**15.01.2021**

**Основные понятия теории баз данных**

**Информация** – любые сведения о каком-либо событии, сущности, процессе и т.п., являющиеся объектом некоторых операций: восприятия, передачи, преобразования, хранения или использования.

**Данные** – это информация, зафиксированная в некоторой форме, пригодной для последующей обработки, передачи и хранения, например, находящаяся в памяти ЭВМ или подготовленная для ввода в ЭВМ.

**Система обработки данных (СОД)** – это набор аппаратных и программных средств, осуществляющих выполнение задач по управлению данными.

**Управление данными** – совокупность функции обеспечения требуемого представления данных, их накопления и хранения, обновления, удаления, поиска по заданному критерию и выдачи данных.

**Предметная область (ПрО)** – часть реального мира, подлежащая изучению с целью организации управления и, в конечном итоге автоматизации.

**База данных (БД)** – совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ.

**Системы управления базами данных**

**(СУБД)** – совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Базовые свойства СУБД:

* Скорость
* Разграничение доступа
* Гибкость
* Целостность
* Отказоустойчивость

Базовые функции СУБД:

* Интерпретация запросов пользователя, сформированных на специальном языке
* Определение данных (создание и поддержка специальных объектов, хранящих поступающие от внутреннего реестра объектов и их характеристик – так называемого словаря данных)
* Исполнение запросов по выбору, изменению или удалению существующих данных или удалению существующих данных или добавлению новых данных.
* Безопасность (контроль запросов пользователя на предмет попытки нарушения правил безопасности и целостности, задаваемых при определении данных)
* Производительность (поддержка специальных структур для обеспечения максимально быстрого поиска нужных данных)
* Архивирование и восстановление данных

**Целостность базы данных** – соответствие, имеющийся в базе данных информации ее внутренней логике, структуре и всем явно заданным правилам.

Каждое правило, налагающее некоторое ограничение на возможное состояние базы данных, называется **ограничением целостности.**

Ограничения целостности могут относиться к разным информационным объектам, атрибутам, кортежам, отношениям, связям между ними и т.д.

**Взаимосвязи в моделях и реляционный подход к построению моделей.**

**Жизненный цикл базы данных** – период времени от момента задумки о разработки базы данных для хранения информации до момента вывода из эксплуатации.

Этапы ЖЦБД:

1. планирование разработки базы данных;
2. определение требований к системе;
3. сбор и анализ требований пользователей;

1. проектирование базы данных:

* концептуальное проектирование базы данных; логическое проектирование базы данных;
* физическое проектирование базы данных;

1. Разработка приложений:

* проектирование транзакций;
* проектирование пользовательского интерфейса;

1. реализация;
2. загрузка данных;
3. тестирование;
4. эксплуатация и сопровождение:

* анализ функционирования и поддержка исходного варианта БД;
* адаптация, модернизация и поддержка переработанных вариантов.

**Планирование разработки –** разработка стратегического плана, в процессе которой осуществляется предварительное планирование конкретной системы управления базами данных. Важной частью разработки является проверка осуществимости проекта.

**Определение требований к системе –** на данном этапе необходимо определить диапазон действия приложения базы данных, состав его пользователей и области применения.

**Сбор и анализ требований пользователей –** необходимо создать модель движения важных материальных объектов и уяснить процесс документооборота.

**Проектирование базы данных –** полный цикл разработки базы данных включает концептуальное, логическое и физическое ее проектирование.

**Концептуальное проектирование базы данных:**

Сперва нужно создать концептуальную модель данных. Проектирование сложных баз данных осуществляется с использованием нисходящего подхода.

Семантическое моделирование данных, связанное со смысловым содержанием данных, независимо от их представлений в ЭВМ.

Созданная концептуальная модель данных является источником информации для фазы логического проектирования базы данных.

**Логическое проектирование базы данных**

Логическая модель, отражающая особенности представления о функционировании предприятия одновременно многих типов пользователей, называется *глобальной логической моделью данных.*

Процесс проектирования БД должен опираться на определенную модель данных.

**Физическое проектирование базы данных**

На данном этапе целью является создание описания СУБД ориентированной модели БД.

Для реляционной модели данных нужно:

Создание описания реляционных таблиц и ограничений для них на основе информации

Определение конкретных структур хранения данных и методов доступа к ним

Разработка средств защиты, создаваемой информацию.

