Лекція 10-2020_2021

- На попередній лекції
- Сортування записів, ORDER BY
- Оператор UNION
- Створення бази даних, корегування
- Створення таблиць

Что было в предыдущей лекции

В разделе **FROM** источники данных могут быть объединены.:

1. За счет введения условия в **Where**.
2. За счет использования JOIN:

– INNER JOIN – объединение по связующему полю, если оно содержит одинаковые значения в обоих источниках данных.

– LEFT JOIN – все записи из первого источника, а из второго только те, которые удовлетворяют условию объединения.

– RIGHT JOIN – все записи из второго источника, а из первого только те, которые удовлетворяют условию объединения.

– FULL JOIN – внутреннее соединение двух источников по связующему полю + записи из первого источника, не вошедшие во внутреннее соединение + записи из второго источника, не вошедшие во внутреннее соединение.

– CROSS JOIN – расширенное декартово произведение.

Что было в предыдущей лекции

NULL – специальное значение, указывающее на то, что мы не знаем, какое значение.

Оператор **IS NULL** используется для сравнения текущего значения со значением **NULL**

IS NOT NULL используется для проверки присутствия значения в поле.

Рассмотрели использование итоговых функций:

Count – количество записей в выходном наборе запроса;

Min/Max – наименьшее или наибольшее из множества значений;

Avg – среднее значение множества значений;

Sum – сумма множества значений, содержащихся в определенном поле отобранных запросом записей.

Что было в предыдущей лекции

Раздел GROUP BY команды SELECT объединяет записи с одинаковыми значениями в указанном списке полей, используется для итоговых запросов.

1. Если в оператор SELECT включено GROUP BY, список выбора должен состоять из выражений с итоговыми функциями или/и из столбцов, указанных в GROUP BY
2. Значения NULL, которые находятся в полях, заданных в предложении GROUP BY группируются и не опускаются, но статистические функции не обрабатывают значение NULL.

Что было в предыдущей лекции

HAVING − условие отбора в группы (фильтр). Применяется только к столбцам, указанным в GROUP BY, к столбцам итоговых функций и к выражениям, содержащим итоговые функции.

ORDER BY −необязательный параметр SELECT, позволяет отсортировать результирующую выборку по значениям одного или более столбцов.

Пример GROUP BY

Пример 1. Задача. Есть упрощенный вариант двух таблиц. Спец {КодС, Спец} Врач {КодВ, КодС, Фам, Стаж}
Запрос на объединение двух таблиц

```
SELECT Спец.Спец, Врачи.Фам, Врачи.Стаж
FROM Спец INNER JOIN Врачи
ON Спец.КодС = Врачи.КодС;
```

Результат сохраним, как **Врач_ст**

Спец	Фамилия	Стаж
Хирург	Попов	20
Терапевт	Котова	16
Хирург	Петров	18
Терапевт	Петрова	10
Невропатолог	Сидоров	10
Невропатолог	Иванова	15

GROUP BY. Почему одинаковые результаты?

По каждой специализации вывести фамилии врачей-ветеранов по стажу. Ветераны те, у кого стаж > 15 лет.

Спец	Фамилия	Стаж
Хирург	Попов	20
Терапевт	Котова	16
Хирург	Петров	18
Терапевт	Петрова	10
Невропатолог	Сидоров	10
Невропатолог	Иванова	15

