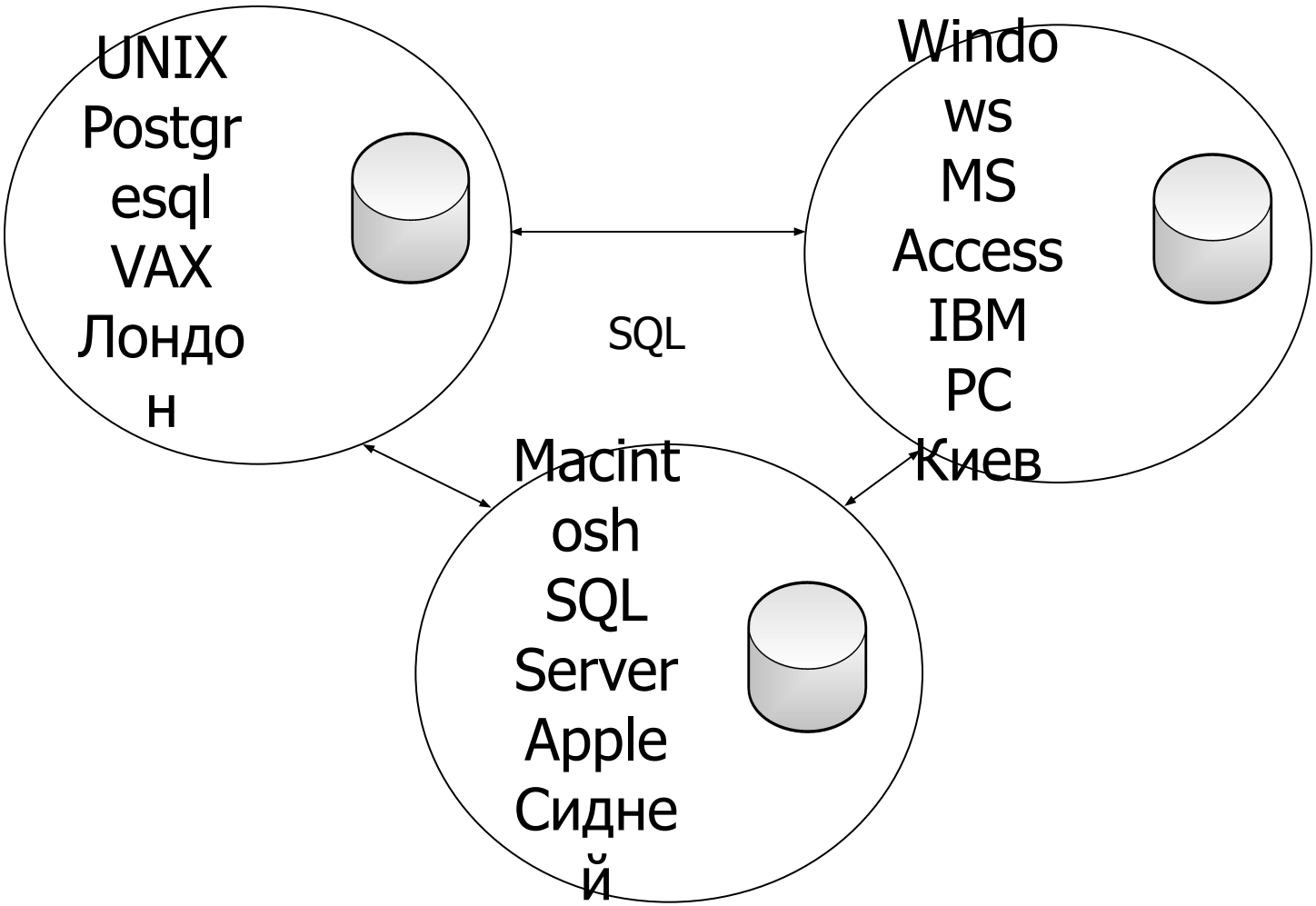
Язык SQL

Лекция 4

SQL (Structured Query Language – структурированный язык запросов) – язык для взаимодействия с БД.


UNIX
Postgr
esql
VAX
Лондо
н
Windo
ws
MS
Access
IBM
PC
Киев
SQL
Macint
osh
SQL
Server
Apple
Сидне
й

Типы команд SQL

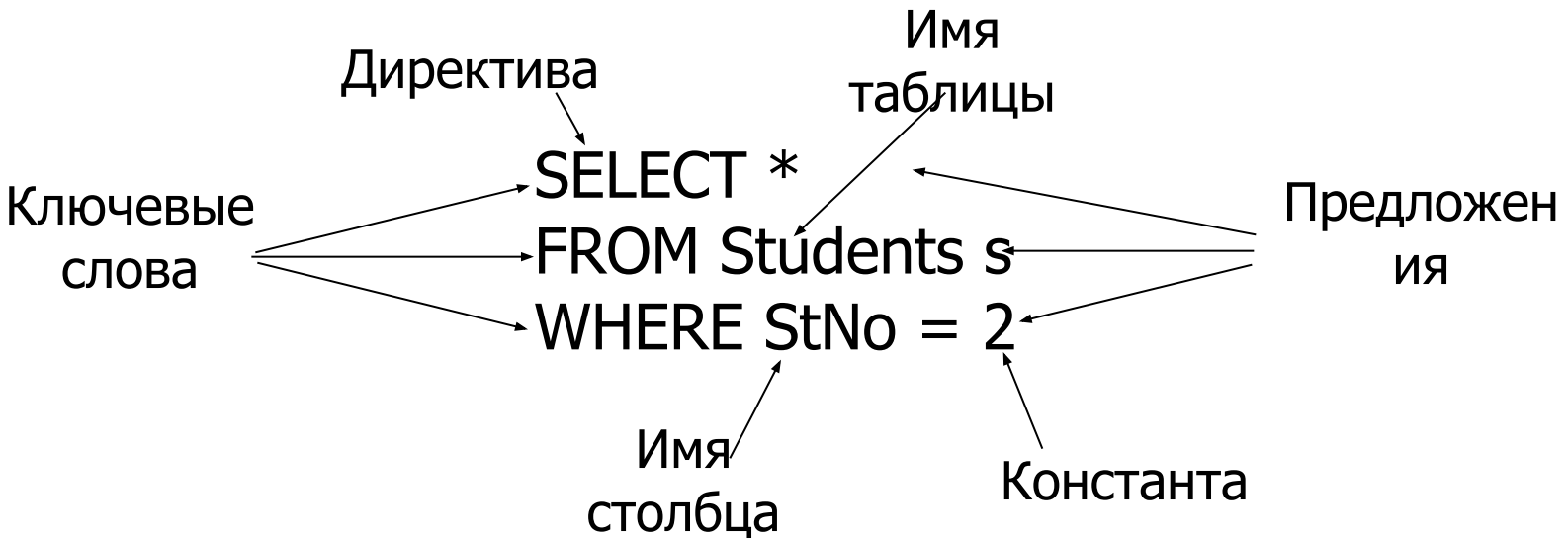
- **DQL** (Data Query Language – язык запросов) – запросы на извлечение данных из таблиц и описания внешнего вида полученных данных (оператор SELECT).
- **DML** (Data Manipulation Language – язык манипулирования данными) – добавление, удаление и изменение данных (операторы INSERT, DELETE, UPDATE).
- **DDL** (Data Definition Language – язык определения данных) создание и уничтожение объектов БД, обеспечение целостности данных (операторы CREATE TABLE/VIEW/INDEX, DROP/ TABLE/VIEW/INDEX, ALTER TABLE/INDEX).

