

# Процедурные расширения SQL

Хранимые процедуры и  
триггеры

# Управляющие конструкции

- BEGIN END
- IF
- CASE
- WHILE
- REPEAT UNTIL
- LOOP

# Операторные скобки и преобразование типов

- [*begin\_label*:] **BEGIN**  
[*statement\_list*]  
**END** [*end\_label*]
- **CAST**(*expr* AS *type*)

# Переменные и параметры

- `DECLARE` *var\_name* [, *var\_name*] ... *type* [DEFAULT *value*]
- `DECLARE @s` VARCHAR(20);
- `SELECT` ... `INTO` *var\_list*
- `SELECT` id, data `INTO @x, @y` FROM test.t1 LIMIT 1;
- `SET` *variable* = *expr* [, *variable* = *expr*] ...
- `SET @name` = 43;  
`SET @total_tax` =  
(`SELECT SUM(tax)` FROM taxable\_transactions);

# Типы переменных

- Пользовательские переменные (с префиксом @)
- Локальные переменные (без префикса)
- Системные переменные сервера (с префиксом @@):

# Типы переменных

- Пользовательские переменные (с префиксом @)
- Можно получить доступ к любой пользовательской переменной без объявления ее или инициализируя его. Если вы ссылаетесь на переменную, которая не была инициализировано, оно имеет значение NULL и тип строки.
- Можно инициализировать переменную с помощью инструкции SET или SELECT: `SELECT @start := 1, @finish := 10;`
- Пользовательским переменным может быть присвоено значение из ограниченного набора данных типы: целочисленные, десятичные, плавающие, двоичные или недвоичные строки, или NULL.
- Пользовательские переменные зависят от сеанса. То есть пользовательская переменная, определенная одним клиентом, не может быть замечена или использована другими клиентами.
- Локальные переменные (без префикса)
- Локальные переменные должны быть объявлены с помощью DECLARE до доступа к ней.
- Они могут использоваться как локальные переменные и входные параметры внутри хранимой процедуры:
- `DECLARE start INT unsigned DEFAULT 1;`
- Если предложение DEFAULT отсутствует, начальное значение NULL.
- Область действия локальной переменной - блок BEGIN ... END внутри которой она объявлена.

# Типы переменных



- Системные переменные сервера (с префиксом @@):
- Сервер MySQL поддерживает множество системных переменных, имеющих значение по умолчанию. Они могут иметь тип GLOBAL, SESSION или BOTH
- `SELECT @@sort_buffer_size;`
- **Глобальные переменные** - инициализируются при старте MySQL сервера, получая значения по умолчанию.
- **Сеансовые переменные** - создаются для каждого соединения клиента с сервером и получают значения, установленные для глобальных переменных.

```
SET @@auto_increment_increment=10;
```

# Пользовательские переменные. Пример

```
33 • SELECT @start := 1;  
34 • set @start2=1;  
35 • select @start2;
```

<

Result Grid   Filter Rows:

	@start2
▶	1



# Условный оператор

- `IF search_condition  
THEN statement_list  
[ELSEIF search_condition  
THEN statement_list] ...  
[ELSE statement_list] END IF`
- `DECLARE s VARCHAR(20);`
- `IF n > m  
THEN  
SET s = '>';  
ELSEIF n = m  
THEN SET s = '=';  
ELSE SET s = '<'; END IF;`

# case

- **CASE** *case\_value*  
**WHEN** *when\_value* **THEN** *statement\_list*  
[**WHEN** *when\_value* **THEN** *statement\_list*]  
... [**ELSE** *statement\_list*] **END CASE**
- **CASE WHEN** *search\_condition* **THEN**  
*statement\_list*  
[**WHEN** *search\_condition* **THEN**  
*statement\_list*] ...  
[**ELSE** *statement\_list*] **END CASE**

# CASE expression vs CASE statement

expression	statement
<pre>SELECT CASE WHEN type = 1 THEN 'foo' WHEN type = 2 THEN 'bar' ELSE 'baz' END AS name_for_numeric_type FROM sometable</pre>	<pre>CASE WHEN action = 'update' THEN UPDATE sometable SET column = value WHERE condition; WHEN action = 'create' THEN INSERT INTO sometable (column) VALUES (value); END CASE</pre>
<pre>SELECT CASE type WHEN 1 THEN 'foo' WHEN 2 THEN 'bar' ELSE 'baz' END AS name_for_numeric_type FROM sometable</pre>	<pre>CASE action WHEN 'update' THEN UPDATE sometable SET column = value WHERE condition; WHEN 'create' THEN INSERT INTO sometable (column) VALUES (value); END CASE</pre>

