Министерство образования и науки Российской Федерации

Калужский филиал
федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

С.А. Глебов

СОЗДАНИЕ ОБЪЕКТОВ БАЗЫ ДАННЫХ СРЕДСТВАМИ ЯЗЫКА DDL

Методические указания по выполнению лабораторной работы
по курсу «Базы данных»

Калуга – 2018

УДК 004.65
ББК 32.972.134
Г53

Методические указания составлены в соответствии с учебным планом КФ МГТУ им. Н.Э. Баумана по направлению подготовки 09.03.04 «Программная инженерия» кафедры «Программного обеспечения ЭВМ, информационных технологий и прикладной математики».

Методические указания рассмотрены и одобрены:

- Кафедрой «Программного обеспечения ЭВМ, информационных технологий и прикладной математики» (ФН1-КФ) протокол № 7 от «21» февраля 2018 г.

И.о. зав. кафедрой ФН1-КФ _____ к.т.н., доцент Ю.Е. Гагарин

- Методической комиссией факультета ФНК протокол № 2 от «28» 02 2018 г.

Председатель методической комиссии факультета ФНК _____ к.х.н., доцент К.Л. Анфилов

- Методической комиссией КФ МГТУ им.Н.Э. Баумана протокол № 2 от «06» 03 2018 г.

Председатель методической комиссии КФ МГТУ им.Н.Э. Баумана _____ д.э.н., профессор О.Л. Перерва

Рецензент:
к.т.н., доцент кафедры ЭИУ6-КФ _____ А.Б. Лачихина

Авторы
к.ф.-м.н., доцент кафедры ФН1-КФ _____ С.А. Глебов

Аннотация

Методические указания по выполнению лабораторной работы по курсу «Базы данных» содержат руководство по созданию баз данных, а также индексов и просмотров в базах данных средствами DDL.

Предназначены для студентов 3-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Настоящие методические указания составлены в соответствии с программой проведения лабораторных работ по курсу «Базы данных» на кафедре «Программное обеспечение ЭВМ, информационные технологии и прикладная математика» факультета фундаментальных наук Калужского филиала МГТУ им. Н.Э. Баумана.

Методические указания, ориентированные на студентов 3-го курса направления подготовки 09.03.04 «Программная инженерия», содержат руководство по созданию баз данных, а также индексов и просмотров в базах данных средствами DDL и задание на выполнение лабораторной работы.

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Целью выполнения лабораторной работы является закрепить навыки использования операторов определения данных, научится определять ограничения, индексы, просмотры.

Основными задачами выполнения лабораторной работы являются:

- создать таблицы базы данных выбранной предметной области, используя СУБД Firebird,
- определить домены, столбцы, вычисляемые поля, ограничения первичного и внешнего ключей,
- организовать необходимые индексы и просмотры.

Результатами работы являются:

- Разработанная база данных выбранной предметной области.
- Созданные индексы и просмотры в разработанной базе данных.
- Подготовленный отчет.

ТИПЫ ДАННЫХ В FIREBIRD

Для создания таблиц используются следующие типы данных.

1. INTEGER — целое число 4 байта (–2 147 483 648 ÷ +2 147 483 647),

2. SMALLINT — короткое целое 2 байта (–32 788 ÷ +32 767)

3. NUMERIC (w, d), DECIMALS (w, d)

w — разрядность (1÷18) — общее количество цифр (в целой и дробной частях без учета десятичной точки);

d — количество десятичных разрядов (0÷w).

Фактически, числа с фиксированной точкой относятся к порядковым типам, т.е. все возможные значения можно пересчитать по порядку. Для них выделяется определенное количество байт в зависимости от разрядности, в которых хранится двоичное представление целого числа, а для этого целого числа просто определяется положение десятичной точки.

Например, если мы определяем столбец как NUMERIC (3,2), это не означает, что максимальное значение, которое можно в нем хранить — 9,99. Для этого столбца выделяется 2 байта и т.о. максимальное значение — 327,67, а минимальное — –327,68.

При w = 1—4 для NUMERIC выделяется 2 байта, для DECIMALS — 4 и это вся разница между этими типами. Например,

CREATE TABLE T1 (A NUMERIC (3, 1), B DECIMALS (3,1));

Если после создания такой таблицы посмотреть на ее структуру, то INTERBASE покажет: A NUMERIC (3, 1), B NUMERIC (3,1), тем не менее максимально допустимые значения будут: для A — 3276,7, а для B — 2 147 483 64,7.

При w = 5—9 и для NUMERIC и для DECIMALS выделяется по 4 байта.

При w = 10—18 — по 6 байт.

4. DOUBLE PRECISION — тип чисел с плавающей точкой для бухгалтерских и научных расчетов. Существует еще тип FLOAT, который не рекомендуется использовать по причине недостаточной точности, а также в нем быстро накапливаются ошибки округления.

5. CHAR (Character) и VARCHAR (Character Varying).

Например,

CREATE TABLE T1 (A CHAR (200), B CHAR);

В результате поле A будет иметь длину 200 символов, а B — 1 символ.

Максимальная длина — 32768 символов.

Отличие между двумя строковыми типами в том, что фактическая длина значений типа CHAR всегда совпадает с максимальной. Т.е. даже если в поле A таблицы T1 занести значение 'ABC', то при выборке мы получим значение 'ABC <197 пробелов> '.

Для значений типа VARCHAR понятия максимально возможная длина и фактическая различаются.

На практике использование типа CHAR довольно ограничено.