Типы команд SQL

- **DCL** (Data Control Language – язык управления данными) – управление доступом пользователей к БД, а также назначение пользователям уровней привилегий доступа (операторы GRANT, REVOKE)
- **TPL** (Transaction Processing Language – язык обработки транзакций) – операторы COMMIT, ROLLBACK, SAVEPOINT
- команды администрирования данных – аудит и анализ операций внутри БД, а также анализ производительности системы данных в целом (операторы START AUDIT, STOP AUDIT)

Основные понятия SQL

1. Структура **оператора** SQL



- **Директивы** описывают действие, выполняемое оператором: SELECT (выбрать), CREATE (создать), INSERT (добавить), DELETE (удалить), UPDATE

Основные понятия SQL

Предложение описывает данные, с которыми работает оператор, или содержит уточняющую информацию о действии, выполняемом оператором: FROM (откуда), WHERE (где), GROUP BY (группировать по), HAVING (имеющий), ORDER BY (упорядочить по), INTO (куда).

2. **Имена (идентификаторы)**
   - длина – до 128 символов
   - используемые символы – только прописные или строчные буквы латинского алфавита, цифры или символ подчеркивания (_). *Первым символом* должна быть буква.
   - составное имя – идентификатор базы данных, ее владельца и (или) объекта базы данных. Например, *полное имя таблицы* состоит из имени владельца таблицы и имени таблицы, разделенных точкой (.): Admin.Students.

Основные понятия SQL

3. **Комментарии**
   - /* и */ – многострочный комментарий
   - -- – однострочный комментарий
4. **Типы данных**
   - INTEGER или INT – целое число (обычно до 10 значащих цифр и знак);
   - SMALLINT – "короткое целое" (обычно до 5 значащих цифр и знак);
   - NUMERIC(p, q) – десятичное число, имеющее p цифр (0<p<16) и знак; с помощью q задается число цифр справа от десятичной точки (q< p, если q = 0, оно может быть опущено);
   - REAL – число с плавающей запятой;

Основные понятия SQL

- CHAR(n) (CHARACTER(n)) – символьная строка фиксированной длины из n символов (0 < n< 256);
- VARCHAR (n) (CHARACTER VARYING (n))– символьная строка переменной длины, не превышающей n символов (n> 0 и разное в разных СУБД, но не более 8 Кб);
- DATE – дата в формате, определяемом специальной командой (по умолчанию mm/dd/yy, например, 10/03/12).
- TIME – время в формате, определяемом специальной командой, по умолчанию hh.mm.ss.
- BOOLEAN – принимает истинностные значения (TRUE или FALSE).

Основные понятия SQL

5. Константы (литералы)

- Числовые константы: 21, -345, +234,6547
  - Константы с плавающей запятой: 1.5E3, -3.14159E1, 2.5 E7
- Строковые константы: ‘Это символьная строка’.

  Если в строковую константу нужно включить одинарную кавычку, то вместо нее надо писать две одинарные кавычки:

  ‘ Здесь внутри будут ‘ ‘ одинарные’ ’ кавычки’.
- Константы даты и времени. Пример для даты: ‘2012-10-03’, ‘1993-12-10’. Пример для времени: ’17:22:10’, ’01:01:01’.
- Логические константы: TRUE, FALSE, UNKNOWN
- Отсутствующие данные (значение NULL)

Основные понятия SQL

6. Встроенные функции

- CAST (значение AS тип данных) – значение, преобразованное к типу данных (например, дата преобразованная в строку)
- CHAR_LENGTH (строка) – длина строки символов
- CURRENT_DATE – текущая дата
- CURRENT_TIME (точность) – текущее время с указанием точности дробной части секунд
- EXTRACT (часть FROM значение) – указанная часть (YEAR, MONTH, DAY, HOUR, MINUTE, SECOND) из значения временного типа: EXTRACT(YEAR FROM DATE ‘2012-10-03’) = 2012
- LOWER(строка), UPPER(строка) – строка, преобразованная к нижнему (верхнему регистру)

