

Севастопольский государственный университет
Кафедра «Информационные системы»

Управление данными

курс лекций

лектор:
ст. преподаватель кафедры ИС Абрамович А.Ю.



Лекция 11

Язык SQL. Представления. Триггеры

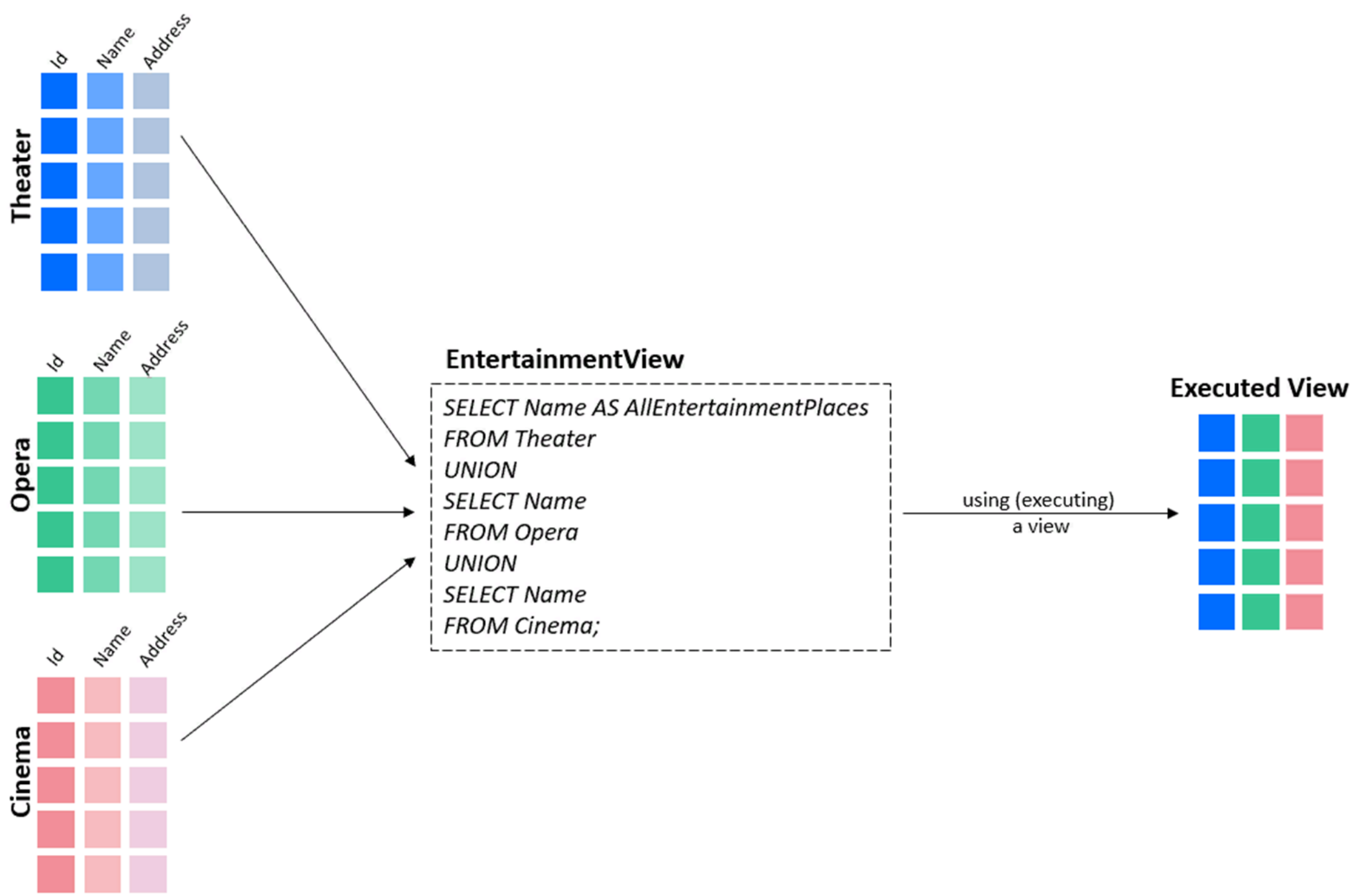
ПРЕДСТАВЛЕНИЯ

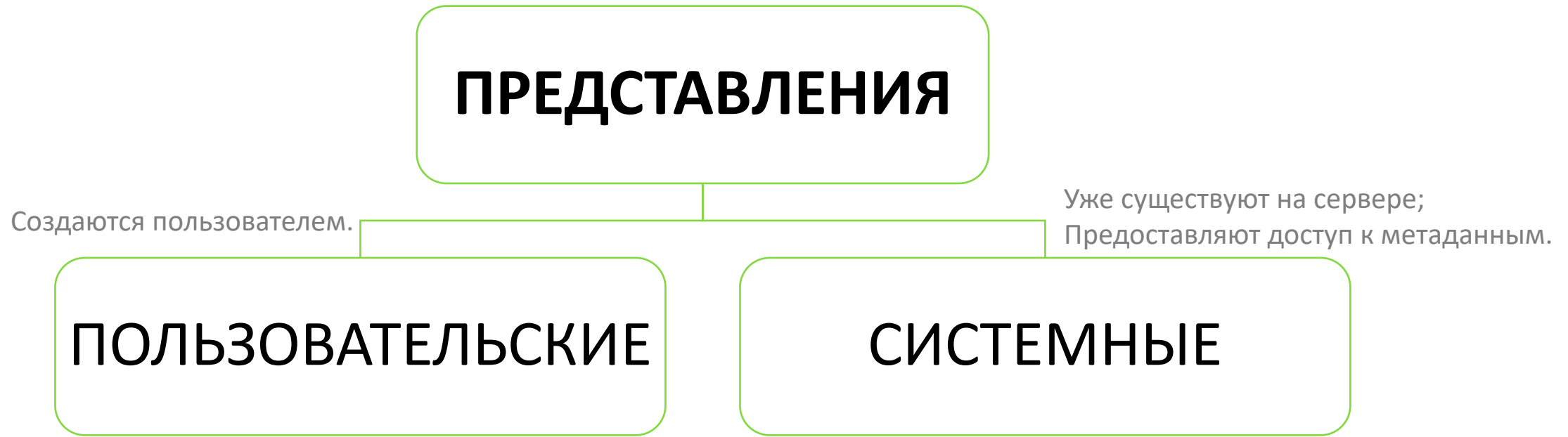
просмотры (VIEW), представляют собой временные, производные таблицы и являются объектами базы данных, информация в которых не хранится постоянно, как в базовых таблицах, а формируется динамически при обращении к ним.

Содержимое представлений выбирается из других таблиц с помощью выполнения запроса, причем при изменении значений в таблицах данные в представлении автоматически меняются.

Представление не может существовать само по себе, а определяется только в терминах одной или нескольких таблиц.

Применение представлений позволяет разработчику базы данных обеспечить каждому пользователю или группе пользователей наиболее подходящие способы работы с данными, что решает проблему простоты их использования и безопасности.





У СУБД ЕСТЬ ДВЕ ВОЗМОЖНОСТИ РЕАЛИЗАЦИИ ПРЕДСТАВЛЕНИЙ:

Если его *определение простое*, то система формирует *каждую запись представления по мере необходимости*, постепенно считывая исходные данные из базовых таблиц.