Для строкового типа важнейшей характеристикой является CHARACTER SET — набор символов. Набор символов определяется для всей БД и по умолчанию используется для всех символьных полей. Однако его можно явно переопределить при создании поля. Например:

```
CREATE TABLE T1 (
  A VARCHAR (20),
  B VARCHAR (20) CHARACTER SET WIN1251);
```

Для символьных полей возможно указывать порядок сортировки COLLATION ORDER. Для кириллического набора символов WIN1251 возможны два варианта:

```
CREATE TABLE T1 (
  A VARCHAR (20) COLLATION ORDER WIN1251,
  B VARCHAR (20) COLLATION ORDER PXW_CYRL);
```

A	**B**
Ф	Ф
б	б
Ц	Ц
ф	ф
Б	Б
А	А
ц	ц

```
SELECT *
FROM T1
ORDER BY A;
```

A	**B**
А	А
Б	Б
Ф	Ф
Ц	Ц
б	б
ф	ф
ц	ц

```
SELECT *
FROM T1
ORDER BY B;
```

A	**B**
А	А
б	б
Б	Б
ф	ф
Ф	Ф
ц	ц
Ц	Ц

6. DATE — дата в диапазоне 1 января 100 г до 29 февраля 32768 года;

7. TIME — время с точностью до 1/10 000 доли секунды, 00:00÷23:59:59.9999

8. TIMESTAMP — дата и время.

С полями этих трех типов возможны операции + и –. С типом DATE операции выполняются в днях, с TIME — в секундах, с TIMESTAMP — в сутках, с учетом доли суток.

Например, 31.10.04 – 29.10.04 = 2; 30.12.04 + 3 = 2.01.05

07:00:00 – 08:00:01 = –3601; 23:00:00 + 7200 = 01:00:00

27.10.2004 7:00:00 – 25.10.2004 1:00:00 = 2,25

9. Binary Large OBject — динамически расширяемый тип данных. В полях этого типа могут хранится данные неограниченного размера. Как правило это примечания, графическая, аудио-, видео- информация. Достигается это за счет физического размещения BLOB-полей отдельно от обычных.

Несмотря на сложность реализации, объявить BLOB-столбец очень просто:

```
CREATE TABLE T1 ( A BLOB );
```

Поля-массивы являются расширением традиционной реляционной модели, по которой каждый атрибут (поле) должен быть атомарным, т.е. неделимым.

```
CREATE TABLE T1 (
  A INTEGER [1:12],
  B CHAR [-5:5, 10],
  C VARCHAR (20) [6, 6, 6]);
```

В таблице T1 объявлено 3 массива, каждый из которых содержит данные различных типов: A — одномерный (можно писать A INTEGER [12]), B — двумерный 11×10, C — трехмерный 6×6×6.

Реализованы массивы на базе полей типа BLOB.

ТАБЛИЦЫ

Определение таблиц выполняется оператором `CREATE TABLE`. В нем определяются столбцы будущей таблицы и ограничения. Его синтаксис:

CREATE TABLE <table_name> [EXTERNAL [FILE] "<file_spec>"]
(<col_def1> [<coldef2>, ...] [<constraint_def>, ...]);

где <table_name> — имя таблицы, если указано [EXTERNAL [FILE] "<file_spec>"], то будет создана т.н. внешняя таблицы, которая хранится в отдельном файле <file_spec>. Работа с внешними таблицами ограничена операциями вставки и выборки.

<col_def> — определение столбца (поля) таблицы. В свою очередь имеет синтаксис:

<col_def> = col_name { datatype | domain | COMPUTED [BY] (<expr>) }
[DEFAULT {<default_value> | NULL | USER}]
[NOT NULL]
[<col_constraint>]
[COLLATE collation]

Здесь, <col_name> — имя столбца, для которого должен быть указан либо тип данных (datetype), либо имя домена (domain), предварительно определенного, либо значение в столбце будет вычисляться (COMPUTED BY), но только при выборке по этому столбцу, т.е. в этом поле ничего не хранится.

Значение столбца может быть определено по умолчанию (DEFAULT). Т.е. если при вставке записи в таблицу пользователь оставляет не заполненным это поле, то ему автоматически присваивается либо значение <default_value>, либо значение NULL, либо имя пользователя, который вставляет запись (USER).

На столбец может быть наложено требование NOT NULL, что не позволит ни при каких операциях (вставка, изменение) оставить это поле без значения. Часто опцию NOT NULL сочетают с DEFAULT.

<CHECK> определяет дополнительные ограничения на возможные значения столбца для реализации бизнес-правил.

<COLLATE> определяет порядок сортировки для символьных столбцов в случае различного регистра.

В качестве примера создадим таблицу Prod.

```
CREATE TABLE prod (
   id_pr INTEGER NOT NULL,
   fio VARCHAR(30) NOT NULL,
   bd DATE,
   adres VARCHAR(50),
   money INTEGER DEFAULT 0);
```

При определении таблиц создаются особые объекты БД — ограничения — CONSTRAINT. Существуют следующие виды ограничений:

– первичный ключ — PRYMARY KEY;
– уникальный ключ — UNIQUE KEY;
– внешний ключ — FOREIGN KEY;
– проверки — CHECK.

При определении столбца синтаксис ограничений следующий:

```
<col_constraint> =
[CONSTRAINT constraint_name]
{ UNIQUE  |  PRYMARY KEY  |  CHECK <condition>  |
 REFERENCES table_name [(column_list)]
 [ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
 [ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
}
```

```
CREATE TABLE prod (
   id_pr INTEGER NOT NULL CONSTRAINT C1 PRIMARY KEY,
   fio VARCHAR(30) NOT NULL,
   bd DATE,
   adres VARCHAR(50),
   money INTEGER DEFAULT 0 CONSTRAINT C2 CHECK (money >= 0));
```

На таблицу PROD наложены два ограничения C1 (id_pr — PRYMARY KEY) и C2 (money должен быть положительным) Объявление столбца id_pr в качестве первичного ключа, автоматически накладывает на него требование уникальности, но требование

непустого значения первичного ключа NOT NULL нужно явно указывать. Если требуется уникальность значений столбца не являющегося первичным ключом, то используется ключевое слово UNIQUE.