Запросы на чтение данных. Оператор SELECT

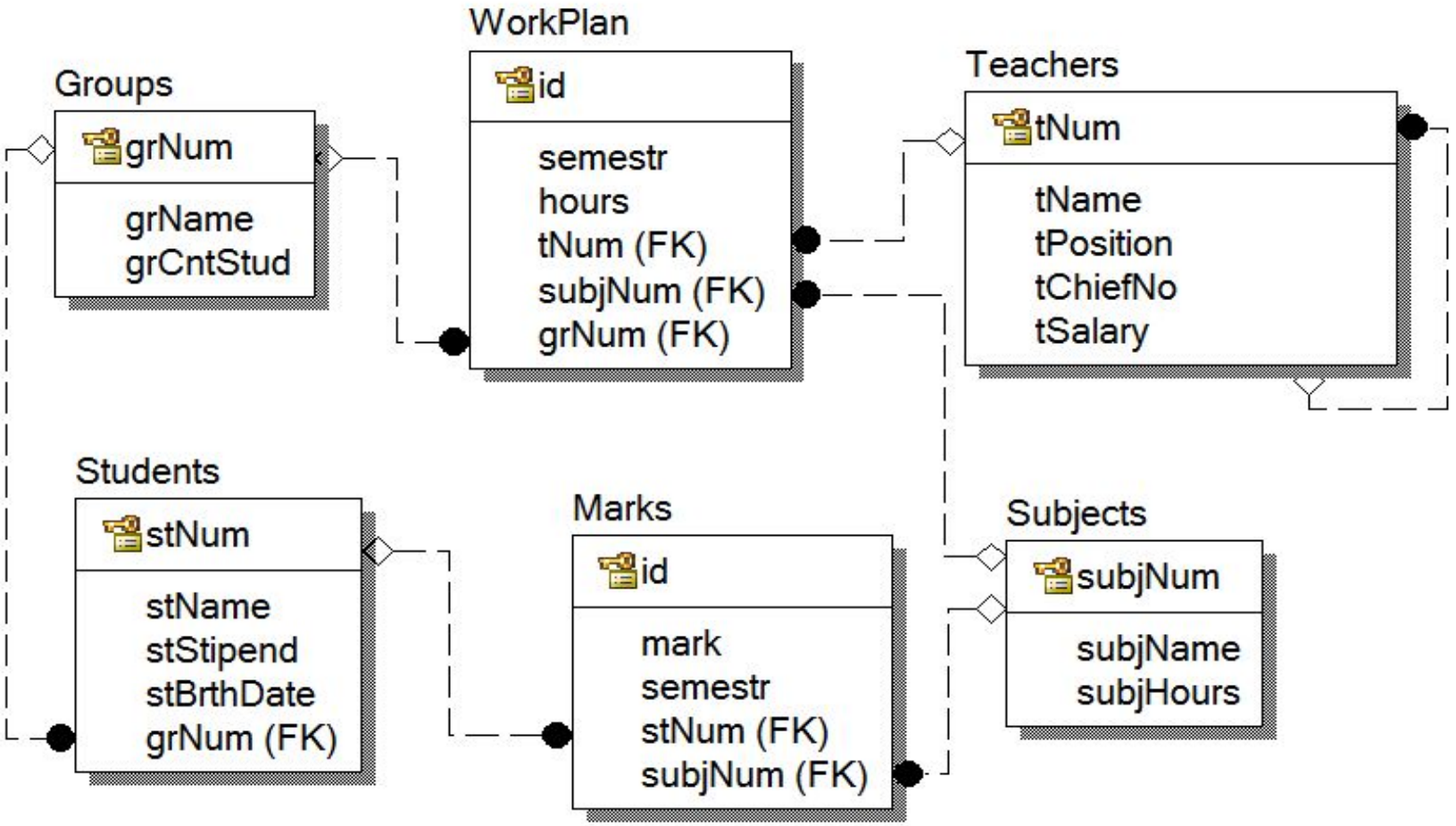
Синтаксис оператора SELECT

SELECT [**ALL** | **DISTINCT**] список_возвращаемых_столбцов|*
FROM список_имен_таблиц
[**WHERE** условие_поиска]
[**GROUP BY** список_имен_столбцов]
[**HAVING** условие_поиска]
[**ORDER BY** имя_столбца [**ASC** | **DESC**],…]

Примечание: в квадратных скобках указаны предложения, которые могут отсутствовать в операторе SELECT.

Запросы на чтение данных. Оператор SELECT

Схема БД (для примеров)



Запросы на чтение данных. Оператор SELECT

1. Цель запроса. Предложение SELECT

Предложение SELECT содержит список возвращаемых столбцов, разделенных символом «запятая» (,).

1.1 Способы указания выводимых столбцов

- **вывод значений определенных столбцов** одной из таблицы, указанной в предложении FROM

  Пример. Получить список студентов и размер их стипендий.

```
SELECT stName, stStipend
FROM Students;
```

Запросы на чтение данных. Оператор SELECT

- для **вывода всех столбцов** таблицы, указанной в предложении FROM, можно перечислить все их названия или воспользоваться символом «звездочка» (*)

  Пример. Вывести все столбцы таблицы Groups.

```
SELECT grNum, grName, grCntStud
FROM Groups;
```

или

```
SELECT *
FROM Groups;
```

Запросы на чтение данных. Оператор SELECT

- **уточнение имен** столбцов путем указания полного имени столбца: **имя_таблицы.имя_столбца**.

Пример:

```
SELECT Groups.grNum, Groups.grName, Groups.grCntStud
FROM Groups ;
```

Запросы на чтение данных. Оператор SELECT

1.2 Исключение повторяющихся строк

Для исключения повторяющихся строк из результирующей таблицы используется ключевое слово **DISTINCT**, которое указывается перед списком возвращаемых столбцов.

Пример. Вывести значения столбца tPosition таблицы Teachers.

```
SELECT tPosition
FROM Teachers;
```

Пример. Вывести уникальные значения столбца tPosition таблицы Teachers.

```
SELECT DISTINCT tPosition
FROM Teachers;
```

Запросы на чтение данных. Оператор SELECT

1.3 Использование вычисляемых выражений

Пример. Вывести фамилии студентов, размер их стипендий в грн. и в $.

```
SELECT stName, stStipend, stStipend / 8,14
FROM Students ;
```

Запросы на чтение данных. Оператор SELECT

1.4 Переопределение имен результирующих столбцов

Для переопределения имени результирующего столбца (создания его синонима) используется ключевое слово **AS**.

Запросы на чтение данных. Оператор SELECT

1.5 Включение текста в результат запроса

В предложении SELECT кроме имен столбцов и выражений с ними можно указывать константы (и константные выражения).

Пример. Вывести фамилии студентов и размер их стипендий, оформив результат предложениями на русском языке.

```
SELECT 'Студент', stName, 'получает стипендию', stStipend
FROM   Students ;
```

Запросы на чтение данных. Оператор SELECT

2. Используемые таблицы. Предложение FROM

Предложение FROM содержит список имен таблиц, разделенных символом «запятая» (,).

Например, **FROM Students, Groups**.

Можно указывать **синонимы (псевдонимы) имен таблиц**.

Например, **FROM** Students **st**, Groups **gr**

Запросы на чтение данных. Оператор SELECT

3. Отбор строк. Предложение WHERE

Предложение WHERE состоит из ключевого слова **WHERE**, за которым следует **условие поиска**, определяющее, какие именно строки требуется выбрать.

Если условие поиска имеет значение TRUE, строка будет включена в результат запроса.

Если условие поиска имеет значение FALSE или NULL, то строка исключается из результата запроса.

