



Операции модификации данных в таблицах



INSERT

```
INSERT INTO target
{DEFAULT VALUES | [(<column_list>)] <value_source>}
[RETURNING <returning_list> [INTO <variables>]]

<column_list> ::= colname [, colname ...]

<value_source> ::= VALUES (<value_list>) | <select_stmt>

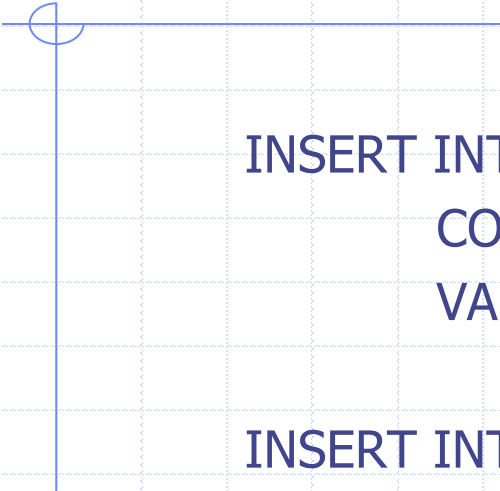
<value_list> ::= value [, value ...]

<returning_list> ::=
    ret_value [[AS] ret_alias] [, ret_value [[AS] ret_alias] ...]

<variables> ::= [:]varname [, [:]varname ...]
```

```
INSERT INTO <таблица> [(<список столбцов>)]
VALUES (<список значений>)
```

```
INSERT INTO <таблица> [(<список столбцов>)]
SELECT ...
```



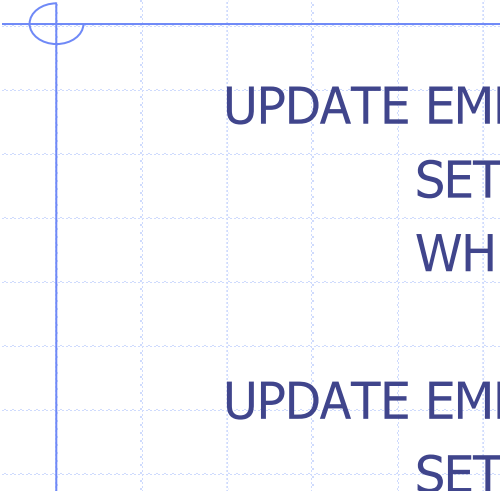
```
INSERT INTO
  COUNTRY (COUNTRY, CURRENCY)
VALUES ('Georgia' , 'Lari');
```

```
INSERT INTO SALES_ARH
  SELECT *
    FROM SALES
    WHERE SHIP_DATE IS NOT NULL
        AND PAID= 'y';
```

UPDATE

```
UPDATE target [[AS] alias]  
SET col = newval [, col = newval ...]  
[WHERE {search-conditions | CURRENT OF cursorname}]  
[PLAN plan_items]  
[ORDER BY sort_items]  
[ROWS m [TO n]]  
[RETURNING <returning_list> [INTO <variables>]]  
  
<returning_list> ::=  
  ret_value [[AS] ret_alias] [, ret_value [[AS] ret_alias] ...]  
  
<variables> ::= [:]varname [, [:]varname ...]
```

```
UPDATE <таблица>  
  SET <столбец> = <значение>  
    [, <столбец> = <значение> .. ]  
[WHERE <θ-условие> ]
```



```
UPDATE EMPLOYEE  
  SET SALARY = 40000  
  WHERE EMP_NO = 65;
```

```
UPDATE EMPLOYEE  
  SET SALARY =SALARY*1,5;
```

DELETE

```
DELETE
FROM target [[AS] alias]
[WHERE {search-conditions | CURRENT OF cursorname}]
[PLAN plan_items]
[ORDER BY sort_items]
[ROWS m [TO n]]
[RETURNING <returning_list> [INTO <variables>]]

<returning_list> ::= ret_value [[AS] alias_val] [, ret]
<variables> ::= [:]varname [, [:]varname ...]
```

```
DELETE FROM <таблица>
[WHERE < $\theta$ -условие>]
```



```
DELETE
    FROM SALARY_HISTORY
    WHERE EMP_NO = 65;
```

```
DELETE
    FROM SALES
    WHERE SHIP_DATE IS NOT NULL
        AND PAID= ' y ';
```

```
DELETE
    FROM SALARY_HISTORY; -- очистка всей таблицы
```



ПРЕДСТАВЛЕНИЯ

Представления (view)

