МИНЕСТЕРСТВО РАЗВИТИЯ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИИ РЕСПУБЛИКИ УЗБЕКИСТАН
«ТАШКЕНСТКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ИМЕНИ АЛЬ-ХОРЕЗМИЙ»

Факультет "Программный инжиниринг"
Кафедра "Системного и прикладного программирования "

ПРОГРАММИРОВАНИЕ НА SQL

Методические указания

по выполнению лабораторных работ

Ташкент 2022 год

Абдувалиева.З.А методическое указание по выполнению лабораторных работ по предмету "Программирование на SQL"Т.:Aloqachi, 2022 49 стр

Данное учебное пособие по предмету "Программирование на SQL разработано для студентов по всем направлениям для получения знаний навыков и способствует выполнению лабораторных работ. В конце учебного методического комплекса приведен список используемой и рекомендованной литературы.

Опубликовано на основе утвержденного учебно-методическим советом Ташкентского Университета Информационных Технологий

Рецензенты:
Рахмонов.А.Т -Доцент кафедры "Системное и прикладное программирование", д.м.н.

С.А.Алибеков-Доцент кафедры "Информационные технологии"ТГПУ Имени Низомий д.ф-м.н.

Ташкентский Университет Информационных Технологий -2022 г
2013

СОДЕРЖАНИЕ

Введение 3
Лабораторная работа № 1 «Создание SQL-запросов» 4
Лабораторная работа № 2 «Создание концептуальной модели данных в среде автоматизированного проектирования» 24
Лабораторная работа № 3 «Генерация физической модели и структуры базы данных» 35
Указания к выполнению курсового проекта 43
Самостоятельная работа 46
Рекомендуемая литература 48
Приложение 1 49
Приложение 2 50

Введение

Лабораторный практикум направлен на приобретение навыков разработки баз данных, создания пользовательских SQL-запросов в среде СУБД MS Access и элементов пользовательского интерфейса.

Процесс изучения дисциплины направлен на формирование следующих компетенций:

- способность к формализации в своей предметной области с учетом ограничений используемых методов исследования (ПК-2);
- навыки использования операционных систем, сетевых технологий, средств разработки программного интерфейса, применения языков и методов формальных спецификаций, систем управления базами данных (ПК-15);
- умение применять основные методы и инструменты разработки программного обеспечения (ПК-17).

По завершении лабораторного практикума студенты с учетом полученных теоретических знаний должны:

Уметь:

- производить моделирование предметной области, уметь строить для нее ER-диаграмму и отображать ER-диаграмму в схему реляционной базы данных;
- разрабатывать все виды запросов на языке SQL;
- разрабатывать информационные системы для работы со сложноструктурированными базами данных: экранные формы, отчеты, разрабатывать для конкретного применения все виды запросов в выбранном диалекте языка SQL;
- строить индексы;

Владеть:

- методикой проектирования баз данных на основе нормализации отношений.
- средствами разработки баз данных и простых элементов пользовательского интерфейса в современных СУБД.

Проверка формирования заявленных компетенций, знаний, умений и навыков осуществляется путем защиты лабораторных работ, обоснования выбранных технических решений и способов достижения результата. Особое внимание при оценке компетенций ПК-2 и ПК 15 уделяется защите индивидуального задания.

На проведение лабораторных работ отводится 18 аудиторных часов.

На самостоятельную подготовку, в том числе на подготовку к лабораторным занятиям, отводится 126 часов.

Лабораторная работа № 1 «Создание SQL-запросов»

Тема: Создание SQL-запросов.

Раздел дисциплины: Реляционные языки.

Цель работы: создать SQL-запросы на создание таблицы, на выборку с параметрами, на обновление записей, на удаление записей, на добавление данных, на удаление таблицы, на создание индексов.

Продолжительность: 8 часов

Основы SQL

Запрос SQL — это запрос, создаваемый при помощи инструкций SQL. Язык SQL (Structured Query Language) используется при создании запросов, а также для обновления и управления реляционными БД.

В среде MS Access, когда пользователь создает запрос в режиме конструктора запроса (с помощью построителя запросов), MS Access автоматически создает эквивалентную инструкцию SQL. Фактически, для большинства свойств запроса, доступных в окне свойств в режиме конструктора, имеются эквивалентные предложения или параметры языка SQL, доступные в режиме SQL. При необходимости, пользователь имеет возможность просматривать и редактировать инструкции SQL в режиме SQL. После внесения изменений в запрос в режиме SQL его вид в режиме конструктора может измениться.

Некоторые запросы, которые называют запросами SQL, невозможно создать в бланке запроса. Для запросов к серверу, управляющих запросов и запросов на объединение необходимо создавать инструкции SQL непосредственно в окно запроса в режиме SQL. Для подчиненного запроса пользователь должен ввести инструкцию SQL в строку Поле или Условие отбора в бланке запроса.

Синтаксиса написания SQL-предложений:

– в описании команд слова, написанные прописными латинскими буквами, являются зарезервированными словами SQL;

– фрагменты SQL-предложений, заключенные в фигурные скобки и разделенные символом «|», являются альтернативными. При формировании соответствующей команды для конкретного случая необходимо выбрать одну из них;

– фрагмент описываемого SQL-предложения, заключенный в квадратные скобки [ ], имеет необязательный характер и может не использоваться;

– многоточие …, стоящее перед закрывающейся скобкой, говорит о том, что фрагмент, указанный в этих скобках, может быть повторен;

Описание команд SQL

Выборка записей

Инструкция **SELECT**. При выполнении инструкции SELECT СУБД находит указанную таблицу или таблицы, извлекает заданные столбцы, выделяет строки, соответствующие условию отбора, и сортирует или группирует результирующие строки в указанном порядке в виде набора записей.

Синтаксис команды:

SELECT [предикат] { * | таблица.* | [таблица.]поле_1
[AS псевдоним_2] [, [таблица.]поле_2[AS псевдоним_2] [, ...]]}
FROM выражение [, ...]
[WHERE...]
[GROUP BY...]
[HAVING...]
[ORDER BY...]

где предикат — один из следующих предикатов отбора: ALL, DISTINCT, DISTINCTROW, TOP. Данные ключевые слова используются для ограничения числа возвращаемых записей. Если они отсутствуют, по умолчанию используется предикат ALL;

* указывает, что результирующий набор записей будет содержать все поля заданной таблицы или таблиц. Следующая инструкция отбирает все поля из таблицы «Студенты»: SELECT * FROM Студенты;

таблица — имя таблицы, из которой выбираются записи;

поле_1, поле_2 — имена полей, из которых должны быть отобраны данные;

псевдоним_1, псевдоним_2 — ассоциации, которые станут заголовками столбцов вместо исходных названий полей в таблице;

выражение — имена одной или нескольких таблиц, которые содержат необходимые для отбора записи;

предложение GROUP BY в SQL-предложении объединяет записи с одинаковыми значениями в указанном списке полей в одну запись. Если инструкция SELECT содержит статистическую функцию SQL, например Sum или Count, то для каждой записи будет вычислено итоговое значение;

предложение HAVING определяет, какие сгруппированные записи, выданные в результате выполнения запроса, отображаются при использовании инструкции SELECT с предложением GROUP BY. После того как записи результирующего набора будут сгруппированы с

помощью предложения GROUP BY, предложение HAVING отберет те из них, которые удовлетворяют условиям отбора, указанным в предложении HAVING;

предложение ORDER BY позволяет отсортировать записи, полученные в результате запроса, в порядке возрастания или убывания на основе значений указанного поля или полей.

Следует отметить, что инструкции SELECT не изменяют данные в базе данных. Приведем минимальный синтаксис инструкции SELECT: SELECT поля FROM таблица.

Если несколько таблиц, включенных в предложение FROM, содержат одноименные поля, перед именем такого поля следует ввести имя таблицы и оператор « . » (точка). Предположим, что поле «Номер_группы» содержится в таблицах «Студенты» и «Группы». Следующая инструкция SQL отберет поле «Номер_группы» и «ФИО_студента» из таблицы «Студенты» и «ФИО_куратора» из таблицы «Группы» при номере группы, равном 432-1:

SELECT Группы.Номер_группы, Группы.ФИО_куратора, Студенты.ФИО_студента

FROM Группы, Студенты

WHERE Группы.Номер_группы = Студенты.Номер_группы AND

На рис. 1 приведен пример выполнения данного запроса.

Таблицы БД

СТУДЕНТЫ

Номер_зачетной книжки	ФИО_студента	Дата рождения	Место рождения	Номер_группы
1992412-11	Карасев А.А.	27.08.75	г. Чита	412-1
1992432-11	Данилов О. В.	27.08.75	г. Алматы	432-1
1992432-12	Раевский А. И.	20.05.75	г. Бишкек	432-1
1992432-22	Глазов О.А	04.07.75	г. Киров	432-1

ГРУППЫ

Номер_Группы	ФИО_куратора
412-1	Самойлов С.С.
432-1	Авдеев Р.М

Результат выполнения запроса

Номер_группы	ФИО_куратора	ФИО_студента
432-1	Авдеев Р.М	Данилов О. В.
432-1	Авдеев Р.М	Раевский А. И.
432-1	Авдеев Р.М	Глазов О.А

Рис. 1. Пример выполнения запроса на выборку

Помимо обычных знаков сравнения (=,<,>,<=,>=, <>) в языке SQL в условии отбора используются ряд ключевых слов:

Is not null — выбрать только непустые значения;

Is null — выбрать только пустые значения;

Between … And определяет принадлежность значения выражения указанному диапазону.

Синтаксис:

выражение [Not] Between значение_1 And значение_2 ,

где выражение — выражение, определяющее поле, значение которого проверяется на принадлежность к диапазону;

значение_1, значение_2 – выражения, задающие границы диапазона.

Если значение поля, определенного в аргументе выражение, попадает в диапазон, задаваемый аргументами значение_1 и значение_2 (включительно), то оператор Between...And возвращает значение True; в противном случае возвращается значение False. Логический оператор Not позволяет проверить противоположное условие: что выражение находится за пределами диапазона, заданного с помощью аргументов значение_1 и значение_2.

Оператор Between...And часто используют для проверки: попадает ли значение поля в указанный диапазон чисел. В следующем примере выдается список студентов, получающих стипендию от 800 до 900 рублей:

```
SELECT ФИО_студента, Размер_стипендии
FROM Студенты
WHERE Размер_стипендии Between 800 And 900
```

На рис. 2 приведен результат выполнения запроса.

СТУДЕНТЫ

Номер_зачетной книжки	ФИО_студента	Размер_стипендии
1992412-11	Карасев А.А.	900
1992432-11	Данилов О. В.	800
1992432-12	Раевский А. И.	950
1992432-22	Глазов О.А	850

Результирующий набор данных

ФИО_студента	Размер_стипендии
Карасев А.А.	900
Данилов О. В.	800
Глазов О.А	850

Рис. 2. Результат выполнения запроса на выборку с использованием операторов Between...And

Если выражение, значение_1 или значение_2 имеет значение Null, оператор Between...And возвращает значение Null.

Оператор Like используется для сравнения строкового выражения с образцом.

Синтаксис:

выражение Like «образец»,

где выражение — выражение SQL, используемое в предложении WHERE; образец — строка, с которой сравнивается выражение.