Запросы на чтение данных. Оператор SELECT

3.1 Условия отбора строк

- **Сравнение**

  Выражение1 =|<>|<|>|<=|>= Выражение2

Запросы на чтение данных. Оператор SELECT

- **Проверка на принадлежность диапазону значений (BETWEEN)**

проверяемое_выражение [**NOT**] **BETWEEN** минимум **AND** максимум

Пример. Получить список студентов, получающих стипендию в диапазоне от 650 до 1100 грн.

```
SELECT stName, stStipend
FROM   Students
WHERE stStipend BETWEEN 650 AND 1100;
```

Запросы на чтение данных. Оператор SELECT

- **Проверка на принадлежность множеству (IN)**

проверяемое_выражение [**NOT**] **IN** (набор_констант)

Пример. Получить список студентов, получающих стипендию 650 или 730, или 900 грн.

```
SELECT stName, stStipend
FROM   Students
WHERE stStipend IN (650, 730, 900);
```

Запросы на чтение данных. Оператор SELECT

- **Проверка на соответствие шаблону (LIKE)**

имя_столбца [**NOT**] **LIKE** шаблон [ESCAPE символ_пропуска), где

шаблон – это строка, в которую может входить один или более подстановочных знаков.

подстановочные знаки:

% – совпадает с любой последовательностью из нуля или более символов

Пример. Получить сведения о студентах, чья фамилия начинается с «Иван».

```
SELECT *
FROM   Students
WHERE stName LIKE 'Иван%';
```

Запросы на чтение данных. Оператор SELECT

_ (символ подчеркивания) – совпадает с любым отдельным символом.

Пример. Получить сведения о студентах, чье имя «Наталья» или «Наталия».

```
SELECT *
FROM   Students
WHERE stName LIKE '%Натал_я';
```

Запросы на чтение данных. Оператор SELECT

символ пропуска используется для проверки наличия в строках символов, использующихся в качестве подстановочных знаков (%, _).

Пример. Получить сведения из таблицы "Data", где в поле результат содержится фрагмент текста "менее 50%" .

```
SELECT *
FROM Data
WHERE Result LIKE '%менее 50$% %' ESCAPE $;
```

Запросы на чтение данных. Оператор SELECT

- **Проверка на равенство значению NULL (IS NULL)**

имя_столбца **IS** [**NOT**] **NULL**

Пример. Получить сведения о студентах, получающих стипендию.

```
SELECT stName, stNum, stStipend
FROM   Students
WHERE stStipend IS NOT NULL;
```

Запросы на чтение данных. Оператор SELECT

- **Составные условия поиска (AND, OR и NOT)**

WHERE [NOT] условие_поиска **[AND|OR]** [NOT] условие_поиска …

Пример. Получить сведения о студентах, которые учатся в группе с кодом «1» и получают стипендию.

```
SELECT *
FROM   Students
WHERE (grNum = 1) AND (stStipend IS NOT NULL);
```

Запросы с многими таблицами

1. **Естественное соединение таблиц**

Объединенную таблицу образуют пары тех строк из различных таблиц, у которых в связанных столбцах содержатся одинаковые значения.

Пример 1. Получить список студентов и названия их групп.

```
SELECT stName, grName
FROM   Students, Groups
WHERE (Students.grNum = Groups.grNum);
```

Связанные столбцы представляют собой пару «внешний ключ – первичный ключ».

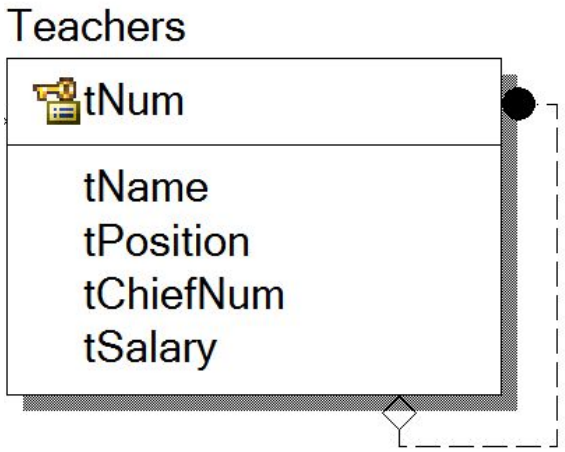
Запросы с многими таблицами

2. Запросы с самообъединением таблиц

В SQL для объединения таблицы с самой собой (самообъединение таблиц) применяется подход, состоящий в использовании "виртуальной копии" таблицы, на которую можно сослаться, используя псевдоним таблицы.

Пример 1. Вывести список всех преподавателей и их руководителей.

```
SELECT Teachers.tName, Chiefs.tName
FROM Teachers, Teachers Chiefs
WHERE Teachers.tChiefNum = Chiefs.tNum;
```



Запросы с многими таблицами

Соединение таблиц может осуществляться и по условиям, отличным от равенства:

SELECT атр1, атр2,…,атрN **FROM** t1, t2

WHERE t1.a Θ t2.b

здесь Θ любой оператор сравнения: >, >=, <, <=, <>

Пример 2. Получить названия пар предметов, имеющих одинаковое количество учебных часов.

```
SELECT a.subjName, b.subjName
 FROM Subjects a, Subjects b
 WHERE a.subjHours = b.subjHours AND
       a.subjName <> b.subjName;
```

Агрегатные функции

Агрегатная функция принимает в качестве аргумента какой-либо столбец данных целиком, а возвращает одно значение, которое определенным образом подытоживает этот столбец.

- SUM(выражение | [DISTINCT] имя_столбца) – сумма [различных] числовых значений
- AVG(выражение | [DISTINCT] имя_столбца) – средняя величина [различных] числовых значений
- MIN(выражение | имя_столбца) – наименьшее среди всех значений

Агрегатные функции

- MAX(выражение | имя_столбца) – наибольшее среди всех значений
- COUNT([DISTINCT] имя_столбца) – подсчитывает количество значений, содержащихся в столбце
- COUNT(*) – подсчитывает количество строк в таблице результатов запроса

Примечание: агрегатные функции нельзя применять в предложении WHERE

Агрегатные функции

Пример 1. Найти суммарное, среднее, минимальное и максимальное значение стипендии студентов.

```
SELECT SUM(stStipend) AS Sm, AVG(stStipend) AS Av,
    MIN(stStipend) AS Mn, MAX(stStipend) AS Mx
FROM Students
```

Пример 2. Найти количество студентов, получающих стипендию.

```
SELECT COUNT(*) AS Cnt
FROM Students
WHERE stStipend > 0
```

Сортировка результатов запроса. Предложение ORDER BY

ORDER BY имя_столбца [ASC | DESC], **...**

где, ASC – возрастающий, DESC – убывающий порядок сортировки.