- ◆ Вводится в архитектуре ANSI/SPARC
- ◆ Позволяет скрыть структуру таблиц
- ◆ Позволяет реализовать суперпозицию реляционных операций на уровне оператора SELECT
- ◆ Но, в большинстве СУБД не позволяет выполнять операции модификации (INSERT, UPDATE, DELETE) на уровне представлений (одно из 12 правил Кодда)
- ◆ Хранит в откомпилированном виде оператор SELECT как элемент БД
- ◆ В реляционной модели эквивалентно таблице

Создание/изменение представления

```
CREATE [OR ALTER] VIEW <name_view>  
    [ (<nm_col1> [,<nm_col2>, ...])]  
AS <select_operator>
```

Пример

```
CREATE VIEW  
EMP_DEPT (EMP_NAME, DEPT_NAME,  
           SALARY, COUNTRY)  
  
AS  
SELECT  FULL_NAME, DEPARTMENT, SALARY, JOB_COUNTRY  
        FROM EMPLOYEE A JOIN DEPARTMENT D  
        ON  A.DEPT_NO = D.DEPT_NO
```

В операторе SELECT нет различий между таблицами и представлениями

```
SELECT EMP_NAME, DEPT_NAME, SALARY, CN.COUNTRY  
FROM EMP_DEPT JOIN COUNTRY CN USING (COUNTRY)  
WHERE CURRENCY='Euro'
```

Пример и БД «Сотрудники»

```
CREATE OR ALTER VIEW PHONE_LIST(  
    EMP_NO,  
    FIRST_NAME,  
    LAST_NAME,  
    PHONE_EXT,  
    LOCATION,  
    PHONE_NO)  
AS  
SELECT  
    emp_no, first_name, last_name, phone_ext, location, phone_no  
FROM employee, department  
WHERE employee.dept_no = department.dept_no
```



Подзапросы и предикаты



Проблема

- ◆ Требование реляционной модели – суперпозиция реляционных операций
- ◆ SELECT обеспечивает ее лишь частично
- ◆ Использование представлений возможно, но не подходит для решения задачи «на лету», т.к. требует работы со схемой БД

Подзапрос

- ◆ Подзапрос – это оператор `SELECT`, включенный в спецификацию другого оператора `SQL`
- ◆ Если подзапрос ссылается на поля из таблиц в основном запросе, то он называется коррелированным

Использование подзапросов

◆ В операторе INSERT

```
INSERT INTO <таблица> [(<список столбцов>)]  
SELECT ...
```

```
INSERT INTO SALES_ARH  
    SELECT * FROM SALES  
        WHERE SHIP_DATE IS NOT NULL AND PAID='y';
```

```
INSERT INTO PROJECT A  
    (A.PROJ_ID,A.PROJ_NAME, A.PRODUCT,A.TEAM_LEADER)  
VALUES ('DB','DataBase','software',  
(SELECT E.EMP_NO FROM EMPLOYEE E WHERE E.LAST_NAME='Williams'))
```


Использование подзапросов

- ◆ для формирования выходного столбца основного оператора SELECT

```
SELECT e.FULL_NAME, e.SALARY,  
(SELECT MAX (m.SALARY) FROM EMPLOYEE m  
    WHERE m.JOB_COUNTRY='USA') AS MAX_SALARY_USA  
FROM EMPLOYEE e  
WHERE e.JOB_COUNTRY='USA'
```

```
SELECT e.FULL_NAME, e.SALARY,  
(SELECT MAX (m.SALARY) FROM EMPLOYEE m  
    WHERE m.JOB_COUNTRY='USA') - e.SALARY AS DIFF_SALARY_USA  
FROM EMPLOYEE e  
WHERE e.JOB_COUNTRY='USA'
```

Использование подзапросов

```
SELECT FULL_NAME, SALARY,  
(SELECT CURRENCY FROM COUNTRY c  
WHERE  
c.COUNTRY=e.JOB_COUNTRY) AS CURRENCY  
FROM EMPLOYEE e
```

Такой подзапрос называется коррелированным.
Это неявная форма операции соединения

```
SELECT FULL_NAME, SALARY, CURRENCY  
FROM EMPLOYEE e JOIN COUNTRY c  
ON c.COUNTRY=e.JOB_COUNTRY
```

Использование подзапросов

- ◆ в предложении FROM (начиная с версии Firebird 2.0)
- ◆ в этом случае подзапрос выступает в роли «производной таблицы» (*derived table*)
- ◆ Это аналог реляционной операции присваивания, т.к. такой таблице дается имя