Оператор Like используется для нахождения в поле значений, соответствующих указанному образцу. Для аргумента образец можно задавать полное значение (например, Like «Иванов») или использовать подстановочные знаки для поиска диапазона значений (например, Like "Ив*").

Приведем перечень подстановочных символов и пример их использования в языке Jet SQL согласно документации Microsoft.

Параметры оператора Like

Тип совпадения	Образец	Совпадение (True)	Несовпадение (False)
Несколько символов	a*a	aa, aBa, aBBBa	aBC
	ab	abc, AABB, Xab	aZb, bac
Специальный символ	a[*]a	a*a	aaa
Несколько символов	ab*	abcdefg, abc	cab, aab
Одиночный символ	a?a	aaa, a3a, aBa	aBBBa
Одиночная цифра	a#a	a0a, a1a, a2a	aaa, a10a
Диапазон символов	[a-z]	f, p, j	2, &
Вне диапазона	[!a-z]	9, &, %	b, a
Не цифра	[!0-9]	A, a, &, ~	0, 1, 9
Комбинированное выражение	a[!b-m]#	An9, az0, a99	abc, aj0

Внутреннее соединение

Операция **INNER JOIN** объединяет записи из двух таблиц, если связующие поля этих таблиц содержат одинаковые значения.

Синтаксис операции:

FROM таблица_1 INNER JOIN таблица_2 ON таблица_1.поле_1 оператор таблица_2.поле_2

где таблица_1, таблица_2 — имена таблиц, записи которых подлежат объединению;

поле_1, поле_2 — имена объединяемых полей. Поля должны иметь одинаковый тип данных и содержать данные одного рода, однако эти поля могут иметь разные имена;

оператор — любой оператор сравнения: "=," "<," ">," "<=," ">=," или "<>".

Операцию INNER JOIN можно использовать в любом предложении FROM. Это самые обычные типы связывания. Они объединяют записи двух таблиц, если связующие поля обеих таблиц содержат одинаковые значения. Предыдущий пример использования команды SELECT можно записать с использованием конструкции INNER JOIN следующим образом:

```
SELECT Группы.Номер_группы, Группы.ФИО_куратора, Студенты.ФИО_студента
FROM Группы INNER JOIN Студенты
ON Группы.Номер_группы = Студенты.Номер_группы;
```

Внешнее соединение

Операции **LEFT JOIN, RIGHT JOIN** объединяют записи исходных таблиц при использовании в любом предложении FROM.

Операция LEFT JOIN используется для создания внешнего соединения, при котором все записи из первой (левой) таблицы включаются в результирующий набор, даже если во второй (правой) таблице нет соответствующих им записей.

Операция RIGHT JOIN используется для создания внешнего объединения, при котором все записи из второй (правой) таблицы включаются в результирующий набор, даже если в первой (левой) таблице нет соответствующих им записей.

Синтаксис операции:

```
FROM таблица_1 [ LEFT | RIGHT ] JOIN таблица_2
ON таблица_1.поле_1 оператор таблица_2.поле_2
```

Например, операцию LEFT JOIN можно использовать с таблицами «Студенты» (левая) и «Задолженность_за_обучение» (правая) для отбора всех студентов, в том числе тех, которые не являются задолжниками:

```
SELECT Студенты.ФИО_студента,
Задолженность_за_обучение.Сумма_задолженности
FROM Студенты LEFT JOIN Задолженность_за_обучение
ON Студенты.Номер_зачетной_книжки =
 Задолженность_за_обучение.Номер_зачетной_книжки;
```

Поле «Номер_зачетной_книжки» в этом примере используется для объединения таблиц, однако, оно не включается в результат выполнения запроса, поскольку не включено в инструкцию SELECT.

Чтобы включить связующее поле (в данном случае поле «Номер_зачетной_книжки») в результат выполнения запроса, его имя необходимо включить в инструкцию SELECT.

Важно отметить, что операции LEFT JOIN или RIGHT JOIN могут быть вложены в операцию INNER JOIN, но операция INNER JOIN не может быть вложена в операцию LEFT JOIN или RIGHT JOIN.

Перекрестные запросы

В некоторых СУБД (в частности в MS Access) существует такой вид запросов как перекрестный. В перекрестном запросе отображаются результаты статистических функций — суммы, средние значения и др., а также количество записей. При этом подсчет выполняется по данным из одного полей таблицы. Результаты группируются по двум наборам данных, один из которых расположен в левом столбце таблицы, а другой в заголовке таблицы. Например, при необходимости вычислить средний балл студентов за семестр, обучающихся на разных кафедрах, необходимо реализовать перекрестный запрос, в результате выполнения которого будет создана таблица, где заголовками строк будут служить номер семестра, заголовками столбцов — названия кафедр, а в полях таблицы будет рассчитан средний балл.

Для создания перекрестного запроса необходимо использовать следующую инструкцию:

TRANSFORM статистическая_функция

инструкция_SELECT

PIVOT поле [IN (значение_1[, значение_2[, ...]])],

где статистическая_функция — статистическая функция SQL, обрабатывающая указанные данные;

инструкция_SELECT — запрос на выборку;

поле — поле или выражение, которое содержит заголовки столбцов для результирующего набора;

значение_1, значение_2 — фиксированные значения, используемые при создании заголовков столбцов.

Составим SQL-запрос, реализующий описанный выше пример. В качестве исходного набора данных используется таблица Успеваемость (рис. 3).

Успеваемость : таблица

НОМЕР ЗАЧЕТНОЙ КНИЖКИ	Семестр	Оценка	Кафедра
102	1	3	АСУ
102	2	3	АСУ
102	3	4	АСУ
102	4	4	АСУ
110	1	5	АОИ
110	2	3	АОИ
110	3	3	АОИ
110	4	4	АОИ
110	5	4	АОИ

Рис. 3. Таблица УСПЕВАЕМОСТЬ

В результате выполнения нижеприведенного перекрестного SQL-запроса формируется следующая таблица (рис. 4):

```
TRANSFORM AVG(Успеваемость.Оценка) AS Сред_балл
SELECT Успеваемость.Семестр
FROM Успеваемость
GROUP BY Успеваемость.Семестр
PIVOT Успеваемость.Кафедра
```

SQL_запрос : перекрестный запрос

Семестр	АОИ	АСУ
1	5	3
2	3	3
3	3	4
4	4	4
5	4	

Рис. 4. Результат выполнения перекрестного запроса

Таким образом, когда данные сгруппированы с помощью перекрестного запроса, можно выбирать значения из заданных столбцов или выражений и определять как заголовки столбцов. Это позволяет просматривать данные в более компактной форме, чем при работе с запросом на выборку.

Подчиненные запросы

Часто возникает ситуация, когда желаемый результат нельзя получить с помощью одного SQL-запроса. Одним из способов решения такой задачи является использование подчиненных запросов в составе главного SQL-запроса. Подчиненным SQL-запросом называют инструкцию SELECT, включаемую в инструкции SELECT, SELECT...INTO, INSERT...INTO, DELETE или UPDATE или в другой подчиненный запрос. Подчиненный запрос может быть создан одним из трех способов, синтаксис которых представлен ниже:

1) сравнение [ANY | ALL | SOME] (инструкцияSQL)

2) выражение [NOT] IN (инструкцияSQL)

3) [NOT] EXISTS (инструкцияSQL),

где сравнение — выражение и оператор сравнения, который сравнивает выражение с результатами подчиненного запроса;

выражение — выражение, для которого проводится поиск в результирующем наборе записей подчиненного запроса;

инструкцияSQL — инструкция SELECT, заключенная круглые скобки.

Подчиненный запрос можно использовать вместо выражения в списке полей инструкции SELECT или в предложениях WHERE и HAVING. Инструкция SELECT используется в подчиненном запросе для задания набора конкретных значений, вычисляемых в выражениях предложений WHERE или HAVING.

Предикаты ANY или SOME, являющиеся синонимами, используются для отбора записей в главном запросе, которые удовлетворяют сравнению с записями, отобранными в подчиненном запросе. В следующем примере отбираются все студенты, средний балл которых за семестр больше 4.

```
SELECT * FROM Студенты
WHERE Номер_зачетной_книжки = ANY
(SELECT Номер_зачетной_книжки FROM Успеваемость
WHERE оценка > 4)
```

Предикат ALL используется для отбора в главном запросе только тех записей, которые удовлетворяют сравнению со всеми записями, отобранными в подчиненном запросе. Если в предыдущем примере предикат ANY заменить предикатом ALL, результат запроса будет включать только тех студентов, у которых средний балл больше 4. Это условие является значительно более жестким.

Предикат IN используется для отбора в главном запросе только тех записей, которые содержат значения, совпадающие с одним из отобранных подчиненным запросом. Следующий пример возвращает сведения обо всех студентах, средний балл которых за семестр был больше 4.

```
SELECT * FROM Студенты
WHERE Номер_зачетной_книжки in
(SELECT Номер_зачетной_книжки FROM Успеваемость
WHERE оценка > 4)
```

Предикат NOT IN используется для отбора в главном запрос только тех записей, которые содержат значения, не совпадающие ни одним из отобранных подчиненным запросом.

Предикат EXISTS (с необязательным зарезервированным слово NOT) используется в логическом выражении для определения тог должен ли подчиненный запрос возвращать какие-либо записи.

В подчиненном запросе можно использовать псевдонимы табли для ссылки на таблицы, перечисленные в предложении FROM, распс ложенном вне подчиненного запроса. В следующем примере отбир ются фамилии и имена студентов, чья стипендия равна или больш средней стипендии студентов, обучающихся в той же группе. В дан ном примере таблица СТУДЕНТЫ получает псевдоним С1:

```
SELECT Фамилия,
Имя, Номер_группы, Стипендия
FROM СТУДЕНТЫ AS C1
WHERE Стипендия >=
(SELECT Avg(Стипендия)
FROM СТУДЕНТЫ
WHERE C1.Номер_группы = СТУДЕНТЫ.Номер_группы) Orde
by Номер_группы;
```

В последнем примере зарезервированное слово AS не являетс обязательным.

Некоторые подчиненные запросы можно использовать в перекре стных запросах как предикаты (в предложении WHERE). Подчинен ные запросы, используемые для вывода результатов (в списк SELECT), нельзя использовать в перекрестных запросах.

Создание новой таблицы

Инструкция **CREATE TABLE** создает новую таблицу и исполь зуется для описания ее полей и индексов. Если для поля добавлено ог раничение NOT NULL, то при добавлении новых записей это пол должно содержать допустимые данные. Синтаксис:

```
CREATE TABLE таблица (поле_1 тип [(размер)]
[NOT NULL] [индекс_1] [, поле_2 тип [(размер)]
[NOT NULL] [индекс_2] [, ...]] [, CONSTRAINT составной Ин
декс [, ...]]),
```

где таблица — имя создаваемой таблицы;

поле_1, поле_2 — имена одного или нескольких полей, создаваемых в новой таблице. Таблица должна содержать хотя бы одно поле;

тип — тип данных поля в новой таблице;

размер — размер поля в символах (только для текстовых и двоичных полей);

индекс_1, индекс_2 — предложение CONSTRAINT, предназначенное для создания простого индекса;

составной Индекс — предложение CONSTRAINT, предназначенное для создания составного индекса.