Как видно из синтаксиса требование явного именования ограничений не является обязательным. Можно было бы написать:

```
CREATE TABLE prod (
   id_pr INTEGER NOT NULL_PRIMARY KEY,
   fio VARCHAR(30) NOT NULL,
   bd DATE,
   adres VARCHAR(50),
   money INTEGER DEFAULT 0 CHECK (money >= 0)   );
```

В этом случае ограничения получат системные имена, которые хранятся в системной таблице RDB$RELATION_CONSTRAINTS.

Ограничения могут быть созданы не только при определении столбца, но и наряду с определениями столбцов в таблице. В этом случае в них могут быть задействованы несколько полей (например, составной ключ) и синтаксис поэтому немного иной:

```
<constraint_def> = [CONSTRAINT constraint_name]
{ PRIMARY KEY | UNIQUIE (column_list) } |
  FOREIGN KEY (column_list) REFERENCES <table_name> [(column_list)]
    [ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
    [ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
```

Например, создание таблицы PROD может выглядеть:

```
CREATE TABLE prod (
      id_pr INTEGER NOT NULL,
      CONSTRAINT C1 PRIMARY KEY (id_pr),
      fio VARCHAR(30) NOT NULL,
      bd DATE,
      adres VARCHAR(50),
      money INTEGER DEFAULT 0,
      CONSTRAINT C2 CHECK (money >=0)      );
```

Общий синтаксис условия в предложении CHECK:

<search_condition> =

{<val> <operator> { <val> | (<select_one>) }

| <val> [NOT] BETWEEN <val> AND <val>

| <val> [NOT] LIKE <val> [ESCAPE <val>]

| <val> [NOT] IN (<val> [, <val> ...] | <select_list>)

| <val> IS [NOT] NULL

| <val> [NOT] {= | < | >} | >= | <=} {ALL | SOME | ANY } (<select_list>)

| EXISTS (<select_expr>)

| SINGULAR (<select_expr>)

| <val> [NOT] CONTAINING <val>

| <val> [NOT] STARTING [WITH] <val>

| (<search_condition>)

| NOT <search_condition>

| <search_condition> OR <search_condition>

| <search_condition> AND <search_condition>}

Проверка CHECK относится только к текущей записи. Не следует строить выражения с использованием других записей этой же таблицы.

Поле может иметь только одно CHECK ограничение.

Если при определении столбца использовался домен со своим ограничением, то его нельзя переопределить на уровне конкретного поля.

Домен представляет собой некоторый базовый тип данных с наложенным на него фильтром (условием). Домен является объектом всей БД, а не отдельной таблицы и может использоватся при определении столбцов различных таблиц наряду со стандартными типами.

В нашей базе данных у таблиц PROD и STAR имеет смысл определить домен для поля MONEY.

CREATE DOMAIN D_Money INTEGER DEFAULT 0 CHECK (VALUE >= 0)

Определение домена практически полностью повторяет определение столбца за исключением того, что при наложении ограничения CHECK вместо имени столбца используется ключевое слово VALUE — просто значение.

```
CREATE TABLE prod (
        id_pr INTEGER NOT NULL,
        CONSTRAINT INT1 PRIMARY KEY (id_pr),
        fio VARCHAR(30) NOT NULL,
        bd DATE,
        adres VARCHAR(50),
        money D_Money)          );
```

Разумеется, нет смысла создавать домен для его одноразового использования. Чем в больше количество таблиц, использующих один домен, тем яснее преимущество. Помимо этого, очевидного плюса, домены сокращают и количество ограничений в БД, т.к. ограничение будет одно для каждого домена.

В нашем примере для таблицы PROD будет создано только одно ограничение (на id_pr — PK).

Как известно, для создания связи между двумя таблицами необходимы два ключевых поля — PRIMARY KEY и FOREIGN KEY. Известно также, что FK может принимать значения только из множества значений PK или быть пустым. Это есть не что иное как ограничение. Поэтому в синтаксисе ограничений есть фраза и для объявления FK (в синтаксисе на основе определения столбца присутствует неявно).

Очевидно, что зависимая (дочерняя) таблица должна быть создана только после создания главной.

В рассматриваемом примере таблица MOVIE является зависимой от таблицы PROD по полю id_pr:

```
CREATE TABLE movie (
    id_m INTEGER NOT NULL PRIMARY KEY,
    title VARCHAR(40) NOT NULL UNIQUE,
    year INTEGER NOT NULL CHECK (year>1910),
    len INTEGER NOT NULL CHECK (len>20),
    kind CHAR(10) CHECK
        (kind IN ('Комедия','Боевик','Мелодрама')),
    id_pr INTEGER,
    FOREIGN KEY (id_pr) REFERENCES PROD        );
```

Предложение FOREIN KEY определяет вторичный ключ id_pr для связи дочерней таблицы MOVIE с родительской таблицей PROD.

Рассмотрим подробнее формат определения:

FOREIN KEY (<список столбцов внешнего ключа>)

REFERENCES <имя родительской таблицы>

[<список столбцов родительской таблицы>]

[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]

[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]

Список столбцов внешнего ключа определяет столбцы дочерней таблицы, по которым строится внешний ключ.

Имя родительской таблицы определяет таблицу, в которой описан первичный ключ (или столбец с атрибутом *UNIQUE).* На этот ключ (столбец) должен ссылаться внешний ключ дочерней таблицы для обеспечения ссылочной целостности.