**Разработка приложений**

Параллельно с проектированием системы базы данных выполняется разработка приложений. Главные составляющие данного процесса – это проектирование транзакций и пользовательского интерфейса.

**Транзакции –** могут состоять из нескольких операций, однако с точки зрения пользователей эти операции представляют собой единое целое, переводящее базу данных из одного непротиворечивого состояния в другое.

**Пользовательский интерфейс –** должен быть удобным и обеспечивать все функциональные возможности, предусмотренные в спецификациях требований пользователей.

**Реализация**

На данном этапе осуществляется реализация базы данных и разработанных приложений, позволяющих пользователю формулировать требуемые запросы к БД и манипулировать данными в БД.

**Загрузка данных**

На этом этапе созданные в соответствии со схемой базы данных пустые файлы, предназначенные для хранения информации, должны быть заполнены данными. Наполнение базы данных может протекать по-разному, в зависимости от того, создается ли база данных вновь или новая база данных предназначена для замены старой.

**Тестирование**

Для оценки законченности и корректности выполнения приложения базы данных может использоваться несколько различных стратегий тестирования.

Существует:

* Нисходящее тестирование – уровень подсистем с модулями, которые представлены заглушками, т.е. простыми компонентами, имеющими такой же интерфейс, как модуль, но без функционального кода.
* Восходящее тестирование – выполняется в противоположном направлении к нисходящему. Начинается с тестирования модулей на самых низких уровнях иерархии системы.
* Тестирование потоков – осуществляется при тестировании работающих в реальном масштабе времени систем, которые обычно состоят из большого количества взаимодействующих процессов, управляемых с помощью прерываний.
* Стратегия интенсивного тестирования – часто включает серию тестов с постепенно возрастающей нагрузкой и продолжается до тех пор, пока системы не выйдет из строя.

**Эксплуатация и сопровождения**

Основные действия, связанные с этим этапом, сводятся к наблюдению за созданной системой и поддержке ее нормального функционирования по окончании развертывания.

**Концептуальное проектирование**

**Фундаментальные понятия**

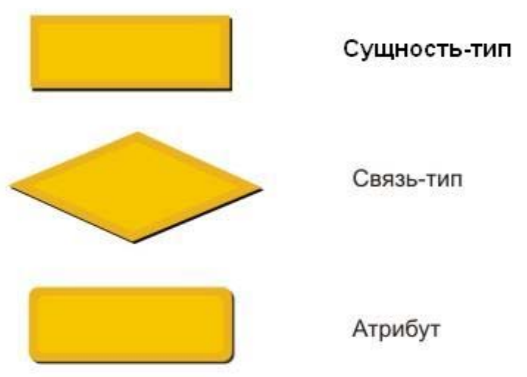
Для нормального функционирования информационной системы необходимо, чтобы концептуальная модель адекватно отображала реалии той предметной области, для которой она разрабатывалась.

Семантические методологии позволяют эффективно отображать существующую смысловую содержательность.

Главными элементами семантической модели данных являются сущности, их атрибуты и типы связей.

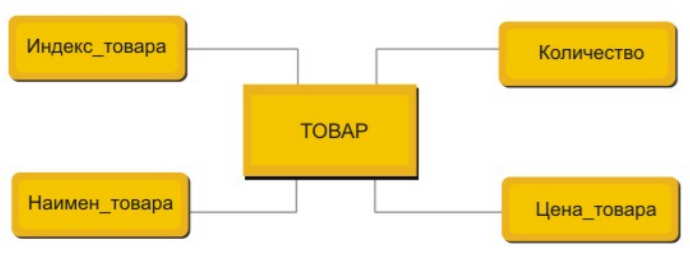
**Сущности**

Сущность – это то, о чем накапливается информация в информационной системе и что может быть однозначно идентифицировано. Характеризуется независимым существованием и представляет множество объектов реального мира с одинаковыми свойствами.



**Атрибуты**

Атрибут – это поименованная характеристика сущности, с помощью которой моделируется ее свойство. Каждой сущности присущи свои атрибты.