следующем примере создается новая таблица с двумя полями:

CREATE TABLE Студенты (Номер_зачетной_книжки integer PRIMARY KEY, ФИО_студента TEXT (50), Место_рождения TEXT (50));

В результате выполнения этого запроса будет создана таблица со следующей схемой (рис. 5):

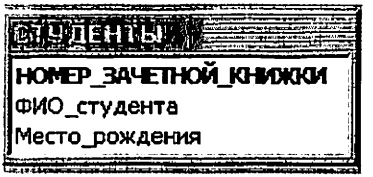

Рис. 5. Схема таблицы СТУДЕНТЫ

Предложение ***CONSTRAINT*** используется в инструкциях ALTE TABLE и CREATE TABLE для создания или удаления индексов. С ществуют два типа предложений CONSTRAINT: для создания прост го индекса (по одному полю) и для создания составного индекса (г нескольким полям). Синтаксис:

простой индекс:

CONSTRAINT имя {PRIMARY KEY|UNIQUE | NOT NULL]}

составной индекс:

CONSTRAINT имя

{PRIMARY KEY (ключевое_1[, ключевое_2 [, ...]]) |

UNIQUE (уникальное_1[, уникальное_2 [, ...]]) |

NOT NULL (непустое_1[, непустое_2 [, ...]]) |

FOREIGN KEY (ссылка_1[, ссылка_2 [, ...]])

REFERENCES внешняя Таблица [(внешнее Поле_1 [, внешн Поле_2 [, ...]])]},

где имя — имя индекса, который следует создать;

ключевое_1, ключевое_2 — имена одного или нескольких полей, которые следует назначить ключевыми;

уникальное_1, уникальное_2 — имена одного или нескольких полей, которые следует включить в уникальный индекс;

непустое_1, непустое_2 — имена одного или нескольких полей, в которых запрещаются значения Null;

ссылка_1, ссылка_2 — имена одного или нескольких полей, включенных во внешний ключ, которые содержат ссылки на поля в другой таблице;

внешняя Таблица — имя внешней таблицы, которая содержит поля, указанные с помощью аргумента внешнееПоле;

внешнее Поле_1, внешнее Поле_2 — имена одного или нескольких полей во внешней Таблице, на которые ссылаются поля, указанные с помощью аргумента ссылка_1, ссылка_2. Это предложение можно опустить, если данное поле является ключом внешней Таблицы.

Предложение CONSTRAINT позволяет создать для поля индекс одного из двух описанных ниже типов:

1) уникальный индекс, использующий для создания зарезервированное слово UNIQUE. Это означает, что в таблице не может быть двух записей, имеющих одно и то же значение в этом поле. Уникальный индекс создается для любого поля или любой группы полей. Если в таблице определен составной уникальный индекс, то комбинация значений включенных в него полей должна быть уникальной для каждой записи таблицы, хотя отдельные поля и могут иметь совпадающие значения;

2) ключ таблицы, состоящий из одного или нескольких полей, использующий зарезервированные слова PRIMARY KEY. Все значения ключа таблицы должны быть уникальными и не значениями Null. Кроме того, в таблице может быть только один ключ.

Для создания внешнего ключа можно использовать зарезервированную конструкцию FOREIGN KEY. Если ключ внешней таблицы состоит из нескольких полей, необходимо использовать предложение CONSTRAINT. При этом следует перечислить все поля, содержащие ссылки на поля во внешней таблице, а также указать имя внешней таблицы и имена полей внешней таблицы, на которые ссылаются поля, перечисленные выше, причем в том же порядке. Однако, если последние поля являются ключом внешней таблицы, то указывать их необя-

зательно, поскольку ядро базы данных считает, что в качестве эти полей следует использовать поля, составляющие ключ внешней табли цы.

В следующем примере создается таблица ЗАДОЛЖЕН НОСТЬ_ЗА_ОБУЧЕНИЕ с единственным полем НО МЕР_ЗАЧЕТНОЙ_КНИЖКИ и внешним ключом fl_i, связанным с по лем НОМЕР_ЗАЧЕТНОЙ_КНИЖКИ в таблице СТУДЕНТЫ:

CREATE TABLE Задолженность_за_обучение

(Код_задолженности integer PRIMARY KEY, Но мер_зачетной_книжки integer, CONSTRAINT fl_i FOREIGN KE (Номер_зачетной_книжки) REFERENCES Студенты (Но мер_зачетной_книжки));

Внешний вид схемы БД, состоящей из таблиц СТУДЕНТЫ и ЗА ДОЛЖЕННОСТЬ_ЗА_ОБУЧЕНИЕ, представлен на рис. 6.

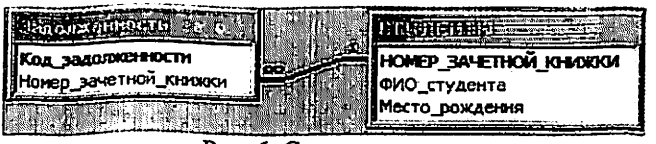

Рис. 6 Схема данных

Изменение структуры таблицы

Инструкция **ALTER TABLE** изменяет структуру таблицы, созданной с помощью инструкции CREATE TABLE.

Синтаксис:

ALTER TABLE таблица {ADD {COLUMN поле тип[(размер)] [NOT NULL]

[CONSTRAINT индекс] | CONSTRAINT составной Индекс} |

DROP {COLUMN поле I CONSTRAINT имя Индекса} }

где таблица — имя изменяемой таблицы;

поле — имя поля, добавляемого в таблицу или удаляемого из нее;

тип — тип данных поля;

размер — размер поля;

индекс — индекс для поля;

составной Индекс — описание составного индекса, добавляемого к таблице;

имя Индекса — имя составного индекса, который следует удалить.

С помощью инструкции ALTER TABLE существующую таблицу можно изменить несколькими способами:

1) добавить новое поле в таблицу с помощью предложения ADD COLUMN. В этом случае необходимо указать имя поля, его тип и размер. Например, следующая инструкция добавляет в таблицу СТУДЕНТЫ текстовое поле ПРИМЕЧАНИЯ длиной 50 символов:

ALTER TABLE Студенты ADD COLUMN Примечания TEXT(50)

Если для поля добавлено ограничение NOT NULL, то при добавлении новых записей это поле должно содержать допустимые данные;

2) добавить составной индекс с помощью зарезервированных слов ADD CONSTRAINT;

3) удалить поле с помощью зарезервированных слов DROP COLUMN. В этом случае необходимо указать только имя поля;

4) удалить составной индекс с помощью зарезервированных слов DROP CONSTRAINT. В этом случае указывается только имя составного индекса, следующее за зарезервированным словом CONSTRAINT.

Создание индекса с помощью инструкции CREATE INDEX

CREATE INDEX создает новый индекс для существующей таблицы. Синтаксис команды:

CREATE [UNIQUE] INDEX индекс
ON таблица (поле [ASC|DESC][, поле [ASC|DESC], ...])
[WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]

где индекс — имя создаваемого индекса;

таблица — имя существующей таблицы, для которой создается индекс;

поле — имена одного или нескольких полей, включаемых в индекс. Для создания простого индекса, состоящего из одного поля, вводится имя поля в круглых скобках сразу после имени таблицы. Для создания составного индекса, состоящего из нескольких полей, перечисляются имена всех этих полей. Для расположения элементов индекса в убывающем порядке используется зарезервированное слово DESC; в противном случае будет принят порядок по возрастанию.

Чтобы запретить совпадение значений индексированных полей в разных записях, используется зарезервированное слово UNIQUE. Необязательное предложение WITH позволяет задать условия на значения. Например:

– с помощью параметра DISALLOW NULL можно запретить значения Null в индексированных полях новых записей;

– параметр IGNORE NULL позволяет запретить включение в индекс записей, имеющих значения Null в индексированных полях;

– зарезервированное слово PRIMARY позволяет назначить индексированные поля ключом. Такой индекс по умолчанию является уникальным, следовательно, зарезервированное слово UNIQUE можно опустить.

Удаление таблицы/индекса

Инструкция **DROP** удаляет существующую таблицу из базы данных или удаляет существующий индекс из таблицы. Синтаксис:

DROP {TABLE таблица | INDEX индекс ON таблица}

где таблица — имя таблицы, которую следует удалить или из которой следует удалить индекс;

индекс — имя индекса, удаляемого из таблицы.

Прежде чем удалить таблицу или удалить из нее индекс, необходимо ее закрыть. Следует отметить, что таблица удаляется из базы данных безвозвратно.

Удаление записей

Инструкция **DELETE** создает запрос на удаление записей из одной или нескольких таблиц, перечисленных в предложении FROM и удовлетворяющих предложению WHERE.

Синтаксис команды:

DELETE [Таблица.*]

FROM таблица

WHERE условие Отбора

где Таблица — необязательное имя таблицы, из которой удаляются записи;

таблица — имя таблицы, из которой удаляются записи;

условие Отбора — выражение, определяющее удаляемые записи.

С помощью инструкция DELETE можно осуществлять удаление большого количества записей. Данные из таблицы также можно удалить и с помощью инструкции DROP, однако при таком удалении теряется структура таблицы. Если же применить инструкцию DELETE, удаляются только данные. При этом сохраняются структура таблицы и все остальные ее свойства, такие, как атрибуты полей и индексы.

Запрос на удаление удаляет записи целиком, а не только содержимое указанных полей. Нельзя восстановить записи, удаленные с помощью запроса на удаление. Чтобы узнать, какие записи будут удалены, необходимо посмотреть результаты запроса на выборку, использующе-

го те же самые условие отбора в предложении Where, а затем выполнить запрос на удаление.

Добавление записей

Инструкция **INSERT INTO** добавляет запись или записи в таблицу.

Синтаксис команды:

а) запрос на добавление нескольких записей:

INSERT INTO назначение [(поле_1[, поле_2[, ...]])]

SELECT [источник.]поле_1[, поле_2[, ...]

FROM выражение

б) запрос на добавление одной записи:

INSERT INTO назначение [(поле_1[, поле_2[, ...]])]

VALUES (значение_1[, значение_2[, ...])

где назначение — имя таблицы или запроса, в который добавляются записи;

источник — имя таблицы или запроса, откуда копируются записи;

поле_1, поле_2 — имена полей для добавления данных, если они следуют за аргументом «Назначение»; имена полей, из которых берутся данные, если они следуют за аргументом источник;

выражение — имена таблицы или таблиц, откуда вставляются данные. Это выражение может быть именем отдельной таблицы или результатом операции INNER JOIN, LEFT JOIN или RIGHT JOIN, а также сохраненным запросом;

значение_1, значение_2 — значения, добавляемые в указанные поля новой записи. Каждое значение будет вставлено в поле, занимающее то же положение в списке: значение_1 вставляется в поле_1 в новой записи, значение_2 — в поле_2 и т.д. Каждое значение текстового поля следует заключать в кавычки (' '), для разделения значений используются запятые.

Инструкцию INSERT INTO можно использовать для добавления одной записи в таблицу с помощью запроса на добавление одной записи, описанного выше. В этом случае инструкция должна содержать имя и значение каждого поля записи. Нужно определить все поля записи, в которые будет помещено значение, и значения для этих полей. Если поля не определены, в недостающие столбцы будет вставлено значение по умолчанию или значение Null. Записи добавляются в конец таблицы.