Параметры ON DELETE, ON UPDATE определяют способы изменения подчиненных записей дочерней таблицы при удалении или изменении поля связи в записи родительской таблицы. Перечислим эти способы:

- NO ACTION — запрет удаления/изменения родительской записи при наличии подчиненных записей в дочерней таблице;
- CASCADE — для оператора ON DELETE: при удалении записи родительской таблицы происходит удаление подчиненных записей в дочерней таблице; для ON UPDATE: при изменении поля связи в записи родительской таблицы происходит изменение на то же значение поля внешнего ключа у всех подчиненных записей в дочерней таблице;
- SET DEFAULT — в поле внешнего ключа у записей дочерней таблицы заносится значение этого поля по умолчанию, указанное при определении поля (параметр DEFAULT*);* если это значение отсутствует в первичном ключе, возбуждается исключение; причем используется значение по умолчанию, имевшее место на момент определения ссылочной целостности; если впоследствии

это значение будет изменено, ссылочная целостность при SET DEFAULT *все* равно будет использовать прежнее значение;

– SET NULL — в поле внешнего ключа у записей дочерней таблицы заносится значение NULL.

Удаление таблицы осуществляется оператором DROP TABLE <имя_таблицы>.

Удаление невозможно для родительских таблиц, если в дочерних таблицах имеются ссылки по внешнему ключу этих таблиц и удаление главной таблицы приведет к нарушению целостности БД. Поэтому нужно сначала либо удалить наложенные ограничения ссылочной целостности, либо удалить сначала дочерние таблицы, а только затем главные.

Оператор ALTER TABLE позволяет добавить|удалить (ADD | DROP) столбец, добавить|удалить (ADD | DROP CONSTRAINT) ограничения.

Изменить атрибуты столбца непосредственно нельзя. Для этого нужно проделать определенную процедуру.

Перед изменением каких-либо атрибутов столбца данные, которые хранятся в нем, нужно сохранить. Для этого в таблице определяют временный столбец, в точности повторяющий все характеристики того столбца, который планируется изменить. Затем данные из изменяемого столбца копируют во временный столбец (используя, например, оператор UPDATE*)*. После этого столбец, подлежащий изменению, попросту удаляют из таблицы, а на его месте создают новый, одноименный столбец с желаемыми атрибутами. В заключение в него копируют данные из временного столбца, а временный столбец уничтожают.

Пусть, например, необходимо изменить характеристики столбца FIO, изменив тип столбца с VARCHAR(4O) на CHAR(50).

1. Добавляем в таблицу новый временный столбец FIO_TMP, полностью повторяющий характеристики изменяемого столбца FIO*:*

```
ALTER TABLE PROD
ADD FIO_TMP VARCHAR(40);
```

2. Копируем данные из FIO в FIO_TMP*:*

```
UPDATE PROD SET FIO_TMP = FIO;
```

3. Удаляем столбец FIO*:*

ALTER TABLE PROD DROP FIO;

4. Создаем новый столбец FIO с необходимыми характеристиками:

ALTER TABLE PROD ADD FIO CHAR(50);

5. Переписываем данные из столбца FIO_TMP в новый столбец FIO:

UPDATE PROD SET FIO = FIO_TMP;

6. Удаляем временный столбец FIO_TMP:

ALTER TABLE PROD DROP FIO_TMP;

Замечание. Следует помнить, что изменение характеристик столбца, а также удаление столбца невозможно, если:

- столбец приобретает атрибуты `PRIMARY KEY` или `UNIQUE`, но старые значение в столбце нарушают требования уникальности данных;
- удаляемый столбец входил как часть в первичный или внешний ключ, что привело к нарушению ссылочной целостности между таблицами;
- столбцу были приписаны ограничения целостности `CHECK` на уровне таблицы;
- столбец использовался в иных компонентах БД — в просмотрах, триггерах, в выражениях для вычисляемых столбцов.

Все вышесказанное свидетельствует о том, что в случае необходимости изменения атрибутов столбца или в случае удаления столбца сначала необходимо тщательно проанализировать, какие последствия для таблицы и базы данных в целом может повлечь такое изменение или удаление.

Оператор INSERT применяется для добавления записей в объект. В качестве объекта может выступать таблица или просмотр (VIEW), созданный оператором CREATE VIEW. В последнем случае записи могут добавляться сразу в несколько таблиц.

Формат оператора INSERT:

INSERT INTO <объект> [(столбец1 [, столбец2 ...])]

{VALUES (<значение1> [, <значение2> ...]) | <оператор SELECT>}

Список столбцов указывает столбцы, которым будут присвоены значения в добавляемых записях. Список столбцов может быть

опущен. В этом случае подразумеваются все столбцы объекта, причем в том порядке, в котором они определены в данном объекте.

Поставить в соответствие столбцам списки значений можно двумя способами. Первый состоит в явном указании значений после слова VALUES, второй — в формировании значений при помощи оператора SELECT.

Например, добавить в таблицу

```
CREATE TABLE movie (
    id_m INTEGER NOT NULL PRIMARY KEY,
    title VARCHAR(40) NOT NULL UNIQUE,
    year INTEGER NOT NULL CHECK (year>1910),
    len INTEGER NOT NULL CHECK (len>20),
    kind CHAR(10) CHECK (kind IN ('Комедия','Боевик','Мелодрама')),
    id_pr INTEGER,
    FOREIGN KEY (id_pr) REFERENCES PROD );
```

еще один фильм:

```
INSERT INTO MOVIE (id_m, title, year, length, kind, id_pr)
VALUES (19, ‘Двенадцать стульев’, 1986, 455, ‘комедия’, 6);
```

Если добавляемые значения указываются для всех столбцов, нет необходимости перечислять столбцы таблицы:

```
INSERT INTO MOVIE
VALUES (19, ‘Двенадцать стульев’, 1986, 455, ‘комедия’, 6);
```

Значения присваиваются столбцам по порядку следования тех и других в операторе: первому по порядку столбцу присваивается первое значение, второму столбцу — второе значение и т. д.

