МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра информационных систем и технологий

УТВЕРЖДАЮ
Проректор по учебной работе
_____О. Г. Локтионова
«6» декабря 2017г.

ПОСТРОЕНИЕ МОДЕЛИ ДАННЫХ В НОТАЦИИ IDEF1X

Методические указания по выполнению
лабораторных работ по дисциплине
«Проектирование информационных систем»
для студентов направления подготовки бакалавров
09.03.02 Информационные системы
09.03.03 Прикладная информатика

Курск 2017

УДК 004.82 (075.8)
Составитель: Т.И.Лапина

Рецензент
Доктор технических наук, профессор *Р.А.Томакова*

Проектирование информационных систем: методические указания по выполнению лабораторных работ / Юго-Зап. гос. ун-т; сост.: Т. И. Лапина, Курск, 2017. 20 с.: ил. 0, табл. 3, Библиогр.: с. 5.

Содержат краткие теоретические сведения о методах разработки требований и проектированию информационных систем, а также об инструментальных средах моделирования архитектуры и поддержки разработки программных средств информационных систем.

Методические указания соответствуют требованиям программ по направлениям подготовки бакалавров: 09.03.02 Информационные системы, 09.03.03 Прикладная информатика.

Предназначены для студентов направления подготовки бакалавров 09.03.02 Информационные системы, 09.03.03, Прикладная информатика дневной и заочной форм обучения.

Текст печатается в авторской редакции

Подписано в печать Формат 60х84 1/16.
Усл. печ. л. . Уч. – изд. л. .Тираж 100 экз. Заказ. Бесплатно.

Юго - Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

Лабораторная работа №5

1. Цель работы

Приобретение навыков построение модели данных в нотации IDEF1x с использованием языка SQL и CASE средств.

2. Основные теоретические положения

В SQL выделяются 2 типа значений: числовые и строковые.

Строковые значения задаются в одинарных кавычках.

В качестве разделителя дробной части в числовых значениях используется точка.

Примеры:

‘Строка’, ‘12/10/01’, 15, 0.2

Таблица1 – Команды определения данных

Команда	Описание
CREATE TABLE	Создаёт таблицу данных
CREATE INDEX	Создаёт индекс по одному или нескольким полям таблицы данных
ALTER TABLE	Изменяет структуры таблицы
DROP TABLE	Удаляет таблицу из базы данных
DROP INDEX	Удаляет индекс

CREATE TABLE

Используется для создания таблицы в базе данных.

Синтаксис:

```
CREATE TABLE table_name ( field1 type1 [nullable] [default_value][pk]
[,field2 type2 [nullable] [default_value][pk]] …
```

table_name – имя создаваемой таблицы.

Имя таблицы не должно содержать пробелов и должно содержать только латинские символы и цифры. Запрещено давать таблицам названия, начинающиеся с цифр.

field1, field2 – названия полей таблицы. Правила по именованию полей такие же, как и у таблиц.

type1, type2 – типы полей данных.

Допустимые типы данных и их обозначение определяются конкретными СУБД. Наиболее поддерживание типы данных приведены в следующей таблице

Таблица1 – Типы данных

Тип данных	Описание
CHAR(n)	Строковое поле фиксированной длины n
VARCHAR(n)	Строковое поле переменной длины (максимальная длина n)
INTEGER	Может хранить 32-разрядное целое число
NUMERIC(n,p)	Может хранить число с плавающей точкой, содержащее n цифр, p из которых – цифры после запятой
DATE	Предназначено для хранения даты

nullable – может принимать значение **NOT NULL**, что означает – поле обязательно должно иметь значение.

default_value - может принимать значение **DEFAULT** const, где const –константа, значение которой будет добавляться в поле. если при вставке не будет указано явно других значений.

pk- может принимать значение **PRIMARY KEY**. Означает, что данное поле также является первичным ключом.