Инструкцию INSERT INTO можно также использовать для добавления набора записей из другой таблицы или запроса с помощью предложения SELECT ... FROM, как показано выше в запросе на до-

бавление нескольких записей. В этом случае предложение SELECT определяет поля, добавляемые в указанную таблицу НАЗНАЧЕНИЕ. Инструкция INSERT INTO является необязательной, однако, если она присутствует, то должна находиться перед инструкцией SELECT.

Запрос на добавление записей копирует записи из одной или нескольких таблиц в другую таблицу. Таблицы, которые содержат добавляемые записи, не изменяются.

Вместо добавления существующих записей из другой таблицы, можно указать значения полей одной новой записи с помощью предложения VALUES. Если список полей опущен, предложение VALUES должно содержать значение для каждого поля таблицы; в противном случае инструкция INSERT не будет выполнена. Можно использовать дополнительную инструкцию INSERT INTO с предложением VALUES для каждой добавляемой новой записи.

Обновление данных

Инструкция **UPDATE** создает запрос на обновление, который изменяет значения полей указанной таблицы на основе заданного условия отбора.

Синтаксис команды:

UPDATE таблица

SET новое Значение

WHERE условие Отбора;

где таблица — имя таблицы, данные в которой следует изменить;

новое Значение — выражение, определяющее значение, которое должно быть вставлено в указанное поле обновленных записей;

условие Отбора — выражение, отбирающее записи, которые должны быть изменены.

При выполнении этой инструкции будут изменены только записи, удовлетворяющие указанному условию. Инструкцию UPDATE особенно удобно использовать для изменения сразу нескольких записей или в том случае, если записи, подлежащие изменению, находятся в разных таблицах. Одновременно можно изменить значения нескольких полей. Следующая инструкция SQL увеличивает стипендию студентов группы 422-1 на 10 %:

```
UPDATE Студенты
SET Стипендия = стипендия * 1.1
WHERE Номер_группы = '422-1';
```

Запрос на объединение

Операция **UNION** создает запрос на объединение, который объе диняет результаты нескольких независимых запросов или таблиц.

Синтаксис команды:

[TABLE] запрос_1 UNION [ALL] [TABLE] запрос_2 [UN ION[ALL] [TABLE] запрос_n [...]]

где запрос_1-n — инструкция SELECT или имя сохраненной таблицы перед которым стоит зарезервированное слово TABLE.

В одной операции UNION можно объединить в любом наборе ре зультаты нескольких запросов, таблиц и инструкций SELECT. В сле дующем примере объединяется существующая таблица СТУДЕНТЫ инструкции SELECT:

```
TABLE Студенты UNION ALL
SELECT *
FROM Абитуриенты
WHERE Общий_балл > 22;
```

По умолчанию повторяющиеся записи не возвращаются при ис пользовании операции UNION, однако в нее можно добавить предика ALL, чтобы гарантировать возврат всех записей. Кроме того, таки запросы выполняются несколько быстрее.

Все запросы, включенные в операцию UNION, должны отбират одинаковое число полей; при этом типы данных и размеры полей н обязаны совпадать. Псевдонимы необходимо использовать только первом предложении SELECT, в остальных они пропускаются.

В каждом аргументе «Запрос» допускается использование пред ложения GROUP BY или HAVING для группировки возвращаемы данных. В конец последнего аргумента «Запрос» можно включит предложение ORDER BY, чтобы отсортировать возвращенные данные.

Основные различия Microsoft Jet SQL и ANSI SQL

Рассмотрим различия двух диалектов языка SQL Microsoft Jet SQL и ANSI SQL, описанные в руководстве разработчика приложений баз данных на СУБД Microsoft Access:

– языки SQL-ядра базы данных Microsoft Jet и ANSI SQL имеют разные наборы зарезервированных слов и типов данных;

– разные правила применимы к оператору Between...And, имею щему следующий синтаксис: выражение [NOT] Between значение_1 And значение_2. В языке SQL Microsoft Jet значение_1 может превы шать значение_2; в ANSI SQL, значение_1 должно быть меньше или равно значение_2;

– разные подстановочные знаки используются с оператором Like Так, в языке SQL Microsoft Jet любой одиночный символ изображаетс знаком «?», а в ANSI SQL знаком «_», любое число символов в язык SQL Microsoft Jet изображается знаком «*», а в ANSI SQL — знако «%»;

– язык SQL-ядра Microsoft Jet обычно предоставляет пользовате лю большую свободу. Например, разрешается группировка и сорти ровка по выражениям.

Особые средства языка SQL Microsoft Jet

В языке SQL Microsoft Jet поддерживаются следующие дополни тельные средства:

– инструкция TRANSFORM, предназначенная для создания пере крестных запросов;

– дополнительные статистические функции, такие как StDev и VarP;

– описание PARAMETERS, предназначенное для создания запро сов с параметрами.

Средства ANSI SQL, не поддерживаемые в языке SQL Microsoft Jet

В языке SQL Microsoft Jet не поддерживаются следующие сред ства ANSI SQL:

– инструкции, управляющие защитой, такие как COMMIT GRANT и LOCK;

– зарезервированное слово DISTINCT в качестве описания аргу мента статистической функции (например, нельзя использовать выра жение SUM(DISTINCT имя Столбца);

– предложение LIMIT TO nn ROWS, используемое для ограниче ния количества строк, возвращаемых в результате выполнения запро са. Для ограничения количества возвращаемых запросом строк можн использовать только предложение WHERE.

Порядок выполнения работы

Все запросы, создаваемые в рамках данной лабораторной работы должны быть реализованы на языке SQL без помощи построителя за просов.

Для создания нового SQL-запроса в окне базы данных нажмит кнопку Создание запроса в режиме конструктора и не выбирая табли цы для запроса нажмите кнопку  на панели инструментов.

1. Используя инструкцию CREATE TABLE, создайте запрос на соз дание новой таблицы для выбранной ранее предметной области

содержащую пять полей, различных типов данных, определив в запросе первичный ключ и проиндексировав соответствующие поля, используя предложение CONSTRAINT. Для запуска запроса нажмите кнопку Запу  на панели инструментов. После чего создайте запрос на создание еще одной таблицы, содержащей внешний ключ по отношению к первичному ключу предыдущей таблицы. Запустите запрос, после чего проверьте, отразились ли изменения в схеме данных.

2. Используя команду SELECT, создайте запрос на выборку записей из двух (или более) таблиц, используя правила внешнего и внутреннего соединения, а также различные условия отбора и сортировки.
3. Используя команду UPDATE, создайте запрос на обновление данных в созданных ранее таблицах.
4. Используя команду INSERT INTO, создайте запросы на добавление группы записей и одной записи в существующую таблицу.
5. Используя команду CREATE INDEX, создайте запрос на создание нового индекса, используя различные условия на значения индексов (IGNORE NULL, DISSALLOW NULL, PRIMARY), а также типы сортировки.
6. Используя команду DROP, создайте запросы на удаление таблицы и индекса, созданных ранее в БД.
7. Сохраните все созданные запросы в базе данных.

Лабораторная работа № 2 «Создание концептуальной модели данных в среде автоматизированного проектирования»

Тема: Создание концептуальной модели данных в среде автоматизированного проектирования.

Раздел дисциплины: Моделирование данных с помощью ER-диаграмм.

Цель работы: спроектировать концептуальную модель, выбранной ранее предметной области в пакете Power Designer.

Продолжительность: 6 часов.

Сущности и атрибуты

Каждая сущность является множеством подобных индивидуальных объектов, называемых экземплярами. Каждый экземпляр индивидуален и должен отличаться от всех остальных экземпляров. Атрибут выражает определенное свойство объекта. С точки зрения БД (физическая модель) сущности соответствует таблица, экземпляру сущности – строка в таблице, а атрибуту – колонка таблицы.

Построение модели данных предполагает определение сущностей и атрибутов, т.е. необходимо определить, какая информация будет храниться в конкретной сущности или атрибуте. Сущность можно определить как объект, событие или концепцию, информация о которой должна сохраняться. Сущности должны иметь наименование с четким смысловым значением, именоваться существительным в единственном лице, не носить «технических» наименований и быть достаточно важными для того, чтобы их моделировать. Именование сущности в единственном числе облегчает в дальнейшем чтение модели. Фактически имя сущности дается по имени ее экземпляра.

Каждая сущность должна быть полностью определена с помощью текстового описания. Каждый атрибут хранит информацию об определенном свойстве сущности, а каждый экземпляр сущности должен быть уникальным. Атрибут или группа атрибутов, которые идентифицируют сущность, называется первичным ключом. При установлении связей между сущностями атрибуты первичного ключа родительской сущности мигрируют в качестве внешних ключей в дочернюю сущность.

Очень важно дать атрибуту правильное имя. Атрибуты должны именоваться в единственном числе и иметь четкое смысловое значение. Соблюдение этого правила позволяет частично решить проблему нормализации данных уже на этапе определения атрибутов.

Связи

Связь является логическим соотношением между сущностями. Каждая связь должна именоваться глаголом или глагольной фразой. Имя связи выражает некоторое ограничение или бизнес-правило и облегчает чтение построенной модели данных.

Различают зависимые и независимые сущности. Тип сущности определяется ее связью с другими сущностями. Идентифицирующая связь устанавливается между независимой (родительский конец связи) и зависимой (дочерний конец связи) сущностями. При установлении идентифицирующей связи атрибуты первичного ключа родительской сущности переносятся в состав первичного ключа дочерней сущности. Эта операция дополнения атрибутов дочерней сущности при создании связи называется миграцией атрибутов. В дочерней сущности атрибуты помечаются как внешний ключ (FK).

При установлении неидентифицирующей связи дочерняя сущность остается независимой, а атрибуты первичного ключа родительской сущности мигрируют в состав неключевых компонентов родительской сущности. Неидентифицирующая связь служит для связывания независимых сущностей.

Имя связи – фраза, характеризующая отношение между родительской и дочерней сущностями. Для связи один-ко-многим идентифицирующей или не идентифицирующей достаточно указать имя, характеризующее отношение от родительской к дочерней сущности.

Тип связи (идентифицирующая/неидентифицирующая). Для неидентифицирующей связи можно указать обязательность. В случае обязательной связи атрибут внешнего ключа получит признак NOT NULL, несмотря на то, что внешний ключ не войдет в состав первичного ключа дочерней сущности. В случае необязательной связи внешний ключ может принимать значение NULL. Необязательная неидентифицирующая связь помечается прозрачным ромбиком со стороны родительской сущности.

Правила ссылочной целостности – логические конструкции, которые выражают бизнес-правила использования данных и представляют собой правила вставки, замены и удаления.

Информацию о предметной области суммируют составлением спецификаций по сущностям, атрибутам и отношениям с использованием графических диаграмм, в чем и заключается процесс моделирование данных.