Пример. Вывести список фамилий студентов, учащихся в группе КИ-125 в обратном алфавитном порядке.

SELECT stName

FROM Students, Groups

WHERE Students.stNum = Groups.grNum AND Groups.grName = ‘КИ-125’

ORDER BY stName DESC

Запросы с группировкой. Предложение GROUP BY

Использование фразы GROUP BY позволяет сгруппировать строки в группы, имеющие одинаковые значения указанного поля:

grName	ORDER BY grName	GROUP BY grName
КИ-121	КИ-101	КИ-101
ПИ-111	= КИ-121	= КИ-121
КИ-101	КИ-121	ПИ-111
КИ-121	ПИ-111	

К группам, полученным после применения GROUP BY, можно применить любую из стандартных агрегатных функций.

Запросы с группировкой. Предложение GROUP BY

Пример 1. Получить список студентов и их средний балл.

```
SELECT stName, AVG(mark) AS AvgMark
FROM Students, Marks
WHERE Students.stNum = Marks.stNum
GROUP BY stName
```

Примечание. В списке отбираемых полей оператора SELECT, содержащего раздел GROUP BY, можно включать только агрегатные функции и поля, которые входят в условие группировки.

Запросы с группировкой. Предложение GROUP BY

Несколько столбцов группировки

Пример. Получить список студентов и их средний балл за каждый семестр.

```
SELECT stName, semestr, AVG(mark) AS AvgMark
FROM Students, Marks
WHERE Students.stNum = Marks.stNum
GROUP BY stName, semestr
```

Значения NULL в столбцах группировки

Строки, имеющие значение NULL в одинаковых столбцах группировки и идентичные значения во всех остальных столбцах группировки, помещаются в одну группу.

Запросы с группировкой. Предложение GROUP BY

Условия поиска групп. Предложение HAVING

Предложение HAVING, используемое совместно с GROUP BY, позволяет исключить из результата группы, неудовлетворяющие условию (так же, как WHERE позволяет исключить строки).

Пример 1. Получить список групп специальности КИ, в которых число студентов меньше 15.

```
SELECT grName, COUNT(*) AS CntStudents
FROM Students, Groups
WHERE Students.gtNum = Groups.grNum AND
Groups.grName LIKE 'КИ%'
GROUP BY grName
HAVING COUNT(*) <  15
```

Вложенные запросы

Вложенным запросом (подзапросом) называется запрос, содержащийся в предложении WHERE или HAVING другого оператора SQL.

Пример 1. Получить список предметов, по которым была получена оценка <4.

```
SELECT subjName
FROM Subjects
WHERE subjNum IN ( SELECT subjNum
                   FROM Marks
                   WHERE mark < 4)
```

Вложенные запросы

Коррелируемым подзапросом называется подзапрос, который содержит ссылку на столбцы таблицы внешнего запроса.

Пример 2. Вывести список студентов, средний балл которых выше 4,5.

```
SELECT stName
FROM Students
WHERE (SELECT AVG(mark)
        FROM Marks
        WHERE Marks.stNum = Students.stNum) > 4.5
```

Вложенные запросы

Особенности вложенных запросов:

- вложенный запрос всегда заключается в круглые скобки;
- таблица результатов вложенного запроса всегда состоит из одного столбца;
- во вложенный запрос не может входить предложение ORDER BY.

Квантор существования EXISTS

В языке SQL предикат с квантором существования представляется выражением вида:

[NOT] EXISTS (SELECT…FROM…WHERE…),

которое следует за фразой WHERE. Такое выражение считается истинным, если подзапрос возвращает непустое множество (существует хотя бы 1 строка, которую возвращает подзапрос). На практике подзапрос всегда будет коррелированным.

Пример. Получить список студентов, сдавших хотя бы один экзамен.

```
SELECT stName FROM Students
WHERE EXISTS ( SELECT * FROM Marks
                    WHERE Marks.stNum = Students.stNum);
```

Многократное сравнение ANY и ALL

Синтаксис многократного сравнения:

проверяемое_выражение = | <> | < | <= | > | >=
ANY | ALL вложенный_запрос

Квантор общности ALL в языке SQL

Пример. Получить список студентов, получающих стипендию большую, чем любой из студентов группы КИ-121.

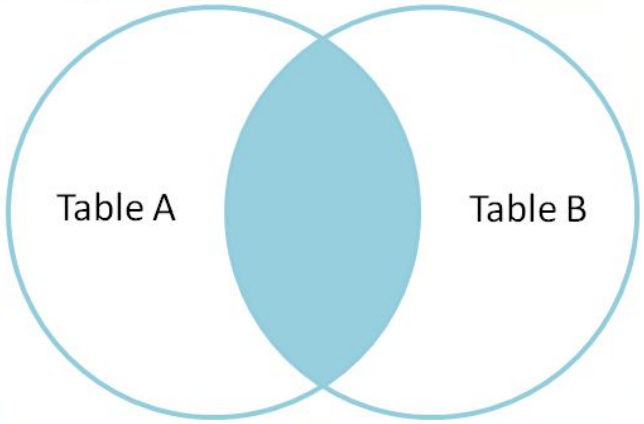
```
SELECT *
FROM Students
WHERE stStipend > ALL (SELECT stStipend
                       FROM Students, Groups
                       WHERE Students.grNum = Groups.grNum
                      AND Groups.grName = 'КИ-121');
```

Квантор ANY (SOME) в языке SQL

Пример. Найти студентов университета, день рождения которых совпадает с днем рождения хотя бы одного из студентов группы КИ-121.

```
SELECT *
FROM Students, Groups
WHERE stBrthDate = ANY (SELECT stBrthDate
                        FROM Students, Groups
                        WHERE Students.grNum = Groups.grNum
                      AND Groups.grName = 'КИ-121')        AND
Students.grNum = Groups.grNum
      AND Groups.grName <> 'КИ-121';
```

Внутреннее соединение таблиц (INNER JOIN)



INNER JOIN

Пример. Вывести список студентов, и названия групп, в которых они учатся.

```
SELECT stName, grName
FROM Students INNER JOIN Groups
    ON Students.grNum = Groups.grNum;
```

Внутреннее соединение таблиц (INNER JOIN)

Если таблицы нужно соединить по равенству столбцов с одинаковыми именами, то вместо предложения ON используется предложение USING, в котором перечисляются названия соединяемых столбцов.