(select-запрос)

[[AS] алиас производной таблицы]

[(<псевдонимы полей производной таблицы>)]

Derived table

```
SELECT P.PROJ_NAME, T.DEPT_NAME
FROM
  PROJECT P LEFT JOIN
  (SELECT PD.PROJ_ID, D.DEPARTMENT
   FROM PROJ_DEPT_BUDGET PD
   JOIN DEPARTMENT D
   ON (PD.DEPT_NO=D.DEPT_NO)
   WHERE PD.FISCAL_YEAR=1995)
  AS T ( PROJ_ID, DEPT_NAME)
ON (P.PROJ_ID=T.PROJ_ID)
```

Чаще всего используется именно для открытых соединений

Альтернатива - представление

```
CREATE VIEW DT ( PROJ_ID, DEPT_NAME)
```


```
AS
```

```
SELECT PD.PROJ_ID, D.DEPARTMENT  
FROM PROJ_DEPT_BUDGET PD  
JOIN DEPARTMENT D  
ON (PD.DEPT_NO=D.DEPT_NO)  
WHERE PD.FISCAL_YEAR=1995;
```

```
SELECT P.PROJ_NAME, T.DEPT_NAME  
FROM PROJECT P LEFT JOIN DT ON  
(P.PROJ_ID=DT.PROJ_ID)
```

Использование подзапросов

- ◆ для получения значений или условий, используемых в предикатах поиска предложения WHERE операторов SELECT, UPDATE и DELETE
- ◆ в предложении HAVING для группирующего запроса

- 
- ◆ Если подзапрос возвращает одно значение (скалярный запрос), его можно использовать в θ -условии для сравнения на равенство или неравенство
- Скалярность требуется на уровне каждой строки, т.е. подзапрос может быть коррелированным

Использование подзапросов

◆ В операторе UPDATE

```
UPDATE employee
```

```
    SET dept_no = (SELECT dept_no
```

```
                    FROM department
```

```
                    WHERE department = 'Field Office: France' )
```

```
WHERE full_name='Lee, Terri'
```

1 record(s) was(were) updated in EMPLOYEE

Использование подзапросов

◆ В операторе UPDATE

```
UPDATE employee SET salary=salary*1.05
  WHERE dept_no = (SELECT dept_no
                   FROM department
                   WHERE department = 'Field Office: France' )
```

1 record(s) was(were) updated in EMPLOYEE

1 record(s) was(were) inserted into SALARY_HISTORY

Использование подзапросов

◆ В операторе DELETE

```
DELETE FROM salary_history  
WHERE emp_no=(SELECT emp_no  
FROM employee  
WHERE full_name='Lee, Terri')
```

2 record(s) was(were) deleted from SALARY_HISTORY

```
SELECT FULL_NAME
  FROM EMPLOYEE a
 WHERE
  a.SALARY >
  (SELECT AVG(b.SALARY)
   FROM EMPLOYEE b);
```

```
SELECT FULL_NAME
  FROM EMPLOYEE a
 WHERE
  a.SALARY >
  (SELECT AVG(B.SALARY)
   FROM EMPLOYEE B
  WHERE
   B.JOB_COUNTRY=A.JOB_COUNT
  RY);
```

- ◆ Если подзапрос возвращает таблицу, содержащую один столбец и произвольное количество строк, для использования его в подзапросе применяются предикаты
- ◆ В этом случае возвращаемый столбец рассматривается как множество

Предикат IN (NOT IN)

Принадлежность множеству

```
SELECT FULL_NAME  
FROM EMPLOYEE  
WHERE DEPT_NO  
      IN  
      (SELECT DEPT_NO  
       FROM DEPARTMENT  
       WHERE MNGR_NO IS NULL)
```

Предикаты ALL, SOME / ANY

```
SELECT PROJ_ID, DEPT_NO, FISCAL_YEAR
FROM PROJ_DEPT_BUDGET
WHERE
PROJECTED_BUDGET >=
    ALL (SELECT BUDGET
         FROM DEPARTMENT)
```

```
SELECT e.EMP_NO, e.FULL_NAME, e.HIRE_DATE
FROM EMPLOYEE e
WHERE e.HIRE_DATE+365 >
    SOME (SELECT sh.CHANGE_DATE
          FROM SALARY_HISTORY sh
          WHERE sh.EMP_NO = e.EMP_NO)
```

-- ВОЗМОЖНО ИСПОЛЬЗОВАНИЕ ANY

Предикат SINGULAR (NOT SINGULAR)