Основы работы в Power Designer

Для запуска пакета Power Designer в меню программы (Windows) найдите папку Sybase и запустите файл Power Designer. Для создания концептуальной модели данных необходимо выбрать File/ New или на панели инструментов выбрать знак . Далее появится окно для выбора создаваемой модели (рис. 7), в котором надо выбрать Conceptual Data Model

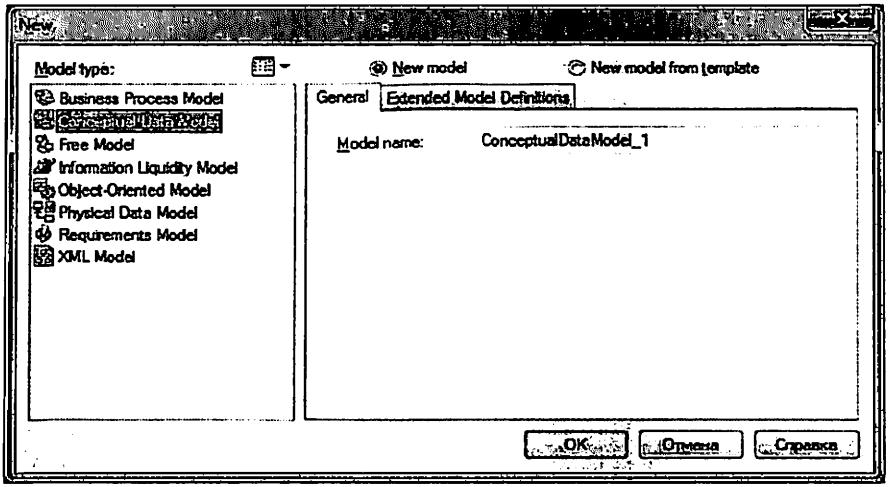

Рис. 7. Окно выбора модели.

После нажатия кнопки ОК появиться окно, в котором создается ER-диаграмма.

Создание сущностей

Для создания сущности, в панели Palette (рис. 8) нажмите кнопку с белым прямоугольником (с подсказкой Entity).

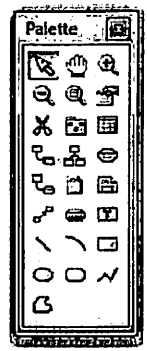

Рис. 8. Панель элементов с выбранным элементом сущность

Далее, поместите указатель мыши на рабочее поле в нужном месте и щелкните кнопкой мыши. Прямоугольник, изображающий сущность появится в указанном месте. При этом, курсор мыши на рабочем поле выглядит как выбранный элемент, т.о. можно создавать несколько выбранных элементов одного типа без повторного их выбора на панели элементов.

Для того, чтобы изменить свойства созданной сущности, дважды щелкните на нее левой кнопкой мыши или нажмите правую кнопку и в выпавшем меню, выберите пункт Properties, в результате чего откроется окно свойств сущности (рис. 9) .

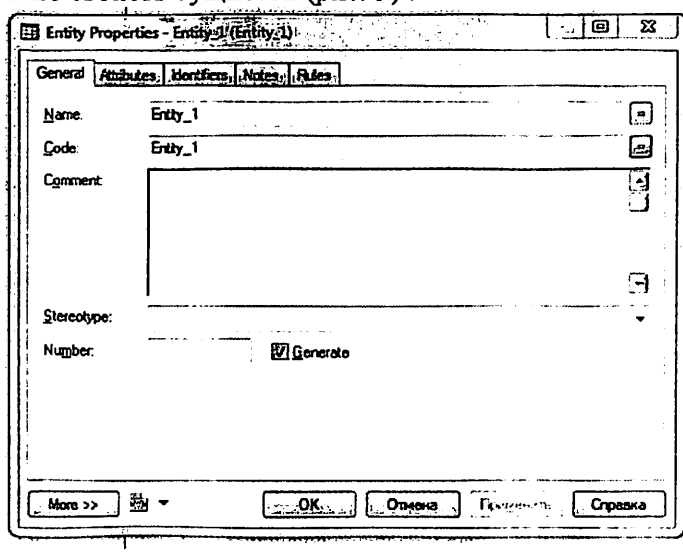

Рис. 9. Окно свойств сущности

В открывшемся окне пять закладок.

Закладка General позволяет ввести следующие основные параметры:

- Name — имя сущности, которое будет видеть пользователь;
- Code — имя кода сущности, которое будет использоваться при генерации физической модели;
- Number — ограничение количества записей в таблице после генерации физической модели;
- Comment — комментарий, предназначенный для улучшения понимания модели.

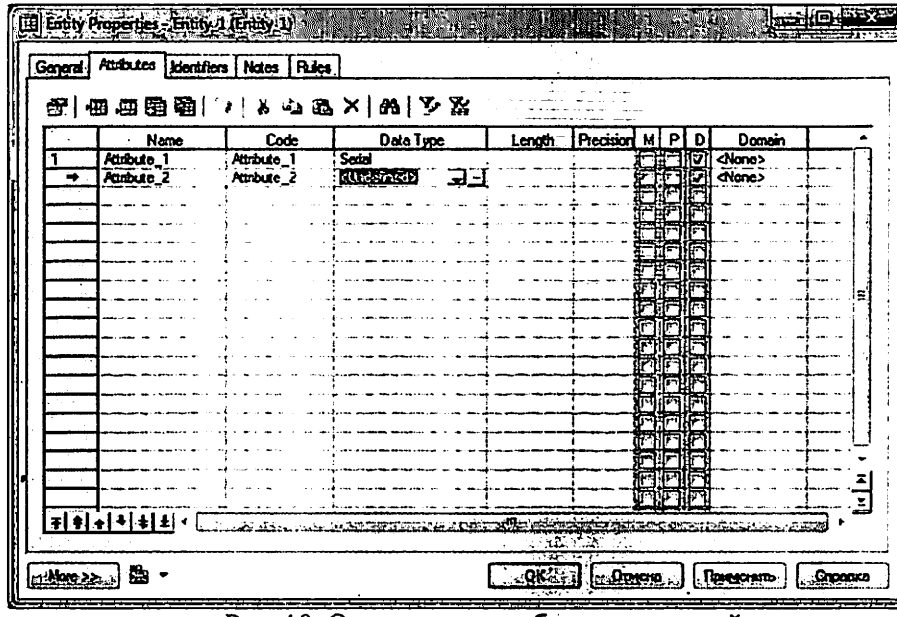

Рис. 10. Окно ввода атрибутов сущностей

Закладка Attributes содержит таблицу (рис. 10) и позволяет определять ***атрибуты*** сущности:

- Name — имя атрибута, которое будет видеть пользователь;
- Code — имя кода атрибута, которое будет использоваться при генерации физической модели;
- Data type — тип данных атрибута, который может быть выбран из выпадающего списка или вручную, щелкнув в поле Data type;

- Domain — принадлежность к домену, если он определен. Использование доменов позволяет, определив один раз пользовательский тип данных, использовать его в дальнейшем при определении типа данных атрибута. О создании домена будет сказано ниже;
- M (mandatory) — обязательный атрибут, указывает может ли данный атрибут принимать неопределенные значения (обязательно ли данное поле для заполнения в таблице БД);
- P (Primary Identifier) — первичный идентификатор сущности (в физической модели данных атрибут будет являться первичным ключом или его составной частью);
- D (Displayed) — отображаемый, т.е. будет ли атрибут показываться в модели.

Более полную информацию по свойствам атрибута можно получить, дважды щелкнув по полю, расположенному слева от поля с именем атрибута. Здесь можно вводить комментарий в поле Comment, задавать список значений для данного атрибута, определять верхние и нижние границы значений

Закладка Identifiers содержит автоматически заполняемую таблицу первичных идентификаторов сущности, но позволяет делать это вручную, когда необходимо создать суррогатный первичный идентификатор.

Закладка Rules позволяет вводить необходимые правила на ввод значений в таблицу.

Остальные закладки Notes, Version info носят описательный характер для улучшения понимания модели.

Для фиксации всех изменений в необходимо нажать кнопку Apply.

Домен

Домен – это множество допустимых значений атрибута определенного типа данных. Домен определяется заданием стандартного типа данных, к которому относятся элементы домена и заданием произвольного логического выражения, применяемому к этому типу данных.

Для создания домена необходимо в главном меню выбрав пункт Model, выбрать пункт Domains. На рис. 11 показано форма определения домена. Данная форма содержит аналогичные поля как для определения свойств атрибутов (в закладке Attributes в свойствах формы), а также дополнительные поля Length и Precision для описания длины и точности значений атрибутов.

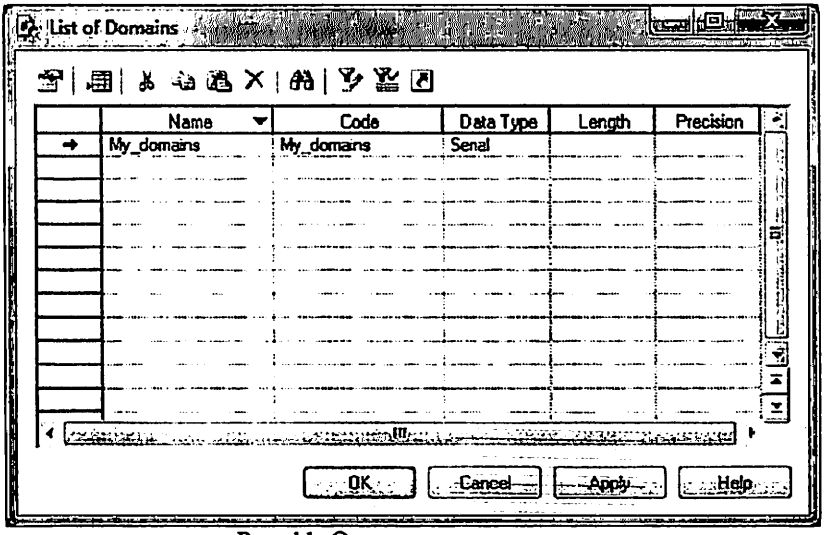

Рис. 11. Окно ввода списка доменов

После создания нового домена его имя появится в списке доменов при задании свойств атрибутов сущностей (рис. 12).

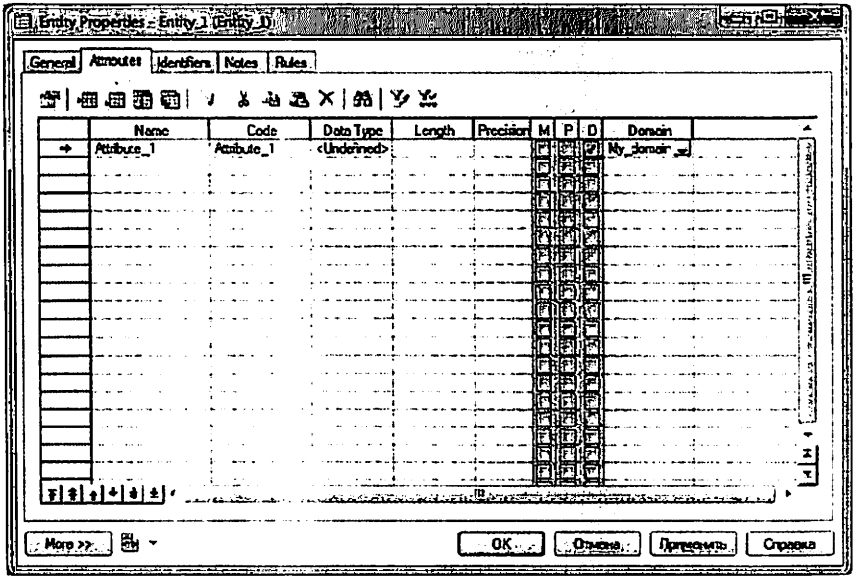

Рис. 12. Определение домена для атрибута сущности

При создании атрибутов сущностей в концептуальной модели не создаются атрибуты, являющиеся внешними ключами сущностей.