# Case пример

```
mysql> SELECT CASE 1 WHEN 1 THEN 'one'
->     WHEN 2 THEN 'two' ELSE 'more' END;
-> 'one'

mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
-> 'true'

mysql> SELECT CASE BINARY 'B'
->     WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
-> NULL
```

# WHILE

- [*begin\_label*:]  
**WHILE** *search\_condition*  
**DO** *statement\_list*  
**END WHILE** [*end\_label*]
- **DECLARE** *v1* INT DEFAULT 5;
- **WHILE** *v1* > 0 **DO**
- ...
- **SET** *v1* = *v1* - 1;
- **END WHILE**;

# REPEAT UNTIL

- [*begin\_label*:]  
REPEAT *statement\_list*  
UNTIL *search\_condition*  
END REPEAT [*end\_label*]
- SET @x = 0;  
REPEAT SET @x = @x + 1;  
UNTIL @x > 8  
END REPEAT;

# LOOP

- *[begin\_label:]*  
LOOP *statement\_list*  
END LOOP *[end\_label];*
- CREATE PROCEDURE doiterate(*p1* INT) BEGIN  
label1: LOOP  
SET *p1* = *p1* + 1;  
IF *p1* < 10 THEN  
ITERATE label1;  
END IF;  
LEAVE label1;  
END LOOP label1;  
SET @x = *p1*;  
END;

# Хранимые процедуры

- Хранимые процедуры представляют, по существу, предварительно откомпилированные программы, которые *хранятся на узле сервера* (и известны серверу). Клиент обращается к хранимой процедуре с помощью механизма вызова удаленных процедур (Remote Procedure Call — RPC).



# Хранимые процедуры и функции.

## Создание

- `CREATE` [DEFINER = *user*]  
`PROCEDURE` *sp\_name* ([*proc\_parameter*[,...]])  
[*characteristic ...*] *routine\_body*
- `CREATE` [DEFINER = *user*]  
`FUNCTION` *sp\_name* ([*func\_parameter*[,...]])  
`RETURNS` *type* [*characteristic ...*] *routine\_body*
- *proc\_parameter*: [ **IN** | **OUT** | **INOUT** ] *param\_name type*  
*func\_parameter*: *param\_name type*
- *characteristic*:
- `COMMENT` '*string*' |
- `LANGUAGE SQL` |
- [NOT] DETERMINISTIC
- | { `CONTAINS SQL` | `NO SQL` | `READS SQL DATA` |  
`MODIFIES SQL DATA` }
- | `SQL SECURITY` { `DEFINER` | `INVOKER` }

# Хранимые процедуры и функции. Вызов

- `CALL sp_name`  
(`[parameter[,...]]`)
- `CALL sp_name[()]`
  
- `SELECT fun_name`  
(`[parameter[,...]]`) [`INTO var`]
- `SET var = fun_name (parameter[,...])`
- ...

# Удаление. Изменение

- DROP {PROCEDURE | FUNCTION} [IF EXISTS]  
*sp\_name*
- ALTER PROCEDURE *proc\_name* [*characteristic ...*]
- ALTER FUNCTION *func\_name* [*characteristic ...*]

• *characteristic*:

COMMENT '*string*' |

LANGUAGE SQL |

{ CONTAINS SQL |

NO SQL |

READS SQL DATA |

MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

# Хранимые процедуры. Пример

- `delimiter //`  
`CREATE PROCEDURE simpleproc`  
`(OUT param1 INT)`  
`BEGIN`
- `SELECT count(*) INTO param1 from t ;`  
`END; //`
- `delimiter ;`
- `call simpleproc(@param);`  
`select @param;`

# Хранимые функции. Пример

- `CREATE FUNCTION hello (s CHAR(20))  
RETURNS CHAR(50) DETERMINISTIC  
RETURN CONCAT('Hello, ',s,'!');`
- `SELECT hello('world');`

# Вспомогательные функции

- `IFNULL(expr1, expr2)`
- Если *expr1* не NULL, `IFNULL()` возвращает *expr1* в противном случае возвращает *expr2*.
- `SELECT IFNULL(max(ID)+1, 0) from tab`
- В MSSQL эту роль играет функция
- `ISNULL(expr1, expr2)`

# Значение ключа

- **last\_insert\_id()**
- `LAST_INSERT_ID ()`, `LAST_INSERT_ID (expr)` Без аргумента `LAST_INSERT_ID ()` возвращает значение `BIGINT UNSIGNED` (64-разрядное), представляющее **первое** автоматически сгенерированное значение, успешно вставленное для столбца `AUTO_INCREMENT` в результате последнего выполненного оператора `INSERT`. **Значение `LAST_INSERT_ID ()` остается неизменным, если строки не были успешно вставлены.** С аргументом `LAST_INSERT_ID ()` возвращает целое число без знака.
- `LAST_INSERT_ID ()` **не сбрасывается между операторами**, потому что значение этой функции хранится на сервере. Другое отличие от `mysql_insert_id ()` состоит в том, что `LAST_INSERT_ID ()` не обновляется, если для столбца `AUTO_INCREMENT` задано определенное неспецифическое значение.