Пример.

```
SELECT stName, grName
FROM Students INNER JOIN Groups
        USING (grNum);
```

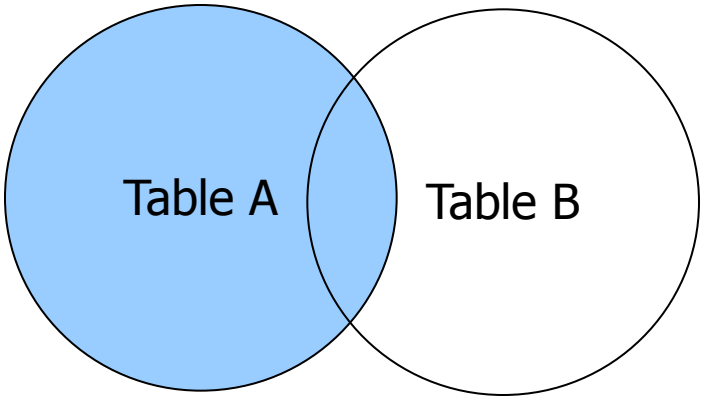
Внешнее соединение таблиц (OUTER JOIN)

В SQL-92 поддерживается понятие **внешнего соединения** двух типов:

- левостороннее (LEFT OUTER JOIN, *=);
- правостороннее (RIGHT OUTER JOIN, =*).

Внешнее соединение таблиц (OUTER JOIN)

LEFT OUTER JOIN



Внешнее соединение таблиц (OUTER JOIN)

Students

stNum	stName	grNum
1	Иванов	1
2	Васильев	1
3	Петров	

Groups

grNum	grName
1	КИ-121
2	ПИ-111

SELECT Students.stName, Groups.grName

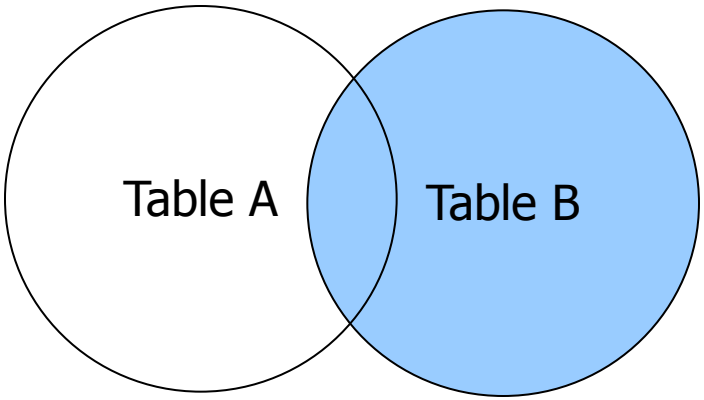
FROM Students **LEFT OUTER JOIN** Groups **ON** Students.grNum = Groups.grNum

Результат:

stName	grName
Иванов	КИ-121
Васильев	КИ-121
Петров	

Внешнее соединение таблиц (OUTER JOIN)

RIGHT OUTER JOIN



Внешнее соединение таблиц (OUTER JOIN)

Students

stNum	stName	grNum
1	Иванов	1
2	Васильев	1
3	Петров	

Groups

grNum	grName
1	КИ-121
2	ПИ-111

SELECT Students.stName, Groups.grName

FROM Students **RIGHT OUTER JOIN** Groups **ON** Students.grNum = Groups.grNum

Результат:

stName	grName
Иванов	КИ-121
Васильев	КИ-121
	ПИ-111

Предложение SELECT INTO

Для сохранения результатов SQL-запроса можно использовать новую таблицу. В этом случае синтаксис операции выборки имеет вид:

SELECT … **INTO** <имя новой таблицы>

FROM …

[**WHERE**…]

Пример: **SELECT** * **INTO** StudentsBackup

FROM Students

Операции модификации данных (DML)

Для модификации данных используются три оператора: INSERT, DELETE и UPDATE.

1. Добавление строки в таблицу БД осуществляется с помощью оператора INSERT (вставка):

   INSERT INTO имя_таблицы (имя_столбца,…)

   VALUES (константа | NULL,…)

Пример. Добавить запись о новой группе ‘КИ-111’ в БД.

INSERT INTO Groups (grNum, grName, grHead)

VALUES (6, 'КИ-111', 11234);

Операции модификации данных (DML)

При **добавлении значений во все столбцы** таблицы список столбцов можно не писать:

INSERT INTO Groups **VALUES** (6, 'КИ-111', 11234);

Многострочный оператор INSERT добавляет в таблицу несколько строк:

INSERT INTO имя_таблицы (имя_столбца,…) запрос

Пример. Скопировать данные из таблицы Groups в таблицу GroupsCopy

INSERT INTO GroupsCopy (grNum, grName, grHead)

SELECT grNum, grName, grHead **FROM** Groups;

Операции модификации данных (DML)

2. Удаление строк из таблицы БД осуществляется с помощью оператора DELETE (удалить):

   DELETE FROM имя_таблицы

   [**WHERE** условие_поиска],

где условие поиска может быть вложенным запросом.

Пример. Удалить сведения о студенте с номером зачетки 12345.

```
DELETE FROM Students
WHERE stNum = 12345;
```

Примечание. Отсутствие предложения WHERE приводит к удалению ВСЕХ строк из указанной таблицы.

Операции модификации данных (DML)

3. Обновление значения одного или нескольких столбцов в выбранных строках одной таблицы БД осуществляется с помощью оператора UPDATE (обновить):

   UPDATE имя_таблицы **SET** имя_столбца = выражение, …

   [**WHERE** условие_поиска]

Пример. Увеличить на 20% размер стипендии студентов, которые ее получают.

UPDATE Students **SET** stStipend = 1.2 * stStipend

WHERE stStipend IS NOT NULL;

Примечание. Отсутствие предложения WHERE приводит к обновлению ВСЕХ строк из указанной таблицы.

Операции определения данных (DDL)

Команды DDL:

- CREATE – создает объект БД;
- ALTER – изменяет определение существующего объекта;
- DROP – удаляет ранее созданный объект.