В случае *сложного определения* СУБД приходится *сначала выполнить такую операцию, как материализация представления*, т.е. сохранить информацию, из которой состоит представление, во временной таблице. Затем *система приступает к выполнению пользовательской команды и формированию ее результатов*, после чего временная таблица удаляется.

```
CREATE VIEW <имя_представления>  
[ (<имя_столбца>, ...) ] [WITH ENCRYPTION]  
AS <запрос> [WITH CHECK OPTION]
```

```
CREATE VIEW studs (fname, lname, number) AS  
SELECT fname, lname, number from students, groups  
WHERE students.group_id = groups.id
```

Параметр **WITH ENCRYPTION** предписывает серверу **шифровать SQL-код запроса**, что гарантирует невозможность его несанкционированного просмотра и использования.

Параметр **WITH CHECK OPTION** предписывает серверу **исполнять проверку изменений**, производимых через представление, на соответствие критериям, определенным в операторе SELECT. **Использование аргумента гарантирует, что сделанные изменения будут отображены в представлении.** Если пользователь пытается выполнить изменения, приводящие к исключению строки из представления, при заданном аргументе WITH CHECK OPTION сервер выдаст сообщение об ошибке и все изменения будут отклонены.

Представление удаляется командой:

```
DROP VIEW имя_представления [ , ... n]
```

Пример: показать в представлении клиентов из Севастополя.