# Значение ключа

- **last\_insert\_id() -SQL**
- **mysql\_insert\_id() – API для C**
- Возвращаемое значение mysql\_insert\_id () всегда **равно нулю**, если явно не обновлено при одном из следующих условий:
- Операторы INSERT, которые сохраняют значение в столбце AUTO\_INCREMENT. Это верно, независимо от того, генерируется ли значение автоматически путем сохранения специальных значений NULL или 0 в столбце, или это явное неспецифическое значение.
- В случае многострочного оператора INSERT mysql\_insert\_id () возвращает **первое** автоматически сгенерированное значение AUTO\_INCREMENT, которое было успешно вставлено.
- Если ни одна строка не была успешно вставлена, mysql\_insert\_id () возвращает 0.
- Если выполняется инструкция INSERT ... SELECT, и автоматически сгенерированное значение не было успешно вставлено, mysql\_insert\_id () возвращает идентификатор **последней вставленной строки**.
- Если инструкция INSERT ... SELECT использует LAST\_INSERT\_ID (**expr**), mysql\_insert\_id () возвращает **expr**. Операторы INSERT, которые генерируют значение AUTO\_INCREMENT путем вставки LAST\_INSERT\_ID (expr) в любой столбец или путем обновления любого столбца до LAST\_INSERT\_ID (expr). Если предыдущий оператор возвратил ошибку, значение mysql\_insert\_id () не определено.



# Преимущества и недостатки ХП

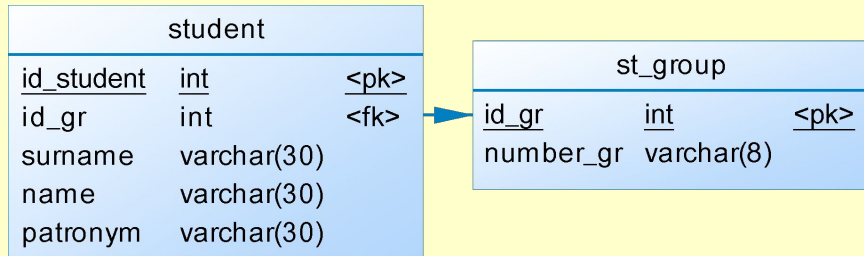
## Преимущества

- Компенсация потерь в производительности , связанных с обработкой данных на уровне записей в системе «клиент/сервер», за счет обработки непосредственно на узле сервера
- Возможность скрыть от пользователя множество специфических особенностей СУБД и базы данных и соответственно более высокая степень независимости от данных по сравнению с тем случаем, когда хранимые процедуры не используются.
- Одна хранимая процедура может совместно использоваться многими клиентами.
- Оптимизация может быть осуществлена при создании ХП, а не во время выполнения.
- Хранимые процедуры позволяют обеспечить более высокую степень безопасности данных

## Недостатки

- поставщики программного обеспечения предоставляют в этой области слишком отличающиеся между собой средства, а расширение языка SQL для поддержки хранимых процедур появилось лишь в 1996 году. Это средство называется SQL/PSM (Persistent Stored Module — постоянный хранимый модуль).
- Работа с кодом и в приложении и в БД

# Часто используемые процедуры



- Процедура вставки с пополнением справочника
- Процедура удаления с очисткой справочника
- Процедура каскадного удаления