Пример:

```
CREATE TABLE ACCOUNT (
CODE INTEGER NOT NULL PRIMARY KEY,
NAME VARCHAR (255) NOT NULL
)
```

Если в таблице используется составной первичный ключ, то необходимо использовать следующую конструкцию **PRIMARY KEY**:

```
PRIMARY KEY(field1 [, field2 …])
```

Пример:

```
CREATE TABLE STUDENT (
FAMILYNAME VARCHAR (64) not null,
NAME VARCHAR (64) not null,
ADDRESS VARCHAR(255),
PRIMARY KEY(FAMILYNAME, NAME)
)
```

Создания связей между таблицами (внешних ключей) используется следующая конструкция:

```
FOREIGN KEY (field1 [,field2 …])
REFERENCES master_table(key_field1 [, key_field2 …])
[ON UPDATE update_rule] [ ON DELETE delete_rule]
```

field1, field2 – поля создаваемой таблицы, участвующие в связи,

master_table – имя таблицы, являющейся главной в создаваемой связи,

key_field1, key_field2 – соответствующие полям **field1, field2** поля главной таблицы (обязательно должны входить в состав первичного ключа).

ON UPDATE и **ON DELETE** задают правила поведения подчинённой таблицы при изменении и удалении записей в главной таблице.

update_rule и **delete_rule** могут принимать следующие значения:

CASCADE - каскадное обновление (при изменении значения ключевых полей в главной таблице автоматически

меняются поля в подчиненной) или удаление (при удалении записи в главной таблице автоматически удаляются все записи в дочерней таблице, соответствующие значению первичного ключа удаляемой записи);

SET NULL – установка значения поля в **NULL** при модификации ключевого поля или удаления записи в главной таблице (при объявлении поля не должно присутствовать **NOT NULL**);

SET DEFAULT – установка значения по умолчанию, предусмотренного для поля, при модификации ключевого поля или удаления записи в главной таблице (при объявлении поля должно присутствовать выражение **DEFAULT**);

Пример:

```
CREATE TABLE DAYBOOK
(
DATE_OP DATE NOT NULL,
SUM_OP  NUMERIC(10,2) DEFAULT 0,
CODE_DEB   INTEGER NOT NULL,
CODE_KRED  INTEGER NOT NULL,
FOREIGN  KEY  (CODE_DEB)  REFERENCES  ACCOUNT
(CODE)
ON UPDATE CASCADE,
FOREIGN  KEY  (CODE_KRED)  REFERENCES  ACCOUNT
(CODE) ON UPDATE CASCADE
)
```

Примечание:

Некоторые СУБД (например, MS Access и MS SQL Server) позволяют использовать при именовании таблиц и полей символы различных языков и пробелы. Кроме того, некоторые из них допускают использование в названиях зарезервированных слов. Такие «особенные» названия должны заключаться в квадратные скобки (для MS Access и MS SQL Server) или в двойные кавычки (для Interbase, Firebird, Oracle).

Следующий пример удачно выполнится в Access:

```
CREATE TABLE [План счетов]
(
[Счёт] INTEGER PRIMARY KEY,
[Название] CHAR(255)
)
```

А этот пример успешно выполнится в Interbase, хотя слова **SUM** и **DATE** являются зарезервированными:

```
CREATE TABLE BOOK
(
"DATE" DATE NOT NULL,
"SUM" NUMERIC DEFAULT 0
)
```

CREATE INDEX

Используется для создания индекса.

Синтаксис:

```
CREATE [ASC|DESC] [UNIQUE] INDEX index_name
ON table_name(field1 [, field2 …]
```

index_name – имя создаваемого индекса,
table_name – имя таблицы для которой индекс создаётся,
field1, field2 – поля таблицы, включаемые в индекс.

ASC означает, что значения включаемых полей будут отсортированы по возрастанию (по умолчанию).
DESC – по убыванию.
UNIQUE означает. что в индекс будут включены только неповторяющиеся значения полей.

Пример:

```
CREATE INDEX IND_DATE_OP ON BOOK(DATE_OP)
```

ALTER TABLE

Используется для редактирования структуры существующей таблицы.

Синтаксис:

```
ALTER TABLE table_name ADD|DROP field_definition [,ADD|REMOVE field_definition …]
```

table_name – имя таблицы, структуру которой следует изменить,

ADD – позволяет добавить поле в таблицу,

REMOVE – позволяет удалить поле из таблицы,

field_definition – выражение вида `field1 type1 [nullable] [default_value][pk]` в случае **ADD** или имя удаляемого поля в случае **REMOVE**.