Результат

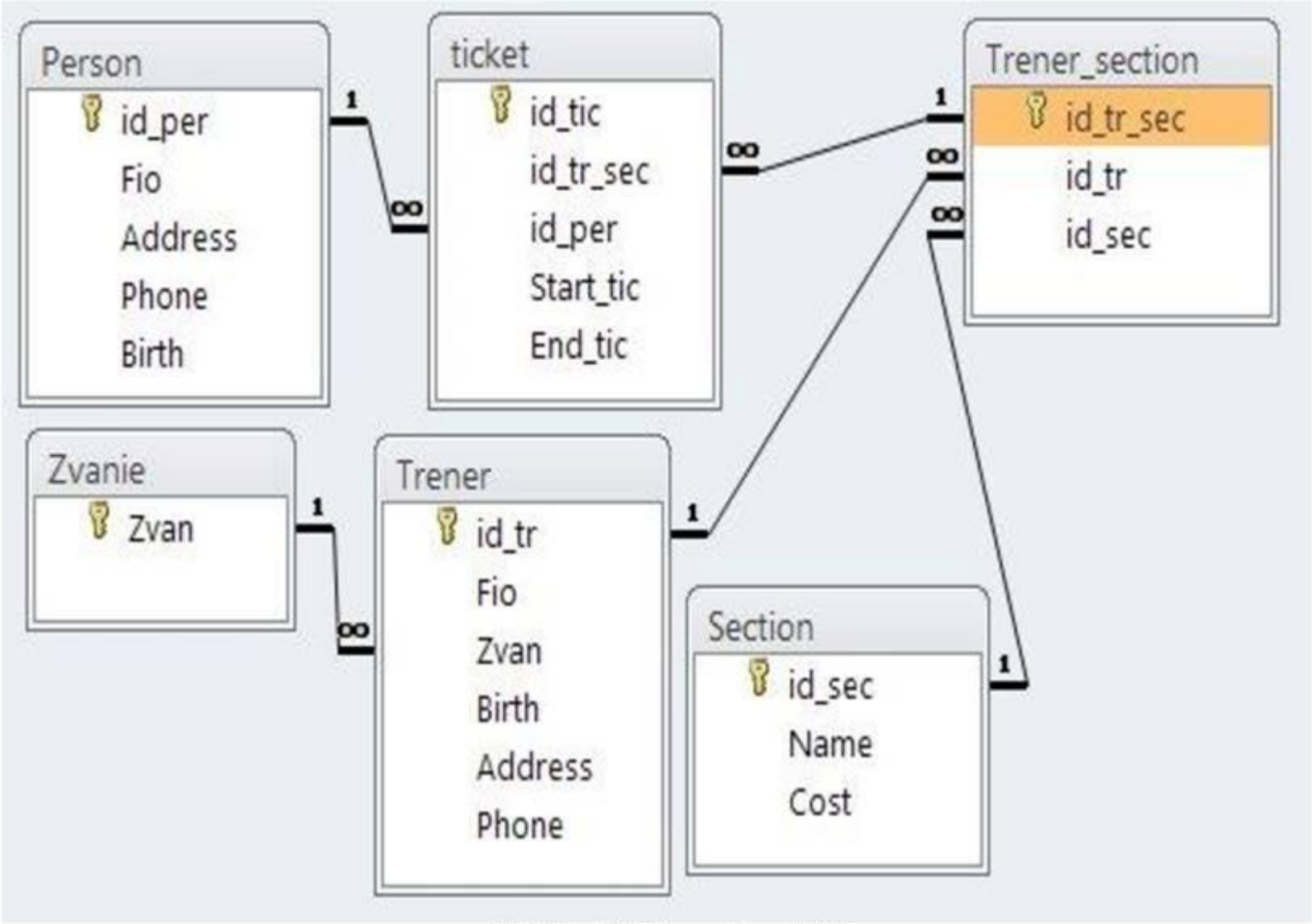
Спец	Фамилия	Стаж
Невропатолог	Иванова	15
Терапевт	Котова	16
Хирург	Попов	20
Хирург	Петров	18

а)
```
SELECT Спец, Фам, Стаж
FROM Врач_ст
GROUP BY Спец, Фам, Стаж
HAVING (Max(Стаж) >=15);
```