Создание представления:

```
CREATE VIEW view1 AS  
SELECT КодКлиента, Фамилия, ГородКлиента  
FROM Клиент  
WHERE ГородКлиента='Севастополь'
```


Выборка данных из представления:

```
SELECT * FROM view1
```


Обращение к представлению осуществляется с помощью оператора SELECT как к обычной таблице.

Выполним команду:

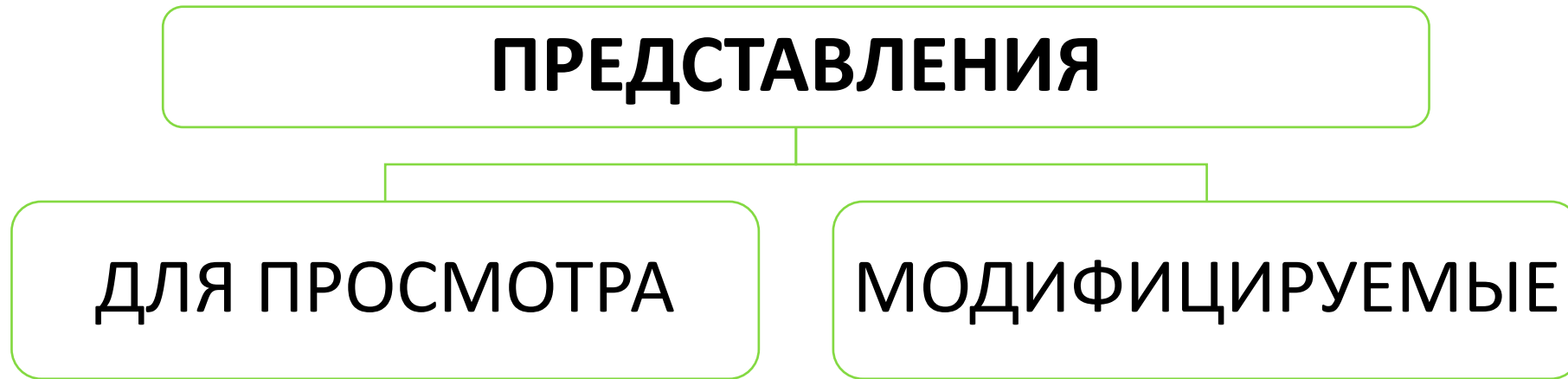
```
INSERT INTO view1 VALUES (12, 'Петров', 'Симферополь')
```



```
ALTER VIEW view1 AS  
SELECT КодКлиента, Фамилия, ГородКлиента  
FROM Клиент  
WHERE ГородКлиента='Москва' WITH CHECK OPTION
```



Представление *может изменяться командами модификации*, но фактически модификация воздействует не на само представление, а на базовую таблицу.



Не все представления в SQL могут быть модифицированы. С **модифицируемыми** представлениями в основном **обходятся точно так же, как и с базовыми таблицами**.

Представления **в режиме <только для чтения>** позволяют **получать и форматировать данные более рационально**. Они создают целый набор сложных запросов, которые можно выполнить и повторить снова, сохраняя полученную информацию. Результаты этих запросов могут затем использоваться в других запросах, что позволит избежать сложных предикатов и снизить вероятность ошибочных действий.

ОСОБЕННОСТИ ПРЕДСТАВЛЕНИЙ:

- основывается только *на одной* базовой таблице;
- содержит *первичный ключ* этой таблицы;
- не содержит **DISTINCT** в своем определении;
- не использует **GROUP BY** или **HAVING** в своем определении;
- по возможности *не применяет в своем определении подзапросы*;
- *не использует константы* или выражения значений среди выбранных полей вывода;
- в просмотр должен быть включен *каждый столбец таблицы*, имеющий атрибут **NOT NULL**;
- оператор SELECT просмотра *не использует агрегирующие (итоговые) функции, соединения таблиц, хранимые процедуры и функции*, определенные пользователем;
- основывается на одиночном запросе, поэтому объединение **UNION** *не разрешено*.

Если просмотр удовлетворяет этим условиям, к нему могут применяться операторы **INSERT, UPDATE, DELETE**.