Существует другой вариант добавления записей — при помощи оператора SELECT. Н-р, отразить в БД, что режиссер Э. Рязанов стал актером.

```
INSERT INTO star (fio, bd, adres, money)
    SELECT fio, bd, adres, money FROM prod
    WHERE fio=’Э. Рязанов’;
```

Т.к. оператор SELECT способен вернуть множество записей, то можно добавить несколько записей сразу. Н-р, предположим, что продюсерами стали сразу все актеры с доходом более 100000.

```
INSERT INTO prod (fio, bd, adres, money)
    SELECT fio, bd, adres, money FROM star
    WHERE money>=100000;
```

Следует обратить внимание, что приведенные примеры носят показательный характер, т.к. практически в таблицу добавляются записи для которых отсутствует идентификатор id_pr и id_st, которые являются в своих таблицах первичными ключами и не могут быть пустыми.

Для этих целей в составе БД используют генераторы — механизм сервера БД, возвращающий уникальные значения, никогда не совпадающие со значениями, выданными этим же генератором в прошлом.

Оператор UPDATE используется для изменения значения в группе записей или (в частном случае) в одной записи объекта. В качестве объекта могут выступать ТБД или просмотр, созданный оператором CREATE VIEW. В последнем случае могут изменяться значения записей из нескольких таблиц.

Формат оператора UPDATE*:*

UPDATE <объект>

SET столбец1 = <значение1> [,столбец2 = <значение2>...]

[WHERE <условие поиска >]

При корректировке каждому из перечисленных столбцов присваивается соответствующее значение. Корректировка выполняется для всех записей, удовлетворяющих условию поиска, которое задается так же, как в операторе SELECT.

Обратите внимание: если опустить WHERE <условие поиска>, в объекте будут изменены все записи!

Н-р, увеличить доход всех актеров, занятых в фильме «Операция Ы» на 20%.

```
UPDATE star
SET star.money = star.money*1.2
WHERE id_st IN (SELECT id_st
        FROM starin si INNER JOIN movie m ON (si.id_m=m.id_m)
        WHERE title='Операция "Ы"');
```

Предназначен для удаления группы записей из объекта. В качестве объекта могут выступать ТБД или просмотр VIEW. Формат оператора:

DELETE FROM <объект>

WHERE <условие поиска>;

Из объекта будут удалены все записи, удовлетворяющие условию поиска. Как и в случае оператора UPDATE, если условие WHERE не указано, будут удалены все записи.

Н-р, удалить фильмы режиссера Э. Рязанова.

```
DELETE FROM movie
WHERE id_pr IN (SELECT id_pr FROM prod
                WHERE fio = 'Э. Рязанов')
```

ИНДЕКСЫ

Индексы предназначены для быстрого поиска записи. Как компьютеру, так и человеку поиск в каком-либо множестве удобнее всего осуществлять по упорядоченным элементам этого множества. Поэтому, если требуется найти запись в таблице, то нужно все записи упорядочить. По какому принципу? Очевидно — по тому полю, по значению которого мы ищем запись. Например, при поиске актера по его фамилии, если расположить записи в алфавитном порядке следования фамилий, то поиск будет происходить значительно быстрее. Но, в то же время, требуется поиск и по идентификатору актера. Понятно, что алфавитный порядок следования фамилий не совпадает с порядком следования идентификаторов. Сортировать таблицу каждый раз для каждого поиска по затратам выйдет даже дороже, чем поиск по неотсортированной таблице. Поэтому, поступают так: основную таблицу оставляют без изменений, записи в ней располагаются в порядке их естественного добавления, отдельно составляют упорядоченный список значений поля и каждому значению ставят в соответствие указатель на местоположение соответствующей записи. Такой список, сохраненный в БД вместе с таблицей и есть индекс.

Понятно, что индекс занимает объем равный объему данных в индексируемом поле плюс объем указателей, т.е. суммарный объем индексов может превышать размер таблицы.

Индексы требуют постоянной поддержки, т.е. при модификации таблицы (удалении, добавлении записей или изменении индексного поля), разумеется индекс также должен быть преобразован. Однако, затраты на поддержание индексов с лихвой окупаются скоростью поиска. Хотя на каждую таблицу можно создать до 64 индексов, вполне очевидно, что индексы нужно создавать только по тем полям, по которым действительно требуются поиск и сортировка. В случае необоснованно большого количества индексов, поддержка индексов может сильно сказаться на производительности всей БД.

Кроме быстрого поиска, индексы позволяют пользователю видеть записи в том или ином порядке, отличном от ее физического

размещения, т.е. индекс позволяет отделить хранение данных от их представления.

Т.о. индекс — это отдельный объект БД, содержащий упорядоченные значения одного или нескольких полей таблицы и связанный с ними указатель на страницу, где расположена запись, содержащая поле с этим значением. Индексы предназначены для быстрого поиска и сортировки.

Общий формат оператора CREATE INDEX:

CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]

INDEX <index_name> ON <table_name> (col1 [,col2 ...]);

- UNIQUE — требует создания уникального индекса, не допускающего одинаковых значений индексных полей для разных записей таблицы;
- ASC[ENDING] — указывает на необходимость сортировки значений индексных полей по возрастанию (режим принят по умолчанию);
- DESC[ENDING] — указывает на необходимость сортировки значений индексных полей по убыванию;
- index_name — имя создаваемого индекса;
- table_name — имя таблицы, для которой создается индекс;
- colN — имена столбцов, по которым создается индекс.

Индексы по полям, на которые наложены ограничения: PRIMARY KEY, FOREIGN KEY, UNIQUE создаются автоматически. Для PRIMARY KEY и UNIQUE они необходимы, т.к. при добавлении или изменении записи нужно проверять наличие такого же значения поля для обеспечения его уникальности. Индекс по FOREIGN KEY при изменении/удалении записи родительской таблицы дает возможность быстрого поиска связанных с ней записей дочерней таблицы для выполнения действий по поддержанию ссылочной целостности.