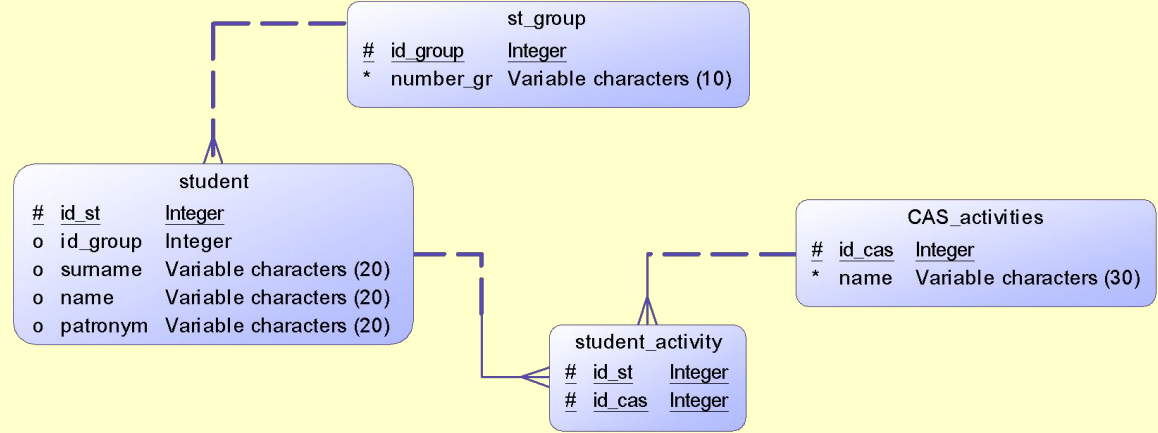
# Процедура вставки с пополнением справочника

```
delimiter //
CREATE PROCEDURE ins_stud (gr_num varchar(8),name_ varchar(15),
surname_ varchar(20),patronym_ varchar(25))
BEGIN
declare id_gr_new int;
declare id_st_new int;
if exists(select * from st_group where num_gr=gr_num)
then select id_gr into id_gr_new from st_group where num_gr=gr_num;
else begin
set id_gr_new=( select ifnull(max(id_gr)+1,0) from st_group );
INSERT INTO st_group(id_gr,num_gr) VALUES (id_gr_new,gr_num);
end;
end if;
set id_st_new=( select ifnull(max(id_st)+1,0) from student);
insert into student (id_st, surname, name,patronym,id_gr)
VALUES (id_st_new,surname_,name_,patronym_,id_gr_new);
END;//
delimiter ;
```

# Процедура удаления с очисткой справочника

```
delimiter //
create procedure del_student_clear_gr(id_st_del int)
BEGIN
declare id_gr_del int;
select id_gr into id_gr_del from student where id_st=id_st_del;
delete from student where id_st=id_st_del;
if not exists(select * from student where id_gr=id_gr_del)
then delete from st_group where id_gr=id_gr_del;
end if;
END;//
delimiter ;
```

# Процедура каскадного удаления



```
delimiter //
create procedure del_group_cascade ( id_gr_del int)
BEGIN
delete from student_activity where id_st in
    (select id_st from student where id_gr= id_gr_del);
delete from student where id_gr= id_gr_del;
delete from st_group where id_gr=id_gr_del;
END;//
delimiter ;
```

# Работа с временной таблицей

```
delimiter //
create procedure cas_statistics ()
BEGIN
create temporary table if not exists cas_stat
(
id_stat int auto_increment primary key,
id_st int,
count_cas int,
count_cas_avg double default 0,
diff_cnt_avg double default 0
);
insert into cas_stat (id_st,count_cas)
select student.id_st, count(id_cas) as count_cas from student
left join student_activity on student.id_st= student_activity.id_st group by student.id_st;

update cas_stat set count_cas_avg=
(select avg(count_cas) from
(select student.id_st, count(id_cas) as count_cas from student
left join student_activity on student.id_st= student_activity.id_st group by student.id_st)q);

update cas_stat set diff_cnt_avg=count_cas-count_cas_avg;

select * from cas_stat;
select avg(diff_cnt_avg*diff_cnt_avg) from cas_stat;

drop table cas_stat;
END;//
delimiter ;
```

ид студента	количество кружков студента	среднее количество кружков у студентов	отклонение количества

count_cas_avg	diff_cnt_avg
0.571428571	0.428571429
0.571428571	1.428571429
0.571428571	-0.571428571
0.571428571	-0.571428571
0.571428571	-0.571428571
0.571428571	0.428571429
0.571428571	0.571428571

Filter Rows:

ff\_cnt\_avg\*diff\_cnt\_avg)

122448979592

# Предупреждение

- Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column. To disable safe mode, toggle the option in Preferences -> SQL Editor and reconnect.
- `SET SQL_SAFE_UPDATES = 0;`
- `update cas_stat set  
diff_cnt_avg=count_cas-count_cas_avg  
where id_stat>0;`

# Работа с временной таблицей

- `SET SQL_SAFE_UPDATES = 0;`
- `call cas_statistics();`
- `SET SQL_SAFE_UPDATES = 1;`

ид студента	количество кружков студента	среднее количество кружков у студентов	отклонение количества

	id_stat	id_st	count_cas	count_cas_avg	diff_cnt_avg
▶	1	1	1	0.571428571	0.428571429
	2	2	2	0.571428571	1.428571429
	3	3	0	0.571428571	-0.571428571
	4	4	0	0.571428571	-0.571428571
	5	5	0	0.571428571	-0.571428571
	6	6	1	0.571428571	0.428571429
	7	7	0	0.571428571	0.571428571

Result Grid	Filter Rows:
	<code>avg(diff_cnt_avg*diff_cnt_avg)</code>
▶	0.5306122448979592