Определение таблиц

1. Создание таблиц с помощью языка SQL

Для создания таблицы в языке SQL используется оператор CREATE TABLE:

CREATE TABLE <имя_таблицы> (

<имя_колонки> <тип_данных>[,

<имя_колонки> <тип_данных>]…)

[[**CONSTRAINT** <имя_ограничения>] <ограничение уровня колонки>]…

[[**CONSTRAINT** <имя_ограничения>] <ограничение уровня таблицы>]

Определение таблиц

Ограничения:

PRIMARY KEY – определение первичного ключа таблицы;

UNIQUE – обеспечение уникальности значений в колонке;

NULL / NOT NULL – разрешение или запрещение неопределенных значений в колонке;

CHECK <условие> – задание условия на значение данных в колонке;

[FOREIGN KEY <имя_колонки>] REFERENCES <имя_таблицы> <имя_колонки> – определение внешнего ключа для таблицы.

Определение таблиц

Пример:

```
CREATE TABLE student (
numZach integer CONSTRAINT pkSt PRIMARY KEY,
fio char(30),
stipend integer CHECK (stipend BETWEEN 500 AND 800),
pol char(1) CHECK (pol='м' OR pol='ж'),
grNum integer REFERENCES groups (grNum) ON DELETE
    CASCADE
);
```

Определение таблиц

2. Изменение таблиц. Оператор ALTER TABLE

ALTER TABLE имя_таблицы

ADD определение_столбца

ALTER имя_столбца

SET DEFAULT значение | **DROP DEFAULT**

DROP имя_столбца **CASCADE** | **RESTRICT**

ADD определение_первичного_ключа

ADD определение_внешнего_ключа

ADD условие_уникальности_данных

ADD условие_проверки

DROP CONSTRAINT имя_ограничения

CASCADE | **RESTRICT**

Определение таблиц

Пример. Добавить первичный ключ в таблицу student

ALTER TABLE student **ADD CONSTRAINT** “pk”
PRIMARY KEY (numZach);

Определение таблиц

3. Удаление таблицы

Для удаления таблицы из БД в языке SQL используется оператор DROP TABLE:

DROP TABLE <имя_таблицы> **CASCADE** | **RESTRICT**

Представления

Представление – это виртуальная таблица, которая актуализируется в результате выполнения указанного запроса на выборку данных.

Создание представлений

CREATE VIEW имя_представления (имя_столбца,...) **AS** запрос

Представления

Пример. Создать представление, содержащее список студентов группы КИ-121.

```
CREATE VIEW GroupKI121 (Name) AS
SELECT stName
FROM students, groups
WHERE students.grNum = groups.grNum AND
    groups.grName = 'КИ-121'
```

Обращение к представлению:

```
SELECT * FROM GroupKI121;
```

Представления

Удаление представлений

DROP VIEW имя_представления

Пример. **DROP VIEW** GroupKI121

Примечание. Удаление представления не означает удаления исходных базовых таблиц, на которых оно было определено.

Представления

Модификация данных через представления

Представление модифицируемое, если относительно него можно использовать все три команды – INSERT, UPDATE, DELETE, и оно создано на основе одной таблицы БД.

Условия модифицируемости представлений:

- представление должно формироваться из одной базовой таблицы;
- оно не должно содержать ключевого слова DISTINCT в списке фразы SELECT;

Представления

- список фразы SELECT должен содержать только имена столбцов;
- представление не должно содержать предложений GROUP BY или HAVING, и подзапросов в предложениях FROM и WHERE;
- для операторов INSERT и UPDATE представление должно содержать все столбцы базовой таблицы, которые имеют ограничения NOT NULL.

Представления

Модификация представлений в PostgreSQL. Оператор CREATE RULE

Синтаксис:

CREATE [**OR REPLACE**] **RULE** имя_правила **AS ON**
{ SELECT | INSERT | UPDATE | DELETE}

TO таблица | представление [**WHERE** условие]

DO [[**ALSO**] | **INSTEAD**] { **NOTHING** | команда | (команда [; ...]) }

Представления

Пример. Создать представление, содержащие данные о студентах, получающих стипендию. Создать правило, изменяющее данное представление.

```
CREATE OR REPLACE VIEW stStip AS
    SELECT students.stNum, students.stName,
    students.stStipend, groups.grName
    FROM students, groups
    WHERE students.stStipend > 0 AND
    students.grNum = groups.grNum;
```

Представления

```
CREATE OR REPLACE RULE upd AS ON UPDATE TO
    stStip
DO INSTEAD
    (UPDATE students  SET stNum = new.stNum,
    stName = new.stName, stStipend=new.stStipend
    WHERE stNum = old.stNum;
    UPDATE groups SET grName = new.grName
    WHERE grName = old.grName);
```

Пример оператора UPDATE для представления:

```
UPDATE stStip SET stStipend = 1.2 * stStipend;
```

Хранимые процедуры

Хранимая процедура (Stored Procedure, SP) – набор заранее скомпилированных операторов SQL и операторов управления программой, который хранится как объект БД.

1. **Преимущества**

SP расширяют стандартные возможности SQL, позволяя использовать входные и выходные параметры, операторы принятия решения и объявления переменных. Обеспечивают значительный выигрыш в быстродействии, поскольку операторы SQL, содержащиеся в SP, заранее скомпилированы.

Хранимые процедуры

2. Использование SP

- SP используются во всех случаях, когда необходимо получить максимальное быстродействие и свести код SQL в единую программу. Чаще всего используются SP, выполняющие вставку, удаление и обновление данных, а также формирующие данные для отчетов.

Хранимые процедуры

- SP используются в качестве **механизма защиты**: если нельзя предоставлять прямой доступ пользователям к таблицам и представлениям БД, тогда им предоставляется доступ к таблицам только по чтению, а для выполнения таких операций, как UPDATE и DELETE, создаются соответствующие SP, на выполнение которых пользователи и получают права, т.е. пользователи получают доступ к таблицам БД только путем выполнения SP.

Хранимые процедуры

3. **Средства реализации хранимых процедур в промышленных СУБД**

СУБД	Язык хранимых процедур
MS SQL Server	Transact-SQL
Oracle	PL/SQL
Informix	SPL
PostgreSQL	PL/pgSQL, C, PL/Perl, PL/Java
MySQL	подобен PL/SQL Oracle
InterBase	PSQL
Cache’	Cache’ Object Script

Хранимые процедуры

4. Хранимые процедуры в PostgreSQL

Хранимая процедура в PostgreSQL называется **функцией**.

Для создания функций используется оператор Create Function языка PL/pgSQL:

```
CREATE [OR REPLACE] FUNCTION <имя_функции>
        ([<параметры> <тип данных> , ...])
RETURNS <тип_данных_результата> AS $$
[DECLARE объявление; ...]
BEGIN
    <блок_операторов>; [...]
    RETURN <имя_переменной>;
END;
$$
LANGUAGE 'plpgsql';
```

Хранимые процедуры

Вызов функции:

SELECT * **FROM** имя_функции([параметр, ...]);

Хранимые процедуры

Пример 1. Добавить сведения о новом студенте.