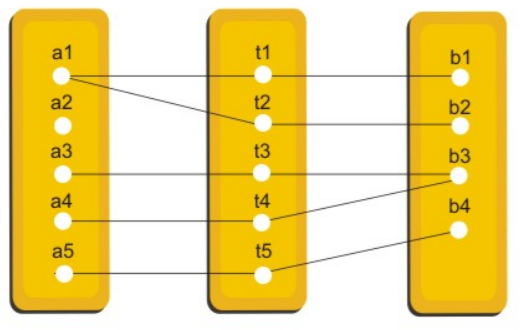
**Ключи**

Среди атрибутов особое положение занимают такие, с помощью которых можно идентифицировать экземпляр сущности. Такие атрибуты называются *ключами*. Атрибут или несколько атрибутов, значения которых уникальным образом идентифицируют каждый экземпляр сущности, являются *потенциальным* ключом данной сущности.

Один из потенциальных ключей может быть выбран в качестве *первичного ключа.*

**Связи между сущностями**

Две сущности могут быть связаны между собой. Подобная связь осуществляется через связь экземпляров одной сущности с экземплярами другой сущности, образуя набор экземпляров связи между двумя сущностями, который называется типов связи.



**Мощность связи**

Мощность обозначает максимальное количество экземпляров одной сущности, связанных с одним экземпляром другой сущности. Например, если допустить, что у человека может быть только один супруг, то мощность связи ЖЕНАТЫ будет равно одному в каждом направлении.

**Показатель кардинальности**

Для того, чтобы указать количество возможных связей для каждого экземпляра участвующего в связи сущности, используют показатель кардинальности

Для бинарных связей показатель кардинальности может иметь следующие значения:

* «Один к одному» (1 : 1)
* «Один ко многим» (1 : N)
* «Многие ко многим» (M : N)

Если максимальная мощность связи в обоих направлениях равна одному, мы называем ее связью «один к одному». (На факультете может быть один декан, и обратно, один и тот же декан может руководить только одним факультетом)

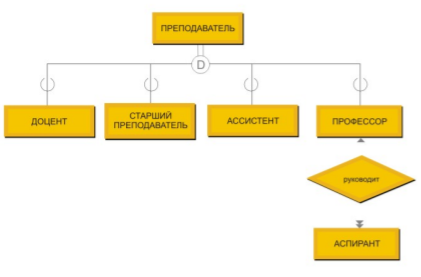
Если максимальная мощность в одном направлении равна одному, а в другом – многим, то связь называется «один ко многим». (в группе учится много студентов, но каждый студент только в одной группе)

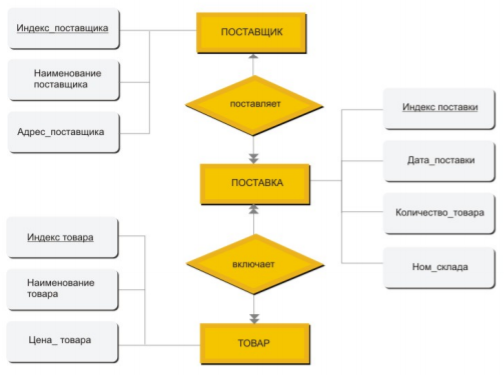
Если максимальная мощность в обоих направлениях равна многим, то такая связь относится к типу «многие ко многим». (преподаватель работает в разных группах, и в одной и той же группе работают разные преподаватели).

**Супертип и подтип**

Супертип – сущность, включающая разные подтипы, которые необходимо представить в модели данных.

Подтип – это сущность, являющаяся членом супертипа, но выполняющая отдельную роль в нем.



**Пример моделирования локальной ПрО**



**Модель данных** – это совокупность правил порождения структур данных в базе данных, операций над ними, а также ограничений целостности, определяющих допустимые связи и значения данных, последовательность их изменения [ГОСТ 20886-85].

Модель данных состоит из трёх частей:

**1. Набор типов структур данных.**

Здесь можно провести аналогию с языками программирования, в которых тоже есть предопределённые типы структур данных, такие как скалярные данные, векторы, массивы, структуры (например, тип *struct* в языке Си) и т.д.

**2. Набор операторов или правил вывода**, которые могут быть применены к любым правильным примерам типов данных, перечисленных в (1), чтобы находить, выводить или преобразовывать информацию, содержащуюся в любых частях этих структур в любых комбинациях.

**3. Набор общих правил целостности**, которые прямо или косвенно определяют множество непротиворечивых состояний базы данных и/или множество изменений её состояния.