Индексы необходимо создавать в случае, когда по столбцу или группе столбцов:

- часто производится поиск в БД (столбец или группа столбцов часто перечисляются в предложении WHERE оператора SELECT);

– часто строятся соединения таблиц (JOIN);
– часто производится сортировка (то есть столбец или столбцы часто используются в предложении ORDER BY оператора SELECT).

Не рекомендуется строить индексы по полю или полям, которые:

– редко используются для поиска, объединения и сортировки результатов запросов;
– часто меняют значение, что приводит к необходимости часто обновлять индекс и способно существенно замедлить скорость работы с БД;
– содержат небольшое число вариантов значения (например, пол).

Помимо двух вышеуказанных причин — занимаемый индесками объем и постоянная поддержка индексов, по которым не следует строить черезчур много индексов, существует еще ряд замечаний.

Как известно, язык SQL является непроцедурным, т.е. позволяет сформулировать ЧТО нужно получить в результате, а КАК добиться нужно результата, т.е. алгоритм или план выполнения запроса, строит т.н. оптимизатор запросов. К сожалению, оптимизатор далеко не идеален и при сложных запросах и при большом количестве индексов способен построить далеко не оптимальный план, задействовав не самые эффективные индексы.

Если запрос возвращает более 20% записей, то использование индексов может существенно замедлить выполнение запроса. Конечно цифра 20% весьма приблизительная, но она показывает тот порог, когда эффективность индекса можно поставить под сомнение.

После многократного внесения изменений в таблицу БД индексы этой таблицы могут быть разбалансированы. Разбалансировка приводит к тому, что «глубина» индекса (depth) возрастает сверх критического значения. «Глубина» индекса — параметр, показывающий максимальное количество операций, необходимых для нахождения искомого значения в таблице БД с использованием данного индекса. В случае разбалансировки индекса его ценность при выполнении запросов снижается из-за увеличения времени выполнения запроса.

Поэтому время от времени необходимо выполнять либо перестройку индекса, либо удалить и заново создать индекс:

Перестройка индекса заключается в пересоздании и балансировке индекса. Эти операции начинаются после деактивизации индекса и последующей его активизации

ALTER INDEX <имя индекса> INACTIVE;

ALTER INDEX <имя индекса> ACTIVE;

Деактивизация индекса полезна также в том случае, когда в таблицу БД вставляется большое число записей одновременно: при активном индексе изменения в него вносятся при добавлении каждой записи, что замедляет доступ к данным.

Удаление индекса: DROP INDEX <имя индекса>;

Нельзя удалить или перестроить индекс:

- созданный сервером по столбцам с ограничениями PRIMARY KEY, FOREIGN KEY, UNIQUE*)*;
- используемый в данный момент в других запросах;
- нет соответствующих соответствующие привилегии доступа к БД.

Кроме того, при восстановлении БД из архивной копии, в которой хранится лишь определение индекса, а не его данные, индекс будет построен заново.

Четвертый способ улучшить производительность индекса — собрать статистику по индексу при помощи команды

SET STATISTICS INDEX <index_name>

Статистика таблицы — это величина в пределах от 0 до 1, значение которой зависит от числа различных (неодинаковых записей в таблице). Оптимизатор использует статистику для определения эффективности применения того или иного индекса в запросе.

SET STATISTICS не перестраивает индекс, поэтому свободен от ограничений налагаемых на ALTER|DROP INDEX, за исключением того что пересчет статистики имеет право выполнять либо создатель индекса, либо администратор.

ПРЕДСТАВЛЕНИЯ

В БД может быть определено представление (просмотр) — виртуальная таблица, в которой представлены записи из одной или нескольких таблиц. Представление реализовано как запрос, хранящийся на сервере и выполняющийся всякий раз, когда происходит обращение к представлению.

Для создания представления применяется оператор

CREATE VIEW <view_name> [(column_view1 [,column_view2 ...])]

AS <select_statement> [WITH CHECK OPTION];

в котором после имени представления следует необязательный список столбцов, оператор <select> есть полнофункциональный оператор SELECT.

вертикальный срез таблицы	горизонтальный срез таблицы
CREATE VIEW movie_vert AS	CREATE VIEW movie_hor AS
SELECT title, kind	SELECT * FROM movie
FROM movie;	WHERE year = 1990;

После этого к нему можно обращаться как к обычной таблице БД:

SELECT * FROM movie_vert;

В том числе и для построения другого представления:

CREATE VIEW V1 AS

SELECT title FROM movie_vert WHERE kind='комедия';

Для удаления представления используется оператор

DROP VIEW <view_name>;

Модифицировать представление нельзя. Можно только удалить и создать его заново.

Порядок формирования записей в представлении определяется оператором SELECT, а точнее — оптимизатором запросов.

В случае если список имен столбцов опущен, имена столбцов считаются идентичными именам полей, возвращаемых оператором SELECT.

Если в операторе SELECT столбец определяется как выражение, тогда список столбцов обязателен.

Список столбцов будущего представления содержит только их имена. Их тип будет таким, как у столбцов, возвращаемых оператором SELECT. Определяющим т.о. является только соответствие количества столбцов, возвращаемых SELECT-ом и перечисленных в CREATE VIEW.

1. Создать просмотр, содержащий имя, дату рождения, адрес всех работников кинобизнеса:

```
CREATE VIEW all_persons (fio, bd, adres, rank) AS
    SELECT fio, bd, adres, 'S' FROM prod
      UNION
    SELECT fio, bd, adres, 'P' FROM star
```

2. Создать просмотр, в котором бы для каждого года 21 века содержалось бы количество фильмов, выпущенных в этом году:

```
CREATE VIEW V2 (Y, C) as
  SELECT year, count(*)
  FROM movie
  WHERE year >1999
  GROUP BY year
```

При создании представлений НЕЛЬЗЯ использовать фразу ORDER BY. Сортировки можно добиться только в запросе, использующим данное представление.

Кроме того, в качестве источника данных для формирования представления НЕ МОГУТ выступать хранимые процедуры.