Эта функция с несколькими входными параметрами, не возвращающая значений.

```
CREATE OR REPLACE FUNCTION "setStudent"( st_no integer,
    st_name character varying, ...)
--Определение типа результата функции
RETURNS void AS $$      -- Указывается тип возвращаемого
                        -- результата void
--Основной блок оператора Create Function
BEGIN
    INSERT INTO Students (StNum, StName, ...)
      VALUES (st_no, st_name, ...)
    RETURN;
END; $$
LANGUAGE 'plpgsql';
```

Хранимые процедуры

Вызов функции:

SELECT * **FROM** "setStudent"(958977, 'Иванов', ...);

Хранимые процедуры

Пример 2. Подсчитать сумму затрат на стипендию всех студентов. Функция без входных параметров. Возвращает одно значение.

```
CREATE OR REPLACE FUNCTION "getSumStipAll"()
    --Определение типа результата функции
RETURNS real AS $$   -- Указывается тип возвращаемого
                     -- результата real
    --Блок определения переменных
DECLARE summa real;
    --Основной блок оператора Create Function
BEGIN
    summa = (SELECT SUM(StStipend) FROM Students);
    RETURN summa;
END; $$
LANGUAGE 'plpgsql';
```

Триггеры

Триггер (trigger) базы данных – это хранимая процедура особого типа, которая вызывается автоматически при наступлении определенных событий (INSERT, UPDATE, DELETE).

Использование

- Триггер используется для внесения изменений в таблицы или для выполнения сложных ограничений, которые нельзя реализовать обычным способом.
- Триггер не должен обращаться ни к каким объектам БД кроме таблицы, с которой он ассоциирован.
- В триггере не допускается выполнение операторов DDL (CREATE, ALTER, DROP).
- В теле триггера можно вызывать любые хранимые процедуры, в том числе и системные.

Описание и программирование триггера в PostgreSQL

Синтаксис определения триггера в PostgreSQL:

CREATE TRIGGER имя_триггера
время_инициирования_триггера событие_триггера
ON имя_таблицы
[уровень_триггера]
[**WHEN** (условие_триггера)]
EXECUTE ROCEDURE заголовок_триггерной_функции;

время_инициирования_триггера:

- до выполнения события – **BEFORE**
- после выполнения события – **AFTER**

Описание и программирование триггера в PostgreSQL

событие_триггера:: INSERT | DELETE | UPDATE [OF список_столбцов]

уровень_триггера указывает, сколько раз триггер будет выполнятся по отношению к инициирующему его событию. Возможны два варианта:

- FOR EACH ROW – по одному разу относительно каждой строки, над которой произойдет событие_триггера;
- FOR EACH STATEMENT – один раз относительно события_триггера, независимо от того, сколько строк оно обрабатывает.

Описание и программирование триггера в PostgreSQL

условие_триггера – логическое выражение, истинность которого разрешает выполнять триггерную функцию (только в триггерах над строками)

заголовок_триггерной_функции – указывается триггерная функция, которая будет выполняться при инициализации триггера.

Примечание. В PostgreSQL **триггерная функция** отличается от хранимой функции тем, что тип ее результата – **trigger**.

Описание и программирование триггера в PostgreSQL

Во время работы триггера доступны специальные переменные:

OLD – запись перед обновлением или перед удалением;

NEW – запись, которая будет вставлена или обновлена.

Событие	**Переменная**
UPDATE	OLD, NEW
INSERT	NEW
DELETE	OLD

Описание и программирование триггера в PostgreSQL

Удаление триггера

DROP TRIGGER имя_триггера

Описание и программирование триггера в PostgreSQL

Пример. При добавлении записи в таблицу Marks БД «Деканат ВУЗа» необходимо автоматически пересчитывать средний балл студента.

```
CREATE OR REPLACE FUNCTION update_SrBall() RETURNS
    trigger
$BODY$
begin
    update Students set srBall = (select AVG(mark) from
                    Marks where stNum =
    new.stNum)
    where stNum = new.stNum;
    return new;
end
$BODY$
```

Описание и программирование триггера в PostgreSQL

Триггер:

```
CREATE TRIGGER change_StSrBall
    AFTER INSERT ON Marks
    FOR EACH ROW
    EXECUTE ROCEDURE update_SrBall
```

Привилегии. Директивы GRANT и REVOKE

GRANT (допуск)

REVOKE (отмена)

Привилегии для таблиц и представлений (SQL1):

- SELECT – позволяет считывать данные
- INSERT – позволяет вставлять новые записи
- UPDATE – позволяет модифицировать записи
- DELETE – позволяет удалять записи

Привилегии. Директивы GRANT и REVOKE

Привилегии для таблиц и представлений (SQL2):

- привилегии стандарта SQL1
- INSERT – для отдельных столбцов подобно привилегии UPDATE
- REFERENCES – для поддержки внешнего ключа
- USAGE – для других объектов БД

Дополнительны привилегии в СУБД

- ALTER – позволяет модифицировать структуру таблиц (DB2, Oracle)
- EXECUTE – позволяет выполнять хранимые процедуры

Привилегии. Директивы GRANT и REVOKE

Синтаксис GRANT:

GRANT {**SELECT**|**INSERT**|**DELETE**|(**UPDATE** столбец, …)},…

ON таблица **TO** {пользователь | **PUBLIC**} [**WITH GRANT OPTION**]

Пример: предоставить пользователю Ivanov полномочия для осуществления выборки и модификации фамилий в таблице Students с правом предоставления полномочий.

GRANT SELECT, UPDATE StName

ON Students **TO** Ivanov **WITH GRANT OPTION**

Привилегии. Директивы GRANT и REVOKE

Синтаксис REVOKE:

REVOKE {{**SELECT** | **INSERT** | **DELETE** | **UPDATE**},…|**ALL PRIVILEGES**}

ON таблица,… **FROM** {пользователь | **PUBLIC**},… {**CASCADE** | **RESTRICT**}

Пример: снять с пользователя Ivanov полномочия для осуществления модификации фамилий в таблице Students. Также снять эту привилегию со всех пользователей, которым она была предоставлена Ивановым.

REVOKE UPDATE
ON Students **FROM** Ivanov **CASCADE**