Пример:

```
ALTER TABLE book ADD comment VARCHAR(1024)
```

DROP TABLE

Используется для удаления таблицы из базы данных.

Синтаксис:

```
DROP TABLE table_name
```

table_name – имя удаляемой таблицы.

Внимание: если удаляемая таблица является главной по отношению к некоторым другим, то раннее должны быть удалены эти таблицы или удалены связь между ними.

Пример:

```
DROP TABLE BOOK
```

DROP INDEX

Используется для удаления индекса.

Синтаксис:

```
DROP INDEX index_name
```

index_name – имя удаляемого индекса.

Пример:

```
DROP INDEX IND_DATE_OP
```

Таблица1 – Команды определения данных

Команда	**Описание**
INSERT	Добавляет запись в таблицу
UPDATE	Редактирует записи в таблице, удовлетворяющие заданному критерию
DELETE	Удаляет из таблицы записи, удовлетворяющие заданному критерию

INSERT

Используется для добавления записи в таблицу.

Синтаксис:

```
INSERT INTO table_name [(field1[, field2…])]
VALUES (val1 [,val2 …])
```

table_name – название таблицы.

field1, field2 – список полей, значения которых в добавляемой записи будут соответственно равны **val1, val2**. Если список полей опущен, то значения определяются в порядке, заданном при создании таблицы (см. ***CREATE TABLE***).

Примеры:

```
INSERT INTO ACCOUNT (CODE, NAME) VALUES (50,
'Денежные средства в кассе')
```

```
INSERT INTO ACCOUNT VALUES (51, 'Денежные
средства на расчётном счёте')
```

UPDATE

Используется для редактирования записей, удовлетворяющих заданному критерию.

Синтаксис:

```
UPDATE table_name SET field1=value1 [, field2=value2 …]
[WHERE conditional]
```

table_name – название таблицы.
field1, field2 – поля таблицы.
value1, value2 – значения полей.
conditional определяет условие, удовлетворяющие которому записи будут модифицированы. Может состоять из одного или нескольких подусловий, объединённых командами **AND** (логическое И) или **OR** (логическое ИЛИ).

Возможные виды условий:

`field1 < | > | = | <> | >= | <= value1` (значение поля **field1** меньше, больше, равно, не равно, больше или равно, меньше или равно значения **value1**).

`field1 BETWEEN value1 AND value2` (значение поля **field1** между **value1** и **value2**)

`field1 LIKE value1` (значение поля **field1** «похоже» на **value1**, используется при проверке соответствия строковых полей некоторой маске).

`field1 IN (value1, value2 [,…])` (значение поля **field1** находится в списке **value1, value2**).

`field1 IS NULL` (значение поля **field1** отсутствует).

`field1 IS NOT NULL` (значение поля **field1** есть).

`NOT conditional` (выбирать только те записи, когда условие **conditional** не выполняется).

Также допустимо использование в условиях вложенных запросов на выборку **SELECT**.

Если выражение **WHERE** отсутствует, то редактируются все записи таблицы.

Примеры:

Устанавливает суммы всех хозяйственных операций, произведённых 1 января 2014 г. в 0

```
UPDATE BOOK SET SUM_OP=0 WHERE DATE_OP ='01.01.04'
```

Переименовывает название счёта 51

```
UPDATE ACCOUNT SET NAME='Расчётный счёт' WHERE CODE=51
```

DELETE

Используется для удаления записей из таблицы.

Синтаксис:

```
DELETE FROM table_name [WHERE conditional]
```

table_name – название таблицы.
conditional – условия отбора удаляемых записей (строится аналогично **UPDATE**).

Пример:

Удаляет все хозяйственные операции до 2003 г.

```
DELETE FROM BOOK WHERE DATE_OP<'01.01.03'
```

Запросы - вероятно наиболее часто используемый аспект SQL. Фактически, для категории SQL пользователей, маловероятно чтобы кто-либо использовал этот язык для чего-то другого.

Запрос - команда, которую дают программе базы данных, и которая сообщает ей чтобы она вывела определенную информацию из таблиц в память. Эта информация обычно посылается непосредственно на экран компьютера или терминала которым вы пользуетесь, хотя, в большинстве случаев, ее можно также послать принтеру, сохранить в файле (как объект в памяти компьютера), или представить как вводную информацию для другой команды или процесса.