Преимущества создания просмотров:

- сложные запросы, выбирающие данные из множества таблиц становятся проще для понимания, снижая вероятность ошибки;
- просмотр может предоставлять ограниченный набор данных, что важно для обеспечения сохранности данных и, возможно, усиления безопасности: пользователь не будет иметь никакого представления о таблицах, лежащих в основе;
- дав пользователю в распоряжение просмотры, а не таблицы, можно менять запрос и сами таблицы, лежащие в основе просмотра, не меняя клиентских приложений.

Коль скоро представление – это виртуальная таблица, то к нему, как к любой таблице могут применятся операторы INSERT, UPDATE и DELETE. Но представление формируется на основе данных физической таблицы, то изменение данных просмотра невозможно без

соответствующего изменения исходных данных. А это не всегда возможно сделать.

Обновляемым (модифицируемым) считается просмотр для который позволяет изменение своих данных. А для этого необходимо выполнение следующих условий:

- просмотр должен формироваться из записей только одной таблицы;
- в просмотр должен быть включен каждый столбец таблицы, имеющий атрибут NOT NULL*;*
- оператор SELECT просмотра не должен использовать статистических функций, режима DISTINCT, предложения HAVING, соединения таблиц, хранимых процедур и функций, определенных пользователем.

В следующем просмотре:

```
CREATE VIEW notupdate AS
SELECT title, kind FROM movie;
```

нельзя добавлять записи, т.к. при добавлении в новой записи окажутся пустыми остальные поля, определенные как NOT NULL. Изменять и удалять записи из такого просмотра можно.

В следующем просмотре нельзя добавлять, корректировать и удалять записи, т.к. он соединяет две таблицы:

```
CREATE VIEW A AS
SELECT m.title, p.fio
FROM movie m, prod p
WHERE m.id_pr = m.id_pr;
```

Вышеперечисленные условия, ограничивающие модификацию представления, можно обойти используя триггеры.

Если для обновляемого просмотра указан параметр CHECK OPTION, будут отвергаться все попытки добавления новых или изменения существующих записей таким образом, чтобы нарушалось условие WHERE оператора SELECT данного просмотра.

В следующий просмотр нельзя добавить записи со значением поля money более 100 000:

```
CREATE VIEW money_check AS
SELECT *
FROM star
WHERE money <= 100000 WITH CHECK OPTION;
```

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Создать базу данных под управлением SQL-сервера Firebird. Создать таблицы, ограничения, вычисляемые столбцы, просмотры предметной области в соответствии с вариантом аналогично приведенному примеру.

ВАРИАНТЫ ЗАДАНИЙ

1. Географические объекты. СТРАНЫ (название, площадь, население, столица), ГОРОДА (название, страна, население, географические координаты: широта, долгота), РЕКИ и ОЗЕРА (тип, название, длина, макс. глубина, в каких странах находится).

2. Протоколы нарушения правил дорожного движения. ПРОТОКОЛ (номер, дата, место совершения, нарушитель, инспектор, автомобиль, нарушенные статьи КоАП, объяснение нарушителя, смягчающие и отягчающие обстоятельства), ИНСПЕКТОРА (ФИО, звание, дата рождения), КоАП (статья, часть, пункт, наименование, меры взыскания)

3. Субъекты и города Российской Федерации. СУБЪЕКТ (код региона, название, тип /область, край, автономный округ/, столица, губернатор, население, площадь, с какими субъектами граничит) НАСЕЛЕННЫЙ ПУНКТ (наименование, телефонный код, тип населенного пункта, население, регион, мэр)

4. Автобусные маршруты. МАРШРУТ (номер маршрута, номер пункт и время отправления, пункт и время прибытия, время в пути, промежуточные пункты с указанием времени прибытия и отправления) НАСЕЛЕННЫЕ ПУНКТЫ (наименование, наличие авто- и ж/д вокзала, аэропорта, гостиниц)

5. Регистрация автотранспортных средств. ТРАНСПОРТНОЕ СРЕДСТВО /ТС/ (номер паспорта, производитель, модель, кузов, двигатель, цвет, мощность и др. возможные характеристики ТС) СВИДЕТЕЛЬСТВА О РЕГИСТРАЦИИ /СоР/ (номер свидетельства, ТС, государственный регистрационный номер, владелец /ФИО, адрес/,

должностное лицо, выдавшее СоР) УЧЕТ СоР (номер свидетельства, дата выдачи или изъятия)

6. Банковские счета. СЧЕТ (номер, владелец счета, дата открытия, сумма, сотрудник, открывший счет) ОПЕРАЦИИ (дата, счет, сумма, сотрудник, проводивший операцию) СОТРУДНИКИ (ФИО, адрес, телефон, дата приема на работу) КЛИЕНТЫ (ФИО, адрес, паспортные данные и др.)

7. Телефонная компания. АБОНЕНТ (номер телефона, тарифный план, сумма на счете, ФИО владельца, дата приобретения) ЗВОНКИ (исходящий номер, входящий номер, время, длительность) ОПЕРАЦИИ (пополнение счета, списание средств /звонок, SMS/, сумма, дата)

8. Учебные планы. УЧЕБНЫЙ ПЛАН (специальность, семестр, даты начала/окончания семестра, начала/окончания сессии, начала/окончания каникул). ДИСЦИПЛИНЫ (наименование, количество часов лекций, упражнений, лабораторных, форма контроля /экзамен, зачет, дифференцированный зачет/, читающая кафедра, учебный план к которому относится дисциплина). КАФЕДРЫ (шифр, наименование, зав. кафедрой). ГРУППЫ (специальность, номер, выпускающая кафедра)

9. Прокат. ОБЪЕКТЫ (наименование, категория, страховочная стоимость, дата ввода в эксплуатацию, стоимость аренды в сутки/в месяц, характеристики, процент износа) КАТЕГОРИИ (наименование, срок эксплуатации) ДОГОВОРА (номер, дата, данные клиента, объекты, взятые в прокат и на какой срок)