После того, как созданы все необходимые сущности и атрибуты, необходимо определить связи между ними.

Установка связей

Для установки связи между двумя сущностями, необходимо нажать кнопку (с двумя белыми прямоугольниками и линией между ними рис. 13).

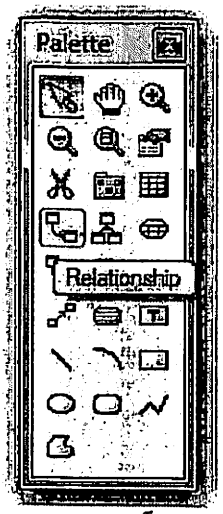

Рис. 13. Панель элементов с выбранным элементом связь

Необходимо перевести курсор мыши на одну сущность и, нажав левую кнопку мыши и, не отпуская ее, перевести курсор на вторую сущность. Далее можно отпустить кнопку мыши – связь установлена.

Для изменения свойств связи, необходимо дважды щелкнуть левой кнопкой мыши на линию связи (или нажать правую кнопку мыши и выпавшем меню выбрать пункт Properties). Откроется окно (рис. 14), с закладками:

- закладки Notes и Version info используются для подробного описания связи. В закладке Rules можно задавать параметры ограничения связи;
- в закладке General указывается имя и код связи, а две кнопки с именами используемых сущностей позволяют вызвать окно со свойствами соответствующей сущности.

- Закладка Cardinalites позволяет указать вид связи (один–к–одному, один–ко–многим, многие–ко–многим и т.д.) и устанавливает свойства связи от Сущности1 к Сущности2 и наоборот:
- Mandatory определяет обязательность связи, показывая, что экземпляр Сущности1 (запись) может существовать только при наличии соответствующего экземпляра в Сущности2;

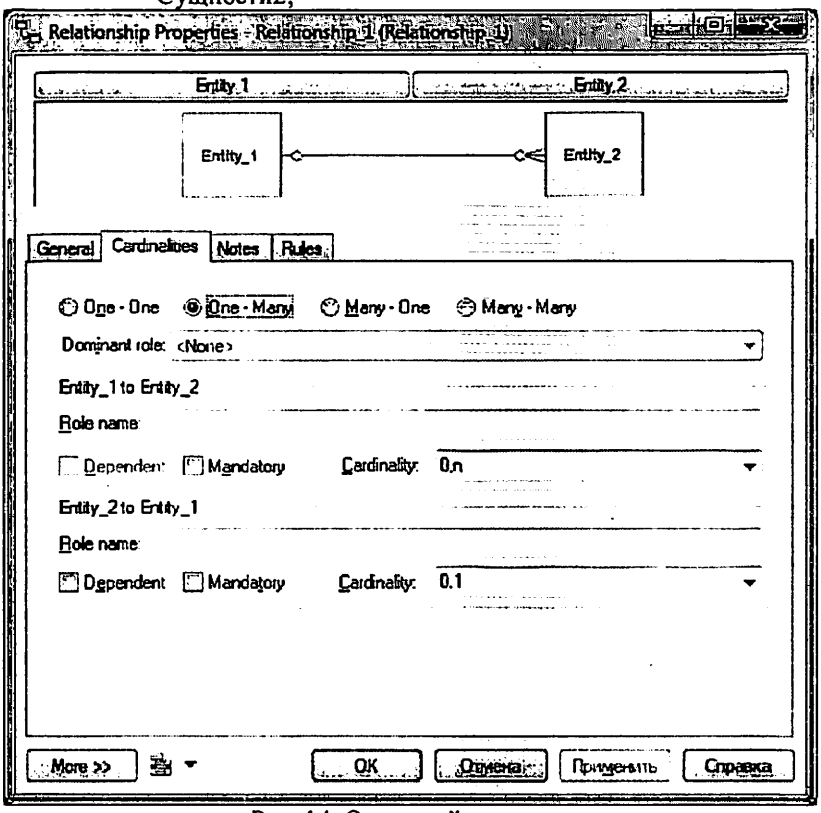

Рис. 14. Окно свойств связи

- Dependent показывает, что каждый экземпляр Сущности1 отождествляется с экземпляром в Сущности2 (первичный ключ на стороне «один» при создании физической модели войдет в состав первичного ключа в таблице на стороне «многие»);

- Role – текст, описывающий связь от Сущности1 к Сущности2.

Связи многие-ко-многим преобразуются в физической модели в промежуточные таблицы.

После того, как созданы все сущности, указаны атрибуты и установлены все связи необходимо проверить концептуальную правильность построения концептуальной модели. Для этого необходимо выбрать в меню Tools/Check Model (или нажать F4). Появится окно (рис. 15), в котором предлагается выбрать объекты для проверки.

Package – система проверит правильность циклических связей.

Domain – система проверит правильность заполнения доменов.

Data items – проверять ли атрибуты.

Entities – система проверит правильность создания сущностей.

Entity attributes – проверка правильности свойств сущности

Entity identifier - проверка правильности идентификаторов сущности.

Relationships – проверка связей.

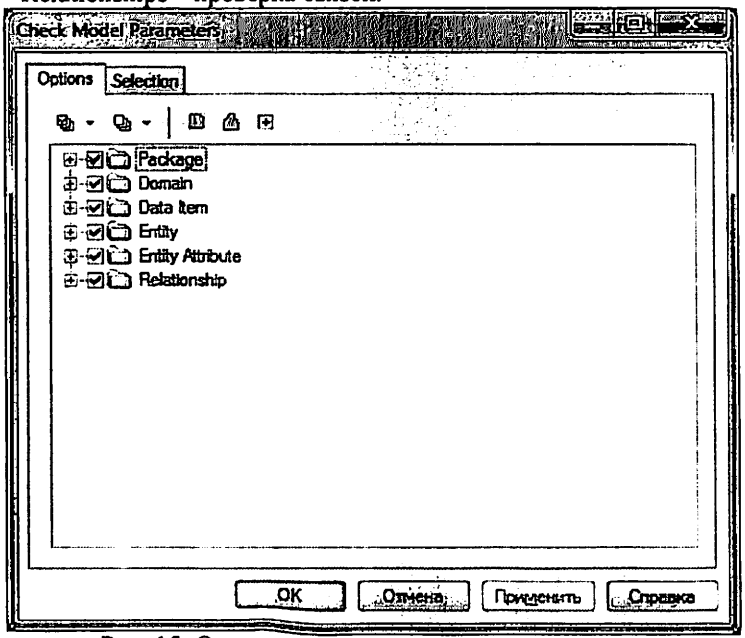

Рис. 15. Окно проверки концептуальной модели

После нажатия кнопки ОК система проверит всю концептуальную модель, выдаст ошибки (или предупреждения), если таковые

имеются. Для просмотра сведений об ошибке необходимо дважды «щелкнуть» по ней кнопкой мыши.

Порядок выполнения работы

1. Создайте концептуальную модель выбранной предметной области, учитывая следующие требования:
    a. Должно быть создано не менее 8-ми сущностей.
    b. В каждой сущности (за исключением справочников-классификаторов) должно быть создано не менее 5-ти атрибутов.
    c. Необходимо использовать домены для определения типов данных атрибутов.
    d. Обязательно соблюдение 3-й нормальной формы для проектируемых сущностей.
    e. Имена полей должны быть представлены по-русски, коды полей должны быть написаны латиницей;
    f. В концептуальной модели данных не должно быть создано внешних ключей;
    g. Необходимо корректно задать имена связей.
2. Выполните проверку качества созданной модели.

Работа считается полностью выполненной, если при проверке модели не выдаются ошибки.

Лабораторная работа № 3 «Генерация физической модели и структуры базы данных»

Тема: Генерация физической модели предметной области.

Цель работы: спроектировать физическую модель, выбранной ранее предметной области на основе созданной концептуальной модели в пакете Power Disigner.

Продолжительность: 2 часа.

Физическая модель данных

На основе спроектированной концептуальной модели создается физическая модель данных, свойственная для конкретной СУБД.

При формировании физической модели данных определяются внешние ключи в связываемых сущностях. Добавляются промежуточные таблицы связи, с целью исключения связей многие-ко-многим (M:M).

Порядок выполнения работы

После того, как проверка концептуальной модели закончится успешно, можно генерировать физическую модель. Для этого необходимо выбрать в меню Tools/Generate Physical Model. Откроется окно генерации физической модели (рис. 16).

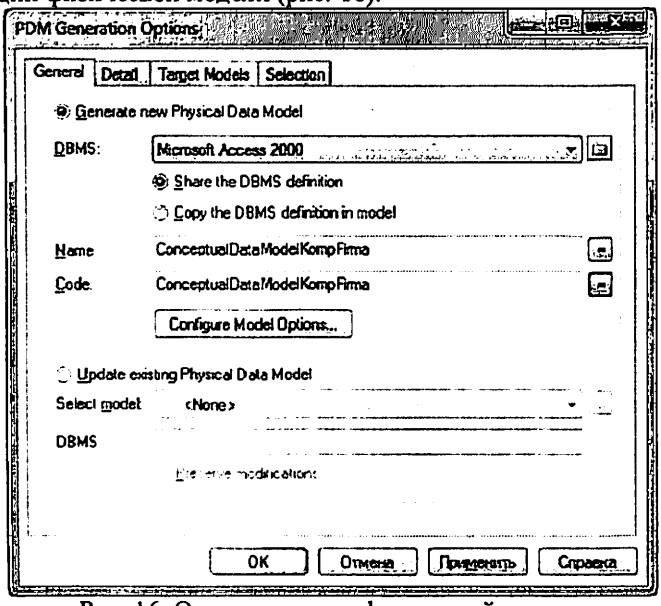

Рис. 16. Окно генерации физической модели

В этом окне необходимо выбрать опцию Generate new Phisical Data Model, и выберать в поле DBMS из выпадающего списка Microsoft Access 2000. *Name* – поле для ввода имени файла для дальнейшей генерации физической модели. Для задания свойств модели щелкните по кнопке Configure Model Options. Откроется окно свойств создаваемой физической модели (рис. 17).