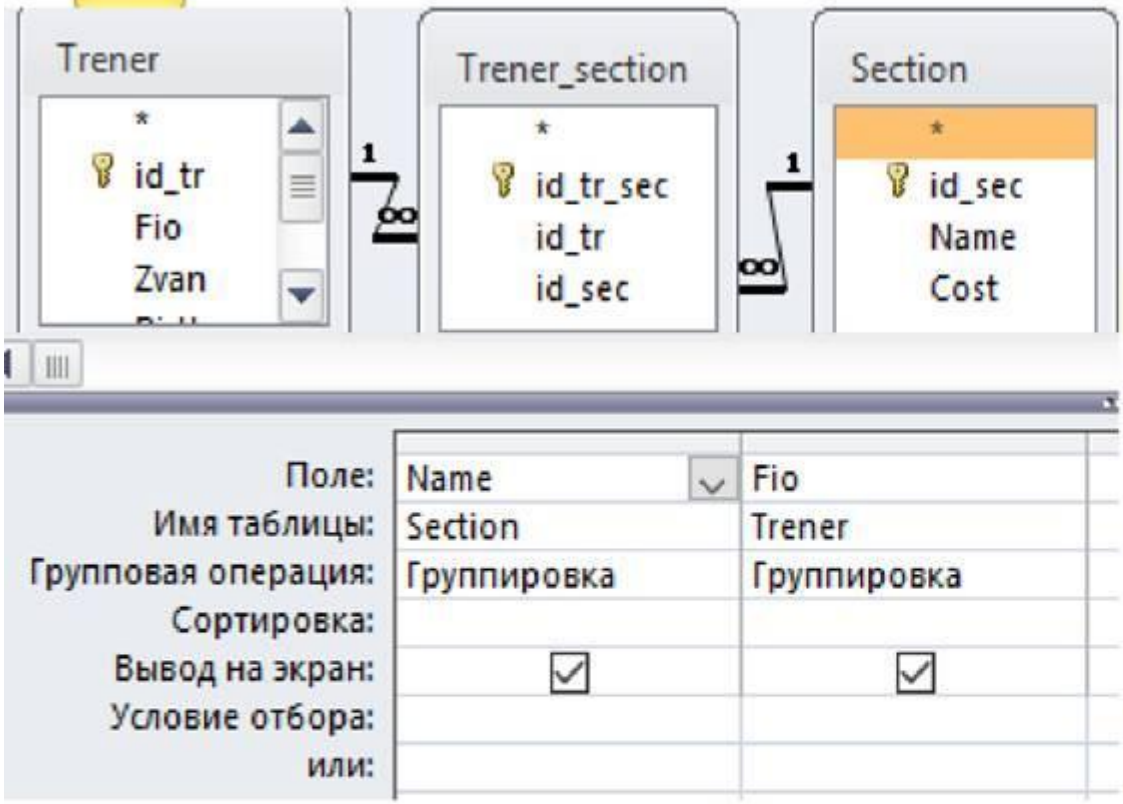
б)
```
SELECT Спец, Фам, Стаж
FROM Врач_ст
GROUP BY Спец, Фам, Стаж
HAVING (Стаж >=15);
```

в)
```
SELECT Спец,Фам, Стаж
FROM Врач_ст
WHERE (Стаж >=15)
GROUP BY Спец, Фам,Стаж;
```