Запросы обычно рассматриваются как часть языка DML. Однако, так как запрос не меняет информацию в таблицах, а просто показывает ее пользователю, мы будем рассматривать запросы как самостоятельную категорию среди команд DML которые производят действие, а не просто показывают содержание базы данных.

Все запросы в SQL состоят из одиночной команды. Эта команда называется - **SELECT**.

Команда SELECT

Синтаксис:

```
SELECT [DISTINCT] field_list FROM table_list
[WHERE conditional] [GROUP BY group_field_list]
[ORDER BY order_field_list]
```

field_list содержит список извлекаемых полей, разделённых запятой. Помимо названий полей этот список может содержать различные выражения по преобразованию информации (сложение, вычитание и т.д.). Если необходимо извлечь значения всех полей, то используется символ ***.**

Список **table_list** содержит одну или несколько таблиц, разделённых запятой с возможным указанием связей между таблицами (например, применение **INNER JOIN**), значения выбранных полей которых будут результатами запроса.

conditional задаёт условия отбора записей из списка таблиц.

group_field_list – содержит список полей, по значениям которых будет группироваться результат запроса, **order_field_list** – список полей, по которым результат будет отсортирован. Последние два списка строятся по тем же правилам, что и **field_list**. Кроме того, они могут содержать числа. Использование чисел равносильно использованию полей списка **field_list**, находящихся в позиции, указываемых этими числами.

Примеры:

Следующие запросы выведут всё содержимое таблицы «Счета»

```
SELECT CODE, NAME FROM ACCOUNT
```

```
SELECT * FROM ACCOUNT
```

А эти запросы отсортируют по возрастанию записи по полю CODE

```
SELECT CODE, NAME FROM ACCOUNT ORDER BY CODE
```

```
SELECT CODE, NAME FROM ACCOUNT ORDER BY 1
```

Ключевое слово **DISTINCT** запрещает вывод повторяющихся записей в результате запроса. Например, чтобы вывести все даты, при которых суммы, участвующие в хозяйственных операциях были не меньше 100 000 р., нужно выполнить следующий запрос

```
SELECT DISTINCT DATE_OP FROM BOOK WHERE SUM_OP>=100000
```

Построение условий запроса аналогично построению условий в команде UPDATE. Следующий запрос выведет даты 2014 года, при которых суммы, участвующие в хозяйственных операциях, находились в диапазоне 25 000 – 75 000 р.

```
SELECT DISTINCT DATE_OP FROM BOOK WHERE DATE_OP >='01.01.14' AND SUM_OP BETWEEN 25000 AND 75000
```

Одна из наиболее важных особенностей запросов SQL - это их способность определять связи между многочисленными таблицами и выводить информацию из них в терминах этих связей, всю внутри одной команды. Этот вид операции называется - объединением, которое является одним из видов операций в реляционных базах данных.

Главное в реляционном подходе - это связи, которые можно создавать между позициями данных в таблицах. Используя объединения, мы непосредственно связываем информацию с

любым номером таблицы, и таким образом способны создавать связи между сравнимыми фрагментами данных. При объединении, таблицы представленные списком в предложении **FROM** запроса, отделяются запятыми. Предикат запроса может ссылаться к любому столбцу любой связанной таблицы и, следовательно, может использоваться для связи между ними. Обычно, предикат сравнивает значения в столбцах различных таблиц, чтобы определить, удовлетворяет ли **WHERE** установленному условию.

Полное имя столбца таблицы фактически состоит из имени таблицы, сопровождаемого точкой и затем именем столбца. Имеются несколько примеров имён:

```
ACCOUNT.CODE
BOOK.SUM_OP
```

До этого, мы могли опускать имена таблиц, потому что вы запрашивали только одну таблицу одновременно, а SQL достаточно интеллектуален, чтобы присвоить соответствующий префикс, имени таблицы. Даже когда мы делаем запрос многочисленных таблиц, мы еще можем опускать имена таблиц, если все ее столбцы имеют различные имена. Но это не всегда так бывает.