◆ проверяет, возвращает ли подзапрос в точности один кортеж

Если возвращается NULL или более одного кортежа, то SINGULAR возвращает ложь

Значение предиката не зависит от количества столбцов в подзапросе и от значений в них

```
SELECT FULL_NAME
      FROM EMPLOYEE e
      WHERE
        SINGULAR (SELECT *
                  FROM SALES s
                  WHERE s.SALES_REP=e.EMP_NO);
```


Предикат EXISTS (NOT EXISTS)

- ◆ определяет, существует ли (или нет), по крайней мере, один кортеж в выходном результате подзапроса
- ◆ Значение предиката не зависит от количества столбцов в подзапросе

```
SELECT a.FULL_NAME  
FROM EMPLOYEE a  
WHERE  
    EXISTS (SELECT 1  
           FROM PROJECT p  
           WHERE p.TEAM_LEADER = a.EMP_NO)
```

Подзапрос в условии на группу

```
SELECT D.LOCATION
FROM DEPARTMENT D
GROUP BY LOCATION
HAVING COUNT(*) >=
    ALL (SELECT COUNT(*)
        FROM DEPARTMENT
        GROUP BY LOCATION )
```


Общие табличные выражения (CTE)

- ◆ Более мощный вариант производных таблиц. Чаще используются в процедурных расширениях SQL

```
WITH EMP_DEPT (EMP_NAME, DEPT_NAME, SALARY, COUNTRY)
AS
(SELECT FULL_NAME, DEPARTMENT, SALARY, JOB_COUNTRY
FROM EMPLOYEE A JOIN DEPARTMENT D
    ON  A.DEPT_NO = D.DEPT_NO)
SELECT EMP_NAME, DEPT_NAME, SALARY
FROM EMP_DEPT
WHERE COUNTRY ='USA'
```

Рекурсивные STE

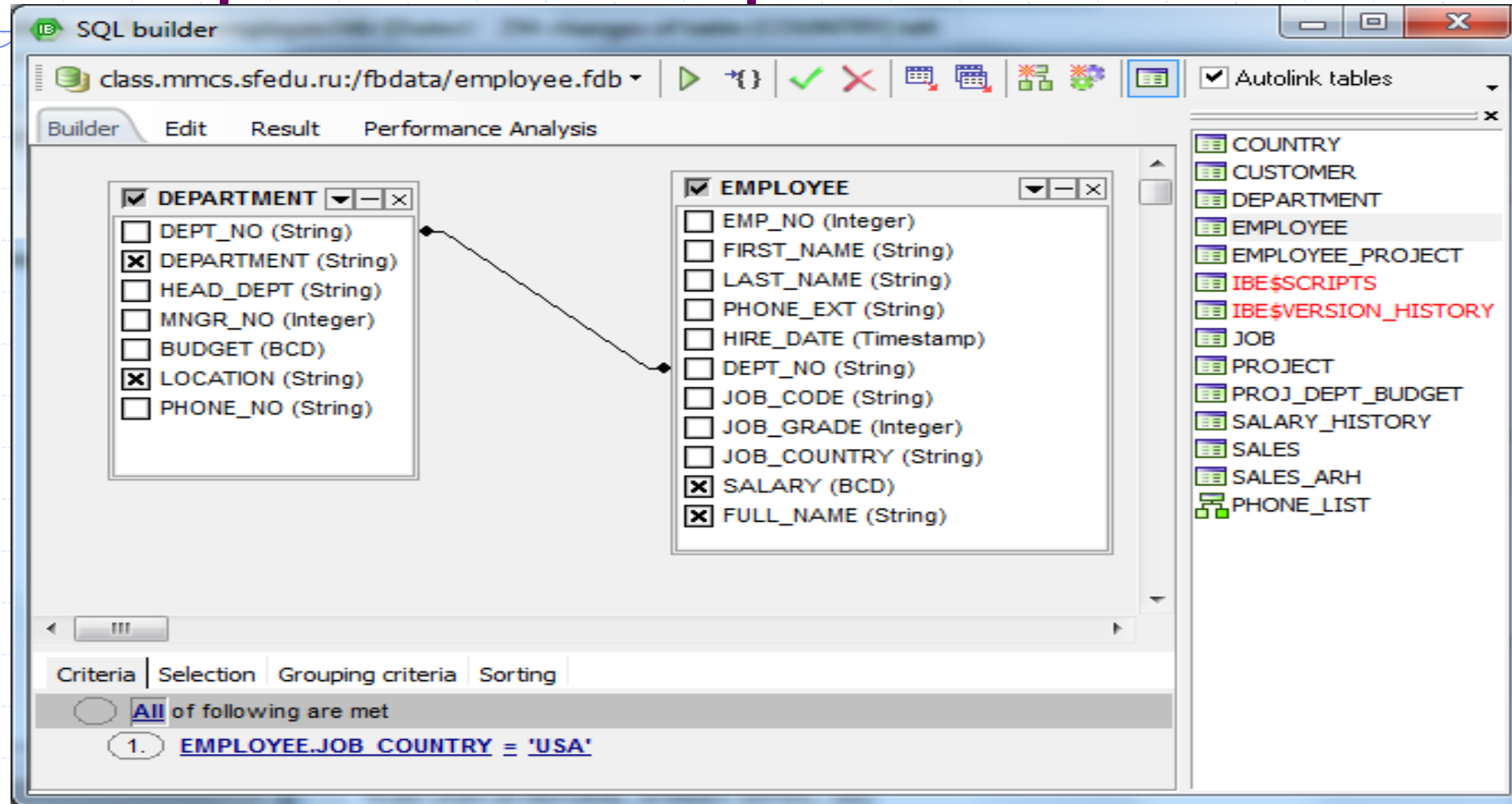
```
WITH RECURSIVE r as (  
    -- стартовая часть рекурсии (т.н. "anchor")  
    SELECT      1 as i,      1 as factorial  
              FROM RDB$DATABASE  
    UNION ALL  -- рекурсивная часть  
    SELECT      i+1 as i,  
              factorial * (i+1) as factorial  
    FROM r  
    WHERE i < 10  
)  
SELECT * FROM r;
```