Пример GROUP BY. БД Спорт клуб



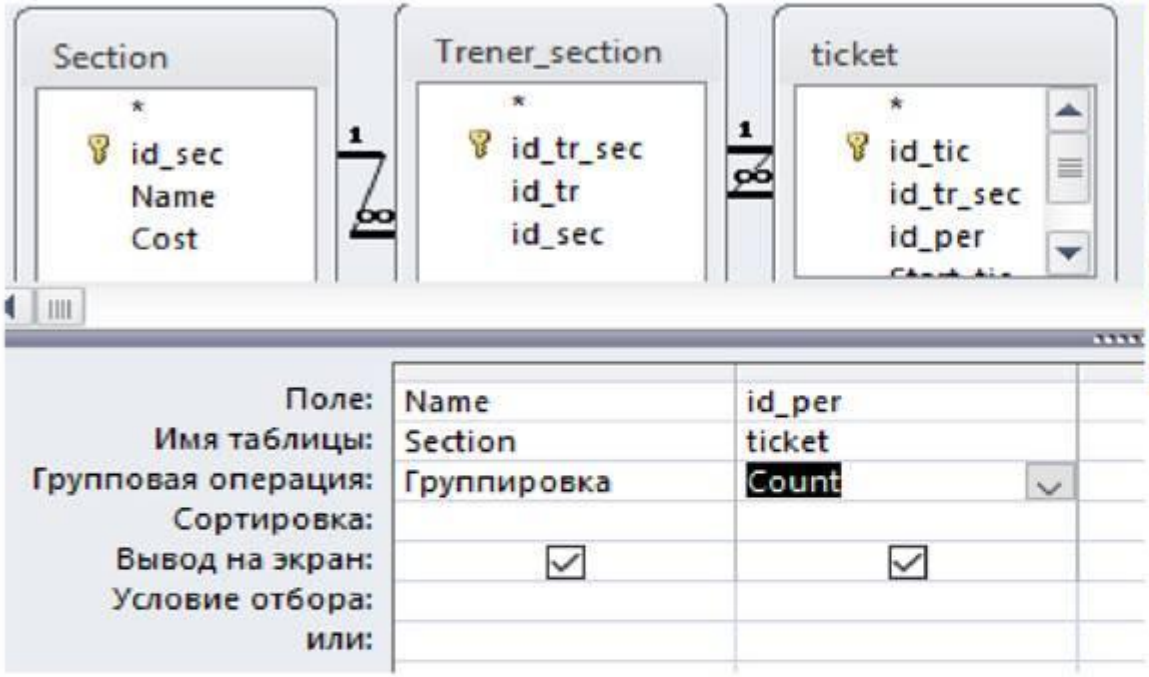
Тренеры по секциям с группировкой



Name	Fio
аэробика	Иванов И.И.
аэробика	Максимова А.П.
бег на месте	дон Румата
волейбол	Карась В.Н.
волейбол	Максимова А.П.
плавание с кругом	дон Румата

```
SELECT Name, Fio FROM Section INNER JOIN (Trener
INNER JOIN Trener_section
ON Trener.id_tr = Trener_section.id_tr)
ON Section.id_sec = Trener_section.id_sec
GROUP BY Section.Name, Trener.Fio;
```

Количество посетителей в каждой секции за весь период времени



Name	Count-id_per
аэробика	8
бег на месте	1
волейбол	3
плавание с кругом	11

```
SELECT Name, Count(ticket.id_per) AS [Count-id_per]
FROM (Section INNER JOIN Trener_section ON
Section.id_sec = Trener_section.id_sec) INNER JOIN
ticket ON Trener_section.id_tr_sec =
ticket.id_tr_sec
GROUP BY Name;
```

Сортировка записей запроса, ORDER BY

ORDER BY –необязательный параметр SELECT, позвопозволяет отсортировать результирующую выборку по значениям одного или более столбцов. Применяется как к числовым столбцам, так и к строковым. Без ORDER BY СУБД возвращает строки в любом порядке. Параметры: ASC (по умолчанию) – по возрастанию, DESC – сортировка по убыванию.

Пример 2.

```
SELECT * FROM t1 ORDER BY key1 ASC;
```

Пример 3.

```
SELECT * FROM t1 ORDER BY key1 DESC;
```

Пример 4.

```
SELECT * FROM t1 ORDER BY key1 DESC, key2 DESC;
```

Пример 5.

```
SELECT * FROM t1 ORDER BY key2 DESC, key1;
```

Оператор UNION

```
SELECT ...
UNION [ALL]
SELECT ...
[UNION [ALL ]
SELECT ...]
```

UNION используется, чтобы объединить в один простой результирующий набор результаты нескольких операторов **SELECT**.

Правила для объединения запросов

1. Столбцы вывода объединяемых запросов должны быть совместимы для объединения. Это означает:

–каждый запрос имеет одинаковое количество столбцов;

– столбцы в запросах должны иметь тип, совместимый с каждым.

– в стандарте символьные поля должны иметь одинаковое количество символов. Большинство СУБД разрешают поля переменной длины, но они могут быть запрещены для использования с UNION. Требуется консультация с документацией по СУБД.

Правила для объединения запросов

2. Пустые значения (NULL) запрещены в любом столбце объединения. Пустые значения (NULL) запрещены ограничением NOT NULL.

3. Нельзя использовать UNION в подзапросах.

4. Нельзя в объединяемых операторах SELECT использовать агрегатные функции. (Большинство СУБД разрешают).

UNION будет автоматически исключает дубликаты строк из вывода.

5. Каждый из объединяемых запросов выполняется отдельно и независимо, результаты объединяются.

UNION при выборке из двух таблиц

Даны две таблицы:

sales2015

person	amount
Иван	1000
Алексей	2000
Сергей	5000

sales2016

person	amount
Иван	2000
Алексей	2000
Петр	35000

UNION при выборке из двух таблиц

```
(SELECT *
FROM sales2015)
UNION
(SELECT *
FROM sales2016);
```

person	amount
Иван	1000
Алексей	2000
Сергей	5000

Иван 2000
Петр 35000

Две строки с Иваном, (различаются значениями в amount) столбцах. Одна строка с Алексеем, поскольку исключен дубль.