**Ограничения целостности** – это правила, которым должны удовлетворять значения элементов данных. Ограничения целостности делятся на:

* **явные** (включаются в структуру базы данных с помощью средств языка контроля данных (DCL, Data Control Language))
* **неявные** (определяются самой структурой данных).

**Язык SQL**

Оператор SELCET

Select [all | distinct | distinctrow] select\_expr…

[FROM table\_references

[where where\_condition]

[group by {col\_name | expr}

[having where\_condition]

[order by {col\_name } expr]

[ASC | DESC],…]

select C:class, name, country from Classes C, Ships S where C.class=S.class and numGuns>=10;

**Соединение таблиц**

Внутреннее соединение:

* соединение по равенству

|  |  |  |  |
| --- | --- | --- | --- |
| id\_student | Id\_subject | Date\_mark | Mark |
| 101 | 1 | 2021-01-09 | 4 |
| 101 | 1 | 2021-01-15 | 5 |
| 102 | 3 | 2021-01-20 | 4 |
| 101 | 2 | 2021-01-20 | 5 |

* inner join

|  |  |
| --- | --- |
| Id\_student | FIO |
| 101 | Иванов А. А |
| 102 | Петров Т. К. |
| 103 | Сидоров П.М |

Select FIO, mark from student st, marks mr where st.id\_student=mr.id\_student;

Select FIO, mark from student st inner join marks mr on st.id\_student=mr.id\_student;

Внешнее соединение:

* left join
* right join
* full join

|  |  |
| --- | --- |
| FIO | Mark |
| Иванов А.А. | 4 |
| Иванов А.А. | 5 |
| Иванов А.А. | 5 |
| Петров Т.К. | 4 |
| Сидоров П.М. | Null |

**left join.** При левом соединении из левой таблицы берутся все записи, из правой соответствующие. Если соответствующих записей нет, то ставится значение 0.

Select FIO, mark from student st left join marks mr on st.id\_student=mr.id\_student;

|  |  |
| --- | --- |
| FIO | Mark |
| Иванов А.А. | 4 |
| Иванов А.А. | 5 |
| Иванов А.А. | 5 |
| Петров Т.К. | 4 |

**right join.**

Select FIO, mark from student st right join marks mr on

st.id\_student=mr.id\_student;

**full join**. Это объединение результатов левого и правого соединений

**Оператор UNION**

Оператор union объединяет результаты двух запросов. Если в результатах этих запросов есть повторяющиеся записи, то дубликаты удаляются. Если необходимо сохранить дубликаты, то необходимо использовать ‘union all’.

select …

union

select …;

**Ограничение на использование оператора UNION**

* Количество столбцов первого и второго запросов должно быть одинаковым
* Имена столбцов результата берутся из первого запроса
* Оператор ‘order by’ может быть использован только в конце второго запроса

Вывести фамилии студентов, у которых по предмету с номером 1 оценка 5 или 3.

Select FIO from student st, marks mr where mr.id\_subject=1 and (st.mark=3 or st.mark=5) and st.id\_student=mr.id\_student;

Select FIO from student st inner join marks mr on st.id\_student=mr.id\_student where mr.id\_subj=1 and mr.mark=3

Union

Select FION from student st inner join marks mr on st.id\_student=mr.id\_student wjere mr.id\_subject=1 and mr.mark=5;

**Кванторы всеобщности и существования**

Предикат ALL / ANY

Select … where … (<,>,=) ALL (select …);

Предикат ALL возвращает значение ИСТИНА, если оператор сравнения верен для всех значений, возвращаемых от запроса.

Предикат ANY истинен, если оператор сравнения верен хотя бы для одного значения, возвращаемых от запроса.

Замечания:

* Подзапрос должен возвращать только один столбец значений.

Предикат EXIST

Select … where exist (select …);

Предикат EXIST возвращается значение ИСТИНА, если подзапрос вернул не пустой результат.

Вывести фамилию студентов, у которых нет троек.

