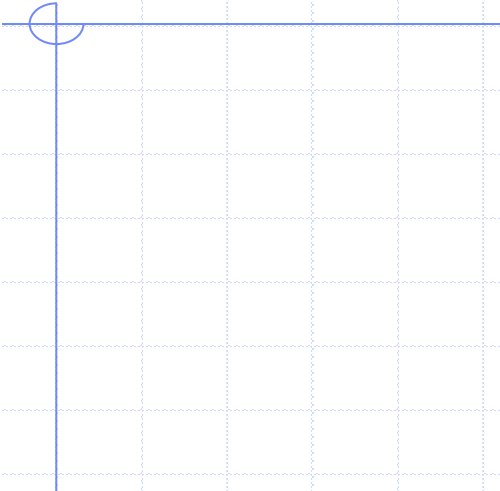


Суперпозиция реляционных операций

Проблема

- ◆ Требование реляционной модели – суперпозиция реляционных операций
- ◆ SELECT обеспечивает ее лишь частично
 - Проекция
 - Соединение
 - Селекция
 - Группировка



ПРЕДСТАВЛЕНИЯ

Представления (view)

- ◆ Вводится в архитектуре ANSI/SPARC
- ◆ В реляционной модели эквивалентно таблице
- ◆ Хранит в откомпилированном виде оператор SELECT как элемент БД
- ◆ 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)

Создание/изменение представления

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

```
ALTER VIEW [ IF EXISTS] <name_view> <action>;
```

```
DROP VIEW [ IF EXISTS ] <name_view> ;
```

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

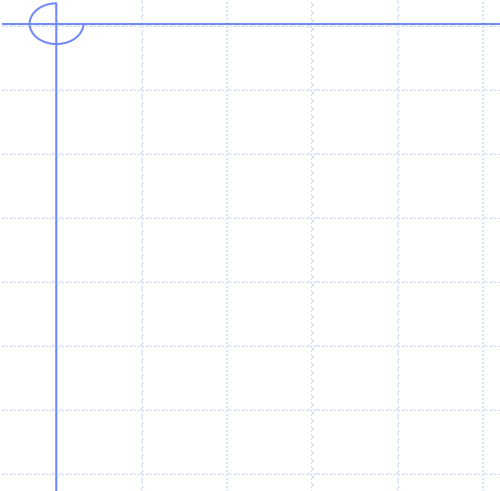
```
SELECT EMP_NAME, DEPT_NAME, SALARY, CN.COUNTRY  
FROM EMP_DEPT JOIN COUNTRY CN USING (COUNTRY)  
WHERE CURRENCY='Euro'
```

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
```

Проблема

- ◆ Использование представлений удобно и обеспечивает возможность суперпозиции реляционных операций, но не подходит для решения задачи «на лету», т.к. требует работы со схемой БД



ПОДЗАПРОСЫ

Подзапрос

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

Использование подзапросов

- ◆ для формирования выходного столбца основного оператора SELECT

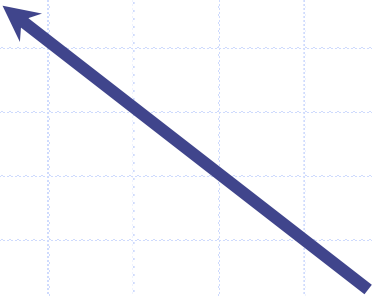
```
SELECT e.FULL_NAME, e.SALARY,  
(SELECT MAX (m.SALARY) FROM EMPLOYEE m  
    WHERE m.JOB_COUNTRY='USA') AS MAX_SALARY_USA  
FROM EMPLOYEE e  
WHERE e.JOB_COUNTRY='USA'
```

```
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

- ◆ Если нас будут интересовать все сотрудники, то max нужно определять для каждой страны

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



коррелированный подзапрос

Использование подзапросов

Коррелированный подзапрос можно использовать вместо операции соединения

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

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

Использование подзапросов

- ◆ в предложении FROM

в этом случае подзапрос выступает в роли «производной таблицы» (*derived table*)