UNION ALL при выборке из двух таблиц

```
(SELECT *
FROM sales2015)
UNION ALL
(SELECT *
FROM sales2016);
```

person	amount
Иван	1000
Алексей	2000
Сергей	5000
Иван	2000
Алексей	2000
Петр	35000

Вычисляемые выражения в объединении

БД хранит сведения по дорогостоящим (Main) и малоценным (Little) приборам, числящимися за отделами (Department).

Department : таблица

Имя поля	Тип данных
Otd	Числовой
Name	Текстовый
Shef	Текстовый
Mat	Текстовый

Main1 : таблица

Имя поля	Тип данных
Otd	Числовой
Number	Текстовый
NameDevice	Текстовый
DateTo	Дата/время
Add	Числовой
DateOff	Дата/время
Remove	Числовой
Price	Числовой

Вычисляемые выражения в объединении

Сформировать запрос, который показывает количество принятых единиц приборов и сумму приема по каждому отделу по обеим категориям приборов.

Отдел	Принято	На суммуП	Категория
11	6	384	Малоценные
11	11	8113	Дорогостоящие
12	4	723.33	Дорогостоящие
13	7	4682	Дорогостоящие
14	33	894	Дорогостоящие
15	3	221	Малоценные
16	4	304	Малоценные
17	3	235.8	Малоценные
31	4	196.4	Малоценные
37	8	13516.5	Дорогостоящие

Вычисляемые выражения в объединении

SQL:

```
SELECT Otd , Sum(Add), Round(Sum(Add*Price),2)
  as [На суммуП], "Дорогостоящие" As Категория

      FROM Main  GROUP BY Otd

UNION

SELECT Otd , Sum(Add), Round(Sum(Add*Price),2)
  as [На суммуП], "Малоценные"

     FROM Little GROUP BY Otd;
```

Создание базы данных

```
CREATE DATABASE [IF NOT EXISTS] db_name
[ [ DEFAULT ] CHARACTER SET character_ name ]
[ [ DEFAULT ] COLLATE collation_ name ]
```

CREATE DATABASE создает базу данных с указанным именем. Если база данных уже существует и не указан ключевой параметр **IF NOT EXISTS**, то возникает ошибка выполнения команды.

CHARACTER SET – кодировка, **COLLATE** – сравнение.

MySQL : С версии 5.5. по умолчанию кодировка – **UTF8**; До версии 5.5. по умолчанию кодировка – **latin1**,режим сравнения - **latin1_swedish_ci**.

суффикс **_ci** (регистронечувствительный), **_cs** (регистрозависимый), **_bin** (двоичный).

Создание базы данных

Если база данных создается в Unix, имя будет регистрозависимым:

`CREATE DATABASE menagerie` и

`CREATE DATABASE Menagerie` создают разные базы данных.

Но даже в случае Unix ключевые слова SQL не чувствительны к регистру.

То же самое касается имен таблиц. На Windows это ограничение не распространяется. Если при создании БД получено сообщение

Access denied for user 'misha'@'localhost' to database 'menagerie' в момент создания БД –это означает, что пользователь **misha** не имеет привилегий на создание БД.

Использование базы данных

Создание БД не выделяет ее для текущего использования, это необходимо сделать отдельно. Оператор **`USE menagerie`**; делает БД текущей. Для того, чтобы увидеть, какая БД текущая используется **`SELECT DATABASE()`**.

Изменение структуры базы данных

```
ALTER DATABASE db_name
[ [ DEAULT ] CHARACTER SET character_ name ]
[ [ DEFAULT ] COLLATE collation_ name ]
```

Создание таблицы

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]
tbl_name  [(create_definition,...)]

          [table_options] [select_statement] ;

CREATE [TEMPORARY] TABLE [IF NOT EXISTS]
tbl_name   LIKE old_tbl_name ;
```

create_definition:

```
  col_name type [NOT NULL | NULL]

  [DEFAULT default_value]

  [AUTO_INCREMENT]

  [PRIMARY KEY]

  [UNIQUE (index_col_name,...)]