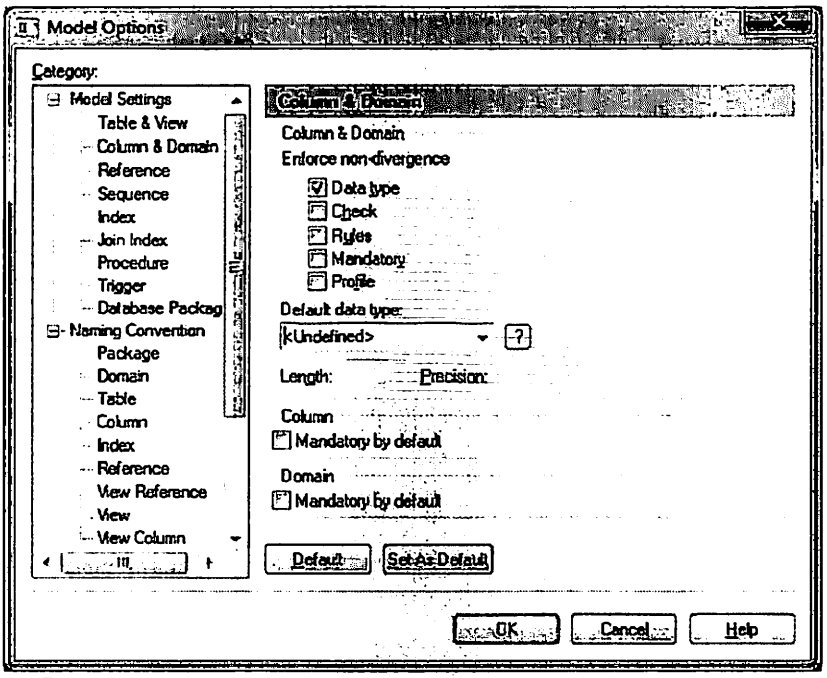

Рис. 17 Окно свойств создаваемой физической модели

Здесь можно выбрать следующие параметры создания физической модели: окно свойств создаваемой физической модели окно свойств создаваемой физической модели

- Data type – учитывать типы данных концептуальной модели при генерации физической модели;
- Check – проверка доменов и соответствие полей атрибутов с выбранным доменом.
- Rules – проверка правил на ввод значений в таблицу.
- Mandatory – учитывать свойства обязательности заполнения.

- Default data type – позволяет установить тип данных по умолчанию для всех не установленных типов данных в атрибутах.
- Domain – использовать ссылки на домен.

В директории Naming convention можно задавать имена шаблонов таблиц, атрибутов и т.д.

Установив необходимые параметры, нажмите кнопку ОК. Если концептуальная модель спроецирована корректно, то система создаст физическую модель базы данных для того типа СУБД, который был указан в параметрах.

Сгенерированную физическую модель также необходимо проверить, нажав клавишу F4

Появится окно (рис. 18), в котором предлагается выбрать объекты для проверки.

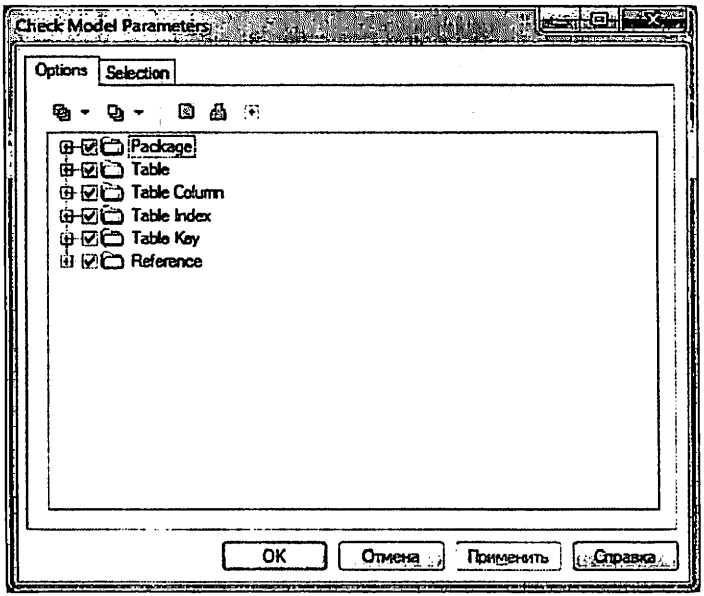

Рис. 18. Окно проверки физической модели

После нажатия кнопки ОК система проверит всю физическую модель, выдаст ошибки (или предупреждения), если таковые имеются.

Если ошибки отсутствуют, на основе данной модели необходимо создать новую базу данных.

Для создания базы данных в меню DataBase выберите пункт Generate DataBase. В появившемся окне (рис. 19) необходимо выбрать опцию Direct generation, путь к файлу, в котором будет сохранен скрипт на создание БД (набор управляющих SQL-запросов), а также дополнительные характеристики БД, перейдя по вкладкам Options, Format, Selection. После чего нажмите кнопку OK.

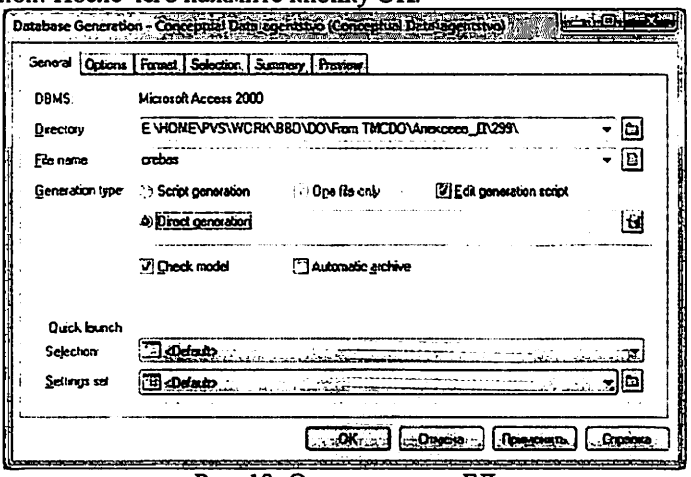

Рис. 19. Окно создания БД

Система попросит указать источник данных (рис. 20).

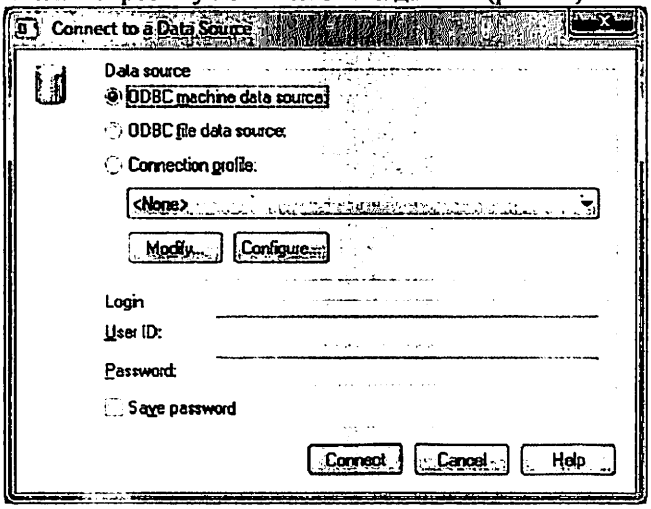

Рис. 20. Окно доступа к БД

Нажмите кнопку Configure для создания собственного источника данных. В появившемся окне «Администратор источников данных ODBC» (рис. 21) нажмите кнопку Добавить.

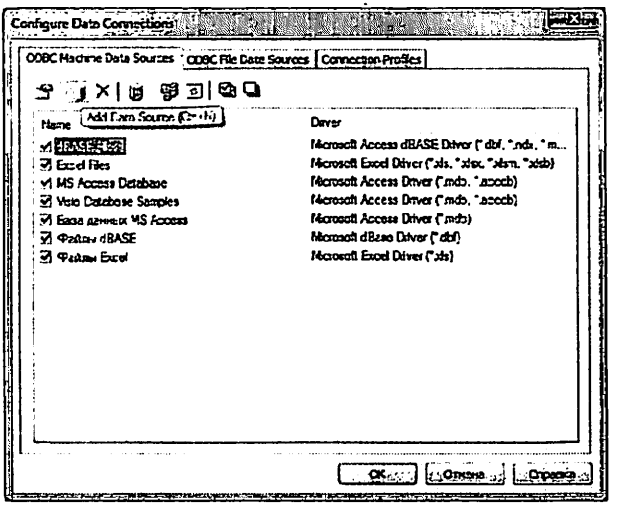

Рис. 21. Окно конфигурации ODBC

В появившемся окне «Создание нового источника» (рис. 22) выберите пользовательский тип источника данных и нажмите кнопку Далее.

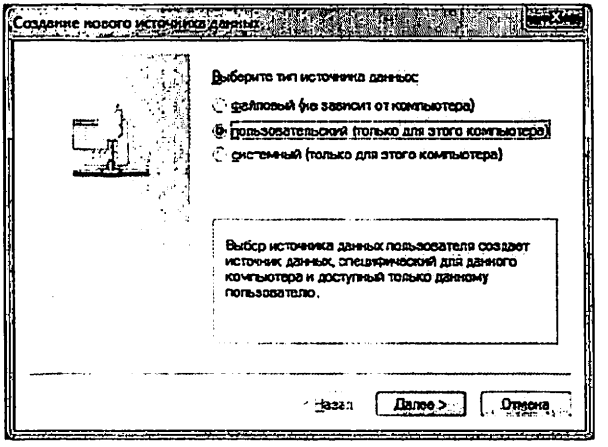

Рис. 22. Окно выбора типа источника данных

В следующем окне (рис. 23) выберите необходимый драйвер ODBC и нажмите кнопку Далее.

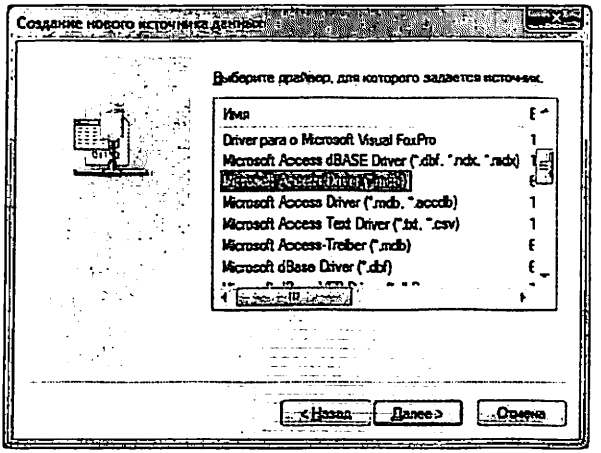

Рис. 23. Окно выбора драйвера ODBC

В следующе окне (рис. 24) нажмите кнопку Готово и задайте имя источника данных и путь в существующему или к новому mdb-файлу (рис. 25).