The screenshot shows a database query results window with tabs for Edit, Results, History, and Plan Analyzer. The Results tab is active, displaying a table with two columns: I and FACTORIAL. The table contains 12 rows of data, with the first row highlighted in blue. The window also shows a toolbar with various icons and a Record: 1 indicator.

| I | FACTORIAL |
|----|-------------|
| 1 | 1 |
| 2 | 2 |
| 3 | 6 |
| 4 | 24 |
| 5 | 120 |
| 6 | 720 |
| 7 | 5 040 |
| 8 | 40 320 |
| 9 | 362 880 |
| 10 | 3 628 800 |
| 11 | 39 916 800 |
| 12 | 479 001 600 |

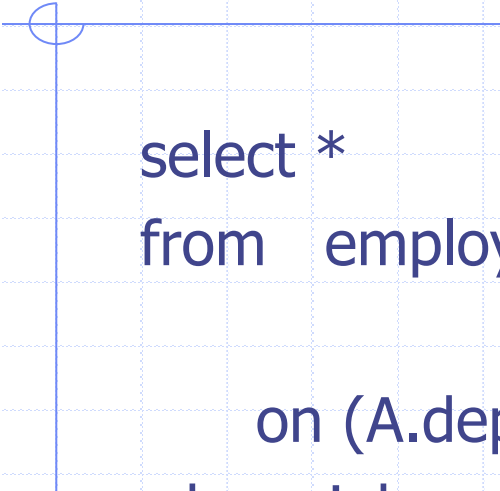
Построитель запросов



```
select
    department.department,
    department.location,
    employee.full_name,
    employee.salary
from employee
    inner join department on (employee.dept_no = department.dept_no)
where
    (employee.job_country = 'USA')
```

План выполнения запроса

- ◆ **План выполнения запроса** — последовательность операций, необходимых для получения результата SQL-запроса в реляционной СУБД
- ◆ План в целом разделяется на две стадии:
 - Выборка результатов
 - Сортировка и группировка, выполнение агрегаций
- ◆ *Сортировка и группировка* — это опциональная стадия, которая выполняется, если не найдено путей доступа для получения результата в запрошенном порядке.



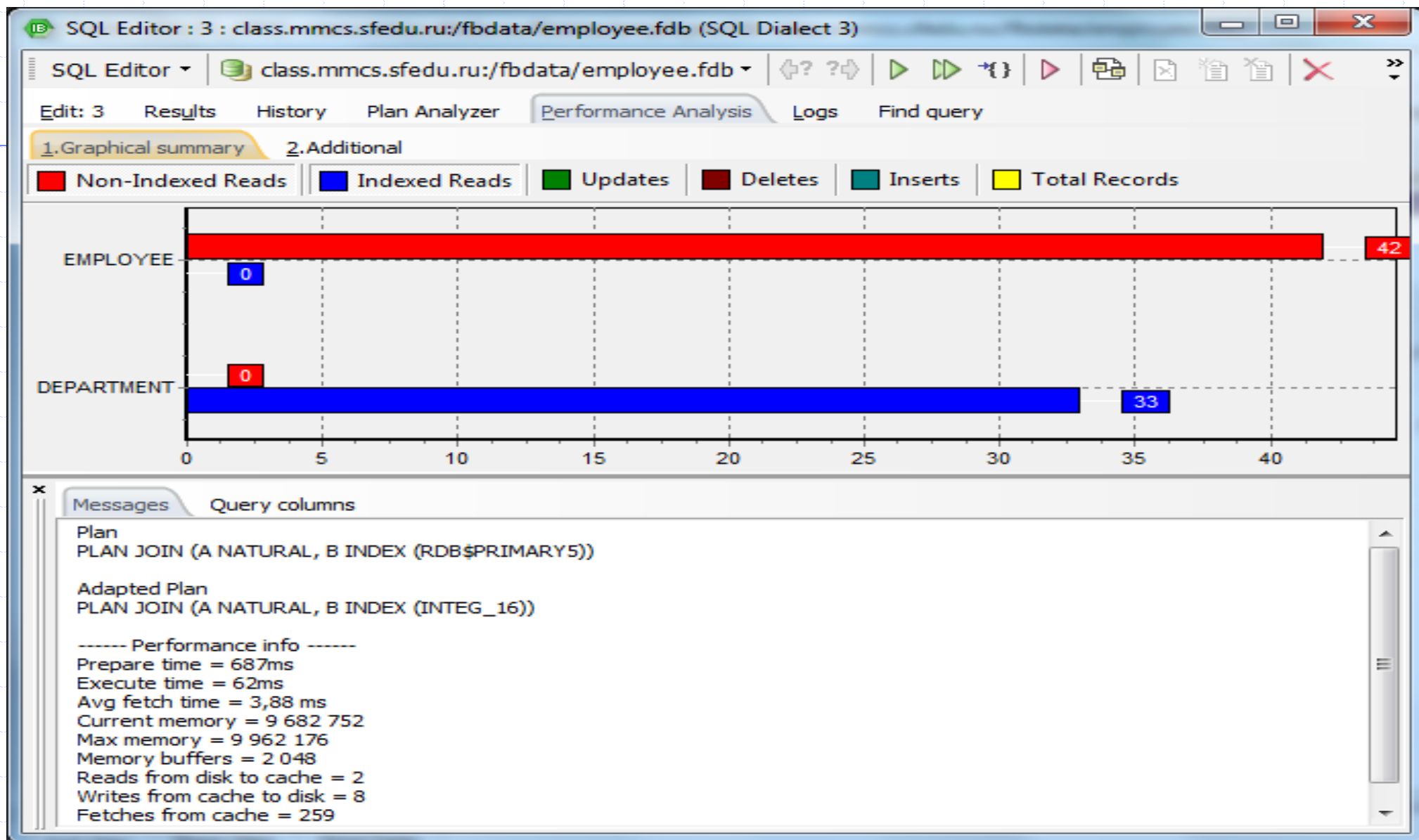
```
select *  
from employee A  
        join department B  
        on (A.dept_no = B.dept_no)  
where job_country = 'USA'
```


PLAN JOIN (A NATURAL, B INDEX (RDB\$PRIMARY5))

Explain Plan

Select Expression

- > Nested Loop Join (inner)
 - > Filter
 - > Table "EMPLOYEE" as "A" Full Scan
 - > Filter
 - > Table "DEPARTMENT" as "B" Access By ID
 - > Bitmap
 - > Index "RDB\$PRIMARY5" Unique Scan



Включение плана в запрос

```
SELECT [DISTINCT | ALL]  
  <select_list>  
FROM <reference_expression_list>  
[ WHERE <search condition> ]  
[ GROUP BY <group_value_list>  
  [ HAVING <group_condition> ] ]  
[ UNION <select operator> ]  
[ PLAN <plan_item_list> ]
```

PLAN <выражение>

<выражение> ::= [JOIN | [SORT] [MERGE]] (<элемент> | <выражение>
[, <элемент> | <выражение> ...])

<элемент> ::= {таблица | псевдоним}
{NATURAL
| INDEX (индекс [, индекс ...])
| ORDER индекс [INDEX (индекс [, индекс...])]}