  [CHECK (expr)]
```

Пустые и непустые значения атрибутов

Пример 1.

Создать таблицу Продавцов (**SalPeople**), табельный номер **snum** и имя **sname** не могут иметь пустое значение (NULL) :

```
CREATE TABLE SalPeople
( snum integer NOT NULL,
  sname  char (10) NOT NULL,
  city  char (10),
  comm   decimal);
```

Уникальные значения и первичный ключ

Пример 2. **PRIMARY KEY** определяет поле первичного ключа. Первичные ключи не могут быть NULL. Любое поле, объявленное как **PRIMARY KEY** должно быть **NOT NULL**. **PRIMARY KEY** может также быть применено для многочисленных полей, составляющих уникальный ключ.

А)
```
CREATE TABLE SalPeople
(snum integer NOT NULL PRIMARY KEY,
 sname char(10) NOT NULL UNIQUE,
 city char(10),  comm  decimal);
```

Б)
```
CREATE TABLE  Name
(firstname  char (10) NOT NULL,
 lastname  char (10) NOT NULL,
 city char (10),
 PRIMARY KEY  (firstname, lastname ));
```

Условие на ввод

Пример 3. Ограничение **CHECK** позволяет установить условие, которому должно удовлетворять значение, вводимое в таблицу, прежде чем оно будет принято. Любая попытка модифицировать или вставить значение поля, которое могло бы сделать этот предикат неверным, отклоняется.

Предположим, что комиссионные 15% и выше, будут разрешены только для продавца из Харькова.

```
CREATE TABLE SalPeople
( snum integer NOT NULL PRIMARY KEY,
   sname  char (10) NOT NULL UNIQUE,
   city char(10),
   comm  decimal,
   CHECK     (comm < .15 OR city = 'Харьков'));
```

Значения по умолчанию

Пример 4. При вставке строки в таблицу без указания значений в ней для каждого поля, SQL должен иметь значение по умолчанию, или же команда будет отклонена. Общим значением по умолчанию является NULL. Можно указать Харьков в качестве значения поля **city** по умолчанию для таблицы **SalPeople** :

```
CREATE TABLE SalPeople
 ( snum  integer NOT NULL PRIMARY KEY,
   sname char(10) NOT NULL UNIQUE,
   city char(10) DEFAULT = 'Харьков',
   comm decimal,
   CHECK    (comm < .15 OR city = 'Харьков'));
```

Значения по умолчанию

Величина **DEFAULT** должна быть константой, она не может быть функцией. Нельзя установить для столбца даты в качестве значения по умолчанию величину функции, такой как **NOW()** или **CURRENT_DATE**.

Если столбец может принимать NULL как допустимую величину, то по умолчанию присваивается значение NULL.

Если столбец объявлен как NOT NULL, то значение по умолчанию зависит от типа столбца: для числовых типов (кроме AUTO_INCREMENT) равно **0,** для AUTO_INCREMENT - следующее значение в последовательности для этого столбца, для типов даты и времени равно нулевой величине данного типа, для строковых типов - пустая строка

Автоинкрементное значение ключа и внешний ключ

Пример 5. Когда поле является внешним ключом, оно связано с таблицей, на которую он ссылается.

Создадим таблицу **Customers** с полем **snum** в качестве внешнего ключа, ссылающегося на таблицу **SalPeople** :

```
CREATE TABLE Customers
( custnum    integer NOT NULL
             AUTO_INCREMENT PRIMARY KEY ,
  firm   char(10),
  city    char(10),
  snum integer,
  FOREIGN KEY (snum)
    REFERENCES SalPeople   ( snum );
```

Автоинкрементное значение ключа и внешний ключ

Целочисленный столбец может быть **AUTO_INCREMENT**.
При записи величины NULL (рекомендуется) или 0 в столбец **AUTO_INCREMENT** данный столбец устанавливается в значение **`value+1`**, где **`value`** представляет собой наибольшее для этого столбца значение в таблице на момент записи.
Последовательность **AUTO_INCREMENT** начинается с 1.

Временные таблицы

При создании таблицы можно использовать ключевое слово **TEMPORARY**.

Временная таблица автоматически удаляется по завершении соединения, а ее имя действительно только в течение данного соединения. Это означает, что в двух разных соединениях могут использоваться временные таблицы с одинаковыми именами без конфликта друг с другом.

Можно использовать ключевые слова **IF NOT EXISTS** для того, чтобы не возникала ошибка, если указанная таблица уже существует.

При создании временных таблиц с **IF NOT EXISTS** не проверяется идентичность структур этих таблиц.

Создание одной таблицы из другой

1. В MySQL можно указать **LIKE,** чтобы создать таблицу, основываясь на определении другой, уже существующей таблицы. При создании таблицы можно использовать ключевое слово **TEMPORARY**.

Пример 6.
```
CREATE TEMPORARY TABLE
    IF NOT EXISTS TempSalPeople
    Like SalPeople
```

2. Можно создавать одну таблицу из другой, добавляя **SELECT** в конце **CREATE TABLE** оператора.

Пример 7. Создадим MySQL таблицу со столбиками **a, b, c**.

