца: идентификатор связи и идентификатор сущности (возможно, составной). Столбец идентификатора связи используется для различения связей, покрываемых дугой исключения. В столбце (столбцах) идентификатора сущности хранятся значения уникального идентификатора сущности на дальнем конце соответствующей связи.

Если результирующие внешние ключи не относятся к одному домену, то приходится прибегать к использованию способа (b), т.е. создавать для каждой связи, покрываемой дугой исключения, явные столбцы внешних ключей; все эти столбцы могут содержать неопределенные значения.

Преимущество подхода (a) состоит в том, что в таблице, соответствующей  
сущности, появляется всего два дополнительных столбца. Очевидным недостатком является усложнение выполнения операции соединения: чтобы воспользоваться для соединения одной из альтернативных связей, нужно сначала произвести ограничение таблицы в соответствии с нужным значением столбца, содержащего идентификаторы связей.

При использовании подхода (b) соединения являются явными (и естественными). Недостаток состоит в том, что требуется иметь столько столбцов, сколько имеется альтернативных связей. Кроме того, в каждом из таких столбцов будет содержаться много неопределенных значений , хранение которых потенциально может привести к серьезным накладным расходам внешней памяти.

**Задача**

EMP {EMP\_NO, EMP\_NAME, EMP\_SAL, EMP\_DEPT\_NO, EMP\_PRP\_NO}

EMP\_NO – первичный ключ, EMP\_DEPT\_NO – внешний ключ, ссылающийся на DEPT (номер отдела служащего), EMP\_PRP\_NO – внешний ключ, ссылающийся на PRO (номер проекта служащего).

DEPT {DEPT\_NO, DEPT\_MNG}

DEPT\_NO – первичный ключ, DEPT\_MNG – внешний ключ, ссылающийся на EMP (номер служащего, являющегося менеджером отдела)

PRO {PRO\_NO, PRO\_MNG)

PRO\_NO – первичный ключ, PRO\_MNG – внешний ключ, ссылающийся на EMP (номер служащего, являющегося менеджером проекта)

Предположим, что мы имеем дело с объектной базой данных. EMP, DEPT и PRO – имена атомарных объектных типов (классов). Для каждого из этих типов существует одноименный экстент. Вместо внешних ключей – связи.

Сформулировать на OCL ограничение «в любом отделе работают хотя бы два менеджера проекта»

**Решение:**

context DEPT inv:

self.EMP -> select(EMP\_PRO\_NO.PRO\_MNG = EMP\_NO) -> size() >= 2