Следующий пример выведет состав каждой хозяйственной операции:

```
SELECT OPERATION.NAME, ACT.NAME, ACT.CODE_DEB, ACT.CODE_KRED FROM OPERATION, ACT WHERE OPERATION.ID =ACT.ID_OP
```

Для того, чтобы в полном названии полей не использовать названия таблиц, длина имён которых может быть достаточно большой, допустимо использовать псевдонимы. Следующая команда будет эквивалентна предыдущей:

```
SELECT O.NAME, A.NAME, A.CODE_DEB, A.CODE_KRED FROM OPERATION O, ACT A WHERE O.ID =A.ID_OP
```

Необходимо обратить внимание на то, что псевдонимы объявляются в списке таблиц, следом за таблицей, отделяясь от неё пробелом.

Стандартом ANSI 92 SQL для представления объединений рекомендуется использовать спецификатор **JOIN** в предложении **FROM** вместо задания условий объединения в предложении **WHERE**.

Операция INNER JOIN

Объединяет записи из двух таблиц, если связующие поля этих таблиц содержат одинаковые значения.

Синтаксис:

```
FROM table_name1 INNER JOIN table_name2 ON
table_name1.field1 = | <= | >= | < | > | <>
table_name2.field2
```

table_name1, table_name2 - имена таблиц, из которых объединяются записи.

field1, field 2 - имена связываемых полей. Если поля не содержат числовых данных, они должны относиться к одному типу данных и содержать некоторые данные. Имена этих полей могут быть разными.

Операцию **INNER JOIN** можно использовать в любом предложении FROM. Это самый распространенный тип объединения. С его помощью происходит объединение записей из двух таблиц по связующему полю, если оно содержит одинаковые значения в обеих таблицах.

Как правило, запросы, содержащие операцию **INNER JOIN**, выполняются быстрее, чем при использовании этих же условий в **WHERE**.

Операции LEFT JOIN, RIGHT JOIN

Синтаксис:

```
FROM таблица1 [ LEFT | RIGHT ] JOIN
ON table_name1.field1 = | <= | >= | < | > |
<>  table_name2.field2
```

Для того, чтобы создать левое внешнее объединение, необходимо использовать **LEFT JOIN**. С помощью левого внешнего объединения выбираются все записи первой (левой) таблицы, даже если они не соответствуют записям во второй (правой) таблице.

RIGHT JOIN позволит создать правое внешнее объединение. С помощью правого внешнего объединения выбираются все записи второй (правой) таблицы, даже если они не соответствуют записям в первой (левой) таблице.

Обобщение данных с помощью агрегатных функций

Рассмотрим, как использовать значения, полученные путём выполнения простых запросов, чтобы получить из них информацию. Это делается с помощью агрегатных или общих функций, которые берут группы значений из поля и сводят их до одиночного значения. Изучим, как использовать эти функции, как определить группы значений, к которым они будут применяться, и как определить какие группы выбираются для вывода. Рассмотрим, при каких условиях можно объединить значения поля с этой полученной информацией в одиночном запросе.

Запросы могут производить обобщенное групповое значение полей точно также как и значение одного поля. Это делает с помощью агрегатных функций. Агрегатные функции производят одиночное значение для всей группы таблицы. Имеется список этих функций:

- **COUNT** производит номера строк или **не-NULL** значения полей которые выбрал запрос.
- **SUM** производит арифметическую сумму всех выбранных значений данного поля.

- **AVG** производит усреднение всех выбранных значений данного поля.
- **MAX** производит наибольшее из всех выбранных значений данного поля.
- **MIN** производит наименьшее из всех выбранных значений данного поля.

Агрегатные функции используются подобно именам полей в предложении **SELECT** запроса, но с одним исключением, они берут имена поля как аргументы. Только числовые поля могут использоваться с **SUM** и **AVG**. **COUNT**, **MAX**, и **MIN**, могут использоваться и числовые или символьные поля. Когда они используются с символьными полями, **MAX** и **MIN** будут транслировать их в эквивалент **ASCII**, который должен сообщать, что **MIN** будет означать первое, а **MAX** последнее значение в алфавитном порядке.

Чтобы найти максимальную сумму, когда-либо фигурировавшую в хозяйственных операциях, необходимо выполнить следующих запрос:

```
SELECT MAX (SUM_OP) FROM BOOK
```

Это конечно, отличается от выбора поля при котором возвращается одиночное значение, независимо от того сколько строк находится в таблице. Из-за этого, агрегатные функции и поля не могут выбираться одновременно, пока предложение **GROUP BY** (описанное далее) не будет использовано.