Select FIO from student st where not exist (select \*from marks mr where mark=3 and st.id\_student=mr.id\_student);

Select FIO from student st, marks mr where st.id\_student=mr.id\_student group by id having count (mark)>=ALL(select count mark from marks group by id\_student);

**Оператор IF**

select surname, if (max(mark)>=4, 'Надбавка', 'Нет надбавки') as 'надбавка' from student s inner join exam\_marks e on s.STUDENT\_ID=e.STUDENT\_ID group by s.STUDENT\_ID

**Функция GROUP\_CONCAT**

Объединяет результаты выборки из таблиц

Синтаксис

Group\_concat ([distinct] expr [, expr …]

[order by {unsigned\_integer | col\_name | expt}

[asc | desc [, col\_name …]]

[separator str\_val])

Задача: вывести фамилию преподавателя и список предметов, которые он ведет

Select concat (surname, ‘ ‘, name) as ‘ФИО’,

Group\_concat(subj\_name order by subj\_name separator \n’) as ‘ПРЕДМЕТЫ’

From lecturer l,

Subject s, subj\_lect sl

Where l.lecturer\_id=sl.lecturer\_id and s.subj\_id=sl.subj\_id

Group by l.lecturer\_id

**Операторы данных**

|  |  |  |
| --- | --- | --- |
| SQL | | |
| DML | DDL (язык определения данных) | DCL |
| SELECT  INSERT  UPDATE  DELETE | CREATE  DROP  ALTER | GRANT  REVOKE |

Атомарный DDL не является транзакционным DDL. Операторы DDL не могут выполняться в рамках другой транзакции, в рамках операторов управления транзакциями.

CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db\_name [create\_option] … create\_option: [DEFAULT] {CHARACTER SET [=] charset\_name | COLLATE [=] collation\_name | ENCRYPTION [=] {`Y` | `N`}}

Create database q3 default charset`latin1`

Create database if not exists q3 default charset`latin1`

Create table t2 as select \* from exam\_marks where 1 (нет связей)

Insert into exam\_marks select \* from t3 (копирует данные из одной в другую)

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name (create\_definition,…) [table\_options] [partition\_options]

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name [(create\_definition,…)] [table\_options] [partition\_options] [IGNORE | REPLACE] [AS] query\_expression

CREATE TABLE new\_tbl LIKE orig\_tbl;

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name {LIKE old\_tbl\_name | (LIKE old\_tbl\_name)}

CREATE TABLE new\_tbl AS SELECT \* FROM orig\_tbl;

**Создание индекса**

Индексы необходимы для поиска и связывания таблиц.

CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index\_name [index\_type] ON tbl\_name (key\_part,…) [index\_option] (algorithm\_option | lock\_option]

Key\_part: (col\_name [(length)] | (expr)) [ASC | DESC]

Create index ind\_fio on student(surname, name)

**Создание представлений**

CREATE

[OR REPLACE]

[ALGORITM = {UNDEFINED | MERGE | TEMTABLE}]

[DEFINER = user]

[SQL SECURITY {DEFINER | INVOKER}]

VIEW view\_name [(column\_list)]

AS select\_statement

[WITH [CASCADED | LOCAL] CHEK OPTION]

Create view mosc\_st as select \* from student where city=`Москва`

Create view mosc\_st as select \* from student where city=`Москва` with check option (нельзя будет изменить данные, если они не будут удовлетворять where)

**Удаление представлений**

DROP VIEW [IF EXISTS]

view\_name [, view\_name] …

[RESTIRCT | CASCADE]

**Хранимые процедуры**

Хранимая процедура – это именованный набор команд языка Transact-SQL, хранящиеся на сервере в качестве самостоятельного объекта БД.

Хранимая процедура хранится на сервере – реализуется модульность и уменьшается объем передаваемой информации по сети.

**Преимущества хранимых процедур**

* Повторное использование кода – после создания хранимой процедуры ее можно вызвать из любых приложений и SQL-запросов
* Безопасность – действия не приведут к нарушению целостности данных, т.к. для выполнения хранимой процедуры пользователь должен иметь привилегия
* Простота доступа – хранимые процедуры позволяют инкапсулировать сложный код и оформить его в виде простого вызова.
* Выполнение бизнес-логики – хранимые процедуры позволяют перенести код сохранения целосности БД из прикладной программы на сервер БД. Бизнес-логика в виде хранимых процедур не зависит от языка разработки приложений.

Создание хранимых процедур

create procedure *имя\_процедуры тип* [,…]]) [характеристика …] *тело\_процедуры*

delimiter //

create procedure name()

begin

select(‘Hello’);

end

//