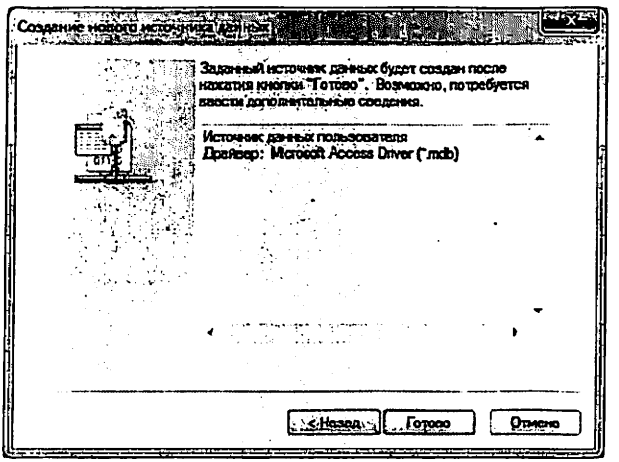

Рис. 24. Подтверждение выбора драйвера ODBC

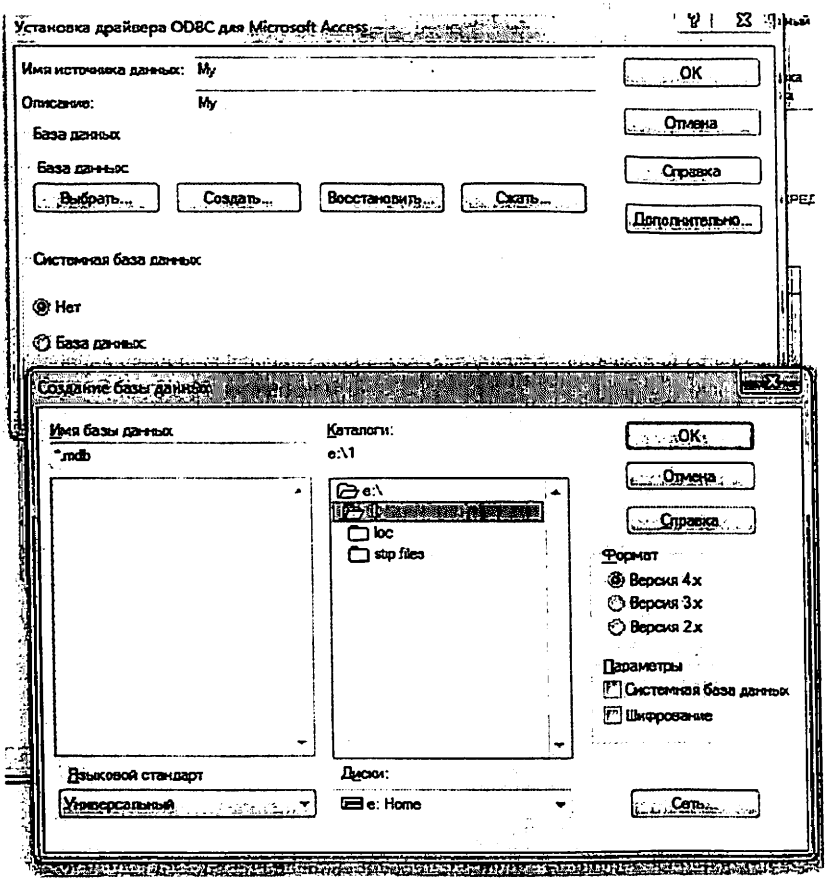

Рис. 25. Подключение к базе данных

Вернитесь в окно доступа к БД (рис. 20) и выберите созданный источник данных (рис. 26).

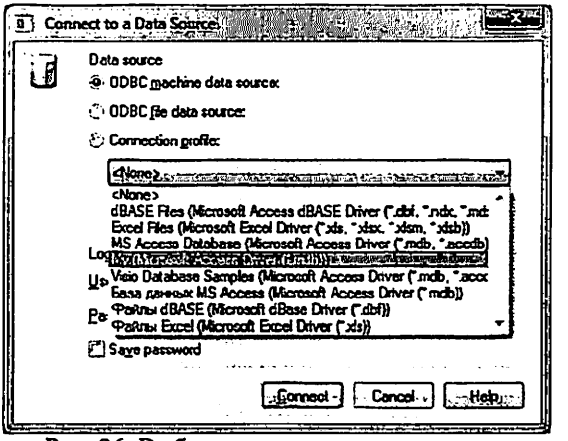

Рис. 26. Выбор своего источника данных

Нажмите кнопку Connect. В появившемся окне (рис 27) будет представлен SQL-скрипт, сгенерированный системой, запуск которого приведет к созданию схемы БД в выбранном mdb-файле.

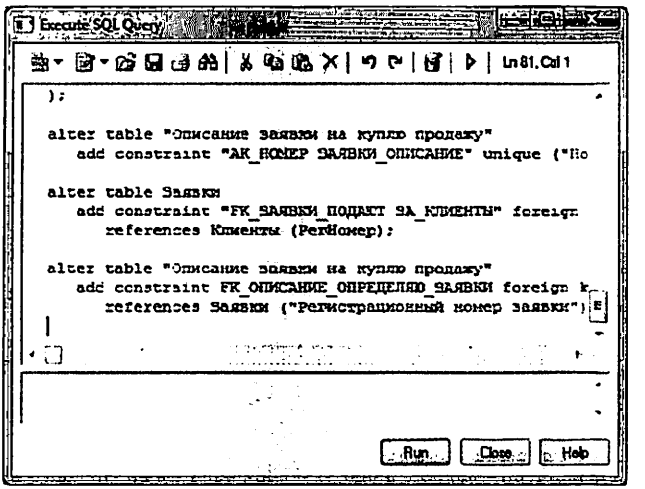

Рис. 27. Окно SQL-скрипта на создание схемы БД

Для запуска SQL-скрипта нажмите кнопку Run. Если система не выдаст сообщений об ошибках, откройте в среде MS Access созданную БД и проверьте ее соответствие физической модели.

Работа считается выполненной, если в схеме данных представлены все таблицы, связи и ключи, соответствующие физической модели данных.

Указания к выполнению курсового проекта

Цель выполнения проекта:

– освоение методики проектирования концептуальной информационной модели предметной области преобразование концептуальной модели в физическую структуру базы данных (БД);

– закрепление теоретических знаний по курсу организация баз данных.

Задачи курсового проекта:

– формализовать исходное описание предметной области;

– построить концептуальную информационную модель, используя методику, изученную в рамках теоретического курса;

– сгенерировать физическую структуру базы данных;

– реализовать простое пользовательское приложение, демонстрирующее накопленные студентом знания по курсу Организация БД.

Средства выполнения и форма отчетности: работа выполняется с использованием любой современной СУБД (MS Access, Oracle, MS SQL, MYSQL, FoxPro for Windows и др.), клиентская часть может быть создана либо средствами выбранной СУБД, либо с помощью любых языков программирования высокого уровня (Delphi, Visual Basic, Visual C и др.). Результат выполнения работы в виде пояснительной записки (отчета), подготовленной в среде MS WinWord, программную систему и базу данных необходимо продемонстрировать на защите курсового проекта

Варианты индивидуального задания

Таблица 2

№	Название предметной области АИС
1.	Библиотека
2.	Магазин продовольственных товаров
3.	ВУЗ
4.	Супермаркет
5.	Документооборот предприятия
6.	Агентство недвижимости
7.	Компьютерная фирма
8.	Поликлиника
9.	Турфирма
10.	Гостиница

Порядок выполнения работы:

1. Создание концептуальной информационной модели предметной области

Каждый студент получает для работы предметную область (Таблица 2).

Концептуальная модель представляется в виде набора ER-диаграмм. Осуществляется формализация исходного описания в виде набора сущностей с последующим их преобразованием и связыванием в концептуальную модель.

Процесс проектирования сопровождается составлением ряда сущностей, необходимыми пояснениями – обоснованиями принимаемых решений

Проектирование концептуальной модели предметной области целесообразно производить с помощью специального средства проектирования: Power Designer.

Основные этапы проектирования концептуальной модели:

1. Первичный анализ информационных потребностей пользователей, выделение объектов предметной области и формирование исходных сущностей:
   - анализ информационных документов;
   - анализ конкретных информационных потребностей (запросов) пользователей.
2. Проектирование исходных сущностей:
   - определение атрибутов сущностей и их типов данных;
   - нормализация сущностей до 3 НФ.
3. Связывание сущностей в концептуальную информационную модель:
   - определение уникальных идентификаторов сущностей (первичных ключей);
   - определение связей между сущностями.

Ограничения концептуальной модели:

- предметная область должна быть описана 8-10 взаимосвязанными сущностями;
- каждая сущность должна содержать не менее 3 атрибутов;
- в каждой сущности должен быть определен уникльный идентификатор сущности.

2. Создание физической модели данных

На основе спроектированной концептуальной модели создается физическая модель данных, свойственная для конкретной СУБД.

При формировании физической модели для определенной СУБД в модели определяются внешние ключи в связываемых сущностях. Добавляются промежуточные таблицы связи, с целью исключения связей многие-ко-многим (M:M), конкретизируются типы данных атрибутов.

Power Designer позволяет произвести автоматическую генерацию физической модели на основе созданной ранее концептуальной. Модели.

3. Создание пользовательского приложения

Приложение, работающее с созданной базой данных должно обеспечивать выполнение следующих функций:

- ввод информации в БД;
- удаление информации из БД;
- редактирование внесенной информации;
- выборка (поиск) данных по таблицам БД с использованием различных критериев;
- формирование отчетов и вывод информации из базы данных на экран и на принтер;

Добавление, замена и удаление информации должны производиться в экранных формах разрабатываемого пользовательского приложения.

4. Оформление пояснительной записки (отчета)

Пояснительная записка к курсовому проекту должна включать:

- титульный лист,
- содержание,
- введение,
- основную часть,
- заключение,
- список использованных литературных источников,
- приложения.

Титульный лист оформляется согласно действующим стандартам (примеры оформления титульного листа и задания на курсовой проект представлены в Приложении 1 и 2).

Введение должно содержать цель выполняемой курсовой работы, основные принципы, положенные в основу ее проведения, область применения.

В основной части должен быть отражен процесс и результат проектирования базы данных и пользовательского приложения. Основная часть должна содержать:

- описание предметной области;
- описание и обоснование выбранного средства реализации (СУБД, средства проектирования, программной среды написания приложения);
- концептуальную информационную модель предметной области с полным описанием выделенных сущностей;
- физическую модель и схему базы данных;
- описание пользовательского приложения.

Заключение должно содержать краткие выводы по результатам выполненной работы.

Список использованных литературных источников оформляется согласно действующим стандартам.

В приложениях приводятся: экранные формы пользовательского приложения, тексты SQL-запросов, создаваемых в приложении и другая сопроводительная информация.

Самостоятельная работа

Согласно рабочей программе отводится следующее количество часов на самостоятельную работу:

- подготовка к контрольным работам – 18 часов;
- подготовка к лабораторным работам – 18 часов;
- подготовка курсового проекта – 90 часов;
- подготовка к сдаче экзамена – 36 часов.

Форма контроля и проверка достижения заявленных компетенций (ПК-2 и ПК 15): проведение контрольных работ (в том числе тестовых), опрос перед проведением лабораторных работ, проверка отчетов, защита индивидуального задания путем представления презентации – выступление на лекции с демонстрацией примера нормализации.

Для проработки лекционного материала студентам, помимо конспектов лекций, рекомендуются следующие главы учебно-методического пособия [1] по разделам курса:

- Моделирование данных с помощью ER-диаграмм – главы 5.2., 5.3.
- Реляционные языки – глава 6.
- Физическая структура данных – глава 7.

- Объектно-ориентированный подход к организации БД – глав 8.3.
- Системы управления базами данных – глава 8.

Для подготовки к экзамену, лабораторным работам и курсовому проекту рекомендуется повторить соответствующие тематике разделы учебно-методического пособия [1], а также ознакомиться с порядком выполнения лабораторных работ, по настоящему руководству.

Для изучения тем теоретической части курса, отводимых на самостоятельную проработку, рекомендуется ознакомление со всеми разделами [1]. Кроме того, рекомендуется повторить разделы предложенной литературы [2-6], посвященные проектированию данных и построению пользовательских приложений.