Функция **COUNT** несколько отличается от всех. Она считает число значений в данном столбце, или число строк в таблице. Чтобы подсчитать общее число строк в таблице, используйте функцию **COUNT** со звездочкой вместо имени поля. Так следующий запрос выведет количество записей в таблице **ACCOUNT**:

```
SELECT COUNT (*)
FROM ACCOUNT
```

Предложение GROUP BY

Предложение **GROUP BY** позволяет определять подмножество значений в особом поле в терминах другого поля, и применять функцию агрегата к подмножеству. Это дает вам возможность объединять поля и агрегатные функции в едином предложении **SELECT**. Например, предположим, что необходимо вывести количество хозяйственных операций, производимых в разрезе каждого дня

```
SELECT DATE_OP , COUNT(*) FROM BOOK GROUP BY DATE_OP
```

GROUP BY применяет агрегатные функции независимо от серий групп которые определяются с помощью значения поля в целом. В этом случае, каждая группа состоит из всех строк с тем же самым значением поля **DATE_OP**, и **COUNT** функция применяется отдельно для каждой такой группы. Это значение поля, к которому применяется **GROUP BY**, имеет, по определению, только одно значение на группу вывода, также как это делает агрегатная функция. Результатом является совместимость, которая позволяет агрегатам и полям объединяться таким образом.

Можно также использовать **GROUP BY** с многочисленными полями.

Предложение HAVING

Предположим, что в предыдущем примере, хотелось бы увидеть только то количество операций, значение которого больше 15. Мы не сможем использовать агрегатную функцию в предложении **WHERE**, потому что предикаты оцениваются в терминах одиночной строки, а агрегатные функции оцениваются в терминах групп строк. Это означает, что мы не сможем сделать что-нибудь подобно следующему:

```
SELECT DATE_OP , COUNT(*) FROM BOOK WHERE COUNT(*)>15 GROUP BY DATE_OP
```

Это будет отклонением от строгой интерпретации ANSI. Чтобы увидеть максимальную стоимость приобретений свыше $3000.00, можно использовать предложение **HAVING**.

Предложение **HAVING** определяет критерии, используемые чтобы удалять определенные группы из вывода, точно также как предложение **WHERE** делает это для индивидуальных строк.

Правильной командой будет следующяя

```
SELECT DATE_OP , COUNT(*) FROM BOOK WHERE COUNT(*)>15 GROUP BY DATE_OP
```

Аргументы в предложении **HAVING** следуют тем же самым правилам что и в предложении **SELECT**, состоящей из команд использующих **GROUP BY**. Они должны иметь одно значение на группу вывода.

3. Контрольные вопросы

1. Назовите ключевые слова языка SQL при таблицы данных?
2. Какие агрегатные функции база данных знаете?
3. Какая команда устанавливает связь между таблицами данных?
4. Как удалить таблицу данных?
5. Как установить индекс для таблицы данных?
6. Назовите ключевое слово языка SQL для группировки данных в таблицах.

Список литературы

1. Лапина, Т. И. Информационные системы. Проектный практикум к выполнению и защите ВКР бакалавра по направлению 09.03.02 Информационные системы, 09.03.03 Прикладная информатика/ Т. И. Лапина//Юго-Западный гос. ун-т, ЗАО «Университетская книга»–Курск, 2016.–99с.
2. Золотов, С. Ю. Проектирование информационных систем [Электронный ресурс] : учебное пособие / С. Ю. Золотов. - Томск : Эль Контент, 2013. - 88 с.
3. Абрамов, Г. В. Проектирование информационных систем [Электронный ресурс] : учебное пособие / Г. В. Абрамов, И. Медведкова, Л. Коробова. - Воронеж : Воронежский

государственный университет инженерных технологий, 2012. - 172 с.
4. Стасышин, В. М. Проектирование информационных систем и баз данных [Электронный ресурс] : учебное пособие / В. М. Стасышин. - Новосибирск : НГТУ, 2012. - 100 с.
5. Вендров, А.М. Проектирование программного обеспечения [Текст] : учебник / А.М. Вендров.— М: Финансы и Статистика, 2006. — 352с.