Немодифицируемое представление с данными из разных таблиц.

```
CREATE VIEW view AS
SELECT Клиент.Фамилия, Клиент.Фирма,
       Сделка
FROM Клиент I
ON Клиент.Код
```

```
[SQL> create view view22 (number, orderdate, price) as select orderid, orderdate,
price*25 from orders;
[SQL> select * from view22;
```

NUMBER	ORDERDATE	PRICE
=====	=====	=====
3023	2023-10-03	1125
3027	2023-08-03	1125
3029	2023-02-15	1125
3234	2023-04-04	1125

Модифицируемое

```
CREATE VIEW v
SELECT КодТов
```

```
[SQL> insert into orders values (3323, 23, '2023-10-04', 76);
[SQL> select * from view22;
```

Тип, Цена,	NUMBER	ORDERDATE	PRICE
	=====	=====	=====
	3023	2023-10-03	1125
	3027	2023-08-03	1125
	3029	2023-02-15	1125
	3234	2023-04-04	1125
	3323	2023-10-04	1900

ПРЕИМУЩЕСТВА ПРЕДСТАВЛЕНИЙ

НЕЗАВИСИМОСТЬ ОТ ДАННЫХ

С помощью представлений можно создать согласованную, неизменяемую картину структуры базы данных, которая будет оставаться стабильной даже в случае изменения формата исходных таблиц (например, добавления или удаления столбцов, изменения связей, разделения таблиц, их реструктуризации или переименования).

АКТУАЛЬНОСТЬ

Изменения данных в любой из таблиц базы данных, указанных в определяющем запросе, немедленно отображаются на содержимом представления.

ПОВЫШЕНИЕ ЗАЩИЩЕННОСТИ ДАННЫХ

Права доступа к данным могут быть предоставлены исключительно через ограниченный набор представлений, содержащих только то подмножество данных, которое необходимо пользователю.

СНИЖЕНИЕ СТОИМОСТИ

Представления позволяют упростить структуру запросов за счет объединения данных из нескольких таблиц в единственную виртуальную таблицу. В результате многотабличные запросы сводятся к простым запросам к одному представлению.

ВОЗМОЖНОСТЬ НАСТРОЙКИ

Представления являются удобным средством настройки индивидуального образа базы данных.

ОБЕСПЕЧЕНИЕ ЦЕЛОСТНОСТИ ДАННЫХ

Если в операторе CREATE VIEW будет указана фраза WITH CHECK OPTION, то СУБД станет осуществлять контроль за тем, чтобы в исходные таблицы базы данных не была введена ни одна из строк, не удовлетворяющих предложению WHERE в определяющем запросе.

НЕДОСТАТКИ ПРЕДСТАВЛЕНИЙ

ОГРАНИЧЕННЫЕ ВОЗМОЖНОСТИ ОБНОВЛЕНИЯ

В некоторых случаях представления не позволяют вносить изменения в содержащиеся в них данные.

СТРУКТУРНЫЕ ОГРАНИЧЕНИЯ

Структура представления устанавливается в момент его создания. Если определяющий запрос представлен в форме `SELECT * FROM _`, то символ `*` ссылается на все столбцы, существующие в исходной таблице на момент создания представления. Если впоследствии в исходную таблицу базы данных добавятся новые столбцы, то они не появятся в данном представлении до тех пор, пока это представление не будет удалено и вновь создано.

СНИЖЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ

Использование представлений связано с определенным снижением производительности. В одних случаях влияние этого фактора совершенно незначительно, тогда как в других оно может послужить источником существенных проблем.

Выполнение разрешения представлений связано с использованием дополнительных вычислительных ресурсов.

ТРИГГЕРЫ

подпрограммы, которые всегда выполняются автоматически на стороне сервера, в ответ на изменение данных в таблицах БД. Это методы, с помощью которых разработчик может обеспечить целостность БД.

Триггер активизируется при попытке изменения данных в таблице, для которой **определен**. SQL выполняет эту процедуру при операциях добавления, обновления и удаления (**INSERT, UPDATE, DELETE**) в данной таблице.

Наиболее общее применение триггера – поддержка целостности в базах данных. Триггеры незначительно влияют на производительность сервера и часто используются для усиления предложений, выполняющих многокаскадные операции в таблицах и строках.

Триггер может выполняться в двух фазах изменения данных: до (**before**) какого-то события, или после (**after**) него.

```
CREATE TRIGGER <trigname> FOR {<table_name> | <view_name>}  
  [ACTIVE | INACTIVE]  
  {BEFORE | AFTER} {DELETE | INSERT | UPDATE}  
  [POSITION <number>]
```

```
AS  
  [DECLARE [VARIABLE] <variable datatype>;]  
BEGIN  
  <compound_statement>[<compound_statement>]  
END  
<compound_statement> = {<block> | statement;}
```

ЗАГОЛОВОК ТРИГГЕРА

- имя триггера, уникальное по БД;
- имя таблицы, с которой ассоциируется триггер;
- момент, когда триггер должен вызываться.