10. Учет основных средств предприятия. ОБЪЕКТЫ (наименование, категория, балансовая стоимость, дата ввода в эксплуатацию, характеристики, материально-ответственное лицо / МОЛ) КАТЕГОРИИ (наименование, процент ежемесячно начисляемого износа) МОЛ (ФИО, должность, дата приема на работу) ИЗНОС (объект, дата начисления, сумма)

11. Библиотечный абонемент. КНИГИ (ISBN, автор, наименование, объем, переплет, стоимость) ЧИТАТЕЛИ (ФИО, паспортные данные, телефон, дата регистрации) ЖУРНАЛ (книга, читатель, дата выдачи, срок)

12. Оплата кредита. КЛИЕНТ (ФИО, паспортные данные, данные работы, доход) КРЕДИТ (сумма, когда, кому, под какой процент, на

какой срок выдан, назначение) ОПЛАТА (дата, сумма, кредит, плательщик)

13. Ж/д билеты БИЛЕТ (поезд, дата, станция и время отправления, станция и время назначения, вагон, место) ПОЕЗД (номер, пункт и время отправления, пункт и время прибытия, кол-во вагонов люкс/купе/плацкарт) СТАНЦИИ (наименование, населенный пункт, область)

14. Путевки. КЛИЕНТЫ (ФИО, паспортные данные, адрес, телефон) ОТЕЛЬ (страна, название, адрес, телефон, управляющий, сведения о номерах, бассейнах и т.п.) ПУТЕВКИ (клиент, отель, даты заезда/отъезда, номер)

15. Заявки ЖЭКа. ДОМА (улица, номер, этажность, дата ввода в эксплуатацию) КВАРТИРЫ (дом, номер, подъезд, этаж, ответственный квартиросъемщик) ЖИЛЬЦЫ (ФИО, квартира, дата прописки, д/р) ЗАЯВКА (Когда и кем из жильцов подана заявка, содержание, исполнитель заявки, срок исполнения)

16. Складской учет. ТОВАРЫ (артикул, наименование, производитель) ПРИХОД (товар, количество) РАСХОД (товар, количество)

17. Учет заказов на изготовление книжной продукции. Книга (название, автор, издательство, формат, объем, тираж, переплет) изготавливается тремя рабочими (наборщик, печатник, переплетчик) из материалов (основная бумага, бумага переплета).

18. Таксопарк. Учет заказов на такси. (время приема заказа, номер телефона клиента, адрес, водитель, автомобиль).

19. Театральные кассы. Какие спектакли, где идут, кто режиссер, актерский состав, сведения о билетах в разные категории мест (партер, амфитеатр, балкон).

20. ОСАГО. Фиксируется факт ДТП (время, место, участники, обстоятельства, объяснения и т.д.), данные страховых полисов участников ДТП (страховая компания, номер полиса).

21. Кадровое агентство (биржа труда). Человек (м.б. безработный, м.б. нет) подает резюме, в котором указывается — помимо очевидных атрибутов — стаж, последнее место работы и должность, последняя зарплата, специальность и т.д. Работодатель составляет вакансии: место и должность, з/п, особенности работы и т.д. В специальной таблице нужно вести подбор вакансий и резюме по специальности, по зарплате и т.д.

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Перечислите основные типы данных в СУБД Firebird.
2. Перечислите виды ограничений.
3. Перечислите способы изменения подчиненных записей дочерней таблицы при удалении или изменении поля связи в записи родительской таблицы.
4. Дайте определение термину «домен».
5. Приведите синтаксис оператора для создания домена.
6. Приведите операторы, которые необходимо использовать для создания базы данных.
7. Дайте определение термину «индекс».
8. Приведите синтаксис оператора для создания индекса.
9. Дайте определение термину «представление».
10. Приведите синтаксис оператора для создания представления.

ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

На выполнение лабораторной работы отводится 2 занятия (4 академических часа: 3 часа на выполнение и сдачу лабораторной работы и 1 час на подготовку отчета).

Номер варианта студенту выдается преподавателем.

Отчет на защиту предоставляется в печатном виде.

Структура отчета (на отдельном листе(-ах)): титульный лист, формулировка задания (вариант), ход выполнения работы, результаты выполнения работы, выводы.

ОСНОВНАЯ ЛИТЕРАТУРА

1. Карпова, Т.С. Базы данных: модели, разработка, реализация : учебное пособие / Т.С. Карпова. - 2-е изд., исправ. - Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 241 с. : ил. ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429003

2. Давыдова, Е.М. Базы данных [Электронный ресурс] : учеб. пособие / Е.М. Давыдова, Н.А. Новгородова. — Электрон. дан. — Москва : ТУСУР, 2007. — 166 с. — Режим доступа: https://e.lanbook.com/book/11636. — Загл. с экрана.

3. Харрингтон, Д. Проектирование объектно ориентированных баз данных [Электронный ресурс] — Электрон. дан. — Москва : ДМК Пресс, 2007. — 272 с. — Режим доступа: https://e.lanbook.com/book/1231. — Загл. с экрана.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

4. Голицина О.Л., Максимов Н.В., Попов И.И. Базы данных: учеб. пособие. – М.: Форум:Инфра-М, 2007.

5. Гагарин Ю.Е. Применение языка SQL в MS Access: учебно-методическое пособие. – М.: МГТУ им. Н.Э. Баумана, 2012.

Электронные ресурсы:

1. Научная электронная библиотека http://eLIBRARY.RU
2. Электронно-библиотечная система http://e.lanbook.com
3. Электронно-библиотечная система «Университетская библиотека онлайн» http://biblioclub.ru
4. Электронно-библиотечная система IPRBook http://www.iprbookshop.ru