```
CREATE TABLE test
  (a INT NOT NULL AUTO_INCREMENT,
   PRIMARY KEY (a)) SELECT b,c FROM test2;
```

Пусть **SELECT b,c FROM test2;** → 1, 2

Тогда **SELECT * FROM test;** → NULL , 1, 2

Создание одной таблицы из другой

MySQL.

Пример 8. Есть таблица **artist** с личными данными артистов и таблица **work** со сведениями о местах работы артистов. Таблицы связаны **artist.id → work.artist_id**

```
CREATE TABLE artists_and_works
   SELECT artist.name,
   COUNT(work.artist_id) AS number_of_works
   FROM artist LEFT JOIN work
    ON artist.id = work.artist_id
   GROUP BY artist.id;
```

Если создается таблица как **CREATE ... SELECT**,

для каждого вычисляемого поля должен быть назначен псевдоним, т.е. **COUNT(work.artist_id) AS number_of_works**.

Создание одной таблицы из другой

Access.

Пример 9.

```
SELECT aw.* INTO artists_and_works
 FROM
 (SELECT artist.name,
  COUNT(work.artist_id) AS number_of_works
  FROM artist LEFT JOIN work
   ON artist.id = work.artist_id
  GROUP BY artist.id) as aw;
```

Индексы

Индексы – это структуры данных, создаваемые с целью повышения производительности поиска записей в таблицах.

Индексы применяются для быстрого поиска строк с указанным значением одного столбца. Без индекса чтение таблицы осуществляется последовательно по всей таблице начиная с первой записи. Индекс дает возможность прямого доступа к нужной записи.

Все индексы MySQL (PRIMARY, UNIQUE, и INDEX) хранятся в виде B-деревьев.

В MySQL данные и индексы хранятся отдельно в разных файлах. В других БД данные и индексы помещаются вместе в одном и том же файле.

Индексы

Создание индекса: **CREATE INDEX** или **CREATE TABLE** . Обычно индекс создается при создании таблицы.

Пример 10. Создание одностолбцового индекса в MySQL.

```
CREATE TABLE test
   ( name CHAR(200) NOT NULL,
     KEY index_name (name(10))
   );
```

Многие СУБД позволяют создавать индексы по нескольким столбцам. В MySQL индекс может включать в себя до 15 столбцов. Многостолбцовый индекс содержит величины, созданные конкатенацией величин проиндексированных столбцов.

Во многих СУБД нельзя индексировать целиком столбец типа BLOB и TEXT.

Создание индекса по нескольким столбцам.

Пример 11.

```
CREATE TABLE test
    (id INT NOT NULL,
     last_name CHAR(30) NOT NULL,
     first_name CHAR(30) NOT NULL,
     PRIMARY KEY (id),
     INDEX name (last_name,first_name));
```

Индекс **name** является индексом по столбцам **last_name** и **first_name**. Этот индекс будет применяться для запросов, указывающих величины в известной области для **last_name** или для обоих столбцов **last_name** и **first_name**.

Использование многостолбцового индекса

Пример 12. Индекс name в запросах:

```
1. SELECT * FROM test WHERE
   last_name="Widenius";

2. SELECT * FROM test WHERE
   last_name="Widenius" AND
   first_name="Michael";

3. SELECT * FROM test WHERE
   last_name="Widenius" AND
    (first_name="Michael" OR
      first_name="Monty");
```

Рекомендации при использовании индексов

1. Не нужно индексировать все.

2. Перед созданием индекса необходимо вначале проанализировать все запросы к БД.

3. Чаще всего создают многостолбцовый индекс. В этом случае важен порядок перечисления столбцов индекса.

4. Для индексов по текстовым полям достаточно индексировать лишь начальную часть текстового значения.