ТЕЛО ТРИГГЕРА

состоит из опционального списка локальных переменных и операторов.

ACTIVE | INACTIVE – необязательный параметр определяет, будет триггер запускаться в ответ на событие, или не будет

{BEFORE | AFTER} {DELETE | INSERT | UPDATE} – два обязательных параметра, комбинация которых может запрограммировать триггер на шесть различных вариантов реагирования на события

POSITION <number> – необязательный параметр, который определяет очередность запуска, если для той же таблицы и для того же события имеется другой триггер

AS – команда, начинающая тело триггера.

Все, что следует за частью **AS** оператора **CREATE TRIGGER** составляет тело триггера. Тело триггера состоит из опционального списка локальных переменных, за которым идет блок операторов. **Блок состоит из набора операторов на языке хранимых процедур и триггеров, заключенных в операторные скобки.**

Хранимые процедуры представляют собой набор команд, состоящий из одного или нескольких операторов SQL или функций и сохраняемый в базе данных в откомпилированном виде.

Язык хранимых процедур и триггеров Interbase (ЯХПТ) является полным языком для написания хранимых процедур и триггеров. Язык включает в себя:

- операторы манипулирования данными SQL (INSERT, UPDATE, DELETE) а также оператор SELECT;
- операторы и выражения SQL, включая функции, определяемые пользователем (UDF);
- расширения SQL, включая, оператор присваивания, операторы управления последовательностью выполнения, локальные переменные, операторы отсылки сообщений, обработка исключительных ситуаций, операторы обработки ошибок.

ПЕРЕМЕННЫЕ NEW И OLD

Эти переменные объявлять не нужно, они уже присутствуют в каждом триггере. Соответственно, переменные хранят старое и новое значения какого-либо поля.

NEW.<имя_поля>

Значения **NEW** могут быть использованы в событиях **INSERT** и **UPDATE**, при удалении записи NEW имеет значение NULL.

OLD.<имя_поля>

Значения **OLD** доступны в событиях **UPDATE** и **DELETE**, а при вставке новой записи OLD имеет значение NULL.

Пример: создадим триггер, который срабатывает перед вставкой новой записи и проверяет входящее целое число. Если оно отрицательно, триггер изменяет его на ноль.

```
SET TERM ^;  
CREATE TRIGGER NotOtric FOR Table_Cel  
ACTIVE BEFORE INSERT  
AS  
BEGIN  
    IF (NEW.Dlinnoe < 0) THEN NEW.Dlinnoe = 0;  
END^  
SET TERM ;^
```

Если затем выполнить следующие команды:

```
INSERT INTO Table_cel(Dlinnoe) VALUES(5);  
INSERT INTO Table_cel(Dlinnoe) VALUES(-10);  
SELECT * FROM Table_cel;
```

то в таблице появились две новые строки: в первом случае значение 5 сохранилось без изменения, а во второй записи триггер изменил значение -10 на 0.

ПРИМЕРЫ РЕАЛИЗАЦИИ ТРИГГЕРОВ

Необходимо создать триггер, который не позволяет добавлять более пяти поставщиков из одного города.

```
CREATE EXCEPTION s_c 'Sellers amount > 5';
```

```
SET TERM !! ;
```

```
CREATE TRIGGER TEMP FOR S
```

```
ACTIVE BEFORE INSERT
```

```
AS
```

```
DECLARE VARIABLE s_amount INTEGER;
```

```
BEGIN
```

```
    s_amount = (SELECT COUNT(*) FROM S WHERE city = NEW.city);
```

```
    IF (s_amount = 5) THEN EXCEPTION s_c;
```

```
END!!
```

```
SET TERM ; !!
```

ПРИМЕРЫ РЕАЛИЗАЦИИ ТРИГГЕРОВ

Сделать для таблицы S поле, значение которого автоматически увеличивается на единицу.

```
CREATE GENERATOR Gen_S_ID;  
ALTER SEQUENCE Gen_S_ID RESTART WITH 0;  
  
SET TERM !! ;  
CREATE TRIGGER auto_id_s FOR S  
ACTIVE BEFORE INSERT  
AS  
BEGIN  
    IF (NEW.ID IS NULL) THEN  
        NEW.ID = GEN_ID(Gen_S_ID, 1) ;  
END !!  
SET TERM ; !!  
  
COMMIT;
```