- ◆ Это аналог реляционной операции присваивания, т.к. такой таблице можно дать 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.D EPT_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
```

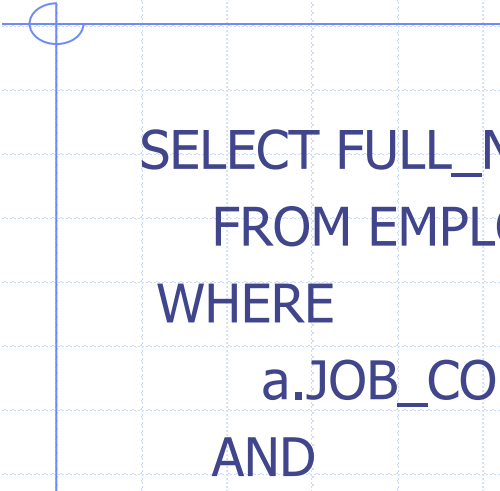
```
SELECT PD.PROJ_ID, D.DEPARTMENT  
FROM PROJ_DEPT_BUDGET PD  
JOIN DEPARTMENT D  
ON (PD.DEPT_NO=D.DEPT_NO)  
WHERE PD.FISCAL_YEAR=1995;
```

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

Использование подзапросов

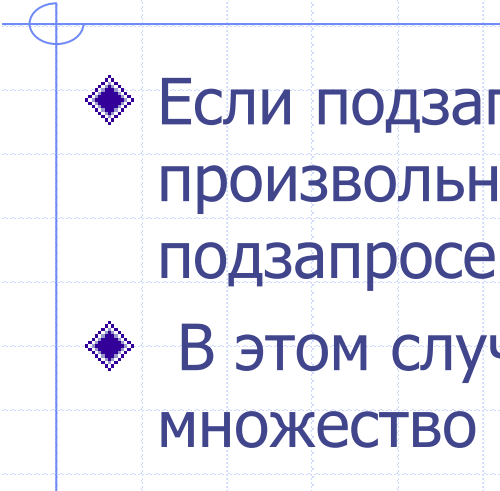
- ◆ для получения значений или условий, используемых в предикатах поиска предложения WHERE и в предложении HAVING для группирующего запроса

- ◆ 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



```
SELECT FULL_NAME
  FROM EMPLOYEE a
 WHERE
    a.JOB_COUNTRY='USA'
 AND
    a.SALARY >
    (SELECT AVG(b.SALARY)
     FROM EMPLOYEE b
     WHERE
        b.JOB_COUNTRY='USA');
```

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

- 
- ◆ 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)
```

Предикаты ALL, SOME / ANY

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

Предикаты ALL, SOME / ANY

```
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 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 -- можно *  
            FROM PROJECT p  
            WHERE p.TEAM_LEADER = a.EMP_NO)
```

Операторы DML

Операции модификации данных в таблицах

INSERT

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

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

Использование подзапросов

◆ В операторе INSERT

```
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'))
```

UPDATE

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;

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'
```

DELETE

```
DELETE FROM <таблица>  
[WHERE <θ-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
```

Использование подзапросов

◆ В операторе DELETE

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

Общие табличные выражения (СТЕ)

- ◆ Более мощный вариант производных таблиц.

```
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'
```

Общие табличные выражения (CTE)

```
WITH EMP_DEPT_USA (NAME, DEPT_NAME, TOWN, SALARY)
  AS
  (SELECT FULL_NAME, DEPARTMENT, LOCATION, SALARY
    FROM EMPLOYEE A JOIN DEPARTMENT D ON A.DEPT_NO=D.DEPT_NO
   WHERE JOB_COUNTRY ='USA')
SELECT NAME, DEPT_NAME, SALARY, TOWN
  FROM EMP_DEPT_USA
 WHERE EMP_DEPT_USA.SALARY >
        (SELECT AVG(SALARY) FROM EMP_DEPT_USA );
```

Рекурсивные STE

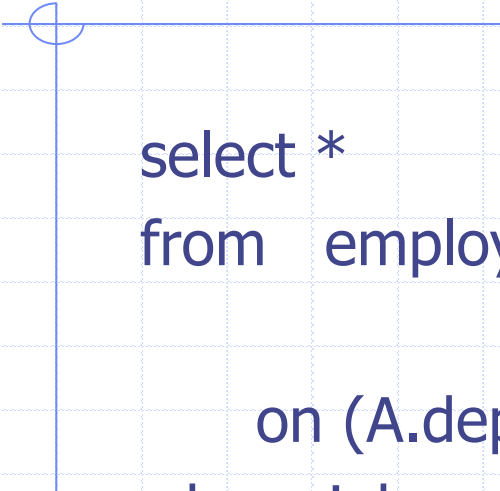
```
WITH RECURSIVE RT as (  
    -- стартовая часть рекурсии (т.н. "anchor")  
    SELECT      1 as i,      1 as factorial  
    -- запрос без from возвращает 1 строку  
    UNION ALL   -- рекурсивная часть  
    SELECT      i+1 as i,  
                factorial * (i+1) as factorial  
    FROM RT  
    WHERE i < 10  
)  
SELECT * FROM RT;
```

WITH RECURSIVE r as (SELECT

Таблица	123 i	123 factorial
1	1	1
2	2	2
3	3	6
4	4	24
5	5	120
6	6	720
7	7	5 040
8	8	40 320
9	9	362 880
10	10	3 628 800

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 опциональные стадии, которые выполняются, 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'
```

ABC QUERY PLAN



Hash Join (cost=12.76..14.06 rows=1 width=554)

Hash Cond: (b.dept_no = a.dept_no)

-> Seq Scan on department b (cost=0.00..1.21 rows=21 width=224)

-> Hash (cost=12.75..12.75 rows=1 width=330)

-> Seq Scan on employee a (cost=0.00..12.75 rows=1 width=330)

Filter: ((job_country)::